JT32
48

To the Graduate Council:

I am submitting herewith a thesis written by Jackson W. Frazier, Jr. entitled "Methods for Performing Discrete Fourier Transforms on the TMS32020 Microprocessor." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.

Bruce W. Bomar

Bruce W. Bomar, Major Professor

We have read this thesis
and recommend its acceptance:

Roy D Joseph

Accepted for the Council:

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at the University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Jackson W. Frazier Jr.

Date Dec 10, 1986

METHODS FOR PERFORMING DISCRETE FOURIER TRANSFORMS ON THE TMS32020 MICROPROCESSOR

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Jackson W. Frazier, Jr.

March 1987

In Memory of My Mother

Mary Ogletree Frazier

ACKNOWLEDGEMENTS

During the course of this project, a number of people gave me a great deal of encouragement and assistance.

I would like to express my sincere appreciation to my advisor and committee chairman, Bruce Bomar, and committee members Roy Joseph and Ronald Belz. The guidance and suggestions I received from them enabled me to make huge improvements in this work.

I also thank Ray Simar and Anita Massey of Texas Instruments Inc. for their timely help in obtaining the necessary TMS32020 software tools.

Many co-workers and friends have offered words of much needed encouragement and prodding which kept me on track to complete this project, and I thank them.

Finally, I can not express enough gratitude to my wife Carol for her encouragement and help, and to my children, Melanie, Jeffrey, and Daniel for their understanding while I completed this project.

ABSTRACT

Computation of the discrete Fourier transform (DFT) with a general purpose microprocessor offers the usual advantages of these chips. They are programmable, small, inexpensive, and use little power. However, DFT computation speed is not as great as could be obtained with an architecture more tailored to the application. The recent introduction of microprocessors with architectures designed specifically for digital signal processing (DSP) should offer much faster computation, but the question remains as to how well the DSP chips actually perform in DFT applications. In this study, one of the popular chips, the Texas Instruments TMS32020, is evaluated for DFT use.

Fourier transform theory is discussed and the general structures of DFT programs are introduced by computer programs written in BASIC. Direct calculation of the DFT and calculation via a Fast Fourier Transform (FFT) algorithm are considered. The architecture of the TMS32020 is examined and assembler programs for the direct DFT and FFT calculation of the DFT are presented. Signals of known amplitude, frequency, phase, and data sample size are used to verify the program performance. Execution speed as a function of DFT size is then determined.

The results show that the architecture and instruction set of the TMS32020 allow fast, compact code to be developed for the DFT. The memory space is large enough for a 32K point single butterfly radix-2 FFT computation. The direct DFT program performs a 64 point complex DFT in 7.16 milliseconds. The FFT speed is excellent, although not comparable to modern digital spectrum analyzers. The single butterfly radix-2 FFT program computes a 1024 point FFT in 85 milliseconds. The execution speed of the FFT program is faster than that of the direct DFT program for any data sample size. However, neither program is optimized for speed.

TABLE OF CONTENTS

CHAPTER		PAGE
1.	INTRODUCTION	1
2.	FOURIER TRANSFORM THEORY	3
	Introduction	3
	Fourier Series of Continuous Time Signals	3
	The Fourier Transform	5
	The Discrete Fourier Transform	6
	The Fast Fourier Transform	8
	Butterfly Diagrams	12
	Terminology	16
3.	PROBLEMS IN DFT RESULTS	17
	Introduction	17
	Aliasing	17
	Windowing and Leakage	21
4.	COMPUTATION OF THE DFT	26
	Introduction	26
	Direct Calculation of the DFT	26
	Calculation of the FFT	30
	A BASIC FFT Program	32
	BASIC FFT Results	36

CHAPTER PAGE

5. THE TMS32020 MICROPROCESSOR 42
Introduction 42
TMS32020 Architecture 42
Cross Assembler 45
Linker 49
TMS32020 Simulator 50
6. THE DFT ON THE TMS32020 52
The Direct DFT 52
TMS32020 Direct DFT Program 61
FFT Program for the TMS32020 66
7. RESULTS OF TMS32020 DFT PROGRAMS 71
8. CONCLUSIONS 83
LIST OF REFERENCES 89
APPENDICES 91
APPENDIX A. BASIC DFT PROGRAM. 92
APPENDIX B. BASIC FFT PROGRAM. 94
APPENDIX C. TMS32020 DIRECT DFT PROGRAM 98
APPENDIX D. TMS32020 FFT PROGRAM 113
VITA 127

LIST OF FIGURES

FIGURE		PAGE
2.1	Comparison of N^2 and N log N Operations	9
2.2	Butterfly Diagram, N = 8	13
2.3	Spreadsheet FFT, N = 8	15
3.1	Illustration of the Sampling Theorem	20
3.2	DFT of a Cosine Wave, No Leakage	22
3.3	DFT of a Cosine Wave, With Leakage	24
4.1	Direct DFT Algorithm	28
4.2	FFT Algorithm	33
4.3	FFT of a 2000 Hz. Cosine Wave	37
4.4	FFT of a 2000 Hz. Cosine Wave With Non-Integer Number of Cycles in the Window	39
4.5	FFT of a 2000 Hz. Sine Wave	40
4.6	FFT of a 2000 Hz. Square Wave (Even)	41
5.1	TMS32020 Functional Block Diagram	44
6.1	Sum of Products Using MAC	54
6.2	Sum of Products Using MPY	56
6.3	Table Multiply With Table End Test	59
6.4	TMS32020 Direct DFT Program Memory Map	62
6.5	Direct DFT Program Flow	65
6.6	TMS32020 FFT Program Memory Map	68
7.1	DFT of a 2000 Hz. Cosine Wave	72
7.2	DFT of a 2000 Hz. Cosine Wave With Non-Integer Number of Cycles in the Window	73
7.3	DFT of a 2000 Hz. Sine Wave	74

FIGURE PAGE

7.4 DFT of a 2000 Hz. Square Wave (Even) 75

7.5 FFT of a 2000 Hz. Cosine Wave 77

7.6 FFT of a 2000 Hz. Cosine Wave With Non-Integer Number of Cycles in the Window 78

7.7 FFT of a 2000 Hz. Sine Wave 79

7.8 FFT of a 2000 Hz. Square Wave (Even) 80

7.9 TMS32020 Machine Cycles for a N-point Complex DFT Using the Direct DFT and FFT Methods 81

7.10 TMS32020 FFT Program Machine Cycles Vs. N for a N-point Complex DFT 82

CHAPTER 1

INTRODUCTION

This investigation will attempt to answer the question of how well suited a modern digital signal processing (DSP) microprocessor is for performing the discrete Fourier transform (DFT). Even with efficient fast Fourier transform (FFT) algorithms, early generation general purpose microprocessors were too slow for most Fourier analysis applications. However, several manufacturers have introduced microprocessors with pipelined architecture and high clock speeds that appear to have excellent potential for fast DFT computation. One of these, the Texas Instruments' TMS32020, is investigated for DFT use.

First, background information on Fourier transform theory and the TMS320 family of DSP microprocessors is presented. Knowledge of Fourier transform theory is necessary to understand how the DFT is calculated on a digital computer and how the results are to be interpreted. Chapters 2 and 3 present the theory and some problems that arise in the use of the DFT.

Chapter 4 introduces the methods used for computing the DFT and FFT. The programs in this chapter are written in BASIC so that the program structure can be demonstrated without the complications of assembler notation and syntax.

The TMS32020 hardware and support software is discussed in Chapter 5. Enough information is provided for one to understand the assembler programs, but reference [13] is recommended as a more complete guide.

Direct DFT and decimation-in-frequency FFT assembler programs which were developed for this investigation are discussed in Chapter 6. Complete listing files are included in the appendices. These programs are not represented as being optimally fast, but care was taken to use the best available features of the microprocessor for the application. Results of the assembler DFT programs, which verify that the programs perform correctly and show how fast they execute, are presented in Chapter 7. Chapter 8 contains the summary and conclusions of this study, showing that the TMS32020 is a good performer for many DFT applications.

CHAPTER 2

FOURIER TRANSFORM THEORY

Introduction

This chapter presents the theory upon which the TMS32020 programs presented in following chapters are based. The sections in this chapter on the Fourier series, the Fourier transform, and the DFT are based on derivations in reference [4]. The FFT derivation follows that of reference [10].

Fourier Series of Continuous Time Signals

A periodic time analog signal $x_a(t)$ which is integrable over it's period can be written as the Fourier Series,

$$x_a(t)= a_0/2 + \sum_{k=1}^{\infty} [a_k\cos(2\pi kt/P) + b_k\sin(2\pi kt/P)] \tag{2.1}$$

where, P is the period in seconds, k/P is the frequency in Hertz and, $a_0,a_1\ldots$ and $b_1,b_2\ldots$ are the Fourier coefficients.

The Fourier coefficients are given by,

$$a_k = 2/P \int_{-P/2}^{P/2} x_a(t) \cos(2\pi kt/P)\, dt \quad k=0,1,2... \qquad (2.2)$$

and,

$$b_k = 2/P \int_{-P/2}^{P/2} x_a(t) \sin(2\pi kt/P)\, dt \quad k=1,2... \; . \qquad (2.3)$$

An examination of this series (Eq. 2.1) shows that $x_a(t)$ can be represented by the sum of a direct current (DC) component, $a_0/2$, and an infinite number of harmonically related sinusoids. The amplitude of the sinusoids, given by a_k and b_k above, are a direct indication of the frequency components of $x_a(t)$. Only those frequencies that make up the periodic signal appear in the Fourier series. The Fourier coefficients for the remaining discrete frequencies will be zero.

The series can be converted to a series with complex coefficients by the use of Euler's identities,

$$\cos \varphi = (e^{j\varphi} + e^{-j\varphi})/2 \qquad (2.4)$$

and,

$$\sin \varphi = (e^{j\varphi} - e^{-j\varphi})/2j \; . \qquad (2.5)$$

Substitution of these identities in Equations 2.1, 2.2, and 2.3 results in,

$$x_a(t)= \sum_{k=-\infty}^{\infty} X_a(k)\ e^{j2\pi kt/P} \qquad (2.6)$$

where,

$$X_a(k)= 1/P \int_{-P/2}^{P/2} x_a(t)\ e^{-j2\pi kt/P}\ dt\ . \qquad (2.7)$$

The Fourier Transform

The frequency of the sinusoids in Equations 2.6 and 2.7 is k/P. If P is allowed to be arbitrarily large, the discrete frequencies 1/P, 2/P, 3/P ... become closer and closer. In the limit the discrete frequencies approach a continuous variable f. The Fourier series then becomes the Fourier transform,

$$X_a(f) = \int_{-\infty}^{\infty} x_a(t)\ e^{-j2\pi ft}\ dt \qquad (2.8)$$

where,

$$x_a(t) = \int_{-\infty}^{\infty} X_a(f)\ e^{j2\pi ft}\ df\ . \qquad (2.9)$$

These two integrals have become a function of the continuous variable f. Since the derivation of the Fourier transform implies that P is arbitrarily large, it is only applicable to analysis of non-periodic signals. However, analysis of a signal is practically limited to a finite interval in time, or time window. Windowing is discussed in more detail in Chapter 3.

The Discrete Fourier Transform

The Fourier series given in Equation 2.7 has the limits of integration -P/2 to P/2. Due to the periodicity of $x_a(t)$, the limits of integration can be changed from 0 to P. Equation 2.7 then becomes,

$$X_a(k) = 1/P \int_0^P x_a(t)\, e^{-j2\pi kt/P}\, dt \ . \qquad (2.10)$$

Now consider the case where the signal $x_a(t)$ is sampled every T seconds to form a discrete time signal $x(n) = x_a(nT)$. If in P seconds a total of N samples are gathered, then $P = NT$.

Samples are taken T seconds apart, so T corresponds to the arbitrarily small interval dt. Replacing the integral over t with a sum over nT then gives the equivalent of (2.10) for a discrete time signal as,

$$X(k)= 1/NT \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi knT/NT}\, T \quad k=0,1,2,\ldots N-1 \tag{2.11}$$

or,

$$X(k)= 1/N \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N} \quad k=0,1,2,\ldots N-1 \; . \tag{2.12}$$

Equation 2.12 is the Discrete Fourier Transform of the sequence x(n). The variable n is the data sequence number or sample number and corresponds to a time of nT seconds, where T = P/N seconds. The transform number k gives the number of cycles of x(n) in P seconds which is the frequency bin number, where f = k/P Hz. . For a normalized period P = 1 second, k is the frequency in Hertz.

The quantity $e^{-j(2\pi/N)kn}$ can be viewed as a phasor rotating clockwise in the complex plane. The $2\pi/N$ term is the smallest angular step that the phasor steps through. As n is incremented, the phasor rotates at a frequency determined by k.

The Fast Fourier Transform

An examination of the DFT,

$$X(k)= 1/N \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N} \quad k=0,1,2,\ldots N-1, \quad (2.13)$$

reveals that the direct computation requires N^2 complex multiplications and additions. For small N this is of little consequence. But the number of operations required to compute the direct DFT becomes unacceptable for many applications of Fourier analysis requiring large N. A method to reduce the number of operations to a quantity proportional to Nlog(N) was published as early as 1903, and again in 1942 [2][11]. Figure 2.1 illustrates that the more efficient Nlog(N) method requires significantly fewer operations than the direct method, requiring N^2 operations, even for a small N.

The more efficient computational method was not well known until 1965 when a paper by Cooley and Tukey [3][11] was published. This method and it's other variations are now in wide use and represent a large class of algorithms known as Fast Fourier Transforms.

The radix-2 decimation-in-frequency (DIF) FFT will be described here as a representative FFT algorithm. The algorithm received it's name from the way that the output sequence X(k) is successively divided into smaller and smaller sequences.

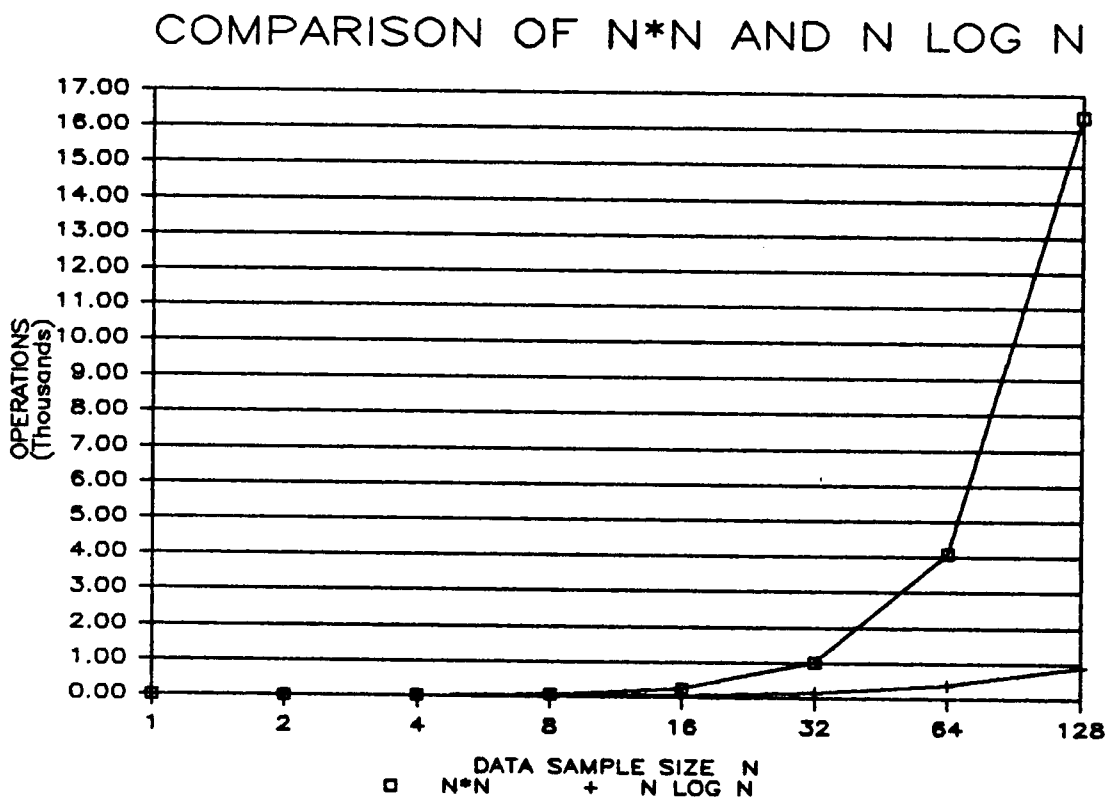


Figure 2.1. Comparison of N^2 and N log N Operations.

Recall the DFT equation,

$$X(k)= 1/N \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N} \quad k=0,1,2,\ldots N-1 \quad . \quad (2.14)$$

Assume that $N = 2^m$ for some m, so the DFT sequence can be divided into a first and a last half. To simplify the notation let,

$$W_N = e^{-j2\pi/N} \quad . \quad (2.15)$$

Thus,

$$X(k) = 1/N \left[\sum_{n=0}^{(N/2)-1} x(n)\, W_N^{kn} + \sum_{n=N/2}^{N-1} x(n)\, W_N^{kn} \right] \quad (2.16)$$

By substituting x(n+N/2) for x(n) in the second term, the summation limits become the same,

$$X(k)=1/N \left[\sum_{n=0}^{(N/2)-1} x(n)\, W_N^{kn} + \sum_{n=0}^{(N/2)-1} x(n+N/2)\, W_N^{k(n+N/2)} \right] \quad (2.17)$$

or,

$$X(k)=1/N \left[\sum_{n=0}^{(N/2)-1} x(n) W_N^{kn} + W_N^{kN/2} \sum_{n=0}^{(N/2)-1} x(n+N/2) W_N^{kn} \right]. \quad (2.18)$$

Noting that ,

$$W^{kN/2} = e^{-j(2\pi/N)kN/2} = (e^{-j\pi})^k = (-1)^k \quad (2.19)$$

and substituting,

$$X(k) = 1/N \left[\sum_{n=0}^{(N/2)-1} x(n)\, W_N^{kn} + (-1)^k \sum_{n=0}^{(N/2)-1} x(n+N/2)\, W_N^{kn} \right]. \quad (2.20)$$

Then,

$$X(k) = 1/N \{ \sum_{n=0}^{(N/2)-1} [x(n) + (-1)^k x(n+N/2)] W_N^{kn} \}. \quad (2.21)$$

Now, separate the even and odd terms of Equation 2.21 by letting 2r be the even k and 2r+1 be the odd k such that,

$$X(2r) = 1/N \sum_{n=0}^{(N/2)-1} x(n) W_N^{2rn} + x(n+N/2) W_N^{2rn} . \quad (2.22)$$

Simplifying, the even terms are,

$$X(2r) = 1/N \sum_{n=0}^{(N/2)-1} [x(n) + x(n+N/2)] W_N^{2rn} . \quad (2.23)$$

The odd terms are,

$$X(2r+1) = 1/N \sum_{n=0}^{(N/2)-1} [x(n) - x(n+N/2)] W_N^{2rn} W_N^{rn} . \quad (2.24)$$

Since,

$$W_N^{2rn} = e^{-j(2\pi/N)2rn} = e^{-j[2\pi/(N/2)]rn} = W_{N/2}^{rn}, \quad (2.25)$$

the even DFT term is in the form of a N/2 point DFT of the sum of the first and last half of the input sequence. The odd DFT term is a N/2 point DFT of the difference of the first and last half of the input sequence, multiplied by W_N^{kn}. The DFT of the original N point sequence can then be computed as two N/2 point DFTs.

At this point, the number of operations to compute the DFT is the same as when using the direct method, but consider that these N/2 point DFTs can likewise be broken into two DFTs each. Since N is a power of two, N/2 is even and each DFT can be broken into an odd and an even part. This process can be repeated until eventually, only a one point DFT remains. For a one point DFT, the DFT is just equal to the data sample. The DFT of the original sequence then, has been computed without actually computing any long sum of products associated with the direct DFT calculation.

Bûtterfly Diagrams

The flow diagram of Figure 2.2 illustrates the structure of the FFT operations for a data sample size of N = 8. Each node on the diagram represents an addition or a subtraction operation. The inputs to an addition or subtraction are on the left and the result on the right. The W_N factors (commonly called twiddle factors in the FFT literature) on the connecting lines represent the multiplication. For example, the first operation is to sum x(n)+x(n+N/2), as in Eq. 2.23, or x(0)+x(4). Then the difference x(n)-x(n+N/2), as in Eq. 2.24, or x(0)-x(4) is computed and multiplied by W_N^0. Add one to the index and compute x(1)+x(5). Likewise compute x(1)-x(5) and multiply by W_N^1. The operations in the diagram are followed from top to bottom, N/2 times resulting in two N/2 point sequences.

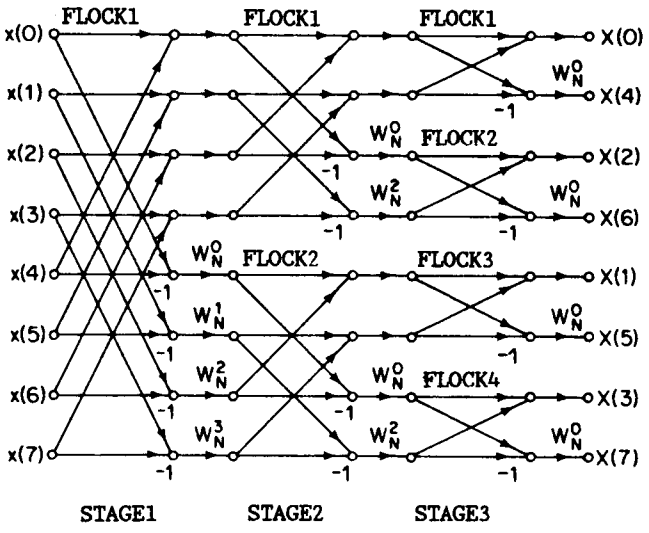


Figure 2.2. Butterfly Diagram, N = 8

This process is repeated, from left to right on the diagram, until the final FFT output is obtained, appearing on the extreme right of the diagram.

The process of separating the odd and even DFT output of each stage of the FFT causes the output to be in scrambled order. The order of the FFT final result can be shown to be in bit reversed order. Bit reversed refers to the order of the bits in the binary index. For example, if N=8, the index can be represented by a three bit binary number 000 to 111. The sixth (binary 110) FFT output is actually the third frequency sample (binary 011) due to bit reversed ordering.

Another means of illustrating the FFT algorithm operations is presented in Figure 2.3. This is a template to calculate the FFT with a spreadsheet program popular with many microcomputer users. In this format, each cell of the spreadsheet either contains a constant or the result of the formula in that particular cell. The input sequence for the FFT is entered in column B with XR representing the real data and XI representing the imaginary part. Upon recalculating, the results ripple through the spreadsheet from column B to column M with the result in column M. Note the bit reversal unscrambling in column L so that the output is in normal order. This spreadsheet form can be compared directly to Figure 2.2 with the cells replacing the nodes.

	A	B	C	D	E	F	G
37	FFT8	DATA	SUM1	W	STAGE 1	SUM2	W
38							
39	XR0	0	(B39+B43)/2		C39	(E39+E41)/2	
40	XR1	1000	(B40+B44)/2		C40	(E40+E42)/2	
41	XR2	0	(B41+B45)/2		C41	(E39-E41)/2	1
42	XR3	-1000	(B42+B46)/2		C42	(E40-E42)/2	0
43	XR4	0	(B39-B43)/2	1	C43*D43+C52*D52	(E43+E45)/2	
44	XR5	1000	(B40-B44)/2	.707106	C44*D44+C53*D53	(E44+E46)/2	
45	XR6	0	(B41-B45)/2	0	C45*D45+C54*D54	(E43-E45)/2	1
46	XR7	-1000	(B42-B46)/2	-.70710	C46*D46+C55*D55	(E44-E46)/2	0
47							
48	XI0	0	(B48+B52)/2		C48	(E48+E50)/2	
49	XI1	0	(B49+B53)/2		C49	(E49+E51)/2	
50	XI2	0	(B50+B54)/2		C50	(E48-E50)/2	0
51	XI3	0	(B51+B55)/2		C51	(E49-E51)/2	1
52	XI4	0	(B48-B52)/2	0	C52*D43-C43*D52	(E52+E54)/2	
53	XI5	0	(B49-B53)/2	.707106	C53*D44-C44*D53	(E53+E55)/2	
54	XI6	0	(B50-B54)/2	1	C54*D45-C45*D54	(E52-E54)/2	0
55	XI7	0	(B51-B55)/2	.707106	C55*D46-C46*D55	(E53-E55)/2	1

	H	I	J	K	L
37	STAGE 2	SUM3	W	STAGE 3	REVERSE
38					
39	F39	(H39+H40)/2		I39	K39
40	F40	(H39-H40)/2	1	I40*J40+I49*J49	K43
41	F41*G41+F50*G50	(H41+H42)/2		I41	K41
42	F42*G42+F51*G51	(H41-H42)/2	1	I42*J42+I51*J51	K45
43	F43	(H43+H44)/2		I43	K40
44	F44	(H43-H44)/2	1	I44*J44+I53*J53	K44
45	F45*G45+F54*G54	(H45+H46)/2		I45	K42
46	F46*G46+F55*G55	(H45-H46)/2	1	I46*J46+I55*J55	K46
47					
48	F48	(H48+H49)/2		I48	K48
49	F49	(H48-H49)/2	0	I49*J40-I40*J49	K52
50	F50*G41-F41*G50	(H50+H51)/2		I50	K50
51	F51*G42-F42*G51	(H50-H51)/2	0	I51*J42-I42*J51	K54
52	F52	(H52+H53)/2		I52	K49
53	F53	(H52-H53)/2	0	I53*J44-I44*J53	K53
54	F54*G45-F45*G54	(H54+H55)/2		I54	K51
55	F55*G46-F46*G55	(H54-H55)/2	0	I55*J46-I46*J55	K55

	M
37	AMPLITUDE/PHASE
38	
39	SQRT(L39*L39+L48*L48)
40	SQRT(L40*L40+L49*L49)
41	SQRT(L41*L41+L50*L50)
42	SQRT(L42*L42+L51*L51)
43	SQRT(L43*L43+L52*L52)
44	SQRT(L44*L44+L53*L53)
45	SQRT(L45*L45+L54*L54)
46	SQRT(L46*L46+L55*L55)
47	
48	ATAN(L48/L39)
49	ATAN(L49/L40)
50	ATAN(L50/L41)
51	ATAN(L51/L42)
52	ATAN(L52/L43)
53	ATAN(L53/L44)
54	ATAN(L54/L45)
55	ATAN(L55/L46)

Figure 2.3. Spreadsheet FFT, N = 8.

Terminology

The resemblance of the sum and difference operation in the flow diagram of Figure 2.2 to butterfly wings has led to use of the term butterfly diagram to describe them. An extension of this terminology is very useful for descriptive variable names when writing FFT computer programs. The process of dividing the DFTs into smaller DFTs in each stage results in a smaller N for each stage. This appears as a smaller butterfly wingspan on the diagram. The name SPAN in this paper will refer to the difference in the x indices from butterfly wing tip-to-tip.

Another term that is useful is FLOCK. Note that on the diagram of Figure 2.2, the butterflys are grouped together, first in four, then two, and then one butterfly in the last stage. The name FLOCK will be used to refer to the grouping of butterflys.

The term STAGE refers to the section of the diagram from left to right containing the butterflys. The algorithm flow is from STAGE to STAGE, from left to right as shown in Figure 2.2.

CHAPTER 3

PROBLEMS IN DFT RESULTS

Introduction

This chapter deals with two of the problems one encounters when using the DFT to perform Fourier analysis of a sampled signal. One problem addressed is aliasing, and the other is leakage due to windowing of the input signal. There are other effects that cause errors, such as those due to using a finite digital word length, and noise, but these are beyond the scope of this work.

The section on aliasing follows the derivation in reference [10] and the section on leakage is based on material from [13].

Aliasing

Consider an analog signal $x_a(t)$ which has the Fourier transform,

$$X_a(j\Omega)= \int_{-\infty}^{\infty} x_a(t)\ e^{-j\Omega t}\ dt\ . \qquad (3.1)$$

and it's inverse,

$$x_a(t)\ =\ 1/2\pi \int_{-\infty}^{\infty} X_a(j\Omega)\ e^{j\Omega t}\ d\Omega\ . \qquad (3.2)$$

The sequence $x(n) = x_a(nT)$ can be derived by periodic sampling, where T is the sampling period. From Eq. 3.2,

$$x(n) = 1/2\pi \int_{-\infty}^{\infty} X_a(j\Omega)\, e^{j\Omega nT}\, d\Omega \ . \qquad (3.3)$$

By breaking the integral into intervals of length $2\pi/T$ and summing them,

$$x(n) = 1/2\pi \sum_{r=-\infty}^{\infty} \int_{(2r-1)\pi/T}^{(2r+1)\pi/T} X_a(j\Omega)\, e^{j\Omega nT}\, d\Omega \ . \qquad (3.4)$$

By a change of variables, the integral range can be changed to the interval $-\pi/T$ to π/T giving,

$$x(n) = 1/2\pi \sum_{r=-\infty}^{\infty} \int_{-\pi/T}^{\pi/T} X_a(j\Omega + j2r\pi/T)\, e^{j\Omega nT} e^{j2\pi rn}\, d\Omega \ . \qquad (3.5)$$

By noting that $e^{j2\pi rn} = 1$ for all integer values of r and n, and reversing the order of the sum and integral,

$$x(n) = 1/2\pi \int_{-\pi/T}^{\pi/T} \sum_{r=-\infty}^{\infty} X_a(j\Omega + j2\pi r/T)\, e^{j\Omega nT}\, d\Omega \ . \qquad (3.6)$$

Make the substitution $\Omega = w/T$ in Eq. 3.6 giving,

$$x(n) = 1/2\pi \int_{-\pi}^{\pi} 1/T \sum_{r=-\infty}^{\infty} X_a(jw/T + j2\pi r/T)\, e^{jwn}\, dw. \qquad (3.7)$$

This can be related to the discrete time Fourier transform representation of x(n) given by,

$$x(n) = 1/2\pi \int_{-\pi}^{\pi} X(e^{jw})\, e^{jwn}\, dw. \qquad (3.8)$$

Equations 3.7 and 3.8 are in the same form, showing that,

$$X(e^{jw}) = 1/T \sum_{r=-\infty}^{\infty} X_a(jw/T + j2\pi r/T). \qquad (3.9)$$

Equation 3.9 shows the relationship of the Fourier transform of a discrete time signal obtained by sampling to the Fourier transform of the continuous time signal. If the Fourier transform of a continuous time signal $X_a(j\Omega)$ is as shown in Figure 3.1a, then the Fourier transform of the discrete time signal, $X(e^{jw})$, will be as shown in Figure 3.1b, if $\Omega_0 T/2 > \pi$. Note that if the sampling period is too large the periodic repetitions of the continuous time transform overlap. The overlap means that the lower frequencies impersonate higher ones and vice versa. This effect is called aliasing. It is undesirable since it is difficult to sort out the aliases from the signal. From Figure 3.1c, it can be seen that if $\Omega_0 T/2 < \pi$ there is no overlap and aliasing does not occur. This means that the sample frequency should be at least twice as high as the highest frequency in the analog signal. This sampling rate is called the Nyquist rate.

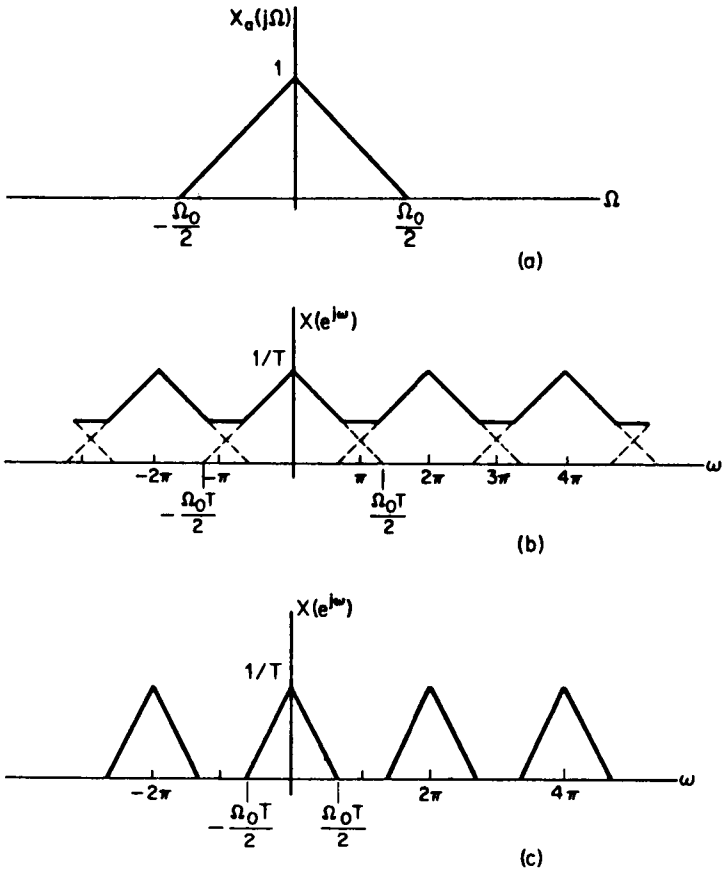


Figure 3.1. Illustration of the Sampling Theorem

Adapted From: Oppenhiem, A. V., and R. W. Schafer. Digital Signal Processing Englewood Cliffs, N. J. : Prentice-Hall, Inc., 1975.

Windowing and Leakage

In the process of performing a DFT of a signal, the signal is restricted in time to a finite interval. This is equivalent to multiplying the signal samples by a time window. This multiplication in the time domain corresponds to a convolution of the spectra of the signal and window in the frequency domain [6].

Consider a cosine signal and a rectangular window. The rectangular window, $w_r(t)$, has amplitude one, from t=-T/2 to T/2, and zero elsewhere. This rectangular pulse has the Fourier transform,

$$W_r(f) = \int_{-T/2}^{T/2} e^{-j2\pi ft}\, dt = \frac{T\sin(\pi fT)}{T\pi f} \qquad (3.10)$$

A cosine signal and its frequency spectrum are shown in Figure 3.2a. A rectangular window and its (sinx)/x shaped frequency spectrum are shown Figure 3.2b.

The windowed cosine signal and it's transform, s(f)*w(f) (* indicates convolution), is shown in Figure 3.2c. The windowed signal is sampled by multiplying it by an impulse train as in Figure 3.2d.

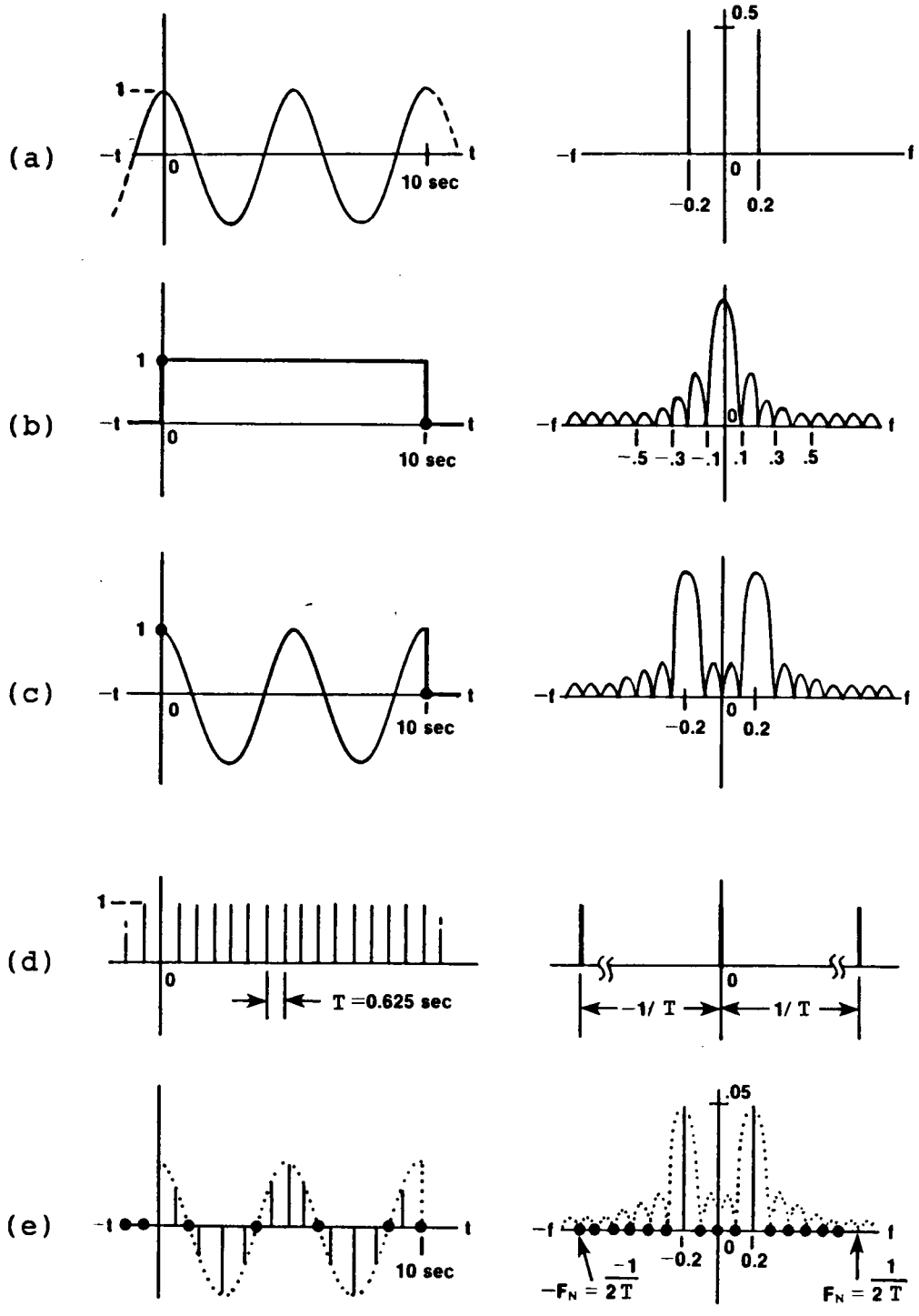


Figure 3.2. DFT of a Cosine Wave, No Leakage.

Adapted From: Ramirez, R. W. The FFT: Fundamentals and Concepts. Beaverton, Oregon: Tektronix, Inc., 1975.

A convolution of the sample train spectrum and the windowed spectrum is shown in Figure 3.2e by the dotted curve. The value of this waveform at the frequency bins are the DFT output values. Herein lies the possible problem.

Figure 3.2 and 3.3 illustrates that the relationship of the window length to the number of cycles of the signal contained in the window can have an effect on the DFT output. If, as in Figure 3.2, the period of the signal is harmonically related to the window length, the convolution results in only two non-zero frequency bins located at the signal frequency. This happens because the frequency bins fall on the zeros of the signal's transform except at the main lobes of the signal spectrum. In this case the DFT output is the actual spectrum of the original signal.

A case is illustrated in Figure 3.3 where the window contains 2.5 cycles of the cosine signal. The window length and the sample frequency is the same as in Figure 3.2. The result of this convolution is shown in Figure 3.3e. It results in a DFT in which the main signal frequency component is divided into components with reduced amplitude on each side of the signal frequency. There are also smaller components away from the signal frequency because the frequency bins do not coincide with the (sin x)/x zeros. This effect is called leakage and can cause the DFT to appear quite different than the actual signal spectrum.

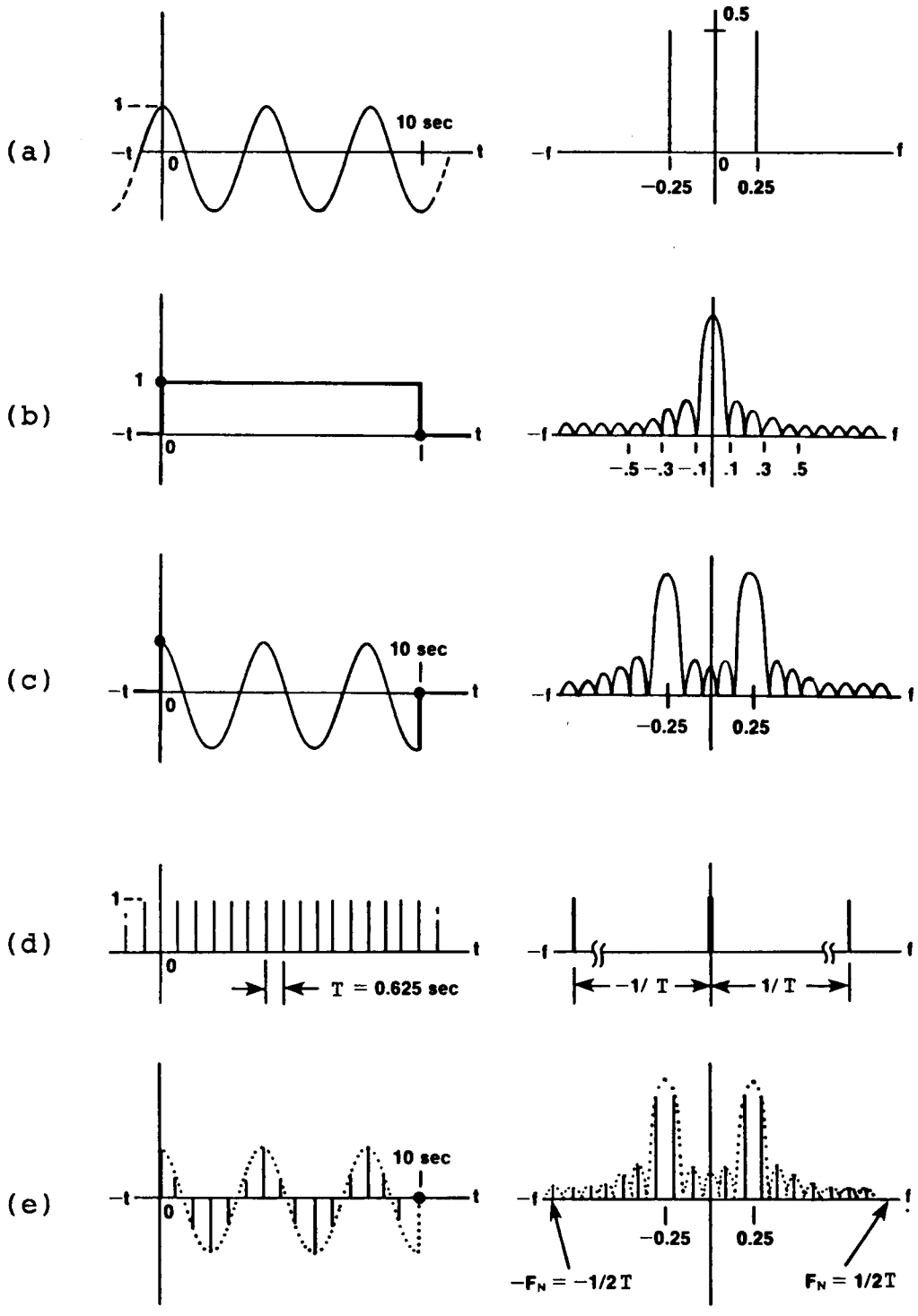


Figure 3.3. DFT of a Cosine Wave, With Leakage.

Adapted From: Ramirez, R. W. The FFT: Fundamentals and Concepts. Beaverton, Oregon: Tektronix, Inc., 1975.

There are techniques used in practice that minimize the leakage. By using a different window shape, other than rectangular, the (sin x)/x shape of the window signal spectrum can be modified to minimize the undesirable effects on the DFT output. A variety of window signals, such as the Hanning and Hamming, are widely used in spectrum analysis for this purpose [12].

CHAPTER 4

COMPUTATION OF THE DFT

Introduction

The previous chapters presented the theory behind DFTs and the possible pitfalls of using them blindly without regard for aliasing, leakage, and other effects. This chapter will present the methods of performing the DFT on a digital computer. The BASIC computer language was chosen to introduce the structure of the DFT algorithms for several reasons. First of all, BASIC is probably the most universal computer language in current use, being available for virtually all computers. Secondly, the commands are easy to follow, even for inexperienced programmers.

Direct Calculation of the DFT

Recall that the DFT equation is,

$$X(k) = 1/N \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi kn/N}, \quad k=0,1,2,..N-1 \quad . \tag{4.1}$$

In general the data sample sequence is complex, containing both amplitude and phase information. The multiplication of the data sequence and the complex exponential is accomplished by separation of the real and imaginary parts as follows. Let,

$$W_N = e^{-j2\pi/N} . \tag{4.2}$$

For the DFT the real part of $(x\ W_N^{kn})$ is,

$$\begin{aligned} \mathrm{Re}\ (x\ W_N^{kn}) &= \mathrm{Re}(x)\ \mathrm{Re}(W_N^{kn}) \\ &\quad + \mathrm{Im}(x)\ \mathrm{Im}(W_N^{kn}) \end{aligned} \tag{4.3}$$

and the imaginary part is,

$$\begin{aligned} \mathrm{Im}\ (x\ W_N^{kn}) &= \mathrm{Im}(x)\ \mathrm{Re}(W_N^{kn}) \\ &\quad + \mathrm{Re}(x)\ \mathrm{Im}(W_N^{kn}), \end{aligned} \tag{4.4}$$

where,

$$\mathrm{Re}(W_N^{kn}) = \mathrm{WREAL} = \cos(2\pi kn/N),$$

and, (4.5)

$$\mathrm{Im}(W_N^{kn}) = -\mathrm{WIMAG} = -\sin(2\pi kn/N).$$

An algorithm to compute the DFT using these identities is shown in Figure 4.1. An inner loop, varying with n, contains the sum-of-products of the data and twiddle factors. An outer loop, varying in k, determines when all of the DFT terms have been computed.

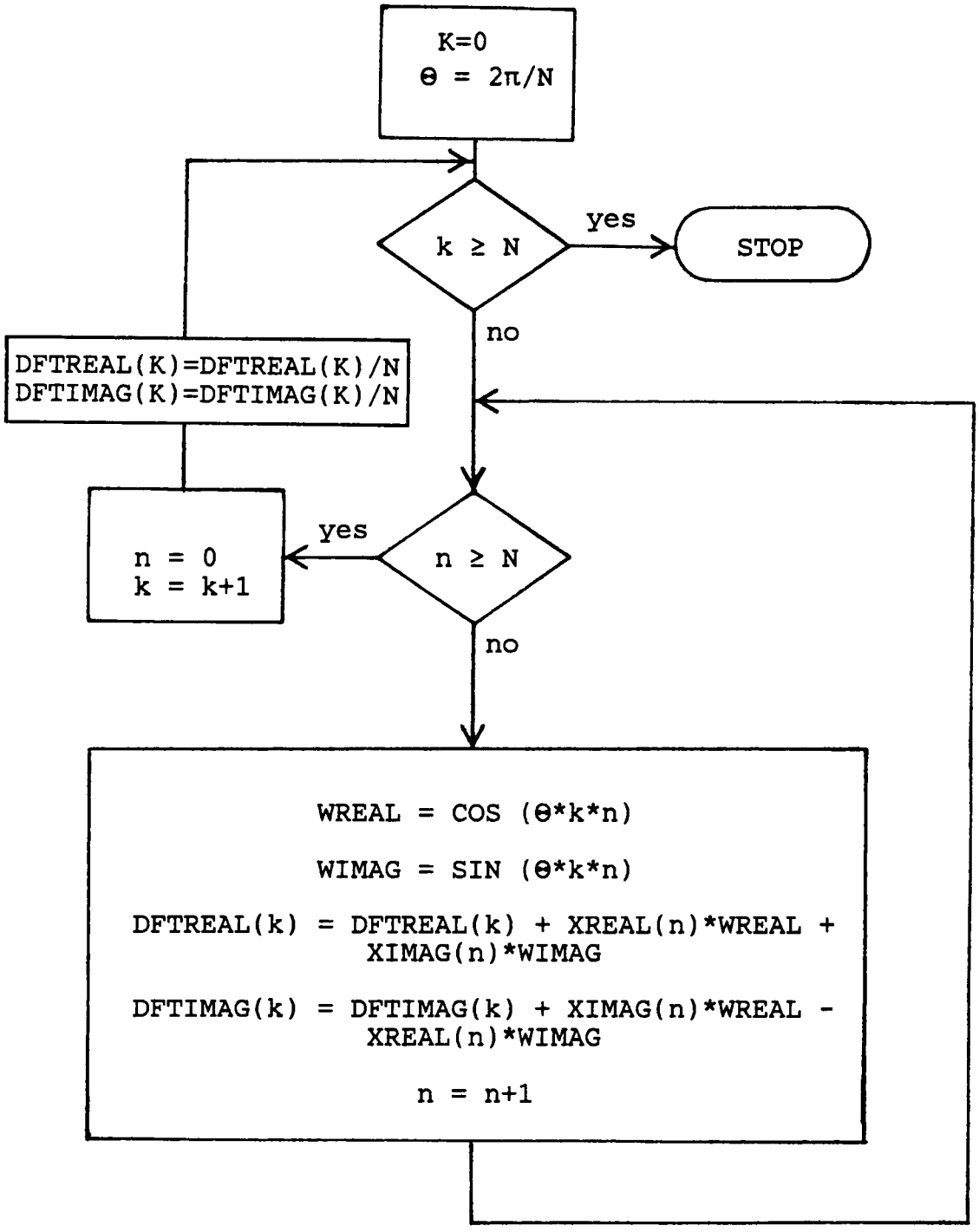


Figure 4.1. Direct DFT Algorithm

A BASIC program to implement this algorithm is contained in Appendix A. The first section of the program, lines 10-30, dimension arrays and define the integer and double precision variables. The second section, lines 40-240, defines pi, prompts the user for filenames for input data and output data, reads the input data file, and sends the input data and header information to the printer, and to the output file.

The input data file consists of N+1 words in the hexadecimal format with one word per record. The first word contains N, followed by N words of real data, then by N words of imaginary data.

The program lines from 250-440 perform the DFT calculations. The outer while/wend loop increments and tests k to determine when to end the program. The inner while/wend loop performs the sum-of-products of the data samples and twiddle factors as described in Eqs. 4.3 and 4.4. If the data is real (XI=0), the terms XI*WI in line 350, and XI*WR in line 360 can be eliminated. Also, due to the folding property of the DFT [4], only N/2 terms of the DFT need to be calculated for real data ($X(N-k)=X^{*}(k)$). For complex data, the term NSAM/2 in line 290 should be changed to NSAM. Integer arithmetic is used. The DFT output is stored in the output file.

Calculation of the FFT

Calculation of the FFT with a digital computer can be accomplished by implementing the butterfly diagram discussed in Chapter 2. The terms STAGE, FLOCK, SPAN and twiddle factor were introduced in Chapter 2 and will be used here to describe the FFT algorithm.

Referring to Figure 2.2, note that there are three stages in the diagram. The first stage contains one flock of four (N/2) butterflies, and the span is four (N/2). The twiddle factor exponent increments by one for each succeeding butterfly (k=1).

Stage two contains two flocks of two (N/4) butterflies, with a span of two (N/4). The twiddle factor exponent increments by two (k=2) each succeeding butterfly.

Stage three contains four (N/2) flocks of one butterfly and the twiddle factor exponent does not increment because there is only one butterfly in each flock.

The STAGE, FLOCK, and SPAN variables are used as loop variables in the FFT algorithm to determine when to go to the next flock or span as the butterfly operations are carried out. The progression of the variable values as the FFT is performed, from left to right on the butterfly diagram, is shown in Table 4.1.

Table 4.1. Variable Values for FFT Algorithm

STAGE	k	SPAN	NUMBER OF FLOCKS
1	1	4	1
2	2	2	2
3	3	1	4

A flow chart for the FFT algorithm is shown in Figure 4.2. The TOP and BOT labels refer to the index of the top and bottom node for the butterfly being processed. Note that the butterfly can be computed in-place, with the result of the sum and product operation replacing the original data sample. This method saves computer memory.

The actual butterfly calculation is done much the same way as for the DFT discussed in the previous section. The complex values are separated into real and imaginary parts and the operations are performed. The complex final results are usually converted to a magnitude and phase.

A BASIC FFT Program

Appendix B contains a structured program which implements the decimation-in-frequency FFT algorithm in BASIC. This program uses the usual BASIC floating point arithmetic. Rather than reading an input data file, the program prompts the user for signal information and generates it's own data sample sequence. There is a screen plot routine to show the FFT results.

The first section, up to line 2000, is initialization of variables and dimensioning of data arrays. This program is dimensioned for a complex data sample size of 1024 samples. The next section, lines 2000-3000, is the main routine that sets up the loop variables and calls the necessary subroutines.

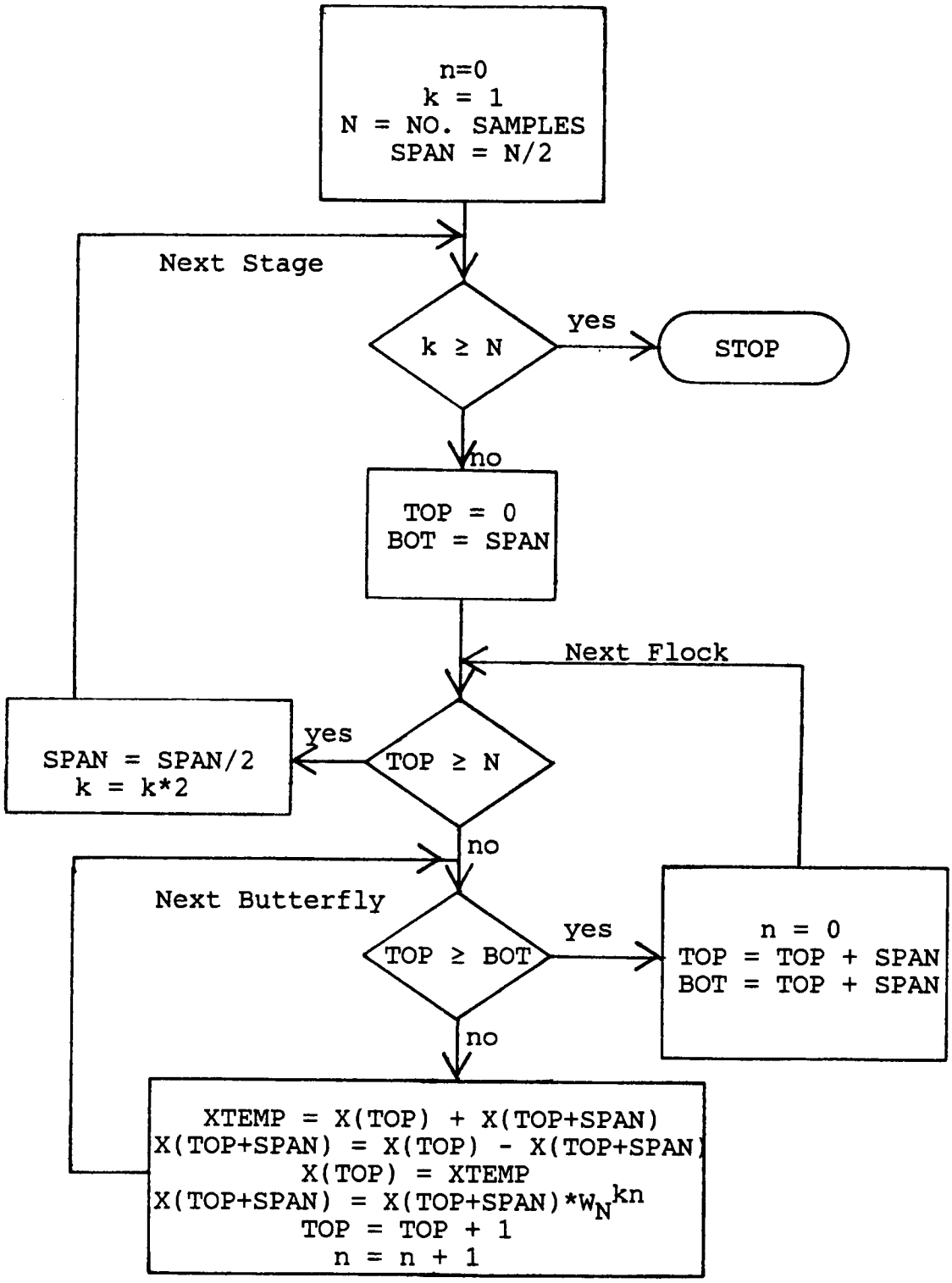


Figure 4.2 FFT Algorithm

The first subroutine, at line 4000, generates a sequence of data samples, XR(J) and XI(J), from a sampled cosine function. Then a subroutine is called, at line 7000, to select a window. Provisions for a Hanning, Hamming or a rectangular window are included to minimize the leakage effect if desired. The data sequence is multiplied by the selected window. The quantity DELTA is calculated. This is the smallest step size for the phasor $e^{-j(2\pi/N)kn}$. SPAN is initialized to N/2. The subroutine at line 8000 calculates the twiddle factor values for each of N/2 points, from 0 to π radians. These values are stored in arrays WRE (real) and WIM (imaginary).

The actual loop that calculates the FFT is then entered. First the top index, NTIP1, for the first data value in butterfly one are initialized. The index for the bottom data value in this butterfly is NTIP1+SPAN. The variable NTIP2=NTIP1+SPAN points to the bottom data value in the flock.

The subroutine at line 3000 does the butterfly sums and products of the complex numbers. After the first butterfly operation, the variable NTIP1 is incremented to go to the next one. This loop repeats until NTIP1 is equal to NTIP2, indicating that all the butterfly operations in this flock have been completed.

The program flow then returns to the main routine. Here SPAN is added to NTIP1, and NTIP2 is replaced by NTIP1 plus SPAN to step to the next flock of butterflies. If after the sum, NTIP1 is equal to ND, the number of data samples, there are no more butterflies in the present stage. In this case SPAN is divided by two, K is multiplied by two, NTIP1 is zeroed, and NTIP2 is equated to SPAN. This places the process at the beginning of the next stage.

If K = N at this point, there are no more stages left and the FFT is completed. The routine at line 6000 is then called to unscramble the order of the output.

The bit reverse unscrambling routine can be explained readily by an example. The two indices, I and M, are used as the normal and bit reverse counters, respectively. If N=16 then a four bit word is necessary for the indices. The index I is incremented from zero to 15. For each value it assumes, each bit of I is tested from bit zero to bit four. If true, the opposite bit is set in index M , from bit four to bit zero. When all four bits are tested, the index M is in bit reversed order from I. The indices are then used to swap X(I) and X(M). This is done only if I is greater than M (so that the swap is made only once). This leaves the FFT output in normal order.

The last routine sends the data to a printer and does a screen plot of the magnitude and phase.

BASIC FFT Results

The following signals were sampled (N=128) and used as the data sample sequence to the BASIC FFT program of Appendix B to demonstrate that the program works correctly:

1. a 2000 Hz. cosine wave, with a sample frequency of 16000 Hz.

2. a 2000 Hz. cosine wave, with a sample frequency of 15515 Hz.

3. a 2000 Hz. sine wave, with a sample frequency of 16000 Hz.

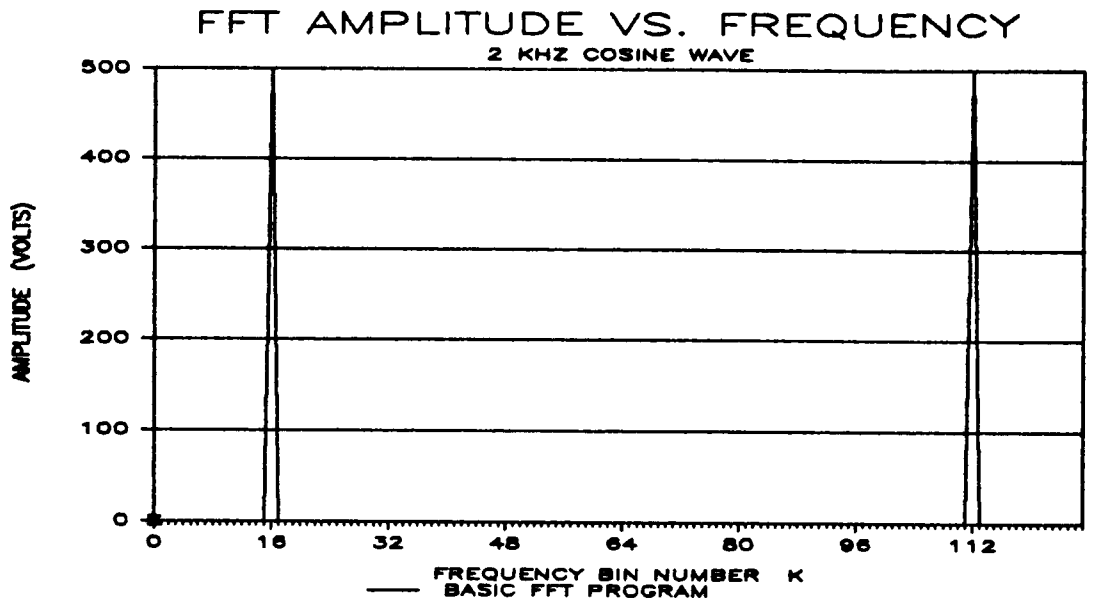
4. a 2000 Hz. square wave, with a sample frequency of 16000 Hz. .

The data sample size was N = 128 in all cases, and the amplitude of the signal is 1000 with units of volts assumed . Case 1 has an integral number of cycles of the signal within the window while case 2 does not. The square wave of case 4 is an even function.

Figure 4.3 shows the results of the BASIC FFT program for case 1. The divisions on the K axis represent,

$$F_S/N = 16000/128 = 125 \text{ Hz.}$$

The peaks of amplitude 500 at k=16 and k=112 are the 2000 Hz. frequency components of the cosine wave. The phase plot correctly shows zero phase for these components.



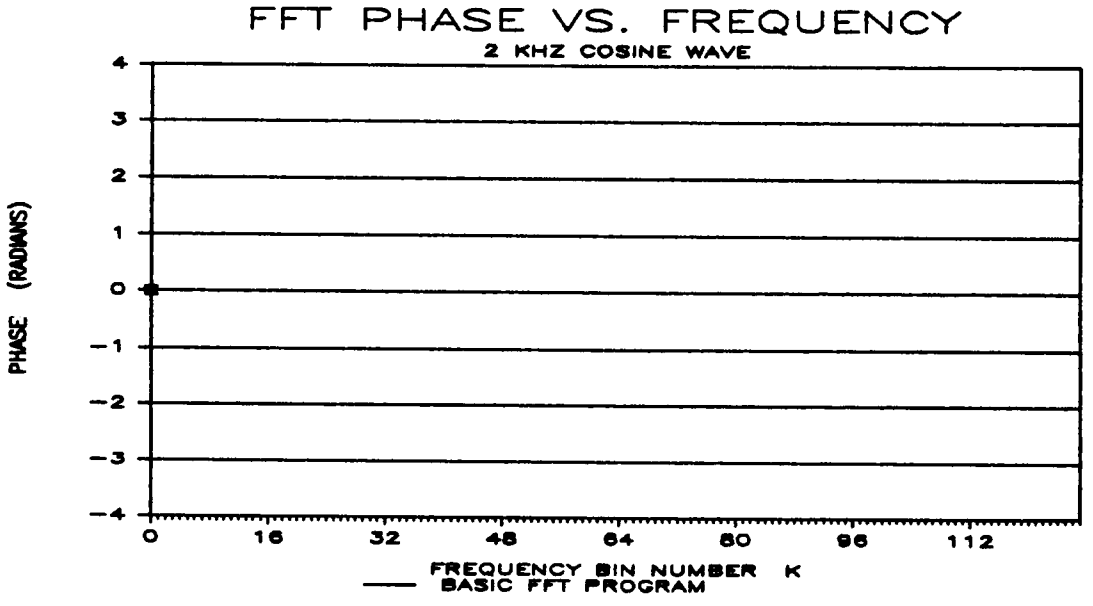
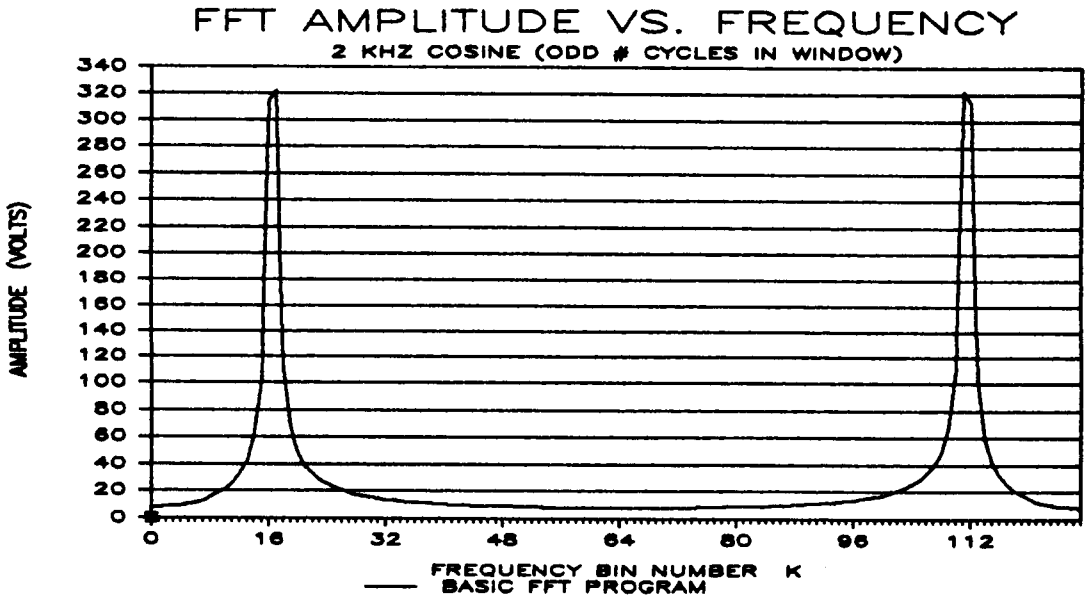


Figure 4.3. FFT of a 2000 Hz. Cosine Wave.

Figure 4.4 shows the FFT results for case 2. This is the same signal with a different window length. The leakage error discussed in Chapter 3 can be seen as a widening of the fundamental frequency plot. This implies the presence of smaller amplitude frequency components at non-zero phase which really do not exist.

Figure 4.5 shows the FFT for the sine wave of case 3. Here the 2000 Hz. frequency components have a 90 degree phase shift (1.57 radians).

Figure 4.6 shows the frequency components of the even square wave. The DC component, fundamental, and even harmonics are present. The phase information appears mostly random due to the arctan(Im(FFT)/Re(FFT)) function. When the Im(FFT) is non-zero, however small, and Re(FFT) is very small the argument of the arctan function can be large.



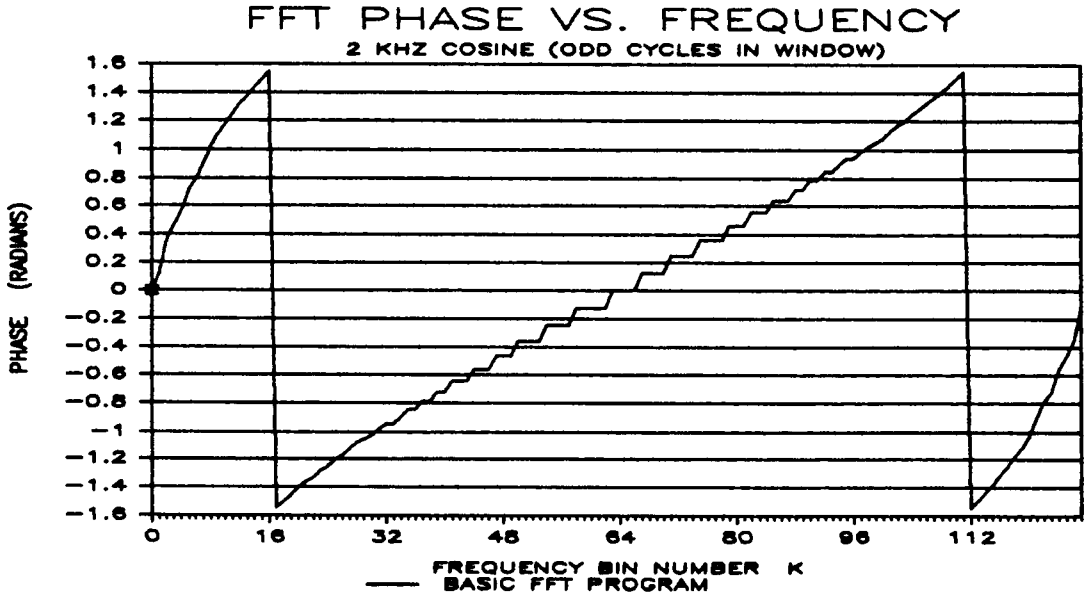


Figure 4.4. FFT of a 2000 Hz. Cosine Wave With Non-Integer Number of Cycles in the Window.

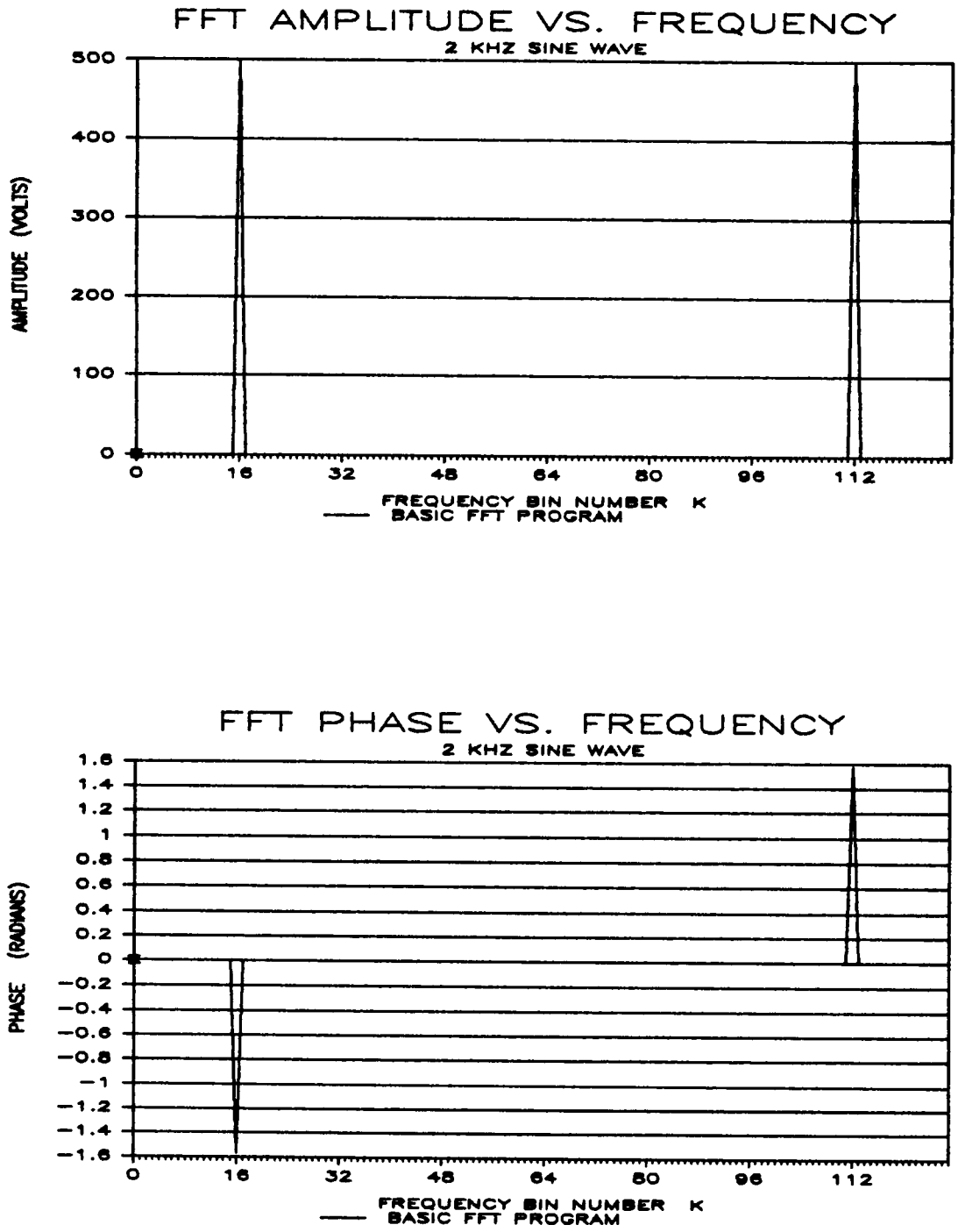


Figure 4.5. FFT of a 2000 Hz. Sine Wave.

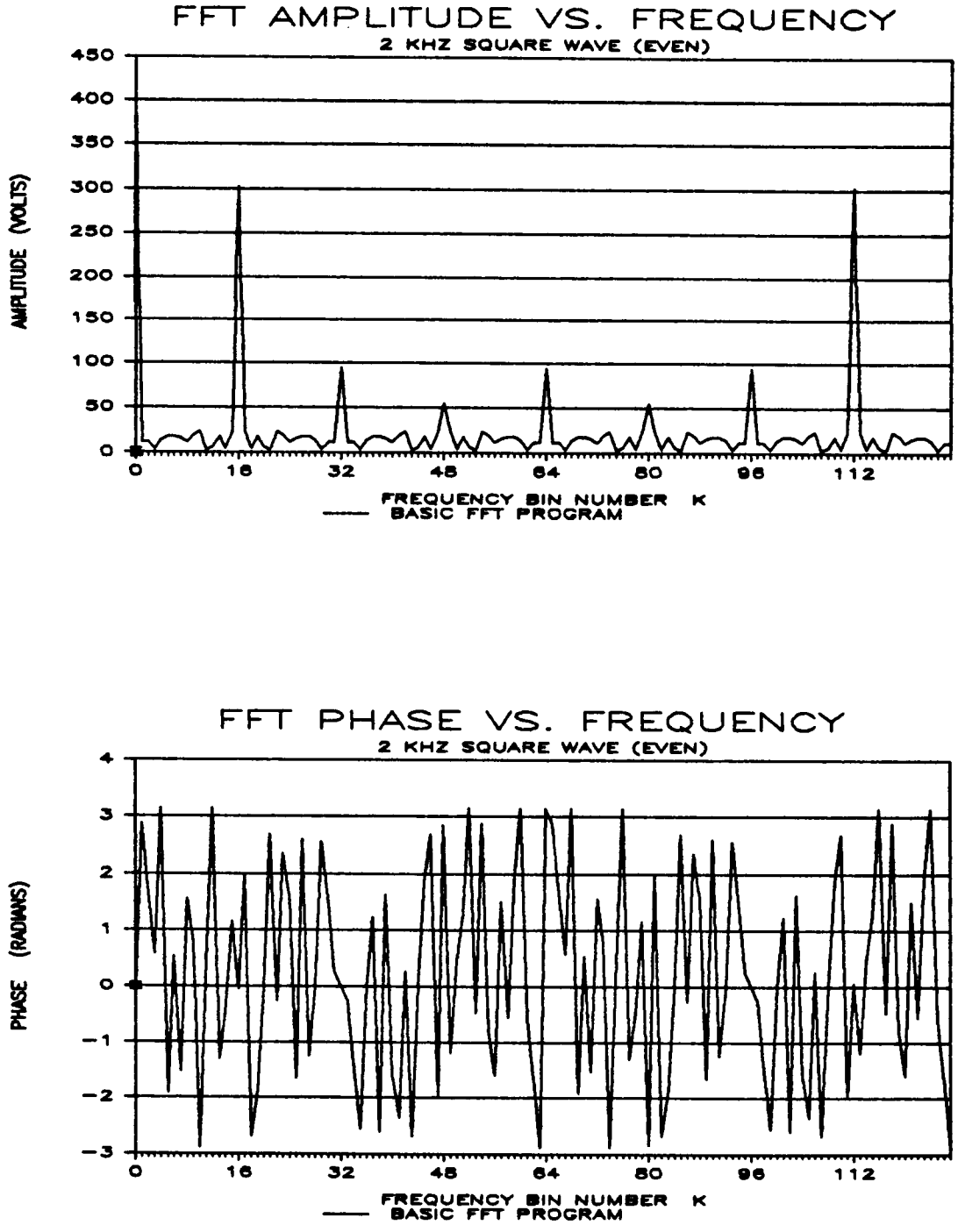


Figure 4.6. FFT of a 2000 Hz. Square Wave (Even).

CHAPTER 5

THE TMS32020 MICROPROCESSOR

Introduction

The TMS32020 is the second microprocessor in a family of chips from Texas Instruments, Inc., designed for digital signal processing (DSP) and other high speed numeric intensive applications. The first chip in this family, the TMS32010, has gained wide acceptance and is one of the most popular DSP microprocessors in current use. The TMS32020 has features which make it potentially more attractive for DFT use than it's predecessor. This chapter contains a brief summary of these hardware features. Reference [13] is recommended for detailed information.

This chapter will discuss the TMS32020 architecture as well as the assembler, linker, and simulator software used to develop the FFT routines in Chapter 6. All of the 32020 DFT software for this study was developed using a cross assembler, linker, and simulator on a host computer.

TMS32020 Architecture

The TMS32020 architecture is based upon the TMS32010, having a single accumulator, modified Harvard-type design. Program and data memory occupy separate address spaces, allowing overlap of instruction fetch and execution.

Figure 5.1 is a functional block diagram of the TMS32020 which illustrates the bus structure and internal design features. The major components are: 1) the Arithmetic Logic Unit (ALU) with a 32 bit accumulator, 2) the Multiplier, 3) the Auxilliary Registers (AR), and 4) the Auxiliary Register Arithmetic Unit (ARAU).

The ALU performs arithmetic and logical instructions, most of which execute in a single clock cycle. Referring to Figure 5.1, one input to the ALU is always from the accumulator. The other input is either from the Product Register (PR), or the input scaling shifter which contains data fetched from data Random Access Memory (RAM) on the data bus. The ALU can also test bits in a word of data memory. The result of the ALU operations is stored in the 32 bit Accumulator, which is split into two 16 bit segments for storage into data memory. A shifter is available at the output of the accumulator.

The Multiplier computes a 32 bit product in a single machine cycle. Normally one operand is contained in the Temporary Register (TR) and the second operand is from data memory. The Multiply and Accumulate (MAC and MACD) instructions, though, access one operand from on-chip program memory Block B0, and the second operand from on-chip data memory Block B1 or B2. The 32 bit PR holds the product. There is a shifter at the output of the PR.

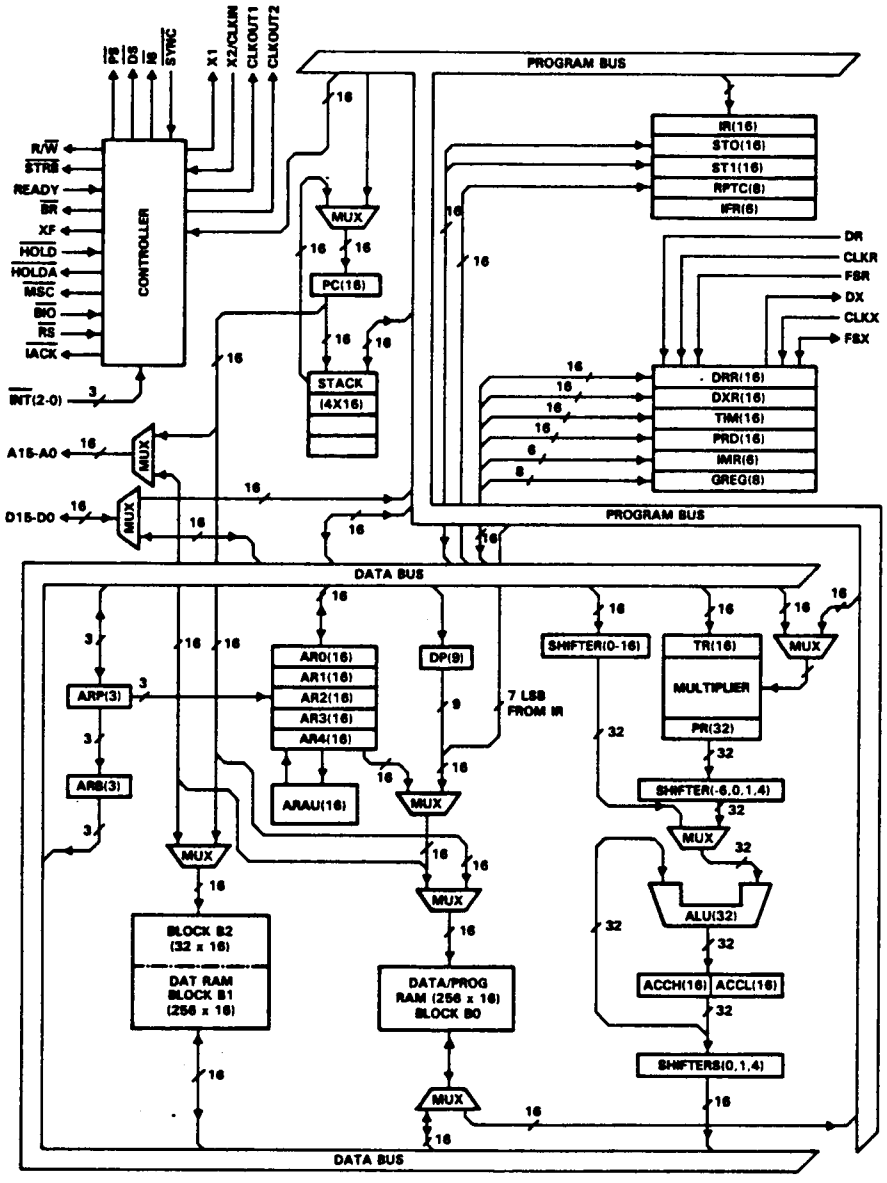


Figure 5.1. TMS32020 Functional Block Diagram

Five Auxiliary Registers provide the TMS32020 with some powerful indirect addressing modes (indirect implies that the instruction operand is an AR that holds a pointer to a data memory address). A specific register is selected to be the current AR by loading the Auxiliary Register Pointer (ARP) with zero through four for AR0 through AR4.

The ARAU can perform arithmetic operations on the Auxiliary Registers. The operations are auto-incrementing, auto-decrementing, and indexing by adding or subtracting the contents of AR0 to the current AR. All operations of the ARAU are performed in the same machine cycle as the original instruction.

Cross Assembler

The 32020 cross assembler translates the instruction mnemonics and assembler directives contained in the source code into object code. The cross assembler is invoked by typing the file name from the Disk Operating System (DOS) when using the personal computer based version of the development software. The user is prompted for a source filename, an object code filename for creation, and a listing filename. After the appropriate filenames are entered, the assembler begins processing the source code statements and creates the object and listing files. Any error messages are displayed on the screen and are written to the list file.

A partial list of the TMS32020 instruction mnemonics, ones which are used in programs presented later, is contained in Table 5.1. The syntax for the source code statements is,

<label> <mnemonic> <operand> <comment> .

Any of the four fields may be optional, (indicated by the brackets) depending upon the instruction. The label field begins in column one. At least one blank must separate fields in the source statement.

Several assembler directives affect the program location counter. The AORG directive allows the assembler to establish the absolute memory address for a block of code. PSEG (program segment) and DSEG (data segment) signify that a section of code is relocatable and that the memory address will be established at link time.

The 32020 instruction set provides for three types of memory addressing modes: 1) direct, 2) indirect, and 3) immediate. The direct addressing mode uses the Data Memory Page Pointer (DP) which is nine bits long, concatenated with the lower seven bits of the instruction word to form the Data Memory Address (DMA). The DP register points to one of 512 possible pages of memory, each page containing 128 words. An example of an instruction using direct addressing is,

```
ADD     20,3 .
```

Table 5.1. TMS32020 Instructions

Mnemonic	Definition
ABS	Absolute value of accumulator
ADD	Add to accumulator with shift
LAC	Load accululator with shift
NEG	Negate accumulator
SACH	Store high accumulator with shift
SACL	Store low accumulator with shift
SFR	Shift accumulator right
SUB	Subtract from accumulator with shift
SUBC	Conditional subtract
ZAC	Zero accumulator
CMPR	Compare AR with AR0
LAR	Load auxiliary register
LRLK	Load auxiliary register long immediate
MAR	Modify auxiliary register
SAR	Store auxiliary register
APAC	Add P register to accumulator
LT	Load T register
MAC	Multiply and accumulate
MPY	Multiply with T register
B	Branch unconditionally
BANZ	Branch on auxiliary register not zero
BBZ	Branch if TC bit = 0
BZ	Branch if accumulator = 0
BIT	Test bit
CNFD	Configure block as data memory
CNFP	Configure block as program memory
POP	Pop top of stack to accumulator low
PUSH	Push low accumulator to stack
RPT	Repeat instruction
SSXM	Set sign extension mode
IN	Input data from port
OUT	Output data to port
TBLR	Table read
TBLW	Table write

This instruction adds to the accumulator the contents of the DMA left shifted three bits. The DMA in this case is at an offset of 20 into the memory page pointed to by the DP register.

Indirect addressing uses the five Auxiliary Registers, AR0 through AR4, as pointers to memory. Any location in the 64K of data memory space can be accessed using the 16 bit pointer contained in the current AR. The DP is not used in indirect addressing.

An example of indirect addressing is,

```
ADD     *,8.
```

The contents of the DMA pointed to by the current AR (*) is left-shifted eight bits and added to the accumulator.

A valuable feature of the indirect addressing mode is the use of the Auxiliary Register Arithmetic Unit. The ARAU can auto-increment or auto-decrement the current AR. An example is,

```
ADD     *+,8.
```

This is the same instruction as the previous one except that the AR is now automatically incremented after execution of the instruction (*- for auto-decrementing).

The ARAU can also add or subtract the contents of AR0 to the current AR for indirect indexed addressing. For example, consider the instruction,

```
ADD     *0+,8.
```

Again, the instruction is the same but this time the contents of AR0 is added to the current AR after execution. (*0- indicates AR0 is subtracted from the current AR).

In immediate addressing the instruction contains the value of the immediate operand. An example is,

```
ADD     16384,2.
```

This instruction adds to the accumulator the value 16384 shifted left two bits, effectively adding 65536 to the accumulator.

Linker

The Link Editor combines separate object modules to form a single linked module that can be installed and executed. The linker can position program and data segments to boundaries for RAM/ROM partitioning.
If the PSEG and DSEG tags are present in the included object modules, the Link Editor organizes segments from each module into program and data segments. All of the program segments are first, followed by the the data segments from all the included modules.

To use the Link Editor, a link control file must be created which contains the link command lines. A link control file for the programs of this study contains a TASK statement to name the program to be linked, the PSEG and DSEG statements to locate the segments in memory, INCLUDE statements to name the modules to be linked to form the program, and an END statement.

The Link Editor is invoked by typing the file name. The user is prompted for the control file name and the name of the listing file.

TMS32020 Simulator

The TMS32020 Simulator is a software product which, when run on a host computer, simulates the 32020 microcomputer hardware operation. The Simulator is designed to be a software development tool to speed the process of debugging code written for the 32020.

The Simulator can be loaded with a user's executable file and provides all of the facilities needed to monitor and debug the user program.

The personal computer version of the Simulator is invoked by typing the filename in the personal computer version. The user is first prompted for the memory configuration. This configures memory block zero as data RAM or program ROM. The Simulator then waits for the user to issue a command.

The first command is usually the L command to load the 32020 program file into the simulator for debugging or running. Breakpoints can be set with the BIAQ command to halt execution of the TMS32020 code at any place in the program. The program is started with the R command. When a breakpoint is reached, memory can be examined with the RAMH command or registers examined with the ST command.

A particularly useful feature is the machine cycle counter. When a timing analysis is needed for checking speed of program execution, the cycle counter can be zeroed with the Z command at the start of a program. At the end of the program the counter will contain the number of machine cycles the program took. One machine cycle takes 200 nanoseconds on the current version of the chip, so the execution time can be determined.

CHAPTER 6

THE DFT ON THE TMS32020

The Direct DFT

The direct DFT computation requires a long sum of products of two quantities, the data sample sequence and the twiddle factor coefficients. A fast computation of the DFT then, requires both a fast multiply and accumulate and a fast method of getting the twiddle factor coefficients. The TMS32020 has the capability of performing these tasks very efficiently.

The 32020 instruction set includes a multiply and accumulate (MAC) instruction that can execute in one machine cycle under certain conditions. In order to use this instruction, both operands must be in on-chip memory. One value must be in data RAM block B1 or B2. The other value must be in program memory block B0 with block B0 configured to be program memory (memory configuration is software controlled by the CNFP/CNFD instructions).

The MAC instruction must also be executed in the repeat mode to run in a single instruction cycle. The repeat instruction (RPT) causes the instruction which follows it to repeat the number of times specified in the RPT operand.

In this pipeline mode the MAC program memory address (PMA) will be automatically incremented each time the MAC is executed. The DMA can be incremented each cycle by using the indirect addressing, auto-increment mode. The syntax for the MAC instruction in the indirect address mode is,

```
MAC     <PMA>,*+,
```

where PMA is the location of one table value to be multiplied and the *+ symbols indicate that the current AR is a pointer to the other table value. The + causes the contents of the AR to be incremented after execution, pointing to the next table value.

For example, Figure 6.1 is a section of code to perform a sum of products of two tables stored in memory. This sequence of instructions clears the accumulator and the product (PR) register. It then loads AR1 with a pointer to a data table. The repeat instruction causes the MAC to repeat 100 times, multiplying the corresponding elements of table one by table two and summing the products. Since the last product is left in the P register, it is added to the accumulator by the APAC instruction.

This algorithm forms the basis of a direct DFT program on the 32020. It is the most efficient way to perform the sum of products required and it is fast.

```
ZAC     0           ZERO ACCUMULATOR
MPYK    0           CLEAR P REGISTER
LARP    1           ARP TO AR1
LRLK    1,DATA      AR1 POINTS TO DATA TABLE
RPTK    99          REPEAT 100 TIMES
MAC     COEFF,*+    MULTIPLY AND ACCUMULATE
APAC                ADD PR TO ACCUMULATOR
```

Figure 6.1. Sum of Products Using MAC

This routine will run in 109 machine cycles per DFT coefficient on the 32020 with a table length of 100. In execution time this is equivalent to 21.8 microseconds with a 200 nanosecond instruction cycle.

The advantage of using the MAC instruction over a separate multiply, add, and end of loop test to do the same task can easily be demonstrated. An algorithm to perform the same table sum of products the conventional way might be as in Figure 6.2. This algorithm requires eight cycles for initialization, and five cycles each time through the loop for a total of 508 cycles or 101.6 microseconds.

The MAC instruction method is faster by almost a factor of five. The disadvantage is that the data and twiddle factor coefficients must be in on-chip memory to use the MAC instruction. This limits the data sample size to 256 samples for real data or 128 for complex data. It would be possible to swap the tables in and out of on-chip memory from external memory storage to take advantage of the MAC instruction. However, the extra code required to accomplish the swap is costly in time.

The other design task to be addressed in the direct DFT algorithm is to implement an efficient method to get the twiddle factor coefficients.

```
        LRLK    0,DATA      AR0 POINTS TO DATA TABLE
        LRLK    1,COEFF     AR1 POINTS TO COEFF TABLE
        LRLK    2,100       LOOP COUNTER = 100
        LARP    0           ARP TO AR0
        ZAC     0           ZERO ACCUMULATOR
LOOP    LT      *+,1        T REG = DATA, INC AR0, ARP TO AR1
        MPY     *+,2        MULT COEFF, INC AR1, ARP TO AR2
        APAC                ADD PREG TO ACCUMULATOR
        BANZ    LOOP,*-     DEC AR2, BRANCH IF NON-ZERO
```

Figure 6.2. Sum of Products Using MPY

Recall that,

$$W_N^{kn} = e^{-j(2\pi/N)kn} = \cos[(2\pi/N)kn] - j\sin[(2\pi/N)kn],$$

$$\text{Re}\ [\ W_N^{kn}\ ] = \cos[(2\pi/N)kn],$$

and,

$$\text{Im}\ [\ W_N^{kn}\ ] = \sin[(2\pi/N)kn].$$

Computing the sine and cosine terms inside the DFT loop would obviously be a slow process. The common method is to precalculate and store a table of these values in memory, so that no processor time is wasted calculating them inside the DFT loop. The length of the table can be minimized by making some observations. The sine and cosine functions above repeat if the product of k times n is greater than N. Therefore the table needs to be of maximum length N sines and N cosines. This can be reduced even more by observing that there is duplication in the two tables. By storing the sine table in memory and placing the cosine table start address N/4 locations into the table, the total table length is only 5N/4 memory locations. The table consists of every sin(l*2π/N) value, where l is an integer, l = 0 to N-1, and N is the maximum data sample size to be used in the program.

To access the twiddle factor coefficients while performing the DFT, indirect indexed addressing is very effective, with kn being the displacement from the beginning of the table.

A test must be made for each kn to determine if this displacement is larger than the length of the table. If it is, the displacement is bumped back to the correct place at the beginning of the table.

The stored table method of accessing the correct twiddle factor coefficients is much faster than calculating them on-the-fly during the DFT loop. However, it does require a branch test each trip through the loop to prevent over-running the end of the table. A table multiply algorithm which uses this table access and test for table end is presented in Figure 6.3. The AR0 is initialized with k at the beginning of the loop.

The AR1 is loaded with the address of the table. The indirect address mode with AR0 indexing is then used in the load T register (LT) instruction to access the coefficient. The instruction,

```
LT      *0+,2 ,
```

loads the T register with the contents of TABLE1 the first pass, TABLE1 + k the second pass, TABLE1 + 2k the third pass and so on. Meanwhile, the multiply (MPY) instruction is accessing the data table and auto incrementing the AR2 pointer by one each pass. So as k is incremented, the twiddle factor table is skipped through to get the correct table values, while each data sequence value is accessed consecutively. The TMS32020 instructions and addressing modes make a very efficient algorithm possible.

```
        LAR     0,K             AR0 = K
        LRLK    1,TABLE1        AR1 POINTS TO COEFF TABLE
        LRLK    2,DATA          AR2 POINTS TO DATA TABLE
        LAR     3,LENGTH        AR3 = LENGTH OF COEFF TABLE
        LARP    1               ARP TO AR1
        ZAC                     ZERO ACCUMULATOR
LOOP    LT      *0+,2           TR=COEFF, AR1=AR1+K
        MPY     *+,3            MULT BY DATA, INC AR2
        APAC                    ADD TO ACCUMULATOR
        BANZ    LOOP,*0-,1      AR3 = AR3-K, BRANCH IF NZ
RESET     (ROUTINE TO BUMP COEFFICIENT TABLE POINTER
                  TO BEGINNING OF TABLE)
          .
          .
          .
        B       LOOP            CONTINUE MULTIPLY
```

Figure 6.3. Table Multiply With Table End Test

However, an equally efficient method to make the end of table test for a DFT application was not as successful. At first, it seems the BANZ instruction should work well here. If AR3 is initialized with the table length and AR0 with k, the instruction,

```
BANZ    LOOP,*0-
```

would decrement AR3 by k each time through the loop. When AR3 becomes equal to zero, the end of the table has been reached, and the pointer should be reset N/2 locations back towards the beginning of the table.
The BANZ instruction is only a branch on auxiliary register non-zero however, as opposed to a branch on auxiliary register greater than or equal zero (this hyothetical instruction does not exist). This means that if AR3 does not hit exactly zero as it is being decremented by k, the test fails, the table end is passed and the program crashes.

Several methods of resetting the twiddle factor table pointer were tried, but they all seemed cumbersome compared to the remainder of the DFT algorithm. A method was devised, however, that eliminates the need of a end of table test altogether.

Since the TMS32020 has 64K words of program memory ROM space, the twiddle factor table can be made long enough, by repeating the sine table over and over, so that the end is never reached.

This means that N^2 values are needed in the table, or 16384 words of memory for a data sample size of N = 128 ($N^2/2$ values are needed for real data since only N/2 terms need to be computed). This method is basically a choice of trading memory for speed. Absolutely no processor time is wasted within the DFT loop for a table over-run test, and in times of plentiful memory and inexpensive prices, the tradeoff seems worthwhile. This method will not work for N larger than 256 on the TMS32020 due to the 64K limitation, but this data sample size is into the range where the FFT algorithm is faster anyway.

Due to the speed advantages, and no serious disadvantages, this method for the twiddle factor table storage and access was adopted for use in the DFT program presented in Appendix C. This program also takes advantage of the repeat (RPT) instruction, as discussed in the next section, to achieve fast execution.

TMS32020 Direct DFT Program

The linker listing of TMS32020 code to perform a direct DFT is contained in Appendix C. A memory map of this program is shown in Figure 6.4 .

The source code is commented but the following description describes some of the less obvious features. Computation of the DFT and FFT for this study is based on two's complement fixed point arithmetic.

This means that N^2 values are needed in the table, or 16384 words of memory for a data sample size of N = 128 ($N^2/2$ values are needed for real data since only N/2 terms need to be computed). This method is basically a choice of trading memory for speed. Absolutely no processor time is wasted within the DFT loop for a table over-run test, and in times of plentiful memory and inexpensive prices, the tradeoff seems worthwhile. This method will not work for N larger than 256 on the TMS32020 due to the 64K limitation, but this data sample size is into the range where the FFT algorithm is faster anyway.

Due to the speed advantages, and no serious disadvantages, this method for the twiddle factor table storage and access was adopted for use in the DFT program presented in Appendix C. This program also takes advantage of the repeat (RPT) instruction, as discussed in the next section, to achieve fast execution.

TMS32020 Direct DFT Program

The linker listing of TMS32020 code to perform a direct DFT is contained in Appendix C. A memory map of this program is shown in Figure 6.4 .

The source code is commented but the following description describes some of the less obvious features. Computation of the DFT and FFT for this study is based on two's complement fixed point arithmetic.

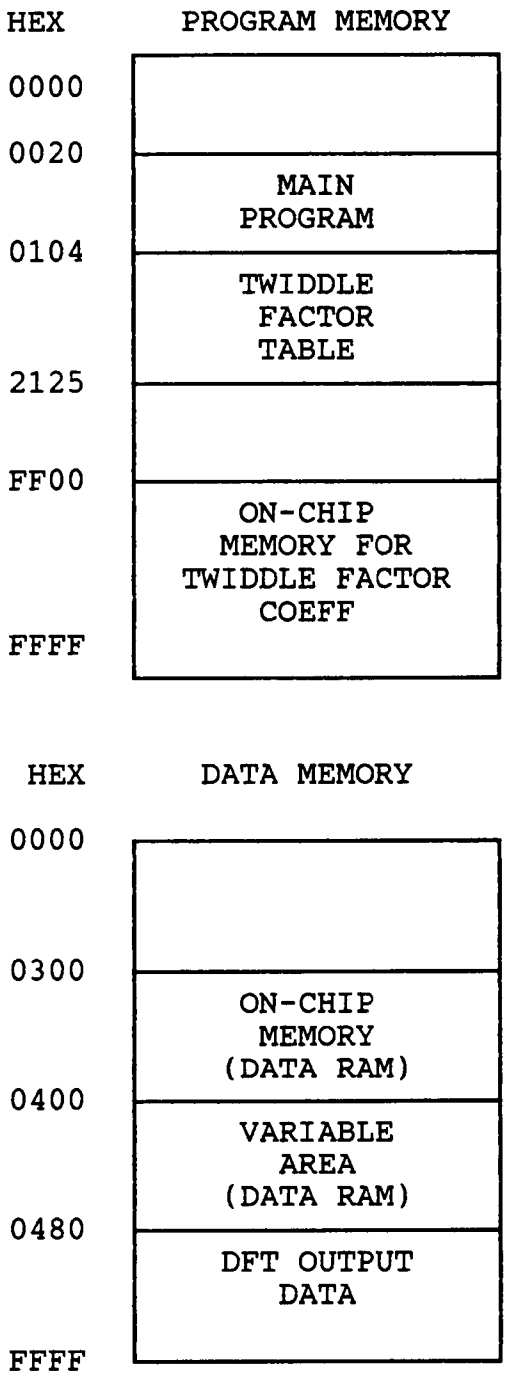


Figure 6.4. TMS32020 Direct DFT Program Memory Map

The format used is Q15, which implies no integer bits and 15 fractional bits (the decimal point is just to the right of the sign bit). The maximum twiddle factor value is an integer 32767 which represents a one in Q15 format. An extra sign bit is gained in the multiply operations, so a left shift is required. The possibility of an overflow exists in addition, so the input data must be scaled to avoid the overflow error.

Refering to the listing file in Appendix C, lines 58 through 180 define the macros used in the program. Macros are convenient when repetitive code is needed. When the macro name is inserted into the source code, the macro is expanded at assemble time, including passing named parameters.

Lines 181-198 initialize variables and the coefficient table in data RAM by doing a table read from program ROM. All constants in data RAM must be initialized this way. Program memory is assumed to be non-volatile such as ROM, while data RAM is volatile and lost upon powering down the system.

Lines 199-222 reads the input data sample sequence from input port zero. The first word read contains the number of words of data to be read (N), followed by the actual data.

Lines 223-226 are the code to calculate the variable MF (MF = 128/N). MF is multiplied by k to form the index KXMF. KXMF is used to index through the twiddle factor tables to allow one table to work for any N less than 128. N must be a power of two.

Lines 227-245 are to calculate the variable SHFT, where $N = 2^{SHFT+1}$. SHFT is used within the DFT loop to divide the accumulator by N, shifting right SHFT times. SHFT is made to be one less shift than required to divide by N. This causes an effective left shift of one to allow for the extra sign bit gained in the multiply.

Lines 250-292 are the DFT loop. To take advantage of the MAC instruction in the repeat mode, each of the two terms of the DFT real part (see Eq. 4.2, p. 27) are computed as a separate sum of products and the results added. The same approach is used for the imaginary term. Figure 6.5 illustrates this program flow. Also, special handling is required for the twiddle factors. A different set of coefficients are read from the off-chip twiddle factor table into on-chip memory block B0 before the DFT loop begins for each k. The direct DFT program of Appendix C was written to use complex data, but the program listing shows a change for a test case using only real data, where only N/2 DFT terms are calculated, and the twiddle factor table length is $N^2/2$ rather than N^2.

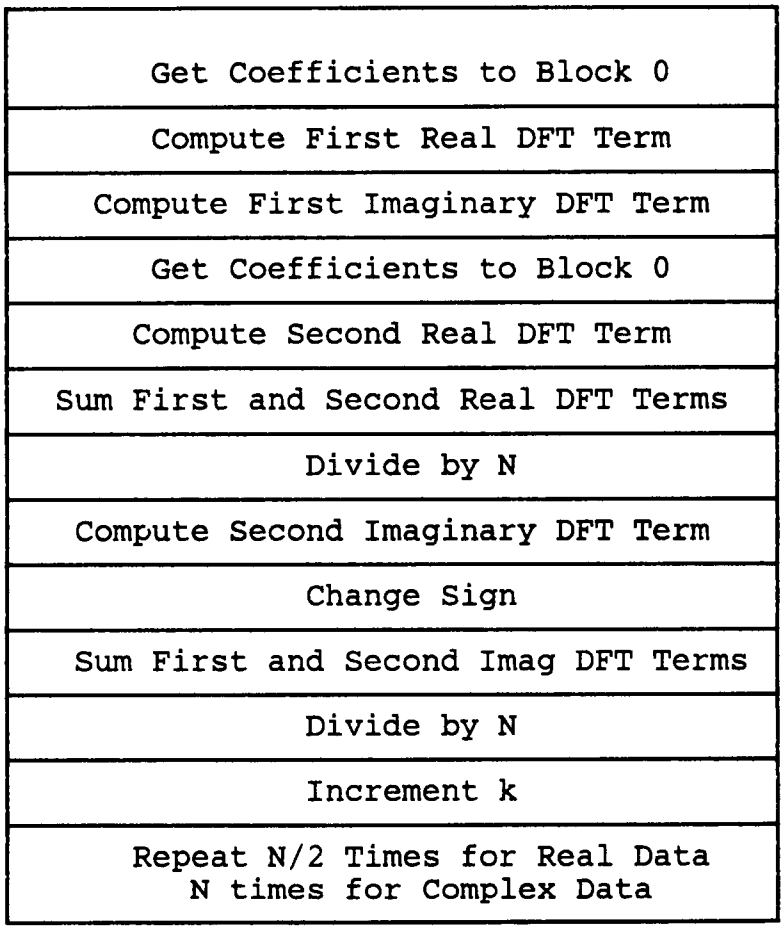


Figure 6.5. Direct DFT Program Flow

The program can be optimized further, if only real data is required, by eliminating the sum and products terms for the imaginary part of the data. If complex data are to be used with the program of Appendix C, the twiddle factor table has to be lengthened, and line 288 changed to load AR2 with N rather than N/2.

FFT Program for the TMS32020

In contrast to the design of the direct DFT program, the FFT design required few decisions to implement on the 32020. Since the butterfly computations of the decimation-in-frequency FFT do not require any long sum of products, the MAC instruction does not show a distinct advantage over the conventional multiply as it did for the direct DFT program. Using the standard multiply has the advantage of fewer restrictions on the memory location of the data and twiddle factors. This means that larger data sample sizes can be used, limited only by the address space of the 32020. Also the twiddle factor table problem is minimized due to the structure of the FFT. One sine table and one cosine table, each of length N/2 is all that is required.

The FFT program presented in Appendix D is a straightforward implementation of the butterfly flow diagram. The 32020 FFT is very similar in structure to the BASIC program of Chapter 4. This program will work for any N, a power of two, up to 1024.

A memory map of the FFT program is shown in Figure 6.6. Refering to Appendix D, the code is commented but the following is a description of the program.

Lines 20-296 define the macros. Lines 297-305 initialize program variables and set up the memory configuration. Lines 306-322 read N (number of data samples) and the input data sequence from input port 0. Lines 324-346 initialize the necessary RAM variables and calculate the multiplication factor (MF) for k. The MF is multiplied by k to form the twiddle factor table index KXMF as in the direct DFT program.

Lines 348-382 create a table for use in the program to unscramble the order of the FFT output data. The table consists of a set of bit reversed numbers (unique for each N) to be used as the index to reorder the FFT output. This method of unscrambling was chosen to remove the bit reversal computation time from within the FFT loop, which is important if repetitive FFTs are to be calculated.

Lines 384-402 write the twiddle factor tables from program ROM to data RAM. The length of the table written to data RAM is N. The values are different for each k.

Lines 404-507 perform the FFT. The section of code from line 432 to line 484 does the butterfly calculation. Register AR0 is loaded with SPAN, the offset between the two data points to be operated upon.

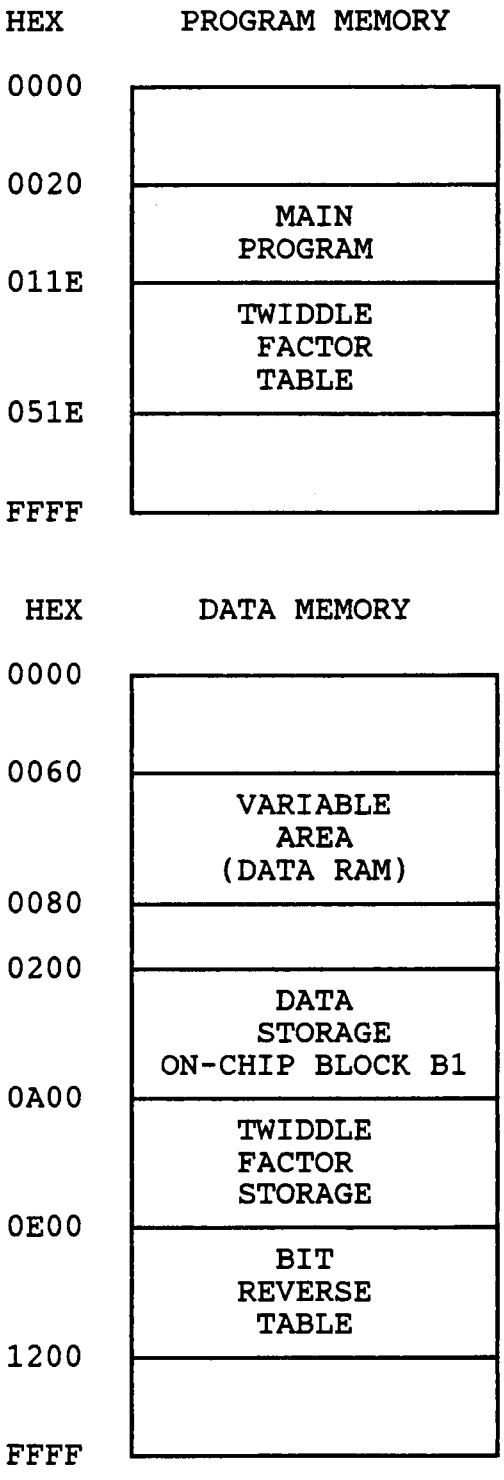


Figure 6.6. TMS32020 FFT Program Memory Map

Register AR1 points to the top real data point, and register AR3 points to the top imaginary data point. Register AR2 points to the real twiddle factor table and register AR4 points to the imaginary twiddle factor table. By setting up the auxiliary registers in this manner, and making use of the indirect addressing modes, the ARAU can perform the automatic memory pointer incrementing and indexing as the program steps from one butterfly calculation to another.

First the sum and difference for the real data and then for the imaginary data is performed. Register AR0 is loaded with KXMF, the index for the twiddle factor table. Then the multiplication by the twiddle factor is done. A divide by two is done on the output of each butterfly sum to prevent overflow. The extra sign bit gained in the multiply of the twiddle factor accomplishes the divide by two this on the difference butterfly. The result of this divide by two at each stage is an overall division by N as required in Eq. 2.12, p. 7.

The memory pointers are bumped to the next butterfly and the process repeats unless all of the butterflies in the flock are finished. When this happens the program branches to line 421, FLOCK, where the memory pointers are initialized to the beginning of the next group of butterflys.

After the last flock in the stage, the program branches to line 408, STAGE, setting up the pointers to the beginning of the next stage. If the end of the last stage has been reached, the program branches to the bit reverse routine.

Lines 509-546 perform the unscrambling of the FFT output through use of the bit reversed table previously stored. Lines 548-561 write the data to the output port one. Lines 564-575 are the sine and cosine twiddle factor table storage in program memory. The UNL assembler directive was used here to prevent listing them due to their length. Lines 576-604 is the section of the program that allocates data memory (RAM) storage. The BSS is a block storage directive that reserves memory for tables and variables.

CHAPTER 7

RESULTS OF TMS32020 DFT PROGRAMS

This chapter will present the results of DFTs that the programs of Appendices C and D computed. The same signals that were used in Chapter 4 for the BASIC FFT program results are used as input signals here for comparison. This will answer the question of whether the programs give the correct answers or not. The other question that will be answered, perhaps the most crucial one of this investigation, is how fast can the TMS32020 perform an FFT.

Figure 7.1 through 7.4 show the plots from the program of Appendix C, the TMS32020 direct DFT program. Figure 7.1 is the DFT amplitude and phase of the cosine wave with an integer number of cycles within the window. Since only N/2 (64) DFT points are calculated, only one amplitude peak appears (at 2000 Hz.).

Figure 7.2 is the same signal except there is not an even number of cycles of the signal in the window so that the leakage effect is evident.

Figure 7.3 is the DFT amplitude and phase of a sine wave. Note the 90 degree phase difference.

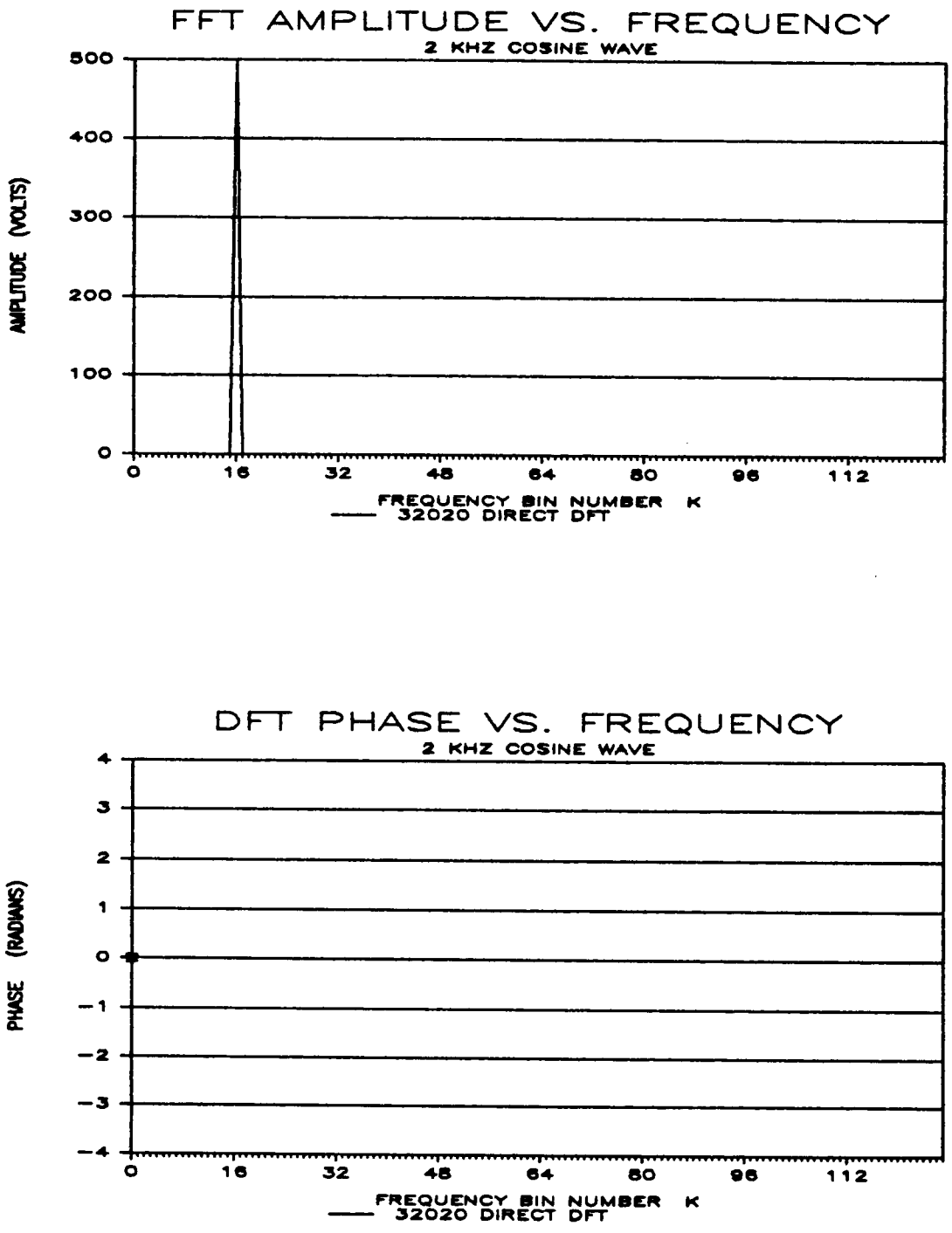
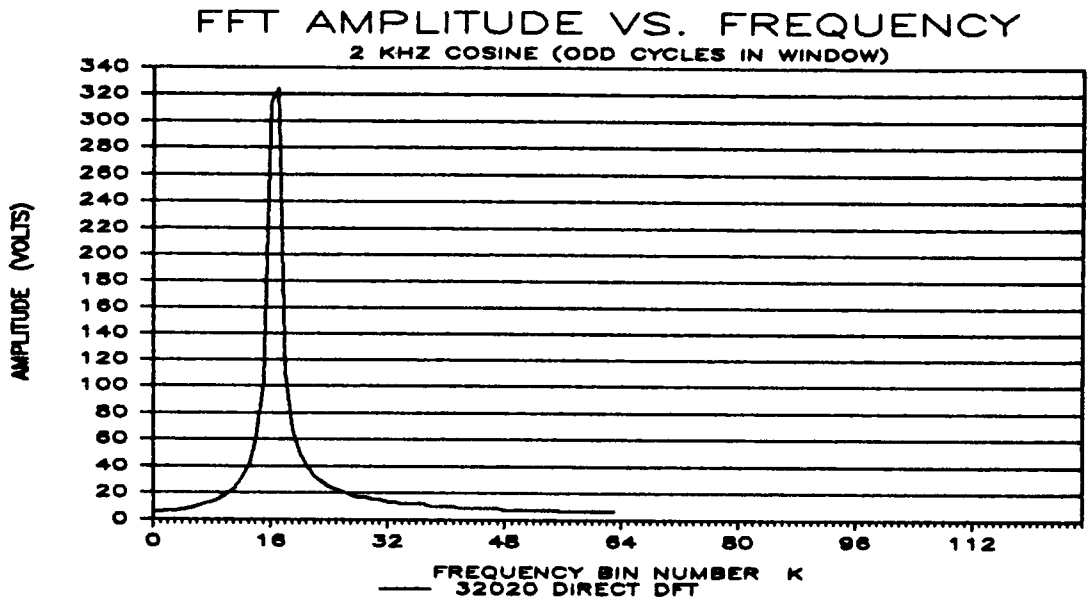


Figure 7.1. DFT of a 2000 Hz. Cosine Wave.



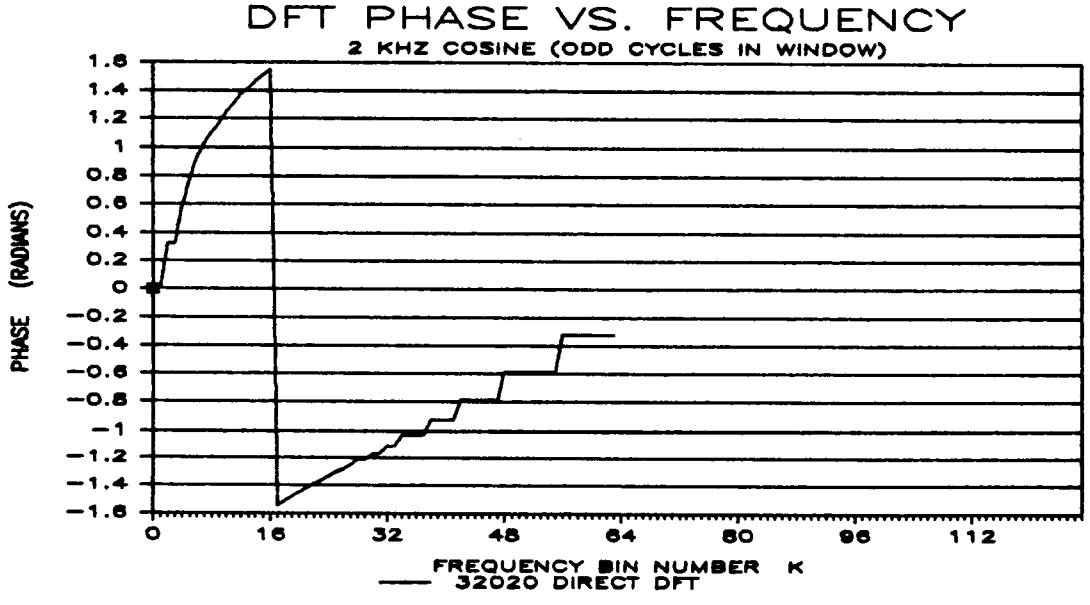
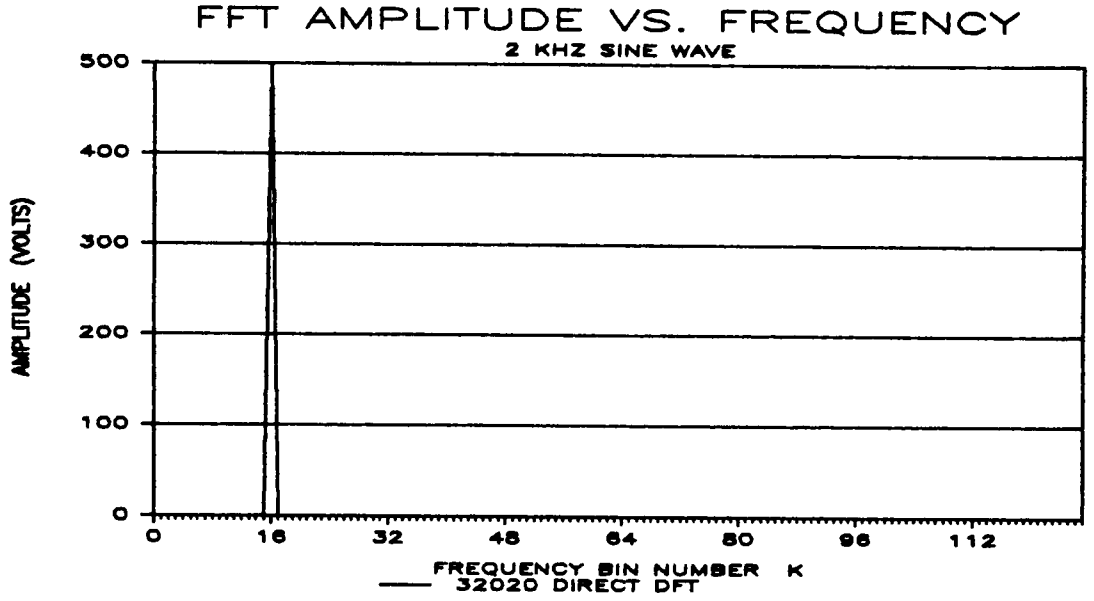


Figure 7.2. DFT of a 2000 Hz. Cosine Wave With Non-Integer Number of Cycles in the Window.



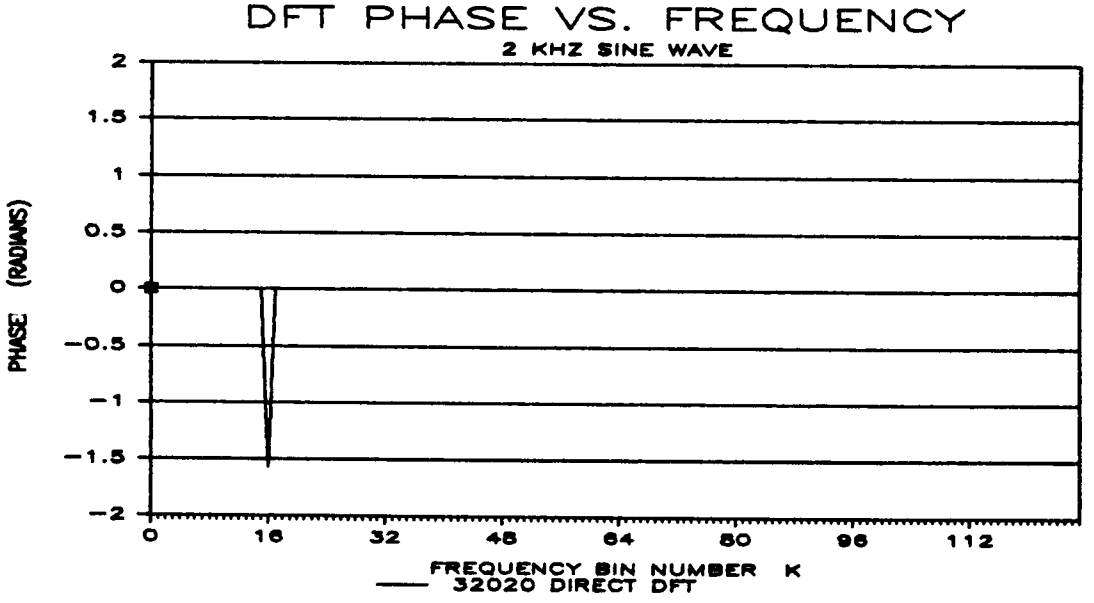


Figure 7.3. DFT of a 2000 Hz. Sine Wave.

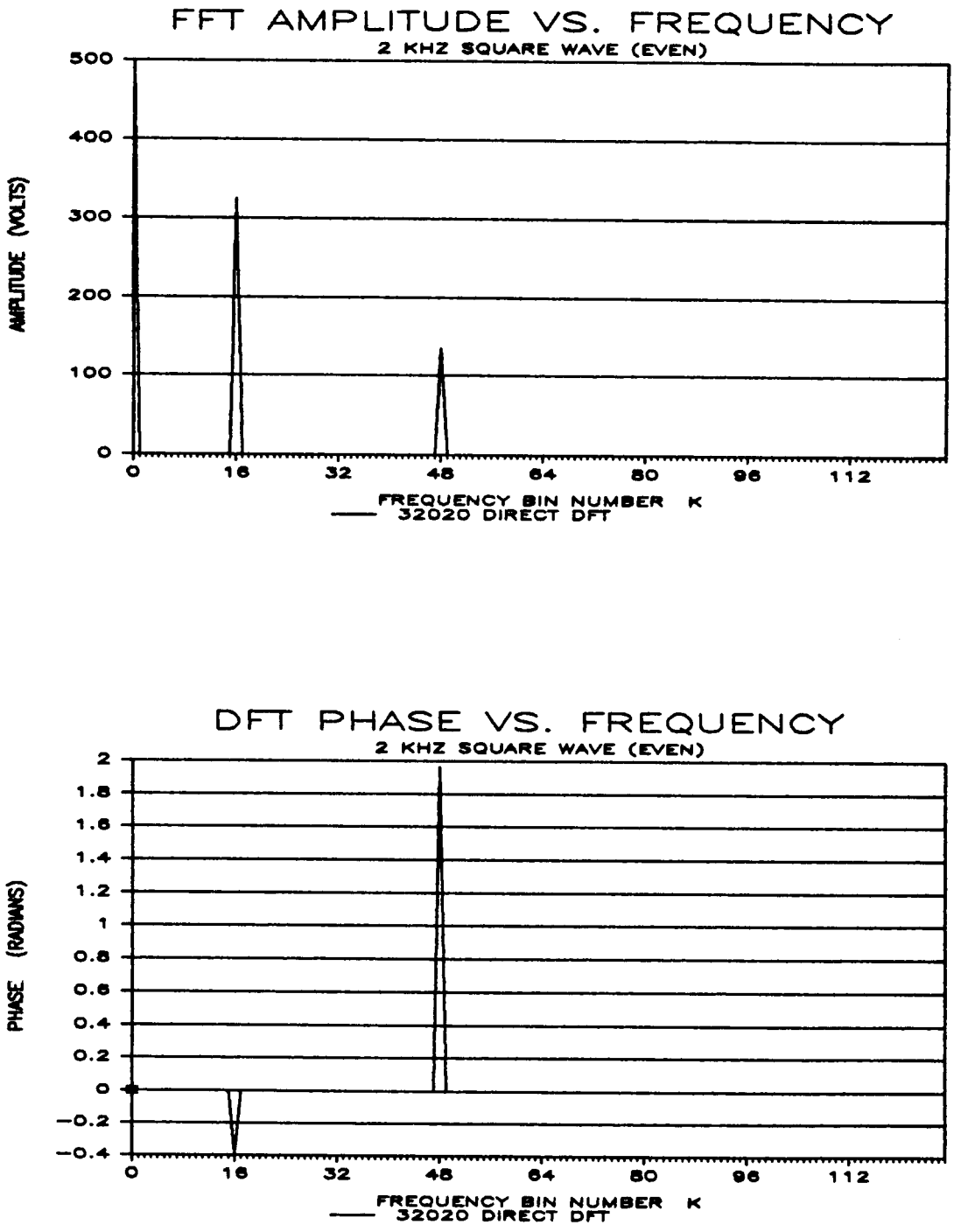


Figure 7.4. DFT of a 2000 Hz. Square Wave (Even).

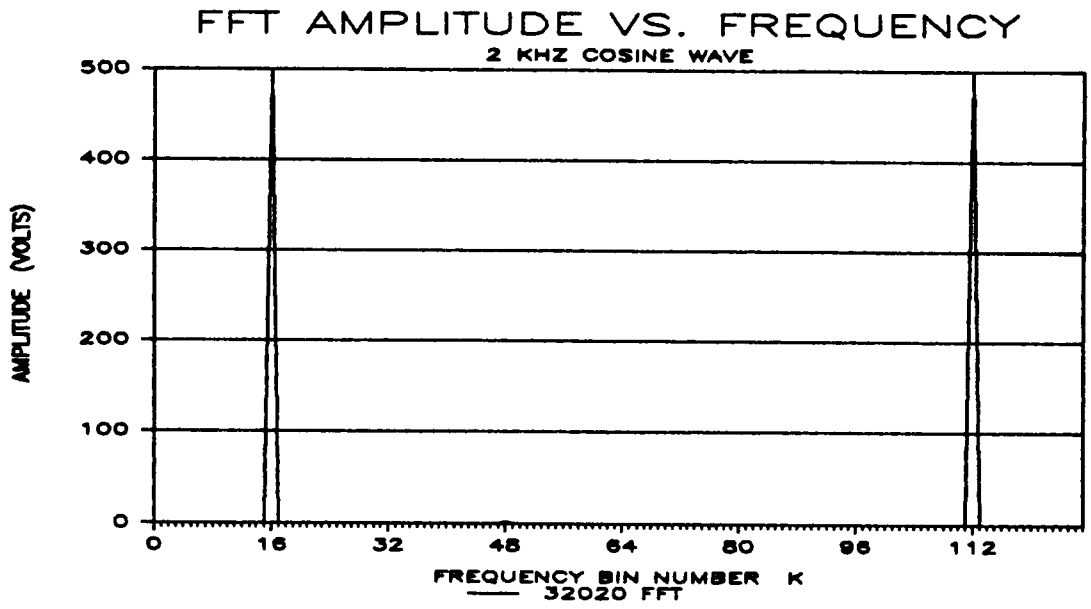
Figure 7.4 is the DFT output for an even square wave which shows the even harmonics.

Figures 7.5 through 7.8 are the amplitude and phase plots from the TMS32020 FFT program of Appendix D for the same input signals. There is a $-\pi/2$ phase shift at k=48 on the phase plot of the sine wave. This was caused by an integer arithmetic roundoff error. The arctan function returned a $-\pi/2$ for a zero real FFT point and an imaginary FFT point of -1 ($\arctan(-1/0)=-\pi/2$).

All of the plots, for the BASIC as well as the TMS32020 programs agree closely, demonstrating that the programs do give the correct answers. The other question is how fast do the TMS32020 programs run.

Figure 7.9 is a plot of N versus machine cycles for the direct DFT and the FFT. The machine cycles for this count were for that portion of the code that would be repetitively executed, as in a real-time spectrum analysis application. This is the code from lines 250 to 291 in the direct DFT program, and lines 408 to 546 in the FFT program. The initialization at the beginning and the data output at the end is not included. The bit reverse unscrambling is included in the FFT machine cycle totals.

Figure 7.10 shows the machine cycle totals for the TMS32020 FFT program for N up to 1024.



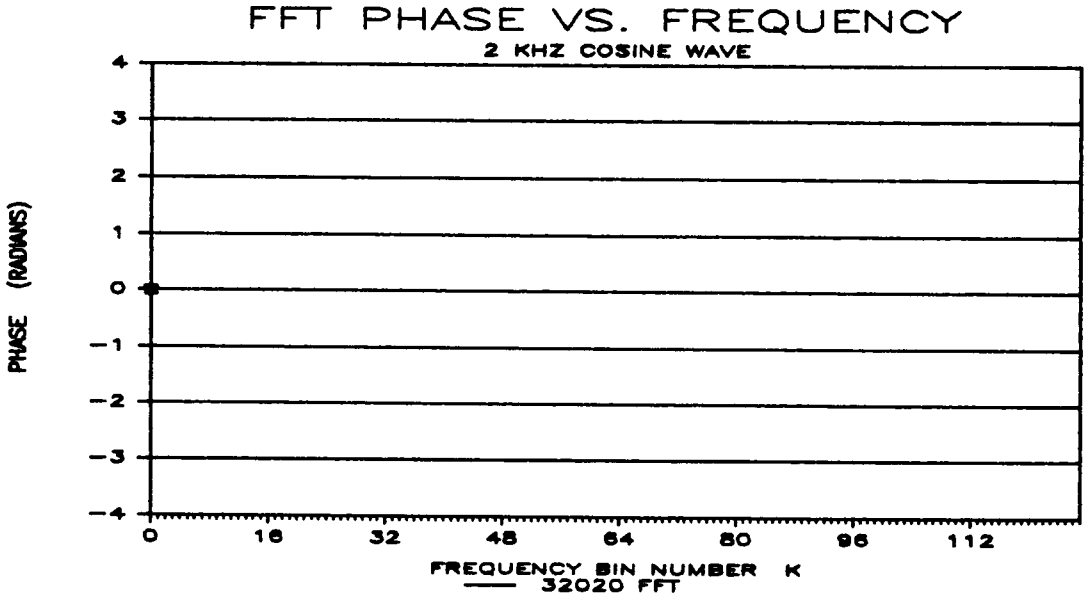
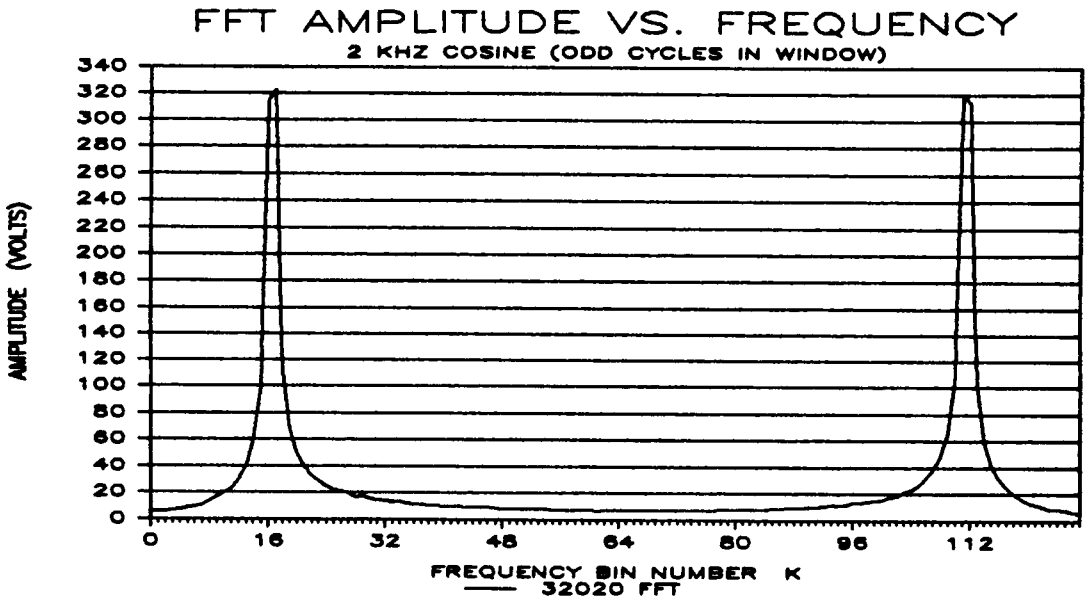


Figure 7.5. FFT of a 2000 Hz. Cosine Wave.



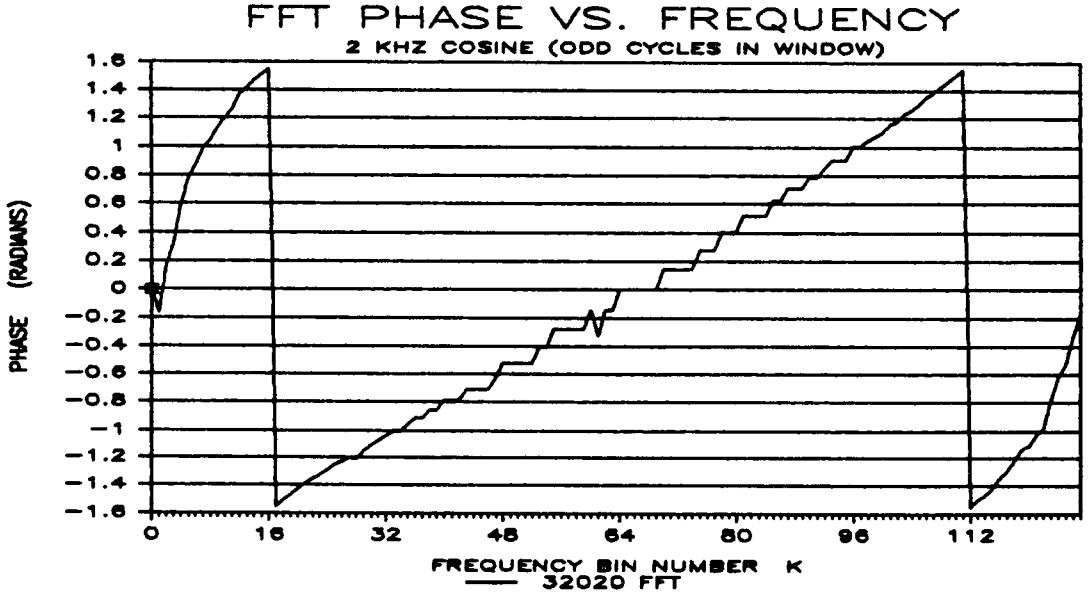


Figure 7.6. FFT of a 2000 Hz. Cosine Wave With Non-Integer Number of Cycles in the Window.

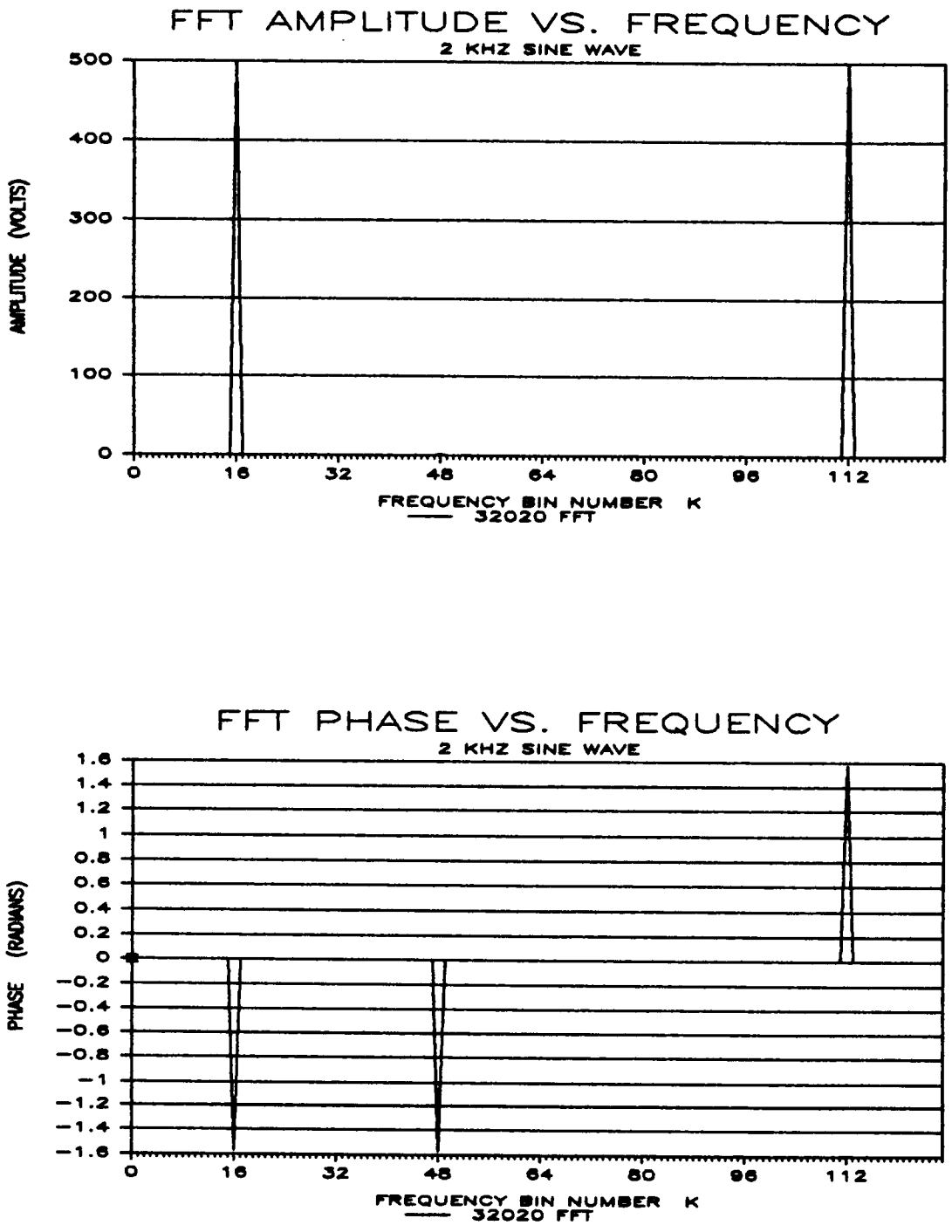
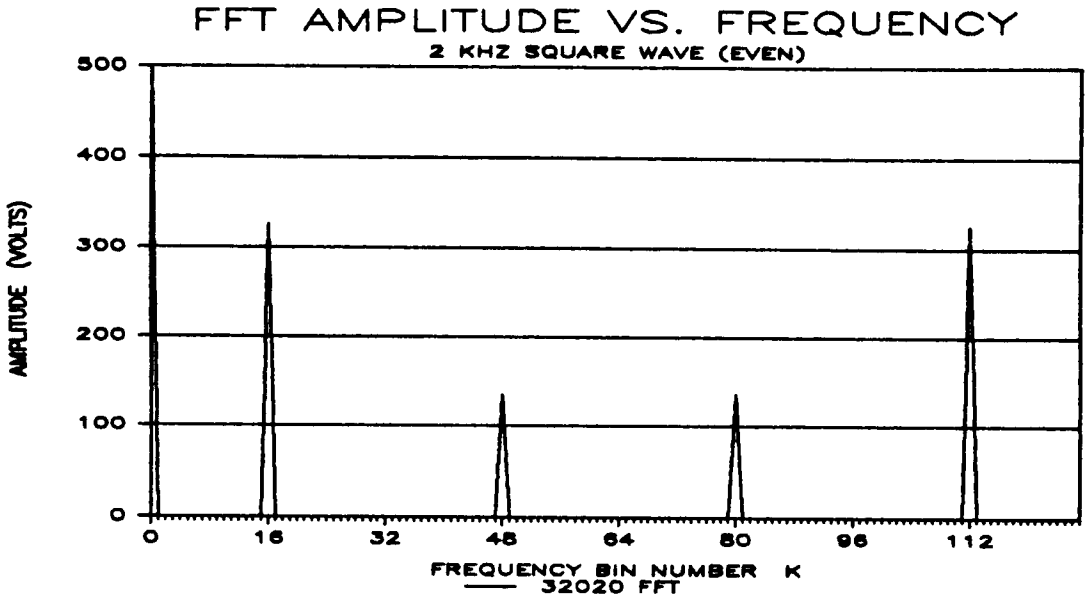


Figure 7.7. FFT of a 2000 Hz. Sine Wave.



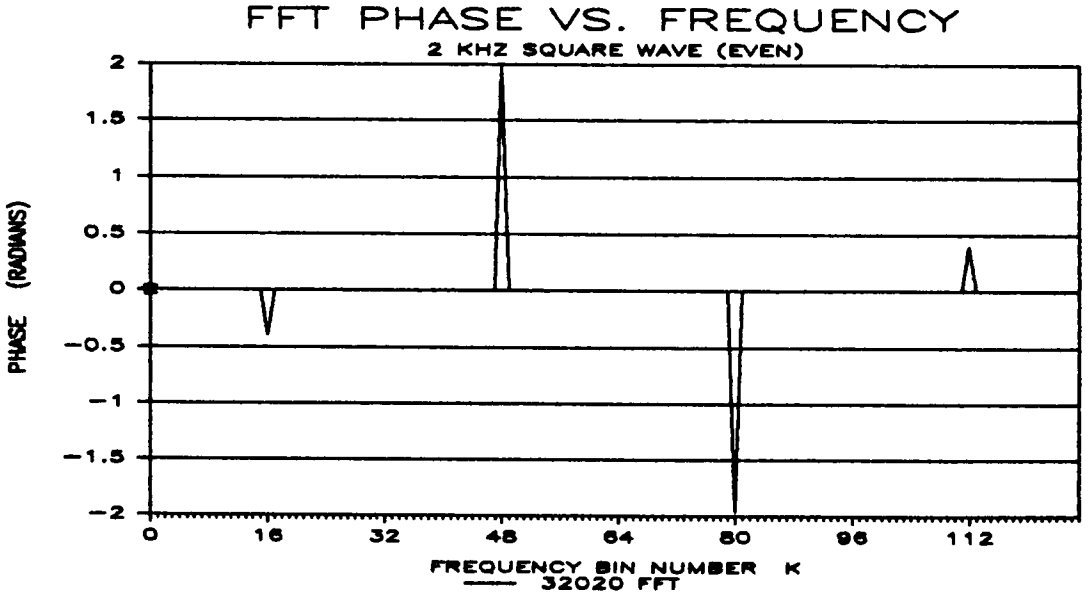


Figure 7.8. FFT of a 2000 Hz. Square Wave (Even).

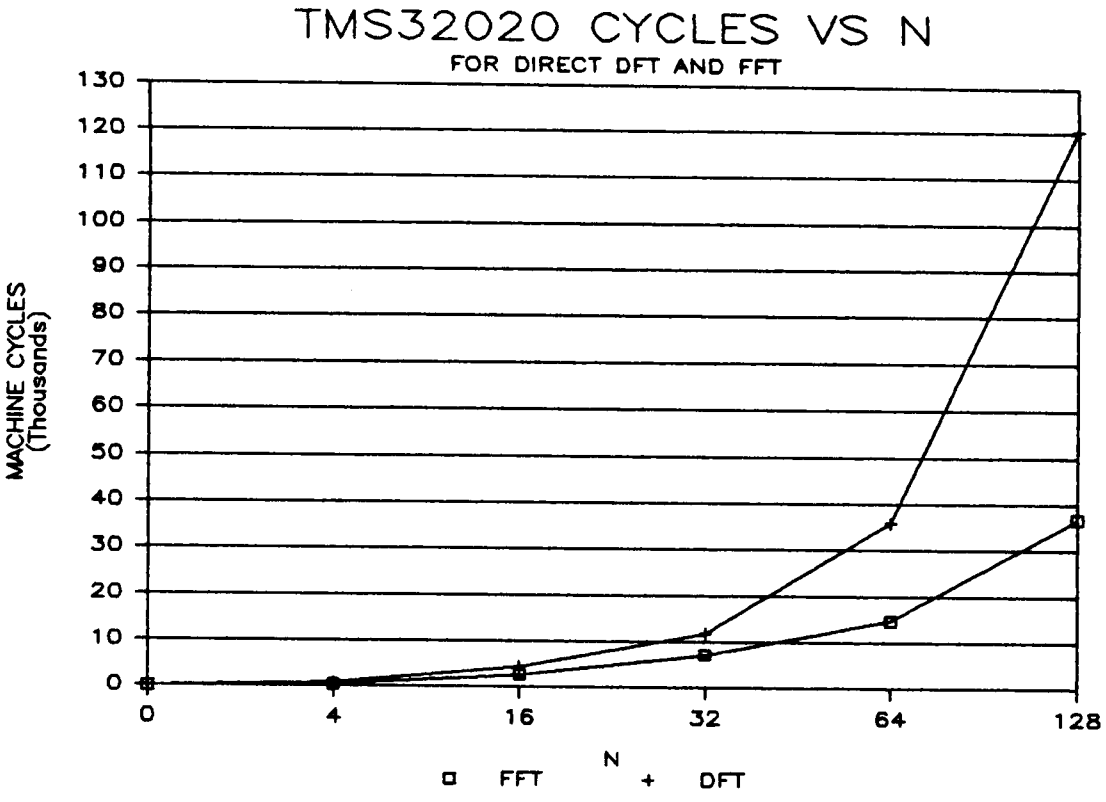


Figure 7.9. TMS32020 Machine Cycles for a N-point Complex DFT Using the Direct DFT and FFT Methods.

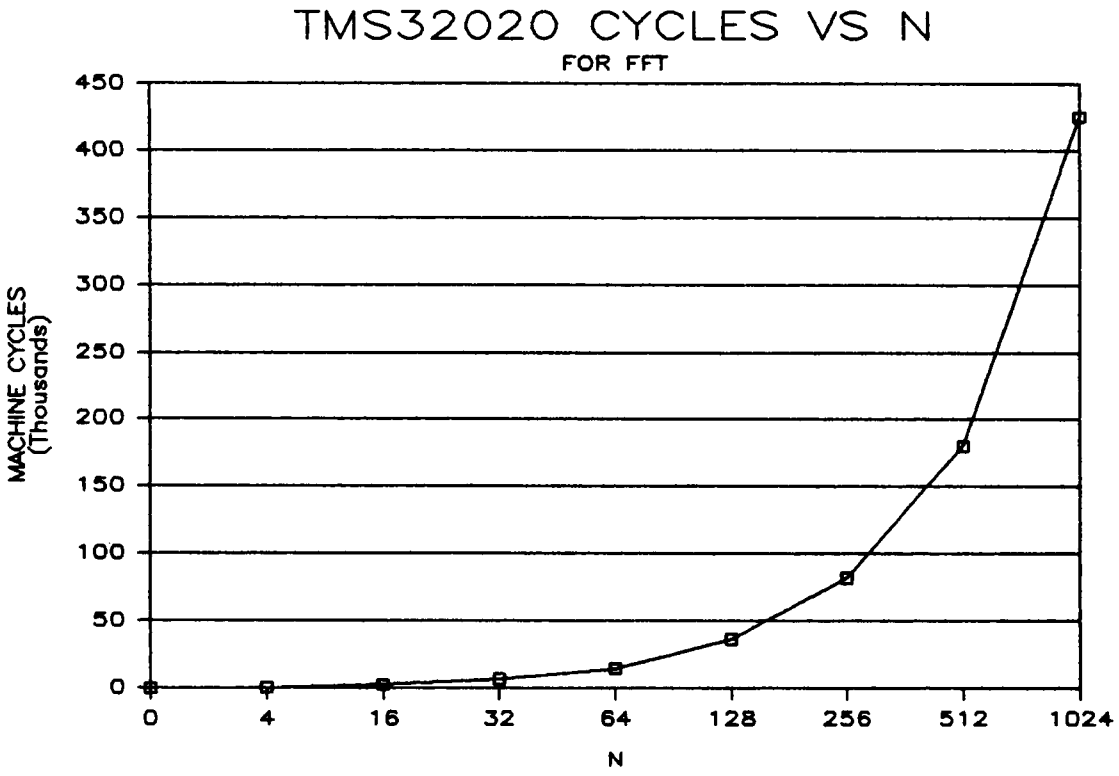


Figure 7.10. TMS32020 FFT Program Machine Cycles Vs. N for a N-point Complex DFT

CHAPTER 8

CONCLUSIONS

The primary considerations in evaluating the performance of the TMS32020 microprocessor for DFT applications are:

1) Can a program be written for the TMS32020 that works accurately for a reasonably large data sample size?

2) How efficient is such a program, judged simply by how much time it takes to perform a DFT for an N point data sample?

These questions can only be answered by writing the programs, using the features of the architecture and instruction set in the most efficient manner that the writer can conceive during development. The program must then be performance tested for speed and accuracy. The code is evaluated for possible improvements, modified, and the process repeated. This development process involves too many decisions and tradeoffs to fully document in a reasonable number of words. Nonetheless, the previous chapters have presented the major design decisions that lead to the programs of Appendices C and D.

The end result of the investigation, however, is the working programs themselves. These programs are the best documentation of how suitable the hardware is for the task. This chapter will summarize the advantages and disadvantages of the programs and discuss the overall performance of the TMS32020 in a DFT application.

The addressing modes and powerful instructions of the TMS32020 make it possible for very compact program code to be written for the DFT. The combination of using the ARAU to perform arithmetic on the auxiliary registers, while executing an instruction using indirect indexed addressing, while in the repeat mode, is a very impressive feat for one instruction to accomplish in 200 nanoseconds. The instruction set seems to be complete, with the possible exception of more choices for conditional branching based on auxiliary register contents.

The TMS32020 direct DFT program performs a 64 point complex data DFT in 7.16 milliseconds. For comparison, a direct DFT program written for the TMS32010 and presented in reference [1], performs a 64 point complex data DFT in 29 milliseconds. The dramatic increase in speed of the TMS32020 program is due mainly to the single-cycle MAC and the ARAU.

Also, the large memory space is a factor in the increased speed, allowing the large twiddle factor table to be used, eliminating the end of table test. The direct DFT program is an application that takes full advantage of the enhanced architecture of the TMS32020. It should also be noted that the DFT program has less error introduced due to roundoff than the FFT program. A full 32 bit running sum is kept throughout the computation of each DFT coefficient where the result of each FFT butterfly computation is divided by one-half to prevent overflow conditions, and then stored to 16 bits of precision decreasing the overall accuracy.

The TMS32020 FFT program of Appendix D performs a 1024 point transform in approximately 85 milliseconds. The FFT program runs faster than the direct DFT program for any N. The direct DFT program does not show an expected speed advantage for small N over the FFT, although if both programs were optimized for speed and for a particular N, the outcome might be different.

Another way to judge the speed of the FFT on the TMS32020, is to calculate the real time bandwidth. This is the bandwidth at which no data is lost waiting on the FFT to finish. The FFT program executes in 85 ms. for a 1024 point FFT.

This places a maximum sample rate of, f_s = 1024/0.085 = 12,047 Hz. for no loss of data, so the Nyquist frequency ($f_s/2$) is approximately 6 KHz. . Practically, this restricts the real time bandwidth to around 4 KHz.. This figure is considerably lower than real-time bandwidths of some of the commercially available digital spectrum analyzers . The real-time bandwidth of currently available digital spectrum analyzers is around 10 Khz. for a 800 frequency point FFT [7].

As for other advantages, the memory space on the TMS32020 is large enough for a 32K point radix-two FFT. There is enough program space for storing the program and 32k twiddle factor table, and enough data memory for 32K data samples. This still leaves almost half of the total memory space free, but the next radix-two FFT would be 64k, which immediately uses all of the data memory space for the data sample storage. The 32K FFT leaves plenty of program memory and data memory space for adding features such as:

1. Using the input/output (I/O) capability of the chip to control analog-to-digital (A/D) converters and communications to other peripherals or processors,

2. Performing averaging and other statistical operations,

3. Using a variety of windows to minimize leakage.

There should be enough memory space for two 16K point FFTs to be performed, and the FFT outputs used to perform convolutions, correlations, and other signal processing for which the FFT is useful.

The final conclusion is that the TMS32020 is a very good performer for FFT applications. Compact efficient assembler code can be written that result in a very respectable speed of execution of the FFT. The FFT programs developed in this investigation do not execute as fast as some of the commercial spectrum analyzers, but the fact that a microprocessor is available that compares favorably with the speed of expensive instruments designed specifically for performing the FFT, is impressive. The TMS32020 programs developed for this study work well, and care was taken to use the advanced features of the microprocessor. But these programs are not represented as being optimally fast, and improvements can be made to both the direct DFT and the FFT program to improve the execution speed. An TMS32020 DSP applications note from Texas Instruments, Inc. [14] which was released toward the end of this study, presents a variety of programs that perform the FFT faster than the programs written for this study.

However, the programs written for this study work for any N (up to a maximum N = 1024 for the FFT, and a maximum N = 128 for the direct DFT) while the programs of [14] are tailored for a specific N. Some of the techniques used in [14] to optimize the FFT computation speed are: 1) radix-4 and radix-8 FFT algorithms, 2) straight line code to eliminate looping (requires more program memory space but runs faster), and 3) special butterfly macros for the cases where a multiply can be eliminated (twiddle factor value is one).

A final note to be made is that Texas Instruments has recently announced a new version of the TMS32020, the TMS320C25, that has a 100 nanosecond machine cycle. This should double the speed performance of the same FFT programs presented here.

LIST OF REFERENCES

LIST OF REFERENCES

1. Burrus, C. S., and T. W. Parks. DFT/FFT and Convolution Algorithms. New York: John Wiley and Sons, 1985.

2. Cooley, J. W., P. A. W. Lewis, and P. D. Welch. "Historical Notes on the Fast Fourier Transform," IEEE Trans. Audio Electroacoust., Vol. AU-15, pp. 76-79, June, 1967.

3. Cooley, J. W., and J. W. Tukey. "An Algorithm for the Machine Calculation of Complex Fourier Series," Mathematics of Computation, Vol. 19, No. 90, pp. 297-301, 1965.

4. Elliot, D. F., and K. R. Rao. Fast Transforms: Algorithms, Analyses, Applications. Orlando: Academic Press, Inc., 1982.

5. Gold, B., and C. M. Rader. Digital Processing of Signals. New York: McGraw-Hill Book Company, 1969.

6. Gupta, S. C. Transform and State Variable Methods in Linear Systems. New York: Robert E. Krieger Publishing Co., Inc., 1971.

7. Hewlett Packard 1986 Catalog. Hewlett Packard, Inc., 1986.

8. Lam, H. Y-F. Analog and Digital Filters. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1979.

9. Link Editor User's Guide. Houston: Texas Instruments, Inc., 1985.

10. Oppenhiem, A. V., and R. W. Schafer. Digital Signal Processing. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1975.

11. Rabiner, L. R., and C. M. Rader (editors). Digital Signal Processing. New York: IEEE Press, 1972.

12. Ramirez, R. W. The FFT: Fundamentals and Concepts. Beaverton, Oregon: Tektronix, Inc., 1975.

13. TMS32020 User's Guide. Houston: Texas Instruments, Inc., 1985.

14. Papamichalis, P. and J. So. Implementation of Fast Fourier Transform Algorithms with the TMS32020. Houston: Texas Instruments, Inc., 1986.

APPENDICES

APPENDIX A

BASIC DFT PROGRAM

```
10 DEFINT I,J,K,L,M,N
20 DEFDBL F,W,X,T
30 DIM XR$(128),XR(128),XI$(128),XI(128),FFTR(128),FFTI(128)
40 '=======================READ INPUT DATA FILE================================
50 '
60 PI=3.141592654#
70 INPUT "PATHNAME FOR DATA FILE ";P$
80 OPEN "I",#1,P$
85 INPUT "PATHNAME FOR OUTPUT FILE ";Q$
87 OPEN "O",#2,Q$
90 INPUT #1,NSAM$
100 NSAM=VAL("&H"+NSAM$)
110 FOR I=0 TO NSAM-1
120     INPUT #1,XR$(I)
130     XR(I)=VAL("&H"+XR$(I))
140 NEXT I
150 FOR I=0 TO NSAM-1
160     INPUT #1,XI$(I)
170     XI(I)=VAL("&H"+XI$(I))
180 NEXT I
190 CLS
200 LPRINT "NO SAMPLES = ";NSAM
205 PRINT #2,"NO SAMPLES = ";NSAM
210 LPRINT "I","XR HEX","XR","XI HEX","XI"
215 PRINT #2,"I","XR HEX","XR","XI HEX","XI"
220 FOR I=0 TO NSAM-1
230     LPRINT I,XR$(I),XR(I),XI$(I),XI(I)
235     PRINT #2,I,XR$(I),XR(I),XI$(I),XI(I)
240 NEXT I
250 '=======================DIRECT DFT=========================================
260 THETA=2*PI/NSAM
270 LPRINT
280 LPRINT "K","FFTR","FFTI"
285 PRINT #2,"K","FFTR","FFTI"
290 WHILE K<NSAM/2
300     WHILE N<NSAM
310             WR=32767*COS(THETA*K*N)
320             WR=INT(WR)
330             WI=32767*SIN(THETA*K*N)
340             WI=INT(WI)
350             FFTR(K)=FFTR(K)+XR(N)*WR+XI(N)*WI
360             FFTI(K)=FFTI(K)+XI(N)*WR-XR(N)*WI
370             N=N+1
380     WEND
390     N=0
400     FFTR(K)=FFTR(K)/NSAM
410     FFTI(K)=FFTI(K)/NSAM
420     LPRINT K,INT(FFTR(K)),INT(FFTI(K))
425     PRINT #2,K,INT(FFTR(K)),INT(FFTI(K))
430     K=K+1
440 WEND
```

APPENDIX B

BASIC FFT PROGRAM

```
10 ' ================================================================================
20 ' |        TUKEY-SANDE (DECIMATION IN FREQUENCY) FFT ALGORITHM                    |
30 ' |               WRITTEN IN MICROSOFT GW-BASIC.                                  |
40 ' |                                                                              |
50 ' |   JACKSON W. FRAZIER   UNIVERSITY OF TENNESSEE SPACE INSTITUTE  1985         |
60 ' |                                                                              |
70 ' |    N=NUMBER OF DATA POINTS, MUST BE A POWER OF TWO.                          |
80 ' |    USES TABLE LOOK UP FOR TWIDDLE FACTOR COEFFICIENTS                        |
90 ' |    INCLUDES ALGORITHM FOR BIT REVERSING THE OUTPUT                           |
100 '|================================================================================
210 '
1000 '=================== INITIALIZATION ============================
1010 '
1020 DIM XR(1024),XI(1024)                    'input data space
1025 DIM WRE(512),WIM(512)
1030 DEFINT I,J,K,L,M,N                       'declare variables integers
1070 K=1                                      '
1080 PI=3.14159265358979#                     'constant
1090 RADEG=180/PI                             'conversion radians to degrees
1100 TWOPI=2*PI                               '
1110 CLS                                      'clear video screen
2000 '=================== MAIN FFT ROUTINE==========================
2002 '
2004 GOSUB 4000                               'generate input signal
2005 GOSUB 7000                               'select window
2006 DELTA=2*PI/ND                            'step size for phasor angle (Wn)
2007 NSPAN=ND/2                               'initial butterfly wing span
2010 GOSUB 8000                               'calculate twiddle coefficients
2020 WHILE K-ND                               'do it only if twid factor exponent<nd
2030   NTIP1=0                                'init butterfly tip index to zero
2040   NTIP2=NSPAN                            'bottom tip index = wing span
2045   ALPHA=DELTA*K                          'product of twiddle factor exp & 2*pi/n
2050   WHILE NTIP1-ND                         'do it only if top index is < nd
2060     GOSUB 3000                           'butterfly sums and products
2070     NTIP1=NTIP1+NSPAN                    'jump to the next flock of butterflys
2080     NTIP2=NTIP1+NSPAN                    'likewise for the bottom index
2090   WEND
2100   NSPAN=NSPAN/2                          'wing span is half for the next stage
2110   K=K*2                                  'exponent for twiddle factor is double
2120 WEND
2140 GOSUB 6000                               'output bit reversed, correct it
2160 GOSUB 5000                               'output results
2164 KEY(1) ON                                'enable trapping of F1
2165 ON KEY(1) GOSUB 2170
2166 GOTO 2165
2170 STOP
3000 '================ BUTTERFLY CALCULATIONS ================================
3010 '
3015 N=0                                        'initialize index
3020 WHILE NTIP1-NTIP2                          'do it only if top index is < ntip2
3030   KN=K*N
3035   WRE=WRE(KN)                              'real part of w^kn
3040   WIM=WIM(KN)                              'imaginary part of w^kn
3050   XTRE=XR(NTIP1)+XR(NTIP1+NSPAN)           'top sum of real components
3060   XTRE1=XR(NTIP1)-XR(NTIP1+NSPAN)          'bottom sum of real comp
3070   XTIM=XI(NTIP1)+XI(NTIP1+NSPAN)           'top sum of imag comp
3075   XTIM1=XI(NTIP1)-XI(NTIP1+NSPAN)          'bottom sum of imag comp
3090   XTRE2=XTRE1*WRE(KN)+XTIM1*WIM(KN)        'real product of sum and w^kn
3100   XTIM2=XTIM1*WRE(KN)-XTRE1*WIM(KN)        'imag product of sum and w^kn
3110   XR(NTIP1)=XTRE/2                         'save x real for in place calc
3120   XI(NTIP1)=XTIM/2                         'save x imag for in place calc
3130   XR(NTIP1+NSPAN)=XTRE2/2                  'ditto
3140   XI(NTIP1+NSPAN)=XTIM2/2                  'ditto
3150   NTIP1=NTIP1+1                            'go to next one
```

```
3155   N=N+1                                  'increment omega
3160 WEND
3170 RETURN
4000 '====================GENERATE INPUT SIGNAL==============================
4010 '
4030 AM=100                                          'amplitude of signal
4040 INPUT "SIGNAL FREQUENCY (HZ)";FREQ              'define frequency of signal
4050 INPUT "SAMPLE FREQUENCY";SAMPFREQ               'define sample frequency
4060 INPUT "NUMBER SAMPLES ( POWER OF TWO ) ";ND     'define sample size
4070 P=ND/SAMPFREQ                                   'length of window in seconds
4080 OMEGA=2*PI*FREQ                                 'step size
4090 FOR J=0 TO ND                                   'for n samples
4100     XR(J)=AM*COS(OMEGA*J/SAMPFREQ)              'calculate sample value
4110     XI(J)=0                                     'imaginary part = 0
4130 NEXT J                                          'loop
4140 RETURN                                          '
5000 '======================PLOT ROUTINE=====================================
5010 '
5020 M=0
5030 INPUT "PLOT LOG <1> OR LINEAR <2> ";LIN
5040 INPUT "SEND OUTPUT TO PRINTER <Y> OR <N> ";LPR$
5050 IF LPR$="Y" OR LPR$="y" THEN LPRINT "m","xr","xi","ax","px"
5060 WHILE M<ND                                       'find max,min for scaling
5070    AX=SQR(XR(M)*XR(M)+XI(M)*XI(M))              'magnitude
5075    IMAG=CINT(100*XI(M))
5077    IREAL=CINT(100*XR(M))
5078    IF IREAL=0 THEN IREAL=1
5080    PX=ATN(IMAG/IREAL)*RADEG                     'phase
5090    IF LPR$="y" OR LPR$="Y" THEN  LPRINT M,XR(M),XI(M),AX,PX
5100    XR(M)=AX                                     'store amplitude in xr and phase
5110    XI(M)=PX                                     'in xi for in place calculation
5120    IF LIN=1 THEN XR(M)=LOG(XR(M))               'log of data for plot
5130    IF XR(M)>XRMAX THEN XRMAX=XR(M)              'save maximum
5140    IF XR(M)<XRMIN THEN XRMIN=XR(M)              'save minimum
5150    M=M+1                                        'next frequency
5160 WEND                                            'do for all frequencies
5180 KEY OFF                                         'turn key at bottom off
5190 SCREEN 2:CLS                                    'set video mode
5205 IF LIN=1 THEN SCAL$="LOG" ELSE SCAL$="LINEAR"
5210 PRINT "         FREQUENCY (HZ)                        SAMPLE LENGTH"
5211 PRINT
5212 PRINT "SIGNAL","SAMPLE","RESOLUTION","NO. SAMPLES","SECONDS";"     SCALE"
5214 PRINT
5215 FORMAT$="#####        #####              ####         ####      ###.####      \    \"
5216 PRINT USING FORMAT$; FREQ,SAMPFREQ,1/P,ND,P,SCAL$
5218 PRINT
5220 PRINT "AMPLITUDE 1 VOLT/DIV   PHASE 30 DEG/DIV    ";WIND$;" WINDOW"
5240 WINDOW (0,-420)-(ND,420)                         'set coordinates to plot phase
5250 FOR LX= -90 TO 90 STEP 30                        'draw grid for phase plot
5260    PSET (0,LX)                                   'draw a point at x=0,y=lx
5270    FOR MX= 1 TO ND                               'want point at each n
5280       PSET (MX,LX)                               'draw a point
5290    NEXT MX
5300 NEXT LX
5310 LINE(ND/2,-90)-(ND/2,90)                         'vertical axis
5315 LINE(0,0)-(ND,0)                                 'horizontal axis
5320 M=0
5330 WHILE M<ND                                       'plot phase
5340    LINE(M,0)-(M,XI(M))                           'draw bars from axis to angle
5350 M=M+1                                            'next frequency
5360 WEND                                             'do for all n
5370 WINDOW (0,0)-(ND,3*XRMAX)                        'scale coord for mag plot
5380 LINE (0,0)-(ND,0)                                'draw horizontal axis
5390 LINE (ND/2,0)-(ND/2,XRMAX)                       'draw vertical axis
```

```
5392 FOR LY= 1 TO XRMAX
5394     PSET (0,0)
5395     FOR MX= 1 TO ND
5396         PSET(MX,LY)
5397     NEXT MX
5398 NEXT LY
5400 M=0
5410 WHILE M<ND                                  'plot amplitude
5420    LINE (M,0)-(M,XR(M))                     'draw bar
5430    M=M+1
5440 WEND
5450 RETURN
6000 '========================== BIT REVERSE THE ORDER ===================
6010 '
6020  FOR I=1 TO ND-1
6030    MASK=1
6040    J=ND/2
6050    M=0
6060    WHILE MASK-ND
6070      IF I AND MASK THEN M=M+J
6080      MASK=MASK*2
6090      J=J\2
6100    WEND
6110    IF M>I THEN GOSUB 6130
6120  NEXT I
6125 RETURN
6130  '
6140  XRT=XR(I)
6150  XIT=XI(I)
6160  XR(I)=XR(M)
6170  XI(I)=XI(M)
6180  XR(M)=XRT
6190  XI(M)=XIT
6200  RETURN
7000 '======================== WINDOW ROUTINE ================================
7010 '
7020  INPUT "SELECT WINDOW ( 1=RECTANGULAR  2=HAMMING  3=HANNING ) ";WIND
7030  ON WIND GOTO 7050,7100,7150
7040  '
7050  WIND$="RECTANGULAR"
7055  RETURN
7100  FOR NS=0 TO ND
7110    XR(NS)=XR(NS)*(.08+.46*(1-COS(TWOPI*NS/ND)))
7120    XI(NS)=XI(NS)*(.08+.46*(1-COS(TWOPI*NS/ND)))
7130  NEXT NS
7132  WIND$="HAMMING"
7135  RETURN
7150  FOR NS=0 TO ND
7160    XR(NS)=XR(NS)*.5*(1-COS(TWOPI*NS/ND))
7170    XI(NS)=XI(NS)*.5*(1-COS(TWOPI*NS/ND))
7180  NEXT NS
7185  WIND$="HANNING"
7190  RETURN
7200  '
8000 ' ================== CALCULATE TWIDDLE FACTOR COEFF ======================
8010  I=0                                        'INITIALIZE INDEX
8020  WHILE I-ND/2                               'DO ND/2 TIMES
8030    WRE(I)=COS(DELTA*I)                      'REAL PART OF TWIDDLE FACTOR
8040    WIM(I)=-SIN(DELTA*I)
8045    I=I+1
8050  WEND                                       '
8060  RETURN
```

APPENDIX C

TMS32020 DIRECT DFT PROGRAM

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157     16:29:22  05-19-86
                                                                PAGE 0001

0001          *                                              *
0002          *  DISCRETE FOURIER TRANSFORM PROGRAM          *
0003          *     FOR THE TEXAS INSTRUMENTS                *
0004          *       32020 MICROPROCESSOR.                  *
0005          *                                              *
0006          *       WRITTEN BY: JACK FRAZIER               *
0007          *                                              *
0008          * UNIVERSITY OF TENNESSEE SPACE INSTITUTE      *
0009          *         TULLAHOMA, TN  37388                 *
0010          *                                              *
0011          *          DATE: 01/29/86                      *
0012          *          REV:  03/03/86                      *
0013          *
0014          *=======================FOREWORD==============================
0015          *
0016          *        DFT PROGRAM FOR THE TI 32020 MICROPROCESSOR
0017          *
0018          *            N <=128 AND  MUST BE A POWER OF TWO
0019          *
0020          * SINGLE CYCLE MULTIPLY AND ACCUMULATE (MAC) USED TO COMPUTE DFT
0021          * THIS REQUIRES THE USE OF THE REPEAT COUNTER AND RESTRICTS THE
0022          * THE DATA TO ON-CHIP MEMORY BUT RESULTS IN VERY EFFICIENT CODE.
0023          *
0024          * ON-CHIP MEMORY HOLDS TABLE OF COEFFICIENTS (NEW TABLE FOR
0025          * EACH K) AND THE DATA TABLE.
0026          *
0027          * EXTERNAL MEMORY HOLDS A LARGE (LENGTH=N*N/2) TABLE OF COEFFICIENTS
0028          * FROM WHICH THE ON-CHIP TABLE IS LOADED . NO TEST FOR OVERUNNING
0029          * THE END IS NECESSARY.
0030          *
0031          *            THE DFT IS CALCULATED AS FOLLOWS:
0032          *              REAL DFT
0033          *              DFTR(K)=XR(N)*COS(KXMF)+XI(N)*SIN(KXMF)
0034          *              IMAG DFT
0035          *              DFTI(K)=XI(N)*COS(KXMF)-XR(N)*SIN(KXMF)
0036          *              WHERE
0037          *              INDEX THROUGH THE COEFF TABLES KXMF=K*MF
0038          *
0039          * THE COMPLEX DATA ARE READ INTO ON-CHIP MEMORY.
0040          * THE COS COEFF ARE LOADED INTO ON CHIP MEMORY.
0041          * THE FIRST TERM OF DFTR IS CALCULATED AND STORED. THEN THE
0042          * FIRST TERM OF DFTI. THE SIN COEFF ARE LOADED INTO ON-CHIP
0043          * MEMORY. THE SECOND TERM OF DFTR IS CALCULATED AND ADDED TO THE
0044          * FIRST. THE SECOND TERM OF DFTI IS CALCULATED AND ADDED TO THE
0045          * FIRST. REPEAT FOR EACH K. THE DATA IS SENT TO AN OUTPUT PORT.
0046          *
0047          * INDIRECT ADDRESSING IS USED THROUGHOUT FOR EASE OF
0048          * GENERATING RELOCATABLE CODE. SOME EFFICIENCY COULD BE
0049          * GAINED BY STORING DATA AT SPECIFIC PAGE LOCATIONS AND
0050          * LOADING THE PAGE POINTER REGISTER.
0051          *
0052          *====================BEGIN ASSEMBLY PROGRAM================
0053          *
0054                   IDT       'DFT128'
0056                   OPTION    XREF
0057          *
0058          *        =============DEFINE MACROS=========================
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                 PAGE 0002

0059            *
0060            *       ----------MACRO TO MUL AND ACC WITH REPEAT---------
0061            MPLY    $MACRO    P,D,N     MULTIPLY AND ACCUMULATE
0062                    ZAC                 CLEAR ACCUMULATOR
0063                    MPYK      0         CLEAR P REGISTER
0064                    LRLK      1,:N.S:   REG 1 POINTS TO REPEAT CT
0065                    LRLK      2,:D.S:   AR2 POINTS TO DATA TABLE
0066                    LARP      1         ARP TO REG 1
0067                    RPT       *,2       REPEAT N TIMES
0068                    MAC       :P.S:,*+  MUL DATA X COEFF, SUM
0069                    APAC                ADD P REG TO ACCUMULATOR
0070                    $END
0071            *       ----MACRO TO TABLE READ PROG TO DATA MEMORY, REPEAT----
0072            TABRD   $MACRO    P,D,N     TABLE READ, PROG TO DATA,
0073                    LRLK      1,:D.S:   AR1 POINTS TO DATA TABLE
0074                    LRLK      2,:N.S:   AR2 POINTS TO REPEAT COUNT
0075                    LALK      :P.S:     ACC POINTS TO PROG SPACE
0076                    LARP      2         ARP TO REG 2
0077                    RPT       *,1       REPEAT N TIMES
0078                    TBLR      *+        DO A TABLE READ
0079                    $END
0080            *       ---MACRO TO READ PROG TO DATA MEMORY, NO REPEAT-------
0081            TABRDN  $MACRO    P,D       TABLE READ, CON LOCATIONS
0082                    LALK      :P.S:     LOAD ACC WITH PROG ADD
0083                    LRLK      1,:D.S:   REG POINTS TO DATA
0084                    LARP      1         ARP TO REG 1
0085                    TBLR      *         READ PROG TO DATA RAM
0086                    $END
0087            *       ----------MACRO TO TABLE WRITE DATA TO PROG MEMORY-
0088            TABWRO  $MACRO    D,P,N     TABLE WRITE, INDEXES
0089                    LRLK      1,:D.S:   AR1 POINTS TO DATA TABLE
0090                    LRLK      2,:N.S:   AR2 POINTS TO REPEAT COUNT
0091                    LALK      :P.S:     ACC POINTS TO PROG SPACE
0092                    LARP      2         ARP TO REG 2
0093                    RPT       *,1       REPEAT N TIMES
0094                    TBLW      *0+       DO A TABLE WRITE
0095                    $END
0096            *       ----------MACRO TO STORE ACCUMULATOR---------------
0097            STORE   $MACRO    REG
0098                    LARP      :REG.V:
0099                    SACL      *+
0100                    SACH      *-
0101                    $END
0102            *       ----------MACRO TO ADD MEMORY TO ACCUMULATOR-------
0103            ADDM    $MACRO    REG
0104                    LARP      :REG.V:
0105                    ADD       *+
0106                    ADDH      *-
0107                    $END
0108            *       -----------MACRO TO DIVIDE POSITIVE INTEGERS-------
0109            DVD     $MACRO    NUM,DEN,QUO   DIVIDE (TOSS REMAINDER)
0110                    LRLK      0,:NUM.S:     AR1 POINTS TO NUM
0111                    LRLK      1,:DEN.S:     AR2 POINTS TO DEN
0112                    LRLK      2,:QUO.S:     AR3 POINTS TO QUO
0113                    LARP      0             ARP TO REG 2
0114                    LAC       *,0,1         LOAD ACC WITH NUM
0115                    RPTK      15            REPEAT 16 TIMES
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0003

0116                    SUBC    *                   CONDITIONAL SUBTRACT
0117                    LARP    2                   ARP TO REG 3
0118                    SACL    *                   STORE ACCUMULATOR QUO
0119                    $END
0120            *       ---MACRO TO DIVIDE ACCUMULATOR BY N=POWER OF 2----
0121            SFT     $MACRO
0122                    LRLK    1,TEMP      AR1 POINTS TO TEMP
0123                    LARP    1           ARP TO REG1
0124                    SACH    *           STORE HIGH ACCUMULATOR
0125                    BIT     *,0         COPY SIGN BIT TO TC BIT
0126                    ABS                 TAKE ABSOLUTE VALUE OF ACC
0127                    LRLK    1,SHFT      REG1 POINTS TO SHIFT
0128                    RPT     *           REPEAT SHFT TIMES
0129                    SFR                 SHIFT ACC RIGHT (DIVIDES BY N)
0130                    BBZ     $+3         BRANCH IF NOT NEGATIVE
0131                    NEG                 TWO'S COMPLEMENT ACC
0132                    NOP                 HERE IF POSITIVE
0133                    $END
0134            *
0135            STAB    $MACRO        SINE TABLE
0136                    DATA        0,1608,3212,4808
0137                    DATA        6393,7962,9512,11039
0138                    DATA        12539,14010,15446,16846
0139                    DATA        18204,19519,20787,22005
0140                    DATA        23170,24279,25329,26319
0141                    DATA        27245,28105,28898,29621
0142                    DATA        30273,30852,31356,31785
0143                    DATA        32137,32412,32609,32728
0144                    $END
0145            *
0146            CTAB    $MACRO          COSINE TABLE
0147                    DATA        32767,32728,32609,32412
0148                    DATA        32137,31785,31356,30852
0149                    DATA        30273,29621,28898,28105
0150                    DATA        27245,26319,25329,24279
0151                    DATA        23170,22005,20787,19519
0152                    DATA        18204,16846,15446,14010
0153                    DATA        12539,11039,9512,7962
0154                    DATA        6393,4808,3212,1608
0155                    DATA        0,-1608,-3212,-4808
0156                    DATA       -6393,-7962,-9512,-11039
0157                    DATA       -12539,-14010,-15446,-16846
0158                    DATA       -18204,-19519,-20787,-22005
0159                    DATA       -23170,-24279,-25329,-26319
0160                    DATA       -27245,-28105,-28898,-29621
0161                    DATA       -30273,-30852,-31356,-31785
0162                    DATA       -32137,-32412,-32609,-32728
0163                    DATA       -32767,-32728,-32609,-32412
0164                    DATA       -32137,-31785,-31356,-30852
0165                    DATA       -30273,-29621,-28898,-28105
0166                    DATA       -27245,-26319,-25329,-24279
0167                    DATA       -23170,-22005,-20787,-19519
0168                    DATA       -18204,-16846,-15446,-14010
0169                    DATA       -12539,-11039,-9512,-7962
0170                    DATA       -6393,-4808,-3212,-1608
0171                    DATA        0,1608,3212,4808
0172                    DATA        6393,7962,9512,11039
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0004

 0173                       DATA      12539,14010,15446,16846
 0174                       DATA      18204,19519,20787,22005
 0175                       DATA      23170,24279,25329,26319
 0176                       DATA      27245,28105,28898,29621
 0177                       DATA      30273,30852,31356,31785
 0178                       DATA      32137,32412,32609,32728
 0179                       $END
 0180             *
 0181             *====================INITIALIZATION=======================
 0182             *
 0183 0000                  PSEG
 0184      0000' START      EQU       $          START OF EXECUTABLE CODE
 0185 0000 CE05             CNFP                 CONFIGURE B0 AS PROG RAM
 0186                       TABRDN    ZERO,K     INIT K IN DATA RAM
 0001 0001 D001             LALK      ZERO       LOAD ACC WITH PROG ADD
      0002 00E4'
 0002 0003 D100             LRLK      1,K    REG POINTS TO DATA
      0004 0403
 0003 0005 5589             LARP      1            ARP TO REG 1
 0004 0006 5880             TBLR      *            READ PROG TO DATA RAM
 0187                       TABRDN    PNMAX,NMAX   INIT MAX N
 0001 0007 D001             LALK      PNMAX        LOAD ACC WITH PROG ADD
      0008 00E5'
 0002 0009 D100             LRLK      1,NMAX     REG POINTS TO DATA
      000A 0407
 0003 000B 5589             LARP      1          ARP TO REG 1
 0004 000C 5880             TBLR      *          READ PROG TO DATA RAM
 0188             *
 0189             *         ----------WRITE COEFF TABLE TO DATA RAM------------
 0190             *
 0191 000D D100             LRLK      1,DSIN     DATA TABLE ADDRESS
      000E 0680
 0192 000F D200             LRLK      2,TLEN     LENGTH OF TABLE
      0010 2020
 0193 0011 D001             LALK      PSIN       ACC=PROG ADDRESS
      0012 00E6'
 0194 0013 5589             LARP      1          ARP TO REG 1
 0195 0014 58AA  TR         TBLR      *+,2       COPY A WORD RAM, ARP=2
 0196 0015 D002             ADLK      1          INCREMENT ACCUMULATOR
      0016 0001
 0197 0017 FB99             BANZ      TR,*-,1    DEC AR3, BRANCH NOT=0
      0018 0014'
 0198             *
 0199             *         ----------READ N, COMPUTE N-1 & N/2----------------
 0200             *
 0201 0019 D100             LRLK      1,N        REG 1 POINTS TO NO SAMPLES
      001A 0400
 0202 001B 5589             LARP      1          ARP TO REG 1
 0203 001C 8080             IN        *,PA0      READ A WORD FROM IN PORT
 0204 001D 20A0             LAC       *+         LOAD ACC WITH N
 0205 001E D003             SBLK      1          SUB 1
      001F 0001
 0206 0020 6090             SACL      *-         STORE AT NM1 (N-1)
 0207 0021 20A0             LAC       *+         LOAD ACC WITH N
 0208 0022 CE19             SFR                  SHIFT RIGHT
 0209 0023 55A0             MAR       *+         INC AR1
 0210 0024 6080             SACL      *          STORE AT ND2 (N/2)
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0005

0211                *
0212                *         ----------READ DATA FROM FILE----------------------
0213                *
0214 0025 D100              LRLK      1,DXR      AR1 POINTS TO DATA DESTINATION
     0026 0000"
0215 0027 D200              LRLK      2,NM1      REG 2 POINTS TO REPEAT COUNT
     0028 0401
0216 0029 558A              LARP      2          ARP TO REG 2
0217 002A 4B89              RPT       *,1        REPEAT N TIMES
0218 002B 80A0              IN        *+,PA0     READ A WORD FROM IN PORT
0219 002C 558A              LARP      2          ARP TO REG 2
0220 002D 4B89              RPT       *,1        REPEAT N TIMES
0221 002E 80A0              IN        *+,PA0     READ A WORD FROM IN PORT
0222                *
0223                *         ----------COMPUTE NMAX/N---------------------------
0224                *
0225                          DVD       NMAX,N,MF      DIVIDE NMAX BY N
0001 002F D000              LRLK      0,NMAX         AR2 POINTS NMAX
     0030 0407
0002 0031 D100              LRLK      1,N            AR2 POINTS N
     0032 0400
0003 0033 D200              LRLK      2,MF           AR3 POINTS MF
     0034 0404
0004 0035 5588              LARP      0              ARP TO REG 0
0005 0036 2089              LAC       *,0,1          ACC=NMAX
0006 0037 CB0F              RPTK      15             REPEAT 16 TIMES
0007 0038 4780              SUBC      *              CONDITIONAL SUBTRACT
0008 0039 558A              LARP      2              ARP TO AR3
0009 003A 6080              SACL      *              STORE ACCUMULATOR
0226                *
0227                *         ----------FIND SHFT WHERE N=2**SHFT----------------
0228                *
0229 003B D000              LRLK      0,2        LOAD AR0 WITH 2
     003C 0002
0230 003D D100              LRLK      1,0        LOAD AR1 WITH 0
     003E 0000
0231 003F D200              LRLK      2,N        AR2 POINTS TO N
     0040 0400
0232 0041 D300              LRLK      3,SHFT     AR3 POINTS TO SHFT
     0042 0406
0233 0043 558A              LARP      2          ARP TO REG 2
0234 0044 2089              LAC       *,0,1      LOAD ACCUMULATOR WITH N
0235 0045 CE19  LOOP        SFR                  SHIFT ACCUMULATOR RIGHT
0236 0046 F5A0              BNZ       LOOP,*+    INC AR1, BRANCH IF NZ
0047 0045'
0237 0048 5590              MAR       *-
0238 0049 5590              MAR       *-         SUBTRACT 2 FROM COUNT
0239 004A 558B              LARP      3          ARP TO AR3
0240 004B 7180              SAR       1,*        STORE SHIFT COUNT ( 2**(SHFT+1)
0241                *
0242 004C D300              LRLK      3,DFTR     REG 3 POINTS TO REAL DFT OUTPUT
     004D 0480
0243 004E D400              LRLK      4,DFTI     REG 4 POINTS TO IMAG DFT OUTPUT
     004F 0580
0244 0050 5589              LARP      1          ARP TO REG 1
0245                *
0246                *=======================COMPUTE DFT TERMS==================
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0006

0247                *
0248                *                   GET NEW TABLE INDEX
0249                *
0250 0051 D000      DFT       LRLK      0,MF        AR0 POINTS TO MF
     0052 0404
0251 0053 D100                LRLK      1,K         AR1 POINTS TO K
     0054 0403
0252 0055 D200                LRLK      2,KXMF      AR2 POINTS TO KXMF
     0056 0405
0253 0057 3C88                LT        *,0         LOAD T REGISTER WITH
0254 0058 388A                MPY       *,2         MULTIPLY K BY MF
0255 0059 CE14                PAC                   STORE PRODUCT IN ACCU
0256 005A 6080                SACL      *           STORE COEFF TABLE IND
0257 005B 3080                LAR       0,*         LOAD AR0 WITH CONTENT
0258                *
0259                *         ----------ACTUAL DFT CALCULATIONS-----------------
0260                *
0261                          TABWR0    DCOS,W,NM1  GET COEFFICIENTS TO B0
0001 005C D100                LRLK      1,DCOS      AR1 POINTS TO DATA TABLE
     005D 06A0
0002 005E D200                LRLK      2,NM1       AR2 POINTS TO REPEAT COUNT
     005F 0401
0003 0060 D001                LALK      W           ACC POINTS TO PROG SPACE
     0061 FF00
0004 0062 558A                LARP      2           ARP TO REG 2
0005 0063 4B89                RPT       *,1         REPEAT N TIMES
0006 0064 59E0                TBLW      *0+         DO A TABLE WRITE
0262                          MPLY      W,DXR,NM1   1ST DFTR TERM
0001 0065 CA00                ZAC                   CLEAR ACCUMULATOR
0002 0066 A000                MPYK      0           CLEAR P REGISTER
0003 0067 D100                LRLK      1,NM1       REG 1 POINTS TO REPEAT CT
     0068 0401
0004 0069 D200                LRLK      2,DXR       AR2 POINTS TO DATA TABLE
     006A 0000"
0005 006B 5589                LARP      1           ARP TO REG 1
0006 006C 4B8A                RPT       *,2         REPEAT N TIMES
0007 006D 5DA0                MAC       W,*+        MUL DATA X COEFF, SUM
     006E FF00
0008 006F CE15                APAC                  ADD P REG TO ACCUMULATOR
0263                          STORE     3                 SAVE
0001 0070 558B                LARP      3
0002 0071 60A0                SACL      *+
0003 0072 6890                SACH      *-
0264                          MPLY      W,DXI,NM1   1ST DFTI TERM
0001 0073 CA00                ZAC                   CLEAR ACCUMULATOR
0002 0074 A000                MPYK      0           CLEAR P REGISTER
0003 0075 D100                LRLK      1,NM1       REG 1 POINTS TO REPEAT CT
     0076 0401
0004 0077 D200                LRLK      2,DXI       AR2 POINTS TO DATA TABLE
     0078 0080"
0005 0079 5589                LARP      1           ARP TO REG 1
0006 007A 4B8A                RPT       *,2         REPEAT N TIMES
0007 007B 5DA0                MAC       W,*+        MUL DATA X COEFF, SUM
     007C FF00
0008 007D CE15                APAC                  ADD P REG TO ACCUMULATOR
0265                          STORE     4                 SAVE
0001 007E 558C                LARP      4
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0007

 0002 007F 60A0          SACL    *+
 0003 0080 6890          SACH    *-
 0266                    TABWR0  DSIN,W,NM1   GET COEFFICIENTS TO B0
 0001 0081 D100          LRLK    1,DSIN       AR1 POINTS TO DATA TABLE
      0082 0680
 0002 0083 D200          LRLK    2,NM1        AR2 POINTS TO REPEAT COUNT
      0084 0401
 0003 0085 D001          LALK    W            ACC POINTS TO PROG SPACE
      0086 FF00
 0004 0087 558A          LARP    2            ARP TO REG 2
 0005 0088 4B89          RPT     *,1          REPEAT N TIMES
 0006 0089 59E0          TBLW    *0+          DO A TABLE WRITE
 0267                    MPLY    W,DXI,NM1    2ND DFTR TERM
 0001 008A CA00          ZAC                  CLEAR ACCUMULATOR
 0002 008B A000          MPYK    0            CLEAR P REGISTER
 0003 008C D100          LRLK    1,NM1        REG 1 POINTS TO REPEAT CT
      008D 0401
 0004 008E D200          LRLK    2,DXI        AR2 POINTS TO DATA TABLE
      008F 0080"
 0005 0090 5589          LARP    1            ARP TO REG 1
 0006 0091 4B8A          RPT     *,2          REPEAT N TIMES
 0007 0092 5DA0          MAC     W,*+         MUL DATA X COEFF, SUM
      0093 FF00
 0008 0094 CE15          APAC                 ADD P REG TO ACCUMULATOR
 0268                    ADDM    3            SUM 1ST & 2ND DFTR
 0001 0095 558B          LARP    3
 0002 0096 00A0          ADD     *+
 0003 0097 4890          ADDH    *-
 0269                    SFT                  DIVIDE ACCUMULATOR BY
 0001 0098 D100          LRLK    1,TEMP       AR1 POINTS TO TEMP
      0099 0408
 0002 009A 5589          LARP    1            ARP TO REG1
 0003 009B 6880          SACH    *            STORE HIGH ACCUMULATOR
 0004 009C 9080          BIT     *,0          COPY SIGN BIT TO TC BIT
 0005 009D CE1B          ABS                  TAKE ABSOLUTE VALUE OF ACC
 0006 009E D100          LRLK    1,SHFT       REG1 POINTS TO SHIFT
      009F 0406
 0007 00A0 4B80          RPT     *            REPEAT SHFT TIMES
 0008 00A1 CE19          SFR                  SHIFT ACC RIGHT (DIVIDES BY N)
 0009 00A2 F880          BBZ     $+3          BRANCH IF NOT NEGATIVE
      00A3 00A5'
 0010 00A4 CE23          NEG
 0011 00A5 5500          NOP                  HERE IF POSITIVE
 0270                    STORE   3                    SAVE
 0001 00A6 558B          LARP    3
 0002 00A7 60A0          SACL    *+
 0003 00A8 6890          SACH    *-
 0271 00A9 55A0          MAR     *+           INCREMENT REG 3
 0272 00AA 55A0          MAR     *+                "
 0273                    MPLY    W,DXR,NM1    2ND DFTI TERM
 0001 00AB CA00          ZAC                  CLEAR ACCUMULATOR
 0002 00AC A000          MPYK    0            CLEAR P REGISTER
 0003 00AD D100          LRLK    1,NM1        REG 1 POINTS TO REPEAT CT
      00AE 0401
 0004 00AF D200          LRLK    2,DXR        AR2 POINTS TO DATA TABLE
      00B0 0000"
 0005 00B1 5589          LARP    1            ARP TO REG 1
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0008

 0006 00B2 4B8A          RPT     *,2             REPEAT N TIMES
 0007 00B3 5DA0          MAC     W,*+            MUL DATA X COEFF, SUM
      00B4 FF00
 0008 00B5 CE15          APAC                    ADD P REG TO ACCUMULATOR
 0274 00B6 CE23          NEG                     CHANGE SIGN OF ACC
 0275                    ADDM    4               SUM 1ST & 2ND DFTI
 0001 00B7 558C          LARP    4
 0002 00B8 00A0          ADD     *+
 0003 00B9 4890          ADDH    *-
 0276                    SFT                     DIVIDE ACCUMULATOR BY
 0001 00BA D100          LRLK    1,TEMP          AR1 POINTS TO TEMP
      00BB 0408
 0002 00BC 5589          LARP    1               ARP TO REG1
 0003 00BD 6880          SACH    *               STORE HIGH ACCUMULATOR
 0004 00BE 9080          BIT     *,0             COPY SIGN BIT TO TC BIT
 0005 00BF CE1B          ABS                     TAKE ABSOLUTE VALUE OF ACC
 0006 00C0 D100          LRLK    1,SHFT          REG1 POINTS TO SHIFT
      00C1 0406
 0007 00C2 4B80          RPT     *               REPEAT SHFT TIMES
 0008 00C3 CE19          SFR                     SHIFT ACC RIGHT (DIVIDES BY N)
 0009 00C4 F880          BBZ     $+3             BRANCH IF NOT NEGATIVE
      00C5 00C7'
 0010 00C6 CE23          NEG
 0011 00C7 5500          NOP                     HERE IF POSITIVE
 0277                    STORE   4                       SAVE
 0001 00C8 558C          LARP    4
 0002 00C9 60A0          SACL    *+
 0003 00CA 6890          SACH    *-
 0278 00CB 55A0          MAR     *+              INCREMENT REG 4
 0279 00CC 55A9          MAR     *+,1                    "
 0280              *
 0281              *     ----------INCREMENT K AND REPEAT N/2 TIMES---------
 0282              *
 0283 00CD D100          LRLK    1,K             AR1 POINTS TO K
      00CE 0403
 0284 00CF 3088          LAR     0,*,0           AR0 LOADED WITH (K)
 0285 00D0 55A9          MAR     *+,1            INCREMENT AR0
 0286 00D1 708A          SAR     0,*,2           STORE INCREMENTED K
 0287              *
 0288 00D2 D200          LRLK    2,ND2           AR2 POINTS TO N/2
      00D3 0402
 0289 00D4 3189          LAR     1,*,1           LOAD REG 1 WITH N/2
 0290 00D5 CE52          CMPR    2               IS N/2 > K, TC=1 IF TRUE
 0291 00D6 F980          BBNZ    DFT             REPEAT IF TRUE
      00D7 0051'
 0292              *
 0293              *     ----------WRITE DATA TO OUTPUT PORT----------------
 0294              *
 0295 00D8 D100          LRLK    1,NM1           AR1 POINTS TO N-1
      00D9 0401
 0296 00DA 5589          LARP    1               ARP TO REG1
 0297 00DB D200          LRLK    2,DFTR          AR2 POINTS TO REAL DFT OUTPUT
      00DC 0480
 0298 00DD 4B8A          RPT     *,2             REPEAT N TIMES
 0299 00DE E1A0          OUT     *+,1            SEND DATA TO OUTPUT PORT 1
 0300              *
 0301 00DF 5589          LARP    1               ARP TO REG1
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0009

 0302 00E0 D200           LRLK      2,DFTI        AR2 POINTS TO IMAG DFT OUTPUT
      00E1 0580
 0303 00E2 4B8A           RPT       *,2           REPEAT N TIMES
 0304 00E3 E1A0           OUT       *+,1          SEND DATA TO OUTPUT PORT 1
 0305              *
 0306              *==================COEFF TABLES, PROGRAM MEMORY ROM=========
 0307              *
 0308 00E4 0000   ZERO     DATA      0
 0309 00E5 0080   PNMAX    DATA      128
 0310      00E6' PSIN     EQU       $
 0311                     STAB
 0001 00E6 0000           DATA       0,1608,3212,4808
      00E7 0648
      00E8 0C8C
      00E9 12C8
 0002 00EA 18F9           DATA       6393,7962,9512,11039
      00EB 1F1A
      00EC 2528
      00ED 2B1F
 0003 00EE 30FB           DATA       12539,14010,15446,16846
      00EF 36BA
      00F0 3C56
      00F1 41CE
 0004 00F2 471C           DATA       18204,19519,20787,22005
      00F3 4C3F
      00F4 5133
      00F5 55F5
 0005 00F6 5A82           DATA       23170,24279,25329,26319
      00F7 5ED7
      00F8 62F1
      00F9 66CF
 0006 00FA 6A6D           DATA       27245,28105,28898,29621
      00FB 6DC9
      00FC 70E2
      00FD 73B5
 0007 00FE 7641           DATA       30273,30852,31356,31785
      00FF 7884
      0100 7A7C
      0101 7C29
 0008 0102 7D89           DATA       32137,32412,32609,32728
      0103 7E9C
      0104 7F61
      0105 7FD8
 0312      0106' PCOS     EQU       $
 0313                     CTAB                1
 0001 0106 7FFF           DATA       32767,32728,32609,32412
      0107 7FD8
      0108 7F61
      0109 7E9C
 0002 010A 7D89           DATA       32137,31785,31356,30852
      010B 7C29
      010C 7A7C
      010D 7884
 0003 010E 7641           DATA       30273,29621,28898,28105
      010F 73B5
      0110 70E2
      0111 6DC9
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0010

 0004 0112 6A6D          DATA      27245,26319,25329,24279
      0113 66CF
      0114 62F1
      0115 5ED7
 0005 0116 5A82          DATA      23170,22005,20787,19519
      0117 55F5
      0118 5133
      0119 4C3F
 0006 011A 471C          DATA      18204,16846,15446,14010
      011B 41CE
      011C 3C56
      011D 36BA
 0007 011E 30FB          DATA      12539,11039,9512,7962
      011F 2B1F
      0120 2528
      0121 1F1A
 0008 0122 18F9          DATA      6393,4808,3212,1608
      0123 12C8
      0124 0C8C
      0125 0648
 0009 0126 0000          DATA      0,-1608,-3212,-4808
      0127 F9B8
      0128 F374
      0129 ED38
 0010 012A E707          DATA      -6393,-7962,-9512,-11039
      012B E0E6
      012C DAD8
      012D D4E1
 0011 012E CF05          DATA      -12539,-14010,-15446,-16846
      012F C946
      0130 C3AA
      0131 BE32
 0012 0132 B8E4          DATA      -18204,-19519,-20787,-22005
      0133 B3C1
      0134 AECD
      0135 AA0B
 0013 0136 A57E          DATA      -23170,-24279,-25329,-26319
      0137 A129
      0138 9D0F
      0139 9931
 0014 013A 9593          DATA      -27245,-28105,-28898,-29621
      013B 9237
      013C 8F1E
      013D 8C4B
 0015 013E 89BF          DATA      -30273,-30852,-31356,-31785
      013F 877C
      0140 8584
      0141 83D7
 0016 0142 8277          DATA      -32137,-32412,-32609,-32728
      0143 8164
      0144 809F
      0145 8028
 0017 0146 8001          DATA      -32767,-32728,-32609,-32412
      0147 8028
      0148 809F
      0149 8164
 0018 014A 8277          DATA      -32137,-31785,-31356,-30852
```

```
      014B 83D7
      014C 8584
      014D 877C
0019  014E 89BF          DATA    -30273,-29621,-28898,-28105
      014F 8C4B
      0150 8F1E
      0151 9237
0020  0152 9593          DATA    -27245,-26319,-25329,-24279
      0153 9931
      0154 9D0F
      0155 A129
0021  0156 A57E          DATA    -23170,-22005,-20787,-19519
      0157 AA0B
      0158 AECD
      0159 B3C1
0022  015A B8E4          DATA    -18204,-16846,-15446,-14010
      015B BE32
      015C C3AA
      015D C946
0023  015E CF05          DATA    -12539,-11039,-9512,-7962
      015F D4E1
      0160 DAD8
      0161 E0E6
0024  0162 E707          DATA    -6393,-4808,-3212,-1608
      0163 ED38
      0164 F374
      0165 F9B8
0025  0166 0000          DATA     0,1608,3212,4808
      0167 0648
      0168 0C8C
      0169 12C8
0026  016A 18F9          DATA     6393,7962,9512,11039
      016B 1F1A
      016C 2528
      016D 2B1F
0027  016E 30FB          DATA     12539,14010,15446,16846
      016F 36BA
      0170 3C56
      0171 41CE
0028  0172 471C          DATA     18204,19519,20787,22005
      0173 4C3F
      0174 5133
      0175 55F5
0029  0176 5A82          DATA     23170,24279,25329,26319
      0177 5ED7
      0178 62F1
      0179 66CF
0030  017A 6A6D          DATA     27245,28105,28898,29621
      017B 6DC9
      017C 70E2
      017D 73B5
0031  017E 7641          DATA     30273,30852,31356,31785
      017F 7884
      0180 7A7C
      0181 7C29
0032  0182 7D89          DATA     32137,32412,32609,32728
      0183 7E9C
```

```
      0184 7F61
      0185 7FD8
 0378                    CTAB            64
 0001 2086 7FFF          DATA      32767,32728,32609,32412
      2087 7FD8
      2088 7F61
      2089 7E9C
 0002 208A 7D89          DATA      32137,31785,31356,30852
      208B 7C29
      208C 7A7C
      208D 7884
 0003 208E 7641          DATA      30273,29621,28898,28105
      208F 73B5
      2090 70E2
      2091 6DC9
 0004 2092 6A6D          DATA      27245,26319,25329,24279
      2093 66CF
      2094 62F1
      2095 5ED7
 0005 2096 5A82          DATA      23170,22005,20787,19519
      2097 55F5
      2098 5133
      2099 4C3F
 0006 209A 471C          DATA      18204,16846,15446,14010
      209B 41CE
      209C 3C56
      209D 36BA
 0007 209E 30FB          DATA      12539,11039,9512,7962
      209F 2B1F
      20A0 2528
      20A1 1F1A
 0008 20A2 18F9          DATA      6393,4808,3212,1608
      20A3 12C8
      20A4 0C8C
      20A5 0648
 0009 20A6 0000          DATA      0,-1608,-3212,-4808
      20A7 F9B8
      20A8 F374
      20A9 ED38
 0010 20AA E707          DATA      -6393,-7962,-9512,-11039
      20AB E0E6
      20AC DAD8
      20AD D4E1
 0011 20AE CF05          DATA      -12539,-14010,-15446,-16846
      20AF C946
      20B0 C3AA
      20B1 BE32
 0012 20B2 B8E4          DATA      -18204,-19519,-20787,-22005
      20B3 B3C1
      20B4 AECD
      20B5 AA0B
 0013 20B6 A57E          DATA      -23170,-24279,-25329,-26319
      20B7 A129
      20B8 9D0F
      20B9 9931
 0014 20BA 9593          DATA      -27245,-28105,-28898,-29621
      20BB 9237
```

```
      20BC 8F1E
      20BD 8C4B
0015  20BE 89BF          DATA    -30273,-30852,-31356,-31785
      20BF 877C
      20C0 8584
      20C1 83D7
0016  20C2 8277          DATA    -32137,-32412,-32609,-32728
      20C3 8164
      20C4 809F
      20C5 8028
0017  20C6 8001          DATA    -32767,-32728,-32609,-32412
      20C7 8028
      20C8 809F
      20C9 8164
0018  20CA 8277          DATA    -32137,-31785,-31356,-30852
      20CB 83D7
      20CC 8584
      20CD 877C
0019  20CE 89BF          DATA    -30273,-29621,-28898,-28105
      20CF 8C4B
      20D0 8F1E
      20D1 9237
0020  20D2 9593          DATA    -27245,-26319,-25329,-24279
      20D3 9931
      20D4 9D0F
      20D5 A129
0021  20D6 A57E          DATA    -23170,-22005,-20787,-19519
      20D7 AA0B
      20D8 AECD
      20D9 B3C1
0022  20DA B8E4          DATA    -18204,-16846,-15446,-14010
      20DB BE32
      20DC C3AA
      20DD C946
0023  20DE CF05          DATA    -12539,-11039,-9512,-7962
      20DF D4E1
      20E0 DAD8
      20E1 E0E6
0024  20E2 E707          DATA    -6393,-4808,-3212,-1608
      20E3 ED38
      20E4 F374
      20E5 F9B8
0025  20E6 0000          DATA     0,1608,3212,4808
      20E7 0648
      20E8 0C8C
      20E9 12C8
0026  20EA 18F9          DATA     6393,7962,9512,11039
      20EB 1F1A
      20EC 2528
      20ED 2B1F
0027  20EE 30FB          DATA     12539,14010,15446,16846
      20EF 36BA
      20F0 3C56
      20F1 41CE
0028  20F2 471C          DATA     18204,19519,20787,22005
      20F3 4C3F
      20F4 5133
```

```
DFT128      32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    16:29:22  05-19-86
DIRECT FOURIER TRANSFORM/TI 32020                                  PAGE 0014

      20F5 55F5
 0029 20F6 5A82            DATA       23170,24279,25329,26319
      20F7 5ED7
      20F8 62F1
      20F9 66CF
 0030 20FA 6A6D            DATA       27245,28105,28898,29621
      20FB 6DC9
      20FC 70E2
      20FD 73B5
 0031 20FE 7641            DATA       30273,30852,31356,31785
      20FF 7884
      2100 7A7C
      2101 7C29
 0032 2102 7D89            DATA       32137,32412,32609,32728
      2103 7E9C
      2104 7F61
      2105 7FD8
 0379      2020  TLEN      EQU        $-PSIN        TABLE TOTAL LENGTH
 0380      0020  SLEN      EQU        PCOS-PSIN     LENGTH OF SIN-COS OFFSET
 0381            *
 0382            *         ==========PROGRAM MEMORY FOR COEFF=================
 0383            *
 0384 FF00                 AORG       65280  BLOCK 0 OF ON-CHIP PROGRAM MEMORY
 0385      FF00  W         EQU        $
 0386            *
 0387            *         ==========DATA STORAGE ALLOCATION==================
 0388            *
 0389 0000                 DSEG
 0390 0000      DXR        BSS        128           STORAGE FOR    DATA VALUES REAL
 0391 0080      DXI        BSS        128           STORAGE FOR    DATA VALUES IMAG
 0392            *
 0393 0400                 AORG       1024          EXTERNAL MEMORY
 0394 0400      N          BSS        1             FIRST RECORD = NO. SAMPLES
 0395 0401      NM1        BSS        1             N-1
 0396 0402      ND2        BSS        1             N/2
 0397 0403      K          BSS        1             COEFF TABLE INDEX
 0398 0404      MF         BSS        1             NMAX/N
 0399 0405      KXMF       BSS        1             K*MF = INDEX THROUGH COEFF TABLE
 0400 0406      SHFT       BSS        1             SHIFT COUNT
 0401 0407      NMAX       BSS        1             MAXIMUM N PROGRAM ALLOWS
 0402 0408      TEMP       BSS        1             TEMP STORAGE LOCATION
 0403 0480                 AORG       1152
 0404 0480      DFTR       BSS        256           DFT OUTPUT VALUES REAL
 0405 0580      DFTI       BSS        256           DFT OUTPUT VALUES IMAG
 0406 0680      DSIN       BSS        TLEN          COEFF TABLE STORAGE
 0407      06A0 DCOS       EQU        DSIN+SLEN
 0408            *
 0409                       END        START         STOP
NO ERRORS, NO WARNINGS
```

APPENDIX D

TMS32020 FFT PROGRAM

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157     15:56:55  10-25-86
                                                                PAGE 0001

0001          *
0002          *      FFT PROGRAM FOR THE 32020          *
0003          *         MICROPROCESSOR.                 *
0004          *                                         *
0005          *       WRITTEN BY: JACK FRAZIER          *
0006          *                                         *
0007          * UNIVERSITY OF TENNESSEE SPACE INSTITUTE *
0008          *         TULLAHOMA, TN  37388            *
0009          *                                         *
0010          *          DATE: 03/14/86                 *
0011          *          REV:  10/17/86                 *
0012          *                                         *
0013          *                                         *
0014          *====================TITLE,OPTION, OTHER DIRECTIVES=======
0015          *
0016                    IDT       'FFT1024'
0018                    OPTION    XREF
0019          *
0020          *         ------------DEFINE MACROS-----------------------
0021          *
0022          DVD       $MACRO    NUM,DEN,QUO         DIVIDE POS INT
0023                    LRLK      0,:NUM.S:           REG 1 POINTS TO NUM
0024                    LRLK      1,:DEN.S:           REG 2 POINTS TO DEN
0025                    LRLK      2,:QUO.S:           REG 3 POINTS TO QUO
0026                    LARP      0                   ARP TO REG 2
0027                    LAC       *,0,1               LOAD ACC WITH NUM
0028                    RPTK      15                  REPEAT 16 TIMES
0029                    SUBC      *                   CONDITIONAL SUBTRACT
0030                    LARP      2                   ARP TO REG 3
0031                    SACL      *                   STORE ACC QUOTIENT
0032                    $END
0033          *
0034          *
0035          STAB      $MACRO     SINE TABLE
0036                    DATA       0,201,402,603
0037                    DATA       804,1005,1206,1407
0038                    DATA       1608,1809,2009,2210
0039                    DATA       2410,2611,2811,3012
0040                    DATA       3212,3412,3612,3811
0041                    DATA       4011,4210,4410,4609
0042                    DATA       4808,5007,5205,5404
0043                    DATA       5602,5800,5998,6195
0044                    DATA       6393,6590,6786,6983
0045                    DATA       7179,7375,7571,7767
0046                    DATA       7962,8157,8351,8545
0047                    DATA       8739,8933,9126,9319
0048                    DATA       9512,9704,9896,10087
0049                    DATA       10278,10469,10659,10849
0050                    DATA       11039,11228,11417,11605
0051                    DATA       11793,11980,12167,12353
0052                    DATA       12539,12725,12910,13094
0053                    DATA       13279,13462,13645,13828
0054                    DATA       14010,14191,14372,14553
0055                    DATA       14732,14912,15090,15269
0056                    DATA       15446,15623,15800,15976
0057                    DATA       16151,16325,16499,16673
0058                    DATA       16846,17018,17189,17360
```

```
0059                    DATA    17530,17700,17869,18037
0060                    DATA    18204,18371,18537,18703
0061                    DATA    18868,19032,19195,19357
0062                    DATA    19519,19680,19841,20000
0063                    DATA    20159,20317,20475,20631
0064                    DATA    20787,20942,21096,21250
0065                    DATA    21403,21554,21705,21856
0066                    DATA    22005,22154,22301,22448
0067                    DATA    22594,22739,22884,23027
0068                    DATA    23170,23311,23452,23592
0069                    DATA    23731,23870,24007,24143
0070                    DATA    24279,24413,24547,24680
0071                    DATA    24811,24942,25072,25201
0072                    DATA    25329,25456,25582,25708
0073                    DATA    25832,25955,26077,26198
0074                    DATA    26319,26438,26556,26674
0075                    DATA    26790,26905,27019,27133
0076                    DATA    27245,27356,27466,27575
0077                    DATA    27683,27790,27896,28001
0078                    DATA    28105,28208,28310,28411
0079                    DATA    28510,28609,28706,28803
0080                    DATA    28898,28992,29085,29177
0081                    DATA    29268,29358,29447,29534
0082                    DATA    29621,29706,29791,29874
0083                    DATA    29956,30037,30117,30195
0084                    DATA    30273,30349,30424,30498
0085                    DATA    30571,30643,30714,30783
0086                    DATA    30852,30919,30985,31050
0087                    DATA    31113,31176,31237,31297
0088                    DATA    31356,31414,31470,31526
0089                    DATA    31580,31633,31685,31736
0090                    DATA    31785,31833,31880,31926
0091                    DATA    31971,32014,32057,32098
0092                    DATA    32137,32176,32213,32250
0093                    DATA    32285,32318,32351,32382
0094                    DATA    32412,32441,32469,32495
0095                    DATA    32521,32545,32567,32589
0096                    DATA    32609,32628,32646,32663
0097                    DATA    32678,32692,32705,32717
0098                    DATA    32728,32737,32745,32752
0099                    DATA    32757,32761,32765,32766
0100                    DATA    32767,32766,32765,32761
0101                    DATA    32757,32752,32745,32737
0102                    DATA    32728,32717,32705,32692
0103                    DATA    32678,32663,32646,32628
0104                    DATA    32609,32589,32567,32545
0105                    DATA    32521,32495,32469,32441
0106                    DATA    32412,32382,32351,32318
0107                    DATA    32285,32250,32213,32176
0108                    DATA    32137,32098,32057,32014
0109                    DATA    31971,31926,31880,31833
0110                    DATA    31785,31736,31685,31633
0111                    DATA    31580,31526,31470,31414
0112                    DATA    31356,31297,31237,31176
0113                    DATA    31113,31050,30985,30919
0114                    DATA    30852,30783,30714,30643
0115                    DATA    30571,30498,30424,30349
```

```
0116                    DATA    30273,30195,30117,30037
0117                    DATA    29956,29874,29791,29706
0118                    DATA    29621,29534,29447,29358
0119                    DATA    29268,29177,29085,28992
0120                    DATA    28898,28803,28706,28609
0121                    DATA    28510,28411,28310,28208
0122                    DATA    28105,28001,27896,27790
0123                    DATA    27683,27575,27466,27356
0124                    DATA    27245,27133,27019,26905
0125                    DATA    26790,26674,26556,26438
0126                    DATA    26319,26198,26077,25955
0127                    DATA    25832,25708,25582,25456
0128                    DATA    25329,25201,25072,24942
0129                    DATA    24811,24680,24547,24413
0130                    DATA    24279,24143,24007,23870
0131                    DATA    23731,23592,23452,23311
0132                    DATA    23170,23027,22884,22739
0133                    DATA    22594,22448,22301,22154
0134                    DATA    22005,21856,21705,21554
0135                    DATA    21403,21250,21096,20942
0136                    DATA    20787,20631,20475,20317
0137                    DATA    20159,20000,19841,19680
0138                    DATA    19519,19357,19195,19032
0139                    DATA    18868,18703,18537,18371
0140                    DATA    18204,18037,17869,17700
0141                    DATA    17530,17360,17189,17018
0142                    DATA    16846,16673,16499,16325
0143                    DATA    16151,15976,15800,15623
0144                    DATA    15446,15269,15090,14912
0145                    DATA    14732,14553,14372,14191
0146                    DATA    14010,13828,13645,13462
0147                    DATA    13279,13094,12910,12725
0148                    DATA    12539,12353,12167,11980
0149                    DATA    11793,11605,11417,11228
0150                    DATA    11039,10849,10659,10469
0151                    DATA    10278,10087,9896,9704
0152                    DATA    9512,9319,9126,8933
0153                    DATA    8739,8545,8351,8157
0154                    DATA    7962,7767,7571,7375
0155                    DATA    7179,6983,6786,6590
0156                    DATA    6393,6195,5998,5800
0157                    DATA    5602,5404,5205,5007
0158                    DATA    4808,4609,4410,4210
0159                    DATA    4011,3811,3612,3412
0160                    DATA    3212,3012,2811,2611
0161                    DATA    2410,2210,2009,1809
0162                    DATA    1608,1407,1206,1005
0163                    DATA    804,603,402,201
0164                    $END
0165            *
0166            CTAB    $MACRO  COSINE TABLE
0167                    DATA    32767,32766,32765,32761
0168                    DATA    32757,32752,32745,32737
0169                    DATA    32728,32717,32705,32692
0170                    DATA    32678,32663,32646,32628
0171                    DATA    32609,32589,32567,32545
0172                    DATA    32521,32495,32469,32441
```

```
0173                    DATA    32412,32382,32351,32318
0174                    DATA    32285,32250,32213,32176
0175                    DATA    32137,32098,32057,32014
0176                    DATA    31971,31926,31880,31833
0177                    DATA    31785,31736,31685,31633
0178                    DATA    31580,31526,31470,31414
0179                    DATA    31356,31297,31237,31176
0180                    DATA    31113,31050,30985,30919
0181                    DATA    30852,30783,30714,30643
0182                    DATA    30571,30498,30424,30349
0183                    DATA    30273,30195,30117,30037
0184                    DATA    29956,29874,29791,29706
0185                    DATA    29621,29534,29447,29358
0186                    DATA    29268,29177,29085,28992
0187                    DATA    28898,28803,28706,28609
0188                    DATA    28510,28411,28310,28208
0189                    DATA    28105,28001,27896,27790
0190                    DATA    27683,27575,27466,27356
0191                    DATA    27245,27133,27019,26905
0192                    DATA    26790,26674,26556,26438
0193                    DATA    26319,26198,26077,25955
0194                    DATA    25832,25708,25582,25456
0195                    DATA    25329,25201,25072,24942
0196                    DATA    24811,24680,24547,24413
0197                    DATA    24279,24143,24007,23870
0198                    DATA    23731,23592,23452,23311
0199                    DATA    23170,23027,22884,22739
0200                    DATA    22594,22448,22301,22154
0201                    DATA    22005,21856,21705,21554
0202                    DATA    21403,21250,21096,20942
0203                    DATA    20787,20631,20475,20317
0204                    DATA    20159,20000,19841,19680
0205                    DATA    19519,19357,19195,19032
0206                    DATA    18868,18703,18537,18371
0207                    DATA    18204,18037,17869,17700
0208                    DATA    17530,17360,17189,17018
0209                    DATA    16846,16673,16499,16325
0210                    DATA    16151,15976,15800,15623
0211                    DATA    15446,15269,15090,14912
0212                    DATA    14732,14553,14372,14191
0213                    DATA    14010,13828,13645,13462
0214                    DATA    13279,13094,12910,12725
0215                    DATA    12539,12353,12167,11980
0216                    DATA    11793,11605,11417,11228
0217                    DATA    11039,10849,10659,10469
0218                    DATA    10278,10087,9896,9704
0219                    DATA    9512,9319,9126,8933
0220                    DATA    8739,8545,8351,8157
0221                    DATA    7962,7767,7571,7375
0222                    DATA    7179,6983,6786,6590
0223                    DATA    6393,6195,5998,5800
0224                    DATA    5602,5404,5205,5007
0225                    DATA    4808,4609,4410,4210
0226                    DATA    4011,3811,3612,3412
0227                    DATA    3212,3012,2811,2611
0228                    DATA    2410,2210,2009,1809
0229                    DATA    1608,1407,1206,1005
```

```
0230                    DATA     804,603,402,201
0231                    DATA     0,-201,-402,-603
0232                    DATA    -804,-1005,-1206,-1407
0233                    DATA    -1608,-1809,-2009,-2210
0234                    DATA    -2410,-2611,-2811,-3012
0235                    DATA    -3212,-3412,-3612,-3811
0236                    DATA    -4011,-4210,-4410,-4609
0237                    DATA    -4808,-5007,-5205,-5404
0238                    DATA    -5602,-5800,-5998,-6195
0239                    DATA    -6393,-6590,-6786,-6983
0240                    DATA    -7179,-7375,-7571,-7767
0241                    DATA    -7962,-8157,-8351,-8545
0242                    DATA    -8739,-8933,-9126,-9319
0243                    DATA    -9512,-9704,-9896,-10087
0244                    DATA    -10278,-10469,-10659,-10849
0245                    DATA    -11039,-11228,-11417,-11605
0246                    DATA    -11793,-11980,-12167,-12353
0247                    DATA    -12539,-12725,-12910,-13094
0248                    DATA    -13279,-13462,-13645,-13828
0249                    DATA    -14010,-14191,-14372,-14553
0250                    DATA    -14732,-14912,-15090,-15269
0251                    DATA    -15446,-15623,-15800,-15976
0252                    DATA    -16151,-16325,-16499,-16673
0253                    DATA    -16846,-17018,-17189,-17360
0254                    DATA    -17530,-17700,-17869,-18037
0255                    DATA    -18204,-18371,-18537,-18703
0256                    DATA    -18868,-19032,-19195,-19357
0257                    DATA    -19519,-19680,-19841,-20000
0258                    DATA    -20159,-20317,-20475,-20631
0259                    DATA    -20787,-20942,-21096,-21250
0260                    DATA    -21403,-21554,-21705,-21856
0261                    DATA    -22005,-22154,-22301,-22448
0262                    DATA    -22594,-22739,-22884,-23027
0263                    DATA    -23170,-23311,-23452,-23592
0264                    DATA    -23731,-23870,-24007,-24143
0265                    DATA    -24279,-24413,-24547,-24680
0266                    DATA    -24811,-24942,-25072,-25201
0267                    DATA    -25329,-25456,-25582,-25708
0268                    DATA    -25832,-25955,-26077,-26198
0269                    DATA    -26319,-26438,-26556,-26674
0270                    DATA    -26790,-26905,-27019,-27133
0271                    DATA    -27245,-27356,-27466,-27575
0272                    DATA    -27683,-27790,-27896,-28001
0273                    DATA    -28105,-28208,-28310,-28411
0274                    DATA    -28510,-28609,-28706,-28803
0275                    DATA    -28898,-28992,-29085,-29177
0276                    DATA    -29268,-29358,-29447,-29534
0277                    DATA    -29621,-29706,-29791,-29874
0278                    DATA    -29956,-30037,-30117,-30195
0279                    DATA    -30273,-30349,-30424,-30498
0280                    DATA    -30571,-30643,-30714,-30783
0281                    DATA    -30852,-30919,-30985,-31050
0282                    DATA    -31113,-31176,-31237,-31297
0283                    DATA    -31356,-31414,-31470,-31526
0284                    DATA    -31580,-31633,-31685.-31736
0285                    DATA    -31785,-31833,-31880,-31926
0286                    DATA    -31971,-32014,-32057,-32098
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    15:56:55  10-25-86
FAST FOURIER TRANSFORM/TI 32020                                  PAGE 0006

 0287                       DATA      -32137,-32176,-32213,-32250
 0288                       DATA      -32285,-32318,-32351,-32382
 0289                       DATA      -32412,-32441,-32469,-32495
 0290                       DATA      -32521,-32545,-32567,-32589
 0291                       DATA      -32609,-32628,-32646,-32663
 0292                       DATA      -32678,-32692,-32705,-32717
 0293                       DATA      -32728,-32737,-32745,-32752
 0294                       DATA      -32757,-32761,-32765,-32766
 0295                       $END
 0296             *
 0297             *====================START EXECUTABLE CODE================
 0298             *
 0299 0000                  PSEG
 0300      0000'  START      EQU       $          START OF EXECUTABLE CODE
 0301      0400   NMAX       EQU       1024       MAXIMUM DATA SAMPLE SIZE
 0302 0000 CE01               DINT                 DISABLE INTERRUPTS
 0303 0001 CE04               CNFD                 CONFIGURE B0 AS DATA RAM
 0304 0002 C800               LDPK      0          PAGE POINTER TO PAGE 0
 0305             *
 0306             *          ----------READ DATA FROM FILE--------------------
 0307             *                     ----------GET N-------------------------
 0308 0003 8000"              IN        N,PA0      READ N=NO DATA SAMP FROM PORT
 0309 0004 2000"              LAC       N          GET N TO ACCUMULATOR
 0310 0005 D003               SBLK      1          SUBTRACT 1 FROM N
      0006 0001
 0311 0007 6001"              SACL      NM1        STORE N-1
 0312             *                     ----------READ IN REAL DATA-------------
 0313 0008 5589               LARP      1          ARP TO REG 1
 0314 0009 D100               LRLK      1,DXR      AR1 POINTS TO REAL DATA MEMORY
      000A 0200
 0315 000B 3201"              LAR       2,NM1      WORD COUNT FOR READING DATA
 0316 000C 80AA   RDR         IN        *+,PA0,2   READ A WORD FROM IN PORT
 0317 000D FB99               BANZ      RDR,*-,1   BRANCH UNTIL N WORDS ARE READ
      000E 000C'
 0318             *                     ----------READ IN IMAG DATA-------------
 0319 000F D100               LRLK      1,DXI      AR1 POINTS TO IMAG DATA MEMORY
      0010 0600
 0320 0011 3201"              LAR       2,NM1      WORD COUNT FOR READING DATA
 0321 0012 80AA   RDI         IN        *+,PA0,2   READ A WORD FROM IN PORT
 0322 0013 FB99               BANZ      RDI,*-,1   BRANCH UNTIL N WORDS ARE READ
      0014 0012'
 0323             *
 0324             *          ----------------INIT DATA RAM VARIABLES-----------
 0325             *
 0326 0015 2000"              LAC       N          LOAD ACC WITH (N)
 0327 0016 600F"              SACL      MASK       MASK=N    (FOR BIT REVERSE)
 0328 0017 CE19               SFR                  SHIFT RIGHT
 0329 0018 6005"              SACL      SPAN       SPAN=N/2
 0330 0019 CA01               LACK      1          ONE
 0331 001A 6004"              SACL      K          K=1    (K USED AS TABLE INDEX)
 0332 001B CA00               ZAC                  CLEAR ACCUMULATOR
 0333 001C 600D"              SACL      NIND       NIND=0 (INDEX FOR BIT REVERSE )
 0334 001D 600E"              SACL      TIND       TIND=0 (TEST INDEX FOR BIT REV)
 0335             *
 0336 001E D001               LALK      DXR        ACC = ADDRESS OF DATA TABLE
      001F 0200
 0337 0020 0000"              ADD       N          ADD N TO GET END OF DATA TABLE
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157     15:56:55  10-25-86
FAST FOURIER TRANSFORM/TI 32020                                  PAGE 0007

0338 0021 600A"          SACL      XREND     STORE AT XREND
0339              *
0340              *      ----------------FIND MUL FACTOR NMAX/N------------
0341              *
0342 0022 D001           LALK      NMAX      LOAD ACCUMULATOR WITH NMAX
     0023 0400
0343 0024 CB0F           RPTK      15        REPEAT 16 TIMES
0344 0025 4700"          SUBC      N         CONDITIONAL SUBTRACT
0345 0026 6002"          SACL      MF        SAVE QUOTIENT
0346 0027 6003"          SACL      KXMF      INDEX FOR TWIDDLE FACTOR TABLE
0347              *
0348              *      ---------------CREATE A BIT REVERSE INDEX TABLE---
0349              *
0350 0028 5589           LARP      1         ARP TO REG 1
0351 0029 D100           LRLK      1,RIND    REG 1 POINTS TO REV INDEX TABLE
     002A 0E00
0352 002B 200D" BEGIN    LAC       NIND      GET NORMAL INDEX
0353 002C 1000"          SUB       N         IF NORMAL INDEX=N THEN FINISHED
0354 002D F680           BZ        INITAB    FINISHED BIT REVERSE TABLE
     002E 004B'
0355              *                ----SET UP INDEX AND MASK BIT FOR TEST---
0356 002F CA00           ZAC                 CLEAR ACCUMULATOR
0357 0030 6080           SACL      *         CLEAR REVERSE INDEX
0358 0031 200F" REV      LAC       MASK      BIT POS IN REVERSE INDEX
0359 0032 CE19           SFR                 MOVE BIT POSITION RIGHT ONE BIT
0360 0033 F680           BZ        NEXIND    IF ZERO FINISHED
     0034 0041'
0361              *                ----------TEST LSB OF TEST INDEX---------
0362 0035 600F"          SACL      MASK      SAVE NEW BIT POSITION
0363 0036 9F0E"          BIT       TIND,15   TEST LSB OF TEST INDEX
0364 0037 F880           BBZ       NEXBIT    IF LSB ZERO FORGET IT
     0038 003C'
0365              *                ----------OTHERWISE SET THE BIT----------
0366 0039 200F"          LAC       MASK      GET BIT POS FOR REV INDEX
0367 003A 0080           ADD       *         SET BIT POS IN REVERSE INDEX
0368 003B 6080           SACL      *         SAVE REVERSE INDEX
0369              *                ----------SHIFT TIND RIGHT--------------
0370 003C 200E" NEXBIT   LAC       TIND      GET TEST INDEX
0371 003D CE19           SFR                 SHIFT RIGHT TO GET NEXT MSB
0372 003E 600E"          SACL      TIND      SAVE SHIFTED TIND
0373 003F FF80           B         REV       GO DO NEXT BIT
     0040 0031'
0374              *                ----------GO TO NEXT INDEX--------------
0375 0041 D001 NEXIND    LALK      1         ACC=1
     0042 0001
0376 0043 000D"          ADD       NIND      INCREMENT NORMAL INDEX
0377 0044 600D"          SACL      NIND      SAVE NEW INDEX
0378 0045 600E"          SACL      TIND      SAVE TO TEST INDEX
0379 0046 2000"          LAC       N         GET N
0380 0047 600F"          SACL      MASK      MASK=N
0381 0048 55A0           MAR       *+        GO TO NEXT ADD FOR NEW RIND
0382 0049 FF80           B         BEGIN     DO IT AGAIN
     004A 002B'
0383              *
0384              *      ----------WRITE COEFF TABLE TO DATA RAM-----------
0385              *
0386 004B D001 INITAB    LALK      PWI       ACC HOLDS PROGRAM ADDRESS
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    15:56:55  10-25-86
FAST FOURIER TRANSFORM/TI 32020                                  PAGE 0008

     004C 00FE'
 0387 004D D200          LRLK      2,WI       AR2 POINTS TO DATA ADDRESS
     004E 0C00
 0388 004F 558A          LARP      2          ARP TO REG 2
 0389 0050 CBFF          RPTK      255        REPEAT 256 TIMES
 0390 0051 58A0          TBLR      *+         READ DATA TABLE TO RAM
 0391 0052 D002          ADLK      256        ADD 256 TO ACCUMULATOR
     0053 0100
 0392 0054 CBFF          RPTK      255        REPEAT 256 TIMES
 0393 0055 58A0          TBLR      *+         READ DATA TABLE TO RAM
 0394              *
 0395 0056 D001          LALK      PWR        ACC HOLDS PROGRAM ADDRESS
     0057 02FE'
 0396 0058 D200          LRLK      2,WR       AR2 POINTS TO DATA ADDRESS
     0059 0A00
 0397 005A 558A          LARP      2          ARP TO REG 2
 0398 005B CBFF          RPTK      255        REPEAT 256 TIMES
 0399 005C 58A0          TBLR      *+         READ DATA TABLE TO RAM
 0400 005D D002          ADLK      256        ADD 256 TO ACCUMULATOR
     005E 0100
 0401 005F CBFF          RPTK      255        REPEAT 256 TIMES
 0402 0060 58A0          TBLR      *+         READ DATA TABLE TO RAM
 0403              *
 0404              *       ----------------COMPUTE FFT TERMS----------------
 0405              *
 0406 0061 CE07          SSXM                 SET SIGN-EXTENTION MODE
 0407 0062 5589          LARP      1          ARP TO REG 1
 0408 0063 2000" STAGE   LAC       N          LOAD ACC WITH (N)
 0409 0064 1004"         SUB       K          SUBTRACT N
 0410 0065 F280          BLEZ      BITREV     BRANCH IF L.E. ZERO
     0066 00C6'
 0411              *
 0412 0067 D001          LALK      DXR        ACC=DXR
     0068 0200
 0413 0069 6006"         SACL      XRTOP      XRTOP=DXR
 0414 006A 0005"         ADD       SPAN       ACC=DXR+SPAN
 0415 006B 6008"         SACL      XRMID      XRMID=DXR+SPAN
 0416 006C D001          LALK      DXI        ACC=DXI
     006D 0600
 0417 006E 6007"         SACL      XITOP      XITOP=DXI
 0418 006F 0005"         ADD       SPAN       ACC=DXI+SPAN
 0419 0070 6009"         SACL      XIMID      XIMID=DXI+SPAN
 0420              *                          ----------FLOCK OF BUTTERFLIES----------
 0421 0071 3106" FLOCK   LAR       1,XRTOP    AR1 POINTS TO XRTOP
 0422 0072 300A"         LAR       0,XREND    LOAD AR0 WITH XREND
 0423 0073 CE50          CMPR      0          COMPARE XRTOP TO XREND
 0424 0074 F980          BBNZ      NEXTS      BRANCH IF EQUAL TO NEXT STAGE
     0075 00BB'
 0425 0076 2005"         LAC       SPAN       LOAD ACC WITH SPAN
 0426 0077 600B"         SACL      COUNT      COUNT=SPAN
 0427              *       --------------------BUTTERFLY CALCULATIONS--------
 0428 0078 3005"         LAR       0,SPAN     OFFSET TO MID DATA POINT
 0429 0079 D200          LRLK      2,WR       REG2 POINTS TO COEFF TABLE
     007A 0A00
 0430 007B 3307"         LAR       3,XITOP    REG3 POINTS TOP IMAG DATA POINT
 0431 007C D400          LRLK      4,WI       REG4 POINTS TO COEFF TABLE
     007D 0C00
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157     15:56:55  10-25-86
FAST FOURIER TRANSFORM/TI 32020                                  PAGE 0009

0432 007E 200B" BFLY     LAC       COUNT     LOAD ACC WITH COUNT
0433 007F D003           SBLK      1         SUBTRACT 1
     0080 0001
0434 0081 600B"          SACL      COUNT     STORE DECREMENTED COUNT
0435 0082 F380           BLZ       NEXTF     BRANCH IF <0 TO NEXT FLOCK
     0083 00AF'
0436              *
0437              *                ----------BUTTERFLY REAL----------------
0438 0084 40E0           ZALH      *0+       ACC=(XRTOP)
0439 0085 48D0           ADDH      *0-       ACC=(XRTOP)+(XRMID)
0440 0086 CE19           SFR                 DIVIDE BY 2
0441 0087 680C"          SACH      THI       SAVE TEMPORARILY
0442 0088 40E0           ZALH      *0+       ACC=(XRTOP)
0443 0089 4480           SUBH      *         ACC=(XRTOP)-(XRMID)
0444 008A 68D0           SACH      *0-       SAVE IN-PLACE
0445 008B 400C"          ZALH      THI       ACC=(XRTOP)+(XRMID)
0446 008C 68EB           SACH      *0+,0,3   SAVE IN-PLACE
0447              *                ----------BUTTERFLY IMAG----------------
0448 008D 40E0           ZALH      *0+       ACC=(XITOP)
0449 008E 48D0           ADDH      *0-       ACC=(XITOP)+(XIMID)
0450 008F CE19           SFR                 DIVIDE BY 2
0451 0090 680C"          SACH      THI       SAVE TEMPORARILY
0452 0091 40E0           ZALH      *0+       ACC=(XITOP)
0453 0092 4480           SUBH      *         ACC=(XITOP)-(XIMID)
0454 0093 68D0           SACH      *0-       SAVE IN-PLACE
0455 0094 400C"          ZALH      THI       ACC=(XITOP)+(XIMID)
0456 0095 68E9           SACH      *0+,0,1   SAVE IN-PLACE
0457              *                -------MULTIPLY BY TWIDDLE FACTORS------
0458 0096 3003"          LAR       0,KXMF    INDEX FOR TWIDDLE TABLES
0459              *                ----------REAL FFT TERM-----------------
0460 0097 3C8A           LT        *,2       LOAD T REGISTER WITH XR
0461 0098 388B           MPY       *,3       MULTIPLY XR*WR
0462 0099 CA00           ZAC                 CLEAR ACCUMULATOR
0463 009A CE15           APAC                ADD P REGISTER TO ACC
0464 009B 3C8C           LT        *,4       LOAD T REGISTER WITH XI
0465 009C 3889           MPY       *,1       MULTIPLY XI*WI
0466 009D CE15           APAC                ADD XR*WR+XI*WI
0467 009E 680C"          SACH      THI       SAVE HIGH ACC (SCALED)
0468              *                ----------IMAG FFT TERM-----------------
0469 009F 3C8C           LT        *,4       LOAD T REGISTER WITH XR
0470 00A0 38EB           MPY       *0+,3     MULTIPLY XR*WI
0471 00A1 CA00           ZAC                 CLEAR ACCUMULATOR
0472 00A2 CE15           APAC                ADD P REGISTER TO ACCUMULATOR
0473 00A3 CE23           NEG                 NEGATE ACCUMULATOR
0474 00A4 3C8A           LT        *,2       LOAD T REGISTER WITH XI
0475 00A5 38EB           MPY       *0+,3     MULTIPLY XI*WR
0476 00A6 CE15           APAC                ADD P REG TO ACCUMULATOR
0477              *                ----------STORE FFT INTERMEDIATE TERMS--
0478 00A7 68A9           SACH      *+,0,1    STORE XI IN-PLACE
0479 00A8 400C"          ZALH      THI       RESTORE REAL TERM TO ACC
0480 00A9 68A0           SACH      *+        STORE XR IN-PLACE
0481 00AA 3005"          LAR       0,SPAN    RESTORE SPAN TO AR0
0482 00AB 55DB           MAR       *0-,3     POINT TO NEXT TOP DATA POINT
0483 00AC 55D9           MAR       *0-,1     POINT TO NEXT MID DATA POINT
0484 00AD FF80           B         BFLY      BRANCH FOR NEXT BUTTERFLY
     00AE 007E'
0485              *      ------------------NEXT FLOCK OF BUTTERFLIES------
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    15:56:55  10-25-86
FAST FOURIER TRANSFORM/TI 32020                                  PAGE 0010

 0486 00AF 2008" NEXTF    LAC       XRMID     LOAD ACC WITH (XRMID)
 0487 00B0 0005"           ADD       SPAN      ADD SPAN TO XRTOP
 0488 00B1 6006"           SACL      XRTOP     STORE NEW XRTOP
 0489 00B2 0005"           ADD       SPAN      ADD SPAN TO XRTOP
 0490 00B3 6008"           SACL      XRMID     STORE XRMID
 0491 00B4 2009"           LAC       XIMID     LOAD ACC WITH (XIMID)
 0492 00B5 0005"           ADD       SPAN      ADD SPAN TO XITOP
 0493 00B6 6007"           SACL      XITOP     STORE NEW XITOP
 0494 00B7 0005"           ADD       SPAN      ADD SPAN TO XITOP
 0495 00B8 6009"           SACL      XIMID     STORE AT XRMID
 0496 00B9 FF80            B         FLOCK     BRANCH TO FLOCK
      00BA 0071'
 0497              *       ----------GO TO NEXT STAGE----------
 0498 00BB 2104" NEXTS     LAC       K,1       LOAD ACC WITH (K)
 0499 00BC 6004"           SACL      K         STORE NEW K
 0500 00BD 3C02"           LT        MF        T REG=MUL FACTOR
 0501 00BE 3804"           MPY       K         MULTIPLY BY K
 0502 00BF CE14            PAC                 LOAD ACC WITH K X MF
 0503 00C0 6003"           SACL      KXMF      SAVE NEW KXMF
 0504 00C1 2005"           LAC       SPAN      LOAD ACC WITH (SPAN)
 0505 00C2 CE19            SFR                 DIVIDE SPAN BY 2
 0506 00C3 6005"           SACL      SPAN      STORE NEW SPAN
 0507 00C4 FF80            B         STAGE     BRANCH TO STAGE
      00C5 0063'
 0508              *
 0509              *==========BIT REVERSE ROUTINE FOR FFT OUTPUT======
 0510              *
 0511 00C6 D100    BITREV  LRLK      1,RIND    REG 1 POINTS TO THE REV INDEX
      00C7 0E00
 0512 00C8 D200            LRLK      2,0       REG 2 CONTAINS THE NORM INDEX
      00C9 0000
 0513 00CA D300            LRLK      3,DXR     POINTS TO THE DATA+NORM INDEX
      00CB 0200
 0514 00CC D400            LRLK      4,DXR     POINTS TO THE DATA+REV INDEX
      00CD 0200
 0515 00CE 2000"           LAC       N         LOOP COUNT
 0516 00CF 5589            LARP      1         ARP TO REG 1
 0517              *       ----------CHECK LOOP COUNT----------
 0518 00D0 D003    REVERS  SBLK      1         DECREMENT LOOP COUNT
      00D1 0001
 0519 00D2 F280            BLEZ      OUTPUT    FINISHED SHUFFLEING
      00D3 00EF'
 0520              *       ---------CHECK IF NORMAL INDEX > REVERSE INDEX----
 0521 00D4 308A            LAR       0,*,2     REG 0 NOW CONTAINS REV INDEX
 0522 00D5 CE52            CMPR      2         IF NORM INDEX > RIND THEN TC=1
 0523 00D6 F880            BBZ       PLUS      DONT SWITCH IF REV >= NORM
      00D7 00E7'
 0524              *       ----------SWAP REAL DATA----------
 0525 00D8 558C            LARP      4         ARP TO REG 4
 0526 00D9 55E0            MAR       *0+       ADD REVERSE INDEX TO BASE
 0527 00DA 548B            PSHD      *,3       SAVE X(RIND) TO STACK
 0528 00DB 548C            PSHD      *,4       SAVE X(NIND) TO STACK
 0529 00DC 7A8B            POPD      *,3       GET X(NIND) TO X(RIND)
 0530 00DD 7A80            POPD      *         GET X(RIND) TO X(NIND)
 0531              *       ----------SWAP IMAG DATA----------
 0532 00DE D000            LRLK      0,OFFSET  REG 0 = OFFSET TO IMAG DATA
      00DF 0400
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    15:56:55  10-25-86
FAST FOURIER TRANSFORM/TI 32020                                  PAGE 0011

 0533 00E0 55EC           MAR       *0+,4     REG 4 POINTS TO IMAG DATA)NORM)
 0534 00E1 55E0           MAR       *0+       REG 3 POINTS TO IMAG DATA (REV)
 0535 00E2 548B           PSHD      *,3       SAVE X(RIND) TO STACK
 0536 00E3 548C           PSHD      *,4       SAVE X(NIND) TO STACK
 0537 00E4 7A8B           POPD      *,3       GET X(NIND) TO X(RIND)
 0538 00E5 7A80           POPD      *         GET X(RIND) TO X(NIND)
 0539 00E6 55D0           MAR       *0-       POINTER, REAL DATA (NORM INDEX)
 0540              *      ------------------INCREMENT TO NEXT ELEMENT------
 0541 00E7 5589  PLUS     LARP      1         ARP TO REG 1
 0542 00E8 55AA           MAR       *+,2      INC AR1 FOR NEXT REV INDEX
 0543 00E9 55AB           MAR       *+,3      INC AR2 FOR NEXT NORMAL INDEX
 0544 00EA 55A9           MAR       *+,1      INC AR3 FOR NEXT TABLE ADDRESS
 0545 00EB D400           LRLK      4,DXR     REG 4 TO TABLE BASE ADDRESS
      00EC 0200
 0546 00ED FF80           B         REVERS
      00EE 00D0'
 0547              *
 0548              *=======================WRITE DATA TO OUTPUT PORT=========
 0549              *
 0550              *       ------------- REAL FFT OUTPUT------------
 0551 00EF 5589  OUTPUT   LARP      1         ARP TO REG1
 0552 00F0 D100           LRLK      1,DXR     AR1 POINTS TO REAL FFT OUTPUT
      00F1 0200
 0553 00F2 3200"          LAR       2,N       WORD COUNT
 0554 00F3 E1AA  OTR      OUT       *+,PA1,2  SEND DATA TO OUTPUT PORT 1
 0555 00F4 FB99           BANZ      OTR,*-,1  BRANCH UNTIL N WORDS SENT
      00F5 00F3'
 0556              *      --------------IMAG FFT OUTPUT------------
 0557 00F6 5589           LARP      1         ARP TO REG1
 0558 00F7 D100           LRLK      1,DXI     AR1 POINTS TO REAL FFT OUTPUT
      00F8 0600
 0559 00F9 3200"          LAR       2,N       WORD COUNT
 0560 00FA E1AA  OTI      OUT       *+,PA1,2  SEND DATA TO OUTPUT PORT 1
 0561 00FB FB99           BANZ      OTI,*-,1  BRANCH UNTIL N WORDS SENT
      00FC 00FA'
 0562 00FD CE1F  FIN      IDLE
 0563              *
 0564              *==================PROGRAM MEMORY FOR COEFF================
 0565      00FE' TBEG     EQU       $
 0566              *                  SINE TABLE IS HERE (TOO LONG TO LIST)
 0570      0200  TABLEN   EQU       $-TBEG
 0571              *                  COSINE TABLE IS HERE
 0575              *
 0576              *==================DATA MEMORY ALLOCATION==================
 0577              *
 0578 0000                 DSEG
 0579 0000        N        BSS       1         NUMBER SAMPLES
 0580 0001        NM1      BSS       1         N-1
 0581 0002        MF       BSS       1         MULTIPLICATION FACTOR=NMAX/N
 0582 0003        KXMF     BSS       1         PRODUCT OF K*MF
 0583 0004        K        BSS       1         INITIAL VALUE, TABLE INDEX
 0584 0005        SPAN     BSS       1         BUTTERFLY WING SPAN
 0585 0006        XRTOP    BSS       1         TOP OF BFLY (REAL DATA)
 0586 0007        XITOP    BSS       1         TOP OF BFLY (IMAG DATA)
 0587 0008        XRMID    BSS       1         MID POINT OF BFLY (REAL)
 0588 0009        XIMID    BSS       1         MID POINT OF BFLY (IMAG)
 0589 000A        XREND    BSS       1         POINTS TO END OF DATA TABLE
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157    15:56:55  10-25-86
FAST FOURIER TRANSFORM/TI 32020                                 PAGE 0012

 0590 000B        COUNT    BSS       1          HOLDS PRESENT COUNT FOR LOOP
 0591 000C        THI      BSS       1          TEMP STORAGE FOR HIGH ACC
 0592 000D        NIND     BSS       1          NORM INDEX FOR BIT REV ROUTINE
 0593 000E        TIND     BSS       1          TEST INDEX "       "         "
 0594 000F        MASK     BSS       1          MASK WORD FOR TEST BIT POS
 0595             *
 0596             *        ------------------------------------------------
 0597 0200                 AORG      512        ADVANCE LOCATION COUNTER TO 768
 0598 0200        DXR      BSS       1024       STORAGE FOR 16 DATA VALUES REAL
 0599 0600        DXI      BSS       1024       STORAGE FOR 16 DATA VALUES IMAG
 0600      0400   OFFSET   EQU       DXI-DXR    FOR USE IN BIT REVERSE ROUTINE
 0601 0A00        WR       BSS       512        STORAGE FOR COEFF TABLE
 0602 0C00        WI       BSS       512
 0603 0E00        RIND     BSS       1024       BIT REVERSE INDEX TABLE
 0604                      END       START      END OF ASSEMBLY
NO ERRORS, NO WARNINGS
```

```
FFT1024     32020 FAMILY MACRO ASSEMBLER  PC 1.0 85.157     15:56:55  10-25-86
LABEL     VALUE   DEFN   REFERENCES                                  PAGE 0013

BEGIN     002B'   0352   0382
BFLY      007E'   0432   0484
BITREV    00C6'   0511   0410
COUNT     000B"   0590   0426  0432  0434
DXI       0600    0599   0319  0416  0558  0600
DXR       0200    0598   0314  0336  0412  0513  0514  0545  0552  0600
FIN       00FD'   0562
FLOCK     0071'   0421   0496
INITAB    004B'   0386   0354
K         0004"   0583   0331  0409  0498  0499  0501
KXMF      0003"   0582   0346  0458  0503
MASK      000F"   0594   0327  0358  0362  0366  0380
MF        0002"   0581   0345  0500
N         0000"   0579   0308  0309  0326  0337  0344  0353  0379  0408  0515
                         0553  0559
NEXBIT    003C'   0370   0364
NEXIND    0041'   0375   0360
NEXTF     00AF'   0486   0435
NEXTS     00BB'   0498   0424
NIND      000D"   0592   0333  0352  0376  0377
NM1       0001"   0580   0311  0315  0320
NMAX      0400    0301   0342
OFFSET    0400    0600   0532
OTI       00FA'   0560   0561
OTR       00F3'   0554   0555
OUTPUT    00EF'   0551   0519
PLUS      00E7'   0541   0523
PWI       00FE'   0568   0386
PWR       02FE'   0573   0395
RDI       0012'   0321   0322
RDR       000C'   0316   0317
REV       0031'   0358   0373
REVERS    00D0'   0518   0546
RIND      0E00    0603   0351  0511
SPAN      0005"   0584   0329  0414  0418  0425  0428  0481  0487  0489  0492
                         0494  0504  0506
STAGE     0063'   0408   0507
START     0000'   0300   0604
TABLEN    0200    0570
TBEG      00FE'   0565   0570
THI       000C"   0591   0441  0445  0451  0455  0467  0479
TIND      000E"   0593   0334  0363  0370  0372  0378
WI        0C00    0602   0387  0431
WR        0A00    0601   0396  0429
XIMID     0009"   0588   0419  0491  0495
XITOP     0007"   0586   0417  0430  0493
XREND     000A"   0589   0338  0422
XRMID     0008"   0587   0415  0486  0490
XRTOP     0006"   0585   0413  0421  0488
```

VITA

Jackson W. Frazier, Jr. is originally from Evergreen, Alabama. He received his early education there and graduated from Evergreen High School in 1963. He entered Auburn University and received a Bachelor of Electrical Engineering degree there in the winter of 1968.

After graduation, he accepted a position with Honeywell Inc., Avionics Division, in St. Petersburg, Florida. While there he worked in the areas of inertial guidance and controls systems, airborne computers, and satellite navigation, both at the home plant and field locations.

The author came to The University of Tennessee Space Institute (UTSI) in late 1979 to accept a position in the Electrical Section of the Energy Conversion Program. He entered graduate school and received the Master of Science degree in Electrical Engineering in March 1987. He is presently working in instrumentation and controls at the coal-fired magnetohydrodynamic (MHD) generator research facility at UTSI.