Tektronix®
COMMITTED TO EXCELLENCE

Please Check for
CHANGE INFORMATION
at the Rear of this Manual

4050 SERIES
R08
SIGNAL PROCESSING ROM PACK NO. 2 (FFT)

INSTRUCTION MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

MANUAL PART NO. 070-2841-00
PRODUCT GROUP 14

First Printing JUN 1979
Revised ~~SEP 1982~~ JUL 1984

This instrument, in whole or in part, may be protected by one or more U.S. or foreign patents or patent applications. Information provided on request by Tektronix, Inc., P.O. Box 500, Beaverton, Oregon 97077.

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the users at their own expense will be required to take whatever measures may be required to correct the interference.

MANUAL REVISION STATUS

PRODUCT: **4051 R08 Signal Processing ROM Pack No. 2 (FFT)**
4052 R08 Signal Processing ROM Pack No. 2 (FFT)

This manual supports the following versions of this product: Serial Numbers B010100 and up.

REV DATE	DESCRIPTION
JUN 1979	Original Issue
JUL 1981	Revised: page ii.
AUG 1981	Revised: pages A-3 and A-5.
SEP 1982	Revised: page 2-16.

CONTENTS

		Page
Section 1	**GENERAL DESCRIPTION**	
	Introduction	1-1
	Specifications	1-2
	Electrical	1-2
	Environmental	1-2
	Physical	1-3
	Standard Accessories	1-3
	Installing the ROM Pack	1-4
Section 2	**COMMAND DESCRIPTIONS**	
	Introduction	2-1
	Definition of Terms	2-1
	The FFT Command	2-3
	The IFT Command	2-8
	The POLAR Command	2-12
	The CORR Command	2-15
	The CONVL Command	2-18
	The TAPER Command	2-21
	The INLEAV Command	2-24
	The UNLEAV Command	2-26
Section 3	**DEMONSTRATION-VERIFICATION PROGRAM**	
	Introduction	3-1
	Running the Sample Program	3-1
	Understanding the Program	3-3
Section 4	**REPLACEABLE PARTS LIST**	
Section 5	**DIAGRAMS**	

Appendix A **AN INTRODUCTION TO THE DFT AND FFT ALGORITHM OF SIGNAL PROCESSING ROM PACK NO. 2**

The Discrete Fourier Transform and Integral Fourier Transform A-1
The Discrete Fourier Transform of ROM Pack No. 2 ..... A-3
Polar Form of Representing the Fourier Transform A-5
Aliasing and Time Truncation Errors A-5
Controlling Errors Due to Alias A-6
Controlling Time Truncation Error A-9
The FFT Algorithm and Accumulated Roundoff Errors A-10

Appendix B **BIBLIOGRAPHY**

Appendix C **COMMAND SUMMARY**

Appendix D **UNDERSTANDING ERRORS**

ILLUSTRATIONS

Figure	Description	Page
1-1	Signal Processing ROM Pack No. 2	v
1-2	Installing the ROM Pack in the Graphic Computing System	1-4
2-1	Illustration of 10%, 25%, and 50% Cosine Window Tapering	2-22
3-1	Original Signal	3-4
3-2	Complex FFT — Interleaved Reals and Imaginaries	3-5
3-3	FFT — Magnitude and Phase	3-6
3-4	FFT — Reals and Imaginaries	3-7
3-5	IFT of FFT of Original Signal	3-8
3-6	Autocorrelation of Original Signal	3-9
3-7	Convolution of Original with IFT of FFT of Original	3-10
3-8	Tapered Windows — 10%, 25%, 50% Tapering	3-11

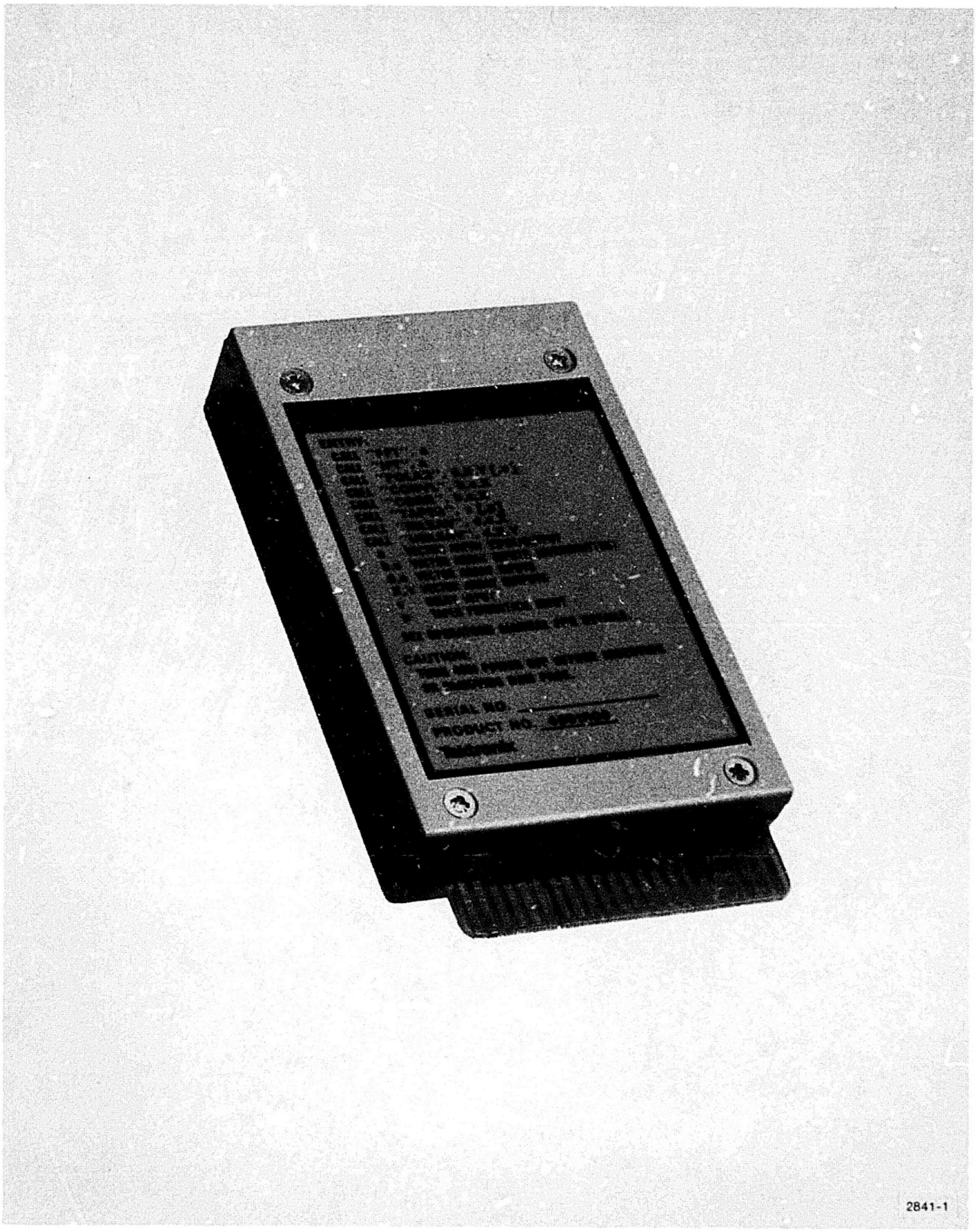


Figure 1-1. Signal Processing ROM Pack No. 2

Section 1

GENERAL DESCRIPTION

INTRODUCTION

The TEKTRONIX 4051 R08 Signal Processing ROM Pack No. 2 (FFT)and the Tektronix 4052 R08 Signal Processing ROM Pack No. 2 (FFT) are Read Only Memory (ROM) devices designed to be used with TEKTRONIX 4050 Series Graphic Computing Systems. The 4051 ROM Pack is used only with a 4051 Graphic Computing System. The 4052 ROM Pack is used with either the 4052 or 4054 Graphic Computing Systems.

The 4051 and 4052 ROM Packs enable the System to perform complicated signal processing operations on waveforms or other data stored in one-dimensional arrays. This additional capability does not alter the operation of the Graphic Computing System or occupy any of its available RAM (Random Access Memory) space.

The Signal Processing ROM Pack provides eight mathematical operations that are essential to advanced signal processing and data analysis. Four of these (FFT, IFT, CONVL, and CORR) are frequently used signal processing operations. The remaining commands (POLAR, TAPER, UNLEAV, and INLEAV) are utility routines designed to refine or convert the data into more useful formats. A summary of each command follows:

FFT — Computes the fast Fourier transform of an array, placing the results in the same array.[1]

IFT — Computes the inverse Fourier transform of an array, placing the results in the same array.[1]

CONVL — Convolves two input arrays, placing the results in a third array.[1]

CORR — Correlates two input arrays, placing the results in a third array.[1]

POLAR — Converts an array of FFT data from rectangular form (reals and imaginaries) to polar form (magnitude and phase).

TAPER — Multiplies an array by a cosine window of program-selectable tapering weights. When tapering is selected to be 50%, it provides a Hanning window.

UNLEAV — Sorts an array of interleaved FFT data into two arrays, one containing real and one containing imaginary components.

---

[1] **Operations must be performed on one-dimensional arrays.**

INLEAV — Interleaves the real and imaginary data from two input arrays into a third array whose format is acceptable to the IFT command.

Operations provided by ROM Pack No. 2 are executed much faster than BASIC program for the same function. For example, the FFT command computes a Fourier transform approximately eight times faster than an equivalent BASIC program.

An additional benefit of the ROM Pack is that the signal processing routines require no RAM space. The routines are permanently stored in ROM Pack memory, leaving the System RAM space free for BASIC programs and data.

SPECIFICATIONS

Electrical

Power Requirements

The Signal Processing ROM Pack No. 2 draws all necessary power from the 4050 Series Graphic Computing System power supplies.

Environmental

The Signal Processing ROM Pack No. 2 meets the environmental specifications of the Graphic Computing System.

Temperature

Non-operating	−40 to + 65 degrees C. (−40 to + 149 degrees F)
Operating	+ 10 to + 40 degrees C. (+ 50 to + 104 degrees F)

Altitude

Non-operating	50,000 feet maximum
Operating	15,000 feet maximum

Humidity

Storage	95% non-condensing
Operating	80% non-condensing

Physical

Dimensions (including edge-board connector)

Length	11.84 centimeters (4.7 inches)
Width	6.65 centimeters (2.620 inches)
Depth	2.22 centimeters (0.875 inches)

Weight 227 grams (8 ounces)

Standard Accessories

4050 Series R08 Signal Processing ROM Pack No. 2 Instruction Manual.

Installing the ROM Pack

1. Be sure the Graphic Computing System power switch is OFF.

CAUTION

Inserting the ROM Pack while the power is ON may cause damage to the ROM Pack. The Graphic Computing System memory contents may also be lost.

2. Insert the ROM Pack into either one of the slots in the rear of the System as shown in Figure 1-2. Press down and gently rock the plastic case until the ROM Pack edge connector is firmly seated in the receptacle.

3. Turn the power ON and wait a few seconds for the System to warm up.

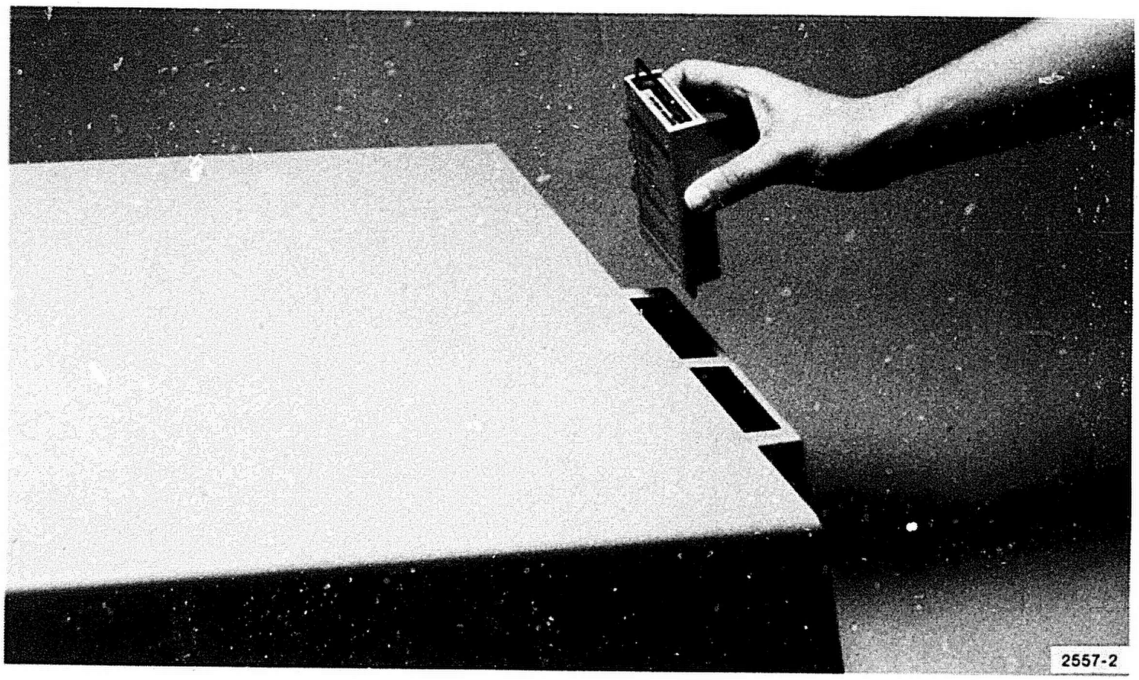


Figure 1-2. Installing the ROM Pack in the Graphic Computing System.

Section 2

COMMAND DESCRIPTIONS

INTRODUCTION

This section contains a detailed description of each of the eight commands provided by the Signal Processing ROM Pack No. 2. The explanations are intended for users who are familiar with the operation of TEKTRONIX 4050 Series Graphic Computing Systems. If you are not familiar with the Graphic Computing System, refer to the 4050 Series Graphic System Operator's Manual.

The explanations include basic information about using the commands and, where necessary, some background information about their algorithms. Section 3 includes a demonstration program that shows how to use the commands described in this section. A Command Summary is also provided in Appendix C for quick reference.

The ROM Pack routines are accessed with the CALL statement in 4050 BASIC. The routine name can be specified as a string constant in the CALL statement. Alternatively, the routine name can be assigned to a string variable and represented in the CALL statement by that string variable. Refer to the 4050 Series Graphic Computing System Reference Manual for more information on CALL statements.

DEFINITION OF TERMS

The command descriptions in this section make frequent reference to certain terms which may be unfamiliar. A review of the following definitions should help in understanding the information that follows.

real number — any rational or irrational number. (A rational number can be expressed as the quotient of two integers.)

imaginary number — a number having the form bj, where b is a real number and $j = \sqrt{-1}$.

complex number — a number having the form $a + bj$, where a is the real part and bj is the imaginary part. ($j = \sqrt{-1}$)

complex conjugation — the process of negating the imaginary part of a complex number to obtain the complex conjugate. (The complex conjugate of $a + bj$ is $a - bj$; the complex conjugate of $3 - 5j$ is $3 + 5j$.)

time domain — refers to a way of representing a signal so that the signal amplitude is expressed as a function of time.

frequency domain — refers to a way of representing a signal so that the signal amplitude is expressed as a function of frequency.

Fourier transform — a mathematical operation for converting a signal from the time domain to the frequency domain.

inverse Fourier transform — a mathematical operation for converting a signal from the frequency domain to the time domain.

rectangular form — an output format of the Fourier transform in which the spectral components are expressed as real and imaginary numbers.

polar form — an output format of the Fourier transform in which the spectral components are expressed in terms of magnitude and phase.

spectral components — refers to significant amplitudes existing at certain frequencies within the spectrum.

spectrum — a graph of signal amplitude (or energy) versus frequency.

THE FFT COMMAND

Syntax Form

[line number] CALL { "FFT" / string variable } , array

Descriptive Form

[line number] CALL routine name, source/destination array

Purpose

The FFT command performs a fast calculation of the discrete Fourier transform, and overwrites the data in the original (input) array. Only the positive half of the spectrum is computed.

Examples

```
CALL "FFT",W1
```

```
CALL A$,F
```

Explanation

The FFT (fast Fourier transform) is an algorithm for quickly computing the discrete Fourier transform. By means of the FFT command, the real and imaginary components of a complicated signal or waveform can be determined. (Also, the magnitude and phase components can be computed via the POLAR command.) The Fourier transform converts a waveform (time domain) or time-series data into a corresponding spectrum (frequency domain). For more information on the FFT, consult the references listed in Appendix B.

The format of the FFT command is illustrated by the following example:

```
CALL "FFT",F
```

The FFT command has a single argument (F in the above example). This argument must be an array with a power-of-two length between 16 and 1024, inclusive (i.e., 16, 32, 64, 128, 256, 512, or 1024). Before the FFT command is executed, the array argument must contain the real-valued signal data on which the Fourier transform is to be performed. After the command has executed, the original array data will be overwritten by the results of the FFT computation. The complex output data will then be arranged in the following order, where F refers to the array argument, and N refers to the array length:

F(1) = A real number representing the DC (direct current) value of the signal.
F(2) = A real number representing the value of the discrete Fourier transform at the Nyquist frequency (1/2 the sampling frequency).
F(3) = Real part of the first Fourier coefficient.
F(4) = Imaginary part of the first Fourier coefficient.
F(5) = Real part of the second Fourier coefficient.
F(6) = Imaginary part of the second Fourier coefficient.
.
.
.
F(N-1) = Real part of N/2-1 th Fourier coefficient.
F(N) = Imaginary part of N/2-1 th Fourier coefficient.

Notice that the first two array elements contain non-recurring data types, but that the remaining array elements alternate between real and imaginary Fourier coefficients. While this pattern of data may not be optimal for all applications, the real and imaginary components can be easily segregated with the UNLEAV command (discussed later in this section).

Theory of the Discrete Fourier Transform

The FFT command performs a fast calculation of the DFT (discrete Fourier transform), which can be expressed mathematically by the following summation:

$$X_d(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \quad \text{for } k = 0, 1, \ldots, N/2$$

In the above equation, N refers to the length of the array argument, and k is an index used in generating the various Fourier coefficients. Also, in accordance with usual mathematical notation, e (2.718281828...) is the base of natural logarithms, and j is the square root of -1. $X_d(k)$ is the kth Fourier coefficient, and x(n) refers to the n+1 th element of the real data input.

To illustrate how the above summation is used to produce the final output array, assume that the following command is being executed, where F is a 256-element array, initially containing waveform data:

```
CALL "FFT",F
```

The following steps result in the computation of array F, ultimately containing FFT data.

NOTE

In the following equations, the real input data x(0), x(1), . . . x(255) is initially stored in F(1), F(2), . . . F(256), respectively, but for reasons of clarity, the x array in the examples represents the input, and the F array represents the output.

1. Compute array element F(1) by setting k to 0, and computing the resulting summation:

$$F(1) = X_d(0) = \sum_{n=0}^{255} x(n)e^0 = \sum_{n=0}^{255} x(n) = x(0) + x(1) + \ldots + x(255)$$

As previously noted, F(1) is the first element of the output array (FFT data), and x(0) through x(255) are the elements of the input array (waveform data).[1]

2. Compute array element F(2) by setting k to N/2 and computing the resulting summation:

$$F(2) = X_d(n/2) = \sum_{n=0}^{255} x(n)e^{-j\pi n} = \sum_{n=0}^{255} x(n)\cos(\pi n)$$

Again, F(2) is the second element of the output array (FFT data), and the summation runs through all the elements of the input array, X.[1]

[1] **Notice that both F(1) and F(2) are real numbers and thus are zero corresponding imaginary parts.**

3. Compute array elements F(3) and F(4) by setting k to 1 and computing the resulting summation (known as the first Fourier coefficient):

$$X_d(1) = \sum_{n=0}^{255} x(n)e^{-j\pi n/128}$$

where F(3) is the real part of $X_d(1)$, and F(4) is the imaginary part of $X_d(1)$.

4. Continue computing array elements F(5) through F(256) in the same manner as outlined in Step 3. For example, the last two elements computed, F(255) and F(256), are found by computing the summation:

$$X_d(127) = \sum_{n=0}^{255} x(n)e^{-j\pi n254/256}$$

where F(255) is the real part of $X_d(127)$, and F(256) is the imaginary part of $X_d(127)$.

In summary, after the FFT command is executed, the following data will reside in array argument F:

$F(1) = X_d(0) = \sum_{n=0}^{N-1} x(n)$, a real number representing the DC (direct current) value of the signal.

$F(2) = X_d(N/2) = \sum_{n=0}^{N-1} x(n)\cos(\pi n)$, a real number representing the value of the DFT at the Nyquist frequency.[2]

[2]The Nyquist frequency is one-half the sampling frequency.

F(3) = Real part of first Fourier coefficient

$X_d(1)$;

F(4) = Imaginary part of first Fourier coefficient

F(5) = Real part of second Fourier coefficient

$X_d(2)$;

F(6) = Imaginary part of second Fourier coefficient

.

.

.

F(N-1) = Real part of N/2-1th Fourier coefficient

$X_d(N/2-1)$.

F(N) = Imaginary part of N/2-1th Fourier coefficient

For more information on the discrete Fourier transform and the FFT of ROM Pack No. 2, refer to Appendix A.

The FFT Algorithm

The preceding discussion describes one method for computing the discrete Fourier transform. It is instructive because it describes the way in which the FFT data is formatted. In the case of the ROM Pack No. 2, however, there is a much faster method for computing the fast Fourier transform: the Sande-Tukey decimation-in-frequency algorithm. It uses a 256-element floating-point table containing one-quarter cycle of negative sine-wave data for generating the necessary complex exponentials. The sine-wave data has sufficient resolution to perform the FFT of any real data array of any power-of-two length not exceeding 1024. The algorithm is a floating-point FFT requiring approximately 65 seconds (in the case of the 4051) or 4 seconds (in the case of the 4052) to compute the 513 complex frequency samples of a real waveform of 1024 points.

For more information on the Sande-Tukey algorithm, refer to Section 7 in the Tektronix concept book entitled **The FFT: Fundamentals and Concepts**. Several other references in Appendix B provide additional in-depth information on various FFT algorithms.

THE IFT COMMAND

Syntax Form

[line number] CALL { "IFT" / string variable } , array

Descriptive Form

[line number] CALL routine name, source/destination array

Purpose

The IFT command performs a fast calculation of the inverse Fourier transform and overwrites the data in the original (input) array.

Examples

```
CALL "IFT",M2
```

```
CALL I$,I
```

Explanation

The IFT (inverse Fourier transform) is an algorithm for computing the inverse operation of the discrete Fourier transform. By means of the IFT command, the complex output data from an FFT command can be transformed back into the original real signal data. In other words, the IFT command transforms spectral data (frequency domain) into waveform data (time domain).

The format of the IFT command is illustrated by the following example:

```
CALL "IFT",I
```

Like the FFT command, the IFT command has a single argument (I in the above example). This argument must be an array with a power-of-two length between 16 and 1024, inclusive (i.e., 16, 32, 64, 128, 256, 512, or 1024). Before the IFT command is executed, the array argument must contain complex input data in the same form as output by the FFT command. When executed, the IFT command returns real signal data, overwriting the input array.

Theory of the Inverse Fourier Transform

The inverse Fourier transform can be expressed mathematically by the following summation:

$$x(n) = (1/N) \sum_{k=0}^{N-1} X_d(k) e^{j2\pi nk/N} \quad \text{for } n = 0, 1, \ldots, N-1$$

In the above equation, N refers to the length of the array argument, and n is an index used in generating the various elements of the output array data. Also, in accordance with usual notation, e (2.718281828...) is the base of natural logarithms, and j is the square root of -1. $X_d(k)$ is the kth complex Fourier coefficient, and x(n) is the **n+1**th element of the real data output.

To illustrate how the above summation is used to produce the final output array, assume that the following command is being executed, where I is a 256-element array, initially containing complex FFT data:

```
CALL "IFT",I
```

The following steps result in the computation of array I, ultimately containing real time-domain signal data.

1. Compute array element (1) by setting n to 0, and computing the resulting summation:

$$I(1) = x(0) = (1/256) \sum_{k=0}^{255} X_d(k) e^0 = (1/256)[X_d(0) + X_d(1) + \ldots + X_d(255)]$$

   As previously noted, I(1) is the first element of the output array (IFT data), and X(0) through X(255) are the 256 complex Fourier coefficients represented in the input array.

2. Compute array element I(2) by setting n to 1, and computing the resulting summation:

$$I(2) = x(1) = (1/256) \sum_{k=0}^{255} X_d(k) e^{j\pi k/128}$$

Again, I(2) is the second element of the output array (IFT data), and X(k) is the kth Fourier coefficient.

3. Continue computing array elements I(3) through I(256) in the same manner as outlined in Step 2. For example, I(256) is found as follows:

$$I(256) = x(255) = (1/256) \sum_{k=0}^{255} X_d(k) e^{j255\pi k/128}$$

To fully understand how the IFT command works, it is important to realize that the FFT command of this ROM Pack displays only the positive half of a frequency spectrum; any negative frequencies are not shown since, for real waveforms, they appear as a mirror image (for the real part of the spectrum) and an inverted mirror image (for the imaginary part of the spectrum). However, the IFT command still requires the entire spectrum (positive and negative frequency components) before it can generate the time-domain waveform. It is therefore necessary for the IFT algorithm of ROM Pack 2 to generate these negative frequency components from the input array argument. This is done by creating an additional set of array elements for the X(k)'s ranging from X(N/2+ 1) to X(N-1). Recall that N is the input array length and X(k) is the kth Fourier coefficient defined by the following inverse DFT relation:

$$x(n) = (1/N) \sum_{k=0}^{N-1} X_d(k) e^{j2\pi nk/N} \quad \text{for } n = 0, 1, \ldots, N-1$$

In the calculation of the IFT, X(N/2+ 1) through X(N-1) are defined by the complex conjugation relationship:

$$X(N-k) = X^*(k)$$

where * denotes the complex conjugate of X(k). This relationship is valid as long as each of the output waveform array values, x(n), are real-valued.

The IFT Algorithm

The preceding discussion describes one method for computing the inverse Fourier transform. It is instructive because it describes the way in which the IFT data is formatted. In the case of the ROM Pack No. 2, however, the IFT is not found by a discrete summation process. Rather, the IFT command uses the same algorithm as the FFT command, except that some of the initialization parameters are different. For more information on the Sande-Tukey FFT algorithm, consult the references listed in the bibliography.

THE POLAR COMMAND

Syntax

[line number] CALL { "POLAR" / string variable } , array, array, array [, { constant / variable / expression }]

Descriptive Form

[line number] CALL routine name, source array (interleaved data), destination array (magnitude), destination array (phase) [,delay parameter]

Purpose

Normally, the FFT command returns its results in rectangular form (with the array components being real and imaginary numbers). Quite often though, the results can be interpreted more easily if they are in polar form (with the array components being magnitude and phase numbers). The POLAR command performs this conversion. The POLAR command converts an array of FFT data from rectangular form (containing interleaved real and imaginary components) to polar form. This polar form consists of two arrays: one containing magnitude components and one containing phase components.

Examples

```
CALL "POLAR",X,M,P,D1

CALL "POLAR",A,B,C

CALL P$,X,Y,Z
```

Explanation

The POLAR command has three required arguments and a fourth optional argument. An example of the command is:

```
CALL "POLAR",X,M,P,D1
```

The first argument (X in the above example) is the input array and must be of length N, where N is an even integer greater than or equal to 2. The array must contain complex data in the same order as produced by the FFT command:

X(1) = Real number representing the DC value of the signal.
X(2) = Real number representing the value of the DFT at the Nyquist frequency.
X(3) = Real part of first complex coefficient.
X(4) = Imaginary part of first complex coefficient.
X(5) = Real part of second complex coefficient.
X(6) = Imaginary part of second complex coefficient.
.
.
.
X(N-1) = Real part of N/2-1 th complex coefficient.
X(N) = Imaginary part of N/2-1 th complex coefficient.

The second and third arguments (arrays M and P in the above example) are the output arrays and contain the magnitude and phase calculations, respectively. They must each be of length N/2+ 1, where N is the length of the first argument (array X). The magnitude array is computed by the Pythagorean Theorem, and therefore contains real data of the form:

M(1) = Magnitude of first real number = $|X(1)|$
M(2) = Magnitude of first complex number = $\sqrt{X(3)^2 + X(4)^2}$
M(3) = Magnitude of second complex number = $\sqrt{X(5)^2 + X(6)^2}$
.
.
.
M(N/2) = Magnitude of N/2-1 th complex number = $\sqrt{X(N-1)^2 + X(N)^2}$
M(N/2+ 1) = Magnitude of second real number = $|X(2)|$

The phase is calculated using the arctangent function according to the following method, where r represents the real part and i the imaginary part of a complex number:

r	i	phase
<0	$=0$	$=-\pi$
<0	<0	$\arctan(i/r)-\pi$
$=0$	<0	$=-\pi/2$
>0	<0	$\arctan(i/r)$
≥ 0	$=0$	$=0$
>0	>0	$\arctan(i/r)$
$=0$	>0	$=\pi/2$
<0	>0	$\arctan(i/r)+\pi$

In the above table, phase is calculated modulo 2π; i.e., all the resultant phase values are within the range of $-\pi$ to $+\pi$. By specifying a delay value in the optional argument (D1 in the preceding example), the phase can be "unwrapped" modulo 2π, giving continuous phase. This unwrapping is accomplished by adding or subtracting the quantity 2π at the points of discontinuity in the modulo 2π phase.

If the optional argument (D1) is not specified, then modulo 2π phase is used.[3] If the optional argument is specified, continuous phase is produced. A value of 0 for D1 means that continuous phase will be used, and the slope will be the same as for the case where modulo 2π phase is used. If the delay parameter (D1) is not equal to zero, then the linear component of phase whose slope is 2πD1 is subtracted from the phase. (For more information on the theory behind continuous and modulo 2π phase, refer to the Tektronix concepts book entitled **The FFT: Fundamentals and Concepts**, beginning at page 4-9.)

When the delay argument is specified, it must be a constant, a floating-point variable, or an expression evaluating to a constant. It cannot be an array element.

Phase can be computed in units of radians, grads, or degrees. Default units are radians, but calculations in grads or degrees can be specified by a 4050 BASIC SET command. (See the Graphic Computing System Operator's Manual for details.)

The POLAR routine is specifically designed to handle the output from an FFT command where the data is formatted as previously described. However, the POLAR command can be used as a general rectangular-to-polar conversion routine provided the input and output format restrictions are satisfied.

---

[3]**Phase should be interpreted only at those points where there exist valid frequency components. Because roundoff error in the FFT algorithm yields real and imaginary components close to but not equal to 0 at points where there are no frequency components, their quotient (i/r) can yield what looks like significant results — but in reality, they should be ignored.**

THE CORR COMMAND

Syntax Form

[line number] CALL { "CORR" / string variable } , array, array, array

Descriptive Form

[line number] CALL routine name, source array, source array, /destination array

Purpose

The CORR command performs a fast correlation on two source arrays and places the results in the destination array.

Examples

```
CALL "CORR",X,Y,Z
```

```
CALL R$,A,B,C
```

Explanation

The CORR command performs a fast correlation operation on two input arrays, placing the result in a third array. If the two input arrays are the same, or contain the same data, the operation is called "auto-correlation." If the two input arrays are different, the operation is called "cross-correlation." Autocorrelation is a useful method of detecting the presence of periodic signals buried in noise. This technique is used in biomedical studies, astronomy, and tone-control systems, to name just a few applications. On the other hand, cross-correlation is a useful tool for detecting whether a known signal is present in a noisy environment. A common application for cross-correlation is the detection and ranging of radar, sonar, and other transmitted signals.

The format of the CORR command is illustrated by the following example:

```
CALL "CORR",X,Y,Z
```

The first two arguments (X and Y in the above example) are the input arrays to be correlated. These input arrays must be of equal length N, where N is a power-of-two integer between 8 and 512, inclusive (i.e., 8, 16, 32, 64, 128, 256, or 512). The third array (Z in the preceding example) must be of length 2N. When the CORR command has finished executing, the third array (Z) contains the correlation results.

NOTE

The first two arrays (X and Y) will be overwritten with intermediate results. *Because the input arrays are overwritten, you may want to save the original array contents elsewhere before executing CORR.*

The Theory of Correlation

The CORR command performs a non-cyclic cross-correlation (or auto-correlation) which can be described by the summation:

$$Z_d(n) = (1/N) \sum_{k=0}^{N-n-1} X(k)Y(n+k) \qquad \text{for } n = -N+1,\ldots,-1,0,1,\ldots,N-1$$

In this equation, N refers to the length of the input arrays, and n is an index used in generating the various elements of the output array. Also, X(k) is the **k+1** th element of the first input array, and Y(n+ k) is the **n+k+1** th element of the second input array; k is merely an index that defines the range of the summation. Finally, $Z_d(n)$ is an element of the output array Z (containing the results of the correlation) which is formatted as follows:

$Z(2) = Z_d(-N+1)$
$Z(3) = Z_d(-N+2)$
.
.
.
$Z(2N-1) = Z_d(N-2)$
$Z(2N) = Z_d(N-1)$

The first element Z(1) is used for storage of intermediate results only. On output, it is always approximately zero, but should not be considered part of the CORR results.

The above summation is computed in a fashion similar to that previously explained under the description of FFT and IFT. That is, we compute the summation of X(k)Y(n+ k) as k ranges from 0 to N-n-1; this is done for each value of n, beginning with n= —N+ 1 and ending with n= N-1. Each value of n then defines a corresponding element of the final output array Z. For some values of n and k, the index n+ k of array Y may be equal to a number less than 0. When this occurs, 0 is used for Y(n+ k).

The CORR Algorithm

The preceding discussion describes one method for computing correlation. It is instructive because of the way in which correlation data is formatted. In the case of the ROM Pack No. 2, however, there is a faster computation method of which takes advantage of the already present FFT and IFT algorithms. This method is based on the fact that the correlation of two signals (time domain) is equivalent to the complex conjugate multiplication of their Fourier transforms (frequency domain). Stated in mathematical terms:

$$CC_{AB} = A^*(f) \cdot B(f)$$

where A*(f) is the complex conjugate of the Fourier transform of a(x), and B(f) is the Fourier transform of b(x) (with a(x) and b(x) being the two time-domain signals); CC_{AB} is the Fourier transform of cc_{ab} (the correlation of a(x) and b(x)).

Before the CORR command actually performs the correlation on two arrays, it appends N zeroes to each of these input arrays (previously called X and Y). This prevents "wrap around" errors in correlations implemented via the discrete Fourier transform. (If this were not done, the assumptions of periodicity that the FFT makes would bias the results.) The input arrays X and Y are then transformed to the frequency domain via the FFT, and a complex-conjugate multiplication is performed on the resulting arrays. (By "complex-conjugate multiplication," it is meant that the imaginary part of one of the Fourier transform pairs is negated before performing the complex multiplication.) Finally, an inverse Fourier transform (IFT) is performed on the product, resulting in a correlation of the original arrays. This procedure is equivalent to the direct evaluation of the non-cyclic correlation summation previously discussed, but the whole process is performed more quickly due to the smaller number of calculations required.

THE CONVL COMMAND

Syntax Form

[line number] CALL { "CONVL" / string variable } ,array, array, array

Descriptive Form

[line number] CALL routine name, source array, source array, /destination array

Purpose

The CONVL command performs a fast convolution on two source arrays and places the results in the destination array.

Examples

```
CALL "CONVL",X,Y,Z
```

```
CALL C$,W1,W2,W3
```

Explanation

The CONVL command performs a fast convolution operation on two input arrays, placing the result in a third array. The CONVL command performs an operation similar to the CORR command. Like correlation, the convolution operation can be thought of as successively shifting, multiplying, and integrating the two arrays (waveforms) to be convolved. However, in the case of convolution, one of the waveforms is reversed in time before performing the correlation process. (This is evident from examining the summation that mathematically describes convolution.) An important application for convolution is determining or predicting the output of a linear, time-invariant system.

The format of the CONVL command is illustrated by the following example:

```
CALL "CONVL",X,Y,Z
```

The first two arguments (X and Y in the above example) are the input arrays to be convolved. These input arrays must be of equal length N, where N is a power-of-two integer between 8 and 512, inclusive (i.e., 8, 16, 32, 64, 128, 256, or 512). The third array (Z in the preceding example) must be of length 2N. When the CONVL command has finished executing, the third array (Z) will contain the results of the convolution.

NOTE

The first two arrays (X and Y) will be overwritten with intermediate results. *Because the input arrays are overwritten, you may want to save the original array contents elsewhere before executing CONVL.*

The Theory of Convolution

The CONVL command performs a non-cyclic convolution which can be described by the summation:

$$Z_d(n) = \sum_{k=0}^{N-1} X(k)Y(n-k) \qquad \text{for } n = 0,1,2,\ldots,2N-2.$$

In the above equation, N refers to the length of the input arrays, and n is an index used in generating the various elements of the output array. Also, X(k) is the **k+1** th element of the first input array, Y(n-k) is the **n-k+1** th element of the second input array; k is merely an index that defines the range of the summation. Finally, $Z_d(n)$ is an element of the output array Z (containing the results of the convolution) which is formatted as follows:

$Z(1) = Z_d(0)$
$Z(2) = Z_d(1)$
.
.
.
$Z(2N-1) = Z_d(2N-2)$

The last element, Z(2N), is used for storage of intermediate results only. On output, it will always approximate zero but should not be considered part of the CONVL results.

The above summation is computed in a fashion similar to that previously explained under the description of FFT and IFT. That is, we compute the summation of X(k)Y(n-k) as k ranges from 0 to N-1; this is done for each value of n, beginning with 0 and ending with 2N-2. Each value of n then defines a corresponding element of the final output array Z. For some values of n and k, the index, n-k of the array Y may be equal to a number less than 0. When this occurs, 0 is used for Y(n-k).

The CONVL Algorithm

The preceding discussion describes one method for computing convolutions. It is instructive because it describes the way in which convolution data is formatted. In the case of the ROM Pack No. 2, however, there is a faster method of computation which takes advantage of the already present FFT and IFT algorithms. This method is based on the fact that the convolution of two signals (time domain) is equivalent to the multiplication of their Fourier transforms (frequency domain). Stated in more mathematical terms:

$$CC_{AB} = A(f) \cdot B(f)$$

where A(f) and B(f) are the Fourier transforms of a(x) and b(x) (the two time-domain signals), and CC_{AB} is the Fourier transform of cc_{ab} (the convolution of a(x) and b(x)).

Before the CONVL command actually performs the convolution on two arrays, it appends N zeroes to each of these input arrays (previously called X and Y). This prevents "wrap around" errors in convolutions implemented via the discrete Fourier transform. (If this were not done, the assumptions of periodicity that the FFT makes would bias the results.) The input arrays X and Y are then transformed to the frequency domain via the FFT, and a complex multiplication is performed on the resulting arrays. Finally, an inverse Fourier transform (IFT) is performed on the product, resulting in a convolution of the original arrays. This procedure is equivalent to the direct evaluation of the non-cyclic convolution summation previously discussed, but the whole process is performed more quickly due to the smaller number of calculations required.

THE TAPER COMMAND

Syntax Form

[line number] CALL { "TAPER" / string variable } , array [, { constant / variable / expression }]

Descriptive Form

[line number] CALL routine name, source/destination array [, tapering parameter]

Purpose

The TAPER command multiplies the source array by a modified Hanning (cosine) windowing function, then replaces the source data with the windowed data.

Examples

```
CALL "TAPER",X,T1

CALL T$,A,P1
```

Explanation

A commonly encountered source of error in using FFT algorithms is referred to as "leakage." Leakage occurs when acquiring a time window of a periodic waveform such that a non-integer number of cycles are transformed. Due to the non-integer number of cycles in the time-window, some invalid assumptions are made that lead to an attenuation and broadening of the spectral lines when the FFT is performed. In the displayed spectrum, it appears that energy has "leaked" from one spectral line to an adjacent one — hence the term "leakage." To reduce leakage, it is necessary to multiply the time-domain waveform by a windowing function, such as a cosine (Hanning) window, before performing the FFT operation. This is the purpose of the TAPER command. Time truncation errors are also reduced by windowing. These errors occur when the implicit rectangular window creates a discontinuity at the window edges. By gently tapering with TAPER, the discontinuities are lessened. (For more information on windowing, refer to the Tektronix application note: **Windowing to Control FFT Leakage**. Additional information on windowing and FFT leakage is

found in the Tektronix concept book entitled **The FFT: Fundamentals and Concepts**, beginning on page 5-13.)

The TAPER command performs a windowing routine with a variable taper. The format of the TAPER command is demonstrated by the following example:

```
CALL "TAPER",X,T1
```

The first argument (X in the above example) must be an array of length greater than four. It initially contains the real waveform data to be windowed. During command execution, the data is multiplied by a cosine (Hanning) window with the specified amount of tapering. This product then overwrites the original data stored in the array.

The optional second argument (T1 in the above example) specifies the amount of tapering. If this optional argument is absent, a default value of 0.1 (10%) will be used, meaning that the first 10% and last 10% of the rectangular acquisition window will be tapered with the cosine window (see Figure 2-1). If the optional argument is present, it must be a constant, floating-point variable, or expression between 0 and 0.5, inclusive, representing the amount of tapering. (If T1 = 0.5, then 50% tapering is accomplished, resulting in the Hanning window which applies full tapering as shown in Figure 2-1.)

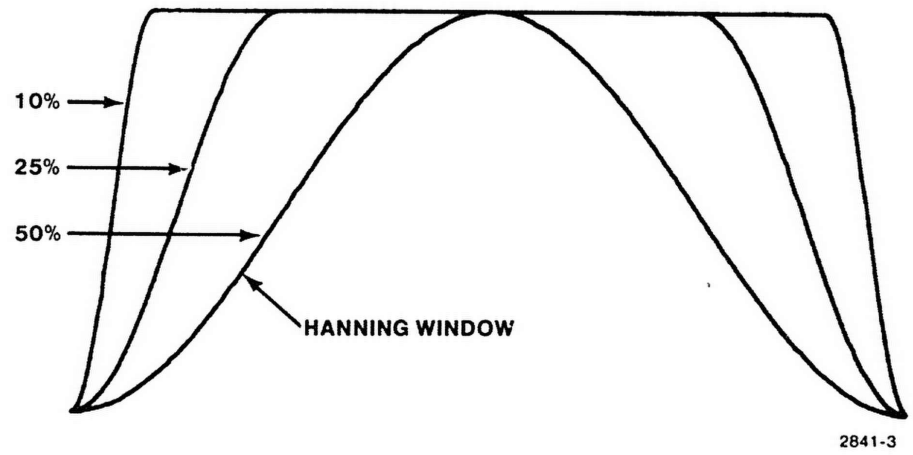


Figure 2-1. Illustration of 10%, 25%, and 50% Cosine Window Tapering.

A fatal error will be issued if the optional parameter evaluates to a number less than 0 or greater than 0.5. The total number of points affected by the tapering is determined by multiplying the percentage of tapering by the number of elements in the array and taking the integer part of the result. If this product is less than 2, no tapering will be performed and no error will be issued. With this restriction, the smallest array size possible using the default taper value of 0.1 is 20 elements. Likewise, 50% tapering can be performed on an array as small as five elements.

THE INLEAV COMMAND

Syntax Form

[line number] CALL { "INLEAV" / string variable } , array, array, array

Descriptive Form

[line number] CALL routine name, source array (real), source array (imaginary), destination array (interleaved data)

Purpose

The INLEAV command interleaves the real and imaginary data from two source arrays into one destination array whose format is acceptable to the IFT command.

Examples

```
CALL "INLEAV",R,I,X
```

```
CALL N$,D,E,F
```

Explanation

The INLEAV command is essentially the inverse of the UNLEAV command. It interleaves the data from two input arrays (containing real and imaginary data) into an output array whose format is acceptable as input to the IFT.

The format of the INLEAV command is illustrated by the following example:

```
CALL "INLEAV",R,I,X
```

The first two arguments (R and I in the above example) are the input arrays containing the real and imaginary data to be interleaved, respectively. These arrays must both be of length N/2+ 1, N being the length of the output array (X). N must be an even integer greater than or equal to 2.

The result of the INLEAV operation is placed in the third argument (array X in the preceding example), which is formatted as follows:

X(1) = R(1)
X(2) = R(N/2+ 1)
X(3) = R(2)
X(4) = I(2)
X(5) = R(3)
X(6) = I(3)
..
..
..
X(N-1) = R(N/2)
X(N) = I(N/2)

The INLEAV command is specifically designed to handle the complex data in two arrays, where the first array (R) contains the real parts, and the second input array (I) contains the imaginary parts. It is further stipulated that:

R(1) = DC term
R(N/2+ 1) = Nyquist term
I(1) = I(N/2+ 1) = 0

When the above conditions are met, INLEAV results in an output array (X) containing real and imaginary data in a format acceptable as input to the IFT command.

THE UNLEAV COMMAND

Syntax Form

[line number] CALL { "UNLEAV" / string variable } , array, array, array

Description

[line number] CALL routine name, source array (interleaved data), destination array (real), destination array (imaginary)

Purpose

The UNLEAV command is used to sort the interleaved data from the FFT command into two arrays, one with just the real components and one with just the imaginary components.

Examples

```
CALL "UNLEAV",X,R,I

CALL U$,A,B,C
```

Explanation

The results of the FFT command are interleaved such that the real and imaginary components alternate with each succeeding element number. However, this may not be the optimum format, since it is often desirable to have the real and imaginary components in separate arrays. The UNLEAV command can be used to achieve this separation.

The format of the UNLEAV command is illustrated by the following example:

```
CALL "UNLEAV",X,R,I
```

The first argument (X in the above example) is the input array containing the data to be unleaved; this array must be of length N, where N is an even integer greater than or equal to 2. The second and third arguments (R and I) are the output arrays and must be of length N/2+ 1. After the UNLEAV command has executed, the output arrays will contain the following data:

R(1) = X(1)	I(1) = 0
R(2) = X(3)	I(2) = X(4)
R(3) = X(5)	I(3) = X(6)
.	.
.	.
.	.
R(N/2) = X(N-1)	I(N/2) = X(N)
R(N/2+ 1) = X(2)	I(N/2+ 1) = 0

The UNLEAV command is specifically designed to handle the complex output of the FFT where X(1) and X(2) are real DC and Nyquist components. The result of UNLEAV places the real components in the first output array, R, and the corresponding imaginary components in the second output array, I. The DC term will appear at R(1), I(1) and the Nyquist term at R(N/2+ 1), I(N/2+ 1). Notice that, because the DC and Nyquist terms are real numbers, the imaginary components of the two terms are zero.

Section 3

DEMONSTRATION-VERIFICATION PROGRAM

INTRODUCTION

This section includes a sample program that exercises the basic functions of the Signal Processing ROM Pack No. 2. This program, written in 4050 Series BASIC, also illustrates the use and syntax of each of the eight commands in the ROM pack. The program begins by generating an initial test signal and then sequentially performing the various signal processing operations. (Hard copies of actual program output following each operation are included.) If you encounter errors while using the ROM pack, this program can be run to verify that the ROM pack is functioning properly.

RUNNING THE SAMPLE PROGRAM

The sample program uses four commands that are found only in Signal Processing ROM Pack No. 1, namely the INTegrate, MAXimum, MINimum, and DISPlay routines. If you do not have access to ROM Pack No. 1, the INTegrate, MAXimum and MINimum routines can be simulated with a FOR loop, and the DISPlay routine can be simulated with a DRAW command in a FOR loop. Assuming that both ROM packs are available, install each ROM pack in one of the two receptacles as discussed in Section 1 of this manual. Then enter the program as shown. (Refer to the 4050 Series Graphic Computing System Reference Manual if you need more information.)

After the program is entered into the Graphic Computing System, type RUN. If you encounter any errors, refer to Appendix D. Find the error number in the appendix and check the probable cause.

```
LIST
10 DIM X(256),Y(256),M(129),P(129),C(512)
20 X=2*PI/128
30 Y=16*X
40 CALL "INT",X,X
50 CALL "INT",Y,Y
60 X=SIN(X)
70 Y=COS(Y)
80 Y=Y/2
90 X=X+Y
95 Y=X
100 L$="ORIGINAL SIGNAL"
110 GOSUB 1000
200 L$="COMPLEX FFT -- INTERLEAVED REALS AND IMAGINARIES"
210 CALL "FFT",Y
220 GOSUB 1000
300 L$="FFT -- MAGNITUDE,.....AND PHASE"
310 CALL "POLAR",Y,M,P
320 GOSUB 2000
330 GOSUB 5000
400 L$="FFT -- REALS,.....AND IMAGINARIES"
410 CALL "UNLEAV",Y,M,P
420 GOSUB 2000
500 L$="IFT OF FFT OF ORIGINAL"
510 CALL "INLEAV",M,P,Y
520 CALL "IFT",Y
530 GOSUB 1000
600 L$="AUTOCORRELATION OF ORIGINAL SIGNAL"
610 CALL "CORR",X,X,C
620 GOSUB 3000
700 L$="CONVOLUTION OF ORIGINAL WITH IFT OF FFT OF ORIGINAL"
710 CALL "CONVL",X,Y,C
720 GOSUB 3000
730 GOSUB 5000
800 L$="TAPERED WINDOWS -- 10%, 25%, AND 50% TAPERING"
810 X=1
820 CALL "TAPER",X
830 GOSUB 4000
840 X=1
850 CALL "TAPER",X,0.25
860 GOSUB 4060
870 X=1
880 CALL "TAPER",X,0.5
890 GOSUB 4060
900 PRINT "_END OF PROGRAM"
910 END
1000 CALL "MAX",Y,M2,I1
1010 CALL "MIN",Y,M1,I1
1020 WINDOW 0,256,M1,M2
1030 VIEWPORT 10,110,10,90
1040 PAGE
1050 PRINT L$
1060 CALL "DISP",Y
1070 RETURN
2000 CALL "MAX",M,M2,I1
2010 CALL "MIN",M,M1,I1
2020 WINDOW 0,129,M1,M2
2030 VIEWPORT 5,55,30,70
2040 PAGE
2050 PRINT L$
2060 CALL "DISP",M
```

(cont)

```
2070 CALL "MAX",P,M2,I1
2080 CALL "MIN",P,M1,I1
2090 WINDOW 0,129,M1,M2
2100 VIEWPORT 65,115,30,70
2110 CALL "DISP",P
2120 RETURN
3000 CALL "MAX",C,M2,I1
3010 CALL "MIN",C,M1,I1
3020 WINDOW 0,512,M1,M2
3030 VIEWPORT 10,110,10,90
3040 PAGE
3050 PRINT L$
3060 CALL "DISP",C
3070 RETURN
4000 CALL "MAX",X,M2,I1
4010 CALL "MIN",X,M1,I1
4020 WINDOW 0,256,M1,M2
4030 VIEWPORT 10,110,20,70
4040 PAGE
4050 PRINT L$
4060 CALL "DISP",X
4070 RETURN
5000 FOR I=0 TO 2000
5010 NEXT I
5020 RETURN
```

UNDERSTANDING THE PROGRAM

The first 10 lines of the sample program are for initialization of parameters and creation of the test signal. Line 10 dimensions five arrays: X, Y, M, P, and C. Arrays X and Y initially contain the original signal and array Y will eventually store the results of the FFT and IFT operations. Arrays M and P will be used to store the results of the INLEAV operation as well as the POLAR operation. Array C will hold the results of the CORRELATE and CONVOLVE operations. Lines 20 through 70 create a 2-cycle sine wave and a 32-cycle cosine wave, respectively. This is done by integrating a constant (to get a ramp function) and then taking the sine and cosine functions of the resulting ramp. At line 80, the amplitude of the cosine wave is halved to give the signal a more interesting spectrum. Line 90 then creates a signal that is the algebraic sum of the sine and cosine waves. (Line 95 merely preserves an extra copy of this waveform for later use.)

At line 100, a label (L$) for the original signal is created and this signal is then graphed on the display screen by the subroutine at line 1000 (Figure 3-1).

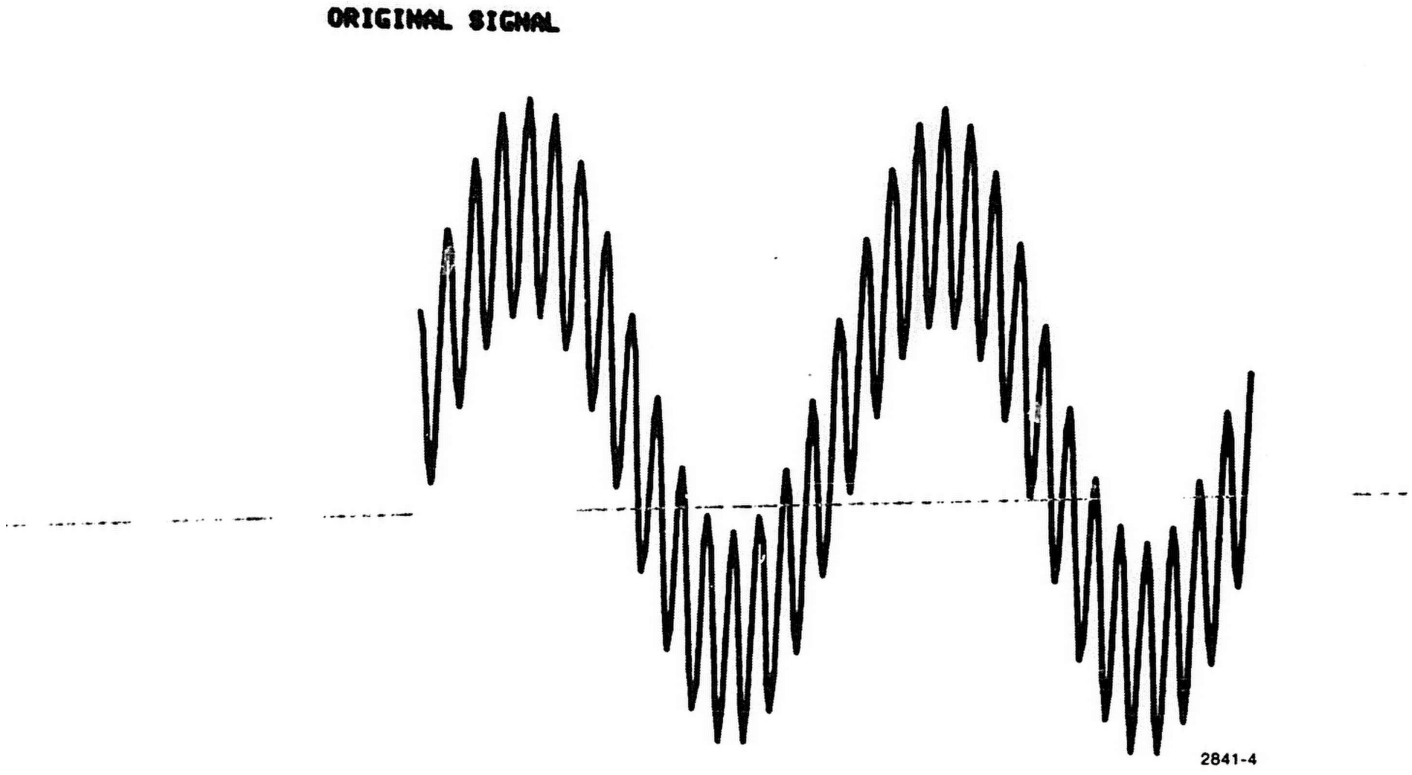


Figure 3-1. Original Signal.

Beginning at line 200, the complex FFT of the original signal is computed and a labeled graph (Figure 3-2) of the resulting array is displayed, again by the subroutine at 1000.

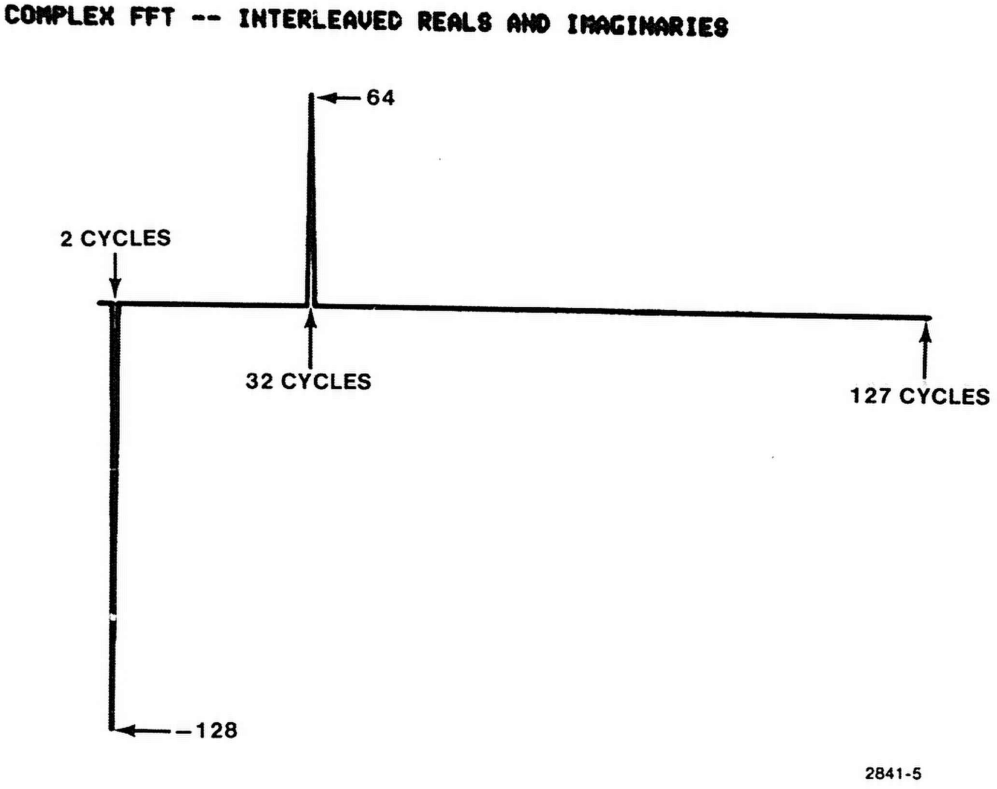


Figure 3-2. Complex FFT — Interleaved Reals and Imaginaries.

Lines 300 through 330 demonstrate the POLAR routine which computes the magnitude and phase components from the just computed FFT. The magnitude components are placed in array M and the phase components in array P. The subroutine at line 2000 graphs M and P in separate viewports of the display screen (Figure 3-3). M and P are graphed such that their minimum and maximum values span the entire viewport. The subroutine at 5000 is a FOR . . . NEXT loop which simply creates a small delay before proceeding to line 400. (This delay extends the viewing time for the graphic display just produced.)

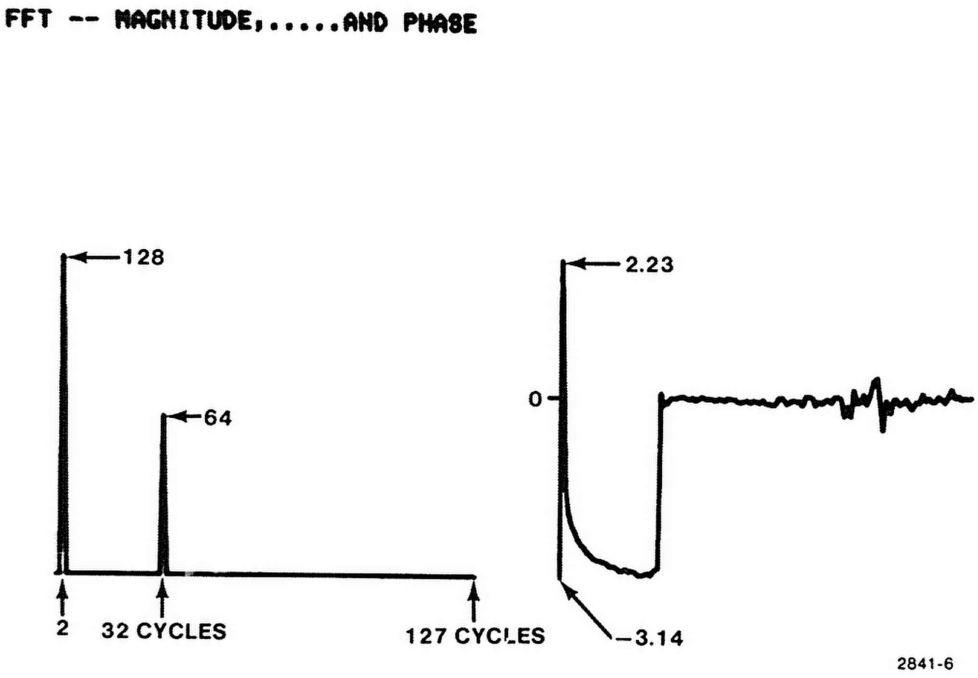


Figure 3-3. FFT — Magnitude and Phase.

NOTE

To increase the viewing time, change line 5000 of the program by increasing the number of iterations.

Lines 400 through 420 demonstrate the UNLEAV routine which segregates the real and imaginary components from the FFT data computed at line 210. The real components are placed in array M and the imaginary components in array P. Again, the subroutine at 2000 is called to graph arrays M and P with maximum resolution in their respective viewports as shown in Figure 3-4.

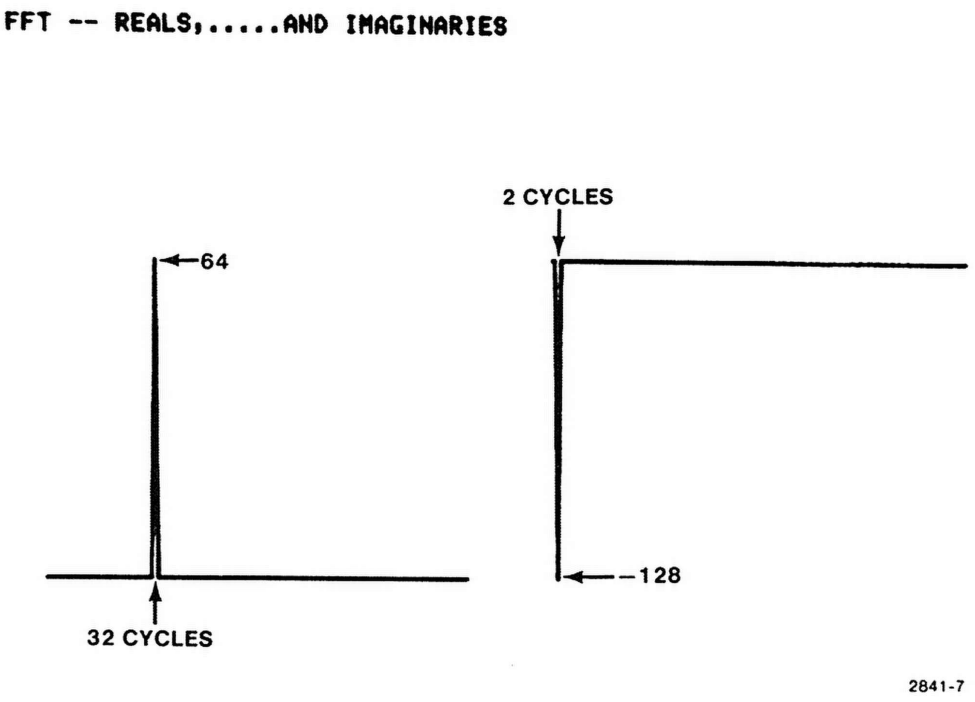


Figure 3-4. FFT — Reals and Imaginaries.

Beginning at line 500, the IFT of the FFT of the original signal is computed. The INLEAV command at line 510 is not actually required in this case since array Y already contains interleaved FFT data. It merely shows that the INLEAV command will return data in the same format as required by the IFT. Line 520 then generates the IFT of Y, and the results are graphed via the subroutine at 1000. Notice that this display (Figure 3-5) is virtually identical to the original signal (Figure 3-1) since the IFT and FFT are inverse operations.

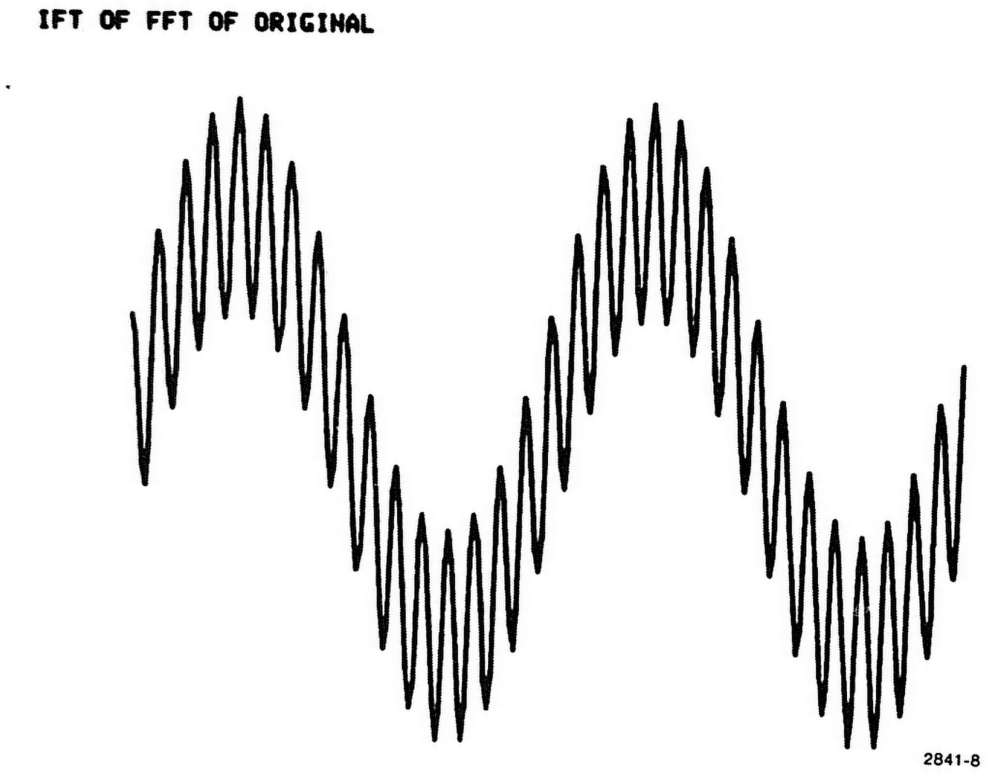


Figure 3-5. IFT of FFT of Original Signal.

Lines 600 through 620 demonstrate autocorrelation. This is actually accomplished in line 610 where array X (the original signal) is correlated with itself. The subroutine at 3000 generates a graph (Figure 3-6) of the results, complete with the label (L$) generated at line 600.

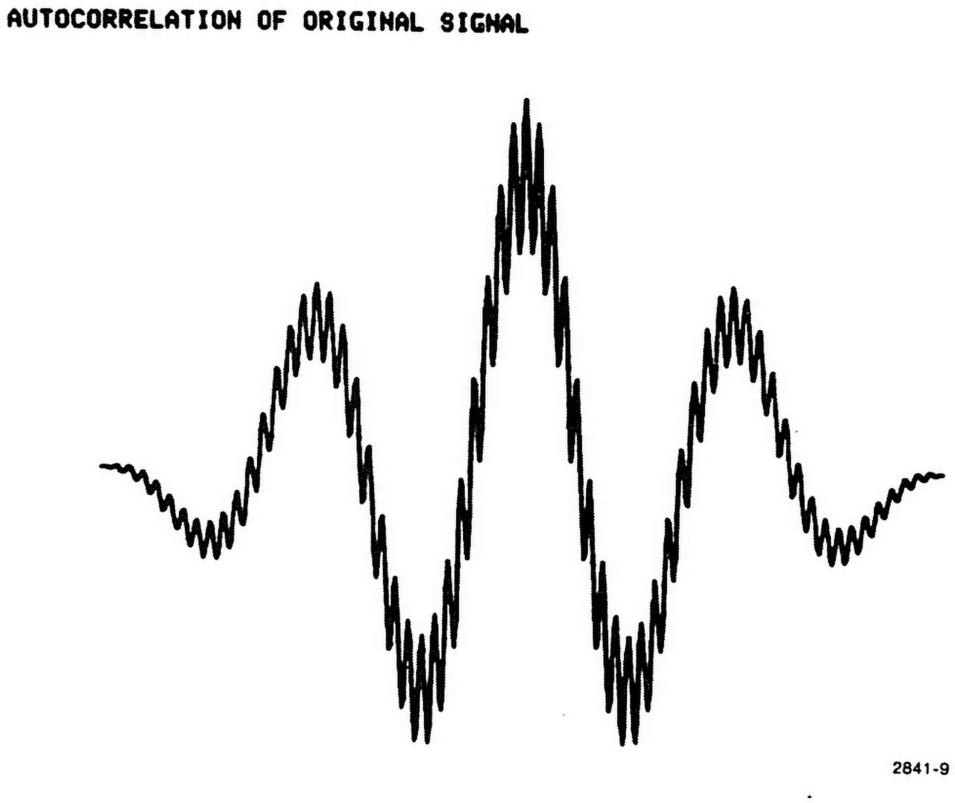


Figure 3-6. Autocorrelation of Original Signal.

Lines 700 through 730 demonstrate convolution by convolving array X (the original signal) with array Y (the IFT of the FFT of the original). Again, the subroutine at line 3000 is used to graph the results of the convolution (Figure 3-7).

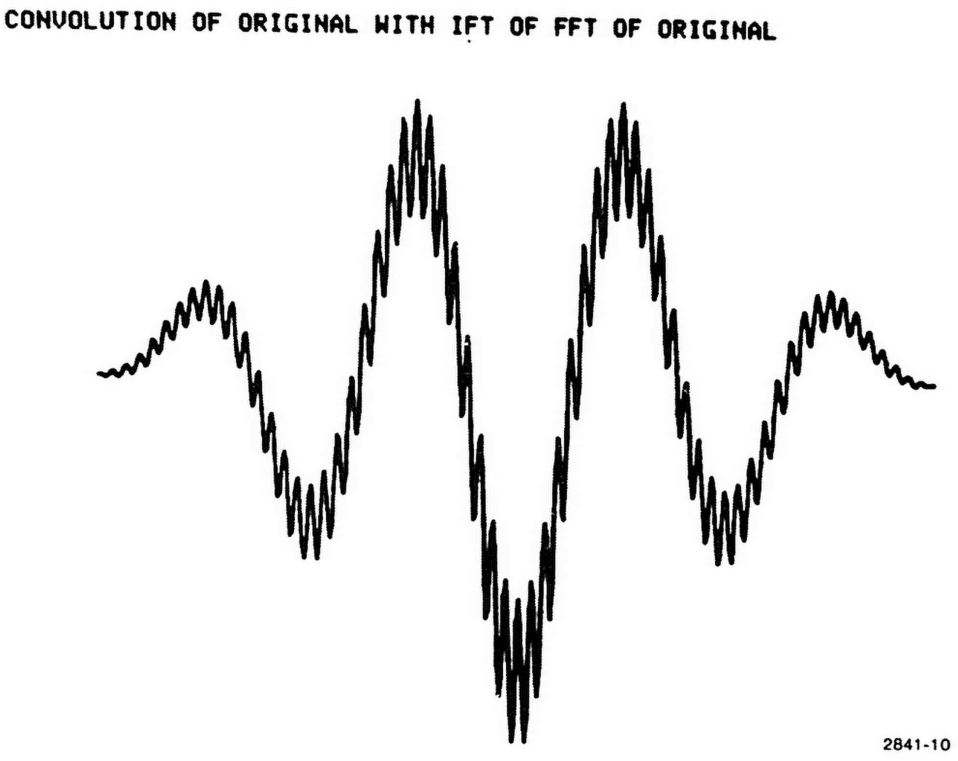
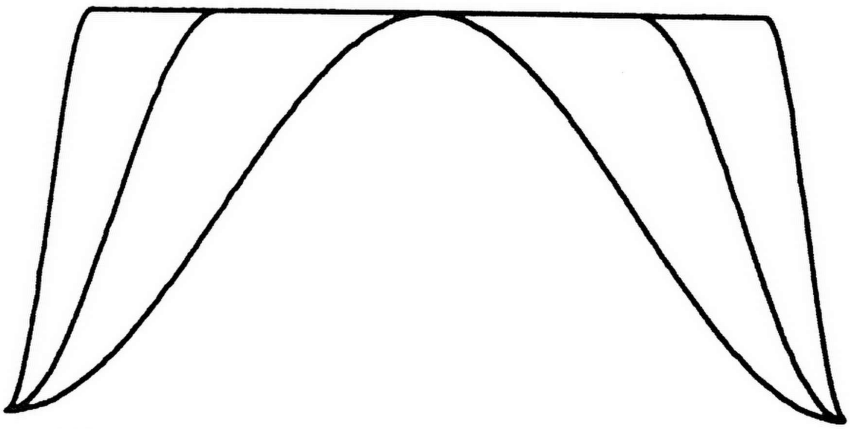


Figure 3-7. Convolution of Original with IFT of FFT of Original.

Line 800 begins a tapering demonstration, in which a unity-amplitude pulse in array X is successively tapered by 10%, 25%, and 50%. The subroutine at 4000 is used to display all three tapering results in the same viewport (Figure 3-8). After these results have been displayed, line 900 prints a message indicating the END of the program.

```
TAPERED WINDOWS -- 10%, 25%, AND 50% TAPERING
```



```
END OF PROGRAM
```

2841-11

Figure 3-8. Tapered Windows — 10%, 25%, and 50% Tapering.

NOTE

To aid in interpreting the data generated by the demonstration program, the amplitude and frequency parameters have been printed in Figures 3-2 through 3-4. These values were obtained via the MIN and MAX commands of ROM Pack No. 1. (The amplitude values were obtained directly; the frequency values were obtained by noting the **K***-value, for the* **K***th Fourier coefficient, corresponding to the array-element number where a spectral component was found.)*

Section 4

REPLACEABLE PARTS

PARTS ORDERING INFORMATION

Replacement parts are available from or through your local Tektronix, Inc. Field Office or representative.

Changes to Tektronix instruments are sometimes made to accommodate improved components as they become available, and to give you the benefit of the latest circuit improvements developed in our engineering department. It is therefore important, when ordering parts, to include the following information in your order: Part number, instrument type or number, serial number, and modification number if applicable.

If a part you have ordered has been replaced with a new or improved part, your local Tektronix, Inc. Field Office or representative will contact you concerning any change in part number.

Change information, if any, is located at the rear of this manual.

SPECIAL NOTES AND SYMBOLS

X000	Part first added at this serial number
00X	Part removed after this serial number

FIGURE AND INDEX NUMBERS

Items in this section are referenced by figure and index numbers to the illustrations.

INDENTATION SYSTEM

This mechanical parts list is indented to indicate item relationships. Following is an example of the indentation system used in the description column.

1 2 3 4 5 Name & Description

Assembly and/or Component
Attaching parts for Assembly and/or Component
\- - - * - - -
Detail Part of Assembly and/or Component
Attaching parts for Detail Part
\- - - * - - -
Parts of Detail Part
Attaching parts for Parts of Detail Part
\- - - * - - -

Attaching Parts always appear in the same indentation as the item it mounts, while the detail parts are indented to the right. Indented items are part of, and included with, the next higher indentation. The separation symbol - - - * - - - indicates the end of attaching parts.

Attaching parts must be purchased separately, unless otherwise specified.

ITEM NAME

In the Parts List, an Item Name is separated from the description by a colon (:). Because of space limitations, an Item Name may sometimes appear as incomplete. For further Item Name identification, the U.S. Federal Cataloging Handbook H6-1 can be utilized where possible.

ABBREVIATIONS

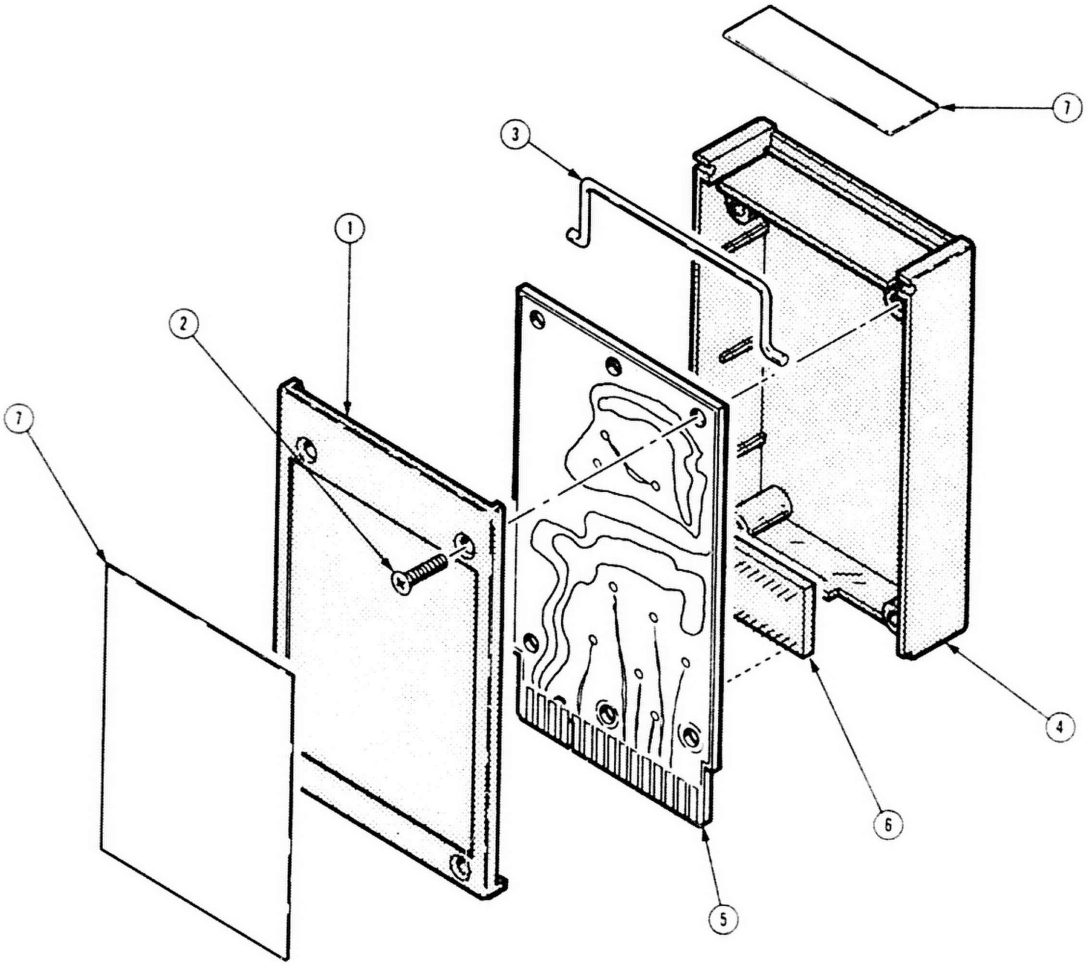
"	INCH
#	NUMBER SIZE
ACTR	ACTUATOR
ADPTR	ADAPTER
ALIGN	ALIGNMENT
AL	ALUMINUM
ASSEM	ASSEMBLED
ASSY	ASSEMBLY
ATTEN	ATTENUATOR
AWG	AMERICAN WIRE GAGE
BD	BOARD
BRKT	BRACKET
BRS	BRASS
BRZ	BRONZE
BSHG	BUSHING
CAB	CABINET
CAP	CAPACITOR
CER	CERAMIC
CHAS	CHASSIS
CKT	CIRCUIT
COMP	COMPOSITION
CONN	CONNECTOR
COV	COVER
CPLG	COUPLING
CRT	CATHODE RAY TUBE
DEG	DEGREE
DWR	DRAWER
ELCTRN	ELECTRON
ELEC	ELECTRICAL
ELCTLT	ELECTROLYTIC
ELEM	ELEMENT
EPL	ELECTRICAL PARTS LIST
EQPT	EQUIPMENT
EXT	EXTERNAL
FIL	FILLISTER HEAD
FLEX	FLEXIBLE
FLH	FLAT HEAD
FLTR	FILTER
FR	FRAME or FRONT
FSTNR	FASTENER
FT	FOOT
FXD	FIXED
GSKT	GASKET
HDL	HANDLE
HEX	HEXAGON
HEX HD	HEXAGONAL HEAD
HEX SOC	HEXAGONAL SOCKET
HLCPS	HELICAL COMPRESSION
HLEXT	HELICAL EXTENSION
HV	HIGH VOLTAGE
IC	INTEGRATED CIRCUIT
ID	INSIDE DIAMETER
IDENT	IDENTIFICATION
IMPLR	IMPELLER
IN	INCH
INCAND	INCANDESCENT
INSUL	INSULATOR
INTL	INTERNAL
LPHLDR	LAMPHOLDER
MACH	MACHINE
MECH	MECHANICAL
MTG	MOUNTING
NIP	NIPPLE
NON WIRE	NOT WIRE WOUND
OBD	ORDER BY DESCRIPTION
OD	OUTSIDE DIAMETER
OVH	OVAL HEAD
PH BRZ	PHOSPHOR BRONZE
PL	PLAIN or PLATE
PLSTC	PLASTIC
PN	PART NUMBER
PNH	PAN HEAD
PWR	POWER
RCPT	RECEPTACLE
RES	RESISTOR
RGD	RIGID
RLF	RELIEF
RTNR	RETAINER
SCH	SOCKET HEAD
SCOPE	OSCILLOSCOPE
SCR	SCREW
SE	SINGLE END
SECT	SECTION
SEMICOND	SEMICONDUCTOR
SHLD	SHIELD
SHLDR	SHOULDERED
SKT	SOCKET
SL	SLIDE
SLFLKG	SELF-LOCKING
SLVG	SLEEVING
SPR	SPRING
SQ	SQUARE
SST	STAINLESS STEEL
STL	STEEL
SW	SWITCH
T	TUBE
TERM	TERMINAL
THD	THREAD
THK	THICK
TNSN	TENSION
TPG	TAPPING
TRH	TRUSS HEAD
V	VOLTAGE
VAR	VARIABLE
W/	WITH
WSHR	WASHER
XFMR	TRANSFORMER
XSTR	TRANSISTOR

CROSS INDEX—MFR. CODE NUMBER TO MANUFACTURER

Mfr. Code	Manufacturer	Address	City, State, Zip
01121	ALLEN-BRADLEY COMPANY	1201 2ND STREET SOUTH	MILWAUKEE, WI 53204
01295	TEXAS INSTRUMENTS, INC. SEMICONDUCTOR GROUP	P.O. BOX 5012	DALLAS, TX 75222
04222	AVX CERAMICS, DIVISION OF AVX CORP.	P O BOX 867	MYRTLE BEACH, SC 29577
04713	MOTOROLA, INC., SEMICONDUCTOR PROD. DIV.	5005 E MCDOWELL RD,PO BOX 20923	PHOENIX, AZ 85036
09922	BURNDY CORPORATION	RICHARDS AVENUE	NORWALK, CT 06852
27014	NATIONAL SEMICONDUCTOR CORP.	2900 SEMICONDUCTOR DR.	SANTA CLARA, CA 95051
55680	NICHICON/AMERICA/CORP.	6435 N PROESEL AVENUE	CHICAGO, IL 60645
56289	SPRAGUE ELECTRIC CO.	87 MARSHALL ST.	NORTH ADAMS, MA 01247
80009	TEKTRONIX, INC.	P O BOX 500	BEAVERTON, OR 97077
83385	CENTRAL SCREW CO.	2530 CRESCENT DR.	BROADVIEW, IL 60153
91260	CONNOR SPRING AND MFG. CO.	1729 JUNCTION AVE.	SAN JOSE, CA 95112
96733	SAN FERNANDO ELECTRIC MFG CO	1501 FIRST ST	SAN FERNANDO, CA 91341

Ckt No.	Tektronix Part No.	Serial/Model No. Eff	Dscont	Name & Description	Mfr Code	Mfr Part Number
				4051R08		
A1	670-6290-00	B010100	B010351	CKT BOARD ASSY:16K ROM PACK	80009	670-6290-00
A1	670-7495-00	B010352		CKT BOARD ASSY:8K ROM PACK	80009	670-7495-00
C1	290-0106-00	B010100	B010351	CAP.,FXD,ELCTLT:10UF,+75-10%,15V	56289	30D106G015BA9
C2	283-0111-00	B010100	B010351	CAP.,FXD,CER DI:0.1UF,20%,50V	96733	R2632
C3	283-0010-00	B010100	B010351	CAP.,FXD,CER DI:0.05UF,+100-20%,50V	56289	1C10Z5U503Z050B
C115	283-0422-00	B010352		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
C205	283-0422-00	B010352		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
C305	283-0422-00	B010352		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
C315	283-0422-00	B010352		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
C401	290-0804-00	B010352		CAP.,FXD,ELCTLT:10UF,+50-10%,25V	55680	ULA1E100TEA
C415	283-0422-00	B010352		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
Q1	151-0324-00	B010100	B010351	TRANSISTOR:SILICON,PNP	04713	SJE915
Q1	----- -----			(REQUIRES FOAM ADHESIVE)		
R1	315-0102-00	B010100	B010351	RES.,FXD,CMPSN:1K OHM,5%,0.25W	01121	CB1[illegible]
R2	315-0391-00	B010100	B010351	RES.,FXD,CMPSN:390 OHM,5%,0.25W	01121	CB39.5
U1	160-0328-00	B010100	B010351	MICROCIRCUIT,DI:2048 X 8 ROM,CUSTOM MASK	80009	160-0328-00
U2	160-0329-00	B010100	B010351	MICROCIRCUIT,DI:2048 X 8 ROM,CUSTOM MASK	80009	160-0329-00
U3	160-0330-00	B010100	B010351	MICROCIRCUIT,DI:2048 X 8 ROM,CUSTOM MASK	80009	160-0330-00
U4	160-0331-00	B010100	B010351	MICROCIRCUIT,DI:2048 X 8 ROM,CUSTOM MASK	80009	160-0331-00
U5	156-0469-02	B010100	B010351	MICROCIRCUIT,DI:3/8 LINE DCDR	01295	SN74LS138NP3
U105	156-0916-02	B010352		MICROCIRCUIT,DI:8-2 INP 3-STATE BFR,BURN	27014	DM81LS97A
U115	160-1453-00	B010352		MICROCIRCUIT,DI:4096 X 8 EPROM,PRGM	80009	160-1453-00
U205	156-0916-02	B010352		MICROCIRCUIT,DI:8-2 INP 3-STATE BFR,BURN	27014	DM81LS97A
U305	156-0480-02	B010352		MICROCIRCUIT,DI:QUAD 2 INP & GATE	01295	SN74LS08NP3
U315	160-1454-00	B010352		MICROCIRCUIT,DI:4096 X 8 EPROM,PRGM	80009	160-1454-00
U405	156-0469-02	B010352		MICROCIRCUIT,DI:3/8 LINE DCDR	01295	SN74LS138NP3
				4052R08		
A2	670-6291-00	B010100	B010690	CKT BOARD ASSY:16K ROM PACK	80009	670-6291-00
A2	670-7487-00	B010691	B011181	CKT BOARD ASSY:16K ROM PACK	80009	670-7487-00
A2	670-7487-01	B011182	B011518	CKT BOARD ASSY:SIGNAL PROCESSING 2 ROM PACK	80009	670-7487-01
A2	670-7487-02	B011519		CKT BOARD ASSY:SIGNAL PROCESSING 2 ROM PACK	80009	670-7487-02
C1	283-0010-00	B010100	B010690	CAP.,FXD,CER DI:0.05UF,+100-20%,50V	56289	1C10Z5U503Z050B
C1	283-0422-00	B010691		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
C11	283-0010-00	B010100	B010690	CAP.,FXD,CER DI:0.05UF,+100-20%,50V	56289	1C10Z5U503Z050B
C13	283-0422-00	B010691		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
C15	290-0804-00	B010691		CAP.,FXD,ELCTLT:10UF,+50-10%,25V	55680	ULA1E100TEA
C21	283-0010-00	B010100	B010690	CAP.,FXD,CER DI:0.05UF,+100-20%,50V	56289	1C10Z5U503Z050B
C111	283-0422-00	B010691		CAP.,FXD,CER DI:0.047UF,+80-20%,50V	04222	DG015E473Z
C201	283-0010-00	B010100	B010690	CAP.,FXD,CER DI:0.05UF,+100-20%,50V	56289	1C10Z5U503Z050B
C301	290-0167-00	B010100	B010690	CAP.,FXD,ELCTLT:10UF,20%,15V	56289	150D106X0015B2
R111	315-0202-00	B010691	B011181	RES.,FXD,CMPSN:2K OHM,5%,0.25W	01121	CB2025
R221	315-0472-00	B010100	B010690	RES.,FXD,CMPSN:4.7K OHM,5%,0.25W	01121	CB4725
U1	160-0348-00	B010100	B010690	MICROCIRCUIT,DI:8192 X 8 ROM,CUSTOM MASK	80009	160-0348-00
U1	160-1417-00	B010691		MICROCIRCUIT,DI:4096 X 8 EPROM,PRGM	80009	160-1417-00
U11	160-1418-00	B010691		MICROCIRCUIT,DI:4096 X 8 EPROM,PRGM	80009	160-1418-00
U21	156-0469-02	B010100	B010690	MICROCIRCUIT,DI:3/8 LINE DCDR	01295	SN74LS138NP3
U111	156-0469-02	B010691		MICROCIRCUIT,DI:3/8 LINE DCDR	01295	SN74LS138NP3

Fig. & Index No.	Tektronix Part No.	Serial/Model No. Eff	Dscont	Qty	1 2 3 4 5	Name & Description	Mfr Code	Mfr Part Number
1-1	380-0384-01			1		HSG HALF,ROM PK:LID,ABS	80009	380-0384-01
						••••••••••••••(ATTACHING PARTS)••••••••••••		
-2	211-0102-00			4		SCREW,MACHINE:4-40 X 0.50,FLH,100 DEG STL	83385	ORD BY DESCR
						••••••••••••(END ATTACHING PARTS)•••••••••		
-3	367-0189-00			1		HANDLE,BOW:2.62 L,SST	91260	ORD BY DESCR
-4	380-0343-01			1		HSG HALF,PTR:INNER,ABS	80009	380-0343-01
-5	----- -----			1		CKT BOARD ASSY:16K ROM PACK(SEE A1 REPL)		
	----- -----			-		(4051R08 ONLY)		
	----- -----			1		CKT BOARD ASSY:16K ROM PACK(SEE A2 REPL)		
	----- -----			-		(4052R08 ONLY)		
-6	136-0578-00			4		.SKT,PL-IN ELEK:MICROCKT,24 PIN,LOW PRFL	09922	DILB24P-108
	----- -----			-		.(4051R08 ONLY)		
	136-0578-00	B010100	B010690	1		.SKT,PL-IN ELEK:MICROCKT,24 PIN,LOW PRFL	09922	DILB24P-108
	----- -----			-		.(4052R08 ONLY)		
	136-0751-00	B010691		4		.SKT,PL-IN ELEK:MICROCKT,24 PIN	09922	DILB24P108
	----- -----			-		.(4052R08 ONLY)		
-7	----- -----			1		MKR SET,IDENT:MKD 4051 SIGNAL PROC #2		
	----- -----			1		MKR SET,IDENT:MKD 4052 SIGNAL PROC #2		
						STANDARD ACCESSORIES		
	070-2841-00			1		MANUAL,TECH:INSTRUCTION	80009	070-2841-00



Section 5
SCHEMATICS

Symbols and Reference Designators

Electrical components shown on the diagrams are in the following units unless noted otherwise:

Capacitors = Values one or greater are in picofarads (pF). Values less than one are in microfarads (μF).

Resistors = Ohms (Ω).

Graphic symbols and class designation letters are based on ANSI Standard Y32.2-1975.

Logic symbology is based on ANSI Y32.14-1973 in terms of positive logic. Logic symbols depict the logic function performed and may differ from the manufacturer's data.

Abbreviations are based on ANSI Y1.1-1972.

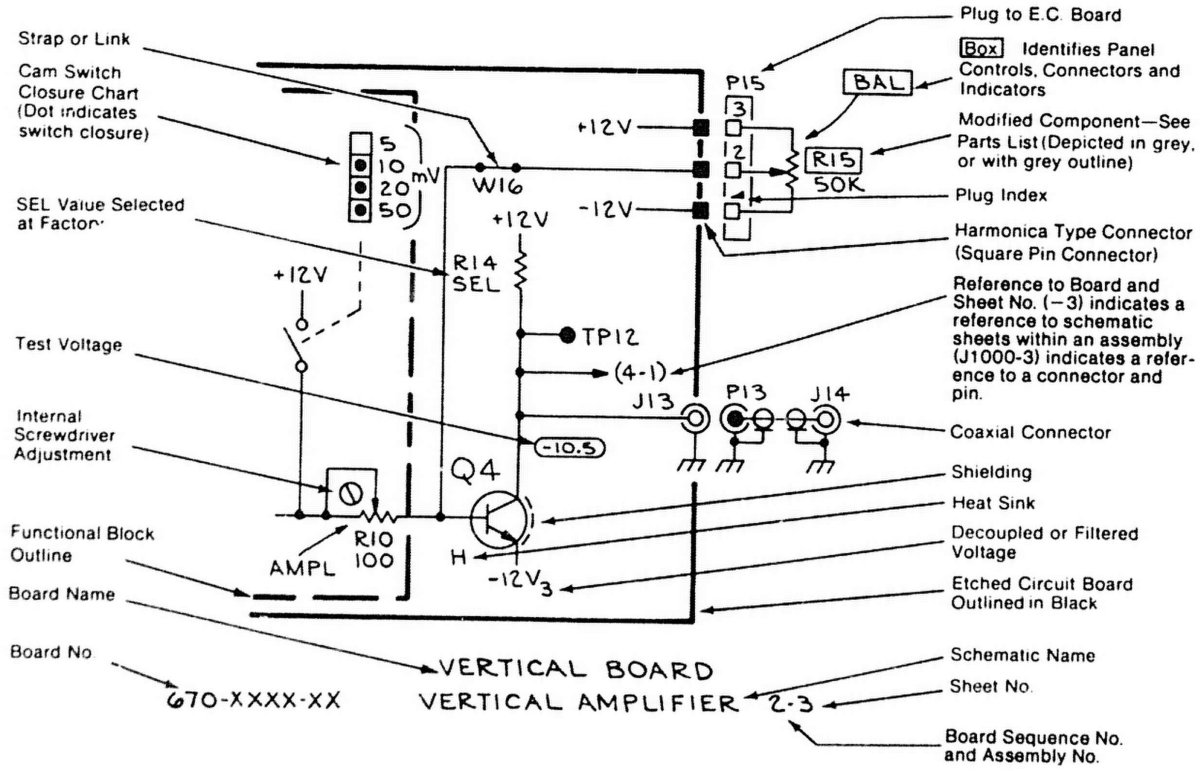
Other ANSI standards that are used in the preparation of diagrams by Tektronix, Inc. are:

Y14.15, 1966	Drafting Practices.
Y14.2, 1973	Line Conventions and Lettering.
Y10.5, 1968	Letter Symbols for Quantities Used in Electrical Science and Electrical Engineering.

The following prefix letters are used as reference designators to identify components or assemblies on the diagrams.

A	Assembly, separable or repairable (circuit board, etc)
AT	Attenuator, fixed or variable
B	Motor
BT	Battery
C	Capacitor, fixed or variable
CB	Circuit breaker
CR	Diode, signal or rectifier
DL	Delay line
DS	Indicating device (lamp)
E	Spark Gap, Ferrite bead
F	Fuse
FL	Filter
H	Heat dissipating device (heat sink, heat radiator, etc)
HR	Heater
HY	Hybrid circuit
J	Connector, stationary portion
K	Relay
L	Inductor, fixed or variable
M	Meter
P	Connector, movable portion
Q	Transistor or silicon-controlled rectifier
R	Resistor, fixed or variable
RT	Thermistor
S	Switch or contactor
T	Transformer
TC	Thermocouple
TP	Test point
U	Assembly, inseparable or non-repairable (integrated circuit, etc.)
V	Electron tube
VR	Voltage regulator (zener diode, etc.)
W	Wirestrap or cable
Y	Crystal
Z	Phase shifter

The following special symbols may appear on the diagrams:



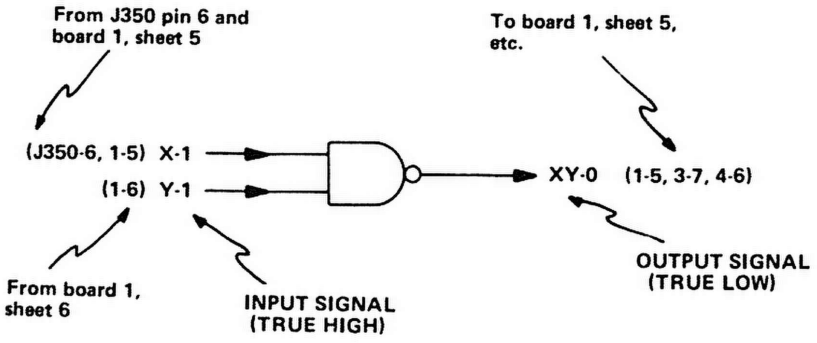
1. TRUE HIGH and TRUE LOW Signals

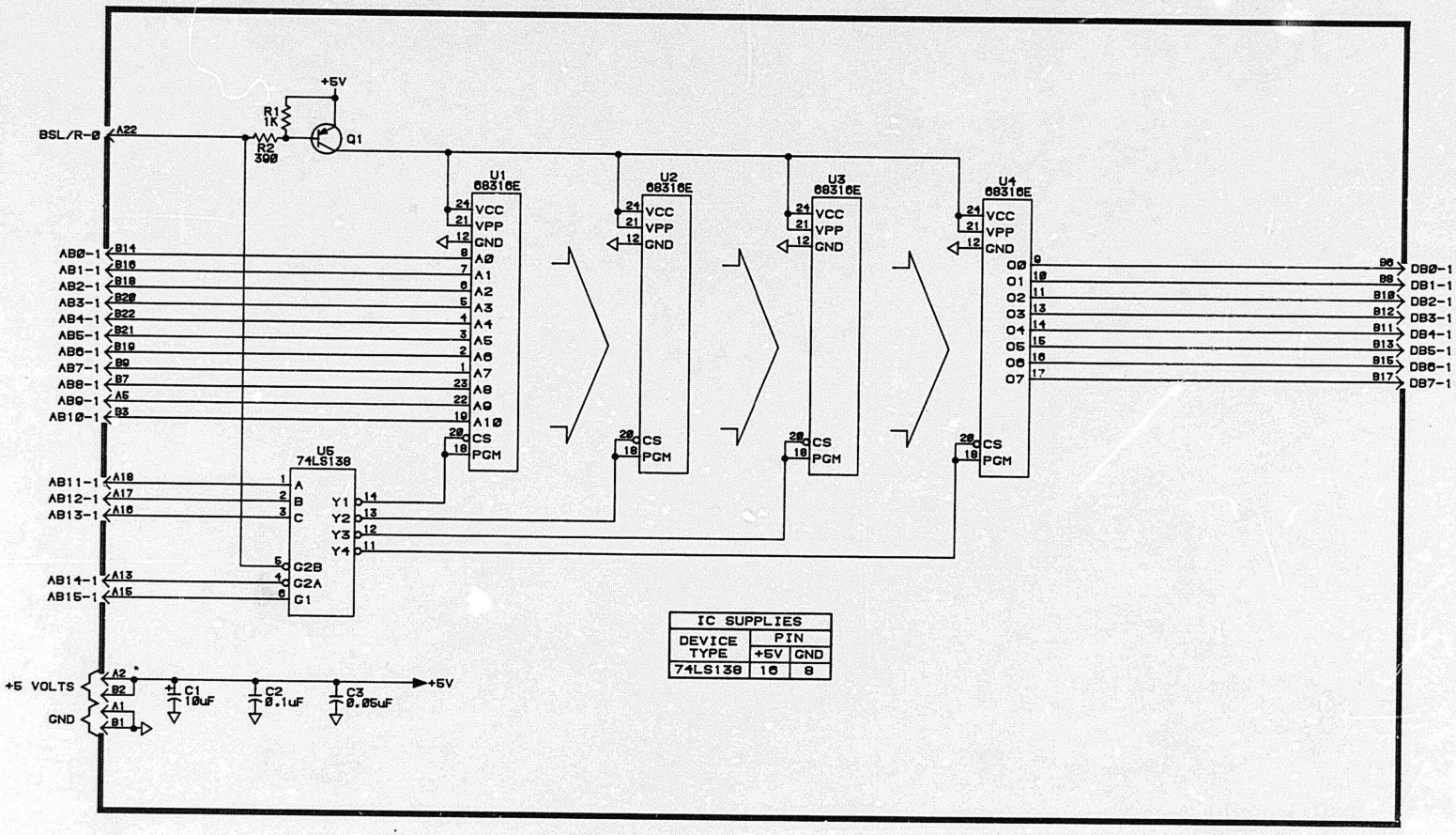
Signal names on the schematics are followed by −1 or −0. A TRUE HIGH signal is indicated by −1, and a TRUE LOW signal is indicated by −0.

SIGNAL−1 = TRUE HIGH
SIGNAL−0 = TRUE LOW

2. Cross-References

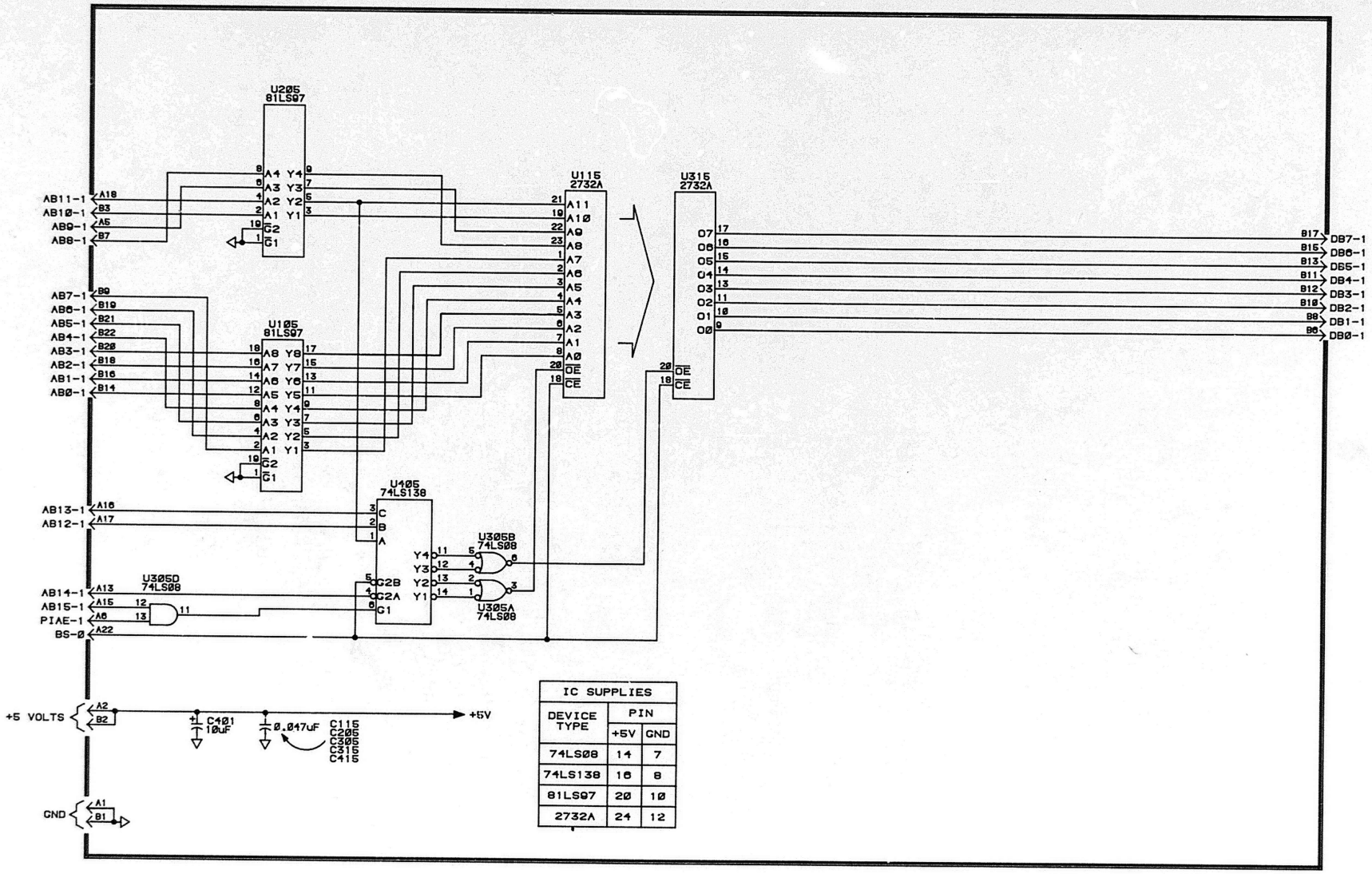
Schematic cross-references (from/to information) are included on the schematics. The "from" reference only indicates the signal "source," and the "to" reference lists all loads where the signal is used. All from/to information will be enclosed in parentheses.





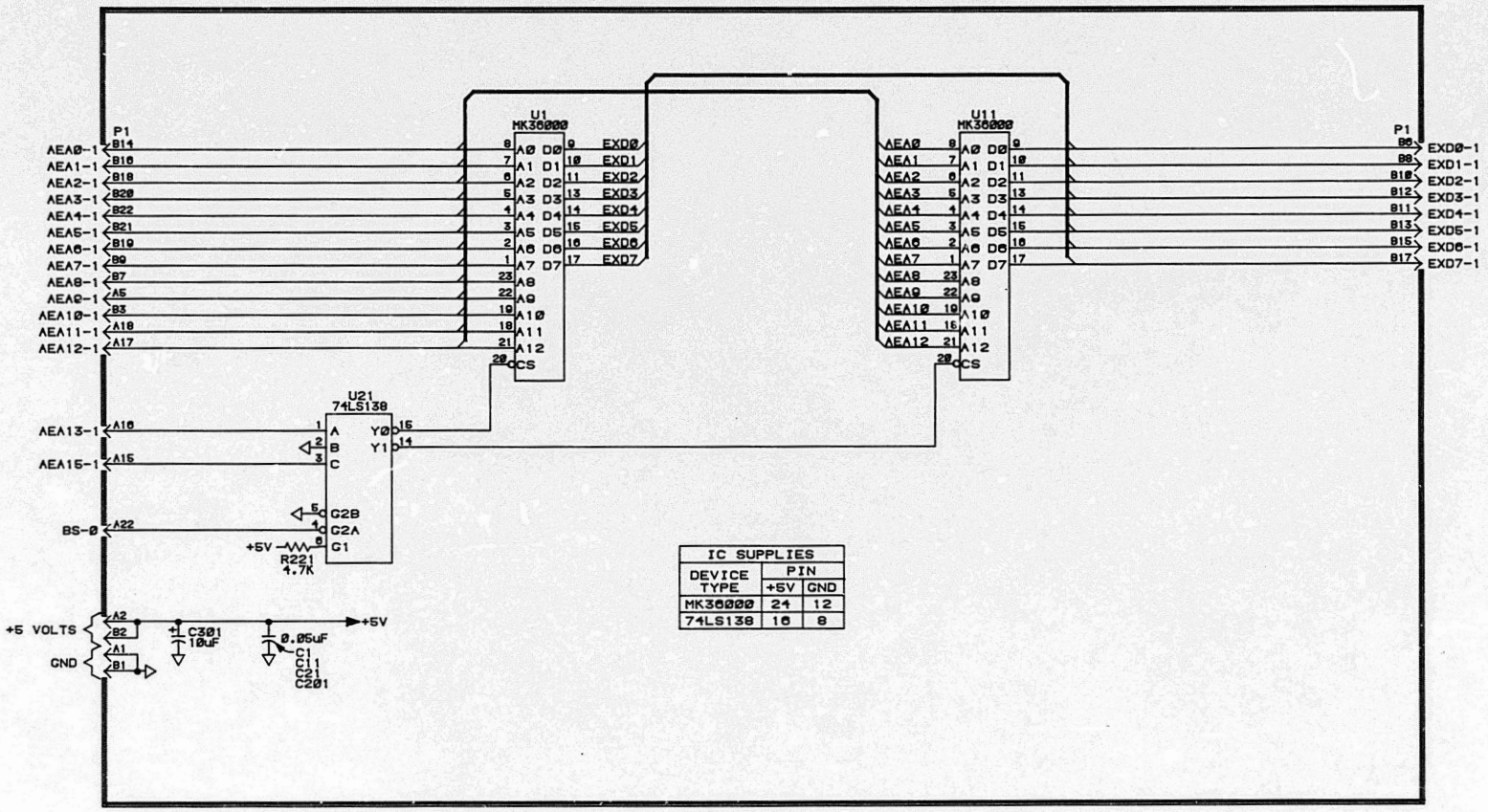
IC SUPPLIES		
DEVICE TYPE	PIN	
	+5V	GND
74LS138	16	8

INSTRUMENT:	NOTES:	BOARD: 670-6290-00	ASSEMBLY-SHEET:
4051R08	REV, SEP 1982 2841-51	SIGNAL PROCESSING ROM PACK NO.2 (FFT)	A1-1 (1 OF 1)



IC SUPPLIES		
DEVICE TYPE	PIN	
	+5V	GND
74LS08	14	7
74LS138	16	8
81LS97	20	10
2732A	24	12

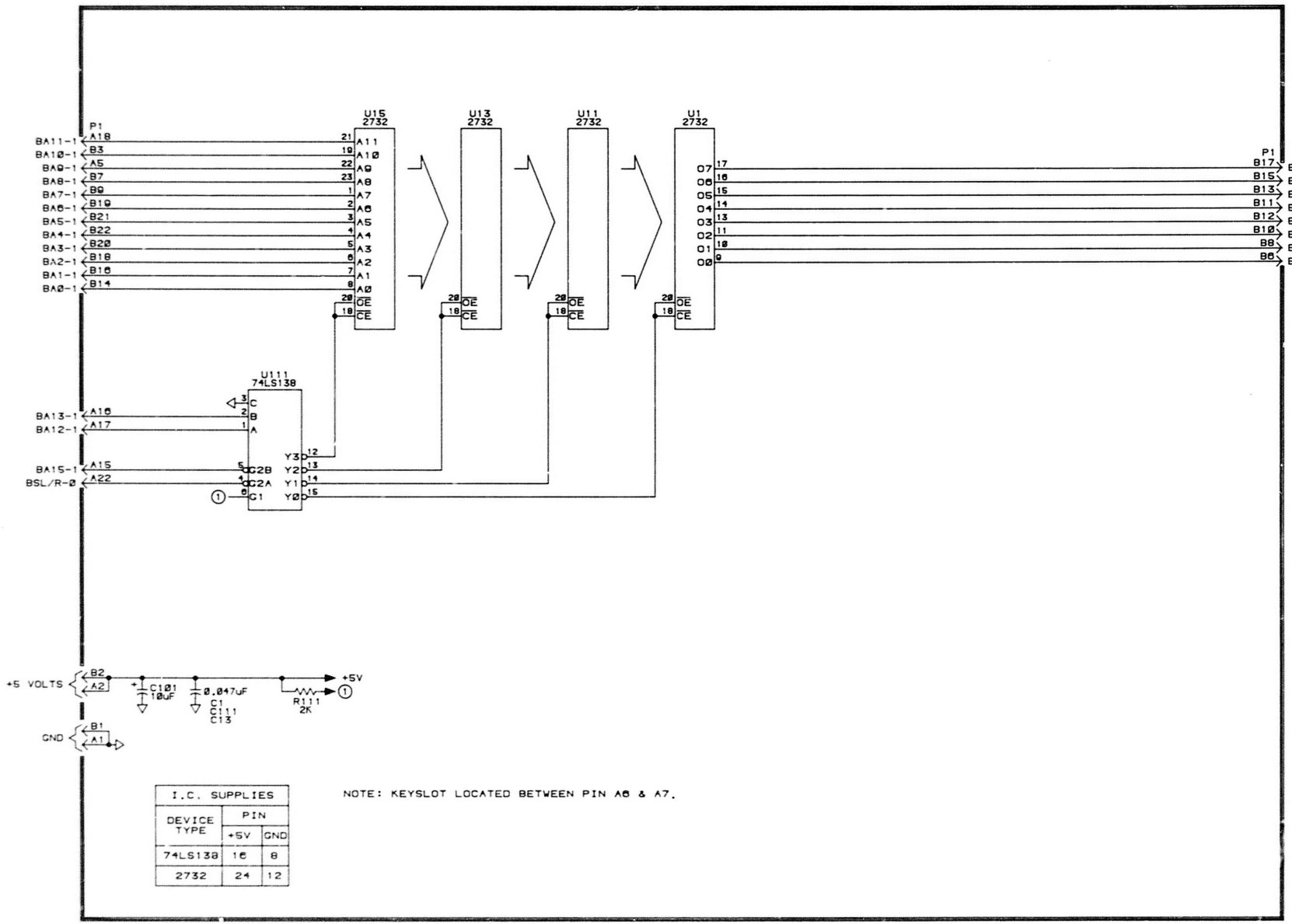
INSTRUMENT:	NOTES:	BOARD: 670-7495-00 8K ROM PACK	ASSEMBLY-SHEET:
4051R08	ADD, SEP 1982 2841-52	SIGNAL PROCESSING ROM PACK NO.2 (FFT)	A1-1 (1 OF 1)



IC SUPPLIES		
DEVICE TYPE	PIN	
	+5V	GND
MK36000	24	12
74LS138	16	8

INSTRUMENT:	NOTES:	BOARD: 670-6291-00	ASSEMBLY-SHEET:
4052R08	KEY SLOT LOCATED BETWEEN PINS A6/B6 & A7/B7 OF P1. REV, SEP 1982 2841-53	SIGNAL PROCESSING ROM PACK NO.2 (FFT)	A2-1 (1 OF 1)

4052R08
670-6291-00
A2-1



I.C. SUPPLIES		
DEVICE TYPE	PIN	
	+5V	GND
74LS138	16	8
2732	24	12

INSTRUMENT:	NOTES:	BOARD: 670-7487-00 16K ROM PACK	ASSEMBLY-SHEET:
4052R08	ADD, SEP 1982 2841-S4	SIGNAL PROCESSING ROM PACK NO.2 (FFT)	A2-1 (1 OF 1)

Appendix A

AN INTRODUCTION TO THE DFT AND FFT ALGORITHMS OF SIGNAL PROCESSING ROM PACK NO. 2

In this brief sketch, the discrete Fourier transform (DFT) is introduced by viewing it as a discrete time approximation to the better known Fourier integral transform for continuous time functions. The discrete Fourier transform, and in particular the fast Fourier transform (FFT) algorithm as implemented in Signal Processing ROM Pack No. 2 are defined. The exposition concludes with a short description of sources of error and their magnitudes in using the ROM Pack No. 2 FFT command.

THE DISCRETE FOURIER TRANSFORM AND INTEGRAL FOURIER TRANSFORM

The Fourier transform of a continuous time function, x(t), for $-\infty < t < \infty$, is defined as:

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

where time, t, is in seconds and frequency, f, is in Hertz. This integral defines a function of frequency, which will be denoted by X(f). The functions x(t) and X(f) form a transform pair; the relationship for recovering x(t) from its frequency domain description, X(f), is given by:

$$x(t) = \int_{-\infty}^{\infty} X(f)e^{j2\pi ft}df \qquad (2)$$

For questions of existence, uniqueness, and for any of the many relevant details not mentioned here, refer to references 4 and 19, in Appendix B.

In processing waveforms with a digitizer (such as the Tektronix 7912AD), a finite set of samples evenly spaced in time are taken to represent an analog signal, x(t), and these are conveniently denoted by $x(n\Delta t)$, $n = 0,1,\ldots,N-1$, where Δt is the sampling interval in seconds. Letting $\Delta f = 1/(N\Delta t)$, the reciprocal of the time record length, a discrete approximation to X(f) can be obtained by replacing the integral sign in (1) with a discrete summation and by replacing the differential with Δt so that:

$$X_d(k\Delta f) = \Delta t \cdot \sum_{n=0}^{N-1} x(n\Delta t) e^{-j2\pi k\Delta f n\Delta t} \qquad (3)$$

$$= \Delta t \cdot \sum_{n=0}^{N-1} x(n\Delta t) e^{-j2\pi kn/N} \qquad \text{for } k = 0,1,\ldots,N-1$$

In the above formulation, the discrete set of time samples $x(n\Delta t)$, for $n = 0,1,\ldots,N-1$, corresponds in the frequency domain to a discrete set of frequency components or Fourier coefficients, $X_d(k\Delta f)$, $k = 0,1\ldots,N-1$, which are, in general, complex numbers. In addition, the fact that the exponentials $e^{-j2\pi kn/N}$ are periodic functions of the index k, with period N, implies that the discrete Fourier transform, $X_d(k\Delta f)$, is also periodic with period N and can be taken to be defined for all integers k. A final notational remark: The subscript attached in denoting discrete Fourier coefficients is in order since usually the value of the discrete transform does not coincide with the continuous Fourier transform at the same frequency; i.e., $X_d(k\Delta f) \neq X(k\Delta f)$. However, in certain instances equality may in fact hold. The effects of aliasing and time truncation lead to the discrepancies between the discrete and continuous Fourier transforms; these discrepancies aliasing and time truncation, are discussed later.

Returning to equation (2) and making substitutions similar to those made in developing the (direct) discrete Fourier transform, one is led to the inverse discrete Fourier transform:

$$x(n\Delta t) = \Delta f \cdot \sum_{k=0}^{N-1} X_d(k\Delta f) e^{j2\pi k\Delta f n\Delta t} \qquad (4)$$

$$= \frac{1}{N\Delta t} \cdot \sum_{k=0}^{N-1} X_d(k\Delta f) e^{j2\pi kn/N}$$

It is not, of course, immediately clear that this equation is actually an equality, and to verify that the N samples, $x(n\Delta t)$, can be recovered from the N Fourier coefficients, $X_d(k\Delta f)$, requires introduction of the discrete orthogonality relationships (see, for example, reference 9). Accepting (4), it follows that $x(n\Delta t)$ is also implicitly defined for integers other than 0, 1, . . ., N-1, and is periodic, due to the periodicity of $e^{j2\pi kn/N}$, this time considered as a function of the index n. Thus, the discrete Fourier transform relates time-domain — frequency-domain pairs that are either explicitly given as periodic sequences or implicitly acknowledged to be such!

Some simplifications of notation lead to more commonly encountered notational forms. Letting $\Delta t = 1$ so that $\Delta f = 1$, and replacing $X_d(k\Delta f) = X_d(k/N)$ by $X_d(k)$, one is left with:

$$X_d(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}$$

and

$$x(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X_d(k)e^{j2\pi kn/N}$$

The Discrete Fourier Transform of ROM Pack No. 2

When using the FFT command of the Signal Processing ROM Pack No. 2, the input array represented by x(n), for n= 0,. . .,N-1, must be real-valued. This fact allows further simplification of the results of the DFT:

$$X_d(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \qquad k = 0,\ldots,N-1.$$

First, when k= 0, the complex term $e^{-j2\pi kn/N}$ equals 1; the result of the DFT:

$$X_d(0) = \sum_{n=0}^{N-1} x(n)$$

is a real number also and is the DC term. Note that by dividing the DC term by the number of elements, N, in the array, the mean value of the input array is calculated.

Second, when k= N/2, the term $e^{-j2\pi kn/N} = \cos(\pi n)$. The DFT result is again real valued, since

$$X_d(N/2) = \sum_{n=0}^{N-1} x(n)\cos(\pi n).$$

This value is called the Nyquist term and represents the value of the DFT at the Nyquist frequency, $f_N = 1/2\Delta t$.

Third, by using symmetry properties, further simplification can be done. Since the Fourier coefficients are periodic ($X_d(k) = X_d(k \pm N)$) as noted above, they are defined for negative integers, i.e., negative frequencies with

$$\begin{aligned} X_d(-1) &= X_d(N-1) \\ X_d(-2) &= X_d(N-2) \\ &\vdots \\ X_d(-N+1) &= X_d(1). \end{aligned}$$

Thus, instead of presenting $X_d(0),\ldots,X_d(N-1)$, the full set of Fourier coefficients may be presented (for N even) as

$$X_d(-N/2+1),\ldots,X_d(0),\ldots,X_d(N/2-1),X_d(N/2).$$

And, by requiring the input array, x(n), to be real-valued, these coefficients display a symmetry about $x_d(0)$: the real parts in the negative frequencies are mirror images of those in the positive frequencies (mathematically, Real $[X_d(-k)]$ = Real $[X_d(N/2-k)]$); also, the imaginary parts in the negative frequencies are inverse mirror images of those in the positive frequencies (Imag $[X_d(-k)] = -\text{Imag}\,[X_d(N/2-k)]$). Expressed another way, the values of the Fourier coefficients at negative frequencies are complex conjugates of their counterparts in the positive frequencies. Therefore, for real-valued functions, x(n), the full set of N Fourier coefficients can be fully represented by the N/2+ 1 values of $X_d(k)$ given by the equation:

$$X_d(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N} \qquad \text{for } k = 0,\ldots,N/2$$

Note that to correctly determine the inverse discrete Fourier transform, the symmetry properties described above must be used to obtain N real-valued time-domain samples from N/2+ 1 complex-valued frequency-domain samples. This is automatically done in performing the IFT.

Polar Form of Representing the Fourier Transform

Commonly one is not interested in the complex Fourier coefficients with their real and imaginary components or rectangular coordinate representation, but in their polar form in terms of amplitude and phase:

$$X_d(k) = A(k)e^{j\phi(k)}$$

where the array of phases (in radians) is restricted to the interval $-\pi \leq \phi(k) < +\pi$. This "discontinuous" phase can be correctly converted to continuous phase by "unwrapping" it modulo 2π if the frequency spacing between coefficients is small enough. More precisely, the continuous phase algorithm of the Signal Processing Rom Pack No. 2 performs valid reconstructions if the correct phase does not exhibit transitions of $\pm\pi$ or more from one Fourier coefficient to the next.

ALIASING AND TIME TRUNCATION ERRORS

Having defined the Fourier integral transform and having introduced the discrete Fourier transform by way of an approximation to the integral transform, some discussion of the errors of this approximation are in order. These systematic errors fall into two categories: The first, called aliasing, is determined by the asymptotic (high frequency) behavior of the integral transform, X(f), and the choice of the sampling interval $\triangle t$; the second, time truncation error, results from the asymptotic properties of the signal, x(t), and the length of the time window ($N\triangle t$ in the above discussion) over which it is observed.

To illustrate the aliasing problem, both mathematically and pictorially (Figure A-1), temporarily remove the restriction of the finite length time window and consider the infinite train of time samples $x(n\triangle t)$, $n = 0, \pm 1, \pm 2, \ldots, \pm\infty$, and let

$$X_s(f) = \triangle t \cdot \sum_{n=-\infty}^{\infty} x(n\triangle t)e^{-j2\pi fn\triangle t} \qquad (7)$$

as an initial approximation to the Fourier integral (1). The relationship between (7) and the corresponding integral transform then defines the errors due to aliasing. This relationship is

$$X_S(f) = \sum_{k=-\infty}^{\infty} X(f - \frac{k}{\Delta t}) \qquad (8)$$

note that the $X_S(f)$ differs from X(f) by the aliases $X(f \pm 1/\Delta t)$, $X(f \pm 2/\Delta t)$, . . ., with each alias separated from the base frequency f by some multiple of twice the Nyquist interval, $f_\eta = 1/2\Delta t$.

Notice that $X_S(f)$ is periodic with period $2 \cdot f_\eta$, twice the Nyquist interval; hence any section of $X_S(f)$ of length $2 \cdot f_\eta$ serves to define $X_S(f)$ for all frequencies. But it is reasonable to consider $X_S(f)$ an estimate of X(f) only for frequencies from $-f_\eta$ to f_η, noting as above that frequency components over this interval have been aliased by higher frequency (above the Nyquist frequency) "foldover."

For an experimental verification of aliasing, take any pulse-like signal with frequency components over a fairly wide range of frequencies (not extremely narrowband) and examine it at a number of Time/Division settings, such that the signal is zero outside the time window defined by each setting. Now compare each of the sampled signals in both the time domain and the frequency domain, the latter by using the FFT command. The last restriction, requiring that the signal be zero outside of each time window, assures that time truncation effects do not arise; hence systematic differences between the discrete Fourier transforms of the various sampled signals are due solely to aliasing effects.

Controlling Errors Due to Aliasing

Should one be examining a signal, x(t), that is bandlimited, say $X(f) = 0$ for $|f| > f_c$, then aliasing can be completely avoided by simply choosing Δt small enough to ensure a Nyquist frequency exceeding f_c; i.e., such that $f_\eta = 1/2\Delta t > f_c$ or $\Delta t < 1/2f_c$. Practically speaking, the waveform should be sampled densely enough to fix all the inherent detail in the signal — the sampled signal must "look like" its analog parent.

Unfortunately, many signals are not, in fact, band-limited. For example, causal energy signals, ones for which $x(t) = 0$ for $t < 0$ and such that $\int_0^\infty x2(t)\, dt < \infty$, cannot have band-limited Fourier transforms (reference 19); hence, some aliasing will always be present in sampling such signals. Furthermore, actually determining a bound on the aliasing error requires some knowledge of the asymptotic behavior (here meaning outside the interval $-f_\eta$ to f_η) of the integral transform, X(f).

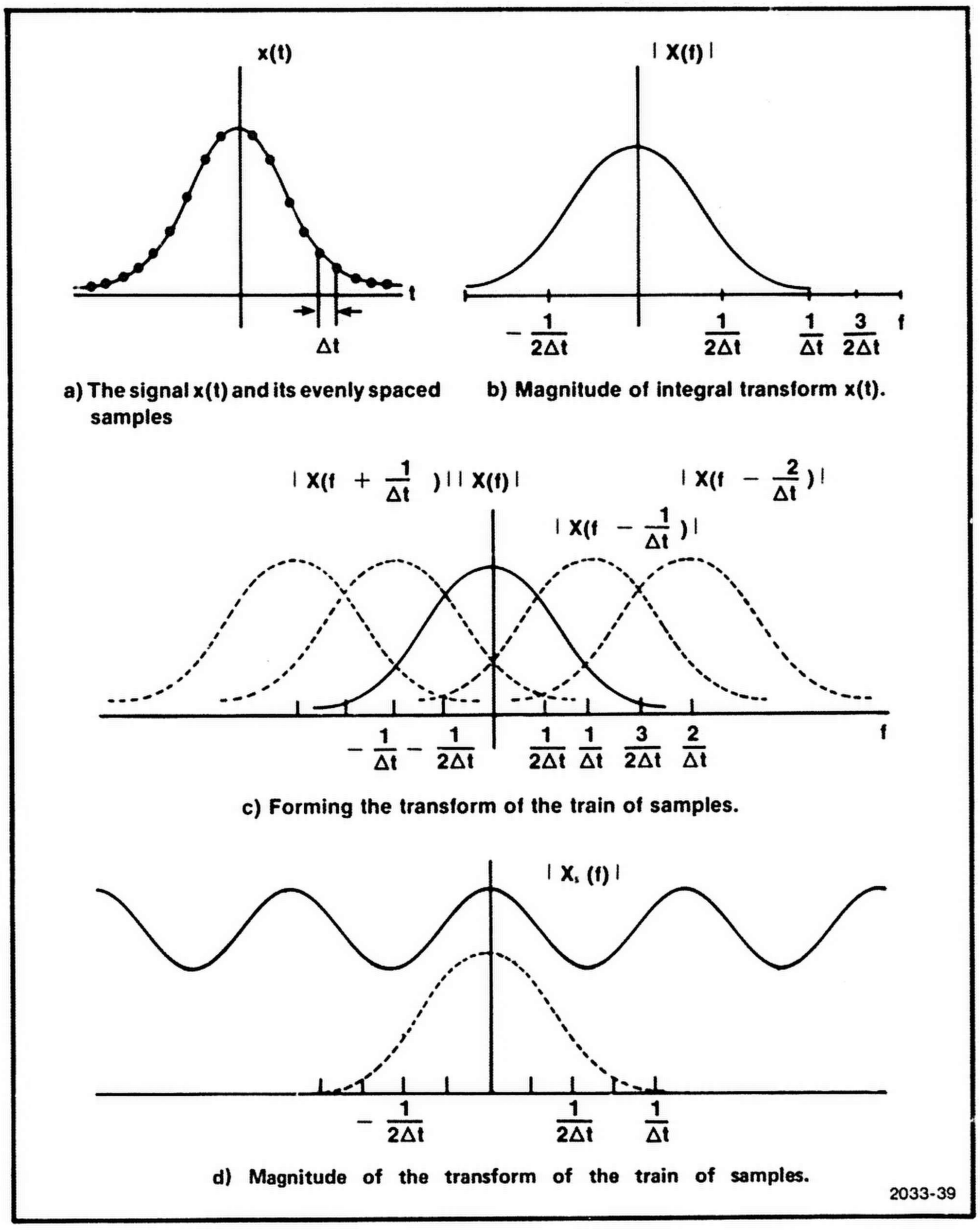


Figure A-1.

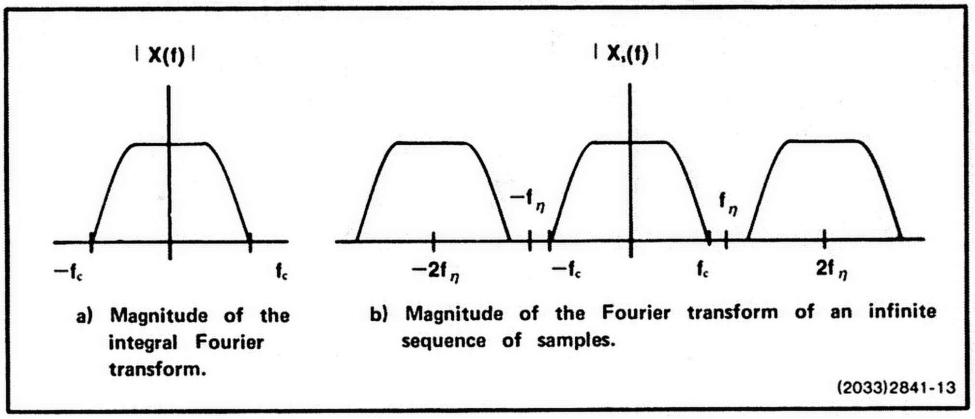


Figure A-2.

A case of special interest, where the magnitude of the aliasing error can be bounded, occurs for signals whose transforms are bounded outside $(-f_\eta, f_\eta)$ by a function proportional to $1/f^2$. The results summarized here are presented in reference 26 and are of interest since: first, they indicate the type of analytical information required to bound aliasing errors; and second, the $1/f^2$ bound on asymptotic behavior often holds in practical measurement situations. Now suppose that

$$|X(f)| \leq A/f^2, \text{ for } |f| > |f_\eta| . \tag{9}$$

Then the alias error, $E(f) = X(f) - X_S(f)$, for $-f_\eta \leq f < f_\eta$, has a magnitude bounded by:

$$|E(f)| \leq A(\pi \triangle t)^2, \text{ for } -f_\eta \leq f < f_\eta .$$

Setting $E_S(f) = 0$ for $|f| > f_\eta$ and using this as an estimate of $X(f)$ for $|f| > f_\eta$, the inequality (9) directly gives a bound on the error outside of $(-f_\eta, f_\eta)$, allowing one to take the zero-extended $X_S(f)$ for an approximation to the entire integral transform $X(f)$.

Lacking information about the asymptotic properties of the amplitude spectrum of a signal, as typified above, what is one to do if a bound on the aliasing error is desired? Analog filtering of x(t) prior to sampling, such that (9) is satisfied, provides an escape route. In fact, the use of anti-aliasing filters is a common practice in the sampling of signals!

Controlling Time Truncation Errors

Restricting attention to causal signals [$x(t)=0, t<0$], the infinite sequence of time samples $x(n\Delta t)$, for $n=0, \pm 1, \pm 2, \ldots, \infty$, with transform $X_S(f)$, provides a convenient vehicle for describing aliasing errors. Since the infinite sample train must necessarily be truncated in time for anything other than mathematical paperwork, additional errors in approximating X(f) result. Specifically, setting $x(n\Delta t)=0$ for $n=0,1,\ldots,N-1$, one is left with a finite sequence Fourier transform:

$$X_d(f) = \Delta t \cdot \sum_{n=0}^{N-1} x(n\Delta t)e^{-j2\pi fn\Delta t}$$

as the approximation for X(f). An additional error of

$$E(f) = \Delta t \cdot \sum_{n=N}^{\infty} x(n\Delta t)e^{-j2\pi fn\Delta t} \qquad (10)$$

is incurred. In a fashion similar to the previous discussion, the ability to bound the signal x(t) outside of the time window is sufficient to bound the truncation error. Suppose that

$$|x(f)| \leq Ae^{-\alpha t}$$

for $t > N\Delta t$, where $\alpha > 0$ and A is an arbitrary constant. This assumption clearly places some bound on the magnitude of (10). As is shown in Chapter 2 of reference 26, a resulting bound on the error is:

$$|E(f)| \leq (1 + 1/\alpha\Delta t)\, \Delta t \cdot A \cdot e^{-\alpha N\Delta t}$$

The assumption that $x(t)=0$ for $t<0$ is not critical to the construction of this error bound, and a similar bound could be developed for non-causal signals provided that x(t) was exponentially bounded for $t<0$.

Apart from increasing the record length to enclose more of the time function, other methods of control are also at one's disposal. For some special signal types, exact or approximate corrections for time truncation errors can be made. A common example is a step with an approximate correction as given in reference 26, Chapter 3. Another approach involves modifying the implicit rectangular shape of the time window, gradually forcing the signal to zero toward the window edges, and thus making the signal appear to

be zero outside the time window. Such windowing techniques essentially modify but do not correct for time truncation errors, and correspond to filtering or smoothing the exact and desired transform X(f).

One problem associated with the implicit rectangular time window is the associated sin(cf)/cf filter response with side lobes that decay slowly as 1/cf, c being a constant. The problem of leakage into these side lobes can be diminished by appropriately tapering the time history x(t). (This tapering can be done with the ROM pack's TAPER function.) This topic is discussed in reference 17, and a qualitative-pictorial discussion of windowing, as well as aliasing, can be found in reference 25.

In summary, time truncation errors generally arise, but such errors are bounded in cases where the signal magnitude is bounded by a known function (such as a negative exponential) outside the time window. Mathematical correction for truncation errors is possible in some simple cases, and windowing techniques allow for modification of time truncation errors in systematic way.

THE FFT ALGORITHM AND ACCUMULATED ROUNDOFF ERRORS

The ROM Pack No. 2 employs the Sande-Tukey decimation-in-frequency FFT algorithm with a quarter-period negative sine table for generating the necessary complex exponentials. The sine wave is sampled densely enough to perform discrete Fourier transforms of any real data arrays whose length is a power of two between 16 and 1024, inclusive. The algorithm is a floating-point fast Fourier transform, requiring approximately 65 seconds in the 4051 (approximately 4 seconds in the 4052) to compute the 513 complex frequency samples of a real waveform of 1024 points.

Rounding errors occur in the course of computing an FFT from 1) the approximate nature of arithmetic operations, and 2) the rescaling of intermediate waveform results caused by overflows. For more information on errors, consult reference 30.

Appendix B

BIBLIOGRAPHY

(1) Bendat, J.S., and A.G. Piersol. **Random Data: Analysis and Measurement Procedures.** New York: Wiley, 1971.

(2) Bergland, G.D. "A Guided Tour of the Fast Fourier Transform," **IEEE Spectrum.** July 1959, pp. 41-52.

(3) Blackman, R.B., and J.W. Tukey. **The Measurement of Power Spectra.** New York: Dover Publications, 1958.

(4) Bracewell, R. **The Fourier Transform and Its Applications.** New York: McGraw-Hill, 1965.

(5) Brigham, E.O. **The Fast Fourier Transform.** Englewood Cliffs: Prentice-Hall, 1974.

(6) Brigham, E.O., and R.E. Morrow. "The Fast Fourier Transform," **IEEE Spectrum.** Dec. 1967, pp. 63-70.

(7) Chirlian, P.M. **Basic Network Theory.** New York: McGraw-Hill, 1969.

(8) Cooley, J.W., and J.W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series," **Mathematics of Computation.** Vol. 19, April 1965, pp. 297-301.

(9) Cooley, J.W., P.A. Lewis, and P.D. Welch. "The Finite Fourier Transform," **IEEE Transactions on Audio and Electroacoustics.** Vol. AU-17, No. 2, June 1969.

(10) Gold, B., and C.M. Rader. **Digital Processing of Signals.** New York: McGraw, 1969.

(11) Griffiths, J.W.R., P.L. Stocklin, and C. Van Schooneveld, eds. **Signal Processing,** Proceedings of the NATO Advanced Study Institute on Signal Processing. New York: Academic Press, 1973.

(12) **IEEE Transactions on Audio and Electroacoustics,** Special issue on the fast Fourier transform and its application to digital filtering and spectral analysis. Vol. AU-15, No. 2, June 1967.

(13) **IEEE Transactions on Audio and Electroacoustics,** Special issue on the fast Fourier transform. Vol. AU-17, No. 2, June 1969.

(14) Jenkins, G.M., and D.G. Watts. **Spectral Analysis and Its Applications.** San Francisco: Holden-Day, 1968.

(15) Lange, F.H. **Correlation Techniques.** London: Iliffe Books Ltd., 1967. Published in the U.S.A. by D. Van Nostrand Company, Princeton, N.J.

(16) Oppenheim, A.V., and R.W. Schafer. **Digital Signal Processing.** Englewood Cliffs: Prentice-Hall, 1975.

(17) Otnes, R.K., and L. Enochson. **Digital Time Series Analysis.** New York: Wiley, 1972.

(18) Papoulis, A. "Error Analysis in Sampling Theory," **Proceedings of the IEEE.** Vol. 54, No. 7, July 1968.

(19) Papoulis, A. **The Fourier Integral and Its Applications.** New York: McGraw-Hall, 1962.

(20) **Proceedings of the IEEE**, Special issue on digital signal processing. Vol. 63, No. 4, April 1975.

(21) Rabiner, L.R., and B. Gold. **Theory and Application of Digital Signal Processing.** Englewood Cliffs: Prentice-Hall 1975.

(22) Rabiner, L.R., et al. "Terminology in Digital Signal Processing," **IEEE Transactions on Audio and Electroacoustics**. Vol. AU-20, Dec. 1972, pp. 322-337.

(23) Ramirez, R.W. "Fast Fourier Transform Makes Correlation Simpler," **Electronics.** June 26, 1975, pp. 98-103.

(24) Ramirez, R.W. "The Fast Fourier Transform's Errors Are Predictable, Therefore Manageable," **Electronics.** June 13, 1974, pp. 96-102.

(25) Ramirez, R.W., **The FFT: Fundamentals and Concepts**, Tektronix, Inc., Dec. 1975 (070-1754-00).

(26) Ross, G.F., A.M. Micolson, and Robert Smith, et. al. **Transient Behaviour of Microwave Networks**. Sperry Rand Research Center, Technical Report No. RADC-TR-68-92, April 1968.

(27) Roth, P.R. "Effective Measurements Using Digital Signal Processing," **IEEE Spectrum.** April 1971, pp. 62-70.

(28) Schafer, R.W., and L.R. Rabiner. "A Digital Signal Processing Approach to Interpolation," **Proceedings of the IEEE.** Vol. 61, June 1973, pp. 692-702.

(29) Van Valkenburg, M.E. **Network Analysis**, 2nd ed. Englewood Cliffs: Prentice-Hall, 1964.

(30) Peter D. Welch. "A Fixed-Point Fast Fourier Transform Error Analysis", **IEEE Transactions on Audio and Electroacoustics.** Vol. AU-17, No. 2, June 1969.

Appendix C

COMMAND SUMMARY

Function	Example	Purpose
FFT	CALL "FFT",X	Performs a fast Fourier transform on array X, and places the results in X.
IFT	CALL "IFT",X	Performs an inverse Fourier transform on array X, and places the results in X.
CONVL	CALL "CONVL",X,Y,Z	Convolves array X with array Y and places the results in Z. (X and Y are overwritten by intermediate results.)
CORR	CALL "CORR",X,Y,Z	Correlates array X with array Y and places the results in Z. (X and Y are overwritten by intermediate results.)
POLAR	CALL "POLAR",X,M,P,D1	Performs a rectangular-to-polar conversion on array X, placing the magnitude components in array M and the phase components in array P. (The optional D1 variable specifies the delay.)
TAPER	CALL "TAPER",X,T1	Multiplies array X by a cosine window and places the results in X. (The optional T1 variable specifies the amount of tapering.
UNLEAV	CALL "UNLEAV",X,R,I	Segregates the interleaved FFT data in array X, placing the real components in array R and the imaginary components in array I.
INLEAVE	CALL "INLEAVE",R,I,X	Interleaves the real components in array R with the imaginary components in array I, placing the results in array X.

Appendix D

UNDERSTANDING ERRORS

Error Number	Error Message
12	INVALID COMMAND ARGUMENT IN IMMEDIATE LINE — MESSAGE NUMBER 12 INVALID COMMAND ARGUMENT IN LINE xx — MESSAGE NUMBER 12 **Probable Cause** a. An array argument is incorrectly dimensioned or is defined as a simple variable. b. An optional argument has been defined as an array or is an undefined variable.
32	CALL NAME INVALID IN IMMEDIATE LINE — MESSAGE NUMBER 32 CALL NAME INVALID IN LINE xx — MESSAGE NUMBER 32 **Probable Cause** a. The correct ROM pack is not installed or incorrectly installed. b. The call name has been typed incorrectly. c. The ROM pack is defective.
36	UNDEFINED VARIABLE IN IMMEDIATE LINE — MESSAGE NUMBER 36 UNDEFINED VARIABLE IN LINE xx — MESSAGE NUMBER 36 **Probable Cause** a. An input array is not defined or contains undefined elements. b. An optional argument is undefined.

NOTE

For all routines except POLAR and TAPER, any extra arguments given are ignored. The correct answer is still returned. POLAR and TAPER, however, will return an error if there are extra arguments.