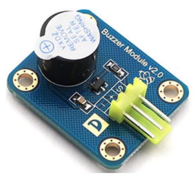
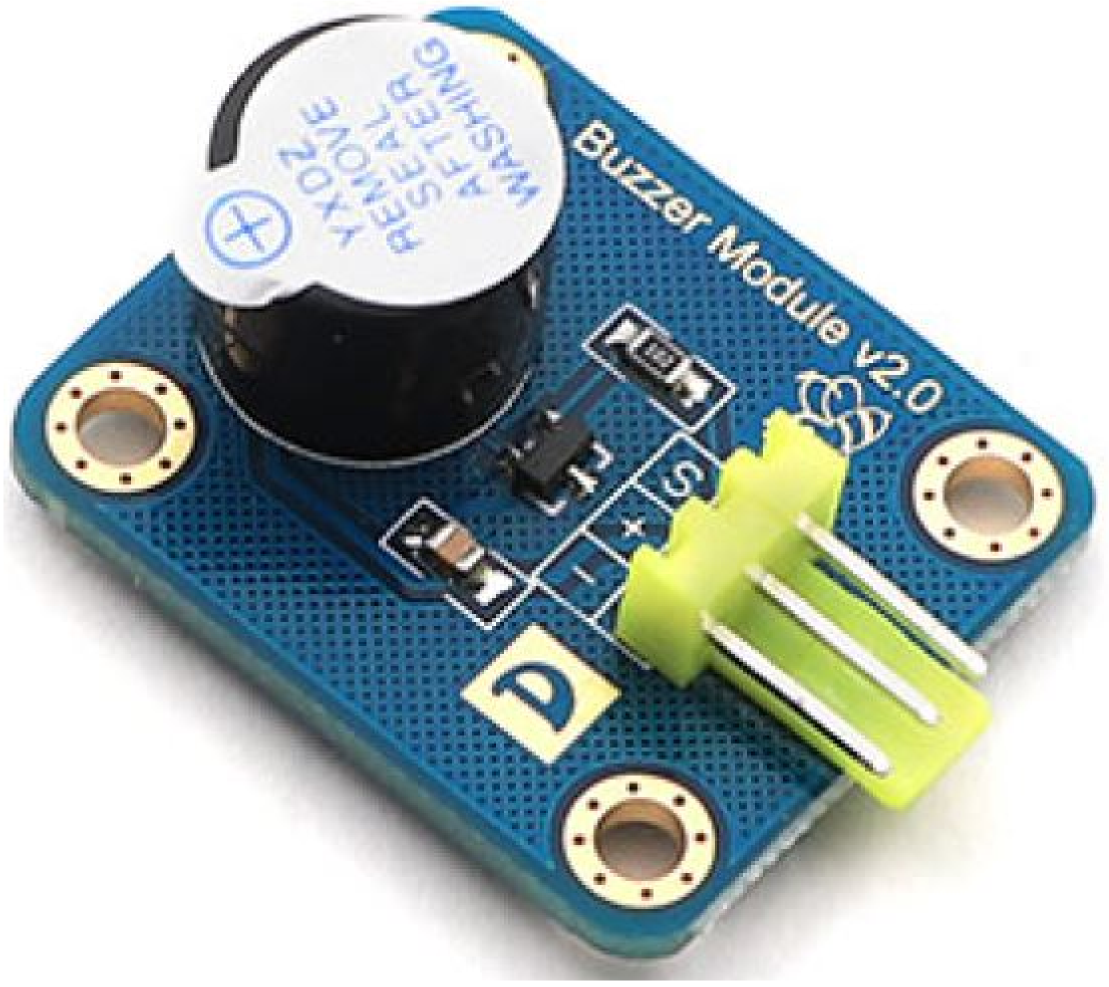


# ARDUINO Sensor Buzzer 5V Module User Manual

Contents

ARDUINO

## ARDUINO Sensor Buzzer 5V Module



**Arduino Sensor Buzzer 5V User Manual**

The Arduino Sensor Buzzer 5V is an electronic device used to play tones and melodies. It takes advantage of the processor's capability to produce PWM signals to play music. The buzzer is plugged into pin number 9, which supports the functionality of writing a PWM signal to it.

It is important to note that buzzers have polarity. Commercial devices typically have a red and black wire, indicating how to plug it into the board.

## Product Connection

| Arduino | 5V | GND | Pin 9 |
|---|---|---|---|
| | + | | S |

**Example 1: Play Melody**

```
// Play Melody
// ----------
// Program to play a simple melody
//
// Tones are created by quickly pulsing a speaker on and off
// using PWM, to create signature frequencies.
//
// Each note has a frequency, created by varying the period of
// vibration, measured in microseconds. We'll use pulse-width
// modulation (PWM) to create that vibration.
//
// We calculate the pulse-width to be half the period; we pulse
```

To use the Arduino Sensor Buzzer 5V, follow these steps:

1. Connect the Arduino board's 5V pin to the buzzer's positive (+) terminal.
2. Connect the Arduino board's GND pin to the buzzer's ground (GND) terminal.
3. Connect the Arduino board's pin 9 to the buzzer's signal (S) terminal.

Once the connections are made, you can upload the provided example code to your Arduino board. This code will play a simple melody using pulse-width modulation (PWM) to create different tones.

## Play Melody

- This example makes use of a buzzer in order to play melodies. We are taking advantage of the processors capability to produce PWM signals in order to play music.
- A buzzer is nothing but an electronic device that is used to play tones In our example we are plugging the buzzer on the pin number 9, that supports the functionality of writing a PWM signal to it, and not just a plain HIGH or LOW value.
- The first example of the code will just send a square wave to the buzzer, while the second one will make use of the PWM functionality to control the volume through changing the Pulse Width.
- The other thing to remember is that buzzers have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board.

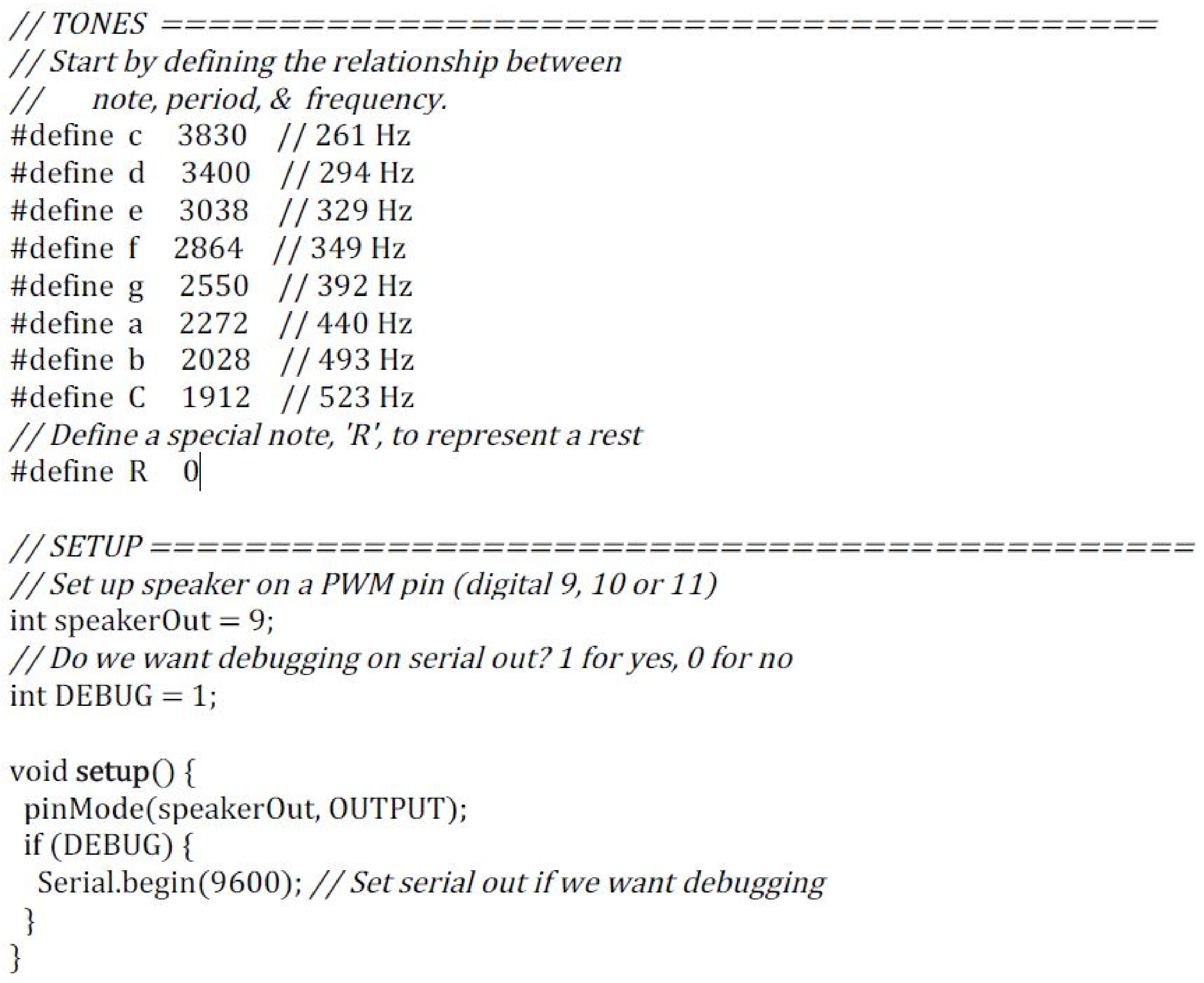
## Connection

- Arduino 412 ARDUINO SENSOR BUZZER 5V
- 5V +
- GND –
- Pin 9 S

### Example 1: Play Melody

- Play Melody
- ———
- Program to play a simple melody

- Tones are created by quickly pulsing a speaker on and off
- using PWM, to create signature frequencies.
- Each note has a frequency, created by varying the period of
- vibration, measured in microseconds. We'll use pulse-width
- modulation (PWM) to create that vibration.
- We calculate the pulse-width to be half the period; we pulse * the speaker HIGH for 'pulse-width' microseconds, then LOW
- for 'pulse-width' microseconds.
- This pulsing creates a vibration of the desired frequency.
- (cleft) 2005 D. Cuartielles for K3
- Refactoring and comments 2006 clay.shirky@nyu.edu
- See NOTES in comments at end for possible improvements



```
// TONES ==========================================
// Start by defining the relationship between
//      note, period, &  frequency.
#define c    3830    // 261 Hz
#define d    3400    // 294 Hz
#define e    3038    // 329 Hz
#define f    2864    // 349 Hz
#define g    2550    // 392 Hz
#define a    2272    // 440 Hz
#define b    2028    // 493 Hz
#define C    1912    // 523 Hz
// Define a special note, 'R', to represent a rest
#define R    0

// SETUP ============================================
// Set up speaker on a PWM pin (digital 9, 10 or 11)
int speakerOut = 9;
// Do we want debugging on serial out? 1 for yes, 0 for no
int DEBUG = 1;

void setup() {
  pinMode(speakerOut, OUTPUT);
  if (DEBUG) {
    Serial.begin(9600); // Set serial out if we want debugging
  }
}
```

```
// MELODY and TIMING =======================================
// melody[] is an array of notes, accompanied by beats[],
// which sets each note's relative length (higher #, longer note)
int melody[] = { C, b, g, C, b, e, R, C, c, g, a, C };
int beats[] = { 16, 16, 16, 8, 8, 16, 32, 16, 16, 16, 8, 8 };
int MAX_COUNT = sizeof(melody) / 2; // Melody length, for looping.

// Set overall tempo
long tempo = 10000;
// Set length of pause between notes
int pause = 1000;
// Loop variable to increase Rest length
int rest_count = 100; //<-BLETCHEROUS HACK; See NOTES

// Initialize core variables
int tone_ = 0;
int beat = 0;
long duration = 0;
// PLAY TONE ==============================================
// Pulse the speaker to play a tone for a particular duration
void playTone() {
 long elapsed_time = 0;
 if (tone_ > 0) { // if this isn't a Rest beat, while the tone has
  // played less long than 'duration', pulse speaker HIGH and LOW
  while (elapsed_time < duration) {

   digitalWrite(speakerOut,HIGH);
   delayMicroseconds(tone_ / 2);

   // DOWN
   digitalWrite(speakerOut, LOW);
   delayMicroseconds(tone_ / 2);

   // Keep track of how long we pulsed
   elapsed_time += (tone_);
  }
 }
 else { // Rest beat; loop times delay
  for (int j = 0; j < rest_count; j++) { // See NOTE on rest_count
   delayMicroseconds(duration);
  }
 }
}
```

```
// LET THE WILD RUMPUS BEGIN =============================
void loop() {
  // Set up a counter to pull from melody[] and beats[]
  for (int i=0; i<MAX_COUNT; i++) {
    tone_ = melody[i];
    beat = beats[i];

    duration = beat * tempo; // Set up timing

    playTone();
    // A pause between notes...
    delayMicroseconds(pause);

    if (DEBUG) { // If debugging, report loop, tone, beat, and duration
      Serial.print(i);
      Serial.print(":");
      Serial.print(beat);
      Serial.print(" ");
      Serial.print(tone_);
      Serial.print(" ");
      Serial.println(duration);
    }
  }
}

/*
 * NOTES
```

- The program purports to hold a tone for 'duration' microseconds.
- Lies lies lies! It holds for at least 'duration' microseconds, _plus_
  - any overhead created by incremeting elapsed_time (could be in excess of
  - 3K microseconds) _plus_ overhead of looping and two digitalWrites()
- As a result, a tone of 'duration' plays much more slowly than a rest
- of 'duration.' rest_count creates a loop variable to bring 'rest' beats
- in line with 'tone' beats of the same length.
- rest_count will be affected by chip architecture and speed, as well as
- 
  - overhead from any program mods. Past behavior is no guarantee of future
  - performance. Your mileage may vary. Light fuse and get away.
- This could use a number of enhancements:
- ADD code to let the programmer specify how many times the melody should
- loop before stopping
- ADD another octave
- MOVE tempo, pause, and rest_count to #define statements
- RE-WRITE to include volume, using analogWrite, as with the second program at
- **http://www.arduino.cc/en/Tutorial/PlayMelody**
- ADD code to make the tempo settable by pot or other input device

- ADD code to take tempo or volume settable by serial communication
- (Requires 0005 or higher.)
- ADD code to create a tone offset (higer or lower) through pot etc
- REPLACE random melody with opening bars to ‘Smoke on the Water’
- Second version, with volume control set using analogWrite()

**Play Melody**

Program to play melodies stored in an array, it requires to know * about timing issues and about how to play tones.

- The calculation of the tones is made following the mathematical * operation:
    - timeHigh = 1/(2 * toneFrequency) = period / 2
    - where the different tones are described as in the table:
    - note frequency period PW (timeHigh)
    - c 261 Hz 3830 1915
    - d 294 Hz 3400 1700
    - e 329 Hz 3038 1519
    - f 349 Hz 2864 1432
    - g 392 Hz 2550 1275
    - a 440 Hz 2272 1136
    - b 493 Hz 2028 1014
    - C 523 Hz 1912 956
    - (cleft) 2005 D. Cuartielles for K3 */

```
int ledPin = 13;
int speakerOut = 9;
byte names[] = {'c', 'd', 'e', 'f', 'g', 'a', 'b', 'C'};
int tones[] = {1915, 1700, 1519, 1432, 1275, 1136, 1014, 956};
byte melody[] = "2d2a1f2c2d2a2d2c2f2d2a2c2d2a1f2c2d2a2a2g2p8p8p8p";
// count length: 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0
//                       10                  20                  30
int count = 0;
int count2 = 0;
int count3 = 0;
int MAX_COUNT = 24;
int statePin = LOW;

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

void loop() {
  analogWrite(speakerOut, 0);
  for (count = 0; count < MAX_COUNT; count++) {
    statePin = !statePin;
    digitalWrite(ledPin, statePin);
    for (count3 = 0; count3 <= (melody[count*2] - 48) * 30; count3++) {
      for (count2=0;count2<8;count2++) {
        if (names[count2] == melody[count*2 + 1]) {
          analogWrite(speakerOut,500);
          delayMicroseconds(tones[count2]);
          analogWrite(speakerOut, 0);
          delayMicroseconds(tones[count2]);
                                }
                                if (melody[count*2 + 1] == 'p') {
                                  // make a pause of a certain size
                                  analogWrite(speakerOut, 0);
                                  delayMicroseconds(500);
                                }
                              }
                            }
                          }
                        }
```

## Documents / Resources

*Arduino Sensor Buzzer 5V*
*User Manual*

*Play Melody:*
This example makes use of a buzzer in order to play melodies. We are taking advantage of the processors capability to produce PWM signals in order to play music.
A buzzer is nothing but an electronic device that is used to play tones in our example we are plugging the buzzer on the pin number 9, that supports the functionality of writing a PWM signal to it, and not just a plain HIGH or LOW value.

The first example of the code will just send a square wave to the buzzer, while the second one will make use of the PWM functionality to control the volume through changing the Pulse Width.

The other thing to remember is that buzzers have polarity, commercial devices are usually having a red and a black wires indicating how to plug it to the board.

*Connection:*

| | |
|---|---|
| Arduino | 412 ARDUINO SENSOR BUZZER 5V |
| 5V | + |
| GND | - |
| Pin 9 | S |

***Example 1: Play Melody***

```
/* Play Melody
 * -----------
 *
 * Program to play a simple melody
 *
 * Tones are created by quickly pulsing a speaker on and off
 *   using PWM, to create signature frequencies.
 *
 * Each note has a frequency, created by varying the period of
 * vibration, measured in microseconds. We'll use pulse-width
 * modulation (PWM) to create that vibration.

 * We calculate the pulse-width to be half the period; we pulse
```

**ARDUINO Sensor Buzzer 5V Module** [pdf] User Manual
412, Sensor Buzzer 5V Module, Buzzer 5V Module, 5V Module