Future Innovators Workshop Handbook

Prepared by the IEEE UNT Student Branch

For comments, questions, or suggestions, contact
Nicholas Chiapputo at nicholaschiapputo@my.unt.edu

Revision Date: February 19, 2021

Contents

1 Introduction

1.1 Welcome to the Future Innovators Workshop

1.2 Arduino

Arduino is an open-source electronics hardware and software company that designs single board microcontrollers. The company produces many different microcontrollers. One of the most popular is the Arduino Uno. This is the one included in your kit.

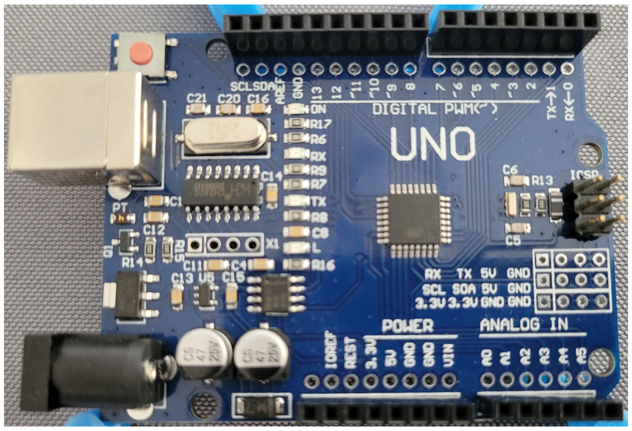


Figure 1.1: Arduino Uno

Arduino's goal is to make easy-to-use tools for learning and teaching circuits and programming with a low-cost system. They are one of the leaders in Do-It-Yourself (DIY) electronics. Using their microcontrollers is one of the best starting points for getting into the world of electronics.

To interface with circuits, the Arduino Uno has multiple *ports* on the device itself. These ports are little holes along the edge of the board. By connecting a wire from one of these holes to a circuit, the Arduino can send and receive signals to perform any number of operations.

The Arduino Uno has 13 digital ports, labeled "1" through "'13" on one side of the board. On the other side, there are 5 analog ports labeled "A0" through "A5".

Additional Resources:
Arduino's website: https://www.arduino.cc/
Arduino tutorials and examples: https://www.arduino.cc/en/Tutorial/HomePage/
https://www.arduino.cc/en/Guide/ArduinoUno/

The Arduino Uno Microcontroller is the key component of the Future Innovators Workshop kit. A Microcontroller is a small computer on an integrated circuit (IC). The microcontroller contains one or more CPUs, memory and programmable input/output peripherals. Microcontrollers are used in many electronic and automated products, such as, implantable medical devices, remote controls, office machines, appliances, power tools and toys.

The microcontroller has ports or registers that allow for a program to control (Outputs) or read (Inputs) the state of the pins. Ports are made up of pins, which are the individual input or output legs of the IC.

For the workshop, we will be using the Arduino Uno microcontroller to control light emitting diodes (LED), operate buzzers, count in binary, make an Infrared remote and more. Pairing the Arduino Uno, various electronic components and imagination virtually anything can be created.

1.3 Arduino IDE

Arduino Integrated Development Environment (IDE) is an application written in C and C++ that is used to write and upload programs to Arduino compatible boards.

Arduino IDE can be downloaded by visiting the Arduino's website
(https://www.arduino.cc/en/software/) and clicking on the download option that matches your computer operating system.

Arduino IDE contains a text editor for writing code, a message area, a text console, a toolbar with buttons for common functions and a series of menus. In the white text editor space, insert the desired code and hit the check mark button to verify the code. Don't forget to save the code often. Each code is called a sketch in Arudino IDE. After the code is verified, connect the Arduino board through the provided usb cord to the computer and click the right arrow button in the Arduino IDE to upload the program to the Arduino Uno microcontroller. Once the program has been uploaded, disconnect the Arduino Uno from the usb cord. The process listed above can be repeated as many times as needed to achieve the desired result.

1.4 Breadboard

A Breadboard is a solderless device with a plastic exterior, numerous openings and internal metal strips that accept many different electrical components. It is used for prototyping or creating early samples/products/models with the key benefit that it is reusable and allows for easy modification of circuits by plugging and unplugging components.

Internally, the outer power rail columns are wired together and the inner rows are wired together (see Figure below for internal connections). Components that are connected in the circuit diagram are inserted into the same row of the breadboard. The large space in the center is built for integrated circuit (IC) chips to straddle so each pin is connected to a different row.

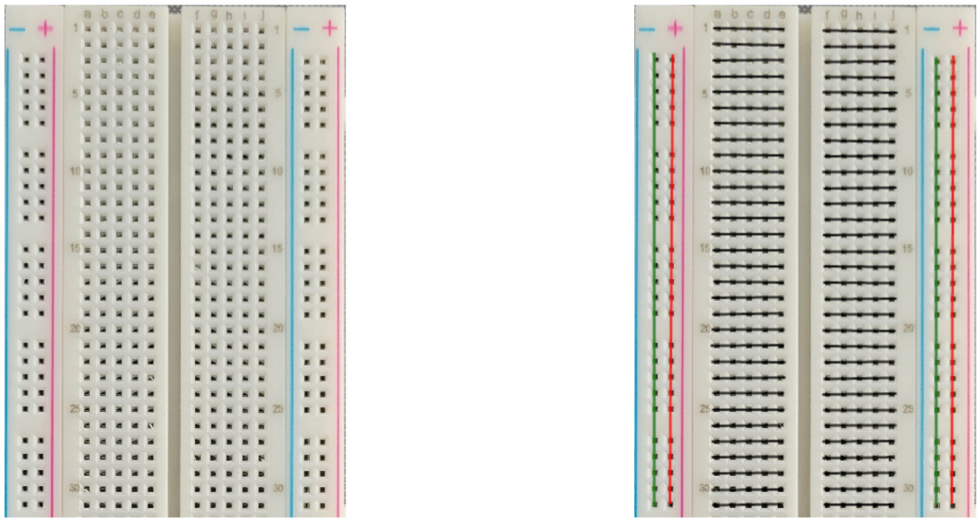


Figure 1.2: Solderless breadboard and internal connections.

It is important to remember how the breadboard rows and columns are connected as it will impact our circuits moving forward. To utilize the breadboard, push the metal legs of the components firmly into the opening. You can test if the component is secured by very lightly pulling up on the component. Another important thing to consider when inserting components into the breadboard is the orientation of the component.

IMPORTANT: For safety, disconnect or turn off the power supply prior to adding or removing any components from the breadboard.

1.5 Voltage

Voltage represents the potential electrical difference between two points. Voltage is the pressure that causes the flow of electric current in a circuit. The mechanical equivalent for voltage would be pressure that pushes water through a pipe.

Voltage is measured in the units of volts, named after the Italian physicist Alessandro Volta. Volts are identified by the symbol V.

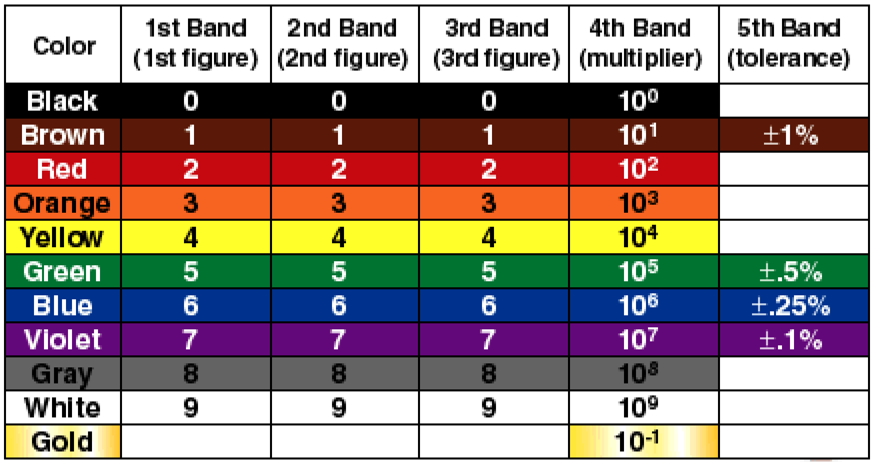
1.6 Current

Current is the rate of flow of electrons in a circuit. The mechanical equivalent for current would be the amount of water passing through a pipe. It is important to know that current flows from a point of higher potential energy to a point of lower potential energy or current follows the path of least resistance in a circuit. Current can be thought of as water flowing from a higher elevation to a lower elevation.

The units of current are Amperes, named after French mathematician and physicist Andre-Marie Ampere. Amperes is often abbreviated to Amps or A.

1.7 Resistance

Resistors are an electronic component that limits or resists the flow of electrons through a circuit. Resistors have a fixed electrical resistance value called ohms, with the symbol Ω. In our mechanical analogy, resistors represent the size of the pipe which limits the amount of water that can pass through it.



Color	1st Band (1st figure)	2nd Band (2nd figure)	3rd Band (3rd figure)	4th Band (multiplier)	5th Band (tolerance)
Black	0	0	0	10^0	
Brown	1	1	1	10^1	±1%
Red	2	2	2	10^2	
Orange	3	3	3	10^3	
Yellow	4	4	4	10^4	
Green	5	5	5	10^5	±.5%
Blue	6	6	6	10^6	±.25%
Violet	7	7	7	10^7	±.1%
Gray	8	8	8	10^8	
White	9	9	9	10^9	
Gold				10^{-1}	

Figure 1.3: Resistor Color Code Chart

The resistance value of the resistors can be identified by the colored bands on the body of the resistor.

2 Level 1 Projects

2.1 Pushbutton LED

Parts Required

- 1x Arduino + USB Cable
- 2x Wires
- 1x Breadboard
- 1x LED
- 1x 330 Ω Resistor
- 1x Pushbutton

Steps

1. Place the pushbutton so that it straddles the river on the breadboard. The legs of the pushbutton should be bending so that they are pointing perpendicular (away from) the river.

2. Place the LED so that the anode (long leg) is in the positive power rail of the breadboard. Place the cathode (short leg) in the same row as one of the legs of the pushbutton.

3. Place the resistor so that one leg is in the negative power rail of the breadboard. Place the other leg in the same row as the other leg of the pushbutton.

4. To power the circuit, take the first wire and plug one end into the "5V" port on the Arduino. Place the other end into the positive power rail on the breadboard.

5. To complete the circuit, take the second wire and plug one end into the "GND" port on the Arduino. Place the other end into the negative power rail on the breadboard.

6. Plug the Arduino into your computer's USB port to turn power on. The LED should remain off by default. To turn the LED on, press and hold the pushbutton

Circuit

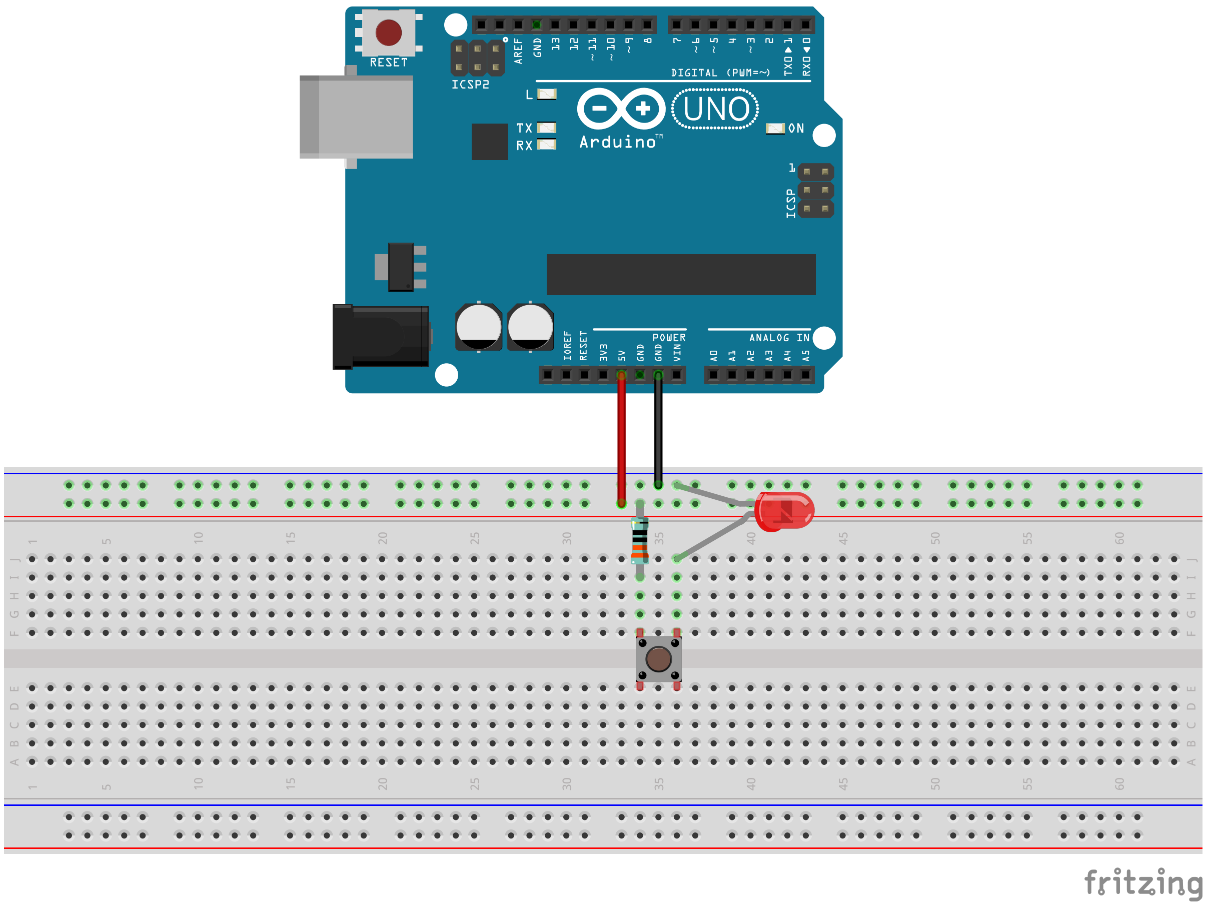


Figure 2.1: Circuit representation for the pushbutton LED project.

2.2 Tunable LED Brightness

Parts Required

- 1x Arduino + USB Cable
- 3x Wires
- 1x Breadboard
- 1x LED
- 1x 330 Ω Resistor
- 1x Potentiometer

Steps

1. Place the potentiometer so that the three pins are all in separate rows on the breadboard.
2. Place the LED so that the short leg (the cathode) is in a row by itself and the long leg (the anode) is in the same row as the middle pin on the potentiometer.
3. Place the resistor so that one leg is in the negative power rail column (next to the blue line) and the other leg is in the same row as the long leg on the LED.
4. Place wires:
   a) Place a wire with one end on the Arduino in the hole labeled “5V”. This will provide power to the circuit. Place the other end of the wire into a row with the leftmost pin of the potentiometer.
   b) Place a second wire with one on the Arduino in one of the holes labeled “GND”. Place the other end of the wire in the same row as the rightmost pin of the potentiometer.
   c) Take a third wire and place one end in the negative power rail column (next to the blue line) and place the other end on the Arduino in the remaining hole with the “GND” label.
5. Plug the Arduino into your computer’s USB port to turn power on. Adjust the potentiometer to change the LEDs brightness. Turning it toward the left will make the LED dimmer until it turns off. Turning it to the right will make the LED increase in brightness.

Circuit

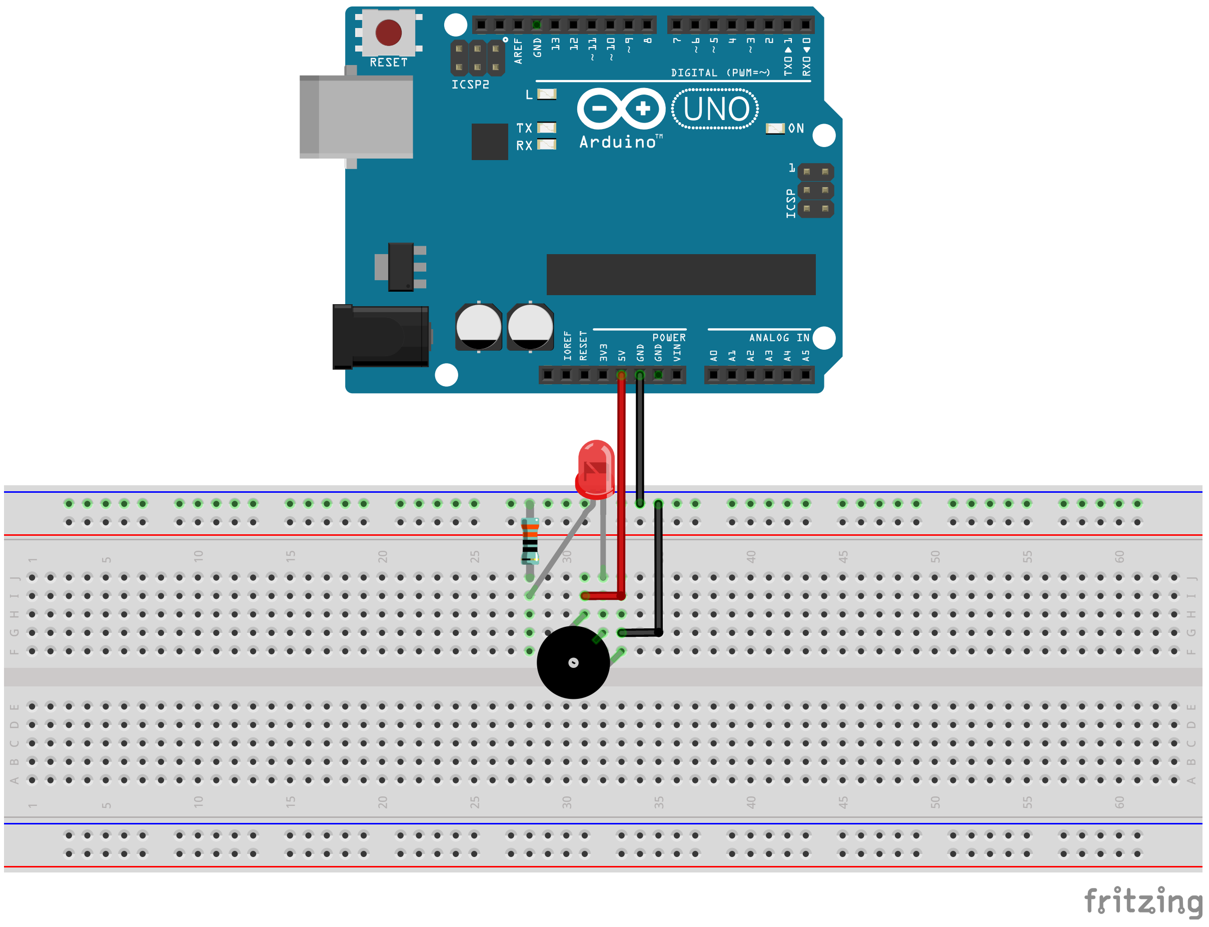


Figure 2.2: Circuit representation for the tunable LED brightness project.

3 Level 2 Projects

3.1 Basic Buzzer

Parts Required

- 1x Arduino + USB Cable
- 2x Wires
- 1x Buzzer

Steps

1. Place the pushbutton so that it straddles the river on the breadboard. The legs of the pushbutton should be bending so that they are pointing perpendicular (away from) the river.

2. Place the buzzer so that the long leg is in the same row as one of the legs of the pushbutton. Placing it further away from the pushbutton will make it easier to place wires later as the buzzer covers most of the row.

3. To power the circuit, take the first wire and plug it into the “5V” port on the Arduino. Place the other end of the wire in the same row as the empty leg above or below the buzzer, depending on where you placed the buzzer.

4. To complete the circuit, take the second wire and plug it into the same row as the short leg of the buzzer (the one not connected to the pushbutton).

5. To turn the buzzer on, press and hold the pushbutton. You should hear a loud, high-pitched sound. To change the tone, follow the “Programmable Buzzer” project example.

Circuit

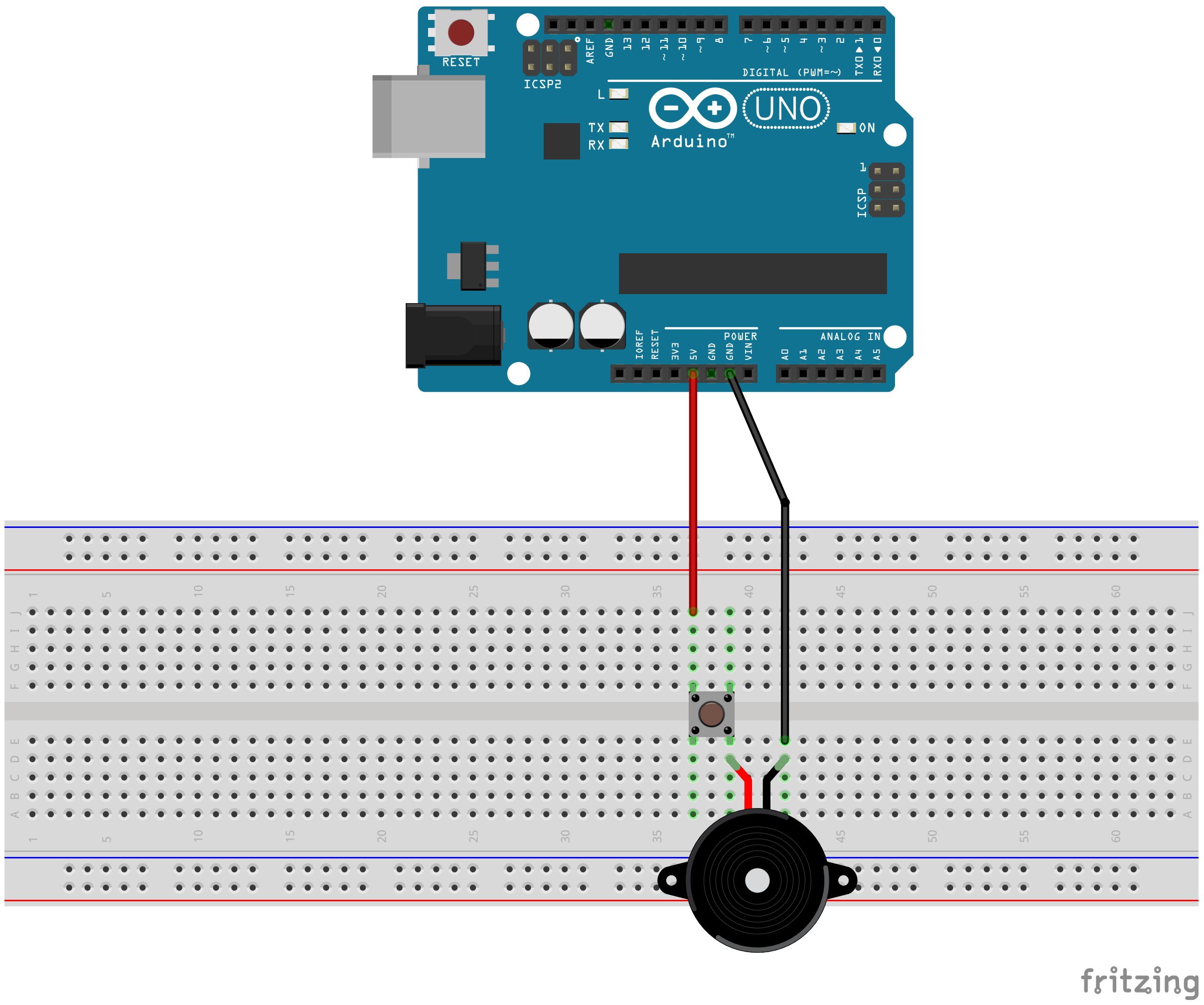


Figure 3.1: Circuit representation for the basic buzzer project.

3.2 Programmable Buzzer

Parts Required

- 1x Arduino + USB Cable
- 2x Wires
- 1x Buzzer

Steps

- **Building the Circuit**

  1. Place the buzzer on the breadboard so that the two pins are in separate rows.
  2. Take the first wire and plug one end into the Arduino port labeled “9”. This is a digital output port with PWM capability. Place the other end of the wire into the same row as the long pin on the buzzer.
  3. To complete the circuit, take the second wire and plug it into the “GND” port on the Arduino. Place the other end of the wire into the same row as the short pin on the buzzer.
  4. You shouldn’t hear anything from the buzzer just yet. To configure how the buzzer will work, we need to write the program using the Arduino IDE and upload it onto the Arduino.

- **Writing the Program**

  1. The script for this program is found below in the Program section. We will walk through the steps to recreate this script in your editor. First, open up the Arduino IDE on your computer. Create a new sketch with `Ctrl+N`.
  2. In the new script, we need to define the port we have connected the buzzer to. In the previous steps, we connected it to pin 9, so we will define that with the line `const uint8_t buzzer = 9;` at the top of the file on line 1.
  3. In the setup function, we need to tell the Arduino that our buzzer pin will be used as an output to send a signal out from the Arduino. To do this, we use the pinMode function. The first argument is the port we are using, buzzer, and the second argument sets the mode for the pin. In this case, we are using the buzzer pin as an output, so we write the line `pinMode( buzzer , OUTPUT );`.

4. The loop function will be executed for as long as the Arduino is powered on. Once it finishes executing the commands within the function, it loops back to the top of the function and continues again. On line 8, we use the tone function to tell the buzzer to emit a tone. This function takes three values: the pin the buzzer is connected to, the frequency of the tone, and the duration of the tone. We pass the variable buzzer to define the pin and use the value 440 to define a 440 Hz tone (or A3). The duration is given as 100 ms so the buzzer will emit a 440 Hz tone for 0.1 seconds.

5. In order for us to hear the buzzer beeping instead of one long tone, we add a delay function call on line 9 after we call the tone function. This will stop the Arduino from continuing for a set amount of time. This time is defined in milliseconds using the single value passed to it. In our example, we have told the Arduino to wait for 200 milliseconds. This value is equal to the duration of the tone plus 100 milliseconds. This small amount of time gives our ears enough time to hear a pause between the tones so that we can hear the buzzer beeping.

6. Now that we have written the program, we can verify that it is correct by clicking the checkmark button on the top left of the Arduino IDE. This button will verify that we have written the code correctly by compiling the program and checking for any syntax errors. The output will be shown in the area at the bottom of the Arduino IDE. If you see any errors, go through the provided script again to make sure you typed everything correctly.

7. Once we have verified that the program is correct, we can upload the program to the Arduino. Plug the Arduino in to the computer using the USB cable and click on the arrow button next to the checkmark button. This will initiate the upload sequence to the Arduino. You can track the progress on the bottom of the window. Once complete, the Arduino IDE will say “Done uploading.” at the bottom of the editor above the output panel.

8. If you have connected the circuit correctly, you will hear the buzzer beeping rapidly about five times per second. Since we haven’t added a way to turn it off, simply unplug one of the wires from the breadboard or unplug the Arduino from the computer to remove power and stop it.

Circuit

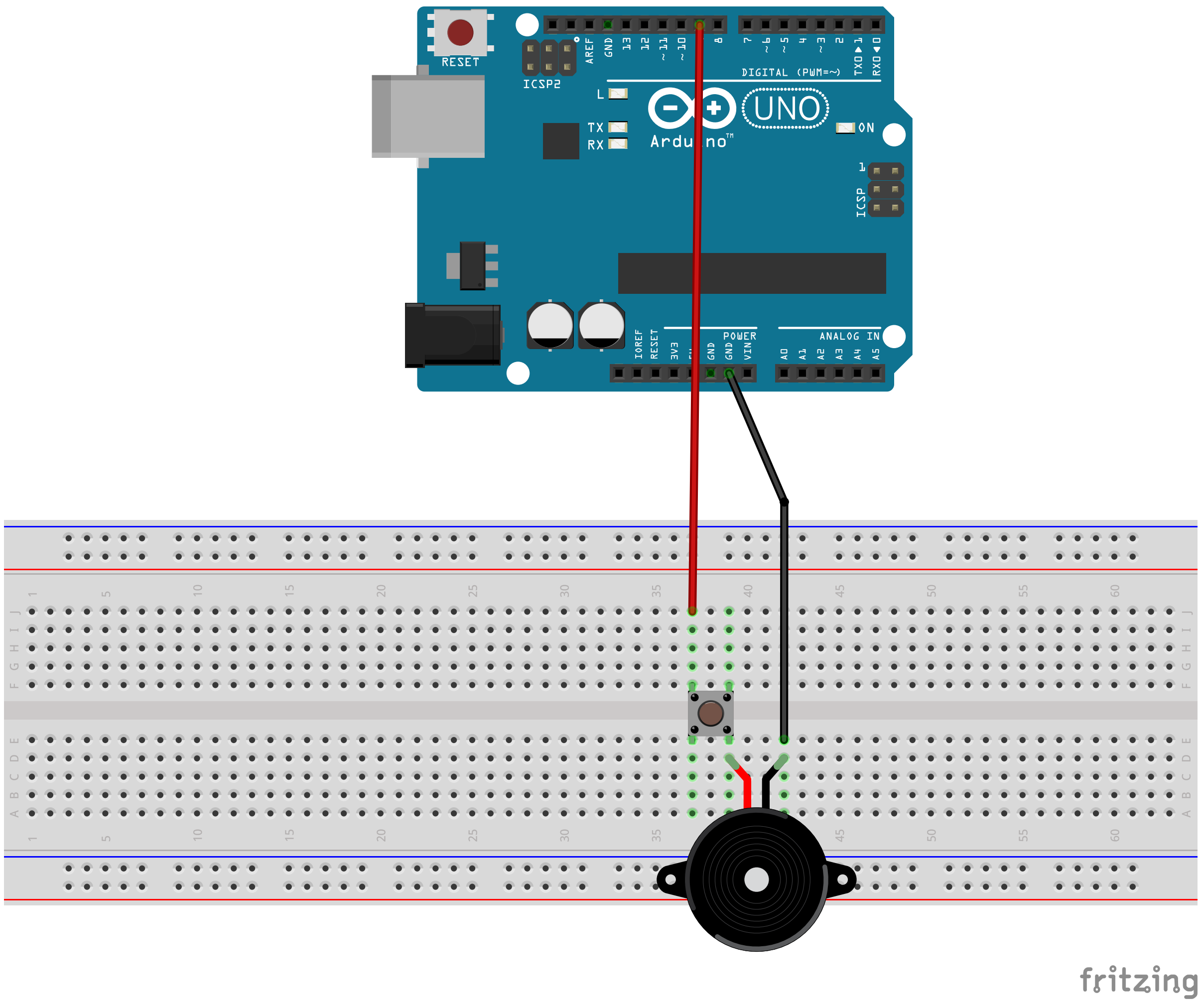


Figure 3.2: Circuit representation for the programmable buzzer project.

Program

```
const uint8_t buzzer = 9;

void setup() {
  pinMode( buzzer, OUTPUT );
}

void loop() {
  tone( buzzer, 440, 100 );
  delay( 200 );
}
```

4 Level 3 Projects

4.1 Ultrasonic Security System

Parts Required

- 1x Arduino + USB Cable
- 1x Ultrasonic Sensor HC-SR04
- 8x Wires
- 1x Buzzer
- 1x LED
- 1x 330Ω Resistor

Steps

- **Building the Circuit**

  1. Place the ultrasonic sensor so that each of the four legs are in a different row on the breadboard. Make sure that the sensor is facing away from the Arduino and any wires so that it does not get any false readings and sound the alarm because of the Arduino or the wires.
  2. Place the buzzer so that the two pins are in separate, empty rows on the breadboard.
  3. Place the resistor so that one leg is in the same row as the short leg of the buzzer and the other leg is in an empty row on the breadboard.
  4. Place the LED so that the anode (long leg) is in an empty row on the breadboard and the cathode (short leg) is in the same row as the second leg of the resistor from the previous step.
  5. Take one wire and plug one end into pin “13” on the Arduino. Plug the other end into a hold on the breadboard in the same row as the pin labeled “Trig” on the ultrasonic sensor.

6. Take a second wire and plug one end into pin “12” on the Arduino. Plug the other end into a hold on the breadboard in the same row as the pin labeled “Echo” on the ultrasonic sensor.
7. To control the LED, we take the third wire and plug one end into pin “8” on the Arduino and the other end into the same row as the anode (long leg) of the LED on the breadboard.
8. To control the buzzer, plug one end of the fourth into pin “9” on the Arduino and the other end into the same row as the long leg of the buzzer on the breadboard.
9. To connect the LED and buzzer to the ground reference, take the fifth wire and plug one end into the negative power rail on the breadboard. Plug the other end into a hole in the breadboard in the row with the short leg of the buzzer and one leg of the resistor.
10. To connect the ultrasonic sensor to the ground reference, plug one end of the sixth wire into the breadboard in the same row as the sensor’s “Gnd” pin and the other end into the negative power rail.
11. To power the sensor, take the seventh wire and plug one end into the breadboard in the same row as the ultrasonic sensor’s “Vcc” pin and the other end into the Arduino’s “5V” port.
12. To complete the circuit, take the eighth wire and plug one end into one of the Arduino’s “GND” ports and plug the other end into any hold in the negative power rail on the breadboard.
13. Once you have built the circuit, continue on to the Arduino steps to program the device.

- **Writing the Program**

1. Create a new project in the Arduino IDE and copy and paste the code in the following Program section.
2. Make sure you have the HCSR04 (the part number for the ultrasonic sensor in your kit) library installed in the Library Manager by selecting the menu options `Tools` > `Manage Libraries...` or pressing `Ctrl+Shift+I`. In the filter search bar, type in “HCSR04”. Scroll down until you see the entry labeled “HCSR04”

with the author name “Martin Sosic”. There are many possible libraries we can use with the ultrasonic sensor, each with slightly different syntax, but this is the library we are using for the example. If you have the library installed, it will say “INSTALLED” next to the version number. Otherwise, hover over the entry and click “Install” in the bottom right corner. You can now close the Library Manager.

3. To verify that the library installed correctly, click on the checkmark icon in the top left to compile the program. If everything is installed correctly, you will see the message “Done compiling” in the output box at the bottom of the screen. If you see an error, check that you copied the provided program correctly and that the library has been installed.

4. Before you upload the program to the Arduino, make sure you read through the program and understand what it is doing. The code below is commented (lines that start with two forward slashes //) to show you what it is doing. The following is a description of how the code works. Once you are done, continue on to Step 5.

   - Line 1 includes the HCSR04 library so that we can use the functions provided by it to read data from our ultrasonic sensor.

```
1 #include <HCSR04.h>
```

   - Lines 3 and 4 define the pins we connected the “Trig” and “Echo” pins to on the Arduino. As described in the previous section, we have connected those to pins 13 and 12, respectively.

```
3 const uint8_t trigger = 13;   // Pin number connected to "
     Trig".
4 const uint8_t echo = 12;      // Pin number connected to "
     Echo".
```

   - Line 5 creates an object that represents our ultrasonic sensor. We need to tell it which pins we connected the trigger and echo pins to so that it knows where to send signals to talk to the sensor.

```
5 UltraSonicDistanceSensor distanceSensor( trigger , echo );
```

- Lines 7 and 8 define the pins where we have connected the buzzer and LED to. In this case, we have connected them to pins 9 and 8, respectively.

```
7 const uint8_t buzzer = 9;      // Pin number connected to
     buzzer.
8 const uint8_t led = 8;         // Pin number connected to
     LED.
```

- Line 9 defines our alert threshold range in centimeters. By default, we have set this to 10 centimeters, but you can change this to any whole number greater than 0. However, the sensor may not be able to pick up very far or very short distances, so you may want to leave this value between 5 and 20 centimeters.

```
9 const uint8_t threshold = 10; // Threshold in centimeters.
```

- In the setup function from line 12 to 17, we first initialize our serial communication with baudrate (the speed at which the data is sent) 9600 bits per second (bps) so that we can send messages from the Arduino back to the serial monitor. This allows us to see on the computer the output of our program. We then define the pin mode for the pins connected to the buzzer and the LED as output modes so that we can turn them on and off.

```
12   // Start serial communication using baudrate 9600.
13   Serial.begin(9600);
14
15   // Set the buzzer and LED pin modes as output.
16   pinMode( buzzer, OUTPUT );
17   pinMode( led, OUTPUT );
```

- The `loop()` function is the main brains of our program and it loops continuously as long as the Arduino is powered on.

- The first thing we do on line 22 is to read the distance to the nearest object in front of the ultrasonic sensor in centimeters. We do this by calling the `measureDistanceCm()` function that is a part of the ultrasonic sensor object we defined on line 5. We store this value as a double (meaning it is a fractional number, e.g., 5.2, 10.6, etc.) in the variable distance.

```
21  // Read the distance in centimeters.
22  double distance = distanceSensor.measureDistanceCm();
```

- We want to send this information back to the computer so we can see what the sensor is reading. To do this, we use the serial communication to first print out the string "Distance: " on line 26. We then print the decimal value of the distance in centimeters on line 27. Finally, we provide the units of the measurement on line 28 by printing out " cm". For this last one, we use the `println()` function (where `ln` stands for `line`) so that our cursor goes to the next line after we display the information. This way each measurement can be shown on a separate line.

```
25  // Print out the distance to the Serial Monitor.
26  Serial.print("Distance: ");
27  Serial.print(distance);
28  Serial.println(" cm");
```

- The if statement on line 33 contains our alert logic. In this statement we are first checking that our distance measurement is valid (meaning that it is a positive value greater than zero: `distance > 0`). We need to check this because, if there is nothing close to the sensor, we don't read anything and the output value is -1.0.

```
31  // Check that we have a valid reading and the distance
32  // is within our alert threshold.
33  if( distance > 0 && distance < threshold )
```

- If we have a valid distance value, we want to also check if the object is within the alert range. We do this by first using the syntax "&&" meaning "and". The second condition is that the distance is less than the threshold range we set on line 9. Line 33 can be read in English as "if distance is greater than 0 AND distance is less than threshold, then do the following". The "following" means we will execute the commands on lines 35 through 46 if the distance is greater than 0 and is less than our threshold alert range.

– When we sense an object within our alert range, we want to buzz the buzzer and blink the LED. To do this, we first turn on the buzzer for 100 milliseconds with a frequency of 440 Hz on line 36 using the `tone()` command. We also turn the LED on on line 37. We then wait 100 milliseconds on line 40 so we can see the LED turn on, then we turn the LED off on line 43. We then wait another 100 milliseconds on line 46 so that our eyes can recognize that the LED has turned off. Otherwise, the code might go so fast that we can not register that the LED is blinking.

```
35     // Turn the buzzer and the LED on.
36     tone( buzzer, 440, 100 );
37     digitalWrite( led, HIGH );
38
39     // Wait for 100 milliseconds.
40     delay( 100 );
41
42     // Turn the LED off.
43     digitalWrite( led, LOW );
44
45     // Wait for 100 milliseconds.
46     delay( 100 );
```

– After the alert actions have been executed, or if there was no object within the alert range, we go back to the beginning of the loop function, check the distance again on line 22, and start the process all over again for as long as the Arduino is on.

5. Plug the Arduino into your computer and make sure the correct port and board options are selected under the `Tools` menu. Open the `Serial Monitor` using

the menu options `Tools` > `Serial Monitor` or by pressing `Ctrl+Shift+M`. This way we can see the output from the code as soon as it is done uploading.

6. Upload the program to the Arduino by clicking on the right arrow icon in the top left, by pressing `Ctrl+U`, or by selecting the menu option `Sketch` > `Upload`. The program will take a few seconds to upload and then you should see output on the Serial Monitor. If you move within the threshold alert range (the default provided in the program is 10 centimeters), the LED will blink and the buzzer will buzz at you. To change the threshold range, change the value of the `threshold` variable in the code to the desired value in centimeters.

Circuit

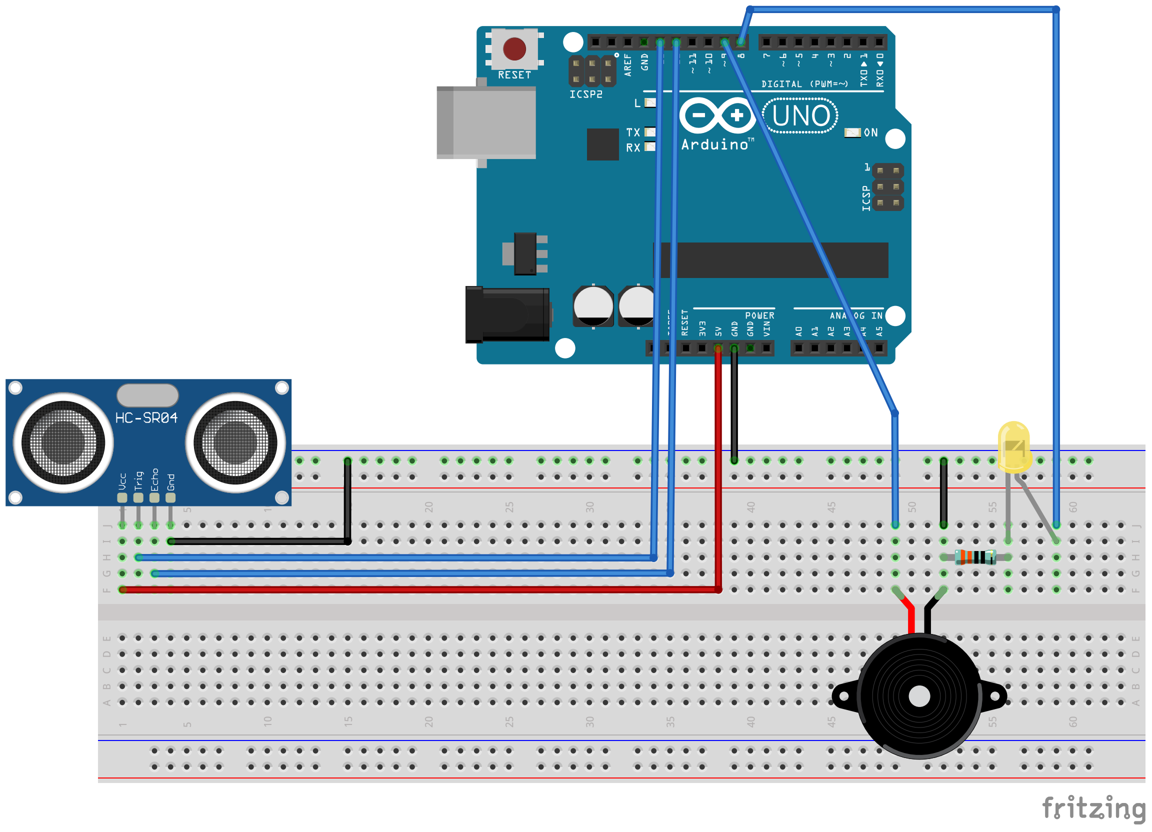


Figure 4.1: Circuit representation for the ultrasonic security project.

Script

```
#include <HCSR04.h>

const uint8_t trigger = 13;   // Pin number connected to "Trig".
const uint8_t echo = 12;      // Pin number connected to "Echo".
UltraSonicDistanceSensor distanceSensor( trigger, echo );

const uint8_t buzzer = 9;     // Pin number connected to buzzer.
const uint8_t led = 8;        // Pin number connected to LED.
const uint8_t threshold = 10; // Threshold in centimeters.

void setup() {
  // Start serial communication using baudrate 9600.
  Serial.begin(9600);

  // Set the buzzer and LED pin modes as output.
  pinMode( buzzer, OUTPUT );
  pinMode( led, OUTPUT );
}

void loop() {
  // Read the distance in centimeters.
  double distance = distanceSensor.measureDistanceCm();


  // Print out the distance to the Serial Monitor.
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");


  // Check that we have a valid reading and the distance
  // is within our alert threshold.
  if( distance > 0 && distance < threshold )
  {
    // Turn the buzzer and the LED on.
    tone( buzzer, 440, 100 );
    digitalWrite( led, HIGH );

    // Wait for 100 milliseconds.
    delay( 100 );

    // Turn the LED off.
    digitalWrite( led, LOW );

    // Wait for 100 milliseconds.
    delay( 100 );
  }
}
```

5 Level 4 Projects

5.1 TFT Etch-a-Sketch

Parts Required

- 1x Arduino + USB Cable
- 1x 1.8” TFT Display
- 18x Wires
- 1x Pushbutton
- 2x Potentiometer
- 1x 1kΩ Resistor

Steps

- Building the Circuit

  1. Place the 1.8” TFT Display so that all eight pins are in separate, empty rows on the breadboard.
  2. Place one potentiometer so that all three pins are in separate, empty rows on the breadboard to one side of the TFT display.
  3. Place the second potentiometer so that all three pins are in separate, empty rows on the breadboard to the other side of the TFT display.
  4. Place the pushbutton so that it straddles the river in the center of the breaboard and all four pins are in empty rows.
  5. Connect the TFT display as shown in Figure 5.1. The pins are labeled as follows from left to right with the screen facing up: `LED`, `SCK`, `SDA`, `A0`, `RESET`, `CS`, `GND`, `VCC`. The labels shown in the figure are the same as those on the bottom of the display provided in the kit. The wiring connections to the Arduino are shown in Table 5.1. The `GND` and `VCC` pins are connected to the negative and positive power rails on the breadboard, respectively.

TFT Pin	**Arduino Pin**
LED	3.3 V
SCK	13
SDA	11
A0	9
RESET	8
CS	10

Table 5.1: 1.8” TFT wiring connections to the Arduino Uno.

6. On both potentiometers, connect wires in the following manner:
   - Take the first wire and plug one end into a hole in the positive power rail on the breadboard. Plug the other end into a hole in the same row as either the left or right pin on the potentiometer.
   - Take the second wire and plug one end into a hole in the negative power rail on the breadboard. Plug the other end into a hole in the same row as the opposite pin on the potentiometer that you used for the first wire.
   - Take the third wire and plug one end into pin “A0” for the left potentiometer or “A1” for the right potentiometer. Plug the other end into a hole in the same row as the center pin on the potentiometer. Note that the potentiometer plugged into “A0” will control the movement along the longer axis of the display and the potentiometer plugged into “A1” will control movement along the shorter axis of the display.
7. To connect the pushbutton, take one wire and plug one end into a hole in the positive power rail on the breadboard. Plug the other end into the same row as one of the pins on the pushbutton. Take a second wire and plug one end into pin “2” on the Arduino. Plug the other end into a hole in the same row as the second pin on the pushbutton.
8. Take the 1 kΩ resistor and plug one leg into a hole in the same row as the wire connecting the pushbutton to pin “2” from the last step. Plug the other leg into the negative power rail on the breadboard. This resistor is known as a *pull-down* resistor that pulls the signal into pin “2” *down* to ground so that the Arduino program can recognize the button as not pressed. If we don’t do this, the output signal from the pushbutton is called *floating*, meaning it has

no defined value. This can potentially result in the Arduino thinking the button is pressed when it is not, erasing our drawings.

9. Once everything is connected, we want to power our circuit by plugging one end of another wire into the "5 V" pin on the Arduino and the other end into the positive power rail on the breadboard. Complete the circuit by taking another wire and plugging one end into one of the "GND" pins and plug the other end into the negative power rail on the breadboard.
10. Now that the circuit is built, go to your computer, open the Arduino IDE, and proceed to the next section to write the Etch-a-Sketch program.

Circuit

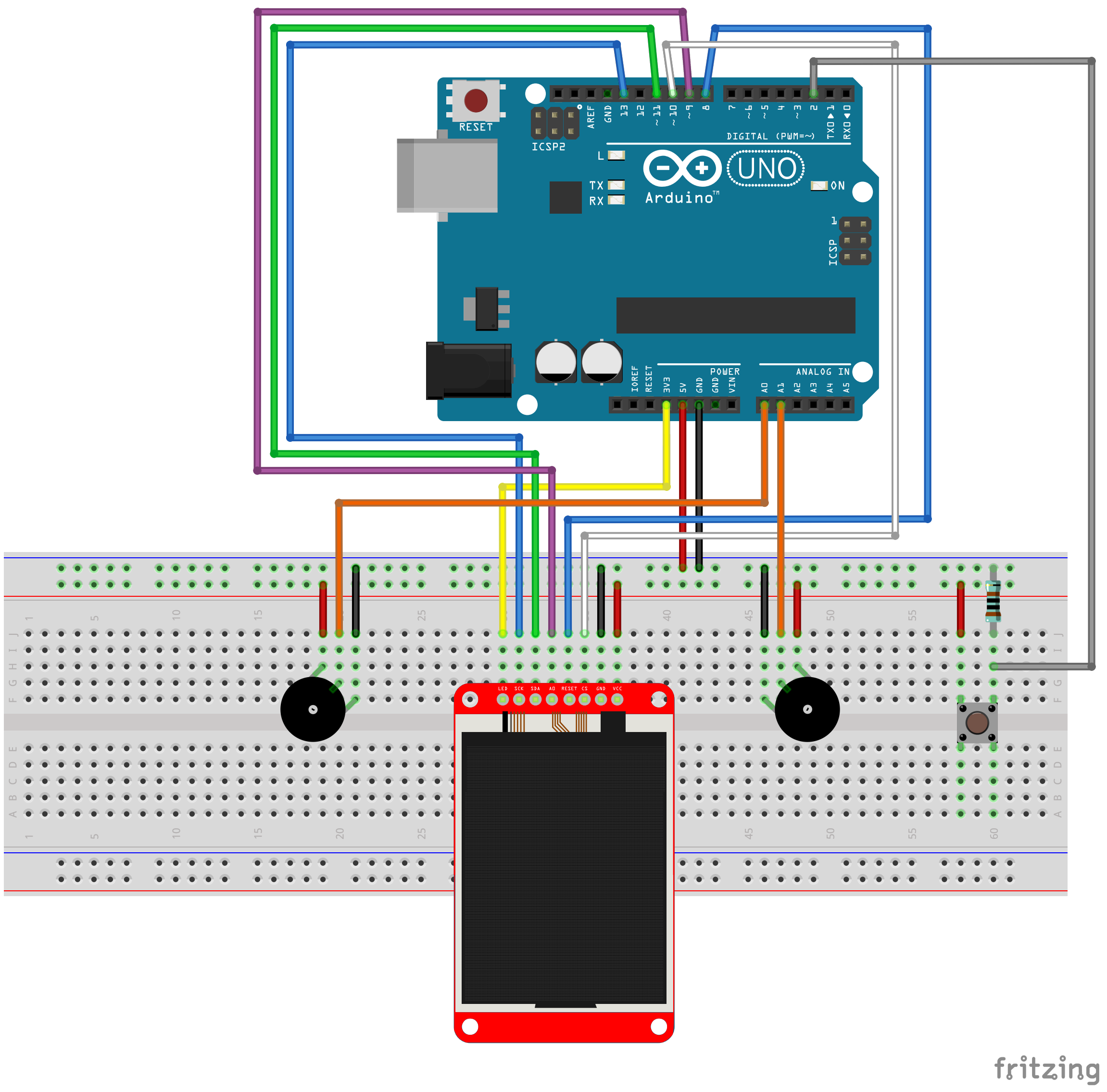


Figure 5.1: Circuit representation for the TFT Etch-a-Sketch project.

Script

```
#include <TFT.h>  // Arduino LCD library
#include <SPI.h>  // Communication with LCD


// Define pin connections for data
// transfer from the TFT display.
const uint8_t cs = 10;  // ChipSselect
const uint8_t dc = 9;   // Data/Command. A0 on our board.
const uint8_t rst = 8;  // Reset


// Create an object representation of the TFT display.
TFT TFTscreen = TFT(cs, dc, rst);


// Define a new object type to represent a pixel point.
typedef struct
{
  uint8_t x;
  uint8_t y;
} Pixel;


// Create objects to track the current and last position.
Pixel currPnt;
Pixel lastPnt;


// pin the erase switch is connected to
const uint8_t erasePin = 2;

void setup() {
  // Swap between RGB and BGR.
  // Valid values: 0, 1
  TFTscreen.invertDisplay( 1 );


  // Set our cursor at the center of the screen.
  currPnt.x = TFTscreen.width() / 2;
  currPnt.y = TFTscreen.height() / 2;
  lastPnt.x = TFTscreen.width() / 2;
  lastPnt.y = TFTscreen.height() / 2;


  // Set the pushbutton as an input signal.
  pinMode(erasePin, INPUT);

  // Initialize the screen
  TFTscreen.begin();

  // Make the background black
  // using the RGB color (0, 0, 0).
  TFTscreen.background(0, 0, 0);
}

void loop() {
  // Read the potentiometers on A0 and A1
  uint16_t xValue = analogRead(A0);
  uint16_t yValue = analogRead(A1);


  // Convert the input values to a valid range.
  // The display is 160 pixels wide, so we take the
  // x range 0 to 159.
  // It is 128 pixels tall, so we take the
  // y range 0 to 127.
  currPnt.x = map( xValue, 0, 1023, 0, 159 );
  currPnt.y = map( yValue, 0, 1023, 0, 127 );


  // Draw the last point location white,
  TFTscreen.stroke( 255, 255, 255 );
  TFTscreen.point( lastPnt.x, lastPnt.y );


  // Draw the current point location blue.
  TFTscreen.stroke( 255, 0, 0 );
  TFTscreen.point( currPnt.x, currPnt.y );


  // Store the current location as the last location
  // for the next iteration.
  lastPnt.x = currPnt.x;
  lastPnt.y = currPnt.y;


  // Read the value of the pushbutton, and erase the screen if pressed
  if( digitalRead( erasePin ) == HIGH )
  {
    TFTscreen.background(0, 0, 0);
  }

  delay( 33 );
}
```