intel®

AP-275

APPLICATION NOTE

An FFT Algorithm For MCS®-96 Products Including Supporting Routines and Examples

IRA HORDEN
ECO APPLICATIONS ENGINEER

October 1988

Order Number: 270189-002

Contact your local Intel sales office or your distributor to obtain the latest specifications before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature, may be obtained from:

Intel Corporation
P.O. Box 7641
Mt. Prospect, IL 60056-7641

or call 1-800-879-4683

AN FFT ALGORITHM FOR MCS®-96 PRODUCTS INCLUDING SUPPORTING ROUTINES AND EXAMPLES

CONTENTS

PAGE

1.0 INTRODUCTION 1

2.0 PROGRAM OVERVIEW 1

3.0 FOURIER TRANSFORMS 2

4.0 THE FFT ALGORITHM 6

5.0 USING THE FFT 7

6.0 BASIC PROGRAM FOR FFTS 10

7.0 ASM96 PROGRAM FOR FFTS 14

8.0 BACKGROUND CONTROL PROGRAM 28

9.0 ANALOG TO DIGITAL CONVERTER MODULE 39

10.0 DATA PLOTTING MODULE 50

11.0 USING THE FFT PROGRAM 58

12.0 APPENDIX A A-1

13.0 APPENDIX B B-1

BIBLIOGRAPHY B-15

Figures

1. Timing of the FFT Program 2
2. Rectangular Pulse and its Fourier Transform 3
3. Graphical Summation of Sine Waves 3
4. Square Waves from Sinusoids 4
5. Discrete Transform of a Square Wave 5
6. Bin Windows 7
7. Waveform is a Multiple of the Window 8
8. Waveform is Not a Multiple of the Window 8
9. Effect of Hanning Window on FFT Input 9
10. Bin Windows after Using Hanning Input Window 9
11. Flowchart of Basic Program 11
12. Butterflies with N = 8 14
13. FFT Output for a Square Wave Input 39

Listings

1. BASIC FFT Program 12
2. ASM96 FFT Program 15
3. Main Routine 29
4. A to D Converter Routine 40
5. The Plot Module 51

1.0 INTRODUCTION

Intel's 8096 is a 16-bit microcontroller with processing power sufficient to perform many tasks which were previously done by microprocessors or special building block computers. A new field of applications is opened by having this much power available on a single chip controller.

The 8096 can be used to increase the performance of existing designs based on 8051s or similar 8-bit controllers. In addition, it can be used for Digital Signal Processing (DSP) applications, as well as matrix manipulations and other processing oriented tasks. One of the tasks that can be performed is the calculation of a Fast Fourier Transform (FFT). The algorithm used is similar to that in many DSP and matrix manipulation applications, so while it is directly applicable to a specific set of applications, it is indirectly applicable to many more.

FFTs are most often used in determining what frequencies are present in an analog signal. By providing a tool to identify specific waveforms by their frequency components, FFTs can be used to compare signals to one another or to set patterns. This type of procedure is used in speech detection and engine knock sensors. FFTs also have uses in vision systems where they identify objects by comparing their outlines, and in radar units to detect the dopler shift created by moving objects.

This application note discusses how FFTs can be calculated using Intel's MCS®-96 microcontrollers. A review of fourier analysis is presented, along with the specific code required for a 64 point real FFT. Throughout this application note, it is assumed that the reader has a working knowledge of the 8096. For those without this background the following two publications will be helpful:

1986 Microcontroller Handbook

Using the 8096, AP-248

These books are listed in the bibliography, along with other good sources of information on the MCS-96 product family and on Fast Fourier Transforms.

2.0 PROGRAM OVERVIEW

This application note contains program modules which are combined to create a program which performs an FFT on an analog signal sampled by the on-board ADC (Analog to Digital Converter) of the 8097. The results of the FFT are then provided over the serial channel to a printer or terminal which displays the results. In the applications listed in the previous section, the data from this FFT program would be used directly by another program instead of being plotted. However, the plotted results are used here to provide an example of what the FFT does. There are four program modules discussed in this application note:

FFTRUN - Runs a 64 point FFT on its data buffer. It produces 32 14-bit complex output values and 32 14-bit output magnitudes. A fast square root routine and log conversion routine are included.

A2DCON - Fills one of two buffers with analog values at a set sample rate. The sample time can be as fast as 50 microseconds using 8x9xBH components.

PLOTSP - Plots the contents of a buffer to a serially connected printer. Routines are provided for console out and hexadecimal to decimal conversion and printing.

FTMAIN - The main module which controls the other modules.

Each of the modules will be described separately. In order to better understand how the programs work together, a brief tutorial on FFTs will be presented first, followed by descriptions of the programs in the order listed above.

The final program uses 64 real data points, taken from either a table or analog input 1. Each of the data points is a 16-bit signed number. The processing takes 12.5 milliseconds when internal RAM is used as the data space. If external RAM is used, 14 milliseconds are required. Larger FFTs can be performed by slightly modifying the programs. A 256-point FFT would take approximately 65 milliseconds, and a 1024-point version would require about 300 milliseconds.

In the program presented, the analog sampling time is set for 1 sample every 100 microseconds, providing the 64 samples in 6.4 milliseconds. The sampling time can be reduced to around 60 microseconds per point by changing a variable, and less than 50 microseconds by using the 8x9xBH series of parts, since they have a 22 microsecond A to D conversion time.

The programs are set up to be run in a sequence instead of concurrently. This provides the fastest operation if the sampling speed were reduced to the minimum possible. For the fastest operation above about 80 microseconds a sample, the programs could be run concurrently, but this would require some minor modifications of the program. Figure 1 shows the timing of the program as presented.

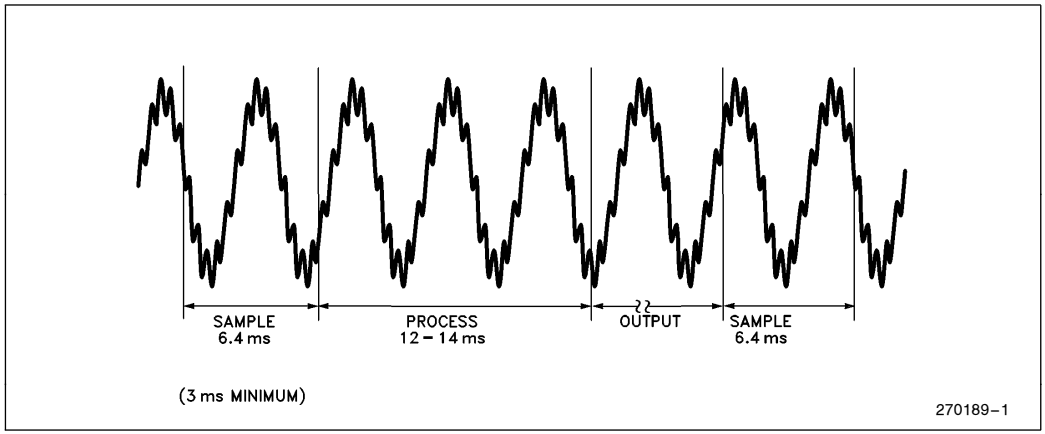


Figure 1. Timing of the FFT Program

These programs have run in the Intel Microcontroller Operation Application's Lab and produced the results presented in this application note. Since the programs have not undergone any further testing, we cannot guarantee them to be bug proof. We, therefore, recommend that they be thoroughly tested before being used for other than demonstration purposes.

3.0 FOURIER TRANSFORMS

A Fourier Transform is a useful analytical tool that is frequently ignored due to its mathematically oriented derivations. This is unfortunate, since Fourier transforms can be used without fully understanding the mathematics behind them. Of course, if one understands the theory behind these transforms, they become much more powerful.

The majority of this application note deals with how a Fast Fourier Transform (FFT) can be used for spectrum analysis. This procedure takes an input signal and separates it into its frequency components. One can almost treat the FFT as a black box, which has as its output, the frequency components and magnitudes of the input signal, much like a spectrum analyzer.

From a mathematical standpoint, Fourier Transforms change information in the time domain into the frequency domain. The theory behind the Fourier transform stems from Fourier analysis, also called frequency analysis.

There are many books on the topic of Fourier analysis, several of which are listed in the bibliography. In this application note, only the pertinent formulas and uses will be presented, not their derivations.

The main idea in Fourier analysis is that a function can be expressed as a summation of sinusoidal functions of different frequencies, phase angles, and magnitudes. This idea is represented by the Fourier Integral:

$$H(f) = \int_{-\infty}^{\infty} h(t)\, e^{-j2\pi ft}\, dt \qquad (1)$$

Where: H(f) is a function of frequency
h(t) is a function of time

Since

$$e^{-j\theta} = \cos\theta - j\sin\theta \qquad (2)$$

$$H(f) = \int_{-\infty}^{\infty} h(t)\,(\cos(2\pi ft) - j\sin(2\pi ft))\, dt \qquad (3)$$

Figure 2 shows a rectangular pulse and its Fourier transform. Note that the results in the frequency domain are continuous rather than discrete. The horizontal axis in Figure 2a is frequency, while that of Figure 2b is time.

In a simplified case, the varying phase angles can be removed, and the integral changed to a summation, known as a Fourier Series. All periodic functions can be described in this way. This series, as shown below, can help provide a more graphical understanding of Fourier analysis.

$$y(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(2\pi nf_0t) + b_n \sin(2\pi nf_0t)] \qquad (4)$$

for $n = 1$ to ∞

Where $f_0 = \frac{1}{T_0}$, the fundamental frequency.

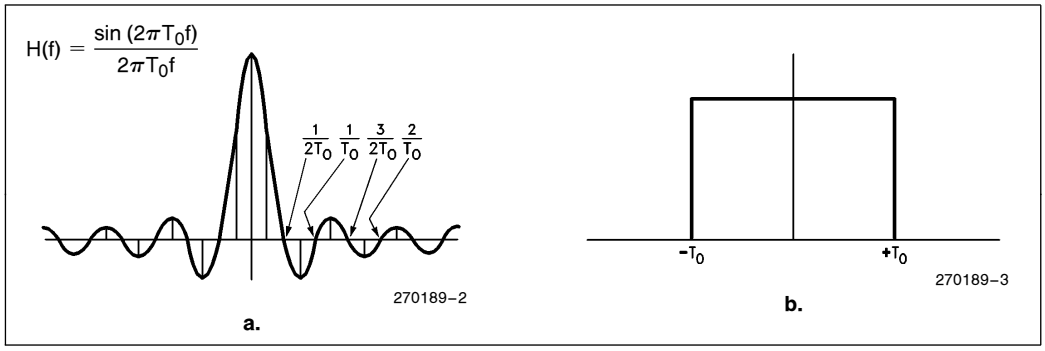


Figure 2. Rectangular Pulse and Its Fourier Transform

This formula can also be represented in complex form as:

$$\sum_{n=-\infty}^{\infty} \alpha n \, e^{j2\pi n f_0 t} \qquad (5)$$

The Fourier series for a square wave is

$$\sum_{K=0}^{\infty} \frac{\sin((2k+1)\,2\pi f_0 t)}{(2k+1)} \qquad (6)$$

If these sinusoids are summed, a square wave will be formed. Figure 3 shows the graphical summation of the first 3 terms of the series. Since the higher frequencies contribute to the squareness of the waveform at the corners, it is reasonable to compare only the flatness of the top of the waveform. The sharpness or risetime of the waveform can be determined by the highest frequency term being summed. With rise and fall times of 10% of the period, the waveform generated by the first 3 terms is within 20% of ideal. At 7 terms it is within 10%, and at 20 terms it is within 5%. With a 5% risetime, it is within 20% of ideal after 5 terms, 10% after 13 terms and 5% after 32 terms. Figure 4 shows the resultant waveforms after the summation of 7, 15 and 30 terms.

Fourier analysis can be used on equation 4 to find the coefficients a_n and b_n. To make this process easier to use with a computer, a discrete form, rather than a continuous one, must be used. The discrete Fourier transform, shown in Equation 7, is a good approximation to the continuous version. The closeness of the approximation depends on several conditions which will be discussed later. The input to this transform is a set of N equally spaced samples of a waveform taken over a period of NT. The period NT is frequently referred to as the "Sampling Window".

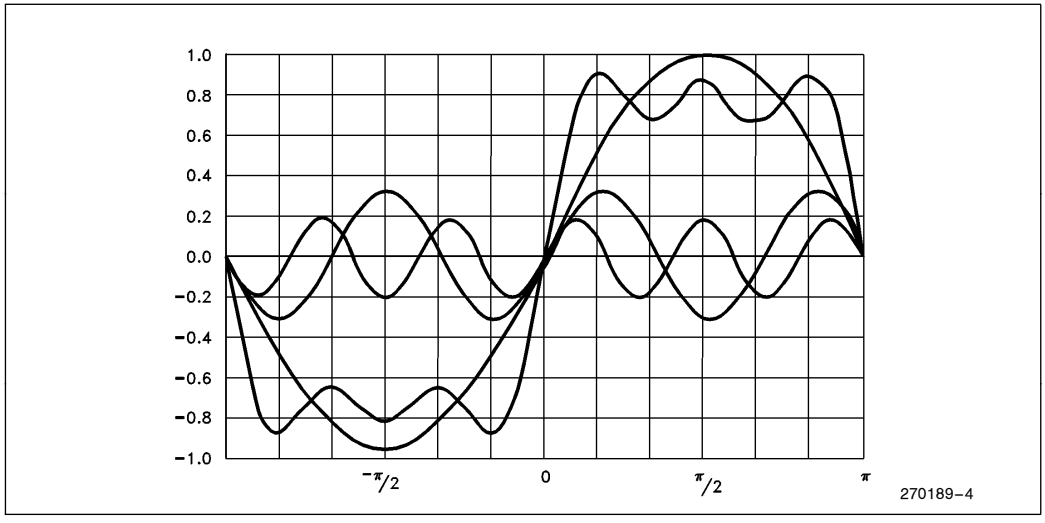
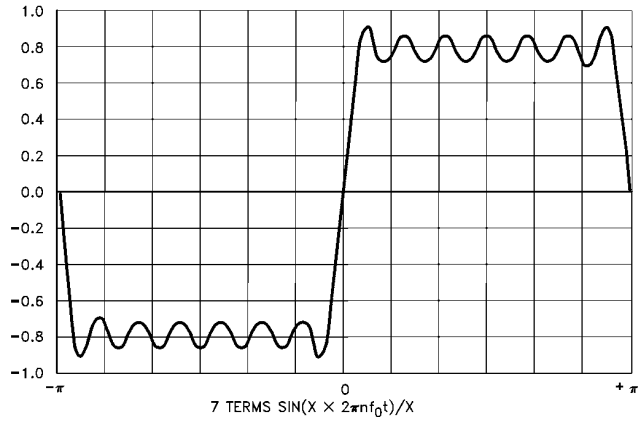
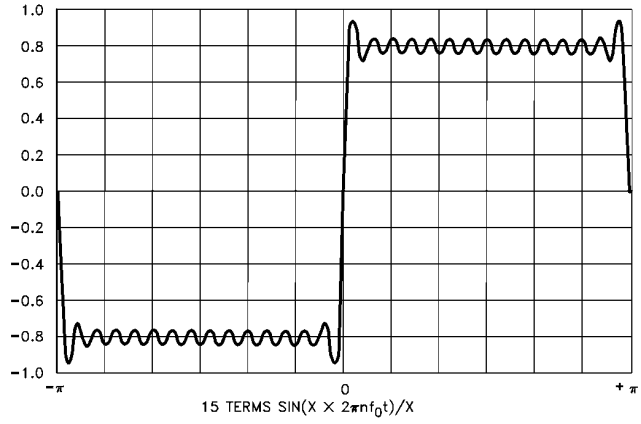


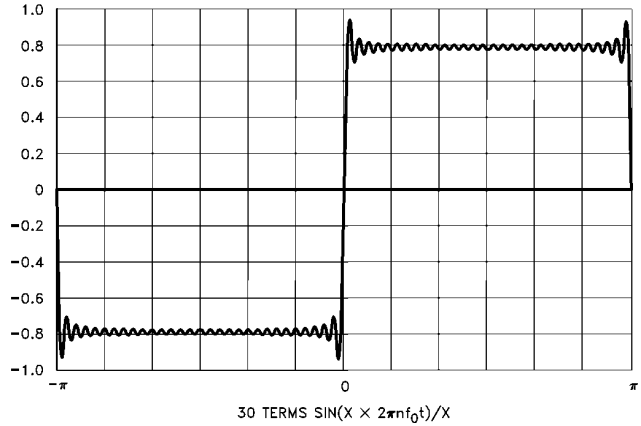
Figure 3. Graphical Summation of Sinewaves



270189–5



270189–6



270189–7

Figure 4. Square Wave from Sinusoids

$$H\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT)e^{-j2\pi nk/N}$$
$$n = 0, 1, \ldots, N-1 \quad (7)$$

Where: H(f) is a function of frequency
h(t) is a function of time
T is the time span between samples
N is the number of samples in the window
n = 0,1,2 ... N-1

This transform is used for many applications, including Fourier Harmonic Analysis. This procedure uses the transform to calculate the coefficients used in Equation 5. In order to do this, the factor T/NT must be added to the transform as follows:

$$H\left(\frac{n}{NT}\right) = \frac{T}{(NT)} \sum_{k=0}^{N-1} h(kT)\, e^{-j2\pi nk/N}$$
$$n = 0, 1, 2, 3, \ldots, N-1 \quad (8)$$

The factor provides compensation for the number of samples taken. Note that the functions H(f) and h(t) are complex variables, so the simplicity of the equation can be misleading. Once the values of h(t) are known, (ie. the value of the input at the discrete times (t)), the Fourier Transform can be used to find the magnitude and phase shift of the signal at the frequencies (f).

A spectrum analyzer can provide similar information on an analog input signal by using analog filters to separate the frequency components. Regardless of its source, the information on component frequencies of a signal can be used to detect specific frequencies present in a signal or to compare one signal to another. Many lab experiments and product development tests can make use of this type of information. Using these methods, the purity of signals can be measured, specific harmonics can be detected in mechanical equipment, and noise bursts can be classified. All of this information can be obtained while still treating the FFT process as a black box.

Consider the discrete transform of a square wave as shown in Figure 5. Note that the component magnitudes, as shown in the series of Equation 6, are shown in a mirrored form in the transform. This will happen whenever only real data is used as the FFT input, if both real and imaginary data were used the output would not be guaranteed to be symmetrical. For this reason, there is duplicate information in the transform for many applications. Later in this section a method to make the most of this characteristic is discussed.

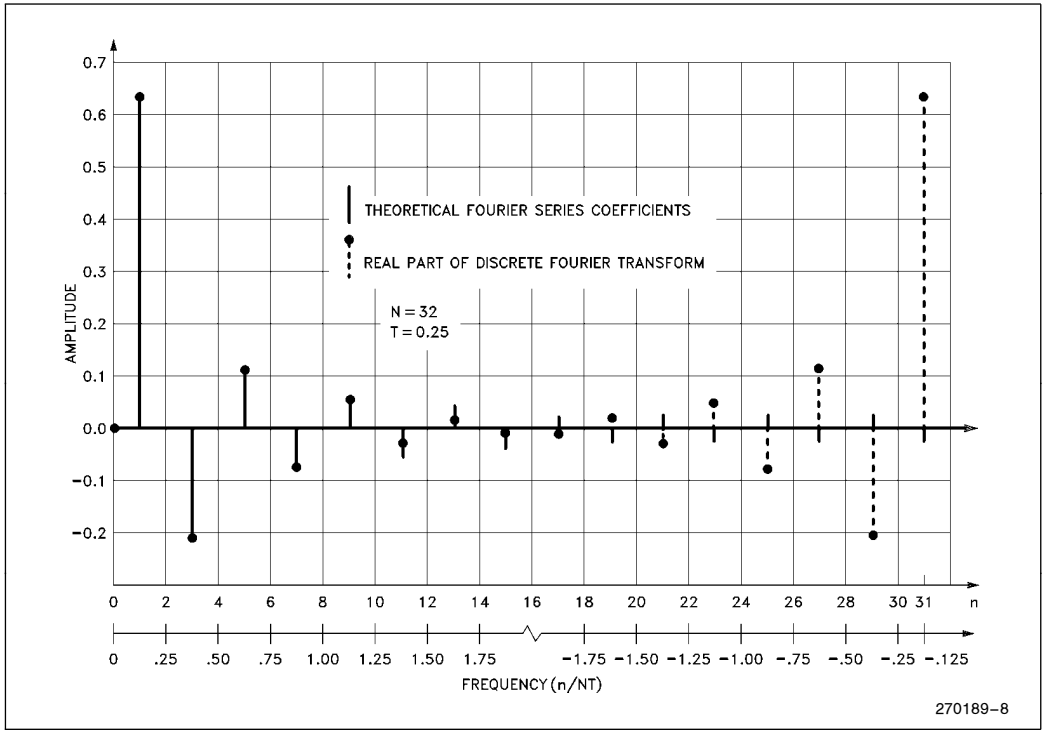


Figure 5. Discrete Transform of a Square Wave

If one looks at Equation 8, it can be seen that the calculation of a discrete Fourier transform requires N squared complex multiplications. If N is large, the calculation time can easily become unrealistic for real-time applications. For example, if a complex multiplication takes 40 microseconds, at N = 16, 10 milliseconds would be used for calculation, while at N = 128, over half a second would be needed. A Fast Fourier Transform is an algorithm which uses less multiplications, and is therefore faster. To calculate the actual time savings, it is first necessary to understand how a FFT works.

4.0 THE FFT ALGORITHM

The FFT algorithm makes use of the periodic nature of waveforms and some matrix algebra tricks to reduce the number of calculations needed for a transform. A more complete discussion of this is in Appendix A, however, the areas that need to be understood to follow the algorithm are presented here. This information need not be read if the reader's intent is to use the program and not to understand the mathematical process of the algorithm

To simplify notation the following substitutions are made in Equation 8.

$$W = e^{-j2\pi/N}$$

$$k = kT$$

$$n = \frac{n}{NT}$$

The resultant equation being

$$x(n) = \sum_{k=0}^{N-1} n(k)W^{nk} \qquad (9)$$

Expressed as a matrix operation

$$\begin{bmatrix} X(1) \\ X(2) \\ X(3) \\ \vdots \\ X(N\text{-}1) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^N \\ W^0 & W^2 & W^4 & \cdots & W^{2N} \\ \vdots & \vdots & \vdots & & \vdots \\ W^0 & W^{(N-1)} & W^{2(N-1)} & \cdots & W^{(N-1)2} \end{bmatrix} \begin{bmatrix} X_0(0) \\ X_0(1) \\ X_0(2) \\ \vdots \\ X_0(N-1) \end{bmatrix}$$

A brief review of matrix properties can be found in Appendix A. Because of the periodic nature of W the following is true:

$$W^{nk} \text{ MOD } N = W^{nk} \qquad (10)$$

$$= \text{COS}(2\pi\, nk/N) - j\,\text{SIN}(2\pi nk/N)$$

$W^0 = 1$ therefore, if $nk \text{ MOD } N = 0$, $W^{nk} = 1$

This reduces the calculations as several of the W terms go to 1 and the highest power of W is N. All of W values are complex, so most of the operations will have to be complex operations. We will continue to use only the W, X(n) and X0(k) symbols to represent these complex quantities.

The FFT algorithm we will use requires that N be an integral power of 2. Other FFT algorithms do not have this restriction, but they are more complex to understand and develop. Additionally, for the relatively small values of N we are using this restriction should not provide much of a problem. We will define EXPONENT as log base 2 of N. Therefore,

$$N = 2^{\text{EXPONENT}}$$

The magic of the FFT, (as detailed in Appendix A), involves factoring the matrix into EXPONENT matrices, each of which has all zeros except for a 1 and a W^{nk} term in each row. When these matrices are multiplied together the result is the same as that of the multiplication indicated in Equation 9, except that the rows are interchanged and there are fewer non-trivial multiplications. To reorder the rows, and thus make the information useful, it is necessary to perform a procedure called "Bit Reversal".

This process requires that N first be converted to a binary number. The least significant bit (lsb) is swapped with the most significant bit (msb). Then the next lsb is swapped with the next msb, and so on until all bits have been swapped once. For N = 8, 3 bits are used, and the values for N and their bit reversals are shown below:

Number	Binary	Bit Reversal	Decimal BR
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Recall that the FFT of real data provides a mirrored image output, but the FFT algorithm can accept inputs with both real and imaginary components. Since the inputs for harmonic analysis provided by a single A to D are real, the FFT algorithm is doing a lot of calculations with one input term equal to zero. This is obviously not very efficient. More information for a given size transform can be obtained by using a few more tricks.

It is possible to perform the FFT of two real functions at the same time by using the imaginary input values to the FFT for the second real function. There is then a post processing performed on the FFT results which separate the FFTs of the two functions. Using a similar procedure one can perform a transform on 2N real samples using an N complex sample transform.

The procedure involves alternating the real sample values between the real and imaginary inputs to the FFT. If, as in our example, the input to the FFT is a 2 by 32 array containing the complex values for 32 inputs, the 64 real samples would be loaded into it as follows:

N	00 01 02 03 04 05 06 07 ..... 30 31
REAL	00 02 04 06 08 10 12 14 ..... 60 62
IMAGINARY	01 03 05 07 09 11 13 15 ..... 61 63

This procedure is referred to as a pre-weave. In order to derive the desired results, the FFT is run, and then a post-weave operation is performed. The formula for the post-weave is shown below:

$$X_r(n) = \left[\frac{R(n)}{2} + \frac{R(N-n)}{2}\right] + \cos\frac{\pi n}{N}\left[\frac{I(N)}{2} + \frac{I(N-n)}{2}\right] - \sin\frac{\pi n}{N}\left[\frac{R(n)}{2} - \frac{R(N-n)}{2}\right] \quad n = 0, 1, \ldots, N-1$$

$$X_i(n) = \left[\frac{I(n)}{2} - \frac{I(N-n)}{2}\right] - \sin\frac{\pi n}{N}\left[\frac{I(n)}{2} + \frac{I(N-n)}{2}\right] - \cos\frac{\pi n}{N}\left[\frac{R(n)}{2} - \frac{R(N-n)}{2}\right] \quad n = 0, 1, \ldots, N-1 \qquad (11)$$

Where R(n) is the real FFT output value
I(n) is the imaginary FFT output value
Xr(n) is the real post-weave output
Xi(n) is the imaginary post-weave output

Note that the output is now one-sided instead of mirrored around the center frequency as it is in Figure 5. The magnitude of the signal at each frequency is calculated by taking the square root of the sum of the squares. The magnitude can now be plotted against frequency, where the frequency steps are defined as:

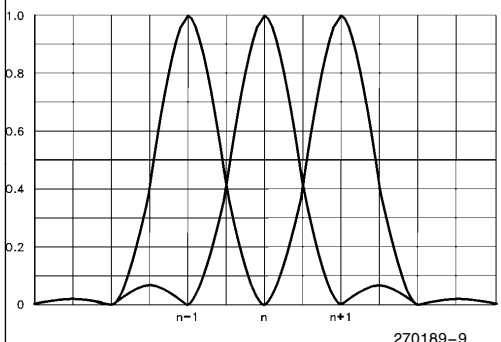
$$\frac{n}{NT} \quad n = 0, 1, 2, 3, \ldots, N-1$$

Where N is the number of complex samples (ie. 32 in this case) T is the time between samples

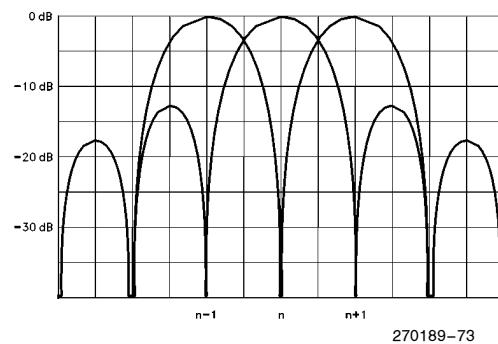
A value of zero on the frequency scale corresponds to the DC component of the waveform. Most signal analysis is done using Decibels (dB), the conversion is dB = 10 LOG (Magnitude squared). Decibels are not used as an absolute measure, instead signals are compared by the difference in decibels. If the ratio between two signals is 1:2 then there will be a 3 dB difference in their power.

5.0 USING THE FFT

There are several things to be aware of when using FFTs, but with the proper cautions, the FFT output can be used just like that of a spectrum analyzer. The



(a.) Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).



(b.) 10 Log Relative Power of Windows (Side Lobes of Side Bins Removed for Clarity).

Figure 6. Bin Windows

first precaution is that the FFT is a discrete approximation to a continuous Fourier Transform, so the output will seldom fit the theoretical values exactly, but it will be very close.

Since the programs in this application note generate a one-sided transform with N = 32, the frequency granularity is fairly course. Each of the frequency components output from the FFT is actually the sum of all energy within a narrow band centered on that frequency. This band of sensitivity is referred to as a "bin". The reported magnitude is the actual magnitude multiplied by the value of the bin window at the actual frequency. Figure 6 shows several bin windows. Note that these windows overlap, so that a frequency midway between the two center frequencies will be reported as energy split between both windows. Be careful not to confuse the *sampling window* NT with *bin windows* or with the *windowing function*.

Another area of caution is the relationship of the sampling window to the frequency of the waveform. For the best accuracy, the window should cover an exact multiple of the period of the waveform being analyzed. If it covers less than one period, the results will be invalid. Other variations from ideal will not produce invalid results, just additional noise in the output.

If the sampling window does not cover an exact multiple of all of the frequency components of a waveform, the FFT results will be noisy. The reason for this is the sharp edge that the FFT sees when the edges of the window cut off the input waveform. Figure 7 shows a waveform that is an exact multiple of the window and

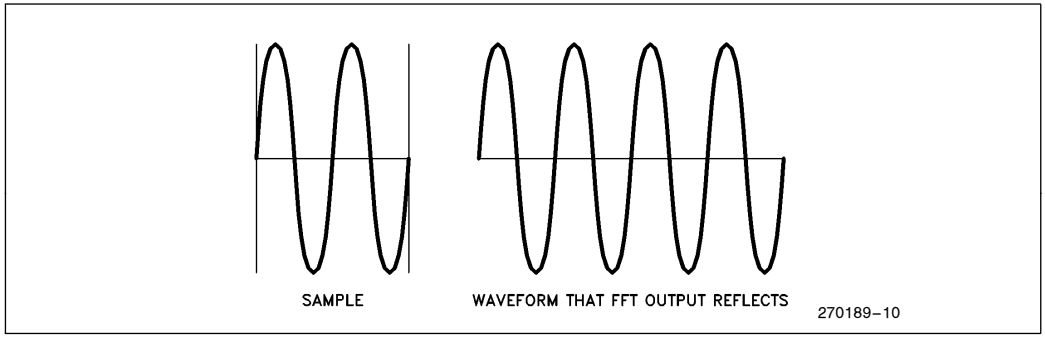


Figure 7. Waveform is a Multiple of the Window

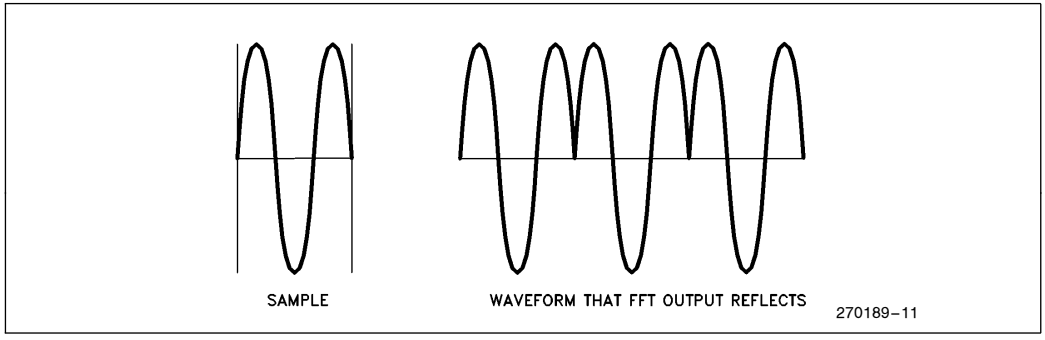
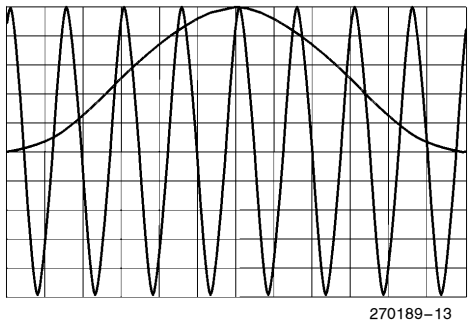


Figure 8. Waveform is Not a Multiple of the Window

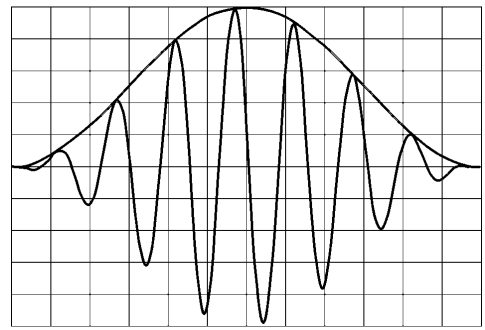
the periodic waveform that the FFT output reflects. In Figure 8, the waveform is not a multiple of the window and the waveform that the FFT output reflects has discontinuities. These discontinuities contribute to the noise in an FFT output. This noise is called "spectral leakage", or simply "leakage", since it is leakage between one frequency spectrum and another which is caused by digitization of an analog process.

To reduce this leakage, a process called windowing is used. In this procedure the input data is multiplied by specific values before being used in the FFT. The term "windowing" is used because these values act as a window through which the input data passes. If the input window goes smoothly to zero at both endpoints of the sampling window, there can be no discontinuities. Figure 9 shows a Hanning window and its effect on the input to an FFT. The Hanning window was named after its creator, Julius Von Hann, and is one of the most commonly used windows. More information on windowing and the types of windows can be found in the paper by Harris listed in the bibliography. As expected, the results of the FFT are changed because of the input windowing, but it is in a very predictable way.

Using the Hanning window results in bin windows which are wider and lower in magnitude than normal, as can be seen by comparing Figure 6 with Figure 10. For an input frequency which is equal to the center frequency of a bin window, the attenuation will be 6 dB on the center frequency. Since the bin windows are

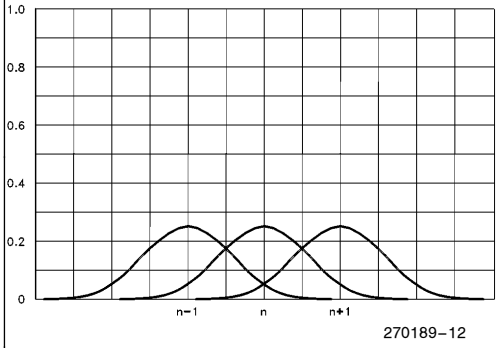


(a). Original Signal and Hanning Window

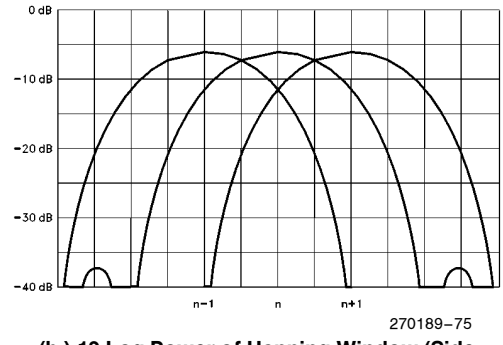


(b). Signal After Hanning Window

Figure 9. Effect of Hanning Window on FFT Input



(a.) Relative Power of Hanning Window



(b.) 10 Log Power of Hanning Window (Side Lobes of Side Bin Window Removed)

Figure 10. Bin Windows after Using Hanning Input Window

wider than normal, the input frequency will also have energy which falls into the bins on either side of center. These side bins will show a reading of 6 dB below the center window. The disadvantage of this spreading is far less than the advantage of removing leakage from the FFT output.

A set of FFT output plots are included in the Appendix. These plots show the effect of windowing on various signals. There are examples of all of the cases described above. A brief discussion of the plots is also presented.

Applications which can make use of this frequency magnitude information include a wide range of signal processing and detection tasks. Many of these tasks use digital filtering and signature analysis to match signals to a standard. This technique has been applied to anti-knock sensors for automobile engines, object identification for vision systems, cardiac arrhythmia detectors, noise separation and many other applications. The ability to do this on a single-chip computer opens a door to new products which would have not been possible or cost effective previously.

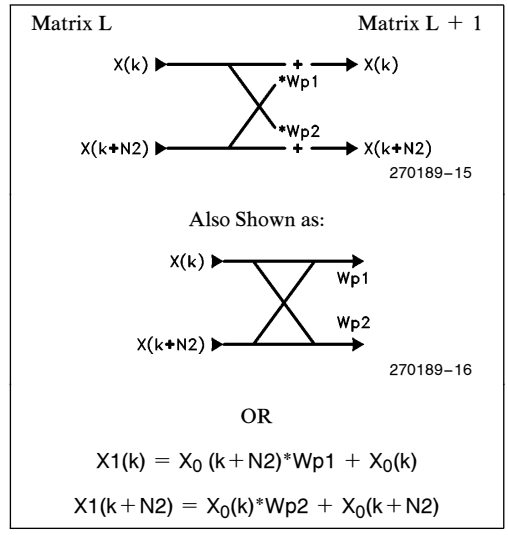
The next four sections of this application note cover the operation of the programs on a line by line basis. Section 6 shows an implementation of the FFT algorithm in BASIC. This code is used as a template to write the ASM96 code in Section 7. Sections 8, 9, and 10 cover the code sections which support the FFT module. After all of the code sections are discussed, an overview of how to use the program is presented in Section 11.

6.0 BASIC PROGRAM FOR FFTS

The algorithm for this FFT is shown in the flowchart in Figure 11 and the BASIC program in Listing 1. There are four sections to this program: initialization, pre-weaving, transform calculation, and post-weaving. The flowchart is generalized, however, the BASIC program has been optimized for assembly language conversion with 64 real samples.

On the flowchart, the initialization and pre-weaving sections are incorporated as "Read in Data". The data to be read includes the raw data as well as the size of the array and the scaling factor. The details for pre-weaving have been discussed earlier, and initialization varies from computer to computer. LOOP COUNT keeps track of which of the factored matrices are being multiplied. SHIFT is the shift count which is used to determine the power of W (as defined earlier) which will be used in the loop.

For each loop N calculations are performed in sets of two. Each calculation set is referred to as a butterfly and has the following form:



In general, the W factors are not the same. However, for the case of this FFT algorithm, Wp1 will always equal (−Wp2). This is because of the way in which "p" is calculated, and the fact that W(x) is a sinusoidal function.

The inner loop in the flowchart is performed N2 times. For LOOP = 1, N2 = N/2 and if INCNT = N2 then k = N2 and k + N2 = N, so the first loop is done and parameters LOOP, N2, and SHIFT are updated. For the first loop, all N/2 sets of calculations are performed contiguously. As LOOP increases, the number of contiguous calculations are cut in half, until LOOP = EXPONENT.

When LOOP = EXPONENT, N2 = 1, the butterfly is then performed on adjacent variables. Figure 12 shows the butterfly arrangement for a calculation where N = 8, so that EXPONENT = 3.

The BASIC program follows this flowchart, but operations have been grouped to make it easier to convert it to assembly language. Also not shown in the flowchart are several divide by 2 operations. There are five in the main section, one per loop. These provide the T/NT factor in equation 8 for N = 32 ($2^5 = 32$). There is also an extra divide by two in the post-weave section. It is required to prevent overflows when performing the 16-bit signed arithmetic in the ASM96 program. As a result of these operations, the input scale factor is $\pm 1 = \pm 32767$ and the output scaling is $\pm 1 = \pm 16384$. Note, the maximum input values are ± 0.99997.

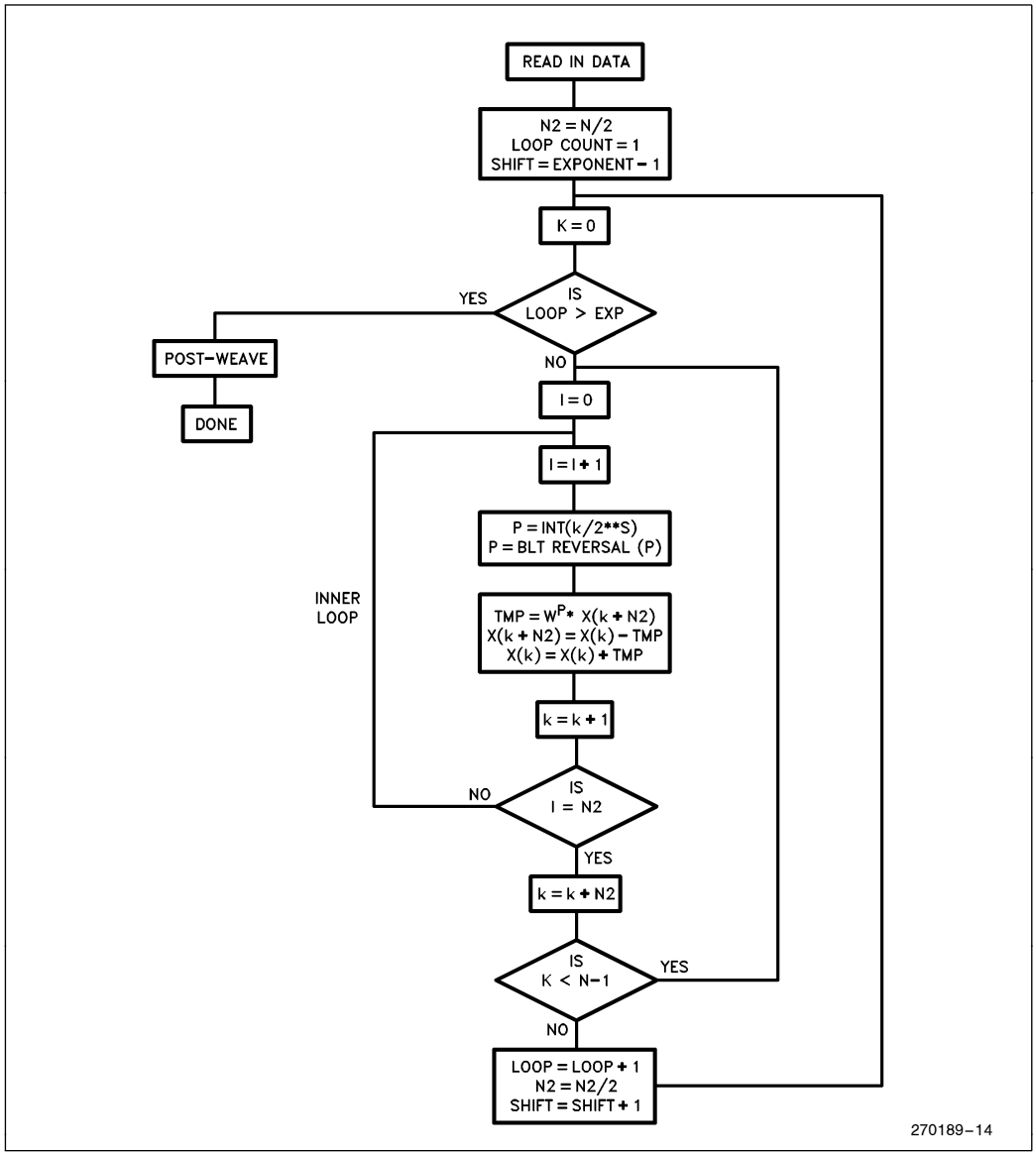


Figure 11. Flowchart of Basic Program

```
100 '  THIS IS FFT13, FEBRUARY 4, 1986
105 '
110     ' COPYRIGHT INTEL CORPORATION, 1985
115     ' BY IRA HORDEN, MCO APPLICATIONS
120 '
125 '   THIS PROGRAM PERFORMS A FAST FOURIER TRANSFORM ON 64 REAL DATA POINTS
130 ' USING A 2N-POINTS WITH AN N-POINT TRANSFORM ALGORITHM.  THE FIRST
135 ' SECTION OF THE PROGRAM PERFORMS A STANDARD TRANSFORM ON DATA THAT HAS
140 ' BEEN INTERLEAVED BETWEEN THE REAL AND IMAGINARY INPUT VALUES.  THE
145 ' RESULTS OF THAT TRANSFORM ARE THEN POST-PROCESSED IN THE SECOND SECTION
150 ' OF THE PROGRAM TO PROVIDE THE 32 OUTPUT BUCKETS.  THE OUTPUT VALUES ARE
155 ' MULTIPLIED BY "M" TO MAKE IT EASY TO COMPARE WITH THE ASM-96 PROGRAM
160 '
165 INPUT "NAME OF LIST FILE"; LST$
170 PRINT
175 OPEN LST$ FOR OUTPUT AS #1
180 '
200                                              ' SET UP VARIABLES FOR BASIC
210 DIM XR(32),XI(32),WR(32),WI(32),BR(32)
220 M=16383                                    ' M=MULT. FACTOR FOR SCALING
230 N=32 : N1=31 : N2=N/2                      ' N=NUMBER OF DATA POINTS
240 LOOP=1  : K=0 : EXPONENT=5 : SHIFT=EXPONENT-1       ' 2**E=N
250 PI=3.141592654# : TPN=2*PI/N : PIN=PI/N
260 '
270                                              ' READ IN CONSTANTS
280 FOR P=0 TO 31 : PN=P*TPN
290 WR(P)=COS(PN) : WI(P)=-SIN(PN) : READ BR(P)
300 NEXT P
310 '
320 FOR K=0 TO 31                               ' READ IN DATA
330 READ XR(K)  : READ XI(K)
350 NEXT K
360 '
400             ' INITIALIZATION OF LOOP
410 K=0
420 IF LOOP>EXPONENT THEN 700
430 INCNT=0
440             ' ACTUAL CALCULATIONS BEGIN HERE
445 '
450 INCNT=INCNT+1
460 P=BR(INT(K/(2^SHIFT)))
470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2       ' WRP AND WIP ARE CONSTANTS BASED ON
480 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2    ' SINES AND COSINES OF BIT REVERSED
490 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2    ' VALUES OF K SHIFTED RIGHT S TIMES
500 TMPR1=XR(K)/2 : TMPI1=XI(K)/2
510 XR(K+N2) = TMPR1 - TMPR      ' TMPR, TMPI ARE THE REAL AND IMAGINARY
520 XI(K+N2) = TMPI1 - TMPI      ' RESULTS OF A COMPLEX MULTIPLICATION
530 XR(K) = TMPR1 + TMPR
540 XI(K) = TMPI1 + TMPI
550 '
560 K=K+1
570 IF INCNT<N2 THEN GOTO 450
580 K=K+N2                     ' SINCE THE ARRAY IS PROCESSED 2 POINTS AT A TIME,
590 IF K<N1 THEN GOTO 430      ' ONLY N/2 LOOPS NEED TO BE MADE.  ON EACH PASS,
600 LOOP=LOOP+1 : N2=N2/2      ' THE VALUE OF N2 CHANGES AND SMALLER CONSECUTIVE
605 SHIFT=SHIFT-1              ' SECTIONS ARE PROCESSED.
610 GOTO 400
620 '
690 '
691 '
692 '
693 '
```

270189-17

Listing 1—BASIC FFT Program

```
694 '
695 '
696 '
697 '
700                              ' POST-PROCESSING AND REORDERING BEGIN HERE
710 '
720 FOR K = 0 TO 31
730 KPIN=K*PIN
740 XRBRK=XR(BR(K)) : XIBRK=XI(BR(K))    ' CONDENSED FOR EASE OF ASM PROGRAMMING
750 XRBRNK=XR(BR(N-K)) : XIBRNK=XI(BR(N-K))
760 TI = (XIBRK+XIBRNK)/2
770 TR = (XRBRK-XRBRNK)/2
780 XRT= (XRBRK+XRBRNK)/4
790 XIT= (XIBRK-XIBRNK)/4
800 OUTR= XRT + TI*COS(KPIN)/2 - TR*SIN(KPIN)/2
810 OUTI= XIT - TI*SIN(KPIN)/2 - TR*COS(KPIN)/2
820 '
830 MAGSQ = OUTR*OUTR+OUTI*OUTI     ' THE ASM-96 PROGRAM USES A TABLE LOOK-UP
840 MAG = SQR(MAGSQ)                ' ROUTINE TO CALCULATE SQUARE ROOTS
845 IF MAGSQ*M < .5 THEN DECIBEL=0 : GOTO 900
847 DBFACT=M/2/32767*M   ' M^2 / 64K
850 DECIBEL=10*LOG(MAGSQ*DBFACT)
860 DECIBEL=DECIBEL * .434294481#
900    GOTO 930
910 PRINT #1, USING "######  "; K,
920 PRINT #1, USING "\       \"; HEX$(M*OUTR), HEX$(M*OUTI), HEX$(M*MAG)
930 ' GOTO 950
942 PRINT #1, USING "##  "; K;
943 PRINT #1, USING "##.#####  "; OUTR,OUTI,MAG;
945 PRINT #1, USING "###.###  "; DECIBEL;
947 PRINT #1, USING "######   "; M*OUTR, M*OUTI, M*MAG
950 NEXT K
960 '
970 IF LST$<>"SCRN:" THEN PRINT #1, CHR$(12)
999 END
1000 END
1010             ' DATA FOR BR(P)  - BIT REVERSAL
1020 DATA 0,16,8,24,4,20,12,28,2,18,10,26,6,22,14,30
1030 DATA 1,17,9,25,5,21,13,29,3,19,11,27,7,23,15,31
1040             ' DATA FOR XR,XI
1050 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1060 DATA 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
1070 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
1080 DATA -2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2,-2
```

270189-18

Listing 1—BASIC FFT Program (Continued)

Lines 165-175 set up the file for printing the data, this can be SCRN:, LPT1:, or any other file.

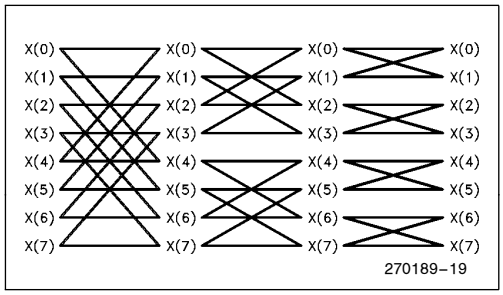


Figure 12. Butterflies with N = 8

Lines 200-310 set up the constants and calculate the W^P terms which are stored in the matrices WR(p) and WI(p), for the real and imaginary component respectively.

Lines 320-350 read in the data, alternately placing it into the real and imaginary arrays. The data is scaled by 2 to make the data table simpler.

Lines 410-430 initialize the loop and test for completion.

Lines 450-620 perform the FFT algorithm. Note that all calculations are complex, with the suffixes "R" and "I" indicating real and imaginary components respectively.

The variables on line 470, TMPR1 and TMPI1 would normally not be used in a BASIC program as more than one operation can be performed on each line. However, indirect table lookups always use a separate line of assembly code, so separate lines have been used here.

Lines 700-810 perform the post-weave. This is not in the flowchart, but can be found in Equation 11. Once again, table look-ups are separated and additional variables are used for clarity. The variables BR(x) are the bit reversal values of x.

Line 830 calculates the magnitude of the harmonic components.

Lines 900-950 print the results of the calculations, with line 900 determining if the print-out should be in hex or decimal.

Lines 1000-1080 are the data for the bit reversal values and input datapoints. The input waveform is one cycle of a square-wave.

7.0 ASM96 PROGRAM FOR FFTS

The BASIC program just presented has been used as an outline for the ASM96 program shown in Listing 2. There are many advantages to using the BASIC program as a model, the main ones being debugging and testing. Since the BASIC program is so similar in program flow to the ASM96 program, it's possible to stop the ASM96 program at almost any point and verify that the results are correct.

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                  02/18/86          PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FFTRUN.A96
OBJECT FILE: :F2:FFTRUN.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                    1   $pagelength(50)
                                    2
                                    3   FFT_RUN  MODULE  STACKSIZE(6)
                                    4
                                    5   ; Intel Corporation, January 24, 1986
                                    6   ; by Ira Horden, MCO Applications
                                    7
                                    8
                                    9   ;       This module performs a fast fourier transform (FFT) on 64 real data
                                   10   ; points using a 2N-point algorithm.  The algorithm involves using a standard
                                   11   ; FFT procedure for 32 real and 32 imaginary numbers.  The real and imaginary
                                   12   ; arrays are filled alternately with real data points, and the output of the
                                   13   ; FFT is run through a post-processor.  The result is a one sided array with 32
                                   14   ; output buckets.  The post processing includes a table lookup algorithm for
                                   15   ; taking the square root of an unsigned 32-bit number.
                                   16
                                   17   ;      All of the calculations in the main FFT program are done using 16-bit
                                   18   ; signed integers.  The maximum value of any frequency component is therefore
                                   19   ; +/- 32K.  (Note that a square wave of +/-32K has a fundamental component
                                   20   ; greater than +/- 40K).  Wherever possible tables are used to increase the
                                   21   ; speed of math operations.  The complete transform, including obtaining the
                                   22   ; absolute magnitude of each frequency component, executes in 12
                                   23   ; milliseconds with internal variables, 14 ms with external.
                                   24
                                   25   ;      The program requires two 32-word input arrays, with the sample values
                                   26   ; alternated between the two.  These start at XREAL and XIMAG.  The resultant
                                   27   ; magnitude will be placed in a 32-word array at FFT_OUT.  These are all
                                   28   ; externally defined variables.  The external constant SCALE_FACTOR is used to
                                   29   ; divide the output when averaging will be used.  Since the program averages
                                   30   ; its output, it is necessary to clear the array based at FFT_OUT before
                                   31   ; calling FFT_CALC to start the program.
                                   32
                                   33   ;      The program was originally written in BASIC for testing purposes.  The
                                   34   ; comments include these BASIC statements to make it easier to follow the
                                   35   ; algorithm.
                                   36
                                   37   $EJECT
```

270189-33

Listing 2—ASM96 FFT Program

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                  02/18/86          PAGE    2

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                 38
    0000                         39  RSEG
                                 40   EXTRN  portl, zero, error
                                 41
    0024                         42  OSEG at 24H
    0024                         43          TMPR:    dsl     1       ; Temporary register, Real
    0028                         44          TMPI:    dsl     1       ; Temporary register, Imaginary
    002C                         45          TMPR1:   dsl     1       ; Temporary register1, Real
    0030                         46          TMPI1:   dsl     1       ; Temporary register1, Imaginary
    0034                         47          XRTMP:   dsl     1       ; Temporary data register, Real
    0038                         48          XITMP:   dsl     1       ; Temporary data register, Imaginary
    003C                         49          XRRK:    dsl     1
    0040                         50          XRRNK:   dsl     1
    0044                         51          XIRK:    dsl     1
    0048                         52          XIRNK:   dsl     1
      003C                       53          diff     equ     xrrk    :long   ; Table difference for square root
      0040                       54          sqrt     equ     xrrnk   :long   ; Square root
      0040                       55          log      equ     xrrnk   :long   ; 10 Log magnitude^2
      0044                       56          nxtloc   equ     xirk    :long   ; Next location in table
                                 57
      003C                       58          WRP      equ     xrrk    :word   ; Multiplication factor, Real
      003E                       59          WIP      equ     xrrk+2  :word   ; Multiplication factor, Imaginary
      0040                       60          PWR      equ     xrrnk   :word
      0042                       61          IN_CNT   equ     xrrnk+2 :word
      0044                       62          NDIV2    equ     xirk    :word   ; n divided by 2  (0 < n < N) *2
                                 63
    004C                         64          KPTR:            dsw     1       ; K for counter *2 to index words
    004E                         65          KN2:             dsw     1       ; KPTR + NDIV2
    0050                         66          N_SUB_K:         dsw     1       ; N-K *2 to index words
    0052                         67          RK:              dsw     1       ; Bit reversed pointer of KPTR
    0054                         68          RNK:             dsw     1       ; Bit reversed pointer of N_SUB_K
    0056                         69          SHFT_CNT:        dsw     1
    0058                         70          LOOP_CNT:        dsb     1
      004E                       71          ptr      equ     kn2     :word   ; Pointer for square root table
    0000                         72  DSEG
                                 73
                                 74  EXTRN   FFT_MODE         ; FFT_MODE: mode for FFT input and graphing
                                 75  EXTRN   XREAL, XIMAG     ; XREAL, XIMAG: Base  addresses for 32 16-bit signed
                                 76                           ; entries for real and imaginary numbers respectively.
                                 77  EXTRN   FFT_OUT          ; FFT_OUT: Starting address for 32 word array
                                 78                           ; of magnitude information.
                                 79
    0000                         80          OUTR:    dsw     32      ; Real component of fft
    0040                         81          OUTI:    dsw     32      ; Imaginary component of waveform
                                 82   PUBLIC OUTR,OUTI
                                 83
                                 84  $EJECT
```

270189-34

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                    02/18/86          PAGE    3

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                 85
    2280                         86  CSEG at 2280H
                                 87
                                 88   PUBLIC fft_calc        ; Starting point for FFT algorithm
                                 89
                                 90   EXTRN  scale_factor    ; Shift factor used to prevent overflow when averaging
                                 91                          ; fft outputs
                                 92
                                 93
                                 94  ;                                ;;;;    START FOURIER CALCULATIONS
    2280                         95  FFT_CALC:                        ;;;;    400  ' INITIALIZATION OF LOOP
    2280 1100              E     96          clrb    error
    2282 B10100            E     97          ldb     portl,#00000001b                    ;**** Indication Only
                                 98
    2285 FC                      99          clrvt
    2286 B10158                 100          ldb     loop_cnt,#1
    2289 B10456                 101          ldb     shft_cnt,#4
    228C A1200044               102          ld      ndiv2,#32
                                103  ;                                ;;;;    410 K=0
    2290                        104  OUT_LOOP:
    2290 950400            E    105          xorb    portl,#00000100B                    ;****  Indication Only
    2293 014C                   106          clr     kptr
                                107  ;                                ;;;;    420 IF LOOP > EXP THEN 700
    2295 990558                 108          cmpb    loop_cnt, #5    ; 32=2^5
    2298 DA0220A3               109  !       bgt     UNWEAVE
                                110
                                111
    229C                        112  MID_LOOP:                        ;;;;    430 INCNT=0
    229C 0142                   113          clr     in_cnt
                                114
                                115                                  ;;;;    440  ' CALCULATIONS BEGIN HERE
    229E                        116  IN_LOOP:
    229E 65020042               117          add     in_cnt,#2       ;;;;    450 INCNT=INCNT+1
                                118  ;                                ;;;;    460 P=BR(INT(K/(2^SHIFT)))
    22A2 A04C40                 119          ld      pwr,kptr
    22A5 085640                 120          shr     pwr,shft_cnt            ;; Calculate multiplication factors
    22A8 71FE40                 121          andb    pwr,#11111110B
    22AB A341003840             122          ld      pwr,brev[pwr]
                                123  ;                                ;;;;    470 WRP=WR(P) : WIP=WI(P) : KN2=K+N2
    22B0 A34144393C             124  gw:     ld      wrp,wr[pwr]
    22B5 A34186393E             125          ld      wip,wi[pwr]
    22BA 44444C4E               126          add     kn2,kptr,ndiv2
                                127  $eject
```

270189-35

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                   02/18/86          PAGE    4

ERR LOC  OBJECT                  LINE        SOURCE STATEMENT
                                  128           ;; Complex multiplication follows
                                  129
                                  130   ;                                 ;;;;    480 TMPR= (WRP*XR(KN2) - WIP*XI(KN2))/2
    22BE FE4F4F00003C24   E       131   gm:     mul     tmpr,wrp,xreal[kn2]
    22C5 FE4F4F00003E28   E       132           mul     tmpi,wip,ximag[kn2]
    22CC 682A26                   133           sub     tmpr+2,tmpi+2
                                  134   ;                                 ;;;;    490 TMPI= (WRP*XI(KN2) + WIP*XR(KN2))/2
    22CF FE4F4F00003C2C   E       135           mul     tmprl,wrp,ximag[kn2]
    22D6 FE4F4F00003E28   E       136           mul     tmpi,wip,xreal[kn2]
    22DD 642E2A                   137           add     tmpi+2,tmprl+2
                                  138
                                  139                   ;;  using the high byte only of a signed multiply
                                  140                   ;;  provides an effective divide by two
                                  141
    22E0 DC55                     142           BVT     ERR1    ; Branch on error in complex multiplications
                                  143
    22E2 A34D00002C       E       144           ld      tmprl,xreal[kptr]       ;;;;    500 TMPR1=XR(K)/2 :
    22E7 0A012C                   145           shra    tmprl,#1                ;;;;        TMPI1=XI(K)/2
    22EA A34D000030       E       146           ld      tmpil,ximag[kptr]
    22EF 0A0130                   147           shra    tmpil,#1
                                  148
                                  149   ;                                 ;;;;    510 XR(KN2) = TMPR1 - TMPR
    22F2 48262C34                 150   gr2:    sub     xrtmp,tmprl,tmpr+2
    22F6 C34F000034       E       151           st      xrtmp,xreal[kn2]
                                  152   ;                                 ;;;;    520 XI(KN2) = TMPI1 - TMPI
    22FB 482A3038                 153   gx2:    sub     xitmp,tmpil,tmpi+2
    22FF C34F000038       E       154           st      xitmp,ximag[kn2]
                                  155   ;                                 ;;;;    530 XR(K) = TMPR1 + TMPR
    2304 44262C34                 156           add     xrtmp,tmprl,tmpr+2
    2308 C34D000034       E       157           st      xrtmp,xreal[kptr]
                                  158   ;                                 ;;;;    540 XI(K) = TMPI1 + TMPI
    230D 442A3038                 159   gx:     add     xitmp,tmpil,tmpi+2
    2311 C34D000038       E       160           st      xitmp,ximag[kptr]
                                  161
    2316 DC23                     162           BVT     ERR2    ; Branch on error in complex additions
                                  163
                                  164   $eject
```

270189-36

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                    02/18/86          PAGE    5

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                165  ;                                ;;;;    560 K=K+1
    2318 6502004C               166  ik:     add     kptr,#2
                                167
                                168  ;                                ;;;;    570 IF INCNT<N2 THEN  GOTO 450
    231C 884442                 169          cmp     in_cnt,ndiv2
    231F D602277B               170  !       blt     IN_LOOP
                                171
                                172  ;                                ;;;;    580 K=K+N2
    2323 64444C                 173          add     kptr,ndiv2
                                174  ;                                ;;;;    590 IF K<N1 THEN GOTO 430
    2326 893E004C               175          cmp     kptr,#62
    232A D602276E               176  !       blt     MID_LOOP
                                177
                                178                                   ;;;;    600 LOOP=LOOP+1 : N2=N2/2
    232E 1758                   179          incb    loop_cnt         ;;;;    605 SHIFT=SHIFT+1
    2330 0A0144                 180          shra    ndiv2,#1
    2333 1556                   181          decb    shft_cnt
                                182  ;                                ;;;;    610 GOTO 400
    2335 2759                   183          br      OUT_LOOP
                                184
                                185
    2337 B10100           E     186  ERR1:   ldb     error,#01       ; overflow error, 1st set of calculations
    233A F0                     187          ret
    233B B10200           E     188  ERR2:   ldb     error,#02       ; overflow error, 2nd set of calculations
    233E F0                     189          ret
                                190
                                191  $EJECT
```

270189-37

Listing 2—ASM96 FFT Program (Continued)

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                    02/18/86          PAGE    6

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
                              192
                              193  ;                        ;;;;    700 ' POST-PROCESING AND REORDERING STARTS HERE
    233F                      194  UNWEAVE:
    233F B10200           E   195          ldb     portl,#00000010b                        ;****
                              196
                              197  ;                                ;;;;    720 FOR K=0 TO 31
    2342 014C                 198          clr     kptr
    2344 A1400050             199          ld      n_sub_k,#64
    2348                      200  UN_LOOP:
                              201  ;                                ;;;;    740 XIBRK=XI(BR(K)) : XRBRK=XR(BR(K)
    2348 A34D003852           202          ld      rk,brev[kptr]
    234D A35300003C       E   203          ld      xrrk,xreal[rk]
    2352 063C                 204          ext     xrrk
    2354 A353000044       E   205          ld      xirk,ximag[rk]
    2359 0644                 206          ext     xirk
                              207  ;                                ;;;;    750 XIBRNK=XI(BR(N-K):XRBRNK=XR(BR(N-K)
    235B A351003854           208          ld      rnk,brev[n_sub_k]
    2360 A355000040       E   209          ld      xrrnk,xreal[rnk]
    2365 0640                 210          ext     xrrnk
    2367 A355000048       E   211          ld      xirnk,ximag[rnk]
    236C 0648                 212          ext     xirnk
                              213  ;                                ;;;;    760 TI=(XIBRK + XIBRNK)/2
    236E 44484428             214  ar:      add     tmpi,xirk,xirnk
    2372 A04A2A               215          ld      tmpi+2,xirnk+2
    2375 A4462A               216          addc    tmpi+2,xirk+2
    2378 0E0128               217          shral   tmpi,#1          ; 16 bit result in tmpi
                              218
                              219                                   ;;;;    770 TR=(XRBRK - XRBRNK)/2
    237B 48403C24             220          sub     tmpr,xrrk,xrrnk
    237F A03E26               221          ld      tmpr+2,xrrk+2
    2382 A84226               222          subc    tmpr+2,xrrnk+2
    2385 0E0124               223          shral   tmpr,#1          ; 16 bit result in tmpr
                              224
                              225                                   ;;;;    780 XRT= (XRBRK + XRBRNK)/4
    2388 44403C34             226          add     xrtmp,xrrk,xrrnk
    238C A03E36               227          ld      xrtmp+2,xrrk+2
    238F A44236               228          addc    xrtmp+2,xrrnk+2
    2392 0D0E34               229          shll    xrtmp,#14         ; 32 bit result in xrtmp
                              230
                              231                                   ;;;;    790 XIT= (XIBRK-XIBRNK)/4
    2395 48484438             232          sub     xitmp,xirk,xirnk
    2399 A0463A               233          ld      xitmp+2,xirk+2
    239C A84A3A               234          subc    xitmp+2,xirnk+2
    239F 0D0E38               235          shll    xitmp,#14         ; 32 bit result in xitmp
                              236
                              237  $eject
```

270189-38

```
ERR LOC  OBJECT                LINE      SOURCE STATEMENT
                                238  ;       ;;;;;                   Multiply will provide effective divide by 2
                                239
                                240  ;                       ;;;;    800 OUTR= (XRT + TI*COSFN(K)/2 - TR*SINFN(K)/2)
                                241
    23A2 FE4F4D4038242C         242  mr:      mul     tmprl,tmpr,sinfn[kptr]
    23A9 FE4F4DC2382830         243          mul     tmpil,tmpi,cosfn[kptr]
    23B0 643034                 244          add     xrtmp,tmpil
    23B3 A43236                 245          addc    xrtmp+2,tmpil+2
    23B6 682C34                 246          sub     xrtmp,tmprl
    23B9 A82E36                 247          subc    xrtmp+2,tmprl+2
    23BC C34D000036       R     248          st      xrtmp+2,outr[kptr]      ;; OUTR = Real Output Values
                                249
                                250
                                251  ;                       ;;;;    810 OUTI= (XIT - TI*SINFN(K)/2 - TR*COSFN(K)/2)
                                252
    23C1 FE4F4DC238242C         253  mi:      mul     tmprl,tmpr,cosfn[kptr]
    23C8 FE4F4D40382830         254          mul     tmpil,tmpi,sinfn[kptr]
    23CF 683038                 255          sub     xitmp,tmpil
    23D2 A8323A                 256          subc    xitmp+2,tmpil+2
    23D5 682C38                 257          sub     xitmp,tmprl
    23D8 682E3A                 258          sub     xitmp+2,tmprl+2
    23DB C34D40003A       R     259          st      xitmp+2,outi[kptr]      ;; OUTI = Imaginary Output values
                                260
                                261
                                262                                  ;;;;    830 MAG =SQR(OUTR*OUTR + OUTI*OUTI)
                                263
    23E0                        264  GET_MAG:                                ;; Get Magnitude of Vector
    23E0 A03624                 265          ld      tmpr,xrtmp+2
    23E3 A03A28                 266          ld      tmpi,xitmp+2
                                267
    23E6 FE6C2424               268          mul     tmpr,tmpr               ; tmpr = tmpi**2 + tmpr**2
    23EA FE6C2828               269          mul     tmpi,tmpi
    23EE 642824                 270          add     tmpr,tmpi
    23F1 A42A26                 271          addc    tmpr+2,tmpi+2
                                272
    23F4 32004C           E     273          bbc     FFT_MODE,2,CALC_SQRT
                                274
                                275  $eject
```

270189-39

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                    02/18/86          PAGE    8

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               276
                               277                ;;;;  ***  CALCULATE 10 log magnitude^2  ***
                               278   ; Output = 512*10*LOG(x)  x=1,2,3 ... 64K
                               279
    23F7                       280   CALC_LOG:
    23F7 0156                  281           clr     shft_cnt
    23F9 0F5624                282           norml   tmpr,shft_cnt   ; Normalize and get normalization factor
    23FC 990F56                283           cmpb    shft_cnt,#15
    23FF DA04                  284           jle     LOG_IN_RANGE    ;  Jump if SHIFT_CNT <= 15
                               285
    2401 0140                  286           clr     log
    2403 202C                  287           br      LOG_STORE
                               288
    2405                       289   LOG_IN_RANGE:
    2405 44565656              290           add     shft_cnt,shft_cnt,shft_cnt      ; Make shift_cnt a pointer
                               291
    2409 AC274E                292           ldbze   ptr,tmpr+3      ; Most significant byte is table pointer
    240C 444E4E4E              293           add     ptr,ptr,ptr     ;
    2410 65083A4E              294           add     ptr,# LOG_TABLE-256     ; ptr= Table + offset  (offset=tmpr+3)
                               295                                   ; Use -256 since tmpr+3 is always >= 128
    2414 A24F40                296           ld      log, [ptr]+
    2417 A24E44                297           ld      nxtloc, [ptr]           ;; Linear Interpolation
                               298
    241A 684044                299           sub     nxtloc,log      ; nxtloc = next log - log
                               300
    241D AC263C                301           ldbze   diff,tmpr+2     ; diff+1 = nxtloc * tmpr+2 / 256
    2420 6C443C                302           mulu    diff,nxtloc
                               303
    2423 0C083C                304           shrl    diff,#8         ; log = log + diff/256
    2426 643C40                305           add     log,diff
    2429 080540                306           shr     log,#5          ; 8192/32 * 20LOG(x) = 256 * 20LOG(x)
                               307
    242C A7570A3C40            308           addc    log,log_offset[shft_cnt]        ; add log of normalization factor
                               309
                               310                   ;; Log (M*N) = Log M + Log N
                               311
    2431                       312   LOG_STORE:
    2431 080040          E     313           shr     log,#SCALE_FACTOR
    2434 A40040          E     314           addc    log,zero                ; Divide to prevent overflow during
    2437 674D000040      E     315           add     log,FFT_OUT[kptr]       ; averaging of outputs
    243C C34D000040      E     316           st      log,FFT_OUT[kptr]
                               317
    2441 2045                  318           BR      ENDL
                               319   $eject
```

270189-40

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                    02/18/86          PAGE    9

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                320                          ;;;;  ***  CALCULATE SQUARE  ROOT  ***
    2443                        321   CALC_SQRT:
                                322
    2443 0156                   323   clr     shft_cnt
    2445 0F5624                 324         norml   tmpr,shft_cnt   ; Normalize and get normalization factor
                                325
    2448 D705                   326         jne     SQRT_IN_RANGE   ;  Jump if tmpr > 0
    244A C04200           E     327         st      zero,sqrt+2
    244D 2029                   328         br      SQRT_STORE
                                329
    244F                        330   SQRT_IN_RANGE:
    244F AC274E                 331         ldbze   ptr,tmpr+3      ; Most significant byte is table pointer
    2452 444E4E4E               332         add     ptr,ptr,ptr
    2456 6508394E               333         add     ptr,# SQ_TABLE-256      ; ptr= Table + offset  (offset=tmpr+3)
                                334                                 ; Use -256 since tmpr+3 is always >= 128
    245A A24F40                 335         ld      sqrt, [ptr]+
    245D A24E44                 336         ld      nxtloc, [ptr]           ;; Linear Interpolation
                                337
    2460 684044                 338         sub     nxtloc,sqrt     ; nxtloc =  sqrt - next sqrt
                                339
    2463 AC263C                 340         ldbze   diff,tmpr+2     ; diff+1 = nxtloc * tmpr+2 / 256
    2466 6C443C                 341         mulu    diff,nxtloc
                                342
    2469 AC3D3C                 343         ldbze   diff,diff+1     ; sqrt = sqrt + delta   (diff < 0FFH)
    246C 643C40                 344         add     sqrt,diff
                                345
    246F 44565656               346         add     shft_cnt,shft_cnt,shft_cnt
                                347
    2473 6F57C83940             348         mulu    sqrt,tab_sqr[shft_cnt]  ; divide by normalization factor
                                349
                                350                                 ;; mulu acts as divide since if tab2=0FFFFH
                                351                                 ;; sqrt would remain essentialy unchanged
    2478                        352   SQRT_STORE:
    2478 080042           E     353         shr     sqrt+2,#SCALE_FACTOR
    247B A40042           E     354         addc    sqrt+2,zero             ; Divide to prevent overflow during
    247E 674D000042       E     355         add     sqrt+2,FFT_OUT[kptr]    ; averaging of outputs
    2483 C34D000042       E     356         st      sqrt+2,FFT_OUT[kptr]
                                357
                                358   ;             ;;;;    ***  END  OF  LOOP  ***
                                359
                                360                                 ;;;;    950 NEXT K
    2488 6502004C               361   ENDL:   add     kptr,#2
    248C 69020050               362         sub     n_sub_k,#2
    2490 DF0226B4               363   !       bne     UN_LOOP
                                364
    2494 F0                     365         RET
                                366   $eject
```

270189-41

Listing 2—ASM96 FFT Program (Continued)

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                          02/18/86          PAGE   10

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                                367   ;$nolist
    3800                        368           CSEG  AT  3800H          ;;;;    Use 2k for tables
                                369
    3800                        370   BREV:   ; 2*bit reversal value
                                371
    3800 0000200010003000       372   DCW     2*0, 2*16, 2*8,  2*24, 2*4, 2*20, 2*12, 2*28
    3810 0400240014003400       373   DCW     2*2, 2*18, 2*10, 2*26, 2*6, 2*22, 2*14, 2*30
    3820 0200220012003200       374   DCW     2*1, 2*17, 2*9,  2*25, 2*5, 2*21, 2*13, 2*29
    3830 0600260016003600       375   DCW     2*3, 2*19, 2*11, 2*27, 2*7, 2*23, 2*15, 2*31
                                376
    3840                        377   SINFN:
    3840 00008C0CF9182825       378   DCW          0,   3212,   6393,   9512,  12539,  15446,  18204,  20787
    3850 825AF1626D6AE270       379   DCW      23170,  25329,  27245,  28898,  30273,  31356,  32137,  32609
    3860 FF7F617F897D7C7A       380   DCW      32767,  32609,  32137,  31356,  30273,  28898,  27245,  25329
    3870 825A33511C47563C       381   DCW      23170,  20787,  18204,  15446,  12539,   9512,   6393,   3212
    3880 000074F307E7D8DA       382   DCW          0,  -3212,  -6393,  -9512, -12539, -15446, -18204, -20787
    3890 7EA50F9D93951E8F       383   DCW     -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
    38A0 01809F8077828485       384   DCW     -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
    38B0 7EA5CDAEE4B8AAC3       385   DCW     -23170, -20787, -18204, -15446, -12539,  -9512,  -6393,  -3212
    38C0 0000                   386   DCW          0
                                387
    38C2                        388   COSFN:
    38C2 FF7F617F897D7C7A       389   DCW      32767,  32609,  32137,  31356,  30273,  28898,  27245,  25329
    38D2 825A33511C47563C       390   DCW      23170,  20787,  18204,  15446,  12539,   9512,   6393,   3212
    38E2 000074F307E7D8DA       391   DCW          0,  -3212,  -6393,  -9512, -12539, -15446, -18204, -20787
    38F2 7EA50F9D93951E8F       392   DCW     -23170, -25329, -27245, -28898, -30273, -31356, -32137, -32609
    3902 01809F8077828485       393   DCW     -32767, -32609, -32137, -31356, -30273, -28898, -27245, -25329
    3912 7EA5CDAEE4B8AAC3       394   DCW     -23170, -20787, -18204, -15446, -12539,  -9512,  -6393,  -3212
    3922 00008C0CF9182825       395   DCW          0,   3212,   6393,   9512,  12539,  15446,  18204,  20787
    3932 825AF1626D6AE270       396   DCW      23170,  25329,  27245,  28898,  30273,  31356,  32137,  32609
    3942 FF7F                   397   DCW     32767
                                398
    3944                        399   WR:             ;;;;    WR = COS(K*2PI/N)
    3944 FF7F897D41766D6A       400   DCW      32767,  32137,  30273,  27245,  23170,  18204,  12539,   6393
    3954 000007E705CFE4B8       401   DCW          0,  -6393, -12539, -18204, -23170, -27245, -30273, -32137
    3964 01807782BF899395       402   DCW     -32767, -32137, -30273, -27245, -23170, -18204, -12539,  -6393
    3974 0000F918FB301C47       403   DCW          0,   6393,  12539,  18204,  23170,  27245,  30273,  32137
    3984 FF7F                   404   DCW     32767
                                405
    3986                        406   WI:             ;;;;    WI = -SIN(K*2PI/N)
    3986 000007E705CFE4B8       407   DCW         -0,  -6393, -12539, -18204, -23170, -27245, -30273, -32137
    3996 01807782BF899395       408   DCW     -32767, -32137, -30273, -27245, -23170, -18204, -12539,  -6393
    39A6 0000F918FB301C47       409   DCW          0,   6393,  12539,  18204,  23170,  27245,  30273,  32137
    39B6 FF7F897D41766D6A       410   DCW      32767,  32137,  30273,  27245,  23170,  18204,  12539,   6393
    39C6 0000                   411   DCW          0
                                412   $eject
```

270189-42

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                          02/18/86          PAGE   11

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                               413
                               414
    39C8                       415   TAB_SQR:        ; 65535/(square root of 2**SHFT_CNT) ;  0<=SHFT_CNT<32
                               416
                               417     ;;    1       2       4       8       16      32      64      128
    39C8 FFFF04B50080825A      418   DCW     65535,  46340,  32768,  23170,  16384,  11585,  8192,   5793
                               419
                               420     ;;    256     512     1024    2048    4096    8192    16384   32768
    39D8 0010500B0008A805      421   DCW     4096,   2896,   2048,   1448,   1024,   724,    512,    362
                               422
                               423     ;;    65536,  131072, 262144, 524288, ...
    39E8 0001B50080005B00      424   DCW     256,    181,    128,    91,     64,     45,     32,     23
    39F8 10000B0008000600      425   DCW     16,     11,     8,      6,      4,      3,      2,      1
                               426
                               427
    3A08                       428   SQ_TABLE:       ; square root of n * 2**24  N=128, 129, 130 ... 255
                               429
    3A08 05B5BAB56EB621B7      430   DCW       46341, 46522, 46702, 46881, 47059, 47237, 47415, 47591
    3A18 97BA46BBF5BBA3BC      431   DCW       47767, 47942, 48117, 48291, 48465, 48637, 48809, 48981
    3A28 00C0AAC054C1FDC1      432   DCW       49152, 49322, 49492, 49661, 49830, 49998, 50166, 50332
    3A38 43C5E9C58EC633C7      433   DCW       50499, 50665, 50830, 50995, 51159, 51323, 51486, 51649
    3A48 63CA04CBA6CB46CC      434   DCW       51811, 51972, 52134, 52294, 52454, 52614, 52773, 52932
    3A58 62CF00D09DD03AD1      435   DCW       53090, 53248, 53405, 53562, 53719, 53874, 54030, 54185
    3A68 44D4DED477D511D6      436   DCW       54340, 54494, 54647, 54801, 54954, 55106, 55258, 55410
    3A78 09D9A0D936DACCDA      437   DCW       55561, 55712, 55862, 56012, 56162, 56311, 56459, 56608
    3A88 B4DD47DEDBDE6EDF      438   DCW       56756, 56903, 57051, 57198, 57344, 57490, 57636, 57781
    3A98 46E2D7E267E3F7E3      439   DCW       57926, 58071, 58215, 58359, 58503, 58646, 58789, 58931
    3AA8 C1E64FE7DDE76AE8      440   DCW       59073, 59215, 59357, 59498, 59639, 59779, 59919, 60059
    3AB8 27EBB2EB3DECC7EC      441   DCW       60199, 60338, 60477, 60615, 60754, 60891, 61029, 61166
    3AC8 77EF00F088F010F1      442   DCW       61303, 61440, 61576, 61712, 61848, 61984, 62119, 62254
    3AD8 B4F33BF4C1F446F5      443   DCW       62388, 62523, 62657, 62790, 62924, 63057, 63190, 63323
    3AE8 DFF763F8E7F86AF9      444   DCW       63455, 63587, 63719, 63850, 63982, 64113, 64243, 64374
    3AF8 F8FB7AFCFBFC7DFD      445   DCW       64504, 64634, 64763, 64893, 65022, 65151, 65280, 65408
                               446
                               447   $eject
```

270189-43

Listing 2—ASM96 FFT Program (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_RUN                                          02/18/86          PAGE   12

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                448
    3B08                        449   LOG_TABLE:  ;   16384*10*LOG(n/128)     n=128,129,130 ... 256
                                450
    3B08 00002A024F047006       451   DCW           0,   554,  1103,  1648,  2190,  2727,  3260,  3789
    3B18 DA10E312E914EA16       452   DCW        4314,  4835,  5353,  5866,  6376,  6883,  7386,  7885
    3B28 BD20A92292247826       453   DCW        8381,  8873,  9362,  9848, 10330, 10810, 11286, 11758
    3B38 C42F973166333335       454   DCW       12228, 12695, 13158, 13619, 14076, 14531, 14983, 15432
    3B48 063EC13F7A413043       455   DCW       15878, 16321, 16762, 17200, 17635, 18067, 18497, 18925
    3B58 954B3C4DDF4E8150       456   DCW       19349, 19772, 20191, 20609, 21024, 21436, 21846, 22254
    3B68 8458175AA85B365D       457   DCW       22660, 23063, 23464, 23862, 24259, 24653, 25045, 25435
    3B78 DE646066E0675D69       458   DCW       25822, 26208, 26592, 26973, 27353, 27730, 28106, 28479
    3B88 B370247294730275       459   DCW       28851, 29220, 29588, 29954, 30318, 30680, 31040, 31399
    3B98 0B7C6E7DCF7E2F80       460   DCW       31755, 32110, 32463, 32815, 33165, 33512, 33859, 34203
    3BA8 F28647889B89ED8A       461   DCW       34546, 34887, 35227, 35565, 35902, 36236, 36570, 36901
    3BB8 7091B892FF934595       462   DCW       37232, 37560, 37887, 38213, 38537, 38860, 39181, 39501
    3BC8 8B9BC89C049E3E9F       463   DCW       39819, 40136, 40452, 40766, 41079, 41390, 41700, 42009
    3BD8 4CA57EA6AFA7DEA8       464   DCW       42316, 42622, 42927, 43230, 43533, 43833, 44133, 44431
    3BE8 B9AEE0AF07B12CB2       465   DCW       44729, 45024, 45319, 45612, 45905, 46196, 46486, 46774
    3BF8 D6B7F4B811BA2DBB       466   DCW       47062, 47348, 47633, 47917, 48200, 48482, 48763, 49042
    3C08 A9C0                   467   DCW       49321
                                468
    3C0A                        469   LOG_OFFSET:     ; 512*10*LOG(2**(15-n))    n= 0,1,2,3 ... 15
                                470                     ; 512*10*LOG(0.5)          n= 16,17,18 ... 31
                                471
    3C0A 4F5A4A54454E3F48       472   DCW       23119, 21578, 20037, 18495, 16954, 15413, 13871, 12330
    3C1A 252A20241A1E1518       473   DCW       10789,  9248,  7706,  6165,  4624,  3083,  1541,     0
                                474
    3C2A                        475   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-44

Listing 2—ASM96 FFT Program (Continued)

The BASIC program is used as comments in the ASM96 program. Some of the variables in the ASM96 program have slightly different names than their counter-parts in the BASIC program. This was to make the comments fit into the ASM96 code. Highlights in this section of code are a table driven square root routine and log conversion routine which can easily be adapted for use by any program.

Both the square root routine and the log conversion routine use the 32-bit value in the variable TMPR. The square root routine calculates the square root of that value in the variable SQRT+2, a 16-bit variable. In this program, the square root value is averaged and stored in a table.

The log conversion routine divides the value in TMPR by 65536 (2^{16}) and uses table lookup to provide the common log. The result is a 16-bit number with the value 512 * 10 Log (TMPR/65536) stored in the variable LOG. This calculation is used to present the results of the FFT in decibels instead of magnitude. With an input of 63095, the output is 512*48 dB. The graph program, (Section 10), prints the output value of the plot as INPUT/512 dB.

The following descriptions of the ASM code point out some of the highlights and not-so-obvious coding:

Lines 1-104 initialize the code and declare variables. The input and output arrays of the program are declared external. Note that many of the registers are overlayable, use caution when implementing this routine with others with overlayable registers.

Lines 116–124 calculate the power of W to be used. Note that KPTR is always incremented by 2. The multiple right shift followed by the AND mask creates an even address and the indirect look to the BR (Bit Reversal) table quickly calculates the power PWR.

Lines 130-138 perform the complex multiplications. Since WIP and WRP range from −32767 to +32767, the multiplication is easy to handle. The automatic divide by two which occurs when using the upper word only of the 32-bit result is a feature in this case.

Lines 144-163 use right shifts for a fast divide, then add or subtract the desired variables and store them in the array. Note that the upper word of TMPR and TMPI is used, and the same array is used for both the input and output of the operations.

Lines 165-189 update the loop variables and then check for errors on the complex multiplications and additions. If there are no overflows at this time the data will run smoothly through the rest of the program.

Lines 200-212 load variables with values based on the bit reversed values of pointers.

Lines 214-236 perform additions and subtractions to prepare for the next set of formulas. Note that XITMP and XRTMP are 32-bit values.

Lines 240-260 perform multiplies and summations resulting in 32-bit variables. This saves a bit or two of accuracy. The upper words are then stored as the results.

Lines 263-272 generate the squared magnitude of the harmonic component as a 32-bit value.

Lines 278-310 calculate 10 Log (TMPR/65536). The 32-bit register TMPR is divided by 65536 so that the output range would be reasonable.

First, the number is normalized. (It is shifted left until a 1 is in the most significant bit, the number of shifts required is placed in SHFT_CNT.) If it had to be shifted more than 15 times the output is set to zero.

Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

The 16-bit result of this table look-up and interpolation is then added to the Log of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 290 bytes of table space.

Lines 321-357 calculate the square root of the 32-bit register TMPR using a table look-up approach.

First, the number is normalized. Next, the most significant BYTE is used as a reference for the look-up table, providing a 16-bit result. The next most significant BYTE is then used to perform linear interpolation between the referenced table value and the one above it. The interpolated value is added to the directly referenced one.

The 16-bit result of this table look-up and interpolation is then divided by the square root of the normalization factor, which is also stored in a table. This table look-up approach works fast and only uses 320 bytes of table space. The results are valid to near 14-bits, more than enough for the FFT algorithm.

Lines 352-360 average the magnitude value, if multiple passes are being performed, and then store the value in the array. The loop-counters are incremented and the process repeats itself.

This concludes the FFT routine. In order to use it, it must be called from a main program. The details for calling this routine are covered in the next section.

8.0 BACKGROUND CONTROL PROGRAM

The main routine is shown in Listing 3. It begins with declarations that can be used in almost any program. Note that these are similar, but not identical, to other 8096 include files that have been published. Comments on controlling the Analog to Digital converter routine follow the declarations.

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                  02/18/86          PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:FTMAIN.A96
OBJECT FILE: :F2:FTMAIN.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                  1   $pagelength(50)
                                  2
                                  3   FFT_MAIN_APNOTE MODULE MAIN, STACKSIZE(6)
                                  4
                                  5   ; Intel Corporation, January 24, 1986
                                  6   ; by Ira Horden, MCO Applications
                                  7
                                  8
                                  9   ;    This program performs an FFT on real data and plots it on a printer.
                                 10   ; It uses the program modules A2DCON, PLOTSP, and FFTRUN.  The adjustable
                                 11   ; parameters of each of the programs are set by this main module.
                                 12
                                 13
                                 14   $INCLUDE (:F0:DEMO96.INC)       ; Include SFR definitions
                           =1    15   ;$nolist  ;  Turn listing off for include file
                           =1    16
                           =1    17   ;*******************************************************************************
                           =1    18   ;
                           =1    19   ;               Copyright 1985, Intel Corporation
                           =1    20   ;               October 28,1985
                           =1    21   ;               by Ira Horden, MCO Applications
                           =1    22   ;
                           =1    23   ; DEMO96.INC - DEFINITION OF SYMBOLIC NAMES FOR THE I/O REGISTERS OF THE 8096
                           =1    24   ;
                           =1    25   ;*******************************************************************************
                           =1    26   ;
0000                       =1    27   ZERO            EQU   00h:WORD      ; R/W   Zero Register
0002                       =1    28   AD_COMMAND      EQU   02H:BYTE      ;   W   A to D command register
0002                       =1    29   AD_RESULT_LO    EQU   02H:BYTE      ; R     Low byte of result and channel
0003                       =1    30   AD_RESULT_HI    EQU   03H:BYTE      ; R     High byte of result
0003                       =1    31   HSI_MODE        EQU   03H:BYTE      ;   W   Controls HSI transition detector
0004                       =1    32   HSO_TIME        EQU   04H:WORD      ;   W   HSI time tag
0004                       =1    33   HSI_TIME        EQU   04H:WORD      ; R     HSO time tag
0006                       =1    34   HSO_COMMAND     EQU   06H:BYTE      ;   W   HSO command tag
0006                       =1    35   HSI_STATUS      EQU   06H:BYTE      ; R     HSI status register (reads fifo)
0007                       =1    36   SBUF            EQU   07H:BYTE      ; R/W   Serial port buffer
0008                       =1    37   INT_MASK        EQU   08H:BYTE      ; R/W   Interrupt mask register
0009                       =1    38   INT_PENDING     EQU   09H:BYTE      ; R/W   Interrupt pending register
0011                       =1    39   SPCON           EQU   11H:BYTE      ;   W   Serial port control register
0011                       =1    40   SPSTAT          EQU   11H:BYTE      ; R     Serial port status register
000A                       =1    41   WATCHDOG        EQU   0AH:BYTE      ;   W   Watchdog timer
```

270189-45

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                  02/18/86          PAGE    2

ERR LOC  OBJECT              LINE       SOURCE STATEMENT
      000A                  =1   42   TIMER1           EQU   0AH:WORD      ; R     Timer1 register
      000C                  =1   43   TIMER2           EQU   0CH:WORD      ; R     Timer2 register
      000E                  =1   44   PORT0            EQU   0EH:BYTE      ; R     I/O port 0
      000E                  =1   45   BAUD_REG         EQU   0EH:BYTE      ;   W   Baud rate register
      000F                  =1   46   PORT1            EQU   0FH:BYTE      ; R/W   I/O port 1
      0010                  =1   47   PORT2            EQU   10H:BYTE      ; R/W   I/O port 2
      0015                  =1   48   IOC0            EQU   15H:BYTE      ;   W   I/O control register 0
      0015                  =1   49   IOS0            EQU   15H:BYTE      ; R     I/O status register 0
      0016                  =1   50   IOC1            EQU   16H:BYTE      ;   W   I/O control register 1
      0016                  =1   51   IOS1            EQU   16H:BYTE      ; R     I/O status register 1
      0017                  =1   52   PWM_CONTROL     EQU   17H:BYTE      ;   W   PWM control register
      0018                  =1   53   SP              EQU   18H:WORD      ; R/W   System stack pointer
                            =1   54
      000D                  =1   55   CR              EQU   0DH
      000A                  =1   56   LF              EQU   0AH
                            =1   57
                            =1   58   PUBLIC ZERO,AD_COMMAND,AD_RESULT_LO,AD_RESULT_HI,HSI_MODE,HSO_TIME,HSI_TIME
                            =1   59   PUBLIC HSO_COMMAND
                            =1   60   PUBLIC HSI_STATUS,SBUF,INT_MASK,INT_PENDING,WATCHDOG,TIMER1,TIMER2
                            =1   61   PUBLIC BAUD_REG, PORT0, PORT1, PORT2,SPSTAT,SPCON,IOC0,IOC1,IOS0,IOS1
                            =1   62   PUBLIC PWM_CONTROL,SP,CR,LF
                            =1   63
      001C                  =1   64   RSEG at 1CH
                            =1   65
      001C                  =1   66           AX:     DSW     1           ; Temp registers used in conformance
      001E                  =1   67           DX:     DSW     1           ; with PLM-96(tm) conventions.
      0020                  =1   68           BX:     DSW     1
      0022                  =1   69           CX:     DSW     1
                            =1   70
        001C                =1   71           AL      EQU     AX      :BYTE
        001D                =1   72           AH      EQU     (AX+1)  :BYTE
        0020                =1   73           BL      EQU     BX      :BYTE
                            =1   74
                            =1   75    public ax, bx, cx, dx, al, ah, bl
                            =1   76
                            =1   77   $list    ;  Turn listing back on
                            =1   78           ;  End of include file
                                 79
                                 80   ; A2D UTILITY COMMANDS/RESPONSES FOR "CONTROL_A2D" ____________________________
                                 81
        0007                     82   busy            equ     7
        0010                     83   con_b0          equ     00010000b;convert to BUFF0
        0028                     84   dump_b0_p_s     equ     00101000b; download  BUFF0  as PAIRED SIGNED data
                                 85   ;_____________________________________________________________________________
                                 86
        0001                     87   AVR_NUM         equ     1       ; Number of times to average the waveform
                                 88                                   ;    AVR_NUM < 256
```

270189-46

```
ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 89
      0000                       90  SCALE_FACTOR    equ     0       ; Number of rights shifts performed on
                                 91                                  ; output of FFT.  Used to prevent overflow
                                 92                                  ; on summation
                                 93
      0100                       94  PLOT_RES        equ     256             ; Number of input units per plot unit
      0080                       95  PLOT_RES_2      equ     plot_res/2
      9100                       96  PLOT_MAX        equ     plot_res*145    ; 145 chrs/row
                                 97
                                 98   PUBLIC scale_factor, plot_res, plot_res_2, plot_max
                                 99
                                100
      0024                       101 OSEG at 24H     ; common oseg area
                                102
      0024                       103         tmpreal:        dsl     1
      0028                       104         tmpimag:        dsl     1
      002C                       105         wndptr:         dsw     1
      002E                       106         varptr:         dsw     1
                                107
      0000                       108 RSEG
      0000                       109         fft_mode:       dsb     1
      0001                       110         error:          dsb     1
      0002                       111         avr_cnt:        dsb     1
                                112   PUBLIC error, fft_mode
                                113
                                114         EXTRN   sample_period, control_a2d
                                115
                                116
      0080                       117 DSEG at 80h
      0080                       118         XREAL:                          ; For FFT routine
      0080                       119         DEST_BUFF_BASE: DSW     64      ; For A2D routine
        00C0                     120         XIMAG equ       XREAL+64        ; For FFT routine
                                121
                                122   PUBLIC DEST_BUFF_BASE, XREAL, XIMAG
                                123
                                124
      0200                       125 DSEG AT 200H
                                126
      0200                       127         PLOT_IN:
      0200                       128         FFT_OUT:        DSW     32      ; For FFT routine
      0240                       129         BUFF0_BASE:     DSW     64      ; For A2D routine
      02C0                       130         BUFF1_BASE:     DSW     64      ; For A2D routine
                                131
                                132         PUBLIC  BUFF0_BASE, BUFF1_BASE, FFT_OUT, PLOT_IN
                                133 $eject
```

270189-47

Listing 3—Main Routine (Continued)

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                  02/18/86          PAGE    4

ERR LOC  OBJECT               LINE        SOURCE STATEMENT
                               134
    2080                       135   CSEG AT 2080H
                               136
                               137           EXTRN   INIT_OUTPUT, DRAW_GRAPH, CON_OUT        ; For Plot Routine
                               138           EXTRN   FFT_CALC                               ; For FFT routine
                               139           EXTRN   A2D_BUFF_UTIL                          ; For A2D routine
                               140
    2080 A1000018          R   141           LD      SP,#STACK
    2084 A30100301C            142           LD      AX,3000H
    2089                       143   SBE_WAIT:
    2089 E01CFD                144           djnz    al,sbe_wait     ; WAIT FOR SBE TO CLEAR SERIAL PORT INTERRUPTS
    208C E01DFA                145           djnz    ah,sbe_wait
                               146
    208F EF0000            E   147   BEGIN:  CALL    INIT_OUTPUT     ; Initialize serial port
                               148
    2092                       149   NEW_TRANSFORM_SET:
    2092 B10000            R   150           ldb     fft_mode,#0000B         ;  Bit 0 - Real data / Tabled data#
                               151                                           ;  Bit 1 - Windowed / Unwindowed#
                               152                                           ;  Bit 2 - 10log Mag^2 / Magnitude#
                               153                                           ;  Bit 3 - 256*db plot / Normal Plot#
    2095 B10102            R   154           ldb     avr_cnt,#avr_num
    2098 0120                  155           clr     bx
    209A C3210000200           156   CLRRAM: st      zero,fft_out[bx]        ; clear fft magnitude array
    209F 65020020              157           add     bx,#2
    20A3 89400020              158           cmp     bx,#64
    20A7 DEF1                  159           blt     CLRRAM
                               160
    20A9 300004            R   161   C_load: bbc     fft_mode,0,do_tab       ; Branch if real data is not used
    20AC 2819                  162           CALL    LOAD_DATA
    20AE 2002                  163           br      C_win
                               164
    20B0 282F                  165   do_tab: CALL    TABLE_LOAD
                               166
    20B2 310002            R   167   C_win:  bbc     fft_mode,1,calc         ; Branch if windowing is not used
    20B5 28CB                  168           CALL    DO_WINDOW
                               169
    20B7 EF0000            E   170   CALC:   CALL    FFT_CALC
    20BA 980001            R   171   errtrp: cmpb    error,zero
    20BD D7FB                  172           jne     errtrp
                               173
    20BF E00205            R   174           DJNZ    avr_cnt, LOAD_DATA      ; repeat for AVR_NUM counts
                               175
    20C2 EF0000            E   176           CALL    DRAW_GRAPH
                               177
    20C5 27CB                  178           BR      NEW_TRANSFORM_SET
                               179   $eject
```

270189-48

```
ERR LOC  OBJECT                LINE      SOURCE STATEMENT
                                180  ;-------------------------------------------------------------------------
    20C7                        181  LOAD_DATA:                    ;;;;    LOAD DATA INTO RAM
                                182
    20C7 B1000F                 183          ldb     port1,#00                       ;**** FOR INDICATION ONLY
                                184
    20CA                        185  SET_A2D:
    20CA B11000           E     186          ldb     control_a2d,#con_b0     ; Set converter for buffer0
    20CD 910100           E     187          orb     control_a2d,#01         ; Convert channel 1
    20D0 A1320000         E     188          ld      sample_period,#50       ; 100 us sample period
                                189
    20D4 EF0000           E     190          CALL    a2d_buff_util           ; Start the conversion process
    20D7 3F00FD           E     191          jbs     control_a2d,busy,$      ; wait for all conversions to be done
                                192
    20DA                        193  Down_load:
    20DA B12800           E     194          ldb     control_a2d,#dump_b0_p_s        ; download b0 paired/signed
    20DD EF0000           E     195          CALL    a2d_buff_util
    20E0 F0                     196          RET
                                197
                                198  ;-------------------------------------------------------------------------
    20E1                        199  TABLE_LOAD:
    20E1 0120                   200          clr     bx
    20E3 A102211C               201          ld      ax,#DATA0       ; Load tabled data for testing
    20E7 A21D22                 202  load:   ld      cx,[ax]+
    20EA A21D1E                 203          ld      dx,[ax]+
    20ED C321800022             204          st      cx,xreal[bx]
    20F2 C321C0001E             205          st      dx,ximag[bx]
    20F7 65020020               206          add     bx,#2
    20FB 89400020               207          cmp     bx,#64
    20FF DEE6                   208          blt     LOAD
    2101 F0                     209          RET
                                210
    2102                        211  DATA0:          ; SQUARE WAVE
                                212
    2102 FF7FFF7FFF7FFF7F       213  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2112 FF7FFF7FFF7FFF7F       214  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2122 FF7FFF7FFF7FFF7F       215  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2132 FF7FFF7FFF7FFF7F       216  DCW      32767,  32767,  32767,  32767,  32767,  32767,  32767,  32767
    2142 0180018001800180       217  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2152 0180018001800180       218  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2162 0180018001800180       219  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
    2172 0180018001800180       220  DCW     -32767, -32767, -32767, -32767, -32767, -32767, -32767, -32767
                                221
                                222  $eject
```

270189-49

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                  02/18/86          PAGE    6

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               223   ;-------------------------------------------------------------------------
    2182                       224   DO_WINDOW:                                ;;;; PERFORM HANNING WINDOW
    2182 012C                  225           clr     wndptr
    2184 012E                  226           clr     varptr                    ; Windowing provides an effective
    2186                       227   WINDOW:                                   ; divide by 2 because of the multiply
    2186 A32DBE211C            228           ld      ax,hanning[wndptr]
    218B A32DC02120            229           ld      bx,hanning+2[wndptr]
    2190 FE4F2F80001C24        230           mul     tmpreal,ax,xreal[varptr]
    2197 FE4F2FC0002028        231           mul     tmpimag,bx,ximag[varptr]
    219E 0D0124                232           shll    tmpreal,#1
    21A1 0D0128                233           shll    tmpimag,#1                        ; Compensate for the divide by 2
    21A4 C32F800026            234           st      tmpreal+2,xreal[varptr]
    21A9 C32FC0002A            235           st      tmpimag+2,ximag[varptr]
    21AE 6504002C              236           add     wndptr,#4
    21B2 6502002E              237           add     varptr,#2
    21B6 8940002E              238           cmp     varptr,#64
    21BA D7CA                  239           jne     window
    21BC F0                    240           RET
                               241
    21BE                       242   HANNING:        ; Windowing function
                               243
    21BE 00004F003B01C102      244   DCW         0,   79,  315,  705, 1247, 1935, 2761, 3719
    21CE BF126617711CD421      245   DCW      4799, 5990, 7281, 8660,10114,11628,13187,14778
    21DE 004045467C4C9352      246   DCW     16384,17989,19580,21139,22653,24107,25486,26777
    21EE 406D787136757078      247   DCW     27968,29048,30006,30832,31520,32062,32452,32688
    21FE FF7FB07FC47E3E7D      248   DCW     32767,32688,32452,32062,31520,30832,30006,29048
    220E 406D99688E632B5E      249   DCW     27968,26777,25486,24107,22653,21139,19580,17989
    221E 0040BA3983336C2D      250   DCW     16384,14778,13187,11628,10114, 8660, 7281, 5990
    222E BF12870EC90A8F07      251   DCW      4799, 3719, 2761, 1935, 1247,  705,  315,   79
    223E 0000                  252   DCW         0
                               253
                               254   $eject
```

270189-50

```
ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                255
    3D00                        256   CSEG AT 3D00H           ; ADDITIONAL  TABLES  FOR  TESTING
                                257                 ; SINE 7.0 X
    3D00                        258   DATA1:
    3D00 00003351897DE270       259   DCW          0, 20787, 32137, 28898, 12539, -9512,-27245,-32609
    3D10 7EA574F31C477C7A       260   DCW     -23170, -3212, 18204, 31356, 30273, 15446, -6393,-25329
    3D20 01800F9D08E7563C       261   DCW     -32767,-25329, -6392, 15446, 30273, 31356, 18204, -3212
    3D30 7EA59F809395D8DA       262   DCW     -23170,-32609,-27245, -9512, 12539, 28898, 32137, 20787
    3D40 0000CDAE77821E8F       263   DCW         -0,-20787,-32137,-28898,-12539,  9512, 27245, 32609
    3D50 825A8C0CE4B88485       264   DCW      23170,  3212,-18204,-31356,-30273,-15446,  6393, 25329
    3D60 FF7FF162F818AAC3       265   DCW      32767, 25329,  6392,-15446,-30273,-31356,-18204,  3212
    3D70 825A617F6D6A2825       266   DCW      23170, 32609, 27245,  9512,-12539,-28898,-32137,-20787
                                267
    3D80                        268   DATA2:          ; SINE 7.5 X
                                269
    3D80 0000F555617FCF66       270   DCW          0, 22005, 32609, 26319,  6393,-16846,-31356,-29621
    3D90 05CF1F2BE270297C       271   DCW     -12539, 11039, 28898, 31785, 18204, -4808,-25329,-32728
    3DA0 7EA5B8F933519C7E       272   DCW     -23170, -1608, 20787, 32412, 27245,  7962,-15446,-30852
    3DB0 BF8946C92825C96D       273   DCW     -30273,-14010,  9512, 28105, 32137, 19519, -3212,-24279
    3DC0 018029A174F33F4C       274   DCW     -32767,-24279, -3212, 19519, 32137, 28105,  9512,-14010
    3DD0 BF897C87AAC31A1F       275   DCW     -30273,-30852,-15446,  7962, 27245, 32412, 20787, -1608
    3DE0 7EA528800F9D38ED       276   DCW     -23170,-32728,-25329, -4808, 18205, 31785, 28898, 11039
    3DF0 05CF4B8C848533BE       277   DCW     -12539,-29621,-31356,-16845,  6393, 26319, 32609, 22005
                                278
    3E00                        279   DATA3:          ; .707*SINE 7.5X
                                280
    3E00 0000C63C0F5AAF48       281   DCW          0, 15558, 23055, 18607,  4520,-11910,-22169,-20942
    3E10 5FDD7C1ECF4FC857       282   DCW      -8865,  7804, 20431, 22472, 12870, -3399,-17908,-23138
    3E20 03C08FFB69398459       283   DCW     -16381, -1137, 14697, 22916, 19262,  5629,-10921,-21812
    3E30 65AC4FD9451A9E4D       284   DCW     -21403, -9905,  6725, 19870, 22721, 13800, -2271,-17165
    3E40 82A5F3BC21F7E835       285   DCW     -23166,-17165, -2271, 13800, 22721, 19870,  6725, -9905
    3E50 65ACCCAA58D5FD15       286   DCW     -21403,-21812,-10920,  5629, 19262, 22916, 14696, -1137
    3E60 03C09EA50CBAB9F2       287   DCW     -16381,-23138,-17908, -3399, 12871, 22472, 20431,  7804
    3E70 5FDD32AE67A97AD1       288   DCW      -8865,-20942,-22169,-11910,  4520, 18607, 23055, 15557
                                289
    3E80                        290   DATA4:          ; .707*SINE(11x) /16
                                291
    3E80 0000FD04B40472FF       292   DCW          0,  1277,  1204,  -142, -1338, -1119,   282,  1386
    3E90 00045CFE74FA69FC       293   DCW       1024,  -420, -1420,  -919,   554,  1441,   804,  -683
    3EA0 58FA55FD2403A105       294   DCW      -1448,  -683,   804,  1441,   554,  -919, -1420,  -420
    3EB0 00046A051A01A1FB       295   DCW       1024,  1386,   282, -1119, -1338,  -142,  1204,  1277
    3EC0 000003FB4CFB8E00       296   DCW         -0, -1277, -1204,   142,  1338,  1119,  -282, -1386
    3ED0 00FCA4018C059703       297   DCW      -1024,   420,  1420,   919,  -554, -1441,  -804,   683
    3EE0 A805AB02DCFC5FFA       298   DCW       1448,   683,  -804, -1441,  -554,   919,  1420,   420
    3EF0 00FC96FAE6FE5F04       299   DCW      -1024, -1386,  -282,  1119,  1338,   142, -1204, -1277
                                300
    3F00                        301   DATA5:          ; .707*(SINE 7.5X + 1/16 SINE 11X)
```

270189-51

Listing 3—Main Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    FFT_MAIN_APNOTE                                  02/18/86          PAGE    8

ERR LOC  OBJECT               LINE        SOURCE STATEMENT
                               302
    3F00 0000C241C35E2148      303   DCW         0, 16834, 24259, 18465,  3182,-13029,-21886,-19557
    3F10 5EE1D81C434A3154      304   DCW      -7842,  7384, 19011, 21553, 13425, -1958,-17103,-23821
    3F20 5BBAE5F88D3C245F      305   DCW     -17829, -1819, 15501, 24356, 19816,  4710,-12341,-22232
    3F30 65B0B9DE5F1B3F49      306   DCW     -20379, -8519,  7007, 18751, 21383, 13658, -1067,-15888
    3F40 82A5F6B76DF27636      307   DCW     -23166,-18442, -3475, 13942, 24059, 20990,  6442,-11290
    3F50 65A870ACE4DA9419      308   DCW     -22427,-21392, -9500,  6548, 18708, 21475, 13892,  -454
    3F60 ABC548A8E8B618ED      309   DCW     -14933,-22456,-18712, -4840, 12317, 23391, 21851,  8225
    3F70 5FD9C8A84DA8D9D5      310   DCW      -9889,-22328,-22451,-10791,  5857, 18749, 21851, 14281
                               311
                               312
    3F80                       313   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-52

Listing 3—Main Routine (Continued)

```
SERIES-III MCS-96 RELOCATOR AND LINKER, V2.0
Copyright 1983 Intel Corporation

INPUT FILES: :F2:FTMAIN.OBJ, :F2:FFTRUN.OBJ, :F2:PLOTSP.OBJ, :F2:A2DCON.OBJ
OUTPUT FILE: :F2:FFTOUT
CONTROLS SPECIFIED IN INVOCATION COMMAND:
   IX


INPUT MODULES INCLUDED:
   :F2:FTMAIN.OBJ(FFT_MAIN_APNOTE)  02/18/86
   :F2:FFTRUN.OBJ(FFT_RUN)  02/18/86
   :F2:PLOTSP.OBJ(PLOT_SERIAL)  02/18/86
   :F2:A2DCON.OBJ(A2D_BUFFERING_UTILITY)  02/18/86


SEGMENT MAP FOR :F2:FFTOUT(FFT_MAIN_APNOTE):

            TYPE    BASE      LENGTH    ALIGNMENT    MODULE NAME
            ----    ----      ------    ---------    -----------

**RESERVED*         0000H     001AH
            REG     001AH     0001H     BYTE         PLOT_SERIAL
*** GAP ***         001BH     0001H
            REG     001CH     0008H     ABSOLUTE     FFT_MAIN_APNOTE
            OVRLY   0024H     0035H     ABSOLUTE     FFT_RUN
**OVERLAP** OVRLY   0024H     0010H     ABSOLUTE     PLOT_SERIAL
**OVERLAP** OVRLY   0024H     000CH     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         0059H     0001H
            OVRLY   005AH     0006H     WORD         A2D_BUFFERING_UTILITY
            REG     0060H     000CH     WORD         A2D_BUFFERING_UTILITY
            REG     006CH     0003H     BYTE         FFT_MAIN_APNOTE
*** GAP ***         006FH     0011H
            DATA    0080H     0080H     ABSOLUTE     FFT_MAIN_APNOTE
            STACK   0100H     001EH     WORD
            DATA    011EH     0080H     WORD         FFT_RUN
*** GAP ***         019EH     0062H
            DATA    0200H     0140H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         0340H     1CC2H
            CODE    2002H     0002H     ABSOLUTE     A2D_BUFFERING_UTILITY
*** GAP ***         2004H     007CH
            CODE    2080H     01C0H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         2240H     0040H
            CODE    2280H     0215H     ABSOLUTE     FFT_RUN
*** GAP ***         2495H     006BH
            CODE    2500H     0168H     ABSOLUTE     PLOT_SERIAL
            CODE    2668H     00ECH     BYTE         A2D_BUFFERING_UTILITY
*** GAP ***         2754H     10ACH
            CODE    3800H     042AH     ABSOLUTE     FFT_RUN
*** GAP ***         3C2AH     00D6H
            CODE    3D00H     0280H     ABSOLUTE     FFT_MAIN_APNOTE
*** GAP ***         3F80H     C080H
```

270189-53

Listing 3—Main Routine (Continued)

Several constants are then setup for other routines. The purpose of centrally locating these constants was the ease of modifying the operation of the routines. Note that AVR__NUM and SCALE__FACTOR must be changed at the same time. SCALE__FACTOR is the shift count used to divide each FFT output value before it is added to the output array. AVR__NUM must be less than 2**SCALE__FACTOR or an overflow could occur. Next, the public variables are declared for the arrays and a few other parameters.

The program then begins by setting the stack pointer and waiting for the SBE-96 to finish talking to the terminal. If this is not done, there may be serial port interrupts occurring for the first twenty five milliseconds of program operation.

Initialization of the plotter is next, followed by setting the FFT__MODE byte. This byte controls the graphing, loading and magnitude calculation of the FFT data. Since FFT__MODE is declared PUBLIC in this module, and EXTERNAL in the PLOT module and FFTRUN module, the extra bits available in this byte can be used for future enhancements.

The next step is to clear the FFT output array. Since the FFT program can be set to average its results by dividing the output before adding it to the magnitude array, the array must be cleared before beginning the program.

Data is then loaded into into the FFT input array by the code at LOAD__DATA, or the code at TABLE__LOAD, depending on the value of FFT__MODE bit 0. The tabled data located at DATA0 is a square wave of magnitude 1. This waveform provides a reasonable test of the FFT algorithm, as many harmonics are generated. The results are also easy to check as the pattern contains half zeros, imaginary values which are always the same, and real values which decrease. Figure 13 shows the output in fractions, hexadecimal and decimal. The hexadecimal and decimal values are based on an output of 16384 being equal to 1.00.

Note that the magnitude is

$$SQR\ (REAL^2 + IMAG^2)$$

and the dB value is

$$10\ LOG\ (\ (REAL^2 + IMAG^2)/65536\ )$$

The divide by 65536 is used for the dB scale to provide a reasonable range for calculations. If this was not done, a 32-bit LOG function would have been needed.

After the data is loaded, the data is optionally windowed, based on FFT__MODE bit 1, and the FFT program is called. Once the loop has been performed AVR__CNT times, the graph is drawn by the plot routine.

Appended to the main routine is the FFTOUT.M96 Listing. This is provided by the relocator and linker, RL96. With this listing and the main program, it is possible to determine which sections of code are at which addresses.

Using the modular programming methods employed here, it is reasonably easy to debug code. By emulating the program in a relatively high level language, each routine can be checked for functionality against a known standard. The closer the high level implementation matches the ASM96 version, the more possible checkpoints there are between the two routines.

Once all of the program routines (modules) can be shown to work individually, the main program should work unless there is unwanted interaction between the modules. These interactions can be checked by verifying the inputs and outputs of each module. The assembly language locations to perform the program breaks can be retrieved by absolutely locating the main module. The other modules can be dynamically located by RL96.

The more interactive program modules are, the more difficult the program becomes to debug. This is especially true when multiple interrupts are occurring, and several of the interrupt routines are themselves interruptable. In these cases, it may be necessary to use debugging equipment with trace capability, like the VLSiCE-96. If this type of equipment is not available, then using I/O ports to indicate the entering and leaving of each routine may be useful. In this way it will be possible to watch the action of the program on an oscilloscope or logic analyzer. There are several places within this code that I/O port toggling has been used as an aid to debugging the program. These lines of code are marked "FOR INDICATION ONLY."

K	Fractional			dB	Decimal			Hexadecimal		
	REAL	IMAG	MAG²		REAL	IMAG	MAG²	REAL	IMAG	MAG²
0	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
1	0.0625	−1.2722	1.2738	38.225	1024	−20843	20868	400	AE95	5184
2	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
3	0.0625	−0.4213	0.4260	28.710	1024	−6903	6978	400	E509	1B42
4	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
5	0.0625	−0.2495	0.2572	24.329	1024	−4088	4214	400	F008	1076
6	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
7	0.0625	−0.1747	0.1855	21.491	1024	−2862	3039	400	F4D2	BDF
8	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
9	0.0625	−0.1321	0.1462	19.421	1024	−2165	2395	400	F78B	95B
10	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
11	0.0625	−0.1043	0.1216	17.820	1024	−1708	1992	400	F954	7C8
12	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
13	0.0625	−0.0843	0.1049	16.540	1024	−1381	1719	400	FA9B	6B7
14	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
15	0.0625	−0.0690	0.0931	15.499	1024	−1130	1525	400	FB96	5F5
16	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
17	0.0625	−0.0566	0.0844	14.645	1024	−928	1382	400	FC60	566
18	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
19	0.0625	−0.0464	0.0778	13.944	1024	−759	1275	400	FD09	4FB
20	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
21	0.0625	−0.0375	0.0729	13.374	1024	−614	1194	400	FD9A	4AA
22	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
23	0.0625	−0.0296	0.0691	12.918	1024	−484	1133	400	FE1C	46D
24	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
25	0.0625	−0.0224	0.0664	12.564	1024	−366	1088	400	FE92	440
26	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
27	0.0625	−0.0157	0.0644	12.305	1024	−256	1056	400	FF00	420
28	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
29	0.0625	−0.0093	0.0632	12.135	1024	−152	1035	400	FF68	40B
30	0.0000	0.0000	0.0000	0.000	0	0	0	0	0	0
31	0.0625	−0.0031	0.0626	12.051	1024	−50	1025	400	FFCE	401

Figure 13. FFT Output for a Square Wave Input

9.0 ANALOG TO DIGITAL CONVERTER MODULE

The module presented in Listing 4 is a general purpose one which converts analog values under interrupt control and stores them in one of two buffers. These buffers can then be downloaded to another buffer, such as the input buffer to the FFT program. During downloading, this module can convert the data into signed or unsigned formats, and fill a linear or a paired array. A paired array is like the one used in the FFT transform program. It requires N data points placed alternately in two arrays, one starting at zero and the other at N/2.

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86          PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:A2DCON.A96
OBJECT FILE: :F2:A2DCON.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 1   $pagelength(50)
                                 2
                                 3   A2D_Buffering_Utility   module  stacksize(12)
                                 4   ;
                                 5   ; Intel Corporation,  July 16, 1985
                                 6   ; by Dave Ryan, Intel Applications Engineer
                                 7   ;
                                 8   ; This utility fills a memory buffer with A/D conversion results.  The
                                 9   ; conversions are done under interrupt control, and are initiated when
                                10   ; A2D_BUFF_Util is called.  The results of the conversions are placed
                                11   ; in one of two buffers, called BUFF0 and BUFF1.
                                12   ;
                                13   ; This utility provides options for the selection of the buffer lengths, data
                                14   ; format, sample period, conversion channel and time base.  The utility also
                                15   ; has a donwload routine that will load either buffer into a register file
                                16   ; buffer.  Output formats can also be chosen for the downloaded buffer.  The
                                17   ; data can be formatted as signed or unsigned linear or paried arrays.
                                18   ;
                                19   ; RUN-TIME OPTIONS
                                20   ;
                                21   ; Rather than use the STACK to pass controls, this utility gets its directions
                                22   ; from 2 control words in memory.  The utility expects that its control words
                                23   ; are valid at the time A2D_BUFF_Util is called and remain valid throughout
                                24   ; A/D interrupt executions and downloads.  The control words are:
                                25   ;
                                26   ;       Sample_Period   ; WORD ; The time between samples in timer counts
                                27   ;                               ; where the timer used has been specified
                                28   ;
                                29   ;
                                30   ;       Control_A2D     ; BYTE ; Control information for the utility:
                                31   ;                                  BIT#
                                32   ;
                                33   ;                                 ;  0-2  ; Channel Number
                                34   ;;                                ;   3   ; Signed Result/Unsigned Result#
                                35   ;                                 ;   4   ; Convert/Download#
                                36   ;                                 ;   5   ; BUFF1/BUFF0# for conversions
                                37   ;                                           ; BUFF0/BUFF1# for downloads
                                38   ;                                 ;   6   ; Linear/Paired#
                                39   ;                                 ;   7   ; Converter BUSY/IDLE#
                                40
                                41   $EJECT
```

270189-54

Listing 4—A to D Converter Routine

```
ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 42   ;
                                 43   ; The following is a table of equates that can be used to simplify the
                                 44   ; bit diddling requirements.  If you are not running conversions concurrently
                                 45   ; with downloads, always LDB Control_A2D with the following command then
                                 46   ; ORB Control_A2D with the channel number you wish to convert if you are
                                 47   ; starting a conversion.
                                 48   ;
                                 49   ; Once the utility is called, care must be taken when Control_A2d is
                                 50   ; modified.  You can cause downloads to occur while conversions are running,
                                 51   ; but you cannot start conversions during a download.  To do this, ORB to the
                                 52   ; control byte with the appropriate bits set.  Do NOT change the BUFF bit or
                                 53   ; the BUSY bit.  Just set the download bit and set the data format bits to the
                                 54   ; correct values.
                                 55   ;
                                 56   ; The BUFF bit has opposite definitions for conversions and downloads.  This
                                 57   ; allows conversions to be done into BUFF0 while downloads come from BUFF1, and
                                 58   ; vice versa.
                                 59   ;
                                 60   ; A2D UTILITY COMMANDS_____________________________________________________
                                 61   ;
                                 62   ;con_b0          equ     00010000b;convert to BUFF0
                                 63   ;con_b1          equ     00110000b;    "       BUFF1
                                 64   ;
                                 65   ;dump_b0_l_u     equ     01100000b; download  BUFF0  as LINEAR USIGNED data
                                 66   ;dump_b1_l_u     equ     01000000b;    "       BUFF1  "    "      "      "
                                 67   ;dump_b0_p_u     equ     00100000b;    "       BUFF0  "  PAIRED   "      "
                                 68   ;dump_b1_p_u     equ     00000000b;    "       BUFF1  "    "      "      "
                                 69   ;dump_b0_l_s     equ     01101000b; download  BUFF0  as LINEAR SIGNED  data
                                 70   ;dump_b1_l_s     equ     01001000b;    "       BUFF1  "    "      "      "
                                 71   ;dump_b0_p_s     equ     00101000b;    "       BUFF0  "  PAIRED   "      "
                                 72   ;dump_b1_p_s     equ     00001000b;    "       BUFF1  "    "      "      "
                                 73   ;___________________________________________________________________________
                                 74   $eject
```

270189-55

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86        PAGE    3

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                 75   ;
                                 76   ; ASSEMBLY-TIME OPTIONS
                                 77   ;
                                 78   ; The base addresses and length of each conversion buffer and the destination
                                 79   ; buffer are DECLARED EXTRNal in this utility. Other options such as selection
                                 80   ; of the timer used as a timebase, the length of the buffer, and the effective
                                 81   ; number of bits in the reported result are set at assembly time through use
                                 82   ; of EQUates in this module.
                                 83   ;
                                 84   ; The following parameters need to be provided at assembly or link time.
                                 85   ; The buffer bases are declared EXTRNal by this utility, while the buffer
                                 86   ; length shift count and HSO commands are EQUated.
                                 87   ;
                                 88   ;       BUFF0_BASE      ; The starting address of BUFF0
                                 89   ;       BUFF1_BASE      ; The starting address of BUFF1
                                 90   ;       DEST_BUFF_BASE  ; The starting address of the download
                                 91   ;                               ; target buffer.
                                 92   ;
                                 93   ;       BUFF_LENGTH     ; The number of SAMPLES that each
                                 94   ;                               ; buffer must hold. must be >1 and <256
                                 95   ;
                                 96   ;       Shift_count     ; The number of times that the conversion result is
                                 97   ;                       ; to be shifted right from its natural left justified
                                 98   ;                       ; position. Setting a shift count greater than 6 will
                                 99   ;                       ; result in lost bits to the right.  Rounding is NOT
                                100   ;                       ; done.
                                101   ;
                                102   ;       CLOCK           ; Specify as either TIMER1 or T2CLK.  This is the
                                103   ;                       ; timebase used for conversions.
                                104   ;
                                105   ;       Samples are stored as words in the buffers.  The program stores
                                106   ; conversions linearly in BUFF0 and BUFF1, and linearly or paired in the
                                107   ; destination buffer as selected.  If the download is to be paired, the first
                                108   ; sample is placed in location DEST_BUFF_BASE, the second sample is placed in
                                109   ; location (DEST_BUFF_BASE + BUFF_LENGTH), the third in (DEST_BUFF_BASE + 2),
                                110   ; the fourth in (DEST_BUFF_BASE + 2 + BUFF_LENGTH), etc.
                                111   ;
                                112   $eject
```

270189-56

Listing 4—A to D Converter Routine (Continued)

```
ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                113   ;
                                114   ;NOTES ON EXECUTION
                                115   ;
                                116   ; When a utility call directs the initiation of a set of A2D conversions, the
                                117   ; first conversion is begun at approximately one sample time plus 50 state
                                118   ; times from when the utility was called.  This assumes that no interrupts are
                                119   ; present.
                                120   ;
                                121   ; The conversion busy bit is set approximately 50 state times after a call
                                122   ; to the utility, if the convert bit was set in the A2D_Control byte.  The
                                123   ; busy bit is cleared after all conversion results have been stored in the
                                124   ; result buffer designated (BUFF0 or BUFF1).
                                125   ;
                                126   ; Take great care in modifying the A2D_Control byte to do a download while
                                127   ; conversions are taking place.  You can never download a buffer that is
                                128   ; being converted into.  The results would be invalid.
                                129   $eject
```

270189-57

Listing 4—A to D Converter Routine (Continued)

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86          PAGE    5

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                130
    0000                        131           RSEG
                                132
                                133   EXTRN BUFF0_BASE, BUFF1_BASE, DEST_BUFF_BASE
                                134   EXTRN ad_command, ad_result_lo, ad_result_hi
                                135   EXTRN hso_command, hso_time,sp
                                136
      0040                      137   BUFF_LENGTH     EQU     64
      0001                      138   Shift_Count     EQU     1
      000A                      139   CLOCK           EQU     TIMER1
                                140
                                141   ; set up hso commands for correct timer *********************************
                                142
      000A                      143   TIMER1          equ     0AH
      000C                      144   T2CLK           equ     0CH
                                145
      0000                      146   MASK            equ     (10h*CLOCK)AND(40h)
                                147
      000F                      148   Start_A2D       equ     (00001111b)OR(MASK)
                                149                           ;start a2d based on timer 1, no interrupt
                                150
      0000                      151   HSO_0_Low       equ     (00000000b)OR(MASK)
                                152                           ; make hso.0 low based on timer1 no interrupt
                                153
      0020                      154   HSO_0_High      equ     (00100000b)OR(MASK)
                                155                           ; make hso.0 hi  based on timer1 no interrupt
                                156
                                157
                                158   ; set up storage ********************************************************
                                159
    0000                        160   adudtemp0:      DSW     1; temp register for download calls
                                161
    0002                        162   aductemp0:      DSW     1; temp registers for conversion calls
    0004                        163   aductemp1:      DSW     1
    0006                        164   top_of_buffer:  DSW     1
    0008                        165   sample_count:   DSB     1
                                166
    0009                        167   Control_A2D:    DSB     1; the byte that controls the utility execution
      0003                      168                   DForm   equ     3       ; Signed/Unsigned#
      0004                      169                   Con_Dwn equ     4       ; Convert/Download#
      0005                      170                   BO_B1   equ     5       ; Buff1/Buff0# for conversions
                                171                                           ; Buff0/Buff1# for downloads
      0006                      172                   Lin_Par equ     6       ; Linear/Paired#
      0080                      173                   Busy    equ     10000000B       ; Bit 8
                                174   $eject
```

270189-58

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86          PAGE    6

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                175
    000A                        176   Sample_Period:  DSW     1; the word that specifies the number of clock ticks
                                177                            ; that elapse between each sample
                                178
                                179   PUBLIC  Control_A2D, Sample_Period
                                180
                                181
    0000                        182           OSEG
                                183
    0000                        184   src_ptr:        DSW     1; some overlayable temp registers
      0000                      185           temp set src_ptr:WORD
    0002                        186   dest_ptr:       DSW     1
    0004                        187   loop_count:     DSW     1
                                188
                                189
    2002                        190           CSEG    at      2002h
                                191
                                192   PUBLIC  A2D_DONE_Vector
                                193
    2002 AC00            R      194   DCW     A2D_DONE_Vector
                                195
                                196
    0000                        197           CSEG
                                198
                                199   PUBLIC  A2D_BUFF_Util
                                200
                                201   Load_HSO_Command MACRO  var     ; Macro to load HSO
                                202
                                203           LDB     hso_command,#var
                                204           LD      hso_time,aductemp0
                                205   ENDM
                                206   $eject
```

270189-59

Listing 4—A to D Converter Routine (Continued)

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86          PAGE    7

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               207
    0000                       208   A2D_BUFF_Util:
                               209
    0000 3C0962          R     210           JBS     Control_A2D, Con_Dwn, Convert   ; Select convert or download
    0003                       211   Download:
    0003 A1000000        E     212           LD      src_ptr,#BUFF1_BASE
    0007 350904          R     213           JBC     Control_A2D, B0_B1,   Set_Data_Format
                               214
    000A                       215   Download_BUFF0:
    000A A1000000        E     216           LD      src_ptr,#BUFF0_BASE
                               217
                               218
    000E                       219   Set_Data_Format:                                ; Choose linear or paired
    000E A1000002        E     220           LD      dest_ptr, #DEST_BUFF_BASE
    0012 B14004          R     221           LDB     loop_count,#BUFF_LENGTH
    0015 3E091D          R     222           JBS     Control_A2D, Lin_Par,    Linear_data_loop
                               223
                               224
    0018 180104          R     225   PAIRED: SHRB    loop_count,#1           ; The paired data routine uses 1/2
                               226                                           ; as many loops as the unpaired
    001B                       227   Paired_Data_loop:
    001B A20000          R     228           LD      adudtemp0,[src_ptr]+            ; Move even word
    001E C20200          R     229           ST      adudtemp0,[dest_ptr]
    0021 65400002        R     230           ADD     dest_ptr,#BUFF_LENGTH   ; Length = # of words = 1/2 # of bytes
                               231
    0025 A20000          R     232           LD      adudtemp0,[src_ptr]+            ; Move odd word
    0028 C20200          R     233           ST      adudtemp0,[dest_ptr]+
    002B 69400002        R     234           SUB     dest_ptr,#BUFF_LENGTH
                               235
    002F E004E9          R     236           DJNZ    loop_count, Paired_Data_loop    ; Loop until done
                               237
    0032 280D                  238           CALL    Convert_Data
    0034 F0                    239           RET
                               240
                               241
    0035                       242   Linear_Data_loop:                               ; Move data linearly
    0035 A20000          R     243           LD      adudtemp0,[src_ptr]+
    0038 C20200          R     244           ST      adudtemp0,[dest_ptr]+
                               245
    003B E004F7          R     246           DJNZ    loop_count, Linear_Data_loop    ; Loop until done
                               247
    003E 2801                  248           CALL    Convert_Data
    0040 F0                    249           RET
                               250   $eject
```

270189-60

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                    02/18/86          PAGE    8

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                251
    0041                        252   Convert_Data:                  ; Convert the data in the destination buffer
                                253
    0041 A1400004          R    254           LD      loop_count,#BUFF_LENGTH
    0045 A1000000          E    255           LD      src_ptr,#DEST_BUFF_BASE
                                256
    0049 A20000            R    257   Again:  LD      adudtemp0,[src_ptr]
    004C 71C000            R    258           ANDB    adudtemp0,#11000000b
    004F 330909            R    259           JBC     Control_A2D, DForm, Unsigned_Result
                                260
    0052                        261   Signed_Result:
    0052 69E07F00          R    262           SUB     adudtemp0,#7fe0H
    0056 0A0100            R    263           SHRA    adudtemp0,#Shift_Count
    0059 2003                   264           BR      Replace_Sample
                                265
    005B                        266   Unsigned_Result:
    005B 080100            R    267           SHR     adudtemp0, #Shift_Count
                                268
    005E                        269   Replace_Sample:
    005E C20000            R    270           ST      adudtemp0,[src_ptr]+
    0061 E004E5            R    271           DJNZ    loop_count,Again        ; Loop until done
                                272
    0064 F0                     273           RET
                                274
                                275
    0065                        276   Convert:        ;; Prepare to Start Conversions
                                277
    0065 F2                     278           PUSHF
                                279
    0066 918009            R    280           ORB     Control_A2D, #Busy              ; set converter busy bit
                                281
    0069 B13F08            R    282           LDB     sample_count,#BUFF_LENGTH - 1
    006C A1000006          E    283           LD      top_of_buffer,#BUFF0_BASE
    0070 A1800004          E    284           LD      aductempl,#(BUFF0_BASE + 2*BUFF_LENGTH)
                                285
    0074 350908            R    286           JBC     Control_A2D, B0_B1, Start_Conversions
    0077 A1000006          E    287           LD      top_of_buffer,#BUFF1_BASE
    007B A1800004          E    288           LD      aductempl,#(BUFF1_BASE + 2*BUFF_LENGTH)
                                289   $eject
```

270189-61

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                          02/18/86          PAGE    9

ERR LOC  OBJECT               LINE        SOURCE STATEMENT
                               290
    007F                       291   Start_Conversions:
                               292
    007F 51070900         E    293           ANDB    ad_command,Control_A2D,#00000111b       ;load channel number
                               294
    0083 440A0A02         R    295           ADD     aductemp0,CLOCK,Sample_Period           ;start first conversion
                               296                                                           ;one sample time from
                               297                                                           ;now
                               298
                               299           Load_HSO_Command Start_A2D              ; Start A2D at Time=aductemp0
                               303
    008D CC00             R    304           POP     temp                            ; get a copy of the psw
                               305
                               306           Load_HSO_Command HSO_0_high             ; set hso.0 high at conversion
                               310                                                   ; start time for external S/H
                               311
    0095 81020200         R    312           OR      temp,#202h                      ; enable a2d interrupts
                               313
    0099 640A02           R    314           ADD     aductemp0,Sample_Period
                               315
                               316           Load_HSO_Command Start_A2D              ; start second convertion one
                               320                                                   ; sample time from the first
                               321
    00A2 C800             R    322           PUSH    temp                            ; put psw back on stack
                               323
                               324           Load_HSO_Command HSO_0_low              ;lower hso.0 for external S/H
                               328
    00AA F3                    329           POPF
    00AB F0                    330           RET
                               331   $eject
```

270189-62

Listing 4—A to D Converter Routine (Continued)

```
MCS-96 MACRO ASSEMBLER    A2D_BUFFERING_UTILITY                                  02/18/86          PAGE   10

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
    00AC                        332            CSEG
                                333
    00AC                        334    A2D_DONE_Vector:                ; A/D INTERRUPT ROUTINE
    00AC F2                     335            PUSHF
                                336
    00AD C60600           E     337            STB     ad_result_lo,[top_of_buffer]+
    00B0 C60600           E     338            STB     ad_result_hi,[top_of_buffer]+
    00B3 51070900         E     339            ANDB    ad_command,Control_A2D,#00000111b       ;load channel number
                                340
    00B7 E00809           R     341            DJNZ    sample_count, Sample_Again
    00BA 1708             R     342            INCB    sample_count
                                343
    00BC 880406           R     344            CMP     top_of_buffer,aductempl         ; Check top of buffer
    00BF DF26                   345            BE      Top_of_buffers
    00C1 F3                     346            POPF
    00C2 F0                     347            RET
                                348
    00C3                        349    Sample_Again:
    00C3 640A02           R     350            ADD     aductemp0,Sample_Period         ; Set next sample time
    00C6 880406           R     351            CMP     top_of_buffer,aductempl         ; Check top of buffer
                                352                                                    ; for later jump
                                353            Load_HSO_Command Start_A2D
                                357
    00CF 30080B           R     358            JBC     sample_count,0,Make_HSO_High
                                359
    00D2                        360    Make_HSO_low:
    00D2 FD                     361            nop                             ; wait 8 states after HSO load
                                362            Load_HSO_Command HSO_0_Low
                                366                                            ; Load for change of HSO to trigger S/H
    00D9 DF0C                   367            BE      Top_of_buffers
    00DB F3                     368            POPF
    00DC F0                     369            RET
                                370
    00DD                        371    Make_HSO_high:
                                372            Load_HSO_Command HSO_0_High     ; Load for change of HSO to trigger S/H
                                376
    00E3 DF02                   377            BE      Top_of_buffers
    00E5 F3                     378            POPF
    00E6 F0                     379            RET
                                380
    00E7                        381    Top_of_buffers:
    00E7 717F09           R     382            ANDB    Control_A2D,#NOT(Busy)  ; Clear converter BUSY bit
    00EA F3                     383            POPF
    00EB F0                     384            RET
    00EC                        385    END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-63

Listing 4—A to D Converter Routine (Continued)

The listing contains a fairly complete description of what the program does. The block by block operations are shown below:

Lines 1-198 describe the program, declare the variables and set up equates. Several of these variables are declared as overlayable, so the user needs to be careful if using this module for other than the FFT program.

Lines 205-210 declare a macro which is used to load the HSO unit. This will be used repeatedly through the code.

Lines 212-253 determine whether a conversion or download has been requested. If a download has been requested, the data is downloaded to the destination array as either paired or linear data. Paired data has been described earlier.

Lines 255-278 contain a subroutine which converts the destination array to either signed or unsigned numbers. The numbers are also shifted right to provide the desired full-scale value as requested by SHIFT_COUNT.

Lines 279-334 initialize the conversion routine. HSO.0 is toggled with the start of each routine so that an external sample and hold can be used. The instructions in lines 308, 316, and 326 have been interweaved with the Load_HSO_Commands to provide the required 8 state delays between HSO loadings. If this was not done, NOPs would have been needed. It is easier to understand the code if these lines are thought of as being gathered at line 326.

Lines 337-353 are the actual A/D interrupt routine. The A/D results are placed BYTE by BYTE on the buffer, the A/D is reloaded, and then the number of samples taken is compared to the number needed. Note that the A/D command register needs to be reloaded even if the channel does not change. INCB on line 348 is used to insure that the DJNZ falls through on the next pass (if sample_count is not reset).

Lines 355-396 complete the routine. The HSO is set up to trigger the next conversion and provide the HSO.0 toggle for an external sample and hold. Once again, the time between consecutive loads of the HSO is 8 states minimum. Note that this section of code has been optimized for speed by reducing branches to an absolute minimum and duplicating code where needed.

This concludes the description of the A to D buffer module. In the FFT program, this module is run, then the FFT transform module, then the plot module. This allows variables to be overlaid, saving RAM space. The time cost for this is not bad, considering the printer is the limiting factor in these conversions. If more RAM was provided, and the FFT was run with its data in external RAM, this module could be run simultaneously with the other modules.

10.0 DATA PLOTTING MODULE

The plot module is relatively straight-forward, and is shown in Listing 5. After the declarations, which include overlayable registers, an initialization routine is listed. This separately called routine sets up the serial port on the 8096 to talk to the printer. In this case, the port has to be set for 300 baud.

A console out routine follows. This routine can also be called by any program, but it is used only by the plot routine in this example. The write to port 1 is used to trace the program flow. The character to be output is passed to this routine on the stack. This conforms to PLM-96 requirements.

Since all stack operations on the 8096 are 16-bits wide, a multiple character feature has been added to the console out routine. If the high byte it receives is non-zero, the ASCII character in that byte is printed after the character in the low byte. If the high byte has a value between 128 and 255, the character in the low byte is repeated the number of times indicated by the least significant 7 bits of the high byte.

The print decimal number routine is next. It is called with two words on the stack. The first word is the unsigned value to be printed. The second byte contains information on the number of places to be printed and zero and blank suppression. This routine is not overflow-proof. The user must declare a sufficient number of places to be printed for all possible numbers.

The DRAW_GRAPH routine provides the plot. It first sends a series of carriage return, line feeds (CRLFs) to clear the printer and provides a margin on the paper. Each row is started with the row number, 2 spaces, and a "+". Asterisks are then plotted until

Number of asterisks > FFT Value / PLOT_RES

Recall that PLOT_RES is a variable set by the main program. When the number of asterisks hits the desired value, the value of the line is printed. If the Decibel mode is selected, the line value is divided by 512 and printed in integer + decimal part form, followed by "dB". If the number of asterisks reaches PLOT_MAX, no value is printed. The next line is then started. A line with only a "!" is printed before the next plot line to provide a more aesthetic display on the printer. If a CRT was used, this extra line would probably not be wanted.

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                   02/18/86          PAGE    1

SERIES-III MCS-96 MACRO ASSEMBLER, V1.0

SOURCE FILE: :F2:PLOTSP.A96
OBJECT FILE: :F2:PLOTSP.OBJ
CONTROLS SPECIFIED IN INVOCATION COMMAND: NOSB

ERR LOC  OBJECT                 LINE        SOURCE STATEMENT
                                   1   $pagelength(50)
                                   2
                                   3   PLOT_SERIAL  MODULE STACKSIZE (6)
                                   4
                                   5   ; Intel Corporation,  December 12, 1985
                                   6   ; by Ira Horden, MCO Applications
                                   7
                                   8   ;     This program produces a plot on serially connected printer.  The
                                   9   ; maginitude of each of the 32 input values is plotted horizontally, with one
                                  10   ; "!" followed by a linefeed between each plot line.  Each plot line starts
                                  11   ; with a "+" and the entire plot begins with 3 line feeds and ends with a form
                                  12   ; feed.  The values to be plotted are 32 unsigned words based at the externally
                                  13   ; defined pointer PLOT_IN.
                                  14
                                  15   ;     The routine INIT_OUTPUT must be run to set up the serial port when the
                                  16   ; system is turned on.  CON_OUT can be used by a program to output to the
                                  17   ; serial port.  DRAW_GRAPH is the routine that automatically plots the data.
                                  18
                                  19   ;     Sizing of the graph can be done using PLOT_RES, which determines how many
                                  20   ; units are needed for each dot, and PLOT_MAX, which is the maximum value the
                                  21   ; program will be passed.  Note that (PLOT_MAX/PLOT_RES) defines the maximum
                                  22   ; number of columns the routine will print.
                                  23   ;
                                  24
    0000                          25   RSEG
                                  26           EXTRN   iocl, baud_reg, spcon, spstat, sbuf, portl
                                  27           EXTRN   zero, ax, bx, cx, dx, FFT_MODE
    0000                          28           sptmp:  dsb     1
                                  29
    0024                          30   OSEG at 24H
    0024                          31           value:          dsl     1
    0028                          32           divisor:        dsl     1
    002C                          33           xptr:           dsw     1
    002E                          34           yptr:           dsw     1
    0030                          35           xval:           dsw     1
    0032                          36           log_val:        dsw     1
                                  37
    0000                          38   DSEG
                                  39           EXTRN   PLOT_IN
                                  40
                                  41   $eject
```

270189-64

```
MCS-96 MACRO ASSEMBLER   PLOT_SERIAL                                  02/18/86      PAGE   2

ERR LOC  OBJECT              LINE     SOURCE STATEMENT
                               42
                               43     CSEG at 2500H              ;;;;          PROGRAM  MODULE BEGINS
                               44
                               45     PUBLIC INIT_OUTPUT, CON_OUT, DRAW_GRAPH
                               46     EXTRN  PLOT_RES, PLOT_RES_2, PLOT_MAX
                               47
2500                           48     INIT_OUTPUT:                          ; INITIALIZE SERIAL PORT
                               49
2500 B12000                E   50          ldb    iocl,#00100000B ; set p2.0 to txd
                               51
0270                           52     baud_val    equ    624         ; 624=300 baud (at 12 MHz)
                               53
0082                           54     Baud_high   equ    ((baud_val-1)/256) OR 80H     ; set for XTAL1 clock
006F                           55     baud_low    equ    (baud_val-1) MOD 256
                               56
2503 B16F00                E   57          ldb    baud_reg,#baud_low
2506 B18200                E   58          ldb    baud_reg,#baud_high
                               59
2509 B14900                E   60          ldb    spcon,#01001001b      ; enable reciver mode 1
250C B12000                R   61          ldb    sptmp,#00100000B      ; set TI-tmp
                               62
250F F0                        63          RET
                               64
                               65     $eject
                                                                                        270189-65
```

Listing 5—The Plot Module (Continued)

```
ERR LOC  OBJECT              LINE      SOURCE STATEMENT
                               66
                               67  ;-------------------------------------------------------------------------
                               68  ;                       CONSOLE OUT ROUTINE
                               69  ;
                               70  ;       Call with a word parameter on stack.  The low byte has the character
                               71  ;       to be sent.  If the high byte has a value between 81H and 8FEH, the
                               72  ;       character is repeated 1 to 126 times respectively.  One repeat means
                               73  ;       that the character will be printed 2 times.  If the high byte contains
                               74  ;       a value between 1 and 7FH, the charater represented by that value will
                               75  ;       be printed after the character in the low byte.  If the high byte
                               76  ;       contains a value of zero only the low byte will be printed.
                               77
    2510                       78  CON_OUT:
    2510 CC00            E     79          pop     ax                      ; cx contains the calling adress
    2512 CC00            E     80          pop     dx
    2514 3F011C          E     81          jbs     dx+1,7,onechr           ; If bit 7 is set print one character
    2517 980001          E     82          cmpb    dx+1,zero
    251A DF17                  83          je      onechr                  ; if highbyte=0 print one character
                               84
    251C 900000          E     85  twochr: orb     sptmp,spstat            ; wait for TI
    251F 3500FA          R     86          jbc     sptmp,5,twochr
    2522 71DF00          R     87          andb    sptmp,#11011111b        ; clear TI-tmp
    2525 900000          E     88          orb     zero,spstat             ; remove possible false TI
                               89
    2528 B00000          E     90          ldb     sbuf,dx
    252B B00100          E     91          ldb     dx,dx+1         ; Load second character
    252E 1101            E     92          clrb    dx+1            ; clear count byte
    2530 717F00          E     93          andb    dx,#07FH        ; mask MSB
                               94
    2533 1701            E     95  onechr: incb    dx+1
    2535 717F01          E     96          andb    dx+1,#7FH
    2538 900000          E     97  wait1:  orb     sptmp,spstat            ; wait for TI
    253B 3500FA          R     98          jbc     sptmp,5,wait1
    253E 71DF00          R     99          andb    sptmp,#11011111b        ; clear TI-tmp
    2541 900000          E    100          orb     zero,spstat             ; remove possible false TI
                              101
    2544 B00000          E    102          ldb     sbuf,dx
    2547 E001EE          E    103          DJNZ    dx+1,wait1
    254A E300            E    104          BR      [ax]                    ; Effectively a RET
                              105
                              106  $eject
```

270189-66

Listing 5—The Plot Module (Continued)

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                  02/18/86          PAGE    4

ERR LOC  OBJECT                LINE       SOURCE STATEMENT
                               107   ;-------------------------------------------------------------------------
                               108   ;                          PRINT  DECIMAL  NUMBER  ROUTINE
                               109
                               110   ;       Call with two words on stack.  The first is the value to be printed.
                               111   ;       The second has mode information in the low byte.
                               112   ;                 MODE:   000 = supress all zeros
                               113   ;                         001 = print all numbers
                               114   ;                         010 = supress all zeros except rightmost
                               115   ;                         1xx = do not print leading blanks
                               116   ;
                               117   ;       The high byte of the 2nd word = 2x the number of places to be printed
                               118
                               119
    254C                       120   PRINT_NUM:                  .  ; Send Decimal number to CON_OUT
    254C CC00              E   121           pop     cx
    254E CC00              E   122           pop     bx              ; bx is mode byte, bx+1 is divisor pointer
    2550 AC0100            E   123           ldbze   dx,bx+1
    2553 A300962528        E   124           ld      divisor,divtab[dx]
    2558 CC24                  125           pop     value
    255A                       126   div_loop:
    255A 0126                  127           clr     value+2
    255C 8C2824                128           divu    value,divisor           ; divide ax,dx by divisor
    255F 380017            E   129           jbs     bx,0,chr_ok             ; print character regardless of value
    2562 980024            E   130           cmpb    value,zero
    2565 D70F                  131           jne     non_0                   ; jump if value is non zero
    2567                       132   Val_0:                          ; Value is zero
    2567 310003            E   133           jbc     bx,1,prntsp             ; Print space instead of 0
    256A 38280C                134           jbs     divisor,0,chr_ok        ; If in rightmost position print 0
    256D 3A0015            E   135   prntsp: jbs     bx,2,cont       ; Do not print space if bit is set
    2570 A1F00024              136           ld      value,#0F0H     ;  0F0h+30h = 20H = space
    2574 2003                  137           br      chr_ok
                               138
    2576 910100            E   139   non_0:  orb     bx,#0001B               ; Set flag so 0's will be printed
    2579 65300024              140   chr_ok: add     value,#30h              ; 30h + n = 0 to 9 ascii
    257D 617F0024              141           and     value,#7Fh      ; send least sig seven bits, clear upper word
    2581 C824                  142           push    value
    2583 2F8B                  143           call    con_out         ; output ascii result (result<9)
    2585 A02624                144   cont:   ld      value,value+2   ; load value with remainder
    2588 012A                  145           clr     divisor+2
    258A 8D0A0028              146           divu    divisor,#10     ; next lower power of ten
    258E 880028            E   147           cmp     divisor,zero
    2591 D7C7                  148           jne     div_loop
    2593                       149   div_done:
    2593 E300              E   150           br      [cx]
                               151
    2596                       152   DIVTAB:         ;       Number of places for result
    2596 000001000A006400      153           dcw     0, 1, 10, 100, 1000, 10000      ; divisor table - 10**n
```

270189-67

Listing 5—The Plot Module (Continued)

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                   02/18/86          PAGE    5

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                               154
                               155
                               156
                               157
    25A2                       158   DRAW_GRAPH:                     ; Graph drawing routine
    25A2 C90D00                159           push    #0dh
    25A5 2F69                  160           call    con_out
    25A7 C90A82                161           push    #820AH          ;;;     Clear 3 lines
    25AA 2F64                  162           call    CON_OUT
    25AC C90000                163           push    #00
    25AF 2F5F                  164           call    CON_out
                               165
    25B1 012C                  166           clr     xptr
    25B3 0130                  167           clr     xval
    25B5                       168   NXT_ROW:
    25B5 C90D0A                169           push    #0A0DH          ; CRLF
    25B8 2F56                  170           call    CON_OUT
    25BA C90000                171           push    #00H            ; nul
    25BD 2F51                  172           call    CON_OUT
                               173
    25BF C830                  174           push    xval
    25C1 C9020A                175           push    #(0A00H or 0010b)       ; supress all zeros except rightmost
    25C4 2F86                  176           call    PRINT_NUM
                               177
    25C6 C92020                178           push    #2020H          ; Print 2 spaces
    25C9 2F45                  179           call    CON_OUT
    25CB C92B00                180           push    #2BH            ; +
    25CE 2F40                  181           call    con_out
                               182
    25D0 A100002E          E   183           ld      yptr,#PLOT_RES_2        ; PLOT_RES_2 = PLOT_RES/2
                               184                                           ; PLOT_RES is defined 7 lines down
                               185
    25D4                       186   NXT_COL:                                ; Next Column
    25D4 8B2D00002E        E   187           cmp     yptr,PLOT_IN[xptr]
    25D9 D911                  188           jh      PRT_NUM
    25DB                       189   PRT_MK:                                 ; Print Mark
    25DB C92A00                190           push    #2AH
    25DE 2F30                  191           call    CON_OUT
    25E0                       192   INC_CNT:
    25E0 6500002E          E   193           add     yptr,#PLOT_RES  ; PLOT_RES = number of inputs per output point
    25E4 8900002E          E   194           cmp     yptr,#PLOT_MAX  ; PLOT_max = maximum line length
    25E8 D1EA                  195           jnh     nxt_col
    25EA 204F                  196           br      NXTLN
                               197   $eject
```

270189-68

Listing 5—The Plot Module (Continued)

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                  02/18/86          PAGE    6

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                               198
    25EC                       199   PRT_NUM:
    25EC 8900002E          E   200           cmp     yptr,#PLOT_RES_2        ; If value is less then minimum needed
    25F0 DF49                  201           be      NXTLN                   ; for a plot, do not print value
                               202
    25F2 C92020                203           push    #2020H                  ; print 2 spaces then value
    25F5 2F19                  204           call    con_out
    25F7 3B000B            E   205           JBS     FFT_MODE,3,db_mode
                               206
    25FA                       207   norm_mode:
    25FA CB2D0000          E   208           push    PLOT_IN[xptr]
    25FE C9000A                209           push    #(0A00H or 0000B)       ; supress all zeros
    2601 2F49                  210           call    PRINT_NUM
    2603 2036                  211           BR      NXTLN
                               212
    2605                       213   db_mode:
    2605 A32D00002E        E   214           ld      yptr,plot_in[xptr]      ; PLOT_IN = 512*10*LOG(x)
    260A 08012E                215           shr     yptr,#1                 ; yptr=265 * 10LOG(x)
    260D AC2F00            E   216           ldbze   ax,yptr+1               ; ax= 10LOG(x) = yptr/256
                               217
    2610 C800              E   218           push    ax                      ; Print AX
    2612 C9020A                219           push    #(0A00H or 0010B)       ; supress all but rightmost zero
    2615 2F35                  220           call    PRINT_NUM
    2617 C92E00                221           push    #2EH                    ; Decimal point
    261A 2EF4                  222           call    con_out
                               223
    261C B02E01            E   224           ldb     ax+1,yptr       ; high byte of ax = fractional portion of
    261F 1100              E   225           clrb    ax              ;                   10LOG(x)
                               226
    2621 6DE60300          E   227           mulu    ax,#3E6H        ; if ax=FF00H then ax+2 now = 998 decimal
    2625 370102            E   228           jbc     ax+1,7,no_rnd
    2628 0700              E   229           inc     dx              ; round value up
                               230
    262A C800              E   231   no_rnd: push    dx      ; dx=ax+2
    262C C90106                232           push    #(600H or 0001B)        ; print all numbers to three places
    262F 2F1B                  233           call    Print_num
    2631 C92000                234           push    #20H            ; space
    2634 2EDA                  235           call    con_out
    2636 C96442                236           push    #4264H          ; "dB"
    2639 2ED5                  237           call    con_out
                               238
                               239   $eject
```

270189-69

```
MCS-96 MACRO ASSEMBLER    PLOT_SERIAL                                   02/18/86          PAGE    7

ERR LOC  OBJECT                LINE        SOURCE STATEMENT
                                240
                                241                                  ; Setup for next line
    263B C90D0A                 242   NXTLN:  push    #0A0DH         ; CRLF
    263E 2ED0                   243           call    CON_OUT
    2640 C90000                 244           push    #00H           ; nul
    2643 2ECB                   245           call    CON_OUT
    2645 C92086                 246           push    #8620H         ; 7 spaces
    2648 2EC6                   247           call    CON_OUT
    264A C92100                 248           push    #21H           ; !
    264D 2EC1                   249           call    con_out
                                250
    264F 0730                   251           inc     xval
    2651 6502002C               252           add     xptr,#2
    2655 893E002C               253           cmp     xptr,#62
    2659 D2022758               254   !       ble     nxt_row        ; Start printing next row
                                255
    265D C90D0A                 256   Done:   push    #0A0DH         ; CRLF  ; Form feed for next graph
    2660 2EAE                   257           call    CON_OUT
    2662 C9000C                 258           push    #0C00H         ; null,FF
    2665 2EA9                   259           call    con_out
                                260
    2667 F0                     261           RET
    2668                        262   END

ASSEMBLY COMPLETED,   NO ERROR(S) FOUND.
```

270189-70

Listing 5—The Plot Module (Continued)

At the end of the plot, a form feed is given to set the printer up for the next graph. Our printer would frequently miss the character after a CRLF. To solve this problem, a null (ASCII 0) is sent after every CRLF to make sure the printer is ready for the next line. This has been found to be a problem with many devices running at close to their maximum capacity, and the nulls work well to solve it.

With the plot completed, the program begins to run again by taking another set of A to D samples.

11.0 USING THE FFT PROGRAM

The program can be used with either real or tabled data. If real data is used, the signal is applied to analog channel 1. The program as written performs A/D samples at 100 microsecond intervals, collecting the 64 samples in 6.4 milliseconds. This sets the sampling window frequency at 156 Hz. If tabled data is used, 64 words of data should be placed in the location pointed to by DATA0 in the TABLE_LOAD routine of the Main Module.

Program control is specified by FFT_MODE which is loaded in the main module. Also within the main module are settings which control the A to D buffer routine and the Plot routine. The intention was to have only one module to change and recompile to vary parameters in the entire program.

The program modules are set up to run one-at-a-time so that the code would be easy to understand. Additionally, the Plot routine takes so long relative to the other sections, that it doesn't pay to try to overlap code sections. If this code were to be converted to run a process instead of print a graph, it might be worthwhile to run the FFT and the A/D routines at the same time.

If the goal of a modified program is to have the highest frequency sampling possible, it might be desirable to streamline the A/D section and run it without interruption. When the A to D routine was complete the FFT routine could be started. The reasoning behind this is that at the fastest A/D speeds the processor will be almost completely tied up processing the A/D information and storing it away. Using an interrupt based A/D routine would slow things down.

A set of programs which will perform a FFT has been presented in this application note. These programs are available from the INSITE users library as program CA-26. More importantly, dozens of programing examples have been made available, making it easier to get started with the 8096. Examples of how to use the hardware on the 8096 have already appeared in AP-248, "Using The 8096". These two applications notes form a good base for the understanding of MCS-96 microcontroller based design.

12.0 APPENDIX A - MATRICES

Matrices are a convenient way to express groups of equations. Consider the complex discrete Fourier Transform in equation 9, with N = 4.

$$Y_n = \sum_{k=0}^{3} X(k)\, W^{nk} \qquad n = 0, 1, 2, 3$$

This can be expanded to

$$\begin{aligned}
Y(0) &= X(0)\,W^0 + X(1)\,W^0 + X(2)\,W^0 + X(3)\,W^0 \\
Y(1) &= X(0)\,W^0 + X(1)\,W^1 + X(2)\,W^2 + X(3)\,W^3 \\
Y(2) &= X(0)\,W^0 + X(1)\,W^2 + X(2)\,W^4 + X(3)\,W^6 \\
Y(3) &= X(0)\,W^0 + X(1)\,W^3 + X(2)\,W^6 + X(3)\,W^9
\end{aligned}$$

In matrix notation, this is shown as

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 \\ W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The first step to simplifying this is to reduce the center matrix. Recalling that

$$W^N = W^{N \bmod N} \quad \text{and} \quad W^0 = 1$$

The matrix can be reduced to have less non-trivial multiplications.

$$\begin{bmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W^1 & W^2 & W^3 \\ 1 & W^2 & W^0 & W^2 \\ 1 & W^3 & W^2 & W^1 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

The square matrix can be factored into

$$\begin{bmatrix} Y(0) \\ Y(2) \\ Y(1) \\ Y(3) \end{bmatrix} = \begin{bmatrix} 1 & W^0 & 0 & 0 \\ 1 & W^2 & 0 & 0 \\ 0 & 0 & 1 & W^1 \\ 0 & 0 & 1 & W^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & W^0 & 0 \\ 0 & 1 & 0 & W^0 \\ 1 & 0 & W^2 & 0 \\ 0 & 1 & 0 & W^2 \end{bmatrix} \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix}$$

For this equation to work, the Y(1) and Y(2) terms need to be swapped, as shown above. This procedure is a Bit Reversal, as described in the text.

Multiplying the two rightmost matrices results in

$$\begin{aligned}
&X(0) + X(2)\,W^0 \\
&X(1) + X(3)\,W^0 \\
&X(0) + X(2)\,W^3 \\
&X(1) + X(3)\,W^2
\end{aligned}$$

requiring 4 complex multiplications & 4 complex additions

Noting that $W^0 = -W^2$, 2 of the complex multiplications can be eliminated, with the following results

$$\begin{aligned}
&X(0) + X(2)\,W^0 \\
&X(1) + X(3)\,W^0 \\
&X(0) - X(2)\,W^0 \\
&X(1) - X(3)\,W^0
\end{aligned}$$

requiring 2 complex multiplications and 4 complex additions

Since $W^1 = -W^3$, a similar result occurs when this vector is multiplied by the remaining square matrix. The resulting equations are:

$$\begin{aligned}
Y(0) &= (X(0) + X(2)\,W^0) + W^0\,(X(0) + X(3)\,W^0) \\
Y(2) &= (X(0) + X(2)\,W^0) - W^0\,(X(1) + X(3)\,W^0) \\
Y(1) &= (X(0) - X(2)\,W^0) + W^1\,(X(1) - X(3)\,W^0) \\
Y(3) &= (X(0) - X(2)\,W^0) - W^1\,(X(1) - X(3)\,W^0)
\end{aligned}$$

The number of complex multiplications required is 4, as compared with 16 for the unfactored matrix.

In general, the FFT requires

$$\frac{N * EXPONENT}{2} \text{ complex multiplications}$$

and

$$N * EXPONENT \text{ complex additions}$$

where

$$EXPONENT = \text{Log}_2\, N$$

A standard Fourier Transform requires

$$N^2 \text{ complex multiplications}$$

and

$$N(N-1) \text{ complex additions}$$

13.0 APPENDIX B - PLOTS

The following plots are examples of output from the FFT program. These plots were generated using tabled data, but very similar plots have also been made using the analog input module. Typically, a plot made using the analog input module will not show quite as much power at each frequency and will show a positive value for the DC component. This is because it is difficult to get exactly a full-scale analog input with no DC offset.

Plot 1 is a Magnitude plot of a square wave of period NT.

Plot 2 is the same data plotted in dB. Note how the dB plot enhances the difference in the small signal values at the high frequencies.

Plot 3 shows the windowed version of this data. Note that the widening of the bins due to windowing shows energy in the even harmonics that is not actually present. For data of this type a different window other than Hanning would normally be used. Many window types are available, the selection of which can be determined by the type of data to be plotted.[3]

Plot 4 shows a sine wave of period NT/7 or frequency 7/NT.

Plot 5 shows the same input with windowing. Note the signal shown in bins 6 and 8.

Plot 6 shows a sine wave of period NT/7.5. Note the noise caused by the discontinuity as discussed earlier.

Plot 7 uses windowing on the data used for plot 6. Note the cleaner appearance.

Plot 8 shows a sine wave input of magnitude 0.707 and period NT/7.5.

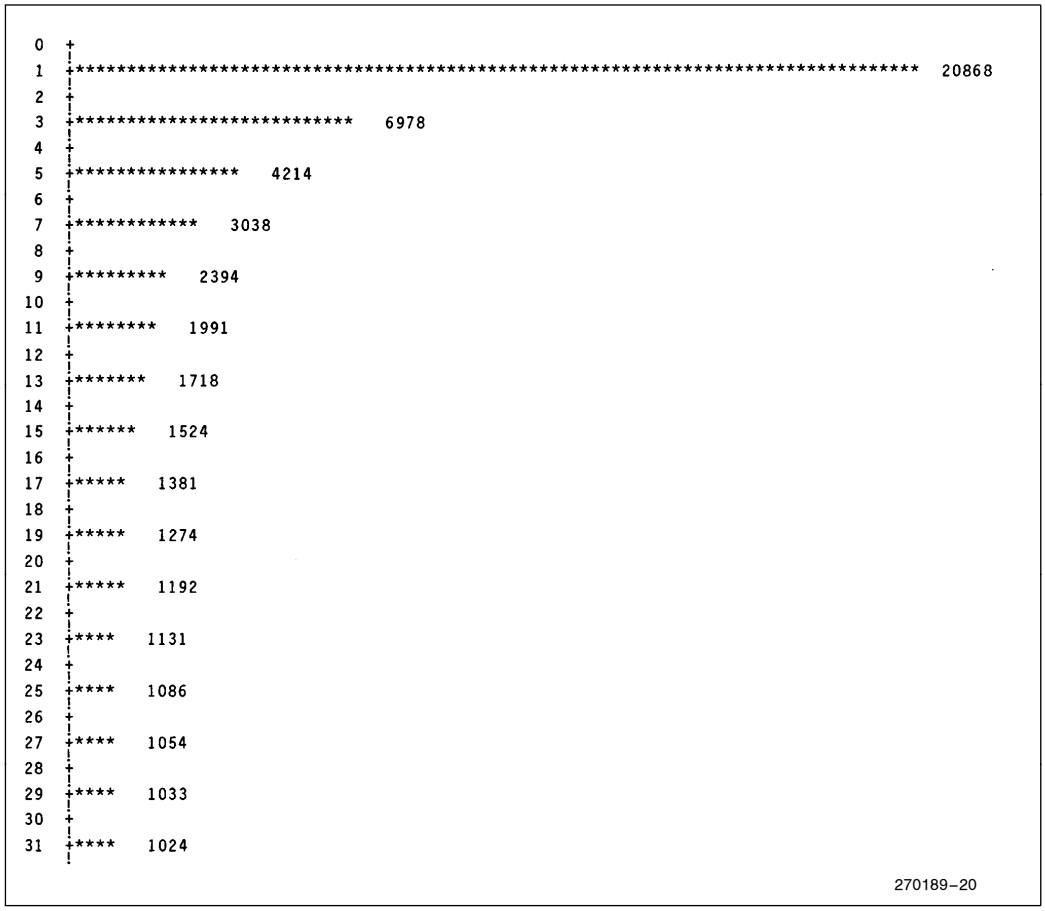
Plot 9 shows same input with windowing.

Plot 10 shows a sine wave of magnitude 0.707/16 and period NT/11.

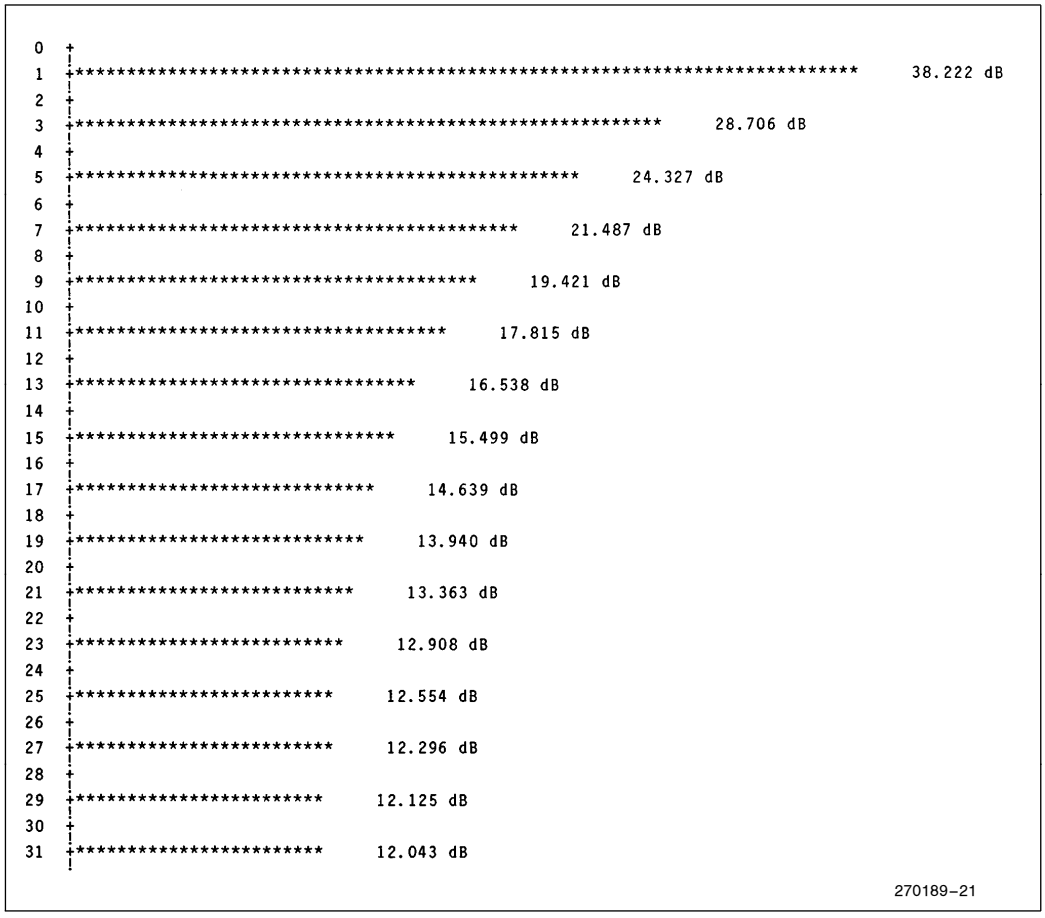
Plot 11 shows the same input with windowing. Note that there is no power shown in bins 10 and 12. This is because at 6 dB down from 3 dB they are nearly equal to zero.

Plot 12 uses the sum of the signals for plots 8 and 10 as inputs. Note that the component at period NT/11 is almost hidden.

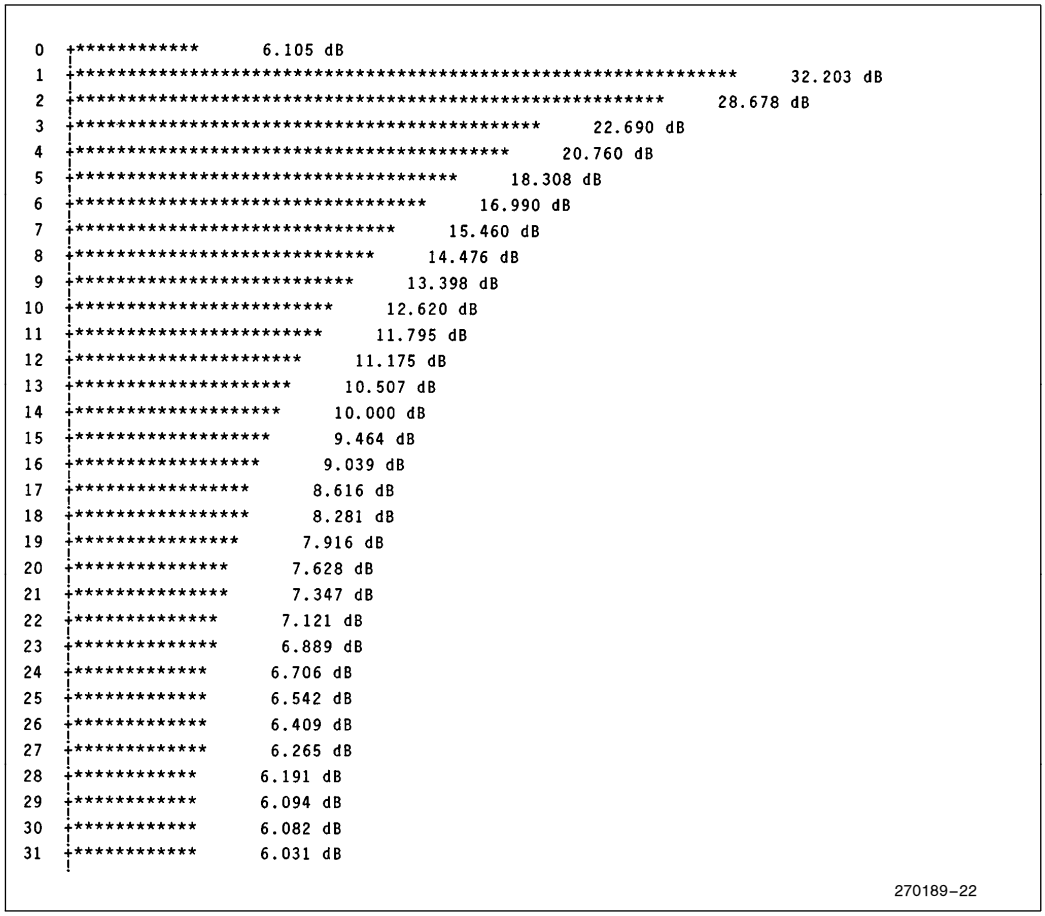
Plot 13 uses the same signal as plot 12 but applies windowing. Now the period component at NT/11 can easily be seen. The Hanning window works well in this case to separate the signal from the leakage. If the signals were closer together the Hanning window may not have worked and another window may have been needed.



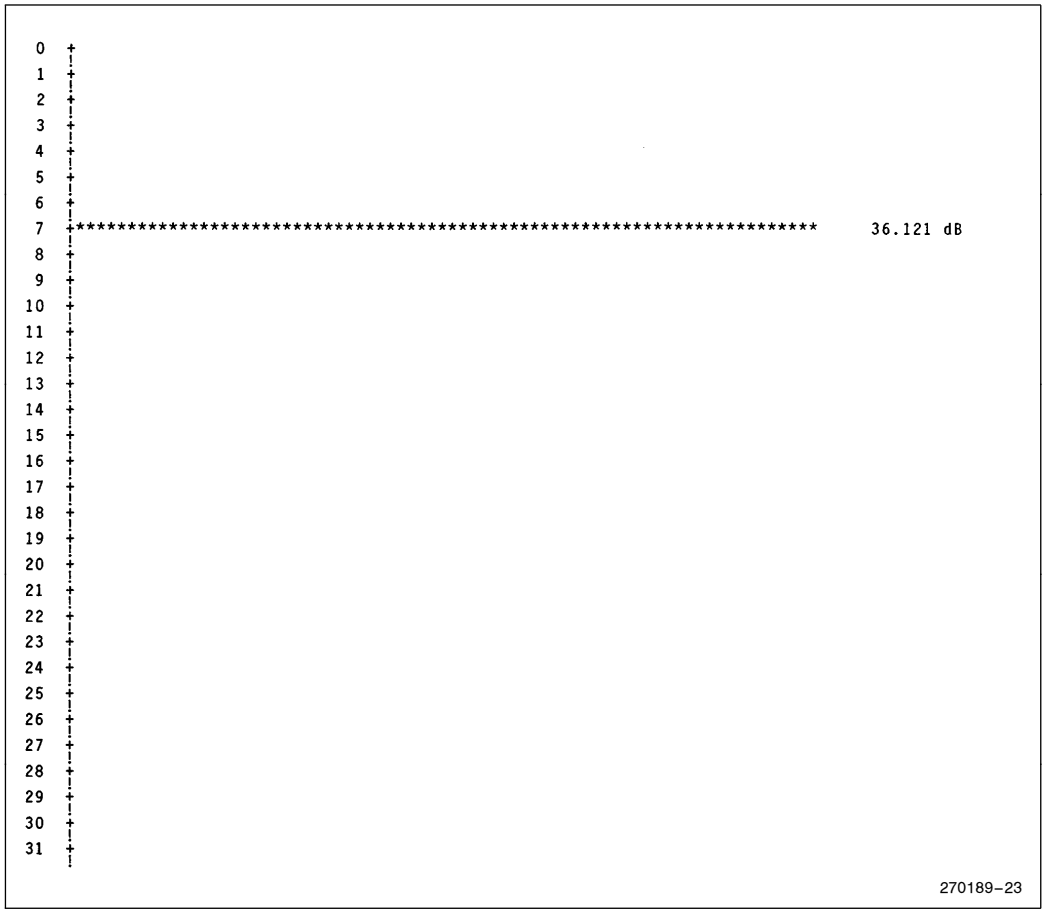
Plot 1—Magnitude Plot of Squarewave



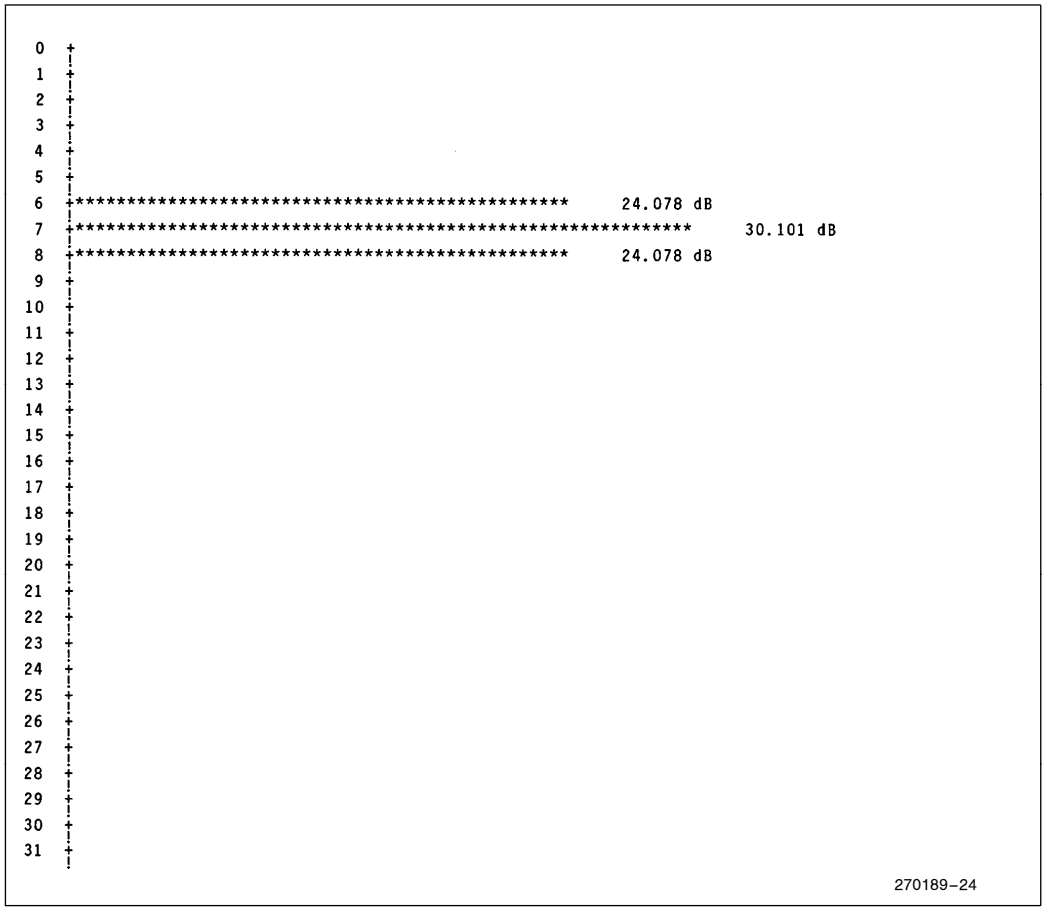
Plot 2—Decibel Plot of Squarewave



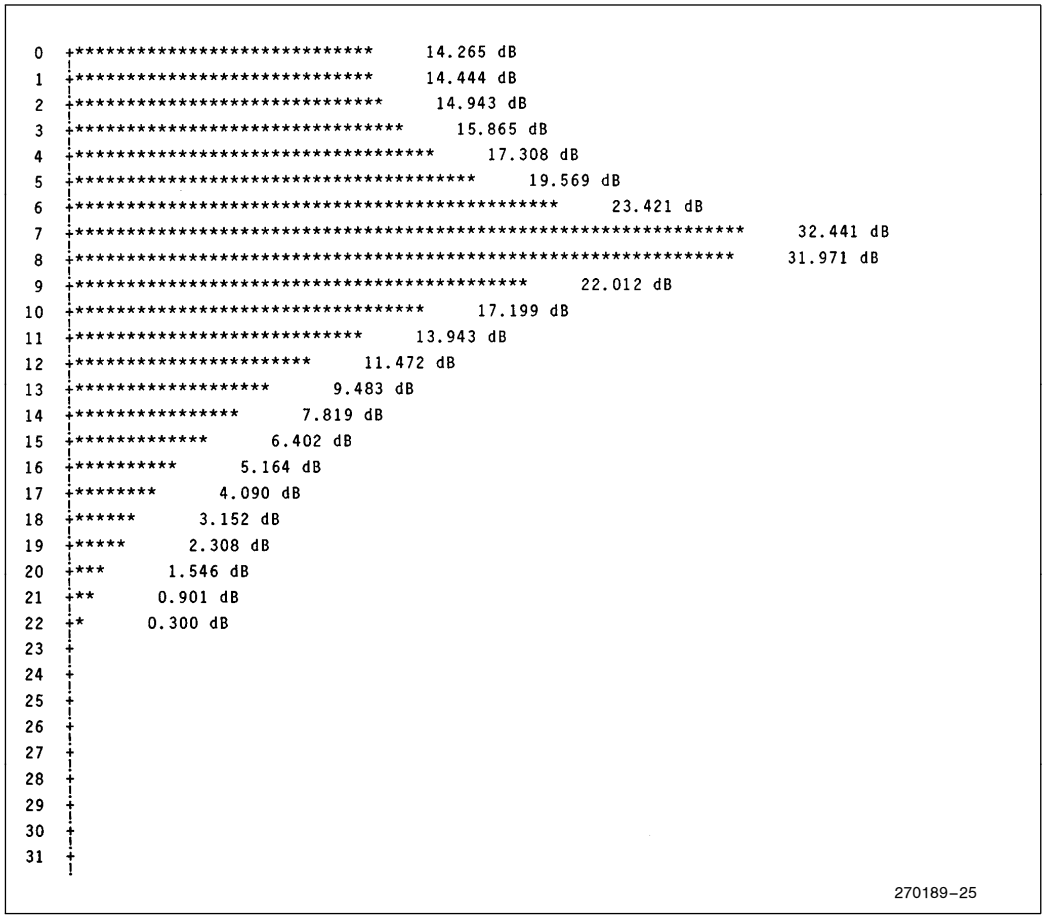
Plot 3—Plot of Squarewave with Window



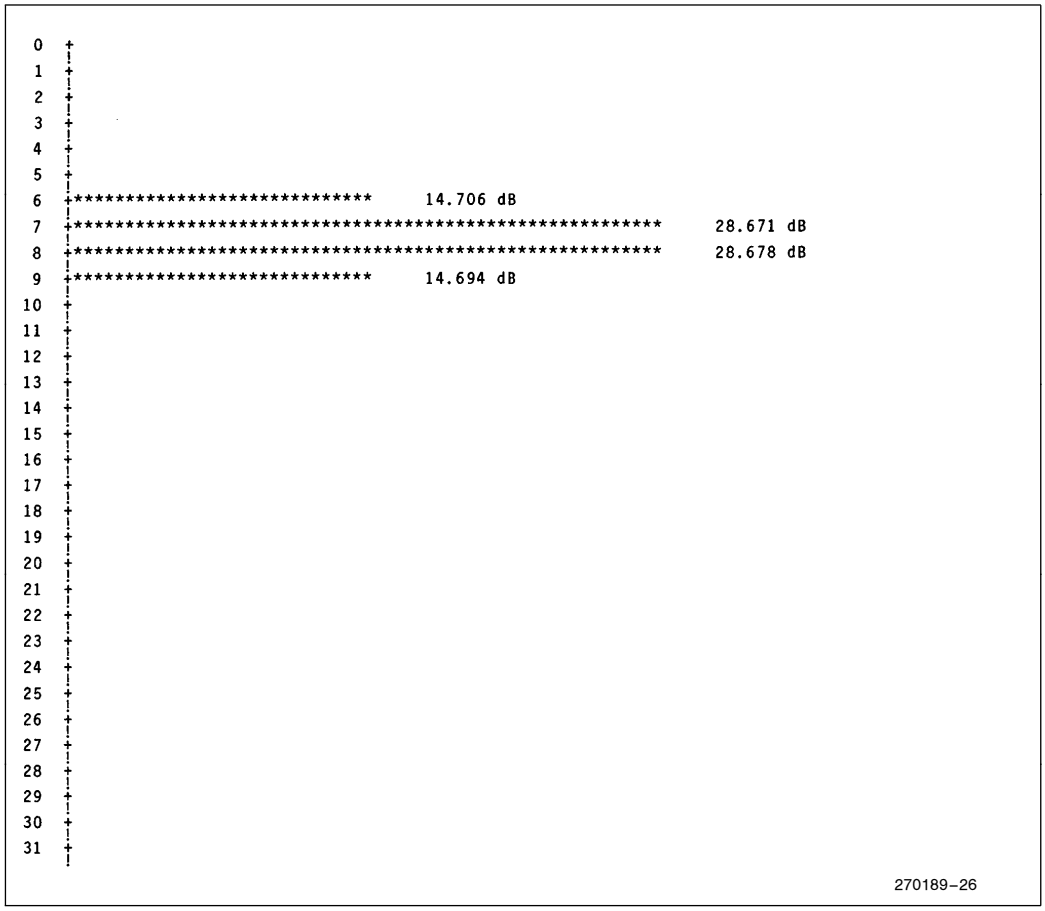
Plot 4—Sin (7.0X) without Window



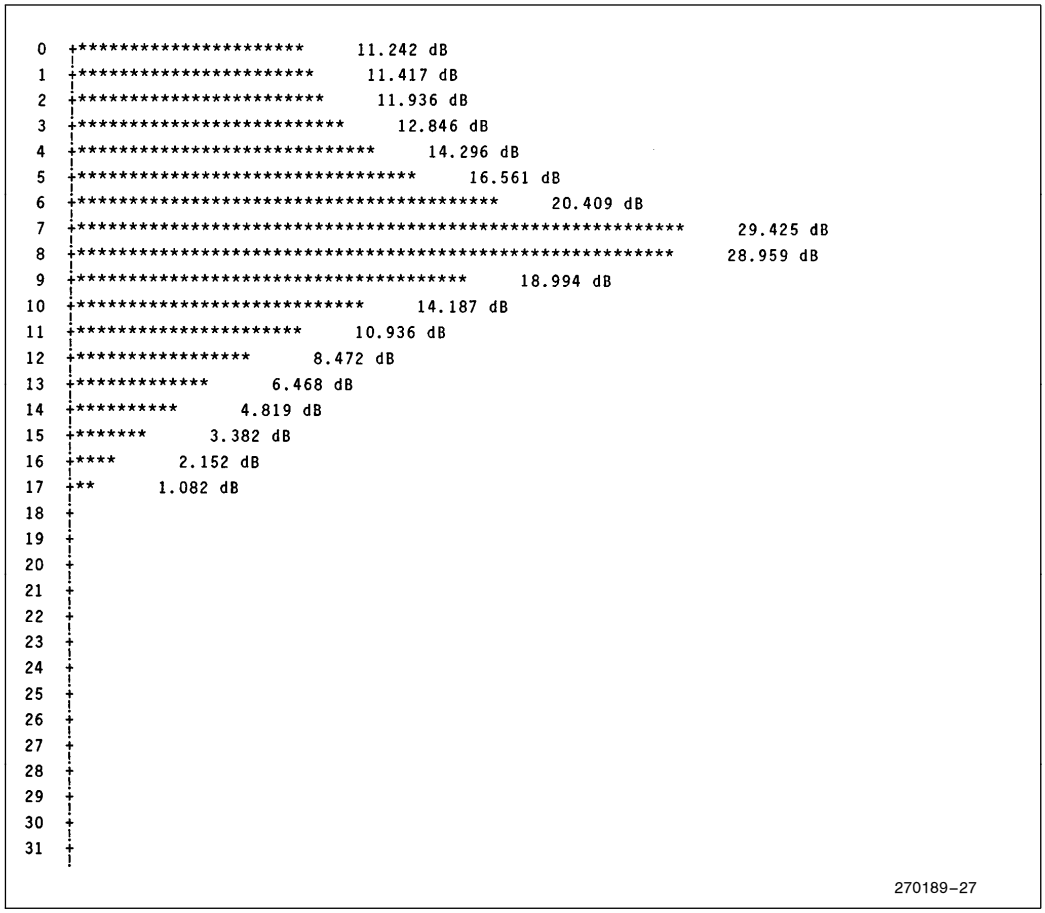
Plot 5—Sin (7.0X) with Window



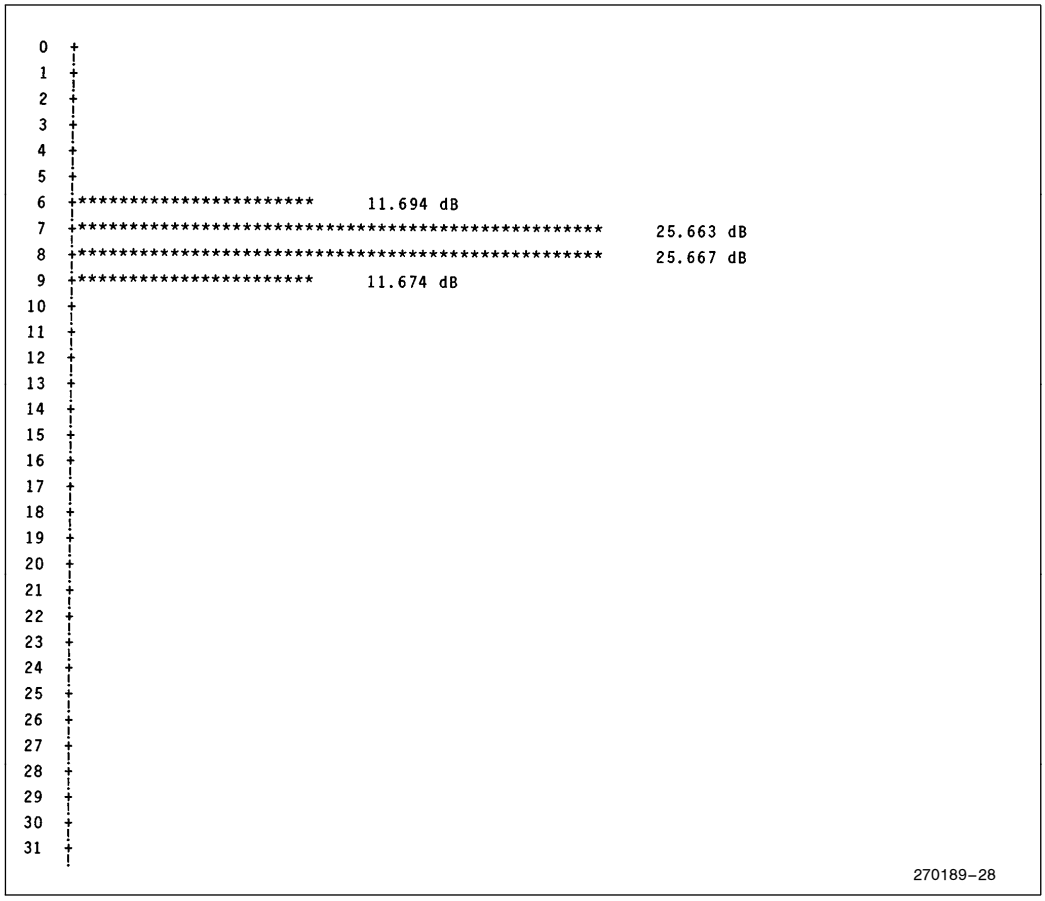
Plot 6—Sin (7.5X) without Window



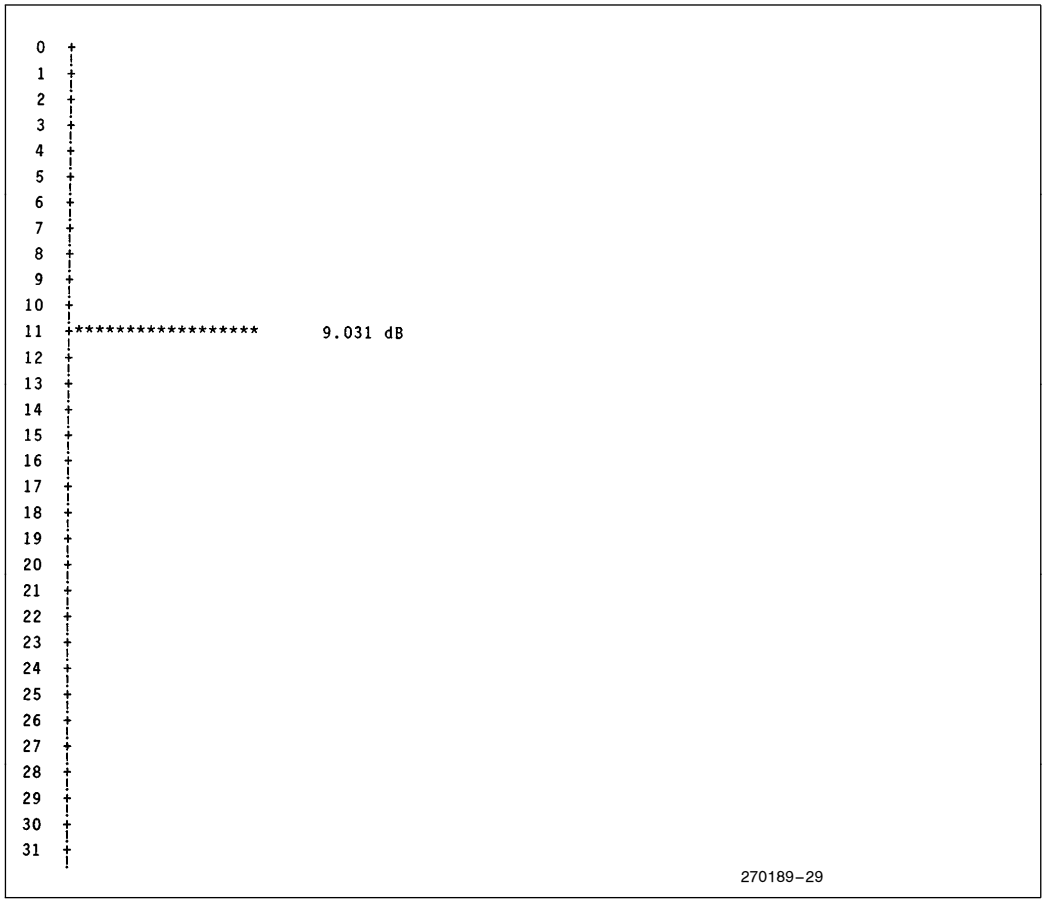
Plot 7—Sin (7.5X) with Window



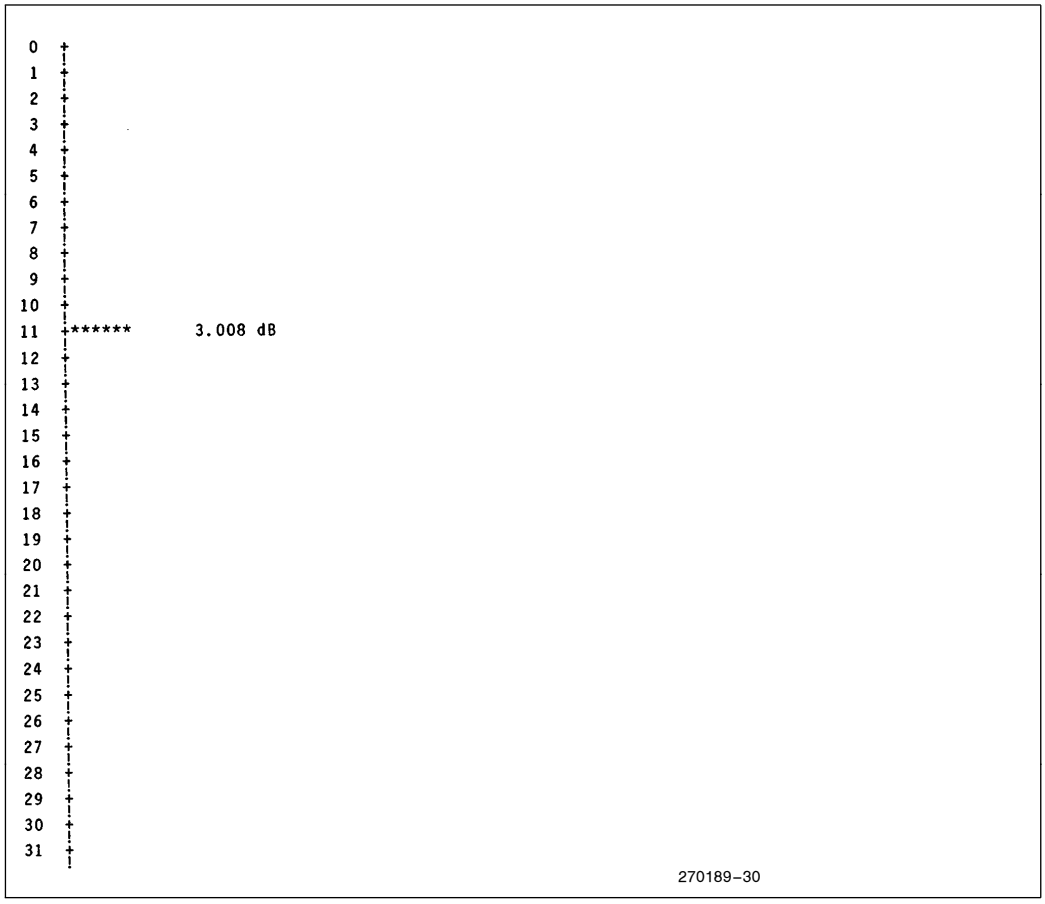
Plot 8—0.707 * Sin (7.5X) without Window



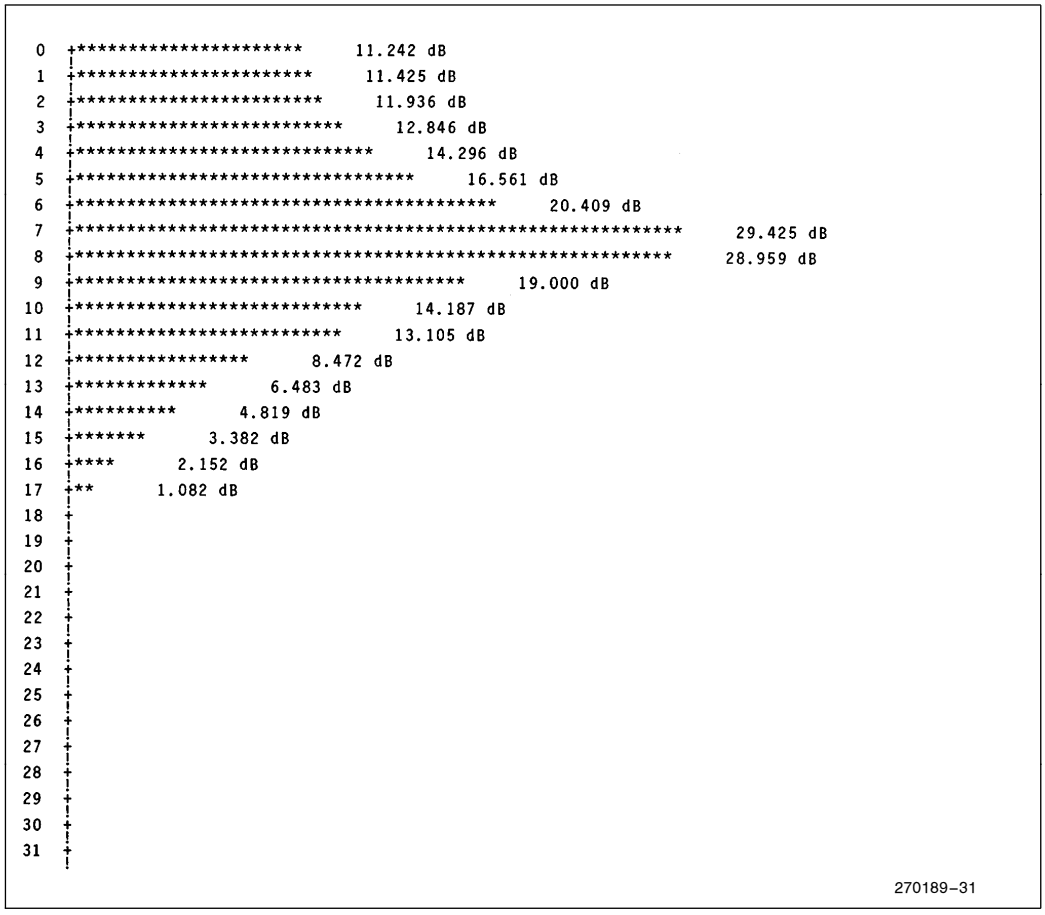
Plot 9—0.707 * Sin (7.5X) with Window



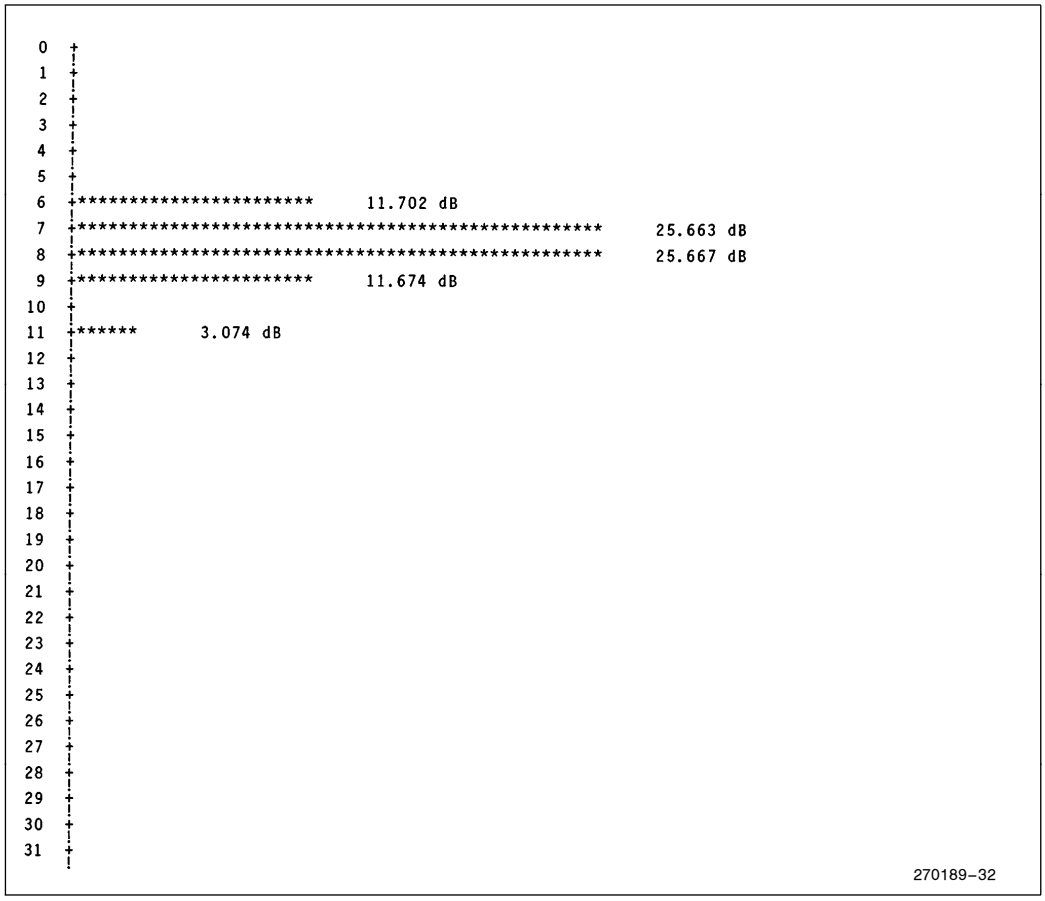
Plot 10—0.707/16 * Sin (11X) without Window



Plot 11—0.707/16 * Sin (11X) with Window



Plot 12—0.707 (Sin (7.5X) + 1/16 Sin (11X)) without Window



Plot 13—0.707 (Sin (7.5X) + 1/16 Sin (11X)) with Window

BIBLIOGRAPHY

1. Boyet, Howard and Katz, Ron, The 16-Bit 8096: Programming, Interfacing, Applications. 1985, Microprocessor Training Inc., New York, NY.
2. Brigham, E. Oran, The Fast Fourier Transform. 1974, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.
3. Harris, Fredric J., On the use of Windows for Harmonic Analysis with the Discrete Fourier Transform. Proceedings of the IEEE, Vol. 66, No. 1, January 1978.
4. Weaver, H. Joseph, Applications of discrete and continuous Fourier analysis. 1983, John Wiley and Sons, New York.

INTEL PUBLICATIONS

1. 1986 Microcontroller Handbook, Order Number 210918-004
2. Using the 8096, AP-248, Order Number 270061-001
3. MCS-96 Macro Assembler User's Guide, Order Number 122048-001
4. MCS-96 Utilities User's Guide, Order Number 122049-001

INTEL CORPORATION, 2200 Mission College Blvd., Santa Clara, CA 95052; Tel. (408) 765-8080

INTEL CORPORATION (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 696 000

INTEL JAPAN k.k., Ibaraki-ken; Tel. 029747-8511

Microcontroller Operation
Printed in U.S.A./TP479/0296/5K/RP SM