ZBasic

Application Note

AN-206 Using PlaySound to Generate Audio

Introduction

The PlaySound subroutine in the ZBasic System Library can be used to generate an audio signal, e.g. music, voice, tones, etc. The data that PlaySound uses to synthesize the audio signal is a series of values representing amplitude of the output signal at equally spaced intervals in time. In most cases, the data for PlaySound is obtained by digitizing an analog audio signal. During digitization, the audio signal is sampled at equally spaced intervals (controlled by the sampling frequency) and the magnitude of the audio signal at each sample point is stored. Since audio signals are inherently AC signals, the samples will typically have both positive and negative excursions from some DC level, typically zero volts.

The higher the sampling rate, the more accurately the digitization will represent the original audio signal. However, the higher the sampling rate, the larger will be the total number of samples to represent a fixed playing time. High fidelity reproduction of audio signals requires a sampling rate of at least two times the highest frequency component. Sampling at a lower rate will result in some loss of audio quality.

In order to produce an audio segment with PlaySound you need sample data that is represented by 8-bit values. If you have and existing audio file, say in .wav or .mp3 format, you can convert it to the required form using a two-step process. The first step is to convert the samples to a list of real number values. Then, you can convert that sample data to the 8-bit values required by PlaySound. Finally, you can use the converted data to produce a sound.

Converting a Digitized Audio Segment – Step 1

To perform the first step you may use any audio editing program that has the ability to export sound files as text. GoldWave is an example of such a program; it can be downloaded freely from http://www.goldwave.com. After loading the audio clip into GoldWave (using File Open), and optionally editing the clip, you may need to resample the audio depending on your requirements. For example, the sound file may have a sample rate of 11025Hz and you want to reduce the sample rate to 5500Hz in order to reduce the memory required. To do this using GoldWave, select "Resample…" from the Effects menu, choose the desired sample rate. Depending on the particulars, you may see a message indicating that the format of the sound file doesn't support the chosen sample rate. You can ignore that message since you're going to be exporting the data in numerical form.

After optionally resampling, the sample data needs to be saved in numerical form. In GoldWave, this is done by choosing "Save As…" from the File menu and then choosing "Numerical Text" from the "Save As Type" drop down box. Once this process is complete, the saved file will contain data something like the following:

```
[ASCII 11025Hz, Channels: 1, Samples: 3821, Flags: 0]
-0.003906
-0.006836
-0.000427
-0.000427
-0.000427
-0.006226
-0.007752
-0.001038
0.006714
0.000580
-0.006745
...
```

The first line of this file gives information about the exported sound file. In this case, the sampling rate was 11025Hz and there are 3821 samples in the file. This represents an audio duration of about 347 milliseconds (3821 / 11025).

Converting a Digitized Audio Segment – Step 2

The content of the exported sound file is not usable by PlaySound since it consists of real numbers and PlaySound requires the samples data to be 8-bit values. The next step in the process is to convert the list of real numbers to a list of integral values in the range 0-255. Assuming that the real number data is centered around zero (as is almost certainly the case) we can map the zero level to the integral value 128, i.e. roughly the center of the range. Next, we have to determine how to map the remaining values. For best effect, we want to map the most positive value to something close to 255 and the most negative value to something close to zero. Note, however, that it is not necessarily the case that the most positive and most negative values have the same absolute value. To address this issue, we must scan the list of numbers and find the most positive and most negative values and then use the largest absolute value of the two as a mapping scale factor. For example, if the most positive value is 0.143 and the most negative value is –0.157, we would choose 0.157 as the mapping scale factor so that 0.157 maps to 255 and –0.157 maps to 0. Note that this is an approximation since there are actually 128 quantization levels below 128 but only 127 quantization levels above 128. This fact can be taken into account in the conversion process.

Once the mapping scale factor has been selected it needs to be applied to every entry in the data file to produce the equivalent 8-bit value. This process can be performed using a spreadsheet or using a special program written for that purpose. In either case, the conversion process is the same. For each value, the following formula would be used:

```
val = sample * 127 / scaleFactor + 128
```

When implementing the conversion formula, attention will need to be given to the rounding that occurs in the conversion from a real number to an integral number. Also, a final check must be made to be certain that the result is limited to the inclusive range 0-255.

A special console application program is provided in the .zip file that accompanies this application note that implements this conversion process. The program, for which C source code is also provided, is named msample and it can be used in this manner:

```
msample music.txt > zxmusic.txt
```

This example assumes that the exported real number representation of the audio clip is in the file music.txt. The converted output of the program is captured in the file zxmusic.txt and will look something like this:

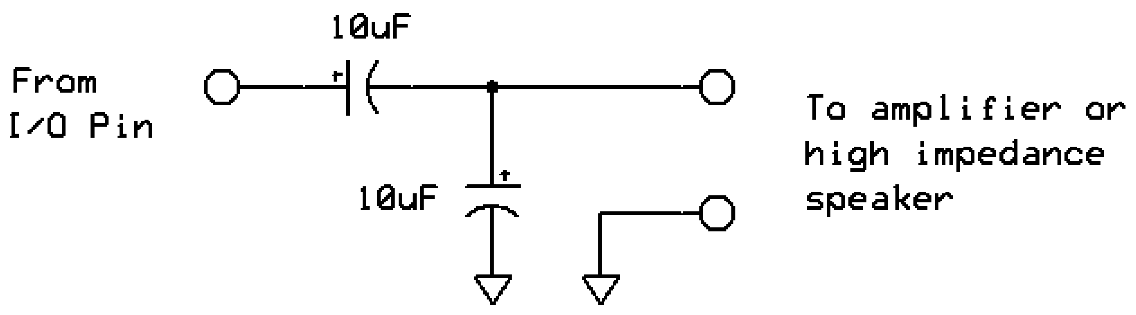
```
'[ASCII 11025Hz, Channels: 1, Samples: 3821, Flags: 0]
128
128
127
154
112
...
```

Using the Data to Generate an Audio Signal

The data format shown above is perfect for use as an initialization file for a `ByteVectorData` item in a ZBasic program. This provides a way to get the sample data into Program Memory where it can be used by the PlaySound subroutine. The sample program below shows how this can be done.

```
Private music As ByteVectorData("zxmusic.txt")
Private Const audioPin As Byte = A.7
Sub Main()
   Do
      Call PlaySound(audioPin, CUInt(music.DataAddress), UBound(music), 11025, 1)
      Call Delay(1.0)
   Loop
End Sub
```

In order to hear the sound, the synthesized output needs to be filtered and coupled to a speaker or amplifier. The example circuit shown below may be used with a high impedance speaker (> 40Ω) or an amplifier. For testing purposes, an 8Ω speaker salvaged from a PC can be used. Note that the generated signal is too large to be fed to the microphone input of an amplifier. Instead, the Auxiliary or Line input should be used.



A more sophisticated filter may improve the quality of the generated sound but that subject is beyond the scope of this application note.

Audio Samples

The accompanying .zip file contains some audio samples for experimentation purposes. Each is audio sample is provided in .wav form, in real number form and in integral number form. The latter is suitable for PlaySound. The largest sample, a short sequence from Vivaldi's *Le Quattro Stagioni*, has been reduced to 60,000 samples. This will need to be edited down further in order to fit into the 32K EEPROM.

Author

Don Kinzer is the founder and CEO of Elba Corporation. He has extensive experience in both hardware and software aspects of microprocessors, microcontrollers and general-purpose computers. Don can be contacted via email at *dkinzer@zbasic.com*.