Part 2

lesson 6

Passive Buzzer

Overview

In this lesson, you will learn how to use a passive buzzer.

The purpose of the experiment is to generate eight different sounds, each sound lasting 0.5 seconds: from Alto Do (523Hz), Re (587Hz), Mi (659Hz), Fa (698Hz), So (784Hz), La (880Hz), Si (988Hz) to Treble Do (1047Hz).

Component Required:

(1) x Elegoo Uno R3

(1) x Passive buzzer

(2) x F-M wires (Female to Male DuPont wires)

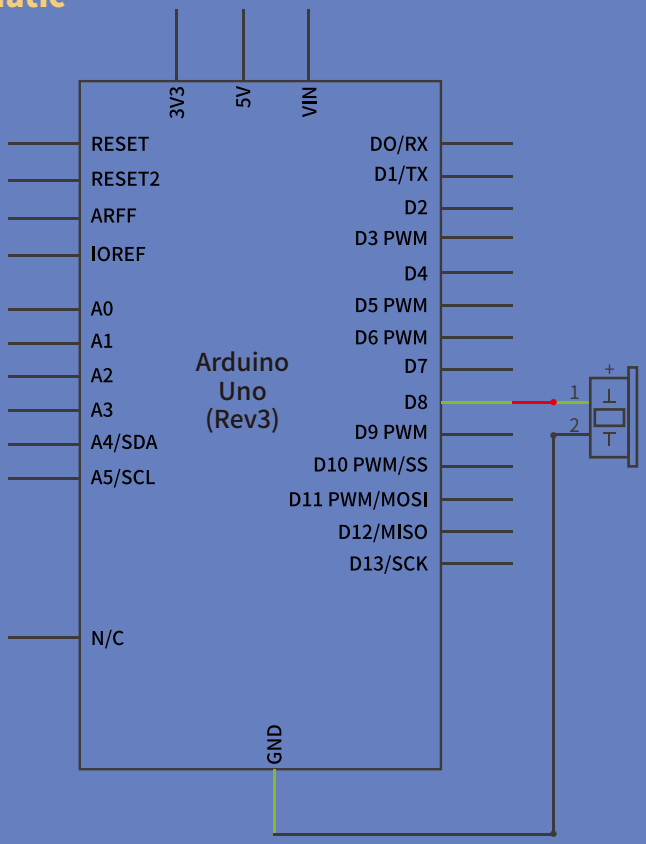
Component Introduction

Passive Buzzer:

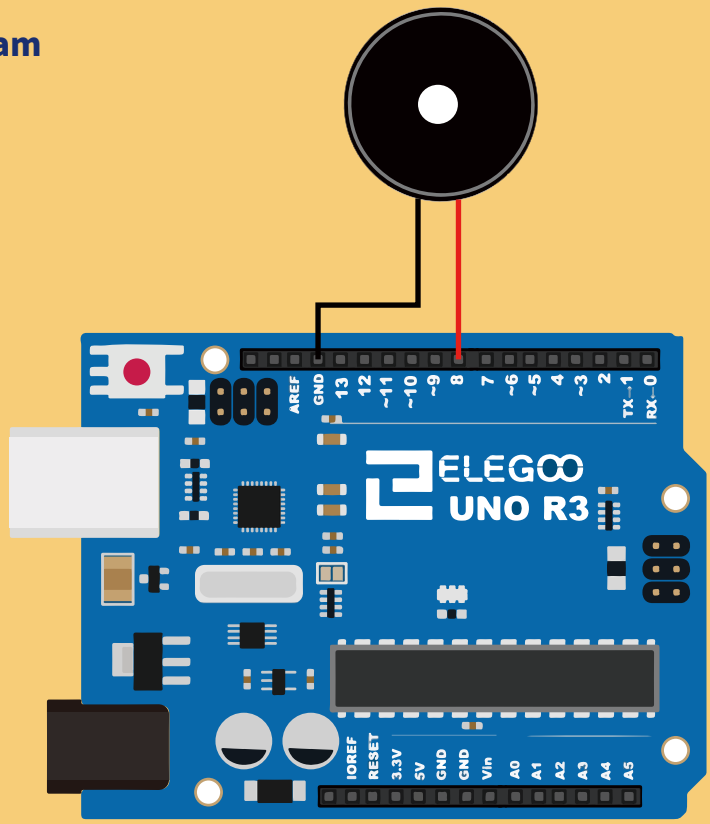
The working principle of passive buzzer is using PWM generating audio to make the air to vibrate. Appropriately changed as long as the vibration frequency, it can generate different sounds. For example, sending a pulse of 523Hz, it can generate Alto Do, pulse of 587Hz, it can generate midrange Re, pulse of 659Hz, it can produce midrange Mi. By the buzzer, you can play a song.

We should be careful not to use the UNO R3 board **analogWrite ()** function to generate a pulse to the **ACTIVE** buzzer, because the pulse output of **analogWrite ()** is fixed (500Hz).

Connection Schematic



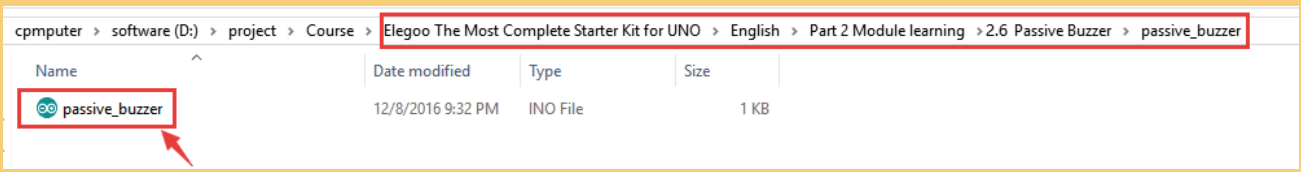
Wiring diagram



Wiring the buzzer to the UNO R3 board, the red (positive) to the pin8, black wire (negative) to the GND.

Code

- Please open the program:



- Before you can run this, make sure that you have installed the **<pitches>** library or re-install it, if necessary. Otherwise, your code won't work.

- For details about loading the library file, see Lesson 5 in part 1.

```
int melody [] = { NOTE_C5, NOTE_D5, NOTE_E5, NOTE_F5, NOTE_G5, NOTE_A5, NOTE_B5, NOTE_C6};
int melody [] = {....}   : is a array.
```

array
[Data Types]
Description

- An array is a collection of variables that are accessed with an index number. Arrays in the C++ programming language Arduino sketches are written in can be complicated, but using simple arrays is relatively straightforward.

Creating (Declaring) an Array

- All of the methods below are valid ways to create (declare) an array.
For example:

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

- You can declare an array without initializing it as in myInts.

- In myPins we declare an array without explicitly choosing a size. The compiler counts the elements and creates an array of the appropriate size.

- Finally you can both initialize and size your array, as in mySensVals. Note that when declaring an array of type char, one more element than your initialization is required, to hold the required null character.

Accessing an Array

- Arrays are zero indexed, that is, referring to the array initialization above, the first element of the array is at index 0, hence.
- mySensVals[0] == 2, mySensVals[1] == 4, and so forth.
- It also means that in an array with ten elements, index nine is the last element.
  For example:

```
int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};
// myArray[9]   contains 11
// myArray[10]   is invalid and contains random information (other memory address)
```

- For this reason you should be careful in accessing arrays. Accessing the end of an array (using an index number greater than your declared array size - 1) is reading from memory that is in use for other purposes. Reading from these locations is probably not going to do much except yield invalid data. Writing to random memory locations is definitely a bad idea and can often lead to unhappy results such as crashes or program malfunction. This can also be a difficult bug to track down.
- Unlike BASIC or JAVA, the C++ compiler does no checking to see if array access is within legal bounds of the array size that you have declared.

Creating (Declaring) an Array

- mySensVals[0] = 10;

To retrieve a value from an array:

- x = mySensVals[4];

```
tone(8 , melody[thisNote] , duration);
```

tone()

- **[Advanced I/O]**

Description

Generates a square wave of the specified frequency (and 50% duty cycle) on a pin. A duration can be specified, otherwise the wave continues until a call to noTone(). The pin can be connected to a piezo buzzer or other speaker to play tones.

- Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone () will have no effect. If the tone is playing on the same pin, the call will set its frequency.
- Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).
- It is not possible to generate tones lower than 31Hz.

Syntax

```
tone(pin, frequency)
tone(pin, frequency, duration)
```

Parameters

pin: the Arduino pin on which to generate the tone.
frequency: the frequency of the tone in hertz. Allowed data types: unsigned int.
duration: the duration of the tone in milliseconds (optional). Allowed data types: unsigned long.

Returns

- Nothing

Notes and Warnings

- If you want to play different pitches on multiple pins, you need to call noTone() on one pin before calling tone() on the next pin.