Hideo Okawara's Mixed Signal Lecture Series

DSP-Based Testing – Fundamentals 13 Inverse FFT

Verigy Japan

May 2009

Preface to the Series

ADC and DAC are the most typical mixed signal devices. In mixed signal testing, analog stimulus signal is generated by an arbitrary waveform generator (AWG) which employs a D/A converter inside, and analog signal is measured by a digitizer or a sampler which employs an A/D converter inside. The stimulus signal is created with mathematical method, and the measured signal is processed with mathematical method, extracting various parameters. It is based on digital signal processing (DSP) so that our test methodologies are often called DSP-based testing.

Test/application engineers in the mixed signal field should have thorough knowledge about DSP-based testing. FFT (Fast Fourier Transform) is the most powerful tool here. This corner will deliver a series of fundamental knowledge of DSP-based testing, especially FFT and its related topics. It will help test/application engineers comprehend what the DSP-based testing is and assorted techniques.

Editor's Note

For other articles in this series, please visit the Verigy web site at www.verigy.com/go/gosemi.

Inverse FFT

In mixed signal testers, digitizers, samplers and AD converters capture waveforms which are time domain data. Discrete Fourier transform (DFT) processes the waveform data into frequency spectrum which is frequency domain data. Frequency spectrum is used to parameterize and analyze signal amplitude, distortion, noise, frequency response, and so on. In mixed signal tests, more than 99% of AC parameter analysis is done by the frequency spectrum. Therefore FFT that is the fast version of DFT is the most useful and powerful tool in mixed signal tests. DFT and FFT were discussed in the previous newsletter articles. Inverse FFT or IFFT is the opposite operation against FFT. IFFT processes frequency domain spectrum into time domain waveform. IFFT is not so popular compared to FFT, however, it can be utilized to manipulate waveform data

such as filtering, compensation, modulation and so forth. There are some points to get IFFT performed successfully. In this article, you will have knowledge of performing IFFT correctly.

Fourier Transform Pair

First off, let's look at the nature of Fourier transform pair. Figure 1 (e) depicts 16-point time domain signals including DC and three different frequency AC signals. Figure 1 (a) is DC 1V. (b) is 3-cycle cosine waveform. (c) is 5-cycle sine waveform. (d) is 7-cycle composite cosine and sine waveform. The waveform (d) can be described as an equation as follows;

$$\begin{aligned}\cos\left(2\pi\cdot\frac{7}{16}k+\frac{\pi}{4}\right) &= \cos\left(2\pi\cdot\frac{7}{16}k\right)\cos\left(\frac{\pi}{4}\right)-\sin\left(2\pi\cdot\frac{7}{16}k\right)\sin\left(\frac{\pi}{4}\right) \\ &= \frac{1}{\sqrt{2}}*\cos\left(2\pi\cdot\frac{7}{16}k\right)-\frac{1}{\sqrt{2}}*\sin\left(2\pi\cdot\frac{7}{16}k\right)\end{aligned} \quad \text{.........................(1)}$$

It contains cosine and sine waveforms with equal amplitudes. Finally the waveform (e) is the summation of (a), (b), (c) and (d) so that it is the multi-tone signal created by the source code in List 1. The waveform (e) is a real number data array.

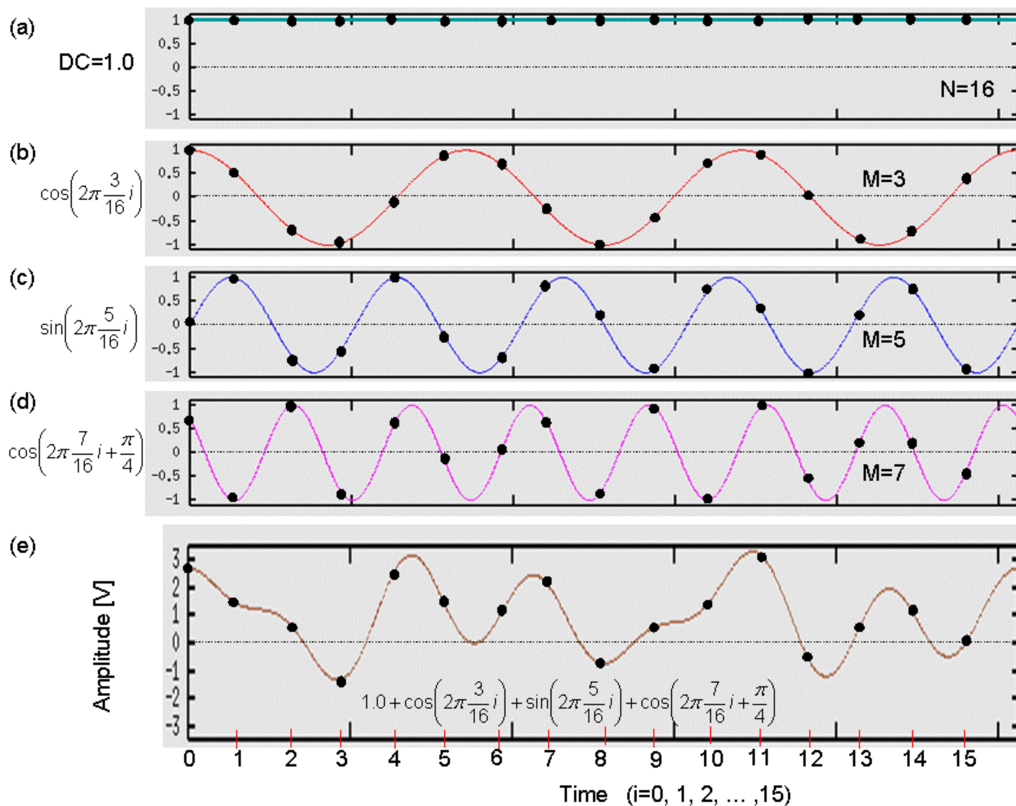

Figure 1: Time Domain Waveform (Multi-tone)

```
 1:   INT              i,M,N;
 2:   DOUBLE          dP;
 3:   ARRAY_D         dWave;
 4:
 5:   N=16;                             // Number of points
 6:   dWave.resize(N);
 7:   for (i=0;i<N;i++) dWave[i]=1.0;   // DC
 8:   M=3;
 9:   dP=2.0*M_PI*M/N;             // + (3-cycle cosine)
10:   for (i=0;i<N;i++) dWave[i]=dWave[i]+cos(dP*i);
```

```
11:    M=5;
12:    dP=2.0*M_PI*M/N;          // + (5-cycle sine)
13:    for (i=0;i<N;i++) dWave[i]=dWave[i]+sin(dP*i);
14:    M=7;
15:    dP=2.0*M_PI*M/N;          // + (7-cycle cosine & sine)
16:    for (i=0;i<N;i++) dWave[i]=dWave[i]+cos(dP*i+M_PI/4);
17:
```

List 1: Waveform Generation (Real Number Array)

Let's see how a cosine signal is displayed in the frequency domain and how a sine signal is displayed in the frequency domain. A cosine signal may be noted as an even function, and a sine signal may be noted as an odd function. We know imaginary numbers in mathematics. But the measured waveform is always real number data. In our daily measurement jobs, test signals are captured as real numbers. Let's perform FFT to the real number data array (e) to see the frequency spectrum. The code is as follows.

```
18:    INT              Nsp;
19:    ARRAY_D          dSp;
20:    ARRAY_COMPLEX   CSp1;
21:
22:    DSP_FFT(dWave,CSp1,RECT);
23:    Nsp=CSp1.size();
24:
```

List 2: FFT of Real Input Data Array

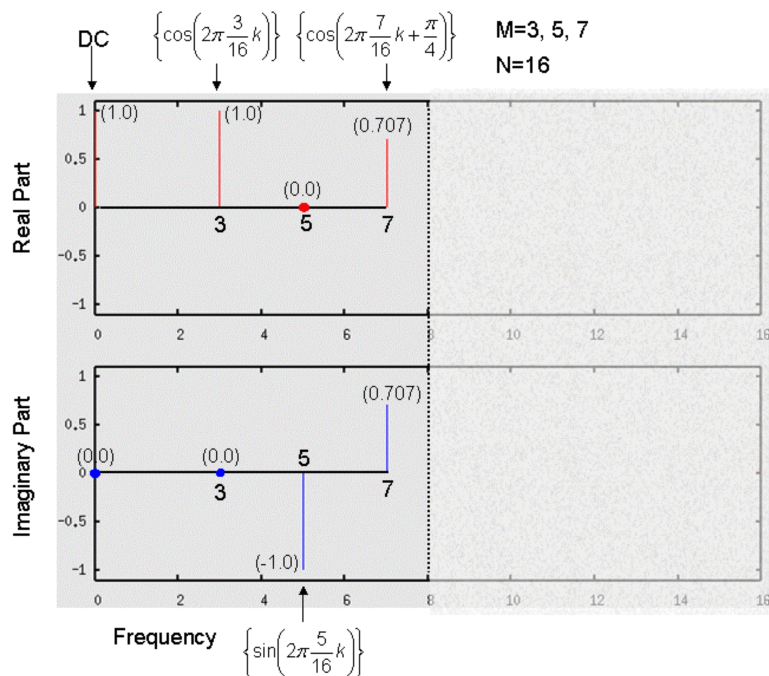

Figure 2: Frequency Domain Spectrum (Half Page Spectrum)

FFT performs at Line 22 in List 2. The input waveform is a real data array "dWave." The API at Line 22 derives a complex number array "CSp1" displaying in Figure 2. The number of spectrum *Nsp* is *N*/2 so that the frequency element index is settled from 0 to 7 here. At Frequency 0, you can see the spectrum (1.0 + j·0.0), which shows DC 1V. You can see the spectrum (1.0 + j·0.0) at Frequency 3, the spectrum (0.0 – j·1.0) at Frequency 5, and the spectrum (0.707 + j· 0.707) at Frequency 7. In this operation, you should understand as follows.

DC is the real part at Frequency 0. The imaginary part is 0.

"cosine" signal appears in the real part. The imaginary part is 0.

"sine" signal appears in the imaginary part. The real part is 0.

The spectrum at Frequency 7 is consistent of the rules 2 and 3. Frequency 7 is the combination of cosine and sine so that the cosine component gives (0.707 + j·0.0) and the sine component gives (0.0 –j·(-0.707)). Consequently the summation of the two complex numbers becomes (0.707 +j·0.707), which is true at Frequency 7 in Figure 2.

In general the FFT is designed to deal with complex input data and complex output data. Now let's see how the FFT performs with complex number input waveform array "CWave". The real number waveform data is expressed as complex numbers formally. The difference is the type of the variable. The real number waveform "dWave" fills in the real component of "CWave", and the imaginary component of "CWave" is filled up with zero perfunctory. The processing is listed as follows.

```
25:    ARRAY_COMPLEX CWave,CSp2;
26:
27:    CWave.resize(N);
28:    for (i=0;i<N;i++) {
29:       CWave[i].real()=dWave[i]; // Real number waveform
30:       CWave[i].imag()=0.0;       // Imaginary data is zero.
31:    }
32:
33:    DSP_FFT(CWave,CSp2,RECT);
34:
```

List 3: FFT with Complex Input Array (Zero Imaginary)

In Lines 27 to 31, the real number waveform data is copied into the real part of the complex number array, and the imaginary part is filled up with zero. FFT performs at Line 33. Now both input waveform and the output spectrum are expressed with complex numbers. It is mathematically elegant. Figure 3 shows the FFT result.

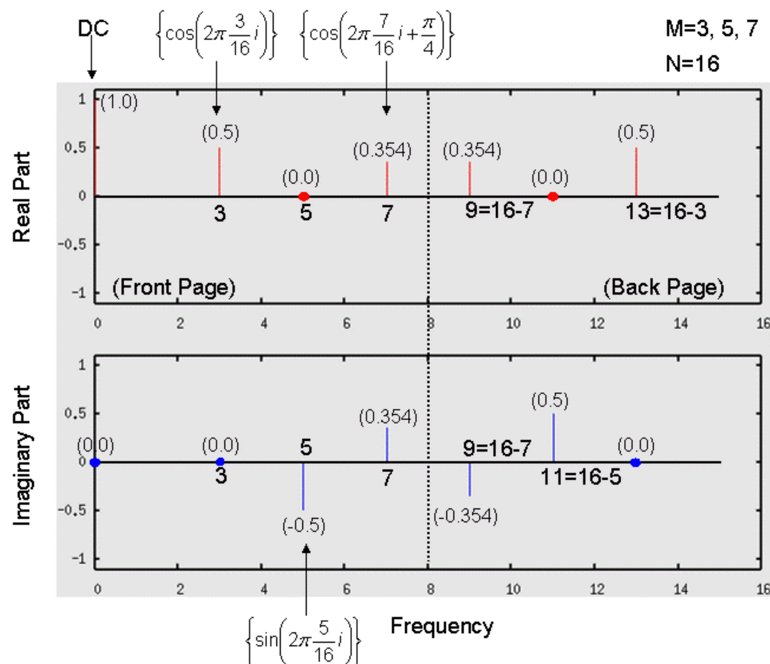

Figure 3: Frequency Domain Spectrum (Full Pages Spectrum)

The size of CSp2[] is double of the size of CSp1[], and equal to the number of input waveform elements *N*. The DC is still (1+j·0); however, AC signal lengths are slashed half. In exchange for it, the left half plane of the real part looks line-symmetrically copied to the right half plane, while the left half plane of the imaginary part looks point-symmetrically copied to the right half plane.

In other words, the right half plane is complex-conjugate-symmetrical to the left half plane. The reason that each length of the vectors is halved is based on the equation as follows.

$$\cos\theta = \frac{1}{2}\left(e^{j\theta} + e^{-j\theta}\right)$$
$$\sin\theta = \frac{1}{2j}\left(e^{j\theta} - e^{-j\theta}\right) \quad \cdots (2)$$

$e^{j\theta}$ and $e^{-j\theta}$ correspond to the left and right planes in Figure 3.

Let's summarize the relationship between the input waveform and the output spectrum. See Figure 4.

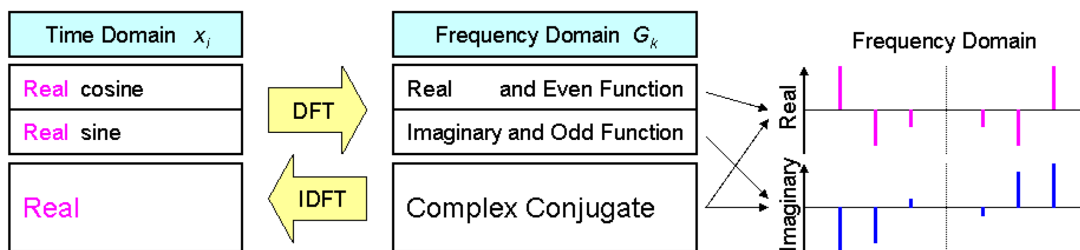

Figure 4: Time Domain vs. Frequency Domain

Real number waveform makes complex conjugate spectrum. This is the point.

Discrete IDFT

For the input complex data series $\{x_i\}$, the k-th element of the discrete Fourier transform G_k is expressed as follows;

$$G_k = \frac{1}{N}\sum_{i=0}^{N-1} x_i e^{-j\frac{2\pi ki}{N}} = \frac{1}{N}\sum_{i=0}^{N-1}\left(x_i \cdot \cos\frac{2\pi ki}{N} - j \cdot x_i \cdot \sin\frac{2\pi ki}{N}\right) \quad \cdots (3)$$

where N is the number of data points, and k=0, 1, 2, ..., N-1. Waveform data can be converted into frequency spectrum data with Equation (3).

The inverse DFT is defined as very close to Equation (3) as follows;

$$x_i = \sum_{k=0}^{N-1} G_k e^{j\frac{2\pi ki}{N}} = \sum_{k=0}^{N-1}\left(G_k \cdot \cos\frac{2\pi ki}{N} + j \cdot G_k \cdot \sin\frac{2\pi ki}{N}\right) \quad \cdots (4)$$

where i=0, 1, 2, ..., N-1. The difference is the sign of the imaginary part and the scaling factor $1/N$. Equations (3) and (4) may be called discrete Fourier transform pair, which should be applied to complex input data and complex output data.

IDFT Program Code

According to Equation (4), you can create your own IDFT program as follows.

```
A01: ////////////////////////////////////////////////////
A02:  COMPLEX  Cmult(COMPLEX  CX, COMPLEX  CY)
A03:  {                      // Complex Numbers Multiplication
A04:      COMPLEX  CZ;
A05:      CZ.real()=CX.real()*CY.real()-CX.imag()*CY.imag();
A06:      CZ.imag()=CX.real()*CY.imag()+CX.imag()*CY.real();
A07:      return (CZ);
A08:  }
```

```
A09:  ////////////////////////////////////////////////////
A10:  void IDFT(
A11:      ARRAY_COMPLEX & CSp,     //  Input Spectrum Array
A12:      ARRAY_COMPLEX & CWave    // Output Waveform Array
A13:  )
A14:  {
A15:      INT      i,j,Ndata;
A16:      DOUBLE  dQ,dAi,dBi,dP;
A17:      COMPLEX CX,CY;
A18:
A19:      Ndata=CSp.size();
A20:      CWave.resize(Ndata);
A21:      dP=2.0*M_PI/Ndata;
A22:      for (i=0;i<Ndata;i++) {
A23:         dAi=0.0;
A24:         dBi=0.0;
A25:         for (j=0;j<Ndata;j++) {
A26:            dQ=(DOUBLE)i*(DOUBLE)j*dP;
A27:            CX.real()=cos(dQ);
A28:            CX.imag()=sin(dQ);
A29:            CY=Cmult(CSp[j],CX);
A30:            dAi=dAi+CY.real();
A31:            dBi=dBi+CY.imag();
A32:         }
A33:         CWave[i].real()=dAi;
A34:         CWave[i].imag()=dBi;
A35:      }
A36:  }
A37:
```

List 4: IDFT Program Code

In List 4, the input spectrum is supposed to be full-page array as Figure 3. The size of the spectrum array is equal to the size of the waveform array. The V93000 SOC tester provides an API for IFFT routine as follows.

DSP_IFFT(CSpectrum, CWaveform);

The point of this API is exactly the same as the IDFT routine. The input spectrum array and the output waveform array are complex numbers. They are the same size arrays.

Guide to Successful IFFT

As discussed previously, the waveform data is real number data. So when generating a waveform with applying IFFT, the result must be real number waveform and zero imaginary part. As Figure 4 shows, the relationship between time domain waveform and frequency domain spectrum is reversible with Fourier transform pair so that, in order to generate any real number waveform, the frequency domain data must be complex conjugate. Therefore you should design your frequency spectrum as complex conjugate. In other words, the real part of the frequency spectrum must be designed line-symmetrically and the imaginary part of the frequency spectrum must be designed point-symmetrically. This is the key to successfully generate a waveform by IFFT. So when using IFFT, you should always keep in mind the key word "complex conjugate" in the spectrum.

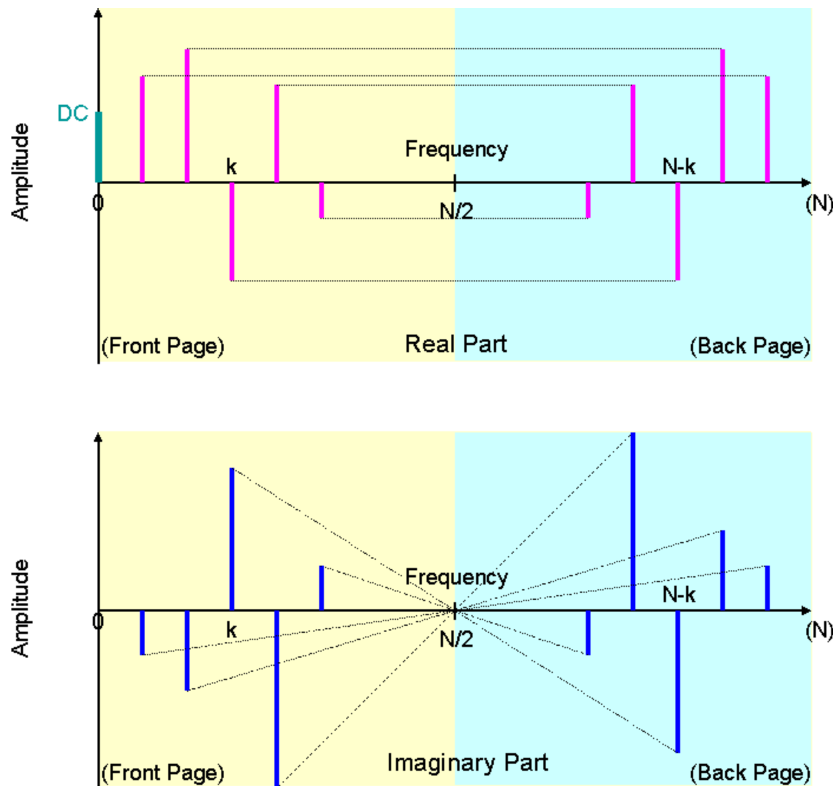

Figure 5: Complex Conjugate

Let's reconstruct the original waveform in Figure 3 from the spectrum calculated in the line 33 in List 3. The spectrum is designated as CSp2[] in List 3, and it is already complex conjugate there. So it is simple.

```
35:   ARRAY_COMPLEX    CWave2;
36:   ARRAY_D           dWave2;
37:
38:   DSP_IFFT(CSp2,CWave2);           // IFFT
39:   dWave2.resize(CWave2.size());   // Waveform container
40:   dWave2=CWave2.getReal();         // Take the Real Part
41:
```

List 5: IFFT Coding with CSp2[] (1)

When using CSp1[] in Figure 2 as the input spectrum, it is half page spectrum so that you should take care of the size, scaling and complex conjugate in the right half page. See List 6.

```
42:   ARRAY_COMPLEX    CSp0,CWave1;
43:   ARRAY_D           dWave1;
44:
45:   Nsp=CSp1.size();              // Half page spectrum size
46:   CSp0.resize(2*Nsp);           // Size doubled
47:        CSp0[0].real()=CSp1[0].real();     // DC
48:        CSp0[0].imag()=0.0;
49:   for (i=1;i<Nsp;i++) {
50:        CSp0[i].real()=0.5*CSp1[i].real(); // Slash half
51:        CSp0[i].imag()=0.5*CSp1[i].imag(); // Slash half
52:        CSp0[N-i].real()= CSp0[i].real(); // Complex-
53:        CSp0[N-i].imag()= -CSp0[i].imag(); // Conjugate
54:   }
55:        CSp0[Nsp].real()=0.0;
56:        CSp0[Nsp].imag()=0.0;
```

```
57:
58:   DSP_IFFT(CSp0,CWave1);          // IFFT
59:   dWave1.resize(CWave1.size()); //  Waveform container
60:   dWave1=CWave1.getReal();       //  Take the real part
61:
```

List 6: IFFT Coding with CSp1[] (2)

In short, for successful IFFT or IDFT, if you have a front page spectrum as Figure 2, you need to create the back page from the front page adjusting the magnitude as Figure 3. Then you can get real waveform data and zero imaginary waveform. At the end, take the real part only for real waveform.

Cases that IFFT Is Needed

When you would like to modify or manipulate a waveform, it is an opportunity for IFFT to be called. For example, the following situations would need the help of IFFT.

- Filtering in Frequency Domain
- FIR(Finite Impulse Response) Generation from Frequency Response
- Refining Fundamental Signal with Selecting Frequency Spectrum
- Waveform Interpolation by Zero Insertion
- S/N Calculation in Time Domain
- Differentiation of Waveform in Frequency Domain Operation
- Multi-tone Generation by Frequency Domain (High-speed)
- AM Modulated Waveform Generation by Frequency Domain

IFFT-employed practical applications will be discussed in the future newsletter articles.

Appendix: Playing IFFT with using FFT Routine

As seen in Equation (3) and (4), the IDFT procedure is quite similar to the DFT procedure. The difference is only the sign of sine term and scaling that is not a big issue. Just in case you have no appropriate IFFT API available for some reason, you could utilize the FFT routine for performing IFFT. This is a kind of workaround. All you have to do is to reverse the order of the input spectrum array. Let's take the full-page spectrum of CSp2[] in List 3. The procedure is as follows.

```
62:  ARRAY_COMPLEX  CSp3,CWave3;
63:  ARRAY_D         dWave3;
64:
65:  CSp3.resize(N);  // Full-page complex conjugate spectrum
66:  CSp3[0]=CSp2[0];                   // Copy the DC only.
67:  for (i=1;i<N;i++) {              //  Reverse order
68:     CSp3[i].real()=CSp2[Ndata-i].real();
69:     CSp3[i].imag()=CSp2[Ndata-i].imag();
70:  }
71:  DSP_FFT(CSp3,CWave3,RECT);     // FFT instead of IFFT
72:  dWave3.resize(N);               // Waveform container
73:  dWave3=CWave3.getReal();       // Take the real part
74:  DSP_MUL_SCL((DOUBLE)N,dWave3,dWave3);    // Scaling
75:
```

List 7: Playing IFFT with FFT API