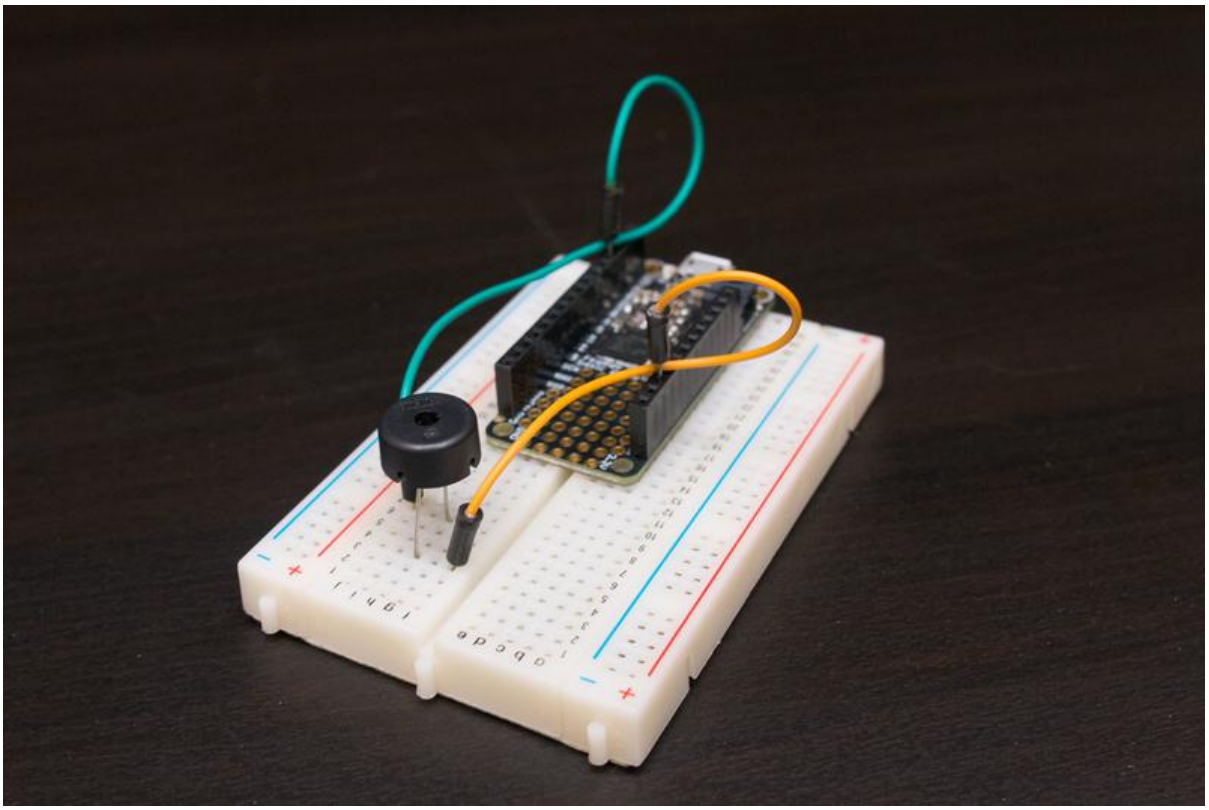
Using Piezo Buzzers with CircuitPython & Arduino

Created by Tony DiCola

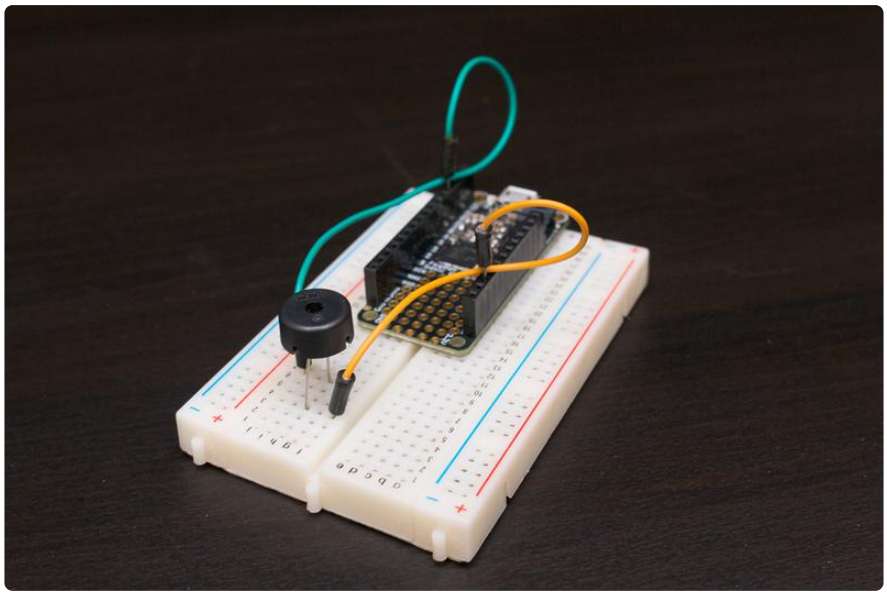


https://learn.adafruit.com/using-piezo-buzzers-with-circuitpython-arduino

Last updated on 2021-11-15 07:05:45 PM EST

Table of Contents

Overview



Piezo buzzers are simple devices that can generate basic beeps and tones. They work by using a piezo crystal, a special material that changes shape when voltage is applied to it. If the crystal pushes against a diaphragm, like a tiny speaker cone, it can generate a pressure wave which the human ear picks up as sound. Simple change the frequency of the voltage sent to the piezo and it will start generating sounds by changing shape very quickly!

This guide will explore how to generate tones with a piezo from CircuitPython or Arduino code.

With CircuitPython, you can use simple Python code to play beeps and music notes with the piezo. You can even use the CircuitPython REPL to make sound interactively!

In addition this guide will also show basic piezo control with Arduino code too, so you can use it for either!

Hardware

To follow this guide you'll need the following parts:

- A piezo buzzer. Piezo buzzers are like tiny little speakers, but unlike speakers they don't require an amplifier or other complex circuitry to drive them (consequently they aren't nearly as loud as a speaker and amplifier either!). When voltage is applied to the piezo it grows and shrinks in size, and by

changing the voltage over time you can make the piezo change shape fast enough to create a pressure wave that your ears interpret as sound.

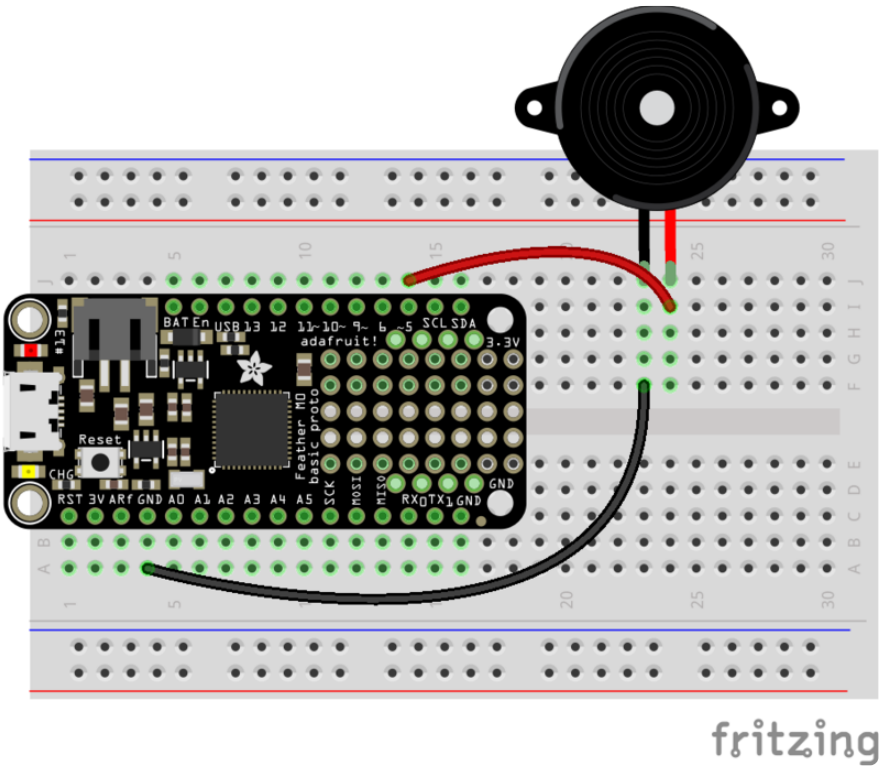
- A board running CircuitPython or Arduino. If you're controlling piezos from CircuitPython you'll need a board like the Feather M0 basic (https://adafru.it/s1d) which can be loaded with CircuitPython firmware. This guide will also show basic piezo code for Arduino too.
- Breadboard (https://adafru.it/kft) and jumper wires (https://adafru.it/fE2). You'll need these parts to connect components to your development board.

Wiring

To connect a piezo buzzer you just need to connect one leg of the buzzer to your board ground, and another leg to a PWM-capable or analog-out output of your board. For the Feather M0 and other M0 boards look for a squiggly line next to a pin to denote that it supports PWM output. For other boards check the guide or documentation to see which pins support PWM output.

Arduino uses an interrupt system for piezos, so you can use any pin.

For example here's how to wire a piezo to a Feather M0:



- One leg of the piezo buzzer (or black wire) to board GND.

- The other leg of the piezo buzzer (or red wire) to board D5 (or any other PWM-capable output).

CircuitPython

To control the piezo from CircuitPython we'll use its built in PWM, or pulse-width modulation, signal generation capabilities. Be sure to read the CircuitPython analog I/O guide (https://adafru.it/BCI) for more details on PWM signals!

Before continuing make sure your hardware is wired up as shown on the previous page.

Next make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board, then connect to the board's serial REPL (https://adafru.it/Awz)so you are at the CircuitPython >>> prompt.

Setup

First you need to import a few modules to access the PWM output capabilities of CircuitPython:

```
import board
import pulseio
```

Now you can create a PWM signal output that will drive the buzzer to make sound:

```
buzzer = pulseio.PWMOut(board.D5, variable_frequency=True)
```

There are a couple important things happening with the line above. This is an initializer which is creating an instance of the PWMOut class and part of that initialization process is specifying these two values:

- The pin that will be the PWM output. In this case it's pin D5 on the development board. If you're using a different output be sure to specify the right pin name here (and make sure the pin supports PWM output as mentioned on the previous page!).
- The variable_frequency boolean is True. This is an optional value that we're specifying as a keyword argument to tell the PWM output that we want to be able to change its frequency, or how often the signal changes. By default PWM

outputs in CircuitPython have a fixed frequency, but if this special keyword is specified you can instead change the frequency of the output.

Making Tones with PWM

Now we can generate tones using the PWM output. Remember from the CircuitPython analog I/O guide PWM page (https://adafru.it/BCJ) a PWM signal is just a fast on/off signal, i.e. a square wave. When a square wave modulates the movement of something, like a buzzer, it generates a pressure wave that the human ear interprets as sound. By changing the frequency, or how often the square wave changes from high to low, you can change the frequency of the tone that you'll hear.

Changing the frequency of the PWM output is easy, just modify the frequency attribute to set a new value in hertz. For example a 440 hz wave is the same pitch as an A4 note in music:

```
buzzer.frequency = 440
```

You might notice after setting the frequency above nothing actually happened--no sound is being made by the buzzer. Is the buzzer broken? Nope! There's one more thing you need to do to create the square wave signal, you need to set the duty cycle of the PWM output.

Like the CircuitPython analog I/O PWM page mentions (https://adafru.it/BCJ) the duty cycle of a signal is the percent of time that it's high vs. low. We want to generate a square wave which is high and low for exactly the same amount of time (i.e. 50% duty cycle). This will generate a square wave of the previously specified frequency on the PWM output, and as a result induce a pressure wave from the buzzer that you'll hear as sound.

To simplify turning on and off the buzzer let's make a couple variables to represent the 0% and 50% duty cycle values that turn the buzzer off and on respectively:

```
OFF = 0
ON = 2**15
```

The ON value is 2^15, or 32768, which is about half of the maximum duty cycle value (65535).

Now set the buzzer duty cycle to ON to start playing the tone at the previously set frequency:

```
buzzer.duty_cycle = ON
```

You should hear the buzzer start to play a tone at 440 hz!

To stop the tone playback change the duty cycle to OFF or 0%:

```
buzzer.duty_cycle = OFF
```

Now the buzzer stops making sound!

You can change the frequency at any time too, whether the buzzer is playing sound or not. Here are some interesting frequency values to try (they're the frequencies for musical notes):

- 262 hz, C4
- 294 hz, D4
- 330 hz, E4
- 349 hz, F4
- 392 hz, G4
- 440 hz, A4
- 494 hz B4

Try turning the buzzer on and changing the frequency between a few note values:

```
buzzer.duty_cycle = ON
buzzer.frequency = 262 # C4
buzzer.frequency = 294 # D4
buzzer.frequency = 330 # E4
buzzer.duty_cycle = OFF
```

That's all there is to simple tone playback with a piezo buzzer and CircuitPython's built-in PWM output!

Here's a complete example that will play a scale of tones up and down repeatedly. Save this as main.py on your board and be ready after saving it as the music playback will immediately start:

```
import time

import board
```

```
import pulseio


# Define a list of tones/music notes to play.
TONE_FREQ = [ 262,  # C4
              294,  # D4
              330,  # E4
              349,  # F4
              392,  # G4
              440,  # A4
              494 ] # B4

# Create piezo buzzer PWM output.
buzzer = pulseio.PWMOut(board.D5, variable_frequency=True)

# Start at the first note and start making sound.
buzzer.frequency = TONE_FREQ[0]
buzzer.duty_cycle = 2**15  # 32768 value is 50% duty cycle, a square wave.

# Main loop will go through each tone in order up and down.
while True:
    # Play tones going from start to end of list.
    for i in range(len(TONE_FREQ)):
        buzzer.frequency = TONE_FREQ[i]
        time.sleep(0.5)  # Half second delay.
    # Then play tones going from end to start of list.
    for i in range(len(TONE_FREQ)-1, -1, -1):
        buzzer.frequency = TONE_FREQ[i]
        time.sleep(0.5)
```

Making Tones With SimpleIO

Another option to generate a tone is with the SimpleIO module (https://adafru.it/Cic) for CircuitPython. This module simplifies tone generation by doing all the PWM duty cycle and frequency logic for you automatically. With a simple tone function call you can play a tone for a period of time--no setup or duty cycle calculation necessary! You might find this approach easier to start with for basic tone playback compared to the PWM approach shown above.

To follow this approach you'll need to install the Adafruit CircuitPython SimpleIO (https://adafru.it/Cic) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). Our introduction guide has a great page on how to install the library bundle (https://adafru.it/ABU) for both express and non-express boards.

Remember for non-express boards like the, you'll need to manually install the necessary libraries from the bundle:

- simpleio.mpy

You can also download the simpleio.mpy from its releases page on Github (https://adafru.it/yrF).

Before continuing make sure your board's lib folder or root filesystem has the simplei o.mpy file copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz)so you are at the CircuitPython >>> prompt. Then import the board and simpleio modules:

```
import board
import simpleio
```

Now you'e ready to use the tone function in SimpleIO to play a tone on a pin connected to a piezo buzzer. Try the following to play a 440 hz tone for 1 second:

```
    simpleio.tone(board.D5, 440, duration=1.0)
```

You should hear a 440 hz tone, or an A4 note, played for one second. The parameters you pass to the tone function control the frequency and duration of the tone:

- The first parameter is the pin connected to the buzzer or speaker, in this case digital pin 5 as shown in the wiring for this guide.
- The second parameter is the frequency in Hz of the tone to play back. This can be a floating point but note that at higher frequencies (> 10KHz) the precision may not match perfectly
- The last parameter is a keyword duration which specifies how long to play the tone in seconds. By default if you don't provide a value here the tone will be played for one second, however you can specify shorter or longer values.

Another example is playing a 400 hz tone for 3 seconds:

```
simpleio.tone(board.D5, 400, duration=3.0)
```

Notice how the frequency and duration parameters changed to specify these different values!

That's all there is to basic tone playback on a piezo with the SimpleIO module! Here's another complete example that plays a scale of notes using SimpleIO's tone function. Again save this as a main.py on your board to have it run:

```
import board

import simpleio


# Define pin connected to piezo buzzer.
PIEZO_PIN = board.D5

# Define a list of tones/music notes to play.
TONE_FREQ = [ 262,  # C4
              294,  # D4
              330,  # E4
              349,  # F4
              392,  # G4
              440,  # A4
              494 ] # B4


# Main loop will go through each tone in order up and down.
while True:
    # Play tones going from start to end of list.
    for i in range(len(TONE_FREQ)):
        simpleio.tone(PIEZO_PIN, TONE_FREQ[i], duration=0.5)
    # Then play tones going from end to start of list.
    for i in range(len(TONE_FREQ)-1, -1, -1):
        simpleio.tone(PIEZO_PIN, TONE_FREQ[i], duration=0.5)
```

Arduino

You can also control a piezo buzzer from Arduino in a similar way as CircuitPython with Arduino's tone function (https://adafru.it/Cie). There are actually quite a few resources and guides for using Arduino to play tones on a piezo, so this page will just highlight how to use a servo in Arduino in the same way as with CircuitPython from the previous page. For more exploration of piezo and Arduino be sure to check out the entire Arduino Lesson 10: Making Sounds (https://adafru.it/tUD).

First make sure your piezo is wired to your board exactly as shown on the hardware page of this guide. You'll also need the Arduino IDE installed and configured to upload to your board. Remember Arduino sketches have to be written entirely up front and uploaded to the board--you can't interactively control servos like you can with CircuitPython.

With Arduino you use the tone (https://adafru.it/Cie) function to play a 50% duty cycle (i.e. square wave) on any digital output. The function needs to be told the pin and frequency of the tone as one of the parameters and will immediately start playback of the tone/signal. To stop playback you just call the noTone (https://adafru.it/Cig) function (providing only the pin).

Here's a complete Arduino sketch that uses the tone function to play a scale of notes just like the CircuitPython example on the previous page. Upload this to your board:

```
#define PIEZO_PIN  5      // Pin connected to the piezo buzzer.

// Define list of tone frequencies to play.
int toneFreq[] = { 262,   // C4
                   294,   // D4
                   330,   // E4
                   349,   // F4
                   392,   // G4
                   440,   // A4
                   494 }; // B4
int toneCount = sizeof(toneFreq)/sizeof(int);

void setup() {
  // No setup necessary!
}

void loop() {
  // Loop up through all the tones from start to finish.
  for (int i=0; i &lt; toneCount; ++i) {
    tone(PIEZO_PIN, toneFreq[i]);
    delay(500);  // Pause for half a second.
  }
  // Loop down through all the tones from finish to start again.
  for (int i=toneCount-1; i &gt;= 0; --i) {
    tone(PIEZO_PIN, toneFreq[i]);
    delay(500);
  }
  // Remember you can stop tone playback with the noTone function:
  // noTone(PIEZEO_PIN);
}
```

After uploading you should hear the piezo play back each tone in sequence up and down. That's all there is to using a piezo buzzer with Arduino to play tones!