By Bryan Bergeron, *NUIN*
30 Gardner Road, Apt. 1G
Brookline, Massachusetts 02146

DIGITAL SIGNAL PROCESSING

Working in the frequency domain

Though most of us are comfortable working in the time domain, when it comes to digital signal processing (DSP), much of the work is done in the frequency domain. There are certain motivations for, and constraints associated with, performing digital signal processing in the frequency domain. I'll discuss these motivations and constraints, and review the Fourier Transform and its descendants.

Frequency domain processing

Like the oscilloscope and the spectrum analyzer, digital time domain and frequency domain signal processing take two very different perspectives of the same phenomena (see **Figure 1**). Time-domain waveforms viewed by an oscilloscope and frequency-domain spectra viewed through a spectrum analyzer are transforms of each other. One view doesn't carry any *more* data about a signal than the other. Rather, each provides a different way to think about and work with the same signal.

Frequency domain processing of a signal has two distinct advantages over processing in the time domain: tractability and efficiency. Often a problem that's virtually unsolvable in the time domain can be deciphered easily in the frequency domain. Consider the challenge of equipping a RADAR system with real-time pattern recognition capabilities. While this is a sizable engineering feat in the time-domain, it becomes a trivial problem in the frequency domain. In the frequency domain, a signal from an aircraft or other object of interest can be looked at independent of the size, orientation, or position of the object!

For cases that can be solved in either domain, working in the frequency domain is almost always simpler. For example, you can easily determine the frequency response of a microphone by sampling its output (using a white noise source) every few milliseconds and then manipulating the signal in the frequency domain. While you could obtain the same results working with a digital filter defined in the time domain, frequency domain work is more straightforward and conceptually clean. It is also more efficient computationally.

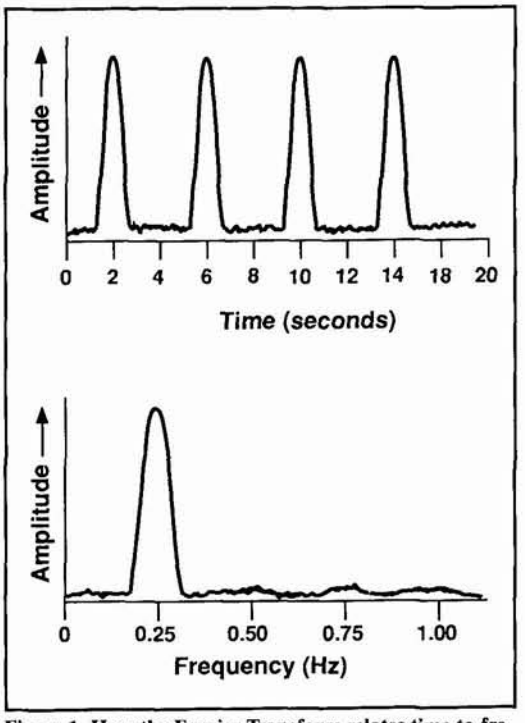

Figure 1. How the Fourier Transform relates time to frequency. In the time domain (top), a signal is composed of short bursts regularly spaced at one per every four seconds (0.25 Hz). When transferred via the Fourier Transform from the time domain to the frequency domain (bottom), the same signal appears as a response at 0.25 Hz.

Here's an illustration of the potential advantages of working in the frequency domain. Think about the DSP challenges associated with constructing adaptive filters[2]* and in filtering unwanted noise

*Adaptive filters are filters that cancel or minimize noise and interference by dynamically updating the filter coefficients to adapt to the characteristics of the interference.

from a signal. Adaptive filters are commonly implemented using digital technology because of the inherent stability and mathematical tractability of the algorithms used for the computation of the filter coefficients. Although the algorithms are straightforward when implemented in the time domain, the performance of these algorithms is typically less than that of equivalent frequency domain algorithms[3] The more computationally efficient frequency domain implementations of adaptive filters are often called frequency domain adaptive filters, or block adaptive filters[4]

Frequency domain work lends itself to the identification and elimination of noise and other artifacts, especially when these undesirable signals are significantly higher or lower in frequency than the desired signal. A 60-Hz noise in a communications signal (bandwidth 300 to 3000 Hz) or 2-kHz instrumentation noise superimposed on a 200-Hz signal are examples of undesirable signals. Think of the procedure for removing high or low frequency noise from a signal as a simple multiplication process in the frequency domain[5] For example, if the 60-Hz noise is distinct and separate from a 300 to 3000-Hz signal spectrum, then a simple rectangle function can zero all data values below 300 Hz. That is, data values between 300 and 3000 Hz are multiplied by 1; all other data values are multiplied by zero (see **Figure 2**).

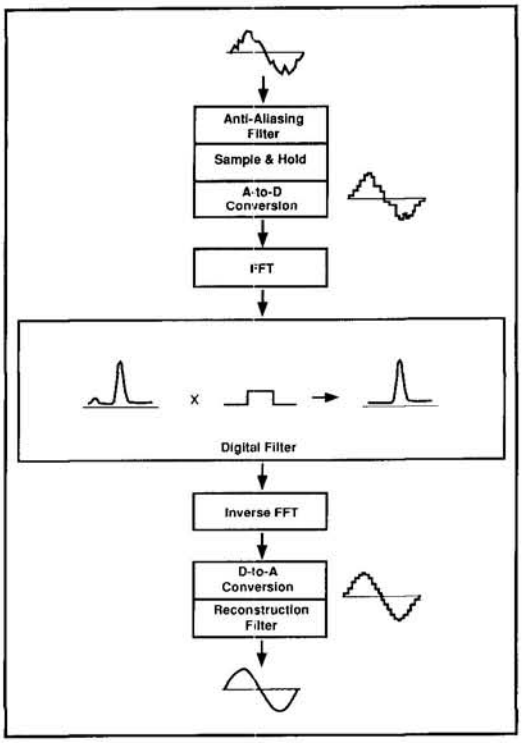

Figure 2. An example of Digital Signal Processing in the frequency domain.

Look at the top of **Figure 2**. It shows that the original noisy analog signal is first preprocessed by a high pass (anti-aliasing) filter. Next the digitizing hardware samples the analog signal and maps the values onto digital values at regular time intervals. A Fast Fourier Transform, (FFT — more on this later), or its equivalent, is then used to move the signal into the frequency domain. The digital filter shown in the middle of **Figure 2** (in this case, a simple rectangular function) removes the unwanted signal (represented by the small peak to the left of the main signal peak). Once the filter function has been applied, the signal is converted back to the time domain by performing an inverse FFT on the data. After digital to analog (D/A) conversion and filtering, the desired signal, free of noise (shown at the bottom of the figure), is available for futher processing or direct use.

If the signal data and the noise aren't clearly distinguishable (as is normally the case), you must select a judicious cutoff, based on the known characteristics of the unwanted noise and the desired signal. Also, the digital filter function you choose must gradually attenuate the unwanted spectra in the frequency domain; the spectra can't suddenly drop off to zero. Such a sudden cutoff would result in false accentuation of frequencies corresponding to the cutoff point in the frequency domain. This idea is developed in the discussion of windowing found in the section on frequency domain DSP considerations.

The Fourier Transform

You can't work effectively with DSP in the frequency domain without a good conceptual understanding of the Fourier Transform (named after the 18th century mathematician Jean Baptiste Joseph Fourier). By relating time to frequency, this transform is the basis of frequency domain processing as it is known today. For the mathematically inclined, the Fourier Transform is defined by the following equations:[6]

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(j\omega) e^{j\omega t} dt \qquad (1)$$

$$X(j\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \qquad (2)$$

where $j = \sqrt{-1}$ and $\omega = 2\pi f$. Notice that the transformation from the frequency domain to the time domain (**Equation 1**), and from the time domain to the frequency domain (**Equation 2**), integrate over the limits from $-\infty$ to $+\infty$. This results in the mathematically correct concept of negative

frequency, which has no basis in physical reality.

In practice, the complex exponential, $e \pm j\omega t$, is usually replaced with the trigonometric expression:

$$e\text{-}j\omega t = \cos \omega t \pm j \sin \omega t$$

Although mathematically elegant and conceptually beneficial, the basic Fourier Transform is of limited practical value when working with digital computers because it assumes that the data to be transformed are continuous. Actually, analog data are sampled and digitized into a machine-readable form at discrete intervals. To allow high speed computers to handle frequency-time domain transformation calculations, the Fourier Transform, which uses the infinitesimal dt, was modified into the Discrete Fourier Transform (DFT) which expects either a quantized continuous signal or a signal of limited duration. Mathematically, the DFT appears as:[6]

$$x_n = \sum_{k=0}^{N-1} X_k \exp\left[j \frac{2\pi nk}{N} \right] \quad (3)$$

$$X_n = \frac{1}{N} \sum_{k=0}^{N-1} x_k \exp\left[-j \frac{2\pi nk}{N} \right] \quad (4)$$

Notice that the integral has been replaced with summation over N discrete data points. The frequency to time transform in **Equation 3** also differs from the time to frequency transform in **Equation 4** as the result of a change in the sign of imaginary j and the scaling multiplier 1/N.

Creating a workable computer program to perform these calculations is relatively straightforward. However, because a merely workable program is seldom good enough for real applications, researchers have spent many years increasing the computational efficiency and memory requirements of DFT algorithms. There is now a wide variety of rapid and efficient methods for computing the DFT. The original Fast Fourier Transform (FFT) and the more recent Fast Hartley Transform (FHT) are two popular ones.

The Fast Fourier Transform

The Fast Fourier Transform may be the best known method for computing the DFT rapidly. The FFT takes advantage of the redundant calculations within the DFT – a primary reason for the increased speed of the FFT over the basic DFT. As its basic premise, the FFT and related algorithms sort data using a data-pairing permutation process (sometimes referred to as the "butterfly" because of the appearance of the associated data flow diagram) until data is separated into pairs. The Fourier transform calculation on these data pairs is rapid. It is computationally more expensive to compute a 32-point DFT than it is to compute 16 two-point DFTs.

A large part of the data-pairing permutation in the FFT algorithm is concerned with a bit reversal procedure which scrambles the order of the output data creating a mirror image of the input. The speed of this bit reversal (or reshuffling of data) defines, to a great degree, the efficiency of a given FFT algorithm. (In cases where execution speed is critical, this bit reversal can be accomplished in ASSEMBLER.) To increase the speed of the actual Fourier transform, you can substitute a trigonometric look-up table for the calculation of trigonometric functions supplied by the host language.

The FFT algorithm has a major restriction. For the data-pairing permutation to function properly, the number of discrete data input values (N) must be an integral power of 2 (2, 4, 8, 16, 32, 64, 128, and so on). Because the FFT expects 2^n data points, zero filling or padding is commonly used to increase the number of data points to the next higher power of 2; that is, from 45 to 64 or from 120 to 128 data points. Unfortunately zero filling can result in phantom responses. These phantoms can be reduced by windowing the data. This means you variably attenuate the first and last few input data values to reduce sharpness of the drop to the zero-padded area. Another problem associated with zero filling is related to the increased storage and computational requirements imposed by the added data, which add nothing to the information content of the signal.

It may seem that you have to go to a lot of trouble to achieve an increase in execution speed, but considerable time can be saved by substituting the FFT for the DFT. For example, the time required to compute a DFT is proportional to N^2, while the time required to compute the FFT is proportional to $N \times \text{Log}_2 N$ – thanks to the data-pairing permutation. For large data sets, this can amount to a significant savings in computer resources (see **Table 1**).

Figure 3 provides a BASIC implementation of an FFT subroutine (compatible with BASICA for the IBM-PC). It is based, in part, on a listing by Brook and Wynne[7] The subroutine assumes that the arrays for holding real data (AR) and imaginary data (AI) have been defined with dimension N. Changing the sign of ID from plus to minus allows the inverse function to be performed on the data array. For example, ID = +1

Samples (N)	DFT	FFT
8	64	24
16	256	64
32	1024	160
64	4096	384
128	16384	896
256	65536	2048
512	262144	4608
1024	1048576	10240

Table 1. A comparison of the number of computations involved in calculating the DFT and the FFT of a signal. Notice that as the sample size (N) increases, FFT superiority becomes more significant. That is, with a 1-K sample size (1024 elements), the DFT/FFT ratio is approximately 100:1.

for time to frequency transformation and ID = −1 for frequency to time transformation. Lines 100 to 130 of the subroutine perform the scaling function when the transformation is from the time to the frequency domain. Lines 140 to 230 perform the data pairing and lines 250 to 420 are responsible for the actual Fourier transform. As noted previously, the data-pairing permutation can be coded in ASSEMBLER to maximize computational efficiency. As you can see, this algorithm produces the same number of data output points as data input points.

```
100 IF ID > 0 THEN FOR J = 1 TO N
110       AR (J) = AR (J) / N
120       AI (J) = AI (J) / N
130 NEXT
140 NHLF = N/2 : NM1 = N-1 : J = 1
150 FOR L = 1 TO NM1
160       IF (L >= J) THEN 190
170       T = AR (J) : AR (J) = AR (L) : AR (L) = T
180       TX = AI (J) : AI (J) = AI (L) : AI (L) = TX
190       K = NHLF
200       IF (K >= J) THEN 230
210       J = J - K : K = K / 2
220       GOTO 200
230       J = J + K
240 NEXT L
250 FOR M1 = 1 TO M
260       UR = 1.0 : UI = 0.0
270       ME = 2^M1 : K = ME / 2
280       CON = PI / K
290       FOR J = 1 TO K
300             FOR L = J TO N STEP ME
310                   LPK = L + K
320                   TR = AR (LPK) * UR - AI (LPK) * UI
330                   TI = AR (LPK) * UI + AI (LPK) * UR
340                   AR (LPK) = AR (L) - TR
350                   AI (LPK) = AI (L) - TI
360                   AR (L) = AR (L) + TR
370                   AI (L) = AI (L) + TI
380             NEXT L
390             UR = COS (CON * J)
400             UI = - SIN (CON * J) * ID
410       NEXT J
420 NEXT M1
```

Figure 3. A simple FFT subroutine in BASIC, compatible with BASICA for the IBM-PC and clones. This code can be easily extended to include plotting functions to produce graphs like those shown in *Figures 4-7*. For time critical applications, lookup tables can be substituted for the SIN and COS functions supplied by your BASIC interpreter.

Notice also how the computationally expensive evaluation of the complex exponential has been replaced with an equivalent trigonometric expression (lines 390 to 400). Replacing the COS and SIN evaluations with a look-up table is an easy way to improve the performance of this subroutine significantly, without resorting to working in ASSEMBLER. The tradeoffs associated with the faster speeds provided by a look-up table include the cost of RAM required to hold the look-up table elements and decreased accuracy of the transform function (because the values returned for each SIN and COS evaluation are limited by the bit length and angle resolution of the table).

Because you can't fully appreciate the operation of the FFT algorithm without illustration, I've included graphs of the actual input and output data from an FFT program similar to the one in **Figure 3**, implemented in MacForth on the Apple Macintosh (see **Figures 4** through **7**). Notice that, for each figure, there are three values plotted for the signal in the frequency domain. The real and imaginary components correspond to the values in the AR and AI data arrays used **Figure 3**. The magnitude component, which corresponds to the familiar power spectra of the signal, is proportional to the absolute value of the square of the real and imaginary components.

The Fast Hartley Transform

The FFT has been a dependable workhorse for Frequency Domain Digital Signal Processing (FDDSP) since the mid-sixties. However, the demands of modern DSP applications and the move from mainframe systems to microcomputers have spurred the development of more efficient algorithms. One of the more notable descendants of the FFT is the Fast Hartley Transform (FHT), based on the continuous transform introduced by R.V.L. Hartley in 1942[8] Like the FFT, the FHT maps a signal from the time domain into the frequency domain (and vice versa). However, where the FFT maps a real function of time into a complex function of frequency, the FHT maps a real function of time into a real function of frequency. Mathematically, the FHT appears as:

$$X(t) = \sum_{t=0}^{N-1} H(f)\, cas\left[\frac{2\pi ft}{N}\right] \qquad (5)$$

$$H(f) = \frac{1}{N}\sum_{t=0}^{N-1} F(t) cas\left[\frac{2\pi ft}{N}\right] \qquad (6)$$

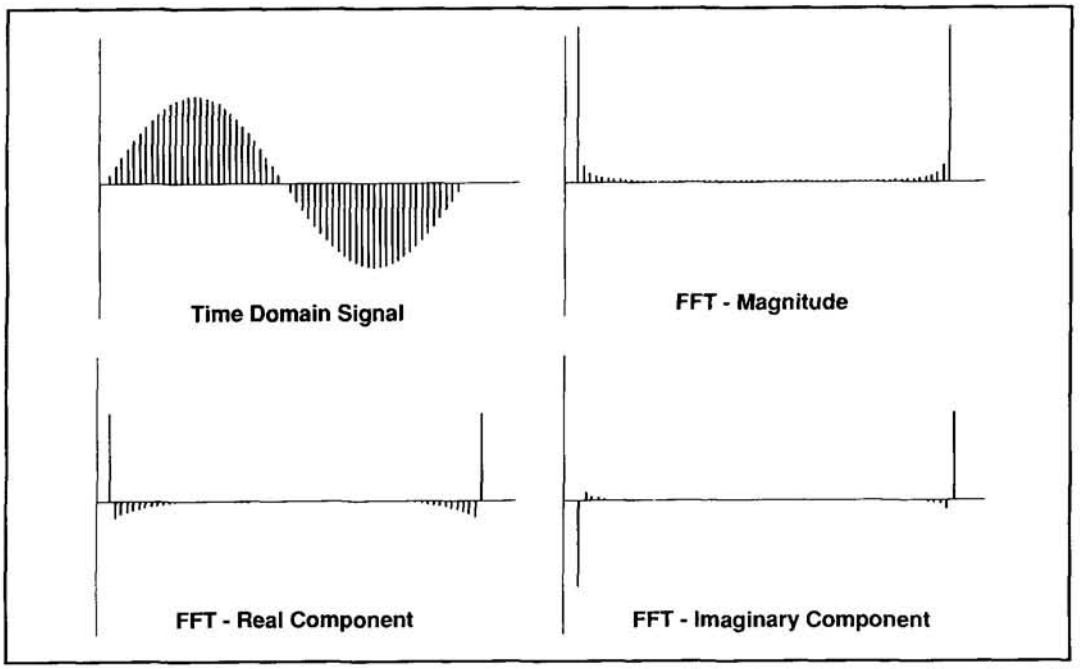

Figure 4. A time domain sinusoidal signal sample with an integer number of complete cycles, and nearly equal amplitude values at either end of the sample (top, left). The magnitude or power spectra (top, right), as well as the real (bottom, left) and imaginary (bottom, right) components of the FFT are also illustrated. Note the relative purity of the magnitude plot. Most of the signal energy is concentrated in a few spectral lines. Note also that in this figure, as well as the following three, the FFT plots contain both positive (left side of each frequency domain plot) and negative (right side of each frequency domain plot) frequency components. For practical purposes, you can ignore the right half of each plot.

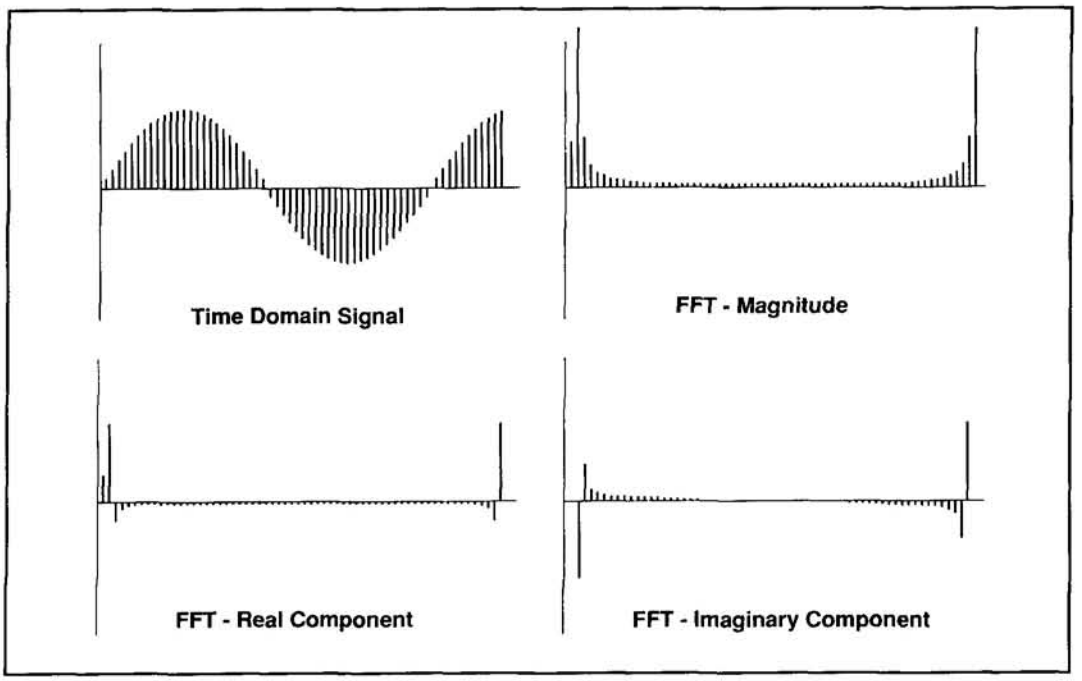

Figure 5. A sinusoidal signal sample in the time domain in which the sampling interval and frequency are such that irregular points in the waveform have been captured (top, left). Even though this signal is of the same amplitude and purity as the sinusoidal signal in *Figure 4*, notice the relative impurity of the magnitude plot; i.e., there is now a considerable amount of energy distributed throughout the frequency domain plot. The solution is to either employ a windowing function, or to adjust the sampling interval so an integer number of complete cycles are captured.

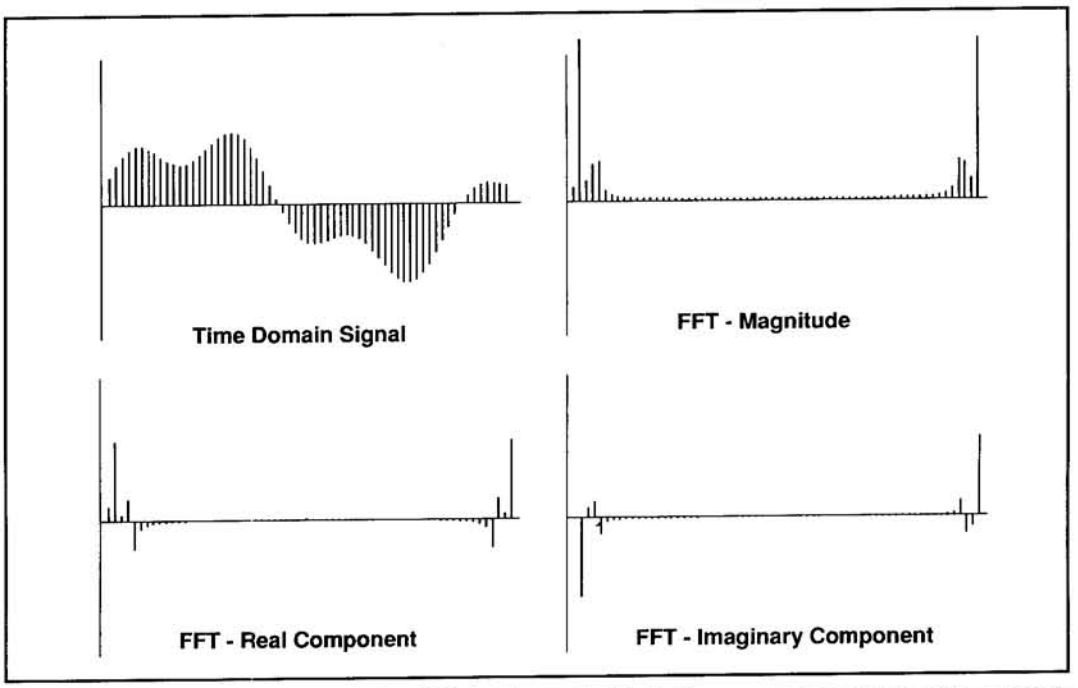

Figure 6. In this example, there are two sinusoidal signals, one double the frequency and one quarter of the amplitude of the other (top, left). Notice the extra responses in the frequency domain in the magnitude (top, right), real (bottom, left), and imaginary (bottom, right) components.

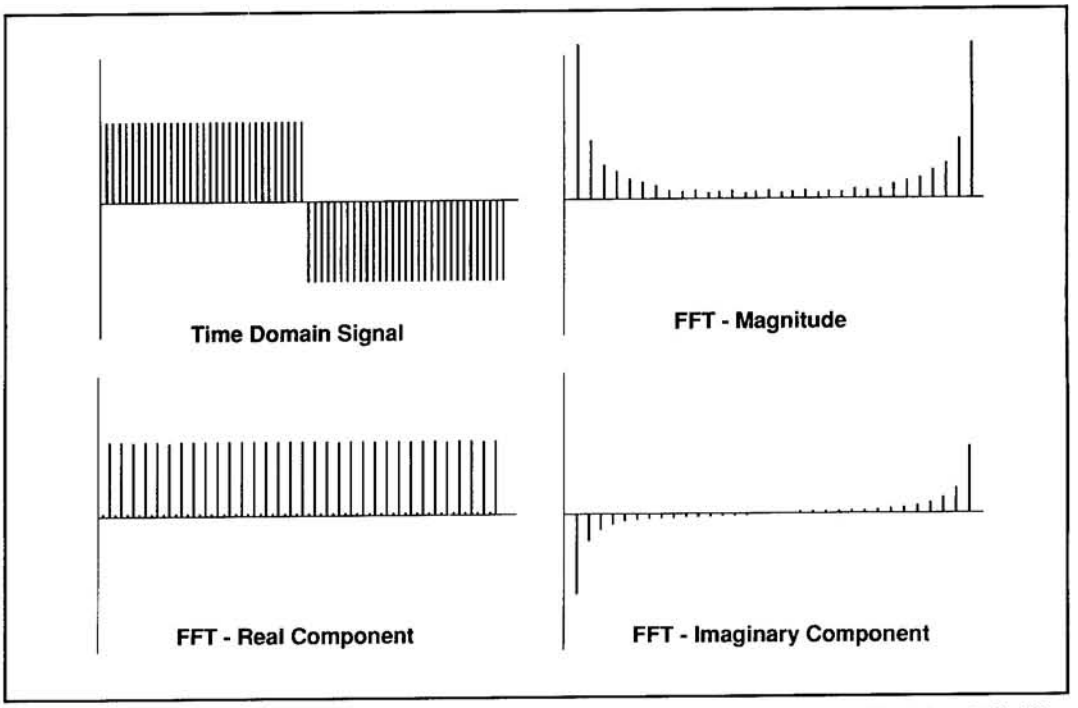

Figure 7. The time domain signal in this example is one complete cycle of a simple square-wave signal (top, left). When compared with a similar sinusoidal signal in *Figure 4*, you'll note there is a relatively large amount of spectral energy distributed above the main signal peak (top, right). This is to be expected, because square waves are composed of a large number of harmonically related sine waves.

where **cas**, in both the frequency to time (**Equation 5**) and time to frequency transform (**Equation 6**), is equivalent to the cosine and sine of the expression in the brackets. That is, cas (β) = cosine (β) + sine (β). Notice the similarity of **Equations 5** and **6** with those that describe the FFT. The main difference is the substitution of the real function cas($2\pi ft$) for the complex exponential term in the FFT.

Computationally, working with real numbers (instead of real and imaginary numbers) is very advantageous. For each arithmetic operation required to compute the FHT, six operations are required to compute the FFT. Four operations are necessary for each complex multiplication or division, and two operations are required for complex addition or subtraction[8] Compared with the FFT, the FHT has associated memory savings in addition to computational savings. FFT calcuation requires the use of complex numbers. Because complex numbers are composed of two distinct parts (real and imaginary), they require twice as much computer storage space as real numbers.

Consequently, the FHT requires only about half as much working memory as the FFT, because complex data arrays require twice as much space as real data arrays[9] The FHT is an especially attractive alternative to the FFT when you have a large volume of data to work with, as in digital image processing. The FHT, while more efficient than the FFT, has considerably more code associated with its implementation than the FFT. If you are interested in coding examples of the FHT, see the excellent text by Bracewell![10]

Frequency domain DSP considerations

The FFT and its derivatives constitute the core software tools used for virtually all FDDSP applications. Like other software tools, they can easily be misused if the operator doesn't understand the underlying assumptions of their design. To reap the greatest benefit from any FDDSP system, you have to understand the characteristics of the signal to be processed and also be aware of the capabilities and limitations of the software and hardware components of your DSP system. Some of the more pertinent aspects of DSP in the frequency domain are outlined in more detail in the sections that follow.

Windowing

In most FDDSP systems, signal samples are collected into blocks of length 2^n and then processed by some type of FFT algorithm. In FDDSP it is important to assume that each successive discrete sample represents part of a continuous signal which repeats indefinitely what is in the sample; that is, the signal is periodic. If the sampling frequency and sampling interval are selected so that complete integer number of cycles are captured in each sample (as in **Figure 4**), then the sample boundaries will be of nearly equal amplitude, and the transition from one sample to the next will be smooth. If, on the other hand, the sampling frequency and interval are such that irregular points in the waveform cycle are captured, there will be high frequency artifacts in the transformed data (as in **Figure 5**). The effect will be most pronounced when the sample contains an exactly odd number of half cycles, because the discontinuity at sample boundaries will be maximum![11]

It's obvious that one condition under which the FFT works best is when the data to be transformed smoothly approaches 0 at both ends of its range (**Figure 4**). If, however, the actual data do not conform to the ideal, you can force them into an acceptable form by multiplying them by a *window function* before calculating the transform. A window function effectively multiplies data values near the center of the sample by unity, data near the ends of the sample by 0, and data between the center and ends by some intermediate value. The nature of these intermediate multiplicative factors defines the nature of the window.

The triangular window is a simple and fast window function, where the multiplication factor decreases linearly and symmetrically toward both ends of the sample. Another popular window function is the Hanning function (see **Figure 8**), which provides better artifact reduction, at the expense of computational efficiency. This function is defined as:

$$w\{i\} = 0.5(1-(\cos(2\pi/N))$$

where i = the sample point number and N = the total number of samples![12] For more information on window functions and their uses, see the work by Press![13]

Sampling jitter

It is a basic (but commonly overlooked) assumption of FFT work that the signal is sampled at *regular* time intervals — every 10 milliseconds, for example. Sampling jitter is the distortion of the sampled signal due to variations in the sampling interval. This jitter, which increases the noise floor of a signal, affects higher frequencies more than lower ones. Sampling jitter is most common in systems that rely on software triggering

of the analog to digital (A/D) conversion process, rather than the more stable and reliable hardware triggering methods.[11] Software-based triggering systems that vary only a few microseconds between samples can add significantly to the noise floor of a system.

Quantization error

Like sampling jitter, quantization error raises the noise floor in a FDDSP system. This is commonly referred to as quantization noise. Like sampling jitter, quantization error is a function of how the data is acquired and processed, before the actual digital signal processing. Quantization error results when the actual, instantaneous value of a continuous signal is mapped onto the nearest integer value supported by the A/D conversion hardware. For instance, quantization error can occur when both 12.157 and 12.234-volt signals are mapped to 12.2 volts by an A/D converter. This error can be minimized by using an A/D converter with greater resolution. For example, you could use a 12-bit digitizer in place of an eight-bit unit. A compromise must always be made between quantization error (noise) and the increased cost, speed penalty, memory requirements, and computational load imposed by a higher resolution A/D converter.

Sampling frequency

Although extrapolation procedures have been developed for the FFT to accurately determine the frequency of signals higher than the Nyquist frequency,[14] it's generally accepted that the sampling frequency must be at least twice the frequency of the signal to be sampled. This means there should be at least two samples per cycle of the highest frequency contained in the signal. It's often

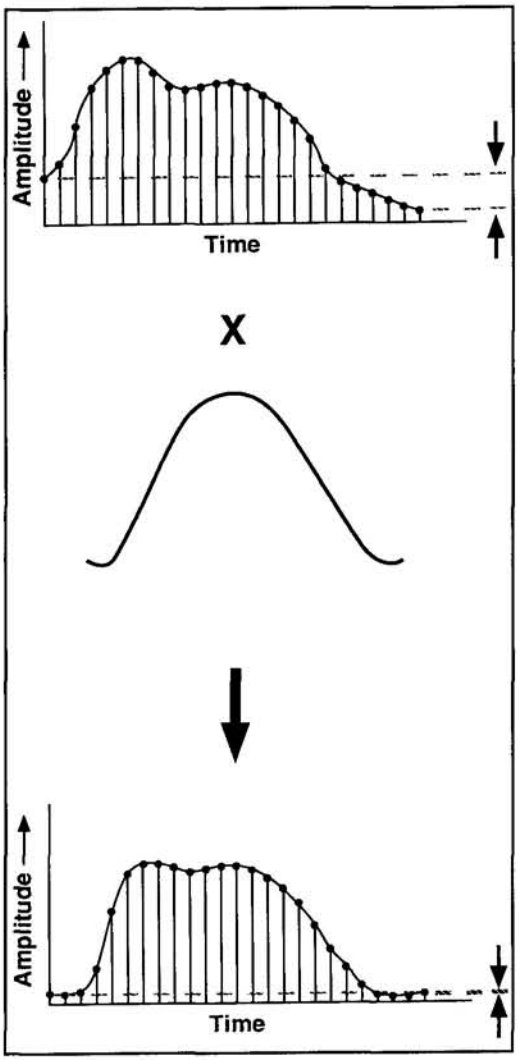

Figure 8. An example of how windowing, when applied to time domain signals prior to FFT processing, can help minimize artifacts due to discontinuities at the ends of the sample. In this example, the original signal (top) is preprocessed with the Hanning function (depicted graphically, center), to equalize signal amplitudes at both ends of the sample (bottom). Notice that one side effect of windowing is that data points at both ends of the sample are thrown away. The consequences of this data loss are described in the text.

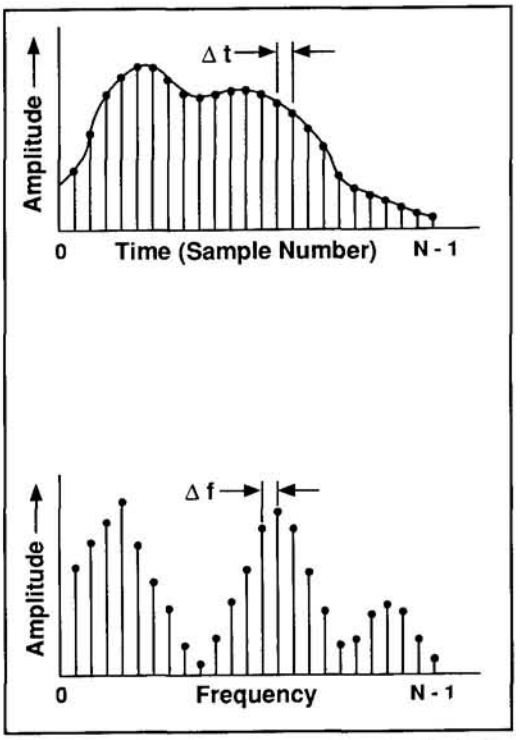

Figure 9. Spectral resolution versus sampling time for the Discrete Fourier Transform (DFT). The spectral resolution (Δf) is equal to $(N \times \Delta t)^{-1}$, where N is the number of samples taken of the signal in the time domain, and Δt equal to the sampling time in seconds. For example, if the sampling time in the time domain (top) is 10 ms, and the number of samples is 100, then the spectral resolution (bottom) will be $(100 \times 0.010)^{-1}$, or 1 Hz.

necessary to use a low-pass filter front end to any A/D converter to assure that only frequencies which can be handled adequately are passed on to the converter. Otherwise, aliasing (the folding down of undersampled signals) can result.

Resolution

The spectral resolution of an FDDSP system is closely related to the sampling frequency, sampling jitter, and windowing. In general, the frequency resolution is about equal to the reciprocal of the sampling interval in the time domain (see **Figure 9**). The smaller the sampling interval, the higher the frequency resolution. Sampling jitter effectively decreases the resolution of a system, because the certainty of the sampling interval, and therefore the frequency interval, is diminished. The noise associated with sampling jitter also diminishes the effective resolution of system, especially higher frequency signals.

Windowing also decreases the effective spectral resolution of a system. While reducing the number of possible artifacts, windowing throws out or gives less weight to the sampled data at both ends. As **Figure 9** illustrates, the FFT and its descendants produce one output data point for each input data point. Throwing out data in the time domain, in effect, spreads the frequency domain signal by a proportional amount. For example, by using windowing to decrease the effective number of input data points by 10 percent, you decrease the frequency resolution by approximately 10 percent.

Aperture time

Aperture time, like quantization error, is largely a function of the A/D hardware used in signal acquisition. The aperture time of an A/D converter – the time during which an analog signal is actually sampled before being digitized – can be likened to the shutter speed of a camera. When the camera shutter is open, light falls on the film emulsion exposing silver halide crystals to light energy. For a given shutter speed, slowly moving (low frequency) objects may be exposed clearly and accurately, while very fast objects (high frequency) might appear as blurs on the developed film. In photography, the solution is to use a faster shutter speed – assuming the film has enough latitude to work at the higher speed. It may be necessary to use a faster film with less resolution to capture a clear image of the faster objects.

The photograph analogy is useful if you think of the A/D resolution as the film resolution, the A/D conversion time as the film speed, and the aperture time as the shutter speed. High frequency signals can be digitized accurately only with a relatively short aperture time. But to use the short aperture time, the sample-and-hold circuit within the A/D conversion hardware must be capable of acquiring the signal in a relatively brief period of time. Of course, the A/D conversion hardware must also be capable of digitizing the signal before the next sample time. A high resolution A/D converter, like a 32-bit system (the photographic equivalent of low speed, high resolution film), will generally support a lower maximum sampling frequency than a low resolution converter, like an eight-bit system (high speed, low resolution film), assuming the converters are in the same price/performance range.

For many DSP applications, working in frequency domain is not only more efficient than working in the time domain, but the only means of arriving at a solution to a particular problem.

Summary

For many DSP applications, working in the frequency domain is not only more efficient than working in the time domain, but the only means of arriving at a solution to a particular problem. The Fourier Transform and its more computer compatible descendants, including the FFT and FHT, serve as the basis for the vast majority of operations in the frequency domain. While powerful algorithms for the FFT and FHT are easy to implement on desktop computer platforms, these frequency domain tools must be used with caution. Intelligent use requires knowledge of the signal to be processed and the features and limitations of the available DSP hardware and software.

With the rapid evolution and introduction of inexpensive DSP software environments, knowledge of the inner workings of both time and frequency domain signal processing is becoming a necessity. There are electronic circuit design prototyping techniques available today that place powerful DSP tools, like the FFT, in the hands of anyone with a personal computer (see **Figure 10**). To use these tools effectively, you must understand the assumptions and tradeoffs made by the software system designer. For instance, if you use a look-up table to increase the computational efficiency of the supplied FFT algorithm, what effect does this have on the accuracy of the trans-

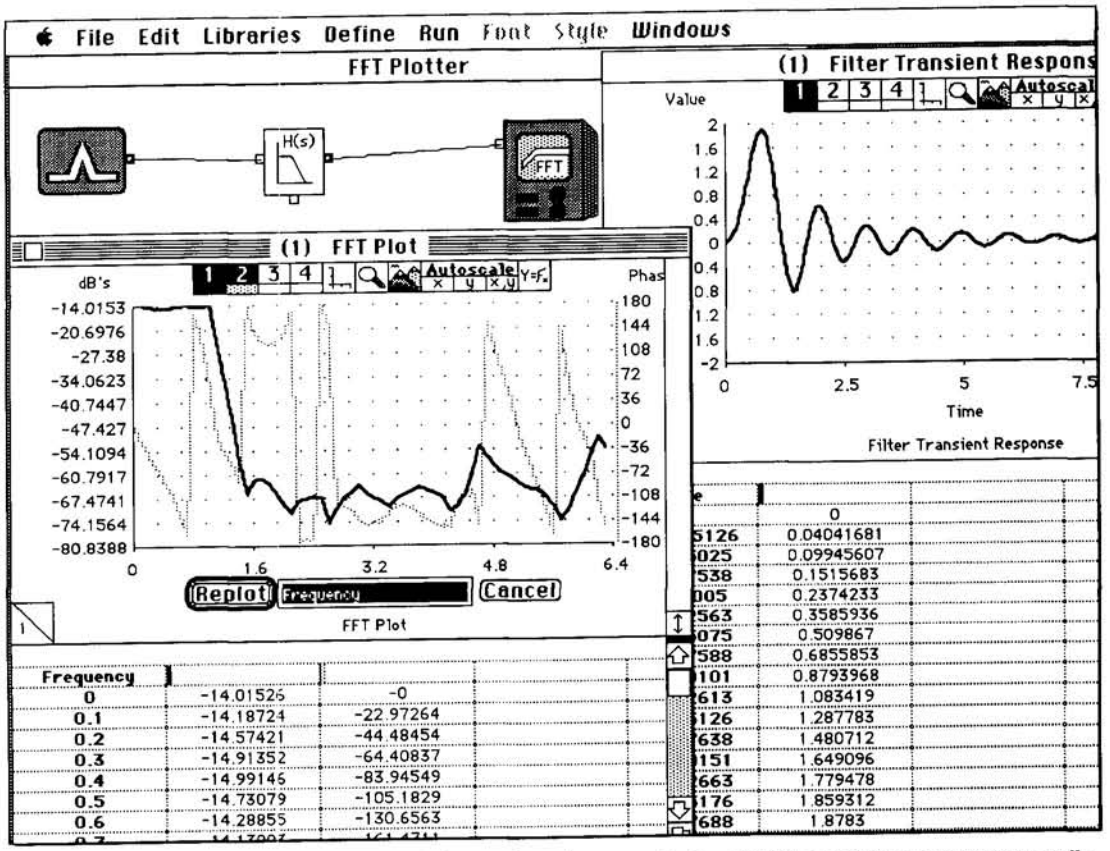

Figure 10. An example of the many, increasingly popular microcomputer-based DSP prototyping environments available to engineers. In this Apple Macintosh program (Extend™, from Imagine That!), you see a simple impulse function, followed by a low-pass filter and an FFT plotter (top, left panel). The icons represent code modules, and the lines connecting them represent data-flow paths. The time domain plot appears in the right hand panel, partially obscured by the FFT plot to the left and center of the display. Tools like this let engineers to design and debug complex DSP systems in hours instead of weeks.

formed waveform? A knowledgeable user is a powerful user.

In the next part of this series, I'll discuss artificial intelligence (AI) techniques that have been applied to digital signal processing in both the time and frequency domains. ■

References

1. D. Brown and S. Silverman, "Flexible High Speed Image Processing with Non-von Neumann Design," *Computer Technology Review*, 1986, 8(16): pages 91-94.
2. S. Haykin, *Introduction to Adaptive Filters*, Macmillan, New York, 1984.
3. S.D. Stearns, "Fundamentals of Adaptive Signal Processing," *Advanced Topics in Signal Processing*, Lim and Oppenheim, editors, Prentice Hall, Englewood Cliffs, New Jersey, 1988.
4. M.J. Dentino, J. McCool, and B. Widrow, "Adaptive Filtering in the Frequency Domain," *Procedure IEEE*, 1978, 66(Dec): pages 1658-1659.
5. E.E. Aubanel and K.B. Oldham, "Fourier Smoothing Without the Fast Fourier Transform," *Byte*, February 1985, pages 207-218.
6. D. Brook and R.J. Wynne, "Frequency Characterization," *Signal Processing: Principles and Applications*, Edward Arnold, London, 1988.
7. D. Brook and R.J. Wynne, *Signal Processing: Principles and Applications*, Edward Arnold, London, 1988.
8. M.A. O'Neil, "Faster Than Fast Fourier," *Byte*, April 1988, pages 293-300.
9. R.N. Bracewell, *The Fourier Transform and Its Applications*, McGraw-Hill Book Company, New York, 1986.
10. R.N. Bracewell, *The Hartley Transform*, Oxford University Press, Oxford, 1986.
11. A. Sewards, "Introduction to DSP," *Electronics and Wireless World*, 1988, 94(1630): pages 741-746.
12. H.J. Hutchings, "Interfacing and Signal Processing with C," *Electronics and Wireless World*, 1988, 94(1631): pages 948-956.
13. W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vettering, "Fourier Transform Spectral Methods," *Numerical Recipes in C*, Press Syndicate of the University of Cambridge, Cambridge, 1988.
14. S.E. Georgeoura, "Fast Fourier Transforms of Sampled Waveforms," *Electronics and Wireless World*, 1988, 94(1633): pages 1122-1126.

Bibliography

1. P. Sewla, "Building a Digital Filter: FIR Filter Features Guarantee Phase Linearity," *Ham Radio*, April 1989, pages 9-17.
2. B.P. Bergeron, NU1N, "Digital Signal Processing: Part 1—The Fundamentals," *Ham Radio*, April 1990.