soberton inc.

How to Drive a Piezo Buzzer from a Microcontroller

A piezo buzzer is loudest when it is driven with a square wave at its resonant frequency, and the drive voltage sets the volume. A single microcontroller pin can do the job for a quiet beep, but two pins in antiphase roughly double the sound pressure, and a transistor or driver IC gets you to the full rated loudness.
This guide covers how to tell which kind of buzzer you have, why a piezo behaves differently from an LED or a resistor, four drive circuits from simplest to loudest, working code for Arduino, ESP32, STM32 and bare AVR, and how to protect your microcontroller.

First, check which kind of buzzer you have

The datasheet tells you whether the part needs a drive signal from you. Get this wrong and the buzzer stays silent or, worse, you damage it.

Type	What's inside	What you supply	Typical datasheet wording
Piezo transducer	Piezo element only	A square wave at the rated frequency	"Externally driven", "transducer", rated in Vp-p
Piezo indicator	Piezo element plus oscillator	DC voltage; it makes its own tone	"Internally driven", "self-drive", "indicator"
Magnetic transducer	Coil, magnet and diaphragm	A square wave, usually through a transistor	"Magnetic", coil resistance in ohms

This article is about **piezo transducers**. An indicator just needs to be switched on and off like an LED. A magnetic buzzer draws real current through its coil (tens of milliamps), so it always needs a transistor and a flyback diode; never drive one straight from a GPIO pin.
Some piezo transducers have a third "feedback" terminal. That pin is for a self-oscillating transistor circuit; for microcontroller drive you can usually leave it unconnected and use the other two.

How a piezo transducer behaves electrically

To your circuit, a piezo transducer is mostly a capacitor, typically 5 to 50 nF. It draws current only while the voltage across it is changing, in a short spike at every edge of the square wave, and almost nothing in between. That has three consequences.

- **Volume follows voltage swing, not current.** Doubling the peak-to-peak voltage raises sound pressure by about 6 dB. That is why the drive methods below are mostly ways of getting more voltage across the element.
- **Frequency matters a lot.** The element is mechanically resonant. At the rated frequency it can be 10 to 20 dB louder than an octave away, so use the frequency from the datasheet, not a round number you like.
- **Edges cause current spikes.** Charging tens of nanofarads from a GPIO pin produces a brief surge limited only by the pin's internal resistance. A small series resistor tames it.

Use a 50% duty cycle. It puts the most energy into the fundamental. Changing the duty cycle is a poor volume control: it mostly changes the harmonic content and makes the tone harsher or

thinner.

Four ways to drive it, from simplest to loudest

Method	Parts	Peak-to-peak across piezo	Relative loudness
1. Direct from one GPIO	1 resistor	VDD (e.g. 3.3 V)	Baseline
2. Two GPIOs in antiphase	2 resistors	2 × VDD	About +6 dB
3a. Transistor with resistor	Transistor, 2 resistors	V+ (e.g. 12 V)	+11 dB at 12 V vs 3.3 V
3b. Transistor with inductor	Transistor, inductor, resistor	Several × V+	Highest from discretes
4. Piezo driver IC	IC, a few passives	Set by the IC, often 3–4 × VDD	Rated output in a small footprint

1. Direct from one GPIO

Connect one terminal of the piezo to a timer or PWM pin through a 100 to 220 Ω resistor, and the other terminal to ground. This is the right choice for keypad clicks, confirmation beeps and anything heard from arm's length. Expect noticeably less than the datasheet SPL, which is usually specified at a higher voltage.
The resistor limits the charging spike. With a 220 Ω resistor and a 20 nF element, the time constant is about 4 µs, far shorter than a half-period at typical buzzer frequencies, so you lose almost no volume.

2. Two GPIOs in antiphase

Connect the piezo between two pins, each through its own resistor, and drive them with opposite square waves. When one pin is high the other is low, so the element sees +VDD and then −VDD: double the swing for no extra parts. Use two outputs of the same timer (or a timer's complementary output pair) so the edges stay aligned. Two independent software-toggled pins will drift and lose most of the benefit.

3a. Transistor with a resistor

When you need full rated loudness, switch a higher supply with an NPN transistor or logic-level N-MOSFET. Put the piezo between V+ and the collector (or drain) with a resistor in parallel with it, and drive the transistor's base through about 1 kΩ (or the gate directly) from the timer pin. When the transistor turns on, the piezo charges to V+; when it turns off, the parallel resistor discharges it.
Size that resistor so the discharge is quick compared with half a period. A good rule is $R \times C \leq 1 / (10 \times f)$. For a 20 nF element at 4 kHz, that gives about 1.2 kΩ; 1 kΩ is a sensible choice.

3b. Transistor with an inductor

Replace the parallel resistor with an inductor and the circuit borrows a trick from flyback converters. When the transistor turns off, the inductor's stored current swings the piezo voltage

well above V+, often to several times the supply. This is how many smoke alarms get very loud tones from a small battery.
Choose the inductor to resonate with the piezo near the drive frequency, using $L = 1 / ((2\pi f)^2 \times C)$, and confirm with the buzzer datasheet if it gives a value. Do not add a flyback diode across the inductor; it would clamp away the boost. Instead, pick a transistor rated well above the peak voltage you measure.

4. Dedicated piezo driver IC

Several manufacturers make small piezo driver chips, for example the Diodes Inc. PAM8904, that take a logic-level signal from your microcontroller and drive the element with a boosted voltage, often in a bridge configuration. They cost more than discretes but save board space and design time, and they make loudness predictable. Check the IC's datasheet for its output voltage and the load capacitance it supports.

Code examples

Generate the tone with a hardware timer, not by toggling a pin in a delay loop. A timer gives a clean, exact frequency and frees the CPU. The examples use 4000 Hz; replace it with your part's rated frequency.

Arduino (single pin)

tone() is the quickest start. On an Uno it uses Timer2, which disables analogWrite() on pins 3 and 11 while a tone plays.

```
const int BUZZER_PIN = 8;
const unsigned int FREQ_HZ = 4000;   // use the datasheet's rated
frequency

void beep(unsigned long ms) {
  tone(BUZZER_PIN, FREQ_HZ, ms);     // non-blocking; stops after ms
}

void setup() {
  beep(200);
}

void loop() {}
```

ESP32 (Arduino core 3.x, LEDC peripheral)

```
const int BUZZER_PIN = 25;
const uint32_t FREQ_HZ = 4000;

void setup() {
  ledcAttach(BUZZER_PIN, FREQ_HZ, 10);  // pin, frequency, resolution
bits
  ledcWriteTone(BUZZER_PIN, 0);         // start silent
}
```

```
void beep(uint32_t ms) {
  ledcWriteTone(BUZZER_PIN, FREQ_HZ);   // 50% duty square wave
  delay(ms);
  ledcWriteTone(BUZZER_PIN, 0);         // stop
}

void loop() {
  beep(150);
  delay(1000);
}
```

Core 2.x used ledcSetup() and ledcAttachPin() with a channel number instead; the idea is the same.

STM32 (HAL, timer PWM)

Configure the timer in CubeMX for PWM on the buzzer pin, with the prescaler set so the counter runs at 1 MHz (PSC = timer clock in MHz − 1). Then:

```
extern TIM_HandleTypeDef htim3;

void buzzer_start(uint32_t freq_hz) {
    uint32_t arr = 1000000UL / freq_hz - 1;              // period in
1 MHz ticks
    __HAL_TIM_SET_AUTORELOAD(&htim3, arr);
    __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, (arr + 1) / 2);  //
50% duty
    HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);
}

void buzzer_stop(void) {
    HAL_TIM_PWM_Stop(&htim3, TIM_CHANNEL_1);
}
```

For antiphase drive on STM32, use an advanced timer (TIM1 or TIM8) and its complementary pair, CH1 and CH1N, with dead time set to zero. Start the second output with HAL_TIMEx_PWMN_Start().

AVR / Arduino Uno (two pins in antiphase)

This uses Timer1 in CTC mode to toggle OC1A (pin 9) and OC1B (pin 10) together, then forces one of them once so they run in opposite phase. Output frequency = 16 MHz / (2 × 8 × (1 + OCR1A)).

```
void buzzerStart(uint16_t freq_hz) {
  pinMode(9, OUTPUT);
  pinMode(10, OUTPUT);
  uint16_t top = F_CPU / (2UL * 8 * freq_hz) - 1;   // 249 for 4 kHz
  TCCR1A = _BV(COM1A0) | _BV(COM1B0);   // toggle OC1A and OC1B on
compare match
  TCCR1B = _BV(WGM12) | _BV(CS11);      // CTC mode, prescaler 8
  OCR1A = top;
  OCR1B = top;
  TCNT1 = 0;
  TCCR1C = _BV(FOC1A);                  // flip OC1A once: outputs
```

```
now opposite
}

void buzzerStop() {
  TCCR1A = 0;                          // release pins from the
timer
  TCCR1B = 0;
  digitalWrite(9, LOW);
  digitalWrite(10, LOW);               // leave no DC across the
element
}
```

Protecting the circuit and troubleshooting

A piezo element works in both directions: a drop, a knock on the enclosure or a sharp temperature change can make it generate a voltage spike. In direct-drive circuits, the series resistor plus the microcontroller's internal clamp diodes usually handle it. In products that face drops or vibration, add a small TVS diode or a pair of Schottky diodes to the supply rails at the pin.

With transistor drive, put a bulk capacitor (10 µF or more) near the buzzer supply and keep its return path away from sensitive analog ground. The current pulses at every edge can otherwise couple into ADC readings or cause brownouts on a weak supply.

Symptom	Likely cause	Fix
Silent	It's an indicator getting a square wave, or a transducer getting DC	Check the datasheet: indicators want DC, transducers want a tone
Much quieter than the datasheet	Off-resonance frequency, or lower drive voltage than the rated test condition	Use the rated frequency; use antiphase, transistor or driver IC drive
Harsh or buzzy tone	Duty cycle not 50%, or the element rattling in its mounting	Fix the duty cycle; check the enclosure and mounting
Clicks at start or stop	Waveform starting or stopping mid-cycle	Gate the timer at a cycle boundary, or ramp the supply on transistor circuits
Microcontroller resets when it beeps	Supply dips from current pulses	Add bulk and local decoupling; separate buzzer supply return
Other PWM outputs stop working	The tone function borrowed a shared timer	Move the buzzer or the PWM to a different timer

The enclosure matters as much as the circuit. Most transducers are specified with a defined front chamber and sound port. A buzzer mounted behind a solid panel, or with its port blocked, can lose more volume than any drive circuit can win back.

Choosing a method

Start with the quietest method that meets your loudness target, because every step up adds parts and board space.

- **Feedback beeps and UI clicks:** one GPIO and a resistor.
- **Louder, still no extra parts:** two timer outputs in antiphase.
- **Alarms and alerts at rated SPL:** a transistor from a higher supply, or a driver IC if space is tight.

- **Maximum loudness from a low-voltage battery:** a transistor with an inductor, or a boosting driver IC.

Whatever you choose, take the frequency and test voltage from the buzzer's datasheet, use a hardware timer at 50% duty, and give the sound port a clear path out of the enclosure. Soberton datasheets list the rated frequency, drive voltage and SPL test conditions for every transducer, and our support pages cover drive circuits and enclosure design in more detail.

Prepared with artificial intelligence assistance and verified by the Soberton engineering team.