Controlling a Speaker

Make clicks, beeps, and tones from a piezoelectric speaker.

Site: iCODE
Course: Machine Science Guides (Arduino Version)
Book: Controlling a Speaker
Printed by: Ivan Rudnicki
Date: Wednesday, July 30, 2014, 03:31 PM

Contents

About Speakers

Adding the Speaker

Making a Click

Making a Sustained Tone

Using the Tone_Out Function

Putting Tones in a Loop

Using a For Loop

Using Pre-Defined Notes

About Speakers

The headphones used to produce sounds from portable MP3 players, like Apple's iPod are just tiny speakers controlled by signals sent from a computer chip. Figure 1 shows some typical earbud style speakers.

Figure 1. Earbud style headphone speakers.

The speaker used in this activity is called a piezo speaker. The key component in the speaker is a material that bends slightly when you supply electricity to it. If the electricity is sent in pulses, the element vibrates back and forth. Just like the movement of a saxophone reed or a guitar string, this vibration produces audible sound waves in the air. To make the piezo-electric element vibrate, you need to raise and lower the voltage on a microcontroller pin, alternating between 0 volts and 5 volts. Figure 2 shows how the voltage signals make the piezo-electric element move.

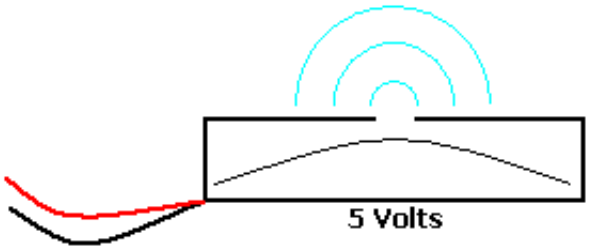

Figure 2. Voltage signals making piezo element move.

Adding the Speaker

Following the schematic in Figure 3, connect the piezoelectric speaker to Port D4. Use a two-prong bent connector to attach the speaker to the breadboard, and then use two jump wires to make the indicated connections to the microcontroller and to power. NOTE: Although the speaker has red and black wires, it doesn't matter which of the two wires connects to power and which connects to the microcontroller.

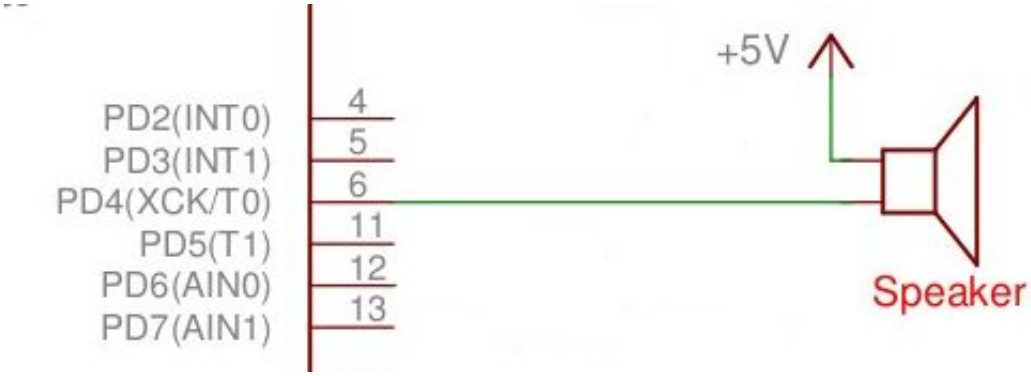

Figure 3. Piezo speaker circuit schematic.

Making a Click

By lowering and raising the voltage on one pin, you can cause the speaker to make a clicking sound.

1. Rename your code file click.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>

int main(void)
{
   output_pin(PORT_D4);  //Set up Port D4 as an output pin
   low_pin(PORT_D4);     //Set the Port D4 pin low
   delay_us(1000);       //Wait 1000 microseconds
   high_pin(PORT_D4);    //Set the Port D4 pin high
   while(1==1);          //End the program
}
```

3. Compile and test your new code.
4. Press the reset button a few times, while pressing the speaker up to your ear, you should hear a clicking sound each time.

Making a Sustained Tone

By putting a bunch of clicks together in a while loop, you can make the speaker produce a sustained tone. Note that in the code file shown here, the delay is in microseconds (delay_us), not in milliseconds (delay_ms).

1. Rename your code file sustained.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>

int main(void)
{
   output_pin(PORT_D4);    //Set up Port D4 as an output pin
   while(1==1)
   {
      high_pin(PORT_D4);   //Set the Port D4 pin high
      delay_us(1000);      //Wait 1000 microseconds
      low_pin(PORT_D4);    //Set the Port D4 pin low
      delay_us(1000);      //Wait 1000 microseconds
   }
}
```

3. Compile and test your new code.

Programming Challenge

Modify your code file to produce different tones. HINT: The value in the delay_us statement is the key.

Using the Tone_Out Function

The ATmega code library has a function called tone_out, which enables you to produce notes of specific frequencies and durations. Each time you use the tone_out function you have to pass three values: the first number is the frequency, in Hertz, of the note you want; the second number is the note's duration, in milliseconds; and the third number is the pin you want to use. For example, the statement tone(PORT_B3, 440, 200); will produce a note of 440 Hertz for 200 milliseconds on Port B3. Note that the tone_out function takes care of setting the pin as an output as well as making the tone. Before using the tone_out function, be sure to include sound.h at the top of your file.

1. Rename your code file tone.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>
#include <sound.h>

int main(void)
{
   tone_out(PORT_D4, 1000, 200);  //Make a 1000Hz tone for 200ms on Port
D4
   while(1==1);
}
```

3. Compile and test your new code.

Putting Tones in a Loop

With a while loop, you can make any sequence of tones repeat over and over.

1. Rename your code file repeattones.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>
#include <sound.h>

int main(void)
{
   while(1==1)
   {
      tone_out(PORT_D4, 440, 200);  // Make a 440-Hertz tone for 200
milliseconds
      tone_out(PORT_D4, 535, 100);  // Make a 535-Hertz tone for 100
milliseconds
   }
}
```

3. Compile and test your new code.

Using a For Loop

With a different kind of loop, you can make the frequency of the tone rise or fall. To do this, you will need to introduce two new elements into your code. The first is a variable. A variable is letter or phrase that is assigned a numerical value, which can change over time. In this case, your variable will be called x. The second is a for... loop. A for... loop repeats a segment of code a specific number of times and then stops.

1. Rename your code file forloop.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>
#include <sound.h>

int main(void)
{
   int x;           //Declare a variable, called x
   for(x=250; x<750; x=x+1)   //Set up a for loop
   {
      tone_out(PORT_D4, x, 2);  //Make a tone of x Hertz for 2
milliseconds
   }
   end();
}
```

3. Compile and test your new code.

Programming Challenge

Modify your code so that the tone falls instead of rising each time through the loop. HINT: You will need to change three things in the statement setting up the for loop. Next, wrap your for loop inside a while loop so that the each rising or falling tone sequence repeats over and over, like a siren. Finally, add code to make the LEDs flash in time with the speaker.

Using Pre-Defined Notes

The tone_out function gives you complete control over each beep's frequency. However, when making music, you don't want to produce every possible frequency. You need only those frequencies that are found in music. For those familiar with musical notation, the sound.h header file predefines note frequencies that the speaker is capable of producing, using standard musical notation, as shown in the table below.

Octave	C	C#/Db	D	D#/Eb	E	F	F#/Gb	G	G#/Ab	A	A#/Bb	B
4	262	277	294	311	330	349	370	392	415	440	466	494
5	523	554	587	622	659	698	740	784	830	880	932	988
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1975

Each define statement comprises the note letter (A-G), the octave number (4-6), and a neutral (N), sharp (S), or flat (F) indicator. Therefore, to produce the lowest note (C in the fourth octave), you would specify C4N. The next note would be C4S or D4F, and so on.

Note durations are also pre-defined, using the following notation scheme.

Note	Notation	Duration (ms)
Sixty-fourth (1/64) note	No64	25
Thirty-second (1/32) note	No32	50
Dotted thirty-second (1/32) note	Nd32	75
Sixteenth note (1/16) note	No16	100
Dotted sixteenth (1/16) note	Nd16	150
Eighth (1/8) note	No08	200
Dotted eighth (1/8) note	Nd08	300
Quarter (1/4) note	No04	400
Dotted quarter (1/4) note	Nd04	600
Half (1/2) note	No02	800
Dotted half (1/2) note	Nd02	1200
Whole note	No01	1600
Dotted whole note	Nd01	2400

1. Rename your code file note_defines.c.
2. Modify your code file, as follows:

```
#include <mxapi.h>
#include <sound.h>

int main(void)
```

```
{
    tone_out(PORT_D4, C4N, No02);  //Make the speaker beep at 262 Hz
    tone_out(PORT_D4, D4N, No02);  //Make the speaker beep at 294 Hz
    tone_out(PORT_D4, E4N, No02);  //Make the speaker beep at 330 Hz
    tone_out(PORT_D4, F4N, No02);  //Make the speaker beep at 349 Hz
    tone_out(PORT_D4, G4N, No02);  //Make the speaker beep at 392 Hz
    tone_out(PORT_D4, A4N, No02);  //Make the speaker beep at 440 Hz
    tone_out(PORT_D4, B4N, No02);  //Make the speaker beep at 494 Hz
    tone_out(PORT_D4, C5N, No02);  //Make the speaker beep at 523 Hz
    while(1==1);                   //End the program
}
```

3. Compile and test your new code.

Programming Challenge

Try making a simple melody, using the tone_out function and the predefined notes and note durations.