

The use of computers to examine and alter analog signals is known as digital signal processing. A small section of this field is a key part of today's electronic music.

The Apple IIGS, with its built-in sound generation system surpasses other personal computers in this respect. Today, there are a large number of programs on the market which allow the Apple IIGS owner to use his computer to generate very sophisticated music and sound patterns. These programs allow the user to create and alter tones by editing frequency and time relationships in the waveforms sent to the tone generation system.

This article presents a behind-the-scenes look at how these transformations from numbers to sounds are accomplished. It also will allow the reader to perform his own digital signal processing experiments with the help of Fast Fourier Transforms (FFT).

Sounds, tones, and music are made up of periodic or repeating waves. These periodic waves, as they exist in the physical world, are continuous-time in nature. Computer generated tones, on the other hand, are discrete-time in nature. The term discrete-time means that the tones, or waveform data are generated from a set of discrete numerical values held in computer memory. These discrete values are presented to a converter at a set rate. The stepwise construction of the signal is passed to an audio system such as an amplifier and a set of speakers. The human ear picks up this stepwise approximation, and along with the brain, translates it back to what is perceived as a continuous-time signal.

To generate the sound of a musical note from A/C basic a waveform must be constructed in an integer array. This array is 256 entries long and contains values for each entry, which are no less than -127 and no greater than 128. The waveform is played repeatedly by the sound generation circuitry to produce a note. This data array which makes up the waveform is a perfect example of a discrete-time signal. The waveform is made up of individual integer values (or discrete elements).

These elements are known as samples. The samples have a limited range of values, set by the hardware. These discrete-time signals are often illustrated graphically. You should be able to fill in the lines to form a sine wave from this picture. The dot at the end of each line represents the discrete value for that sample point, which is held in the integer array.

When repeated continuously the sample will have a frequency associated with it. The note at A above middle C has a frequency of 440 cycles per second (cps). The frequency is also known as frequency spectrum value, or spectral point. If a waveform comprised of one sine wave at a given frequency, it has only one spectral point. A waveform comprised of two sine waves added together will have two spectral points. Waveforms, which are not pure sine waves (like square waves) will have a set of spectral points.

There are many ways to construct a waveform in an array A/C Basic. One method uses a FOR NEXT loop to set the amplitude of each of the 256 elements. This waveform could then be used in conjunction with the WAVE and SOUND statements in A/C Basic to produce a tone. Or one might copy the contents of a small file into the data array limiting the values to the proper range. This last method will produce strange noises in most cases.

Another way to produce waveforms is to translate frequency domain information into time domain information. The mathematical translation from one domain to the other is accomplished by using Fourier transforms. Fourier transforms were discovered by clever Frenchman J.B.J. Fourier.

Fourier discovered any periodic function could be described by an infinite series of sinusoids of harmonically related frequencies. Fourier's analysis is also known as frequency analysis. The Fourier integral expresses the summation of frequency values, phase angles, and frequency magnitudes.

In the equation above, $H(f)$ is the representation of the signal in the frequency domain and $h(t)$ is the representation of the signal in the time domain. This equation can be expanded to accommodate a computer algorithm. The following substitution is needed for this expansion.

This expands the original integral to:

Finally, the integral is expanded as a summation. This summation is known as the Fourier Series. The equation below shows a periodic waveform is comprised of a set of harmonically related frequencies with finite amplitudes.

where $f_0 = 1/T_0$ which is the fundamental frequency of the periodic waveform.

These equations simply state that if you know the set of frequencies and their amplitudes that make up the waveform you can reconstruct the waveform. This process can be thought of as waveform synthesis. This also means if you reverse this process you can find the frequencies and the amplitudes that make up a periodic waveform. This is also known as spectrum analysis.

The equations listed above have been translated into a computer algorithm known as a Fast Fourier Transform. This algorithm is responsible for turning waveforms into frequency spectrum and frequency spectrum into waveforms. The A/C Basic implementation of this algorithm is named FFT1. This routine has a set of requirements the calling program must adhere to. Those requirements are as follows:

- 1) The number of samples on which the algorithm will operate must be a power of 2 (i.e. 64, 238, 256...).
- 2) The array containing the input data, whether the waveform samples or the frequency spectrum samples, must be twice the size of the number of samples.
- 3) The input data array must be comprised of DOUBLES size elements.

The FFT algorithm has been implemented in BASIC to allow the user the simplest access to the Apple IIGS's tone generation hardware. However, implementation in Basic causes the FFT operation to be quite slow in calculation (about 30 seconds for a 256-point FFT). This article provides the listing of the two example programs and the listing for the FFT implementation in C. With SIN and COS tables, the algorithm can also be implemented in assembler. The algorithm is listed below.

FFT1 Algorithm:

[STEP 1] Initialize variables used in the algorithm.

[STEP 2] Perform a pre-weave butterfly operation of the input data array.

[STEP 3] Update loop counters and index variables, etc. for completion;

IF YES fall through to STEP 4 ELSE branch back to STEP 2.

[STEP 4] Setup multiplication coefficients.

[STEP 5] Perform weave (butterfly) operation and multiply each weave operator by coefficients.

[STEP 6] Check for completion. IF NOT complete increment loop counters, indexes, form ew multiplier coefficients, and branch back to STEP 5, ELSE fall through to STEP 7.

[STEP 7] Control falls through to this point when secondary weave is complete, and control is returned to the caller FFT1.

Note the four main sections of this algorithm are preceded by a comment block identifying each section. The most complex parts of the algorithm are the weaving sections. The first weave intermixes the elements of the array, keeping the real and imaginary parts of the complex number separate. The second weave causes the translation between the time and frequency domain by intermixing real and imaginary parts of each complex element of the array multiplied by a changing coefficient.

Now it is time to put the FFT to work. The first task for the FFT is to generate the correct waveform data for a set of frequency spectrum entries. To do this correctly, first insure the correct input data are presented to the FFT subroutine, These steps are as follows:

1) The frequency indexes chosen are integer multiples of the first frequency entered fundamental.

2) All entries fall in the range from +1.0 to -1.0

3) All entries should be made to even-numbered indexes in the data entry array. The constitutes the real portion of a complex number array. The odd-numbered index entries constitute the imaginary portion of the complex number array.

The programming steps listed below have been implemented in the FLOW base program (Listing 1). These steps will allow the user of the FFT1 algorithm to correctly generate new sound waveforms. In this program the user inputs the spectral frequencies and the amplitude values directly into the data array at the start of the program.

1) Dimension the DOUBLE array of 512 bytes.

2) Zero out the array.

3) On an even index (real portion of the complex number), enter the frequency spectrum points into the array. The values of these data entries should be between -1.0 and +1.0. (Index for real data points start at a value of 2).

4) Call the FFT routine.

5) Scan the even elements of the result array and form a scaling factor for a maximum value of 128 and minimum value of -127.

6) Transfer the waveform data from the even bytes of the DOUBLE array to a 256 byte array of integers, multiplying by the scaling factor.

7) Call the built-in WAVE function in A/C Basic with INTEGER array as its input.

8) Call the built-in SOUND function in A/C Basic to hear the constructed waveform.

When the program is run, the screen will be cleared and two display boxes will be drawn. The top display box will contain a spectrum plot of the input data. This will show the user the frequency values and the relative amplitudes they have chosen.

You should note that if you enter a spectral component with a negative amplitude, the resulting waveform will differ from that when the spectral point had a positive amplitude.

This is because the frequency component is being added into the waveform calculations out of phase with the other entries. You should experiment with different amplitudes, but keep the frequency values the same polarity.

The second display box will contain the constructed waveform which will be scaled before displaying, so the largest amplitude in the waveform will be a full scale reading in the display box.

The next step in the program allows the user to listen to the constructed waveform. To do this, transfer the complex data array values to the integer array to be passed to the WAVE subroutine. Before transfer in the real portion of the data to the packed integer array, scan the array for the maximum value.

This maximum value will be used as a scaling factor during the data transfer, to prevent the data from falling out of the range of the tone generation system limits (-127 to +128).

The waveform information is transferred to channel number 2 in the Apple IIGS Hardware. It is placed in channel 2 so the default tone in channel 0 is preserved and both tones are directed to the same channel. For comparative purposes, the program will play the default tone (which is a single frequency sine wave) and then play the newly constructed tone.

Once you understand the function of the program, you can add file utilities to the program so that constructed tones can be saved. Loading waveforms from a file for use in this program or in another program will be much faster than generating them through the FFT algorithm each time.

Some example data points have been selected with known good results. These examples are in the top of the FTOW program. They have been commented out so that they do not interact with each other at all. Try each of the examples by deleting the comment in the first character of each line and run the program. Each time you move on to the next example, remember to comment out the current example data statements so they will not interact.

The second program WTOF will use the FFT algorithm to translate time domain information to frequency domain information. This function is known as spectrum analysis. There are many instruments in the engineering market today that will perform this function in real time. These instruments are manufactured by companies such as HP, Techtronics, IFR, and Easton. Prices range from \$10,000 to \$60,000. Although the Apple IIGS may not be as fast as or have all of the features of these dedicated devices, it will allow you to experiment in this area of electrical engineering.

As with the FTOW program, you will find a section of example data at the beginning of the main section of the program. To examine the results of the translation from the time domain to the frequency domain, you must uncomment the data entry statements and run the program.

When the program executes, the top display, as before, will show the input data graphically. The program will then calculate the frequency spectrum from the time domain information. The frequency spectrum will be displayed in the second graphics window.

One difference between the output of this program and the first program is the effect of aliasing. This is where frequency information displayed will seem to have a mirror image at about the halfway mark. This effect is present in the frequency-to-time conversion also, but is not seen because the waveform generated is symmetrical and periodic. Another name for this effect is the Nyquist frequency.

This article has been an introduction to Fourier Transforms and the conversion between the time and frequency domains. This transformation has been accomplished by the use of an FFT subroutine which was implemented in A/C Basic and Orca C.

Although the implementation is not fast enough for real time data analysis the same methods apply. To form a real time spectrum analyzer FFT routine and data zeroing routine should be implemented in assembler with care taken to make sure they execute as fast as possible.

1)

$$H(f) = \int_{-\infty}^{+\infty} h(t) e^{-j2\pi f t} dt$$

$$e^{-j0} = \cos(0) - j \sin(0)$$

$$H(f) = \int_{-\infty}^{+\infty} h(t) (\cos(2\pi f t) - j \sin(2\pi f t)) dt$$

$$H(f) = \frac{a_0}{2} + \sum_{n=1}^{n \rightarrow \infty} [a_n \cos(2\pi n f_0 t) + b_n \sin(2\pi n f_0 t)]$$