



#### APPLICATION NOTE 4148

## Piezoelectric Tone Generation Using the MAXQ3210

*Abstract: The MAXQ3210 microcontroller integrates a piezoelectric horn driver to drive a high-volume horn at a fixed frequency. This application note demonstrates how to modify the feedback element of the horn driver so that tones of varying frequency can be generated with software.*

### Introduction

A piezoelectric buzzer or speaker translates electric signals into sound by using a piezoelectric crystal element, which deforms slightly when a voltage is applied to it. This crystal element is connected to a speaker cone or panel, which vibrates quickly when a rapidly changing voltage is applied. This vibration, in turn, sets up sound waves in the air, thus generating a tone at a frequency based on the frequency of the voltage wave.

Early computers and electronic games used every type of sound that piezoelectric speakers can generate, from single-pitch tones to siren sounds and white noise. Sound generation technology in personal computers has made great strides from those early beginnings. Today, even if multiple channels, digitized sound and music, and MIDI-based music synthesizer systems are added to a system, the simple piezoelectric speaker is still useful. Car alarms, smoke detectors, point-of-sale (PoS) terminals, small electronic toys and games, and many other applications rely on piezoelectric sound generation to create simple tones, high-decibel alerts, and sound effects.

### Hardware Modifications to Drive the Speaker

Adding the hardware for a piezoelectric speaker in a system is fairly straightforward.

In applications where only a single output frequency is desired, one can use a piezoelectric horn optimized to oscillate at a particular audible frequency. By adding a feedback network and an inverter, the horn self-oscillates at its resonant frequency when it is switched on. These types of networks are often used to generate high-decibel alert tones in smoke detectors, carbon-monoxide sensors, and security systems.

Maxim's [MAXQ3210](#) microcontroller integrates a drive circuit for this type of piezoelectric speaker (**Figure 1**). The software interface is extremely simple, consisting of a single control bit which switches the horn on and off. The frequency and volume of the output tone are determined entirely by the piezoelectric speaker and the feedback network components.

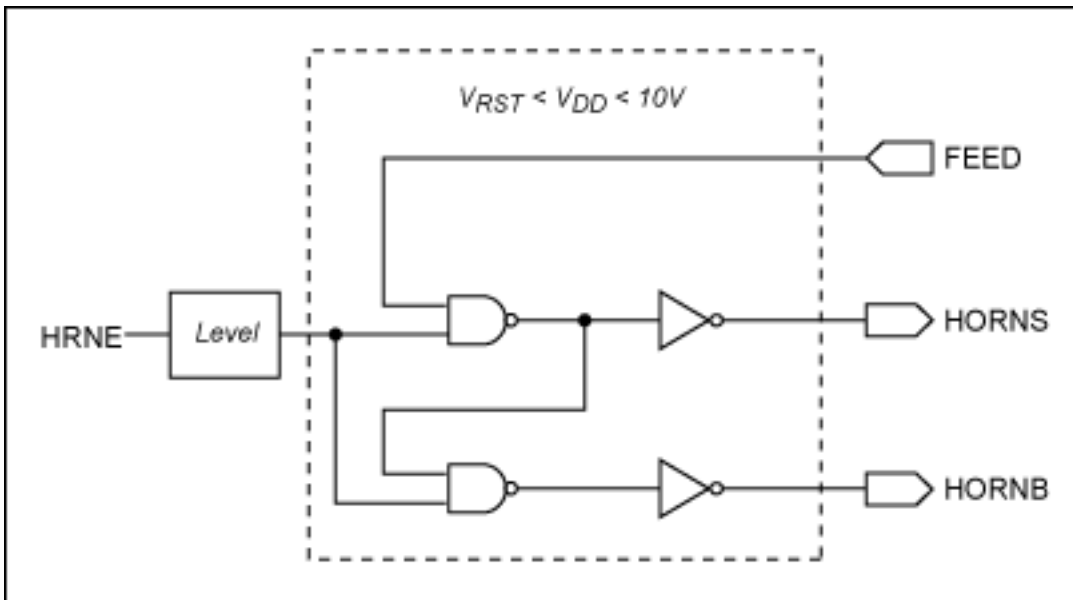


Figure 1. Piezoelectric horn-drive circuit for the MAXQ3210 microcontroller.

Switching the horn rapidly on and off under software control can create a variety of clicks, chirps, and other sound effects, but the output frequency will always remain the same. What can one do to generate different output frequencies?

To change the output frequency, one must remove the automatic feedback loop and control the horn output directly through a port pin. A digital speaker, which has inputs designed to be switched on and off between a fixed voltage and ground, can be driven directly from the digital I/O pins of a microcontroller. Depending on the size and intended decibel output of the piezoelectric speaker, a specialized driver circuit or IC may be needed to handle high voltages at the pins of the piezoelectric speaker or to drive high current into the speaker to produce higher volumes. In these situations, the microcontroller I/O pins are used to turn the speaker on and off and set the output frequency. The I/O pins do not drive the piezoelectric element directly.

The MAXQ3210 can also be used to generate multiple output frequencies for the speaker. The feedback input pin is simply connected to one of the other port pins on the device. As long as the built-in horn inverter can keep up with the frequency driven at the port pin, any desired tone can be generated by toggling the port pin at the proper rate.

**Note:** source code for this application note is available for [download](#) (ZIP, 4.5kB).

## A Basic Music Synthesizer

To generate simple musical tones, the speaker must be switched on for a certain period of time and then off for the same amount of time. The frequency of the sound generated is given by:

$$\text{Output Frequency} = 1/(\text{high period} + \text{low period})$$

To generate the sound for a particular length of time, calculate the number of complete cycles to drive:

$$\text{Number of Cycles} = \text{Sound Duration} \times \text{Output Frequency}$$

The frequencies for a basic piano scale beginning at Middle C serve as an example. These frequencies are tuned so that the note "A over Middle C" is equivalent to 440Hz. The values shown in **Table 1** are approximations.

**Table 1. Musical Note and Approximate Sound Frequency**

| Note               | Frequency (Hz) |
|--------------------|----------------|
| Middle C           | 261            |
| C sharp/D flat     | 277            |
| D                  | 294            |
| D sharp/E flat     | 311            |
| E                  | 330            |
| F                  | 349            |
| F sharp/G flat     | 370            |
| G                  | 392            |
| G sharp/A flat     | 415            |
| A                  | 440            |
| A sharp/B flat     | 466            |
| B                  | 494            |
| C (next octave up) | 523            |

The simplest way to switch the MAXQ3210's port pin on and off at the proper frequency is to use a software loop. The delay for the inner loop is calculated using the microcontroller's single-cycle instruction execution frequency, which is typically 3.57MHz.

```

    move    HRNC, #1          ; Turn the piezoelectric horn driver on.

; ; Play Middle C for one second.

    move    LC[1], #261      ; Outer loop counter = 261 cycles (1s * 261Hz)
middleC:
    move    PO0.0, #1       ; Switch output high.
    move    LC[0], #6839    ; Half period : (1/261Hz) / (1/3.57MHz) / 2
    djnz   LC[0], $         ; Decrement and jump, if not zero, to current
                          ; instruction.
    move    PO0.0, #0       ; Switch output low.
    move    LC[0], #6839    ; Half period : (1/261Hz) / (1/3.57MHz) / 2
    djnz   LC[0], $         ; Decrement and jump if not zero to current
                          ; instruction.
    djnz   LC[1], middleC   ; Decrement and jump, if not zero, to top of loop.

```

The output frequency (which corresponds to the note produced) is changed by altering the value loaded into the loop counter LC[0]. The duration of the note is changed by altering the value loaded into the loop counter LC [1]. By wrapping this piece of code inside a simple macro and defining a few constants, one can easily create code to play a short piece of music.

```

#define NOTE_C      261
#define NOTE_C_SH  277
#define NOTE_D_FL  277
#define NOTE_D      294
#define NOTE_D_SH  311
#define NOTE_E_FL  311
#define NOTE_E      330
#define NOTE_F      349
#define NOTE_F_SH  370
#define NOTE_G_FL  370
#define NOTE_G      392

```

```

#define EIGHTH      1      ; 120 beats per minute, 4/4 time
#define QUARTER     2      ; 120 beats per minute, 4/4 time
#define QUARTERDOT  3      ; 120 beats per minute, 4/4 time
#define HALF        4      ; 120 beats per minute, 4/4 time
#define WHOLE       8      ; 120 beats per minute, 4/4 time

```

```

play macro note, duration

```

```

local L1, L2

```

```

    move    HRNC, #1
    move    LC[1], #(note * duration / 8)

```

```

L1:

```

```

    move    PO0.0, #1      ; Switch output high.
    move    LC[0], #(1785000 / note)
    djnz   LC[0], $        ; Decrement and jump, if not zero, to current
                          ; instruction.
    move    PO0.0, #0      ; Switch output low.
    move    LC[0], #(1785000 / note)
    djnz   LC[0], $        ; Decrement and jump, if not zero, to current
                          ; instruction.
    djnz   LC[1], L1       ; Decrement and jump, if not zero, to top of loop.
    move    HRNC, #1
    move    LC[1], #50     ; 50ms of silence

```

```

L2:

```

```

    move    LC[0], #3570   ; 1ms (inner loop)
    djnz   LC[0], $
    djnz   LC[1], L2

```

```

endm

```

```

;; First 8 bars of Beethoven's "Ode to Joy"

```

```

play    NOTE_E,    QUARTER
play    NOTE_E,    QUARTER
play    NOTE_F,    QUARTER
play    NOTE_G,    QUARTER

```

```

play    NOTE_G,    QUARTER
play    NOTE_F,    QUARTER
play    NOTE_E,    QUARTER
play    NOTE_D,    QUARTER

```

```

play    NOTE_C,    QUARTER
play    NOTE_C,    QUARTER
play    NOTE_D,    QUARTER
play    NOTE_E,    QUARTER

```

```

play    NOTE_E,    QUARTERDOT
play    NOTE_D,    EIGHTH
play    NOTE_D,    HALF

```

```

play    NOTE_E,    QUARTER
play    NOTE_E,    QUARTER
play    NOTE_F,    QUARTER
play    NOTE_G,    QUARTER

```

```

play    NOTE_G,    QUARTER
play    NOTE_F,    QUARTER
play    NOTE_E,    QUARTER
play    NOTE_D,    QUARTER

```

```
play    NOTE_C,    QUARTER
play    NOTE_C,    QUARTER
play    NOTE_D,    QUARTER
play    NOTE_E,    QUARTER

play    NOTE_D,    QUARTERDOT
play    NOTE_C,    EIGHTH
play    NOTE_C,    HALF
```

The amount of code space required for the example can be reduced. Simply use subroutines (instead of unrolled code macros) and lookup tables for the note values. The basic principles remain the same.

## Timer-Driven Music Synthesis

The code above, while simple enough to follow, requires that the microcontroller dedicate all of its time to tone generation. For some simple applications like a musical greeting card, this single use of the microcontroller is acceptable. For more complex applications, however, tones need to be played in the background while the microcontroller tends to other operations. To accomplish this dual role, the microcontroller must be freed from the task of constantly toggling the port pin on and off for the duration of a particular note.

The MAXQ3210 integrates a counter/timer function. In one mode, this timer can generate an output waveform of a particular frequency at a port pin. By using this mode and connecting the timer output pin to the feedback input, the microcontroller simply starts the timer at the beginning of the note and stops it at the end of the note.

The timer can also be used to control the duration of the notes themselves. The MAXQ3210 contains an additional long-period timer, which is perfect for this task. By generating an interrupt at the beginning of each note interval, this timer can advance through a precoded lookup table of notes, thus allowing longer pieces of music to be encoded in less space.

## Conclusion

Piezoelectric tone generation is used in a number of applications to create musical tones, alerts, and other sound effects. By using a microcontroller with an integrated piezoelectric horn driver such as the MAXQ3210, piezoelectric tones can be generated with minimal hardware and software overhead. The integrated, programmable timer allows most of the tone generation to happen in the background, freeing the microcontroller to focus on the main application.

---

Application Note 4148: [www.maxim-ic.com/an4148](http://www.maxim-ic.com/an4148)

### More Information

For technical support: [www.maxim-ic.com/support](http://www.maxim-ic.com/support)

For samples: [www.maxim-ic.com/samples](http://www.maxim-ic.com/samples)

Other questions and comments: [www.maxim-ic.com/contact](http://www.maxim-ic.com/contact)

---

### Keep Me Informed

Preview new application notes in your areas of interest as soon as they are published. Subscribe to [EE-Mail - Application Notes](#) for weekly updates.

**Related Parts**

MAXQ3210: [QuickView](#) -- [Full \(PDF\) Data Sheet](#) -- [Free Samples](#)

AN4148, AN 4148, APP4148, Appnote4148, Appnote 4148

Copyright © by Maxim Integrated Products

Additional legal notices: [www.maxim-ic.com/legal](http://www.maxim-ic.com/legal)