Future Innovators Workshop Handbook

Prepared by IEEE-HKN Lambda Zeta and the IEEE UNT Student Branch

For comments, questions, or suggestions, contact Nicholas Chiapputo at chiapputo@ieee.org.

Revision Date: November 13, 2021

Welcome to the Future Innovators Workshop

Welcome to the Future Innovators Workshop (FIW)! FIW is designed, developed, and hosted by IEEE student members at the University of North Texas. Our goal is to introduce K-12 students to the wide world of programming and electronics by providing Arduino electronics kits. Each session of FIW provides students with a one-hour workshop where we introduce the components in the kit and walk students through a complete project. After the workshop, we encourage students to continue to explore with their kit and make new and exciting projects!

This handbook serves as a first introduction to electronics and Arduino programming to get students up and running. Our aim is to include enough information to quickly form a basic foundation for students to begin exploring. We highly encourage students and parents alike to continue to research and learn about any topics found in this handbook. We also encourage contacting us for any help or questions in getting started or clarifications needed in this handbook. This handbook is expected to be updated over time with more projects and learning material, so check back every month or two and look at the revision table for new updates!

The first section introduces the Arduino, the electronic components in the kit, and the basics of electronics such as voltage, current, resistance. The following sections include projects to give students ideas on what they can make. The projects are sorted by difficulty level starting with basic electronics and increasing into Arduino programming and then more advanced electronics and programming that requires some theory background. Our goal with this handbook is to give students some direction and a starting point to continue learning and exploring well after the conclusion of the workshop.

Revision History

Changes from February 2021 to November 2021

Contents

1 Introduction

1.1 Arduino

Arduino is an open-source electronics hardware and software company that designs many single-board microcontrollers. A microcontroller is a small computer that has a microprocessor, memory, and programmable input/output peripherals. Microcontrollers are used in many electronic and automated products such as remote controls, e-readers, toys, appliances, power tools, and so much more.

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. Because of their easy-to-use development environment, their microcontrollers are one of the most popular starting points for getting into the world of programming and electronics. Their most popular microcontroller, the Arduino Uno (Figure 1.1), is the one included in your kit.

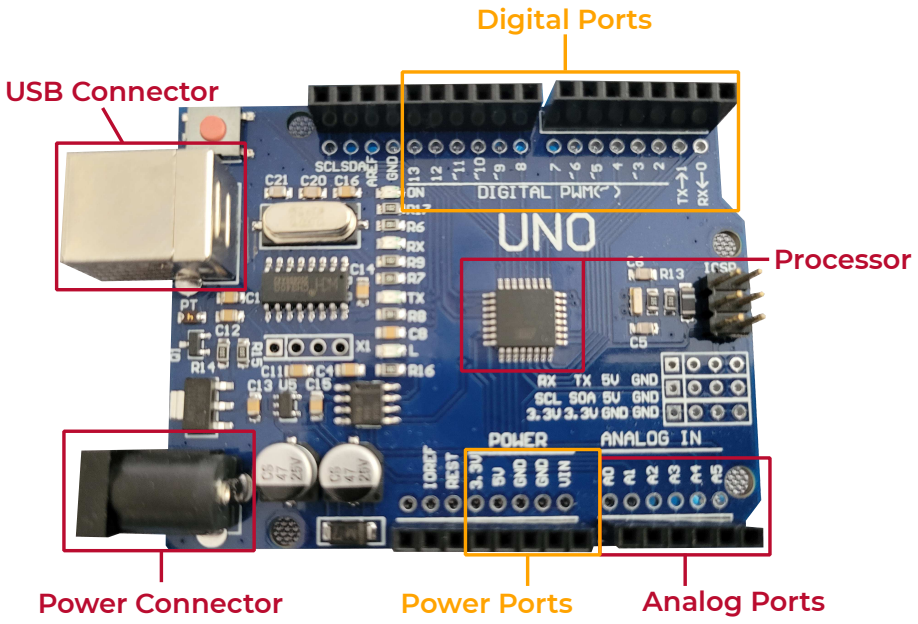


Figure 1.1: An Arduino Uno.

Arduino Uno

Processor

The brain of the Arduino Uno is in the microprocessor. This processor (specifically, an ATmega328p) is the small integrated circuit (IC) chip located in the center of the board. When you write a program, this is the part that will execute all of the instructions that you give it.

Ports

To measure input signals or to send output signals, the Arduino Uno has multiple ports along the edge of the board that are connected directly to the microprocessor. These ports are in the square holes where you can plug in your jumper wires to connect to an external circuit. They allow the Uno to control a circuit or read in information from the circuit using voltage values.

Of these ports, 13 are digital and 6 are analog. The digital ports are 1 through 13 and the analog ports are labeled "A0" through "A5". Digital ports can only read in high or low values (0 V and 5 V, respectively) while analog ports can read and write any voltage value between 0 V and 5 V.

Some of the digital ports also have other functionalities. The digital ports with a "~" next to the number can also produce pulse width modulation (PWM) signals. This type of signal is a square wave that is high (at 5 V) for a percentage of the time (known as the duty cycle). Digital ports 0 and 1 are by default mapped to the serial communication that occurs over USB to your computer. Port 0 receives the data from the computer and port 1 transmits it.

Next to the analog ports are a number of ports used for power. In particular, there are two ports labeled "GND" (ground), one port labeled "5V", and one port labeled "3.3V". The GND ports are used as a reference voltage, or 0 V, for your circuits while the other two ports supply 5 V and 3.3 V, respectively. Usually, you will connect just the 5 V and GND ports to your circuit to power it, though sometimes you will need to use the 3.3-V port for some electronic components that will break if you supply too high of a voltage.

USB

In order to program the Arduino Uno and communicate with it, you need to connect it to

a computer using a USB cable. In your kit, the Uno comes with a blue USB cable. The large end (USB type B) connects to the large port on the end of the Uno while the smaller end (USB type A) connects to a USB port on your computer.

Power

While the USB port does supply power to the Arduino Uno, you can also power it using an external battery. You can connect a 9-V battery and plug it into the barrel jack connector at the end of the board. On the Uno, there is a voltage regulator that will step down the 9-V input to 5 V and 3.3 V in order to protect the components on the board.

Additional Resources

- **Arduino Website** https://www.arduino.cc/
- **Arduino Tutorials** https://www.arduino.cc/en/Tutorial/HomePage/ https://www.arduino.cc/en/Guide/ArduinoUno/
- **Arduino Uno Projects** https://create.arduino.cc/projecthub?by=part&part_id=8233

1.2 Arduino IDE

The Arduino Integrated Development Environment (IDE) is an application that allows you to write code to execute on an Arduino microcontroller or other Arduino-compatible devices (such as ESP32). The IDE can be downloaded from Arduino's website (https://www.arduino.cc/en/software/) by clicking on the download link appropriate for the operating system you are using[1]. Alternatively, Windows users can use the Windows app store to download the software.

[1] For Chromebook users, you must install the Arduino Create for Education app from the Chrome store or use the Arduino Web Editor.

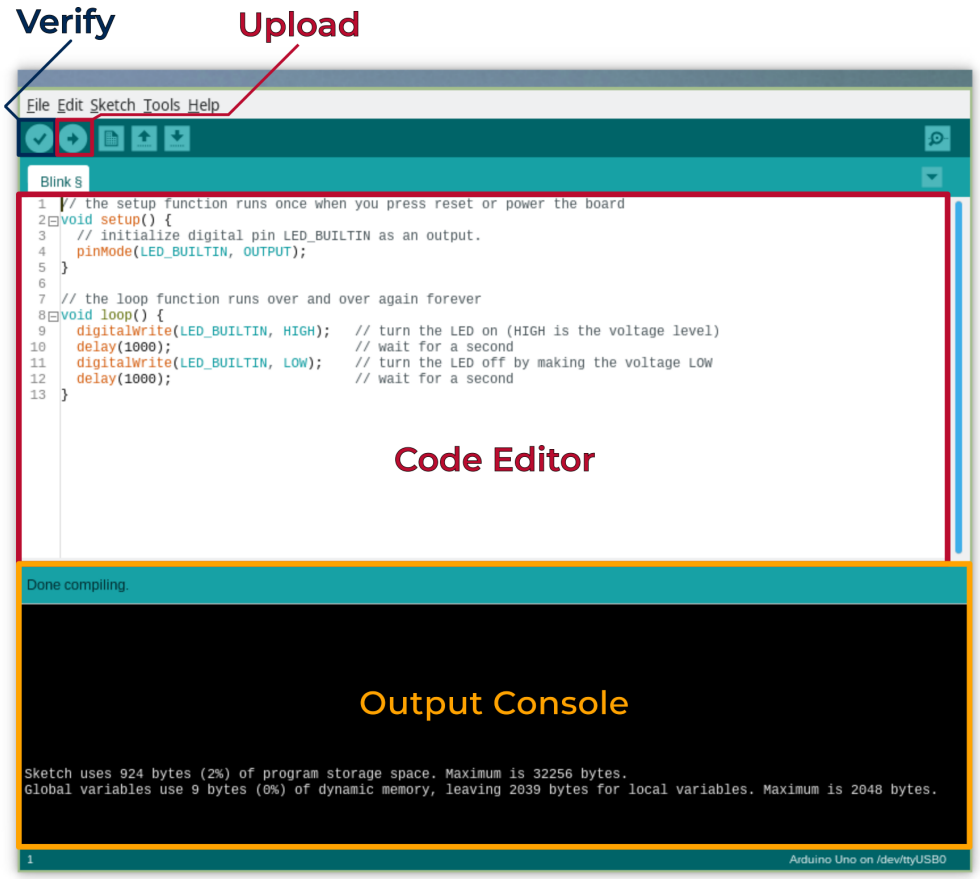


Figure 1.2: The Arduino Integrated Development Environment (IDE).

Figure 1.2 shows the Arduino IDE. The IDE contains a code editor; an output console; a button to verify the code; and a button to upload the executable code to the connected Arduino. In the code editor, you can write and edit the code that will be executed on your Arduino. The output console will show any error messages when verifying or uploading your code to the Arduino. This can be used to debug any problems in the code.

Built-in Examples

The IDE comes with some basic examples that can be found through the menu bar in the `File > Examples` section. To test that the connection to your Arduino is working, you can use the `Blink` example under `01.Basics` that requires no circuit to function. This example will blink the on-board LED labeled *L* on your Arduino Uno. This is the default program that comes loaded onto your Arduino Uno from the factory, so you may notice

that the LED is already blinking when you plug it in to your computer. There are many other examples built into the IDE as well as some that come when you install third-party libraries. These examples are usually well-documented so that you can read through them and understand exactly how they work.

Verify Your Code

The verify button (with the check mark) is used to compile the code you have written in the editor and make sure that the syntax (how it is written) is correct and will execute on your board. However, this doesn't check that there won't be any errors while the code is running (for example, that you added instead of subtracted). A successful verification will display a message in the output console that begins with `Sketch uses...`. This tells you how much of your program memory (also known as flash memory) is taken up by the program you have written. This is a very important part of developing for microcontrollers as they don't have very much memory compared to your computer! The message will also tell you how much of your dynamic memory (also known as SRAM) is used by your global variables. If there are any errors, the top of the output console will turn orange, an error message will appear in the console, and the line where the error occurred will be highlighted in orange in the code editor. If you can't figure out what the error message means, look to the internet!

Upload Your Code

If you have verified your code and the IDE recognizes your device, you're ready to upload the code to the device. The upload button (with the right arrow) is used to upload the code to your device. This button will also verify before it uploads the code, but it is best practice to manually verify before you click the upload button. If the upload is successful, the top bar of the output console will read "Done uploading." and the code will be executing on your device.

Connecting a Device

If your device is not automatically recognized by the IDE after plugging it in to the computer, you must manually select the port. To find the port your Arduino is connected to on Windows, open `Device Manager`, expand the `Ports (COM & LPT)` section, and look

for an item that either has “Arduino” or contains “CH340” (this is the serial communication chip many Arduino clones use). The port number will look like `COMxx`. This is the port you will select in the IDE. For Mac, open a terminal, type `ls /dev/*`, and note the port number listed for either `/dev/tty.usbmodem*` or `/dev/tty.usbserial*`. For Linux, open a terminal, type `ls /dev/tty*`, and note the port number listed for `/dev/ttyUSB*` or `/dev/ttyACM*`. If you are not able to find the port your device is connected to, you may need to install the drivers. Some computers do not have the CH340 drivers installed by default or with the Arduino IDE. To download and install them, follow the guide at https://learn.sparkfun.com/tutorials/how-to-install-ch340-drivers.

Once you have found the port your device is connected to, select it in the menu option `Tools > Port` and choose the serial port that you just found in the previous step. Make sure to also tell the IDE that you are using an Arduino Uno by going into the menu bar and selecting `Tools > Board > Arduino AVR Boards > Arduino Uno`.

Saving Your Code

If you have modified the example code or written your own code, you will need to save the file somewhere on your computer. Arduino script files are saved in `.ino` files. While Arduino code is written in C++, these `.ino` files tell the Arduino IDE to treat them a little differently when compiling. When saving the `.ino` files, they must have the same name as the folder that they’re in.

1.3 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.

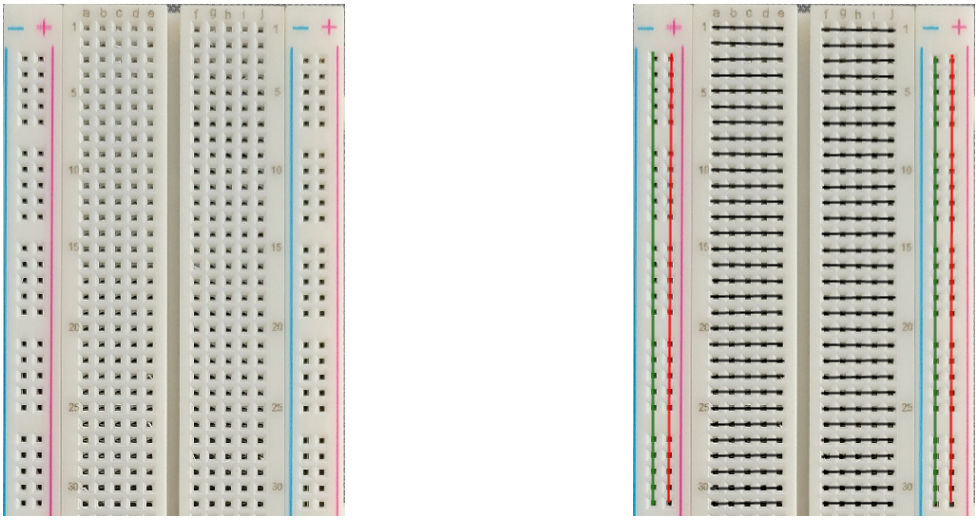


Figure 1.3: Solderless breadboard and internal connections.

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.

1.4 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.

1.5 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.

1.6 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 1.4: Resistor Color Code Chart

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

1.7 Oscilloscope

An oscilloscope is an electronic test instrument that measures and displays voltage over time. Engineers commonly use oscilloscopes to debug circuits to either find problems or to validate that a circuit is working as intended. The display of an oscilloscope plots the voltage on a 2-dimensional graph with the x-axis (horizontal) being time and the y-axis (vertical) being voltage. Oscilloscopes can sometimes be combined with function generators (equipment that can generate arbitrary waveforms and signals) and power supplies to form what are known as test benches. These test benches are all-in-one instruments for testing and designing circuits.

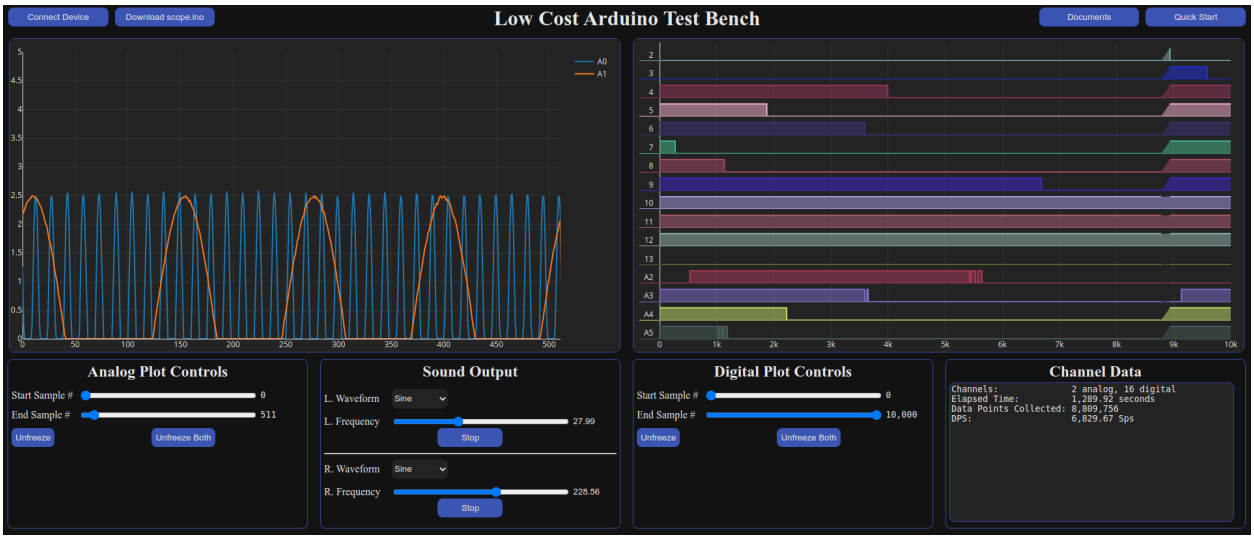


Figure 1.5: The online Arduino oscilloscope at https://ieeeunt.org/lco/.

Typically, these oscilloscopes and test benches can be very expensive and cumbersome. We have created a simple, free-to-use, browser-based Arduino oscilloscope that is hosted on our website at https://ieeeunt.org/lco/ and shown in Figure 1.5. This oscilloscope relies on the Serial API developed by Google Chrome and as such, can currently only be used on Chrome-based browsers such as Vivaldi and Google Chrome. If you want to check if your browser is supported, open the page and the "Channel Data" text box in the bottom left will display "Serial API found!" if your browser is supported.

In order to use the oscilloscope, you must download the `.ino` Arduino script by clicking on the "Download scope.ino" button on the web page. This script will read the analog

voltage (over the 0-V to 5-V range) from the analog ports “A0” and “A1” and digital inputs (either 0 or 1, HIGH or LOW) for the remaining analog ports “A2” through “A5” and for the digital ports “2” through “13”.

To run the script on your Arduino Uno, first load the script into the Arduino IDE and upload it to your Arduino Uno[2]. To test the analog oscilloscope, a simple circuit can be built using only a potentiometer as shown in Figure 1.6. In this circuit, the voltage read into the first channel through “A0” can be adjusted using the potentiometer. This reading will be reflected in the analog oscilloscope on the web page. To test the digital logic analyzer on the page, buttons or switches can be connected to the digital ports and switched or pressed on and off.

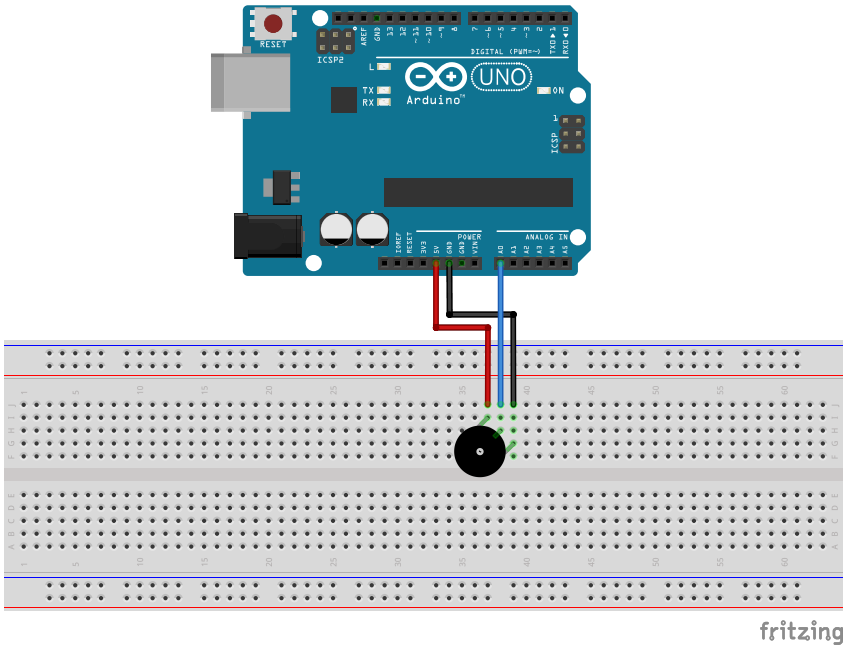


Figure 1.6: A basic circuit to showcase the analog functionality of the online oscilloscope.

On the page beneath the two plots are four other panels. The first and third panels both have two sliders that control the viewing window on the analog and digital views, respectively. Both panels also have two buttons to freeze only the analog or digital view or freeze both views.

[2]The Arduino script has not been tested on devices other than the Arduino Uno. Because it uses register-level programming to maximize the sampling rate, it may not be portable to other devices.

The second panel has two controls to output sound from the left and right channels out of your computer. Here, you can select different output waveforms (sine, square, sawtooth, and triangle) and the frequency, or tone, of the output waveform for each channel separately. By plugging the 3.5-mm audio cable from the computer into the audio jack included in the FIW kit, you can use this audio output as a basic function generator for your circuits. Remember though, that the output from your computer jack oscillates between a positive and negative voltage and an Arduino can only handle positive voltage between 0 V and 5 V. So you will need to build a simple circuit to either rectify the voltage (i.e., remove the negative component), or add a DC offset to increase the voltage so that the lowest point is at or above 0 V and the highest point is at or below 5 V.

1.8 Integrated Circuit (IC)

An integrated circuit (IC) is an electronic circuit that is formed on a small piece of semiconductor material. An IC is housed inside a plastic enclosure with pins connecting to the internal circuitry. An IC performs the same function as a larger circuit that was made from individual components. Each IC has an indentation to signify the 'top' or 'starting pin' on the chip. Pin numbering on ICs begins with pin 1 on the top left of the chip and continues in a 'U' shape as seen in Figure 1.7.

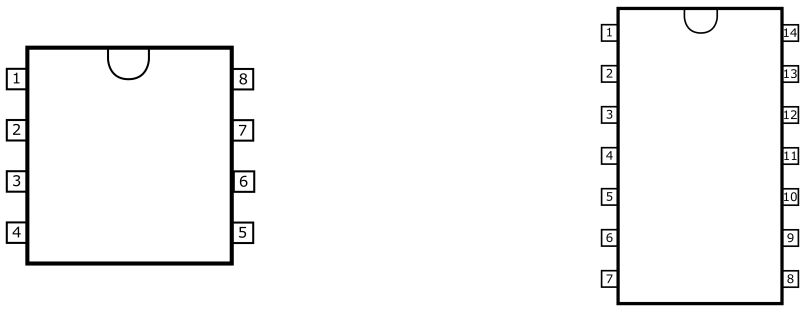


Figure 1.7: 8-pin and 14-pin IC chip outline.

Every chip has a corresponding pinout diagram to explain how to connect IC to the rest of your circuit. All chips need a supply of power and ground in order to properly function.

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

3. Place the LED so that the anode (long leg) is in the same row as the other leg of the pushbutton and the cathode (short leg) is in the negative power rail of the breadboard.

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

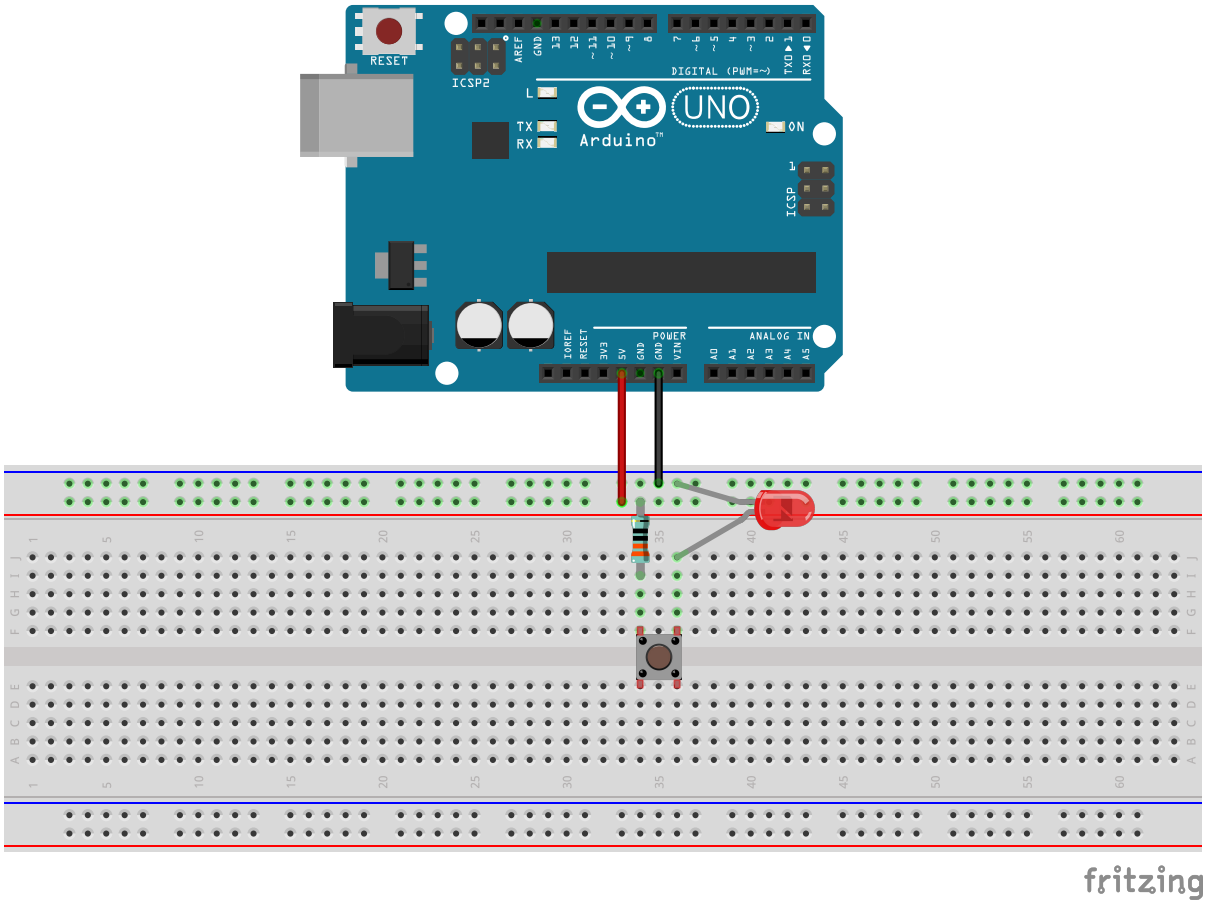


Figure 2.1: Circuit representation for the pushbutton LED project.

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. Using the dial potentiometer, you may need to place potentiometer at an angle.

2. Place the LED so that the long leg (the anode) is in a row by itself and the short leg (the cathode) 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 port 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 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 port with the “GND” label.

    c) Take a third wire with one end in the negative power rail and the other end in the same row as the rightmost pin of the potentiometer.

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

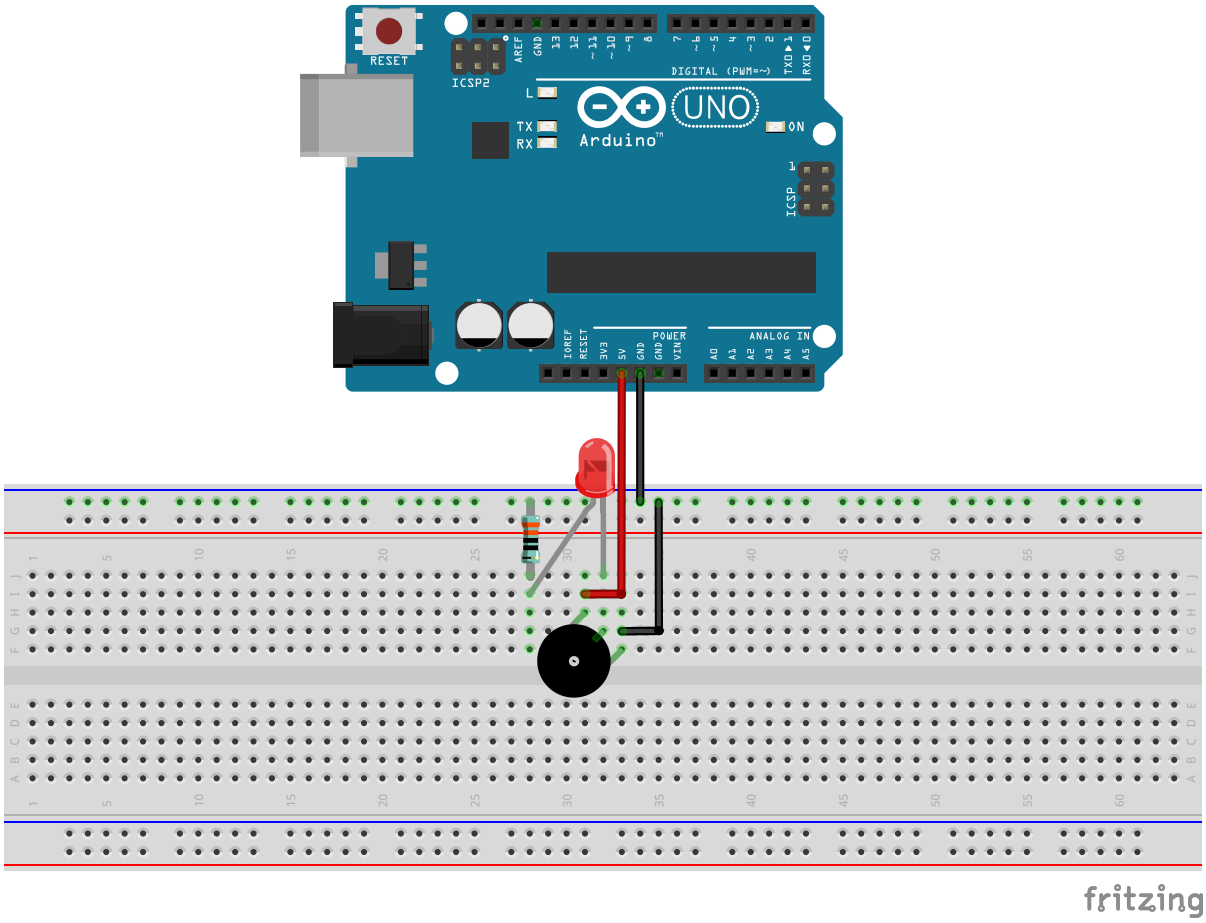


Figure 2.2: Circuit representation for the tunable LED brightness project.

3 Level 2 Projects

3.1 Basic Buzzer

Parts Required

- 1x Arduino + USB Cable
- 2x Wires
- 1x Breadboard
- 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 in Chapter 3.2.

Circuit

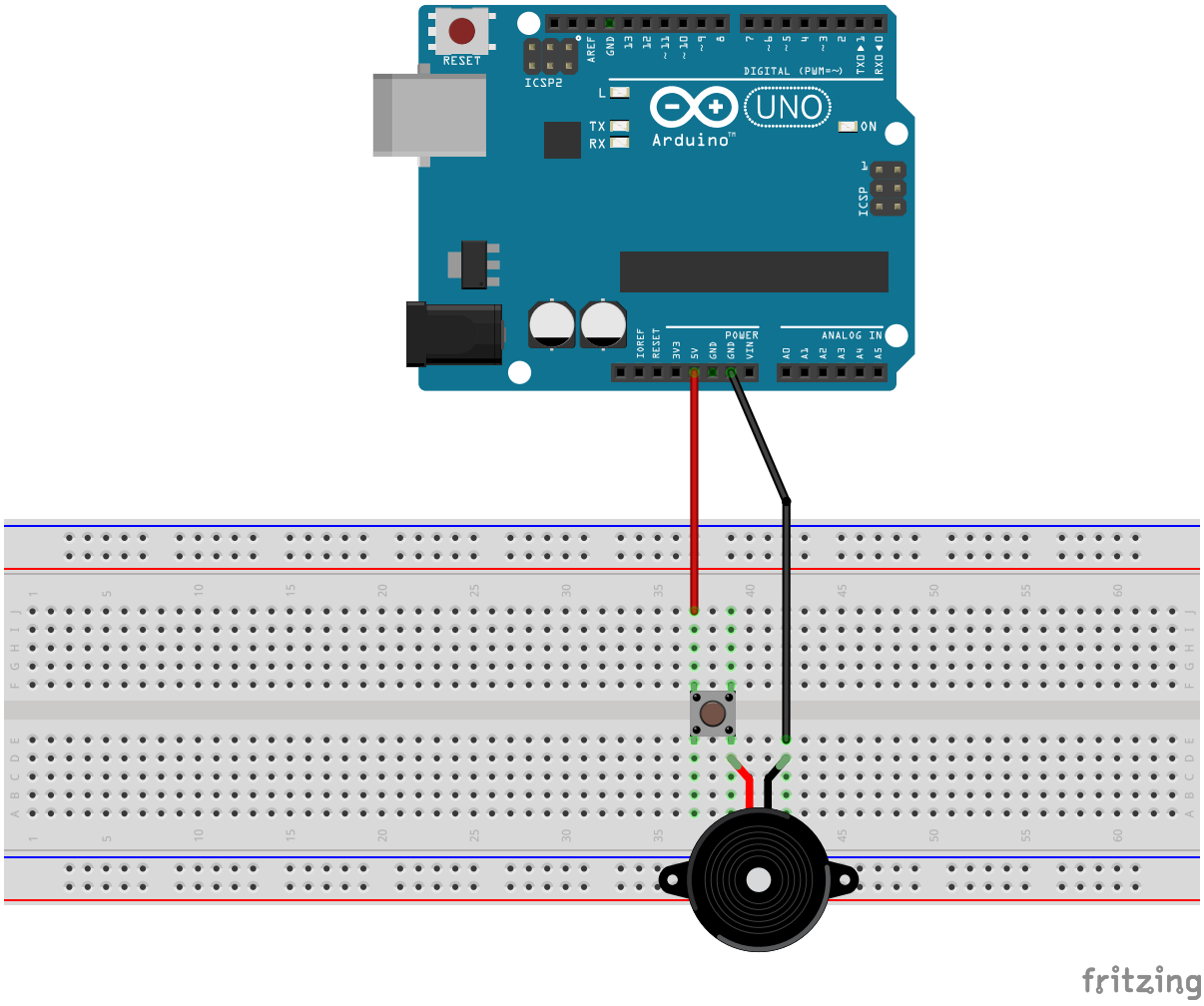


Figure 3.1: Circuit representation for the basic buzzer project.

3.2 Programmable Buzzer

Parts Required

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

Steps

- **Building the Circuit**

  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. 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 in the same row as the empty leg of the pushbutton. Remember that the legs of the pushbutton are connected across the ravine and the legs next to each other are not electrically connected.
  4. 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.
  5. 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

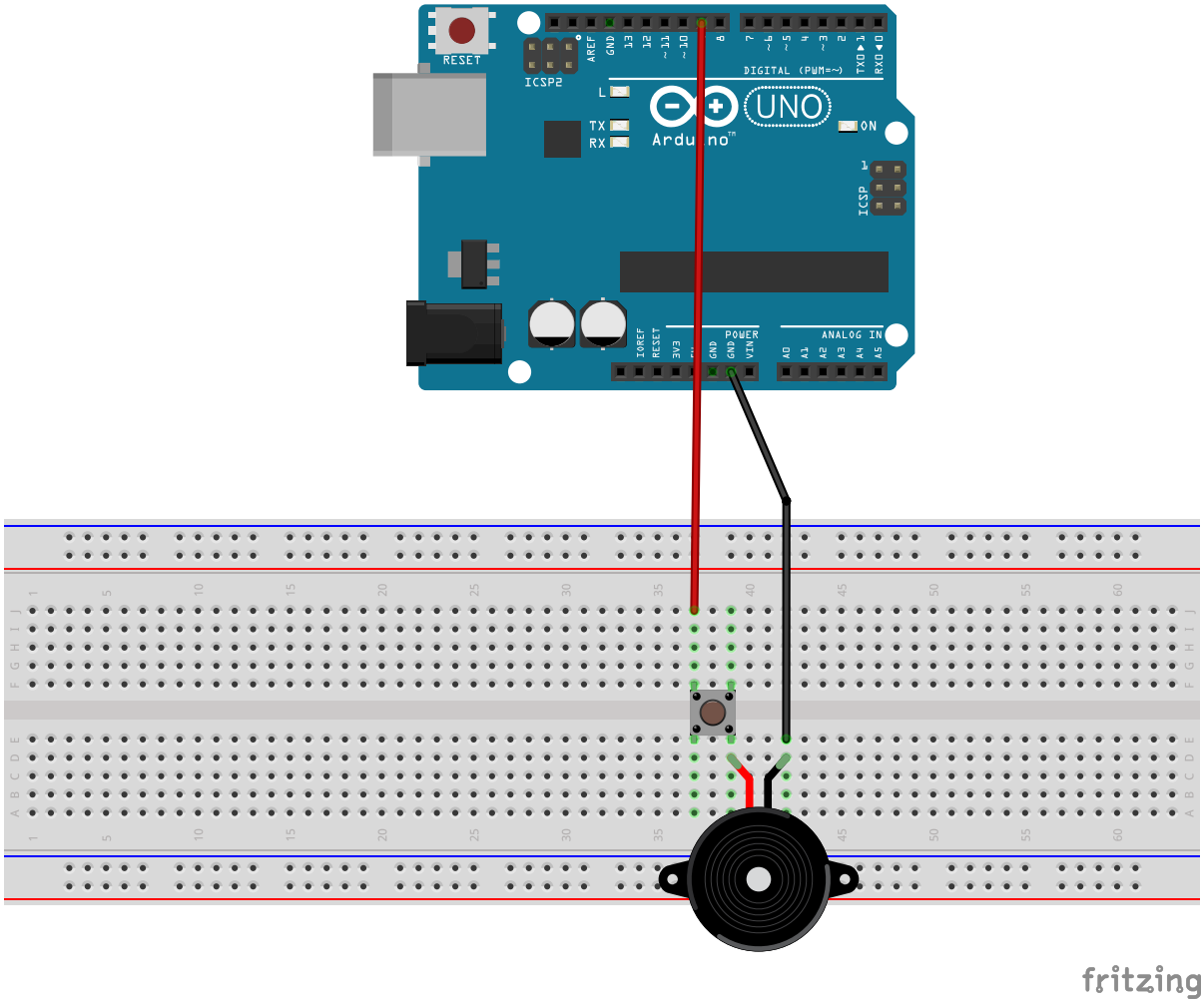


Figure 3.2: Circuit representation for the programmable buzzer project.

Program

```
const uint8_t buzzer = 9;

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

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

3.3 Mini Piano

Parts Required

- 1x Arduino + USB Cable
- 13x Wires
- 1x Breadboard
- 5x Pushbuttons
- 1x Buzzer

Steps

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

2. Place the buzzer so that both legs are in separate rows on the breadboard away from the pushbuttons and so that it is closer to the river so that there is room to place wires next to it. It is recommended to place the legs of the buzzer in the same column so that it is easier to tell where the legs are after placing it on the breadboard. If you do it this way, there will be two empty rows in between the legs.

3. The pushbuttons are set up in what is known as “pull-down” mode. This means that, when pressed, they are grounding whatever they are connected to. To do this for each pushbutton, take a wire and place one end in the same row as one leg of the button. Take the other end and plug it into the negative power rail on the breadboard next to the blue line with the ‘-’ sign.

4. Take another wire and plug one end into the negative power rail and the other end into one of the ports on the Arduino labeled “GND”.

5. For each of the pushbuttons, take a wire and place one end in the same row as the empty leg of the pushbutton. From left to right, place the loose end of the wire into the digital ports on the Arduino labeled “6”, “5”, “4”, “3”, and “2”. These are the ports that will read the pushbutton to determine if we are pressing it.

6. To connect the buzzer, take one wire and place one end in the same row as the short leg of the buzzer and the other end in the negative power rail of the breadboard.

Take a second wire and place one end in the same row as the long leg of the buzzer and the other end in the port labeled “9” on the Arduino Uno. This is the port that will control the frequency, or tone, of the buzzer to make the piano sounds.

7. Now that the circuit is built, go through the code in the Program section, load it into the Arduino IDE, and upload it to your Arduino Uno.

Circuit

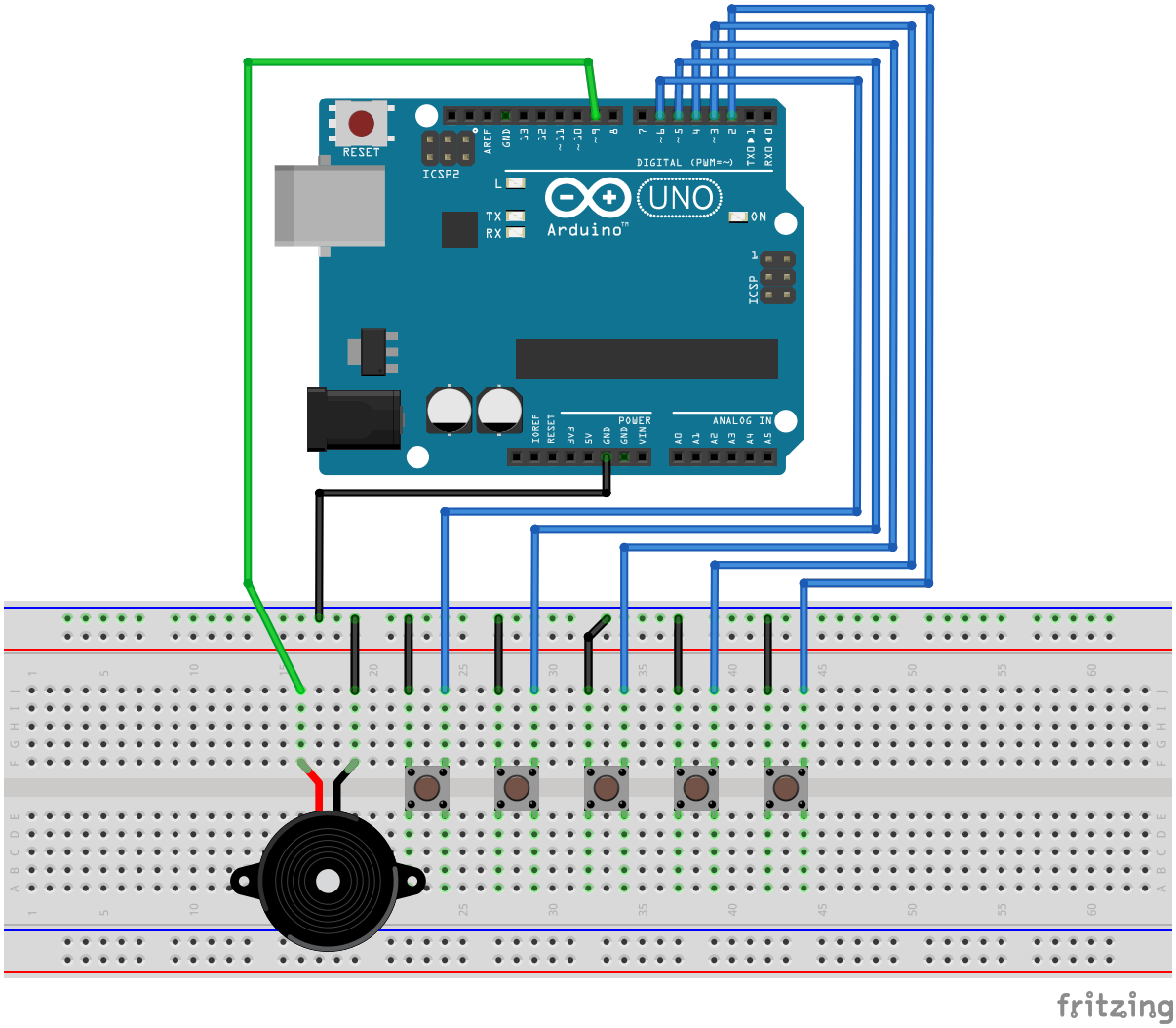


Figure 3.3: Circuit representation for the mini piano project.

Program

```
// Define the frequencies for each
// of the buttons representing
// keys on a piano.
#define FREQ_C 262
#define FREQ_D 294
#define FREQ_E 330
#define FREQ_F 349
#define FREQ_G 392

// Set the input ports connected
// to each of the buttons as constant
// values, meaning they can not be changed.
const int C = 6;
const int D = 5;
const int E = 4;
const int F = 3;
const int G = 2;

// Set the output port connected
// to the buzzer.
const int buzzer = 9;

// Create variables to calculate
// the total frequency of the pressed buttons
// and track the number of buttons pressed.
uint16_t frequency = 0;
uint8_t count = 0;

void setup()
{
  /* Set all of the ports connected
   * to the pushbuttons as inputs.
   *
```

```
34     * Drive all of them high initially
35     * so they are turned off to start.
36    */
37    pinMode( C, INPUT );
38    digitalWrite( C,HIGH );
39
40    pinMode( D, INPUT );
41    digitalWrite( D,HIGH );
42
43    pinMode( E, INPUT );
44    digitalWrite( E,HIGH );
45
46    pinMode( F, INPUT );
47    digitalWrite( F,HIGH );
48
49    pinMode(G, INPUT);
50    digitalWrite(G,HIGH);
51  }
52
53  void loop()
54  {
55      // Reset the frequency sum and
56      // count of buttons pressed.
57      frequency = 0;
58      count = 0;
59
60
61      /* Go through each of the pushbutton
62       * inputs and check if they are pressed.
63       * If pressed, add the frequency of the
64       * selected note and increment the count
65       * of buttons pressed.
66      */
67      if( digitalRead( C ) == LOW )
```

```
68      {
69          frequency += FREQ_C;
70          count++;
71      }
72
73      if( digitalRead( D ) == LOW )
74      {
75          frequency += FREQ_D;
76          count++;
77      }
78
79      if( digitalRead( E ) == LOW )
80      {
81          frequency += FREQ_E;
82          count++;
83      }
84
85      if( digitalRead( F ) == LOW )
86      {
87          frequency += FREQ_F;
88          count++;
89      }
90
91      if( digitalRead( G ) == LOW )
92      {
93          frequency += FREQ_G;
94          count++;
95      }
96
97
98      /* If there are any buttons pressed,
99       * take the average of the frequency
100      * and turn the buzzer on.
101      * Otherwise, turn the buzzer off.
```

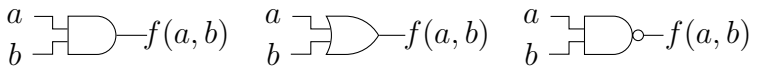
```
102     */
103     if( count )
104     {
105         frequency /= count;
106         tone( buzzer, frequency );
107     }
108     else
109     {
110         noTone( buzzer );
111     }
112 
113 }
```

3.4 Logic Gates

Parts Required

- 1x Arduino + USB Cable
- 2x Wires
- 1x Breadboard
- 2x Slide Switch
- 1x LED
- 1x Resistor
- 1x Quad Logic IC (AND 7408, OR 7432, or NAND 7400)

A two input OR gate produces a true or high output when either one or the other input is true or high. A two input AND gate produces a true or high output only when both inputs are true or high. A two input NAND gate produces the opposite response of an AND gate. By connecting the inputs of a logic gate to a switch between high and low voltage ('5V' and 'GND' on the Arduino) and the output to an LED, we can observe how changing the switch positions will cause the LED to light up or remain off. Each 14-pin AND/OR/NAND logic gate has 4 two input gates inside.



(a) AND Gate (b) OR Gate (c) NAND Gate

Figure 3.4: Logic Gate Symbols

a	b	f(a,b)
0	0	0
0	1	0
1	0	0
1	1	1

(a) AND

a	b	f(a,b)
0	0	0
0	1	1
1	0	1
1	1	1

(b) OR

a	b	f(a,b)
0	0	1
0	1	1
1	0	1
1	1	0

(c) NAND

Table 3.1: Logic Gate Truth Tables

Steps

1. Place the IC chip so that it straddles the river on the breadboard. Note that the indentation on the chip signifies the start of the pin numbers.

2. Place the LED so that the short leg is in the negative power rail (next to the blue line) and the long leg is in a row by itself.

3. Place one of the resistors so that one leg is in the same row as the long leg of the LED and the other leg connects to the output of a logic gate on the IC chip (pin 3 in the diagram).

4. Place both slide switches so that all three pins are in their own row.

5. For each switch, use a resistor to connect the first pin to the power rail (next to the red line). Use a wire to connect the last pin to the negative power rail (next to the blue line). The middle pin will connect to the logic gate inputs (pins 1 and 2 for the logic gate in the diagram).

6. To supply the IC chip with power, take one wire and connect the power rail to the Vcc input on the chip (pin 14 in the diagram). Take another wire and connect the negative power rail to the GND input on the IC chip (pin 7 in the diagram).

7. To power the circuit, take one wire and plug it into the “5V” port on the Arduino. Place the other end of the wire into the power rail on the breadboard (next to the red line).

8. To complete the circuit, take the second wire and plug it into the “GND” port on the Arduino. Place the other end in the negative power rail on the breadboard (next to the blue line).

9. Plug the Arduino into your computer’s USB port to turn power on. Adjust the slide switches to turn each logic gate input ‘ON’ and ‘OFF’ and observe when the LED remains off or lights up.

10. Replace the IC chip with a different logic gate chip and observe the new response.

Circuit

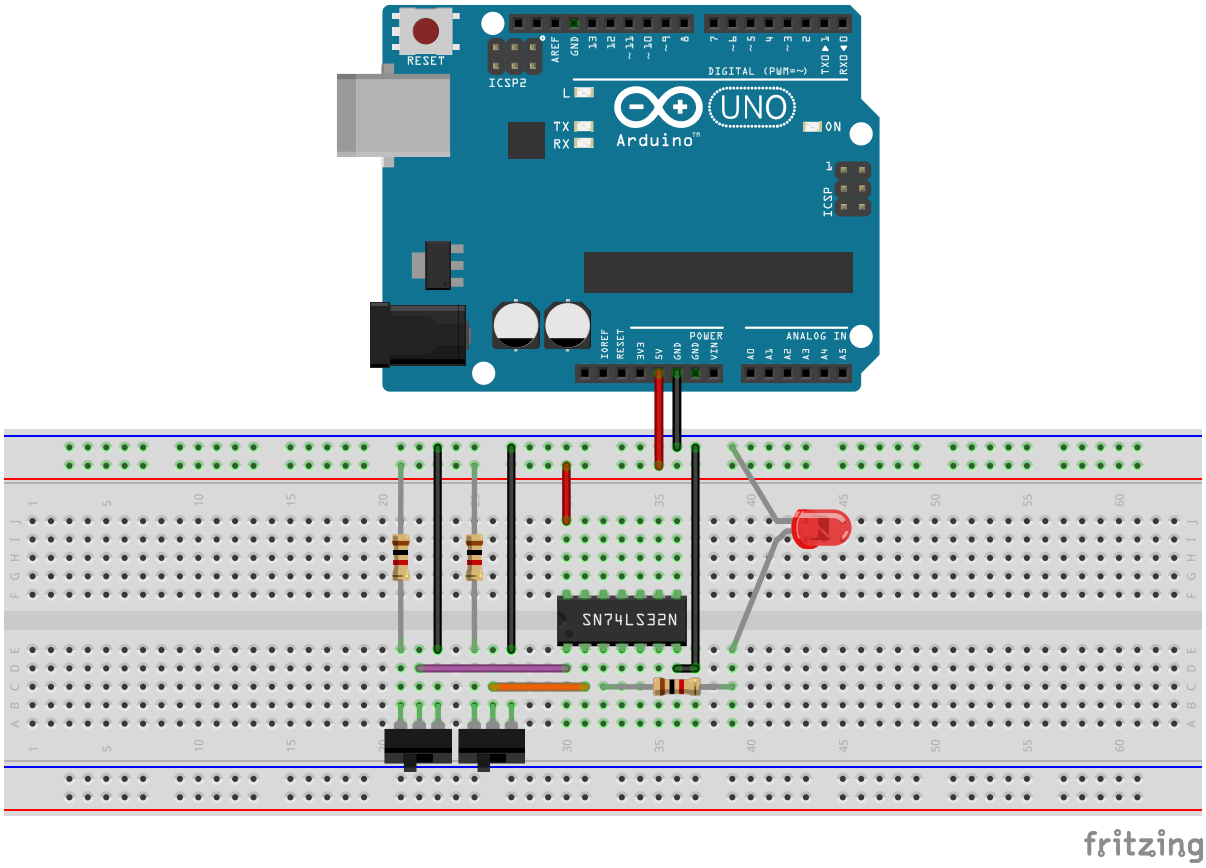


Figure 3.5: Circuit representation for the logic gate project.

4 Level 3 Projects

4.1 Ultrasonic Security System

Parts Required

- 1x Arduino + USB Cable
- 8x Wires
- 1x Breadboard
- 1x Ultrasonic Sensor HC-SR04
- 1x Buzzer
- 1x LED
- 1x 330-Ω 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 port “13” on the Arduino. Plug the other end into a hold on the breadboard in the same row as the port labeled “Trig” on the ultrasonic sensor.

6. Take a second wire and plug one end into port “12” on the Arduino. Plug the other end into a hold on the breadboard in the same row as the port labeled “Echo” on the ultrasonic sensor.
7. To control the LED, we take the third wire and plug one end into port “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 port “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” port 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” port 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

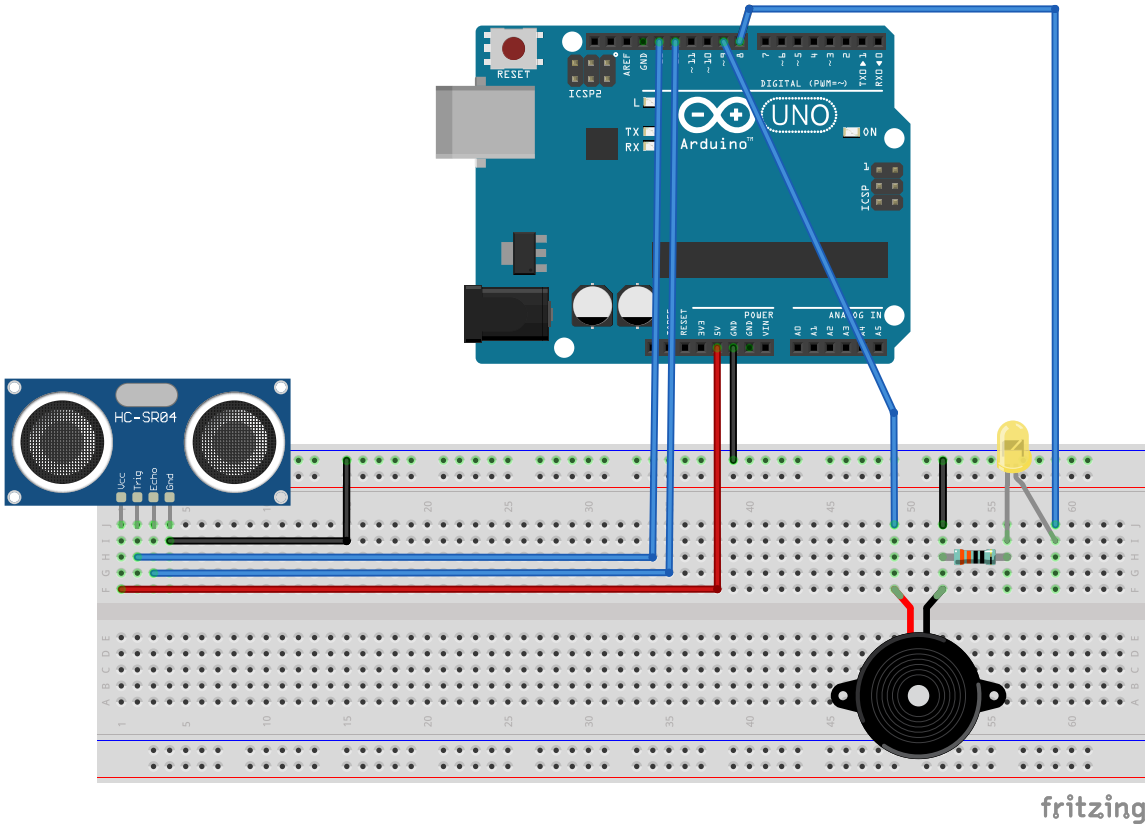


Figure 4.1: Circuit representation for the ultrasonic security project.

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 )
```

```
34   {
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 );
47   }
48 }
```

5 Level 4 Projects

5.1 TFT Etch-a-Sketch

Parts Required

- 1x Arduino + USB Cable
- 18x Wires
- 1x Breadboard
- 1x 1.8" TFT Display
- 1x Pushbutton
- 2x Potentiometer
- 1x 1k-Ω 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 breadboard 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 Port
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 "5V" 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

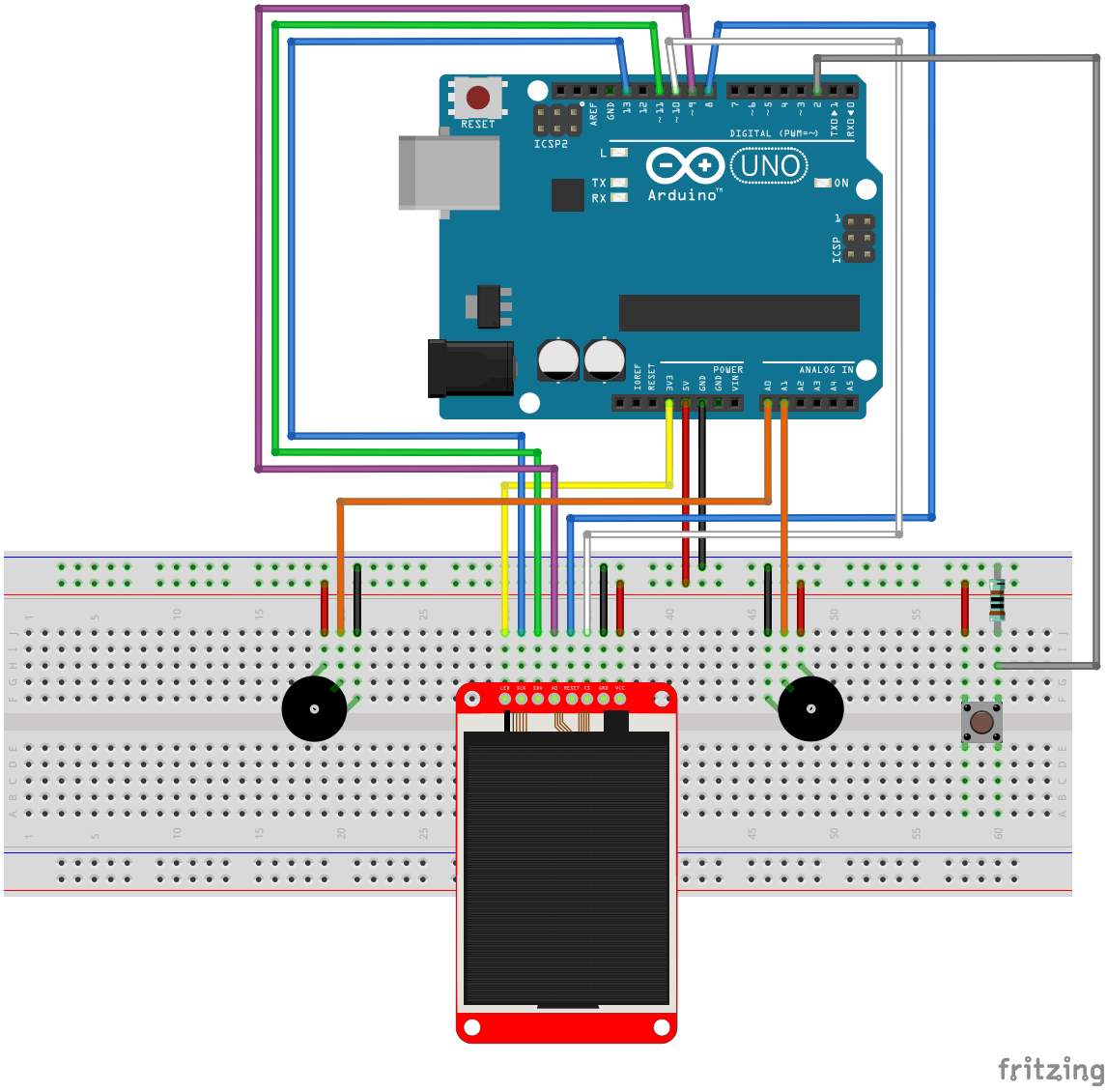


Figure 5.1: Circuit representation for the TFT Etch-a-Sketch project.

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 );
```

```
68    currPnt.y = map( yValue, 0, 1023, 0, 127 );
69
70
71    // Draw the last point location white,
72    TFTscreen.stroke( 255, 255, 255 );
73    TFTscreen.point( lastPnt.x, lastPnt.y );
74
75
76    // Draw the current point location blue.
77    TFTscreen.stroke( 255, 0, 0 );
78    TFTscreen.point( currPnt.x, currPnt.y );
79
80
81    // Store the current location as the last location
82    // for the next iteration.
83    lastPnt.x = currPnt.x;
84    lastPnt.y = currPnt.y;
85
86
87    // Read the value of the pushbutton, and erase the screen if pressed
88    if( digitalRead( erasePin ) == HIGH )
89    {
90      TFTscreen.background(0, 0, 0);
91    }
92
93    delay( 33 );
94  }
```

5.2 Pong

In this project we will create an Arduino based version of the classic video game Pong. The controller inputs are built using a variable resistor or potentiometer and the on board analog to digital converter of the Arduino. The court is rendered on the TFT display with the TFT and SPI libraries.

Parts Required

- 1x Arduino + USB Cable
- 16x Wires
- 1x Breadboard
- 1x 1.8" TFT Display
- 2x Potentiometer

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 legs are in separate, empty rows on the breadboard to one side of the TFT display.
    3. Place the second potentiometer so that all three legs are in separate, empty rows on the breadboard to the other side of the TFT display.
    4. 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 Port
LED	3.3 V
SCK	13
SDA	11
A0	9
RESET	8
CS	10

Table 5.2: 1.8" TFT wiring connections to the Arduino Uno.

5. 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 leg 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 leg on the potentiometer that you used for the first wire.
   - Take the third wire and plug one end into port "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 leg on the potentiometer. Note that the potentiometer plugged into "A0" will control the movement of the first players paddle and the potentiometer plugged into "A1" will control the movement of the 2nd players paddle.
6. Once everything is connected, we want to power our circuit by taking another wire and plugging one end into the "5V" port 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" ports and plug the other end into the negative power rail on the breadboard.
7. Now that the circuit is built, go to your computer, open the Arduino IDE, and proceed to the next section to write the Pong program and upload it to your Arduino Uno.

Circuit

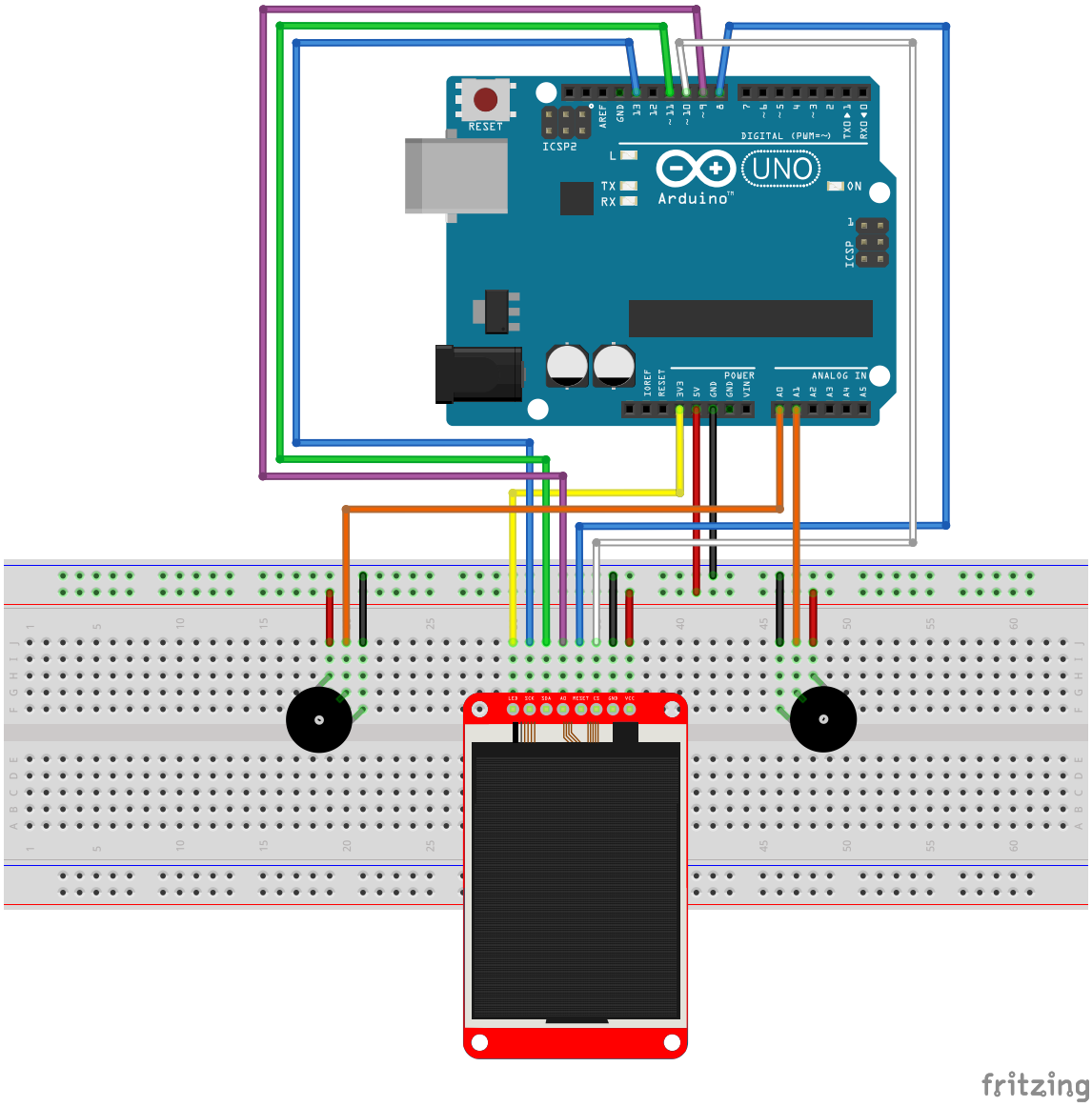


Figure 5.2: Circuit representation for the Pong project.

Script

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


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


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

// Create constants to remember the
// size of the TFT screen.
const uint8_t TFT_WIDTH  = TFTscreen.height();
const uint8_t TFT_HEIGHT = TFTscreen.width();


// Create constants to set the size
// of the bar and the pong ball.
#define BAR_LENGTH 40
#define BAR_WIDTH 5
#define BAR_OFFSET 10


// Define a new object type to
// represent a player.
typedef struct
{
  uint8_t barPosition;
```

```
34   uint8_t points;
35 } Player;
36
37
38 // Define constant radius
39 // and velocity of the ball.
40 #define BALL_RADIUS 3
41 #define BALL_VELOCITY 2
42
43 // Define a new object type to represent
44 // the pong ball.
45 typedef struct
46 {
47   uint8_t x;
48   uint8_t y;
49
50   uint8_t directionX;
51   uint8_t directionY;
52 } Ball;
53
54
55 // Create objects to track the two players
56 // and the pong ball.
57 Player playerLeft;
58 Player playerRight;
59 Ball ball;
60 unsigned long lastBallDraw;
61 uint8_t winner;
62
63
64 void setup() {
65   // Set the bars vertically
66   // in the middle.
67   playerLeft.barPosition = ( TFT_HEIGHT / 2 ) - ( BAR_LENGTH / 2 );
```

```
68    playerRight.barPosition = ( TFT_HEIGHT / 2 ) - ( BAR_LENGTH / 2 );
69
70    // Set scores to 0.
71    playerLeft.points = 0;
72    playerRight.points = 0;
73
74
75    // Set the pong ball in the center
76    // of the playing field.
77    ball.x = ( TFT_WIDTH / 2 );
78    ball.y = ( TFT_HEIGHT / 2 );
79    ball.directionX = 1;
80    ball.directionY = 0;
81
82
83    // Initialize the screen
84    TFTscreen.begin();
85    TFTscreen.setRotation( 2 );
86
87    // Make the background black
88    // using the RGB color (0, 0, 0).
89    TFTscreen.background(0, 0, 0);
90
91
92    drawBars( playerLeft.barPosition, playerRight.barPosition );
93    drawBall();
94    lastBallDraw = millis();
95  }
96
97  void loop() {
98    // Read the potentiometers on A0 and A1
99    uint8_t rightPos = map( analogRead( A1 ), 0, 1023, 0, TFT_HEIGHT - 1 -
       BAR_LENGTH );
```

```
100   uint8_t leftPos = map( analogRead( A0 ), 0, 1023, 0, TFT_HEIGHT - 1 -
       BAR_LENGTH );
101
102   drawBars( rightPos, leftPos );
103   drawScore();
104
105   if( millis() - lastBallDraw > 10 )
106   {
107     if( winner = drawBall() )
108     {
109       if( winner == 1 )
110         playerLeft.points++;
111       else
112         playerRight.points++;
113
114
115       ball.x = TFT_WIDTH / 2;
116       ball.y = TFT_HEIGHT / 2;
117       TFTscreen.background( 0, 0, 0 );
118
119       drawScore();
120
121       delay( 500 );
122
123       playerLeft.barPosition = -10;
124       playerRight.barPosition = -10;
125       drawBars( rightPos, leftPos );
126       drawBall();
127
128       delay( 500 );
129     }
130     lastBallDraw = millis();
131   }
132 }
```

```
133
134 void drawBars( uint8_t rightPos, uint8_t leftPos ) {
135   // If the new position has not moved significantly
136   // from the previous position, don't write it again
137   // to prevent any stuttering on the screen.
138   if( abs( leftPos - playerLeft.barPosition ) > 3 )
139   {
140     // Clear the current bars.
141     TFTscreen.fill( 0, 0, 0 );
142     TFTscreen.rect( 0 + BAR_OFFSET, playerLeft.barPosition,
143                     BAR_WIDTH, BAR_LENGTH );
144
145
146     // Calculate the new position of the
147     // left player's barand then draw it.
148     uint8_t startX = 0 + BAR_OFFSET;
149     uint8_t startY = leftPos;
150
151     TFTscreen.fill( 255, 255, 255 );
152     TFTscreen.rect( startX, startY, BAR_WIDTH, BAR_LENGTH );
153
154
155     // Set the new bar position for the left player.
156     playerLeft.barPosition = leftPos;
157   }
158
159
160   if( abs( rightPos - playerRight.barPosition ) > 3 )
161   {
162     // Clear the current bars.
163     TFTscreen.fill( 0, 0, 0 );
164     TFTscreen.rect( TFT_WIDTH - BAR_OFFSET, playerRight.barPosition,
165                     BAR_WIDTH, BAR_LENGTH );
166
```

```
167
168     // Calculate the new position of the
169     // left player's barand then draw it.
170     uint8_t startX = TFT_WIDTH - BAR_OFFSET;
171     uint8_t startY = rightPos;
172
173     TFTscreen.fill( 255, 255, 255 );
174     TFTscreen.rect( startX, startY, BAR_WIDTH, BAR_LENGTH );
175
176
177     // Set the new bar position for the right player.
178     playerRight.barPosition = rightPos;
179   }
180
181   return;
182 }
183
184 uint8_t drawBall() {
185   // Calculate new location.
186   int16_t newX = ball.x + ( ball.directionX ? BALL_VELOCITY : -BALL_VELOCITY
        );
187   int16_t newY = ball.y + ( ball.directionY ? BALL_VELOCITY : -BALL_VELOCITY
        );
188
189   // Check if the ball is hitting the
190   // right or leftborder of the screen.
191   if( newX >= TFT_WIDTH - BALL_RADIUS )
192   {
193     ball.directionX = !ball.directionX;
194     newX = TFT_WIDTH / 2;
195     newY = TFT_HEIGHT / 2;
196     return 1;
197   }
198   else if( newX < 0 + BALL_RADIUS )
```

```
199   {
200     ball.directionX = !ball.directionX;
201     newX = TFT_WIDTH / 2;
202     newY = TFT_HEIGHT / 2;
203     return 2;
204   }
205
206   // Check if the ball is hitting the
207   // top or bottom of the screen.
208   if( newY >= TFT_HEIGHT - BALL_RADIUS )
209   {
210     ball.directionY = !ball.directionY;
211     newY = TFT_HEIGHT - BALL_RADIUS;
212   }
213   else if( newY < 0 + BALL_RADIUS )
214   {
215     ball.directionY = !ball.directionY;
216     newY = 0 + BALL_RADIUS;
217   }
218
219   // Check if the ball is within the
220   // left or right player's bar.
221   if( ( newX - BALL_RADIUS >= BAR_OFFSET && newX - BALL_RADIUS <= BAR_OFFSET
        + BAR_WIDTH ) &&
222       ( newY + BALL_RADIUS >= playerLeft.barPosition && newY + BALL_RADIUS
       <= playerLeft.barPosition + BAR_LENGTH ) )
223   {
224     ball.directionX = !ball.directionX;
225   }
226   else if( ( newX + BALL_RADIUS >= TFT_WIDTH - BAR_OFFSET - BAR_WIDTH &&
       newX + BALL_RADIUS <= TFT_WIDTH - BAR_OFFSET ) &&
227           ( newY - BALL_RADIUS >= playerRight.barPosition && newY -
       BALL_RADIUS <= playerRight.barPosition + BAR_LENGTH ) )
228   {
```

```
229     ball.directionX = !ball.directionX;
230   }
231
232   // Clear previous location.
233   TFTscreen.fill( 0, 0, 0 );
234   TFTscreen.circle( ball.x, ball.y, BALL_RADIUS );
235
236   // Update the ball's location.
237   ball.x = newX;
238   ball.y = newY;
239
240   // Display new location.
241   TFTscreen.fill( 255, 255, 255 );
242   TFTscreen.circle( ball.x, ball.y, BALL_RADIUS );
243
244   return 0;
245 }
246
247 void drawScore() {
248   char score[ 2 ];
249   TFTscreen.stroke( 255, 255, 255 );
250
251   sprintf( score, "%02d", playerLeft.points );
252   TFTscreen.text( score, BAR_OFFSET + BAR_WIDTH, 0 );
253
254   sprintf( score, "%02d", playerRight.points );
255   TFTscreen.text( score, TFT_WIDTH - BAR_OFFSET - BAR_WIDTH, 0 );
256
257   TFTscreen.stroke( 0, 0, 0 );
258 }
```