Interfacing the Raspberry Pi 4 with Buzzers Using Python

Buzzers are both very simple and very useful components for building circuits. They can provide auditory feedback in circuits that may not always be visible, play music and alert users of events.

The buzzers we will be focusing on are piezoelectric buzzers, called piezo buzzers for short. They are simple electronic components with two terminals: voltage and ground. This makes them very simple to integrate with a micro-controller/digital device (in our case, the Raspberry Pi 4).

Table of Contents

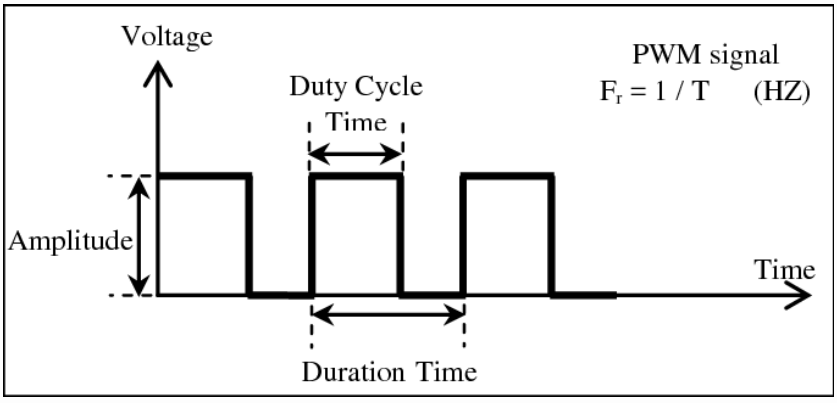
Types of Buzzers

There are two main types of piezo buzzers: passive and active. One thing to keep in mind is that both buzzers will exhibit a louder volume when connected to *higher voltage*. Make sure, however, that you keep the supply voltage within the limit that your buzzer is rated for.

Passive

Passive buzzers require a changing signal in order to produce noise. They are called "passive" because they don't do anything on their own with just a DC (direct current) signal.

The type of signal usually provided to a buzzer is a rapidly pulsing signal, called a PWM or pulse-width modulated signal. The signal itself looks like the image below:



A sample PWM signal and its characteristics. [1]

With a passive buzzer, it's possible to control the frequency or tone of the produced sound by changing the frequency of the PWM signal. Thankfully, Python will abstract away those details for us and provide a simple interface for selecting tones.

Active

Active buzzers are buzzers that produce noise with just a constant DC signal. You can connect an active buzzer directly to a battery and it will produce noise. You can also control this buzzer with a PWM signal, but a DC signal is all you need.

What Kind of Buzzer do I Have

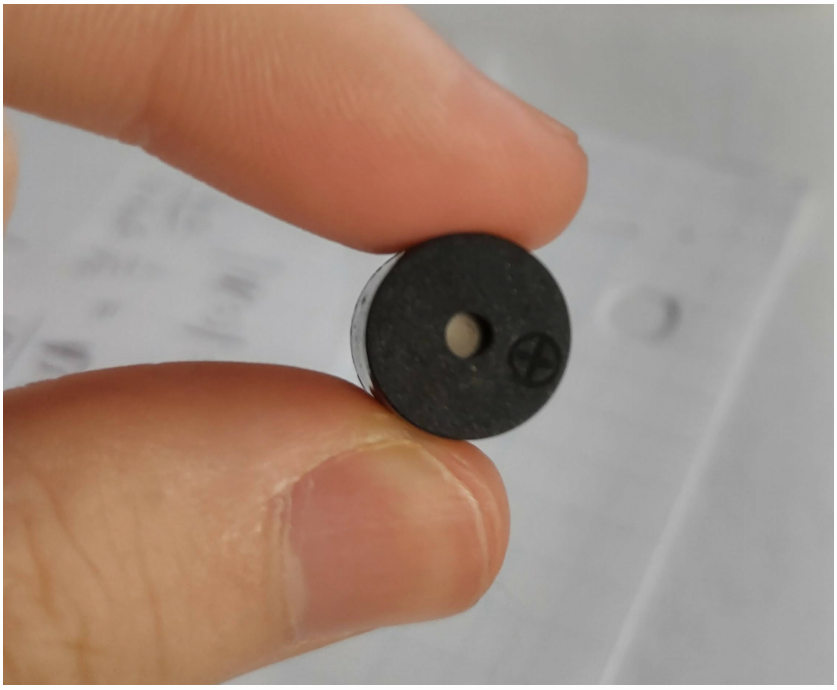
In order to control your buzzer with the Raspberry Pi, you should first determine which type of buzzer you have. If you aren't sure which type of buzzer you've purchased, you can perform a simple test.

Identifying Positive and Negative Terminals

The indication of positive and negative terminals of piezo buzzers is manufacturer dependent. The most common types are as follows:

Positive Marker in Shield

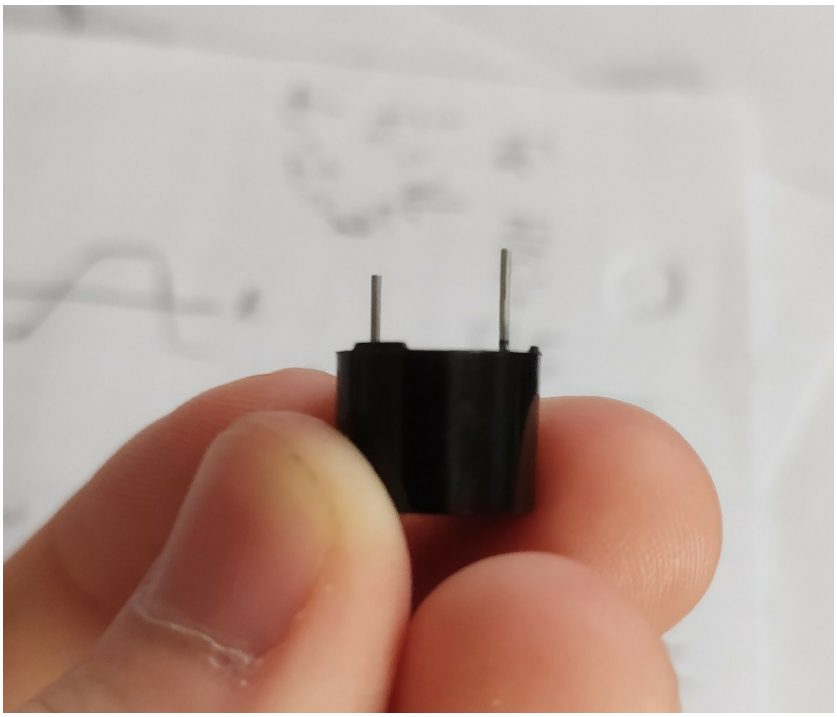
Some piezo buzzers have a small marking on their shield with a "+" sign, indicating which terminal is the positive terminal. The symbol is directly over the positive lead.



A piezo buzzer with a marking on its casing indicating the positive terminal.

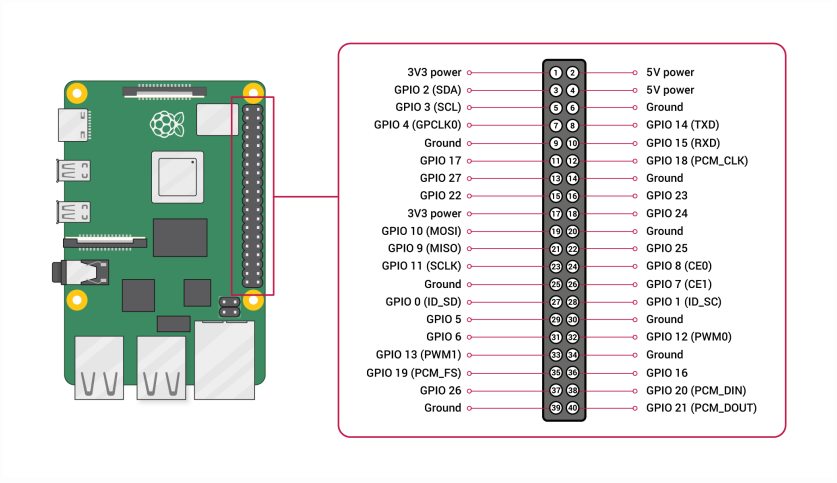
Different Terminal Lengths

Similar to LEDs, some piezo buzzers will use different lead lengths to indicate the polarity of the terminals. As is convention, the **shorter lead is the negative terminal** (ground), and the **longer lead is the positive terminal** (supply voltage).



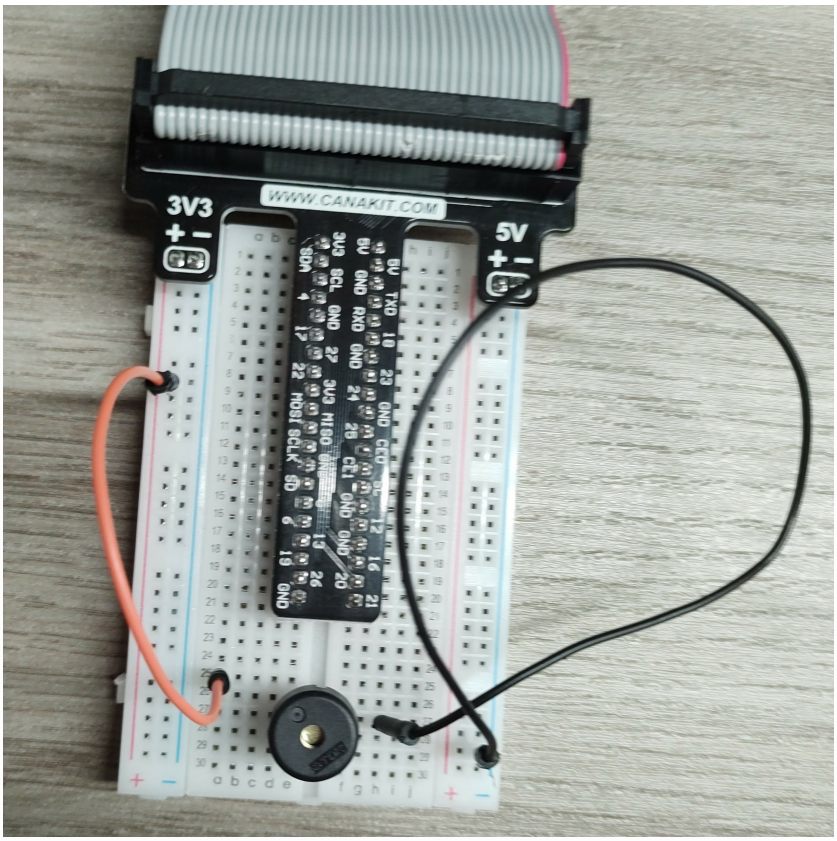
A piezo buzzer with differing lead lengths for determining polarity.

Test Circuit



The Raspberry Pi 4 pinout diagram. [4]

Using the ground connection on the Raspberry Pi, connect the negative terminal of the buzzer to ground using a wire onto your breadboard. Then, using one of the Raspberry Pi's 3V output pins (see the pinout diagram), connect the positive terminal of the buzzer to 3V. If the buzzer produces noise, it is an active buzzer. If not, it is a passive buzzer and requires a changing signal.

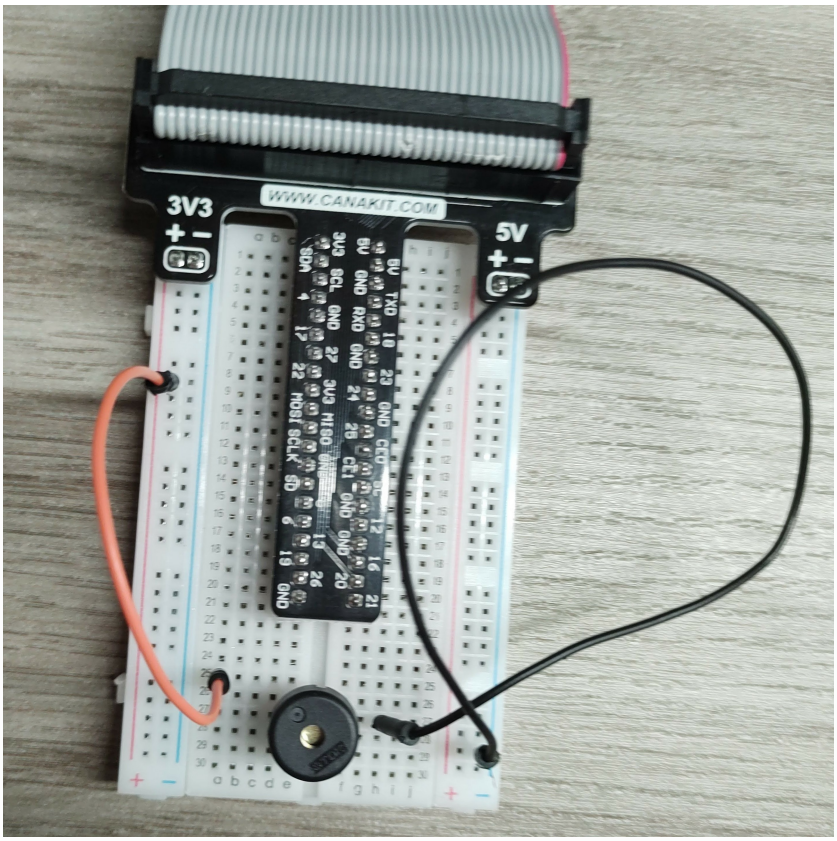


The test circuit with the buzzer connected to 3V.

Interfacing with the Pi

Now that you've determined which type of piezo buzzer you have and you've tried hooking it up to a circuit, interfacing with a Pi isn't much more difficult!

You will want to modify the test circuit so that the buzzer is connected to a GPIO pin instead of the 3V pin. Pin 22 has been chosen for this demo.



The circuit with the buzzer connected to GPIO 22.

In your Raspberry Pi's console window/terminal, open the directory where you want to write your test code. Use the following commands to create a virtual environment with the required dependencies for development:

```
python3 -m venv venv
source venv/bin/activate
pip install gpiozero
```

Once you've finished writing your script (outlined in the next sections), you'll be able to execute it using the following command.

```
python3 <your_script_name.py>
```

Active Buzzer Controller

The active buzzer controller is very akin to a controller for toggling an LED, as our active buzzer can be either on (producing noise) or off.

The following code is all that's required for interfacing with the active buzzer. You can replace `22` with your selected GPIO pin in the`GPIO_PIN` constant.

```
from gpiozero import Buzzer
import time

GPIO_PIN: int = 22  # The GPIO pin the buzzer is connected to (22 for this demo)

buzzer = Buzzer(GPIO_PIN) # Instantiate the buzzer controller

# In an infinite loop, turn the buzzer on and off in 1 second intervals
while True:
    buzzer.on()
    time.sleep(1)
    buzzer.off()
    time.sleep(1)
```

As long as you have the gpiozero module installed, you can run this Python code and listen to your active buzzer turn on and off in a loop! Make sure to kill the program before your ears start ringing :)

Passive Buzzer Controller

The passive buzzer is a little more interesting because we have control over the tone the buzzer produces. Just the like the active buzzer, the passive buzzer has two states: playing and stopped (on and off). However, when on, the buzzer can play several different tones. The gpiozero module has convenient tools for selecting tone as well.

```
from gpiozero import TonalBuzzer
from gpiozero.tones import Tone
import time

GPIO_PIN: int = 22  # The GPIO pin the buzzer is connected to (22 for this demo)

buzzer = TonalBuzzer(GPIO_PIN) # Instantiate the buzzer controller

# Create a couple of tones to play
B_FLAT: Tone = Tone.from_frequency(466.164)
A_4: Tone = Tone("A4")

# In an infinite loop, turn the buzzer on (playing a B flat) and off in 1 second intervals
while True:
    buzzer.play(B_FLAT) # You can change the tone to something else if you want
    time.sleep(1)
    buzzer.stop()
    time.sleep(1)
```

The program is very similar to the active buzzer, but when we turn our buzzer on we use the .play(tone) method to play a specific tone. In this demo I used B flat, and also declared A_4 to be a concert A note if I wanted to change the note later.

There are several ways to select tones in gpiozero, which you can find documented here. [2] Some of the methods include:

- From a frequency in Hz (e.g. 466.164)
- From a string representing the note (e.g. "A4")
- From a midi note (e.g. 69 for A4)

A useful resource when defining tones for more musical applications is this chart of notes and their frequencies. [3]

References

[1] Mohamed, A. M. Elmahalawy, and H. M. Harb, "Developing the pulse width modulation tool (PWMT) for two timer mechanism technique in microcontrollers," Dec. 2013, doi: https://doi.org/10.1109/jec-ecc.2013.6766403.

[2] "gpiozero.tones — gpiozero 2.0.1 Documentation," gpiozero.readthedocs.io. https://gpiozero.readthedocs.io/en/stable/_modules/gpiozero/tones.html (accessed Mar. 18, 2024).

[3] R. M. Mottola, "Liutaio Mottola Lutherie Information Website," Liutaio Mottola Lutherie Information Website, Jan. 09, 2020. https://www.liutaiomottola.com/formulae/freqtab.htm

[4] "Raspberry Pi Documentation - Raspberry Pi Hardware,"www.raspberrypi.com. https://www.raspberrypi.com/documentation/computers/raspberry-pi.html