

2

DAB FILE COPY

AD-A219 894

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

DTIC
ELECTE
MAR 29 1990
S E D

IMPLEMENTATION OF MULTI-FREQUENCY
MODULATION
ON AN INDUSTRY STANDARD COMPUTER

by

Terry K. Gantenbein

September 1989

Thesis Advisor:

P. H. Moose

Approved for public release; distribution is unlimited.

90

Unclassified

security classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified			1b Restrictive Markings		
2a Security Classification Authority			3 Distribution Availability of Report		
2b Declassification Downgrading Schedule			Approved for public release; distribution is unlimited.		
4 Performing Organization Report Number(s)			5 Monitoring Organization Report Number(s)		
6a Name of Performing Organization Naval Postgraduate School		6b Office Symbol (if applicable) 32	7a Name of Monitoring Organization Naval Postgraduate School		
6c Address (city, state, and ZIP code) Monterey, CA 93943-5000			7b Address (city, state, and ZIP code) Monterey, CA 93943-5000		
8a Name of Funding Sponsoring Organization		8b Office Symbol (if applicable)	9 Procurement Instrument Identification Number		
8c Address (city, state, and ZIP code)			10 Source of Funding Numbers		
			Program Element No	Project No	Task No
			Work Unit Accession No		
11 Title (include security classification) IMPLEMENTATION OF MULTI-FREQUENCY MODULATION ON AN INDUSTRY STANDARD COMPUTER					
12 Personal Author(s) Terry K. Gantenbein					
13a Type of Report Master's Thesis		13b Time Covered From To		14 Date of Report (year, month, day) September 1989	15 Page Count 94
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17 Cosati Codes			18 Subject Terms (continue on reverse if necessary and identify by block number)		
Field	Group	Subgroup	communications, implementation, multi-frequency modulation; data transfer;		
			computer computer links; computer communications; (KT)		
19 Abstract (continue on reverse if necessary and identify by block number)					
This report discusses the theory, design, implementation and testing of a personal computer-based Multi-Frequency Modulation (MFM) packet communications system. Transmitter, receiver programs provide software drivers for D/A and A/D boards and perform symbol encoding, modulating, demodulating and decoding. The design and construction of a polarity coincidence correlator for receiver packet synchronization is presented. Experimental results show that the implemented MFM communication system conforms to theoretical analysis with acceptable bit error. Results also show that MFM can be uniquely adapted to a specific channel. <i>Keywords:</i>					
20 Distribution Availability of Abstract			21 Abstract Security Classification		
☒ unclassified unlimited ☐ same as report ☐ DTIC users			Unclassified		
22a Name of Responsible Individual P. H. Moore			22b Telephone (include Area code) (408) 646-2838		22c Office Symbol 62ME

DD FORM 1473.84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

Implementation of Multi-Frequency Modulation
on an Industry Standard Computer

by

Terry K. Gantenbein
Lieutenant, United States Navy
B.S.O.E., Wayland Baptist University, 1982

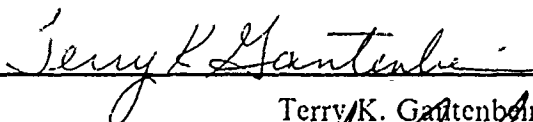
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1989

Author:

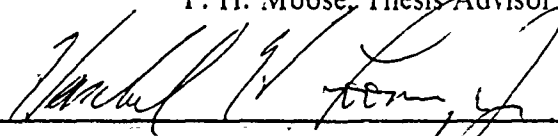


Terry K. Gantenbein

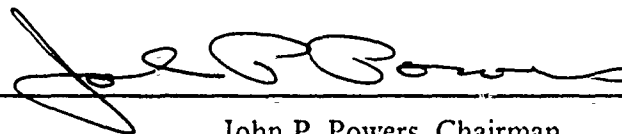
Adviser:



P. H. Moose, Thesis Advisor



H. H. Loomis, Jr., Second Reader



John P. Powers, Chairman,
Department of Electrical and Computer Engineering

ABSTRACT

This report discusses the theory, design, implementation and testing of a personal computer-based Multi-Frequency Modulation (MFM) packet communications system. Transmitter/receiver programs provide software drivers for D/A and A/D boards and perform symbol encoding, modulating, demodulating and decoding. The design and construction of a polarity coincidence correlator for receiver packet synchronization is presented. Experimental results show that the implemented MFM communication system conforms to theoretical analysis with acceptable bit error. Results also show that MFM can be uniquely adapted to a specific channel.

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



THESIS DISCLAIMER

The reader is cautioned that computer programs developed in this research may not have been exercised for all cases of interest. While every effort has been made, within the time available, to ensure that the programs are free of computational and logic errors, they cannot be considered validated. Any application of these programs without additional verification is at the risk of the user.

TABLE OF CONTENTS

I. INTRODUCTION	1
II. THEORY OF MULTI-FREQUENCY MODULATION	2
A. PACKET CONSTRUCTION	2
B. SIGNAL GENERATION AND DEMODULATION	4
C. PROPERTIES OF MFM	4
D. MODULATION	6
III. SYSTEM DEVELOPMENT	12
A. SYSTEM DESCRIPTION	12
B. BLOCK DESCRIPTION	12
1. Transmitter	12
2. Receiver	13
3. Synchronization	14
IV. SYSTEM IMPLEMENTATION	16
A. SIGNAL PARAMETERS	16
B. SOFTWARE	17
1. Transmitter	17
a. TRANSMIT	18
b. XMITMES	19
2. Receiver	20
a. RECEIVE	21
b. RECMES	22
C. HARDWARE	23
V. PERFORMANCE EVALUATION	25
A. SYSTEM FREQUENCY RESPONSE	25

B. SNR PERFORMANCE	28
VI. CONCLUSIONS AND FURTHER STUDY	32
APPENDIX A. DESIGN PARAMETERS	33
APPENDIX B. OPERATING INSTRUCTIONS	34
A. PRELIMINARIES	34
1. Hardware setup	34
a. System setup	34
b. DASH-16F switch setting:	34
c. Trigger Requirements	35
d. Power Supplies	35
2. Software setup	35
a. Transmitter	35
b. Receiver	36
B. MESSAGE TRANSMISSION PROCEDURE	36
APPENDIX C. TESTING PROCEDURES	37
A. RESPONSE TESTING PROCEDURE	37
B. SNR TESTING PROCEDURE	37
APPENDIX D. CORRELATOR SCHEMATIC	38
APPENDIX E. TRANSMIT	40
APPENDIX F. XMITMES	51
APPENDIX G. DMAINIT	58
APPENDIX H. DMASTOP	59

APPENDIX I. RECEIVE	60
APPENDIX J. RECMES	66
APPENDIX K. RESPONSE	72
APPENDIX L. STATISTICS	74
LIST OF REFERENCES	81
INITIAL DISTRIBUTION LIST	83

LIST OF TABLES

Table 1.	STORAGE REQUIREMENTS AND PROCESSING TIME ..	15
Table 2.	PHASE ERROR FROM A LINEAR CHANNEL	17
Table 3.	BIT ERRORS IN 2500 BITS TRANSMITTED VS BAUD TYPE AND SNR.	31
Table 4.	DESIGN PARAMETERS FOR A 1/15TH SECOND SIGNAL PACKET IN A 16-20KHZ BANDPASS CHANNEL.	33
Table 5.	SYNCHRONIZATION BAUD SYMBOL SEQUENCE	36

LIST OF FIGURES

Figure 1. MFM Signal Packet (after Ref. 1: p. 3.)	3
Figure 2. Data Structure (after Ref. 1: p. 9.)	5
Figure 3. Orthogonal tone spacing.	7
Figure 4. ACF of a white bandpass sequence (after Ref. 1: p. 16.)	8
Figure 5. QPSK signal constellation.	9
Figure 6. DQPSK encoding scheme.	10
Figure 7. MFM Communication System.	12
Figure 8. Transmitter Functional Block Diagram.	13
Figure 9. Receiver Functional Block Diagram.	14
Figure 10. TRANSMIT algorithm.	18
Figure 11. XMITMES algorithm.	20
Figure 12. RECEIVE algorithm.	21
Figure 13. Baud magnitude spectrum.	22
Figure 14. RECMES algorithm.	23
Figure 15. Polarity coincidence correlator.	24
Figure 16. Initial system response.	26
Figure 17. Improved system response.	27
Figure 18. System response with 2-bit synchronization delay.	28
Figure 19. Maximum SNR output.	29
Figure 20. SNR performance.	30
Figure 21. System interconnection diagram.	34
Figure 22. Trigger specifications.	35
Figure 23. Test equipment interconnection.	37

I. INTRODUCTION

Reliability and flexibility are two fundamental advantages of digital communications. They are achieved through the speed and processing power of digital integrated circuits. Multi-Frequency Modulation (MFM) uniquely harnesses this power found in the modern personal computer to encode, modulate, demodulate and decode digitally formatted information, for sample rates currently up to 100,000 samples per second. With computer-to-computer communication links implemented in a variety of mediums, such as wire, optical fiber, radio frequency, or acoustic, MFM readily adapts to a given medium and can emulate most existing signal modulation formats.

The focus of this thesis is the implementation of a MFM packet communications system using industry standard personal computers (PC). Because of the limited processing speed of the PC, modulation and demodulation of the packet is performed in longer than real time. During the course of this project, transmitter and receiver software were developed to process data for establishing a link between PCs using D/A and A/D data acquisition boards. A correlator was designed and constructed to synchronize the receiver to the transmitted packets using a synchronization baud at the beginning of the packets.

Analysis of the system phase response led to signal modifications which resulted in improved signal quality. Signal-to-noise performance was evaluated on the system for various packet constructions. The scope of the report encompasses the theory of MFM, a description of its specific development and implementation, and a summary of the evaluation of the system's performance.

II. THEORY OF MULTI-FREQUENCY MODULATION

A. PACKET CONSTRUCTION

The MFM signal set consists of "packets" of multiple tones which are amplitude and/or phase modulated. These tones are present simultaneously during a subinterval of the packet known as a baud. Packets can be located arbitrarily in the frequency spectrum and time as seen in Figure 1.

The follow definitions are used in MFM [Ref. 1: pp. 5-6]:

- T : Packet length in seconds
- ΔT : Baud length in seconds
- k_x : Baud length in number of samples
- ϕ_{lk} : Symbol set. ϕ_{lk} is the phase of the k^{th} tone in the l^{th} baud
- L : Number of baud per packet
- Δt : Time between samples in seconds
- $f_x = 1/\Delta t$: Sampling or clock frequency for D/A and A/D conversion in Hz
- $\Delta f = 1/\Delta T$: Frequency spacing between tones
- K : Number of MFM tones

To ensure the packet can be uniquely represented by the k_x samples, the Sampling Theorem states that f_x , the sampling frequency, must be greater than twice the highest frequency in the signal set [Ref. 2: pp. 46-56]. Since $f_x = k_x \Delta f$ and Δf is fixed by ΔT , the highest tone is $k_x/2 - 1$. Therefore, the signal set can consist of an arbitrary selection of tones from dc to $f_x/2 - \Delta f$.

The analog representation for the MFM signal during the l^{th} baud is

$$x_l(t) = \sum_{k=0}^{k_x/2} A_{lk} \cos(2\pi k \Delta f t + \phi_{lk}), \quad (l-1)\Delta T \leq t \leq l\Delta T. \quad (1)$$

The first baud begins at $t = 0$ and the last baud, L , ends at $T = L\Delta T$. Sampling $x_l(t)$ at $t = n\Delta t$, where n is discrete time, and substituting $\Delta f = 1/\Delta T$ and $\Delta t = \Delta T/k_x$, produces a sampled output sequence of

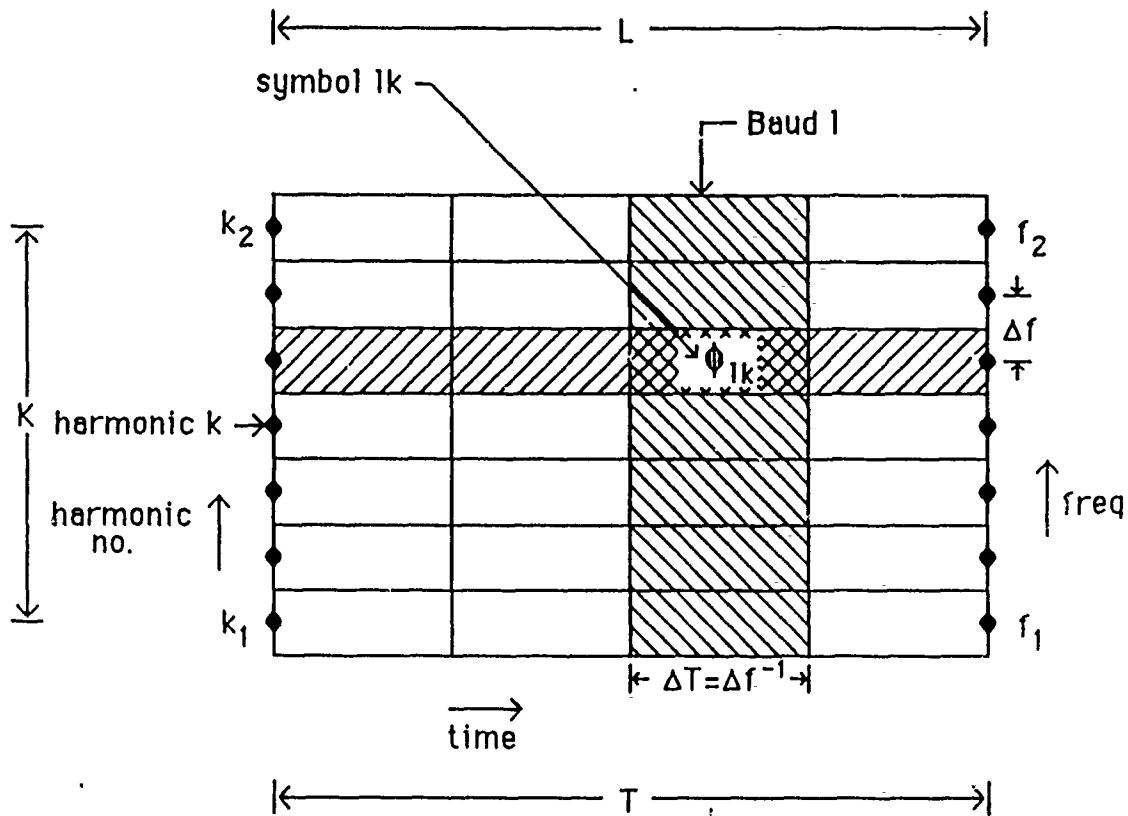


Figure 1. MFM Signal Packet (after Ref. 1: p. 3.)

$$x_l(n) = \sum_{k=0}^{k_x/2} A_{lk} \cos\left(\frac{2\pi kn}{k_x} + \phi_{lk}\right), \quad 0 \leq n \leq k_x, \quad (2)$$

a digital signal sequence of length k_x samples. The k_x point Discrete Fourier Transform (DFT) of $x_l(n)$ is

$$X_l(k') = \sum_{k=0}^{k_x/2} \frac{1}{2} k_x A_{lk} \{e^{j\phi_{lk}} \delta(k' - k) + e^{-j\phi_{lk}} \delta(k' - (k_x - k))\}, \quad 0 \leq k' \leq k_x - 1. \quad (3)$$

From (3), it can be seen that the upper half of $X_l(k')$ is the complex conjugate image of the lower half, which is consistent with the symmetry property of a real sequence [Ref. 2: p. 402].

B. SIGNAL GENERATION AND DEMODULATION

In the past, generation of $x_i(t)$ was attempted by constructing multiple phase lock loops, which were harmonically related. In December 1985, LCDR Deborah E. DeFrank, then at NPS, designed a coherent multifrequency synthesizer as a first step in producing $x_i(t)$ [Ref. 3]. This method proved to be difficult to implement due to the phase jitter in the phase lock loops and changes required in hardware in order to change the harmonic frequencies used in the tone set.

However, using the properties of DFT's, a baud of $x_i(n)$ can be generated by the host transmit computer by loading the first half of a complex valued array (0 to $k_x/2 - 1$) with the magnitude and phase of the tones to be included. To ensure $x_i(n)$ is real, the upper half, $(k_x/2)$ to $(k_x - 1)$, is loaded with the complex conjugate of the values in the first half of the array at the image harmonics. An example of the DFT of a real $x(n)$ having a period of $k_x = 16$ and 3 tones is shown in Figure 2. The Inverse Discrete Fourier Transform (IDFT) generates the real discrete sequence, $x_i(n)$, which is clocked out of the computer thru a D/A converter, at f_x samples per second. A signal packet is generated by L repetitions of the above process.

Demodulation of MFM is the inverse of the signal generation process. The analog signal, $x(t)$, is sampled at f_x samples per second and converted to digital format with an A/D converter. The sampled values are loaded into the real components of a k_x point complex array, with the imaginary component set to zero. A DFT is computed of the array to obtain the complex frequency representation of the sampled input. Since the upper half of the DFT is redundant information, only the lower half is retained for further processing.

C. PROPERTIES OF MFM

Several important properties of MFM signals are presented in this section.

(1) *Orthogonality of signal during a baud.* In continuous time

$$\int_0^{\Delta T} x_k(t)x_i(t)dt = \begin{cases} (1/2)A_k^2\Delta T & k = i \\ 0 & k \neq i \end{cases} \quad (4)$$

k	Re(X(k))	Im(X(k))
0	0	0
1	0	0
2	0	0
3	XR3	XI3
4	XR4	XI4
5	XR5	XI5
6	0	0
7	0	0
8	0	0
9	0	0
10	0	0
11	XR5	-XI5
12	XR4	-XI4
13	XR3	-XI3
14	0	0
15	0	0

Figure 2. Data Structure (after Ref. 1: p. 9.)

and in discrete time

$$\sum_{n=0}^{n=k_x-1} x_k(n)x_i(n) = \begin{cases} (1/2)A_k^2 k_x & k = i \\ 0 & k \neq i \end{cases} \quad (5)$$

The derivation of the orthogonality of harmonically related signals is well known [Ref. 4: pp. 152-154]. Two advantages of an orthogonal signal set are: 1) the noise, assuming additive white Gaussian noise (AWGN), affects each transmitted tone independently, thus simplifying statistical computations, and 2) no tone in a baud interferes with any of the other tones. This is shown by representing each tone in (1), neglecting phase and magnitude, as

$$s_k(t) = \cos(2\pi k \Delta f t) \text{ rect}(t/\Delta T) \quad (6)$$

where

$$\text{rect}(t/\Delta T) = \begin{cases} 1 & 0 \leq t \leq \Delta T \\ 0 & \text{otherwise} \end{cases} \quad (7)$$

The Fourier transform of $s_k(t)$ is

$$S_k(f) = \Delta T \text{sinc}(f - k\Delta f) \Delta T. \quad (8)$$

The spectrum of three adjacent tones, plotted in Figure 3, shows the peak of each tone coinciding with zero crossings of the spectrum of all other, thus minimizing their mutual interference.

(2) *Autocorrelation function (acf)*. Assuming $x(n)$ is periodic, the circular acf of (2) is

$$r_x(p) = \sum_{n=0}^{k_x-1} x(n)x(n \oplus p), \quad 0 \leq p \leq k_x - 1 \quad (9)$$

where $\oplus$ is a left circular shift. For a white bandlimited sequence with an even number of harmonics, the acf is given by

$$r_x(p) = (1/2)A^2 k_x \cos(2\pi k_0 p / k_x) \frac{\sin(\pi K p / k_x)}{\sin(\pi p / k_x)} \quad (10)$$

where $k_0 = (k_1 + k_2)/2$, the midband harmonic, and $K = k_1 - k_2 + 1$, the number of tones in the baud. Figure 4 shows the acf of a bandpass sequence with $k_x = 256$, $k_1 = 68$ and $k_2 = 83$ ($K = 16$, $k_p = 75.5$). These signal parameters conform to those of the synchronization baud, which will be discussed later. Note the peak of the acf occurs at $p(0)$, this feature is fundamental to synchronizing the receiver to the incoming packet.

(3) *Matched filter*. It is known that a matched filter maximizes the signal-to-noise ratio for additive white noise [Ref. 4: pp. 88-89]. The DFT of $x(n)$ at the frequency k is identical to the output of a filter matched to $x_k(n)$. This is due to the orthogonality property of the MFM signal and the linearity of the DFT [Ref. 1: pp. 17-21].

D. MODULATION

Modulation is the process of encoding the source information onto a bandpass signal with a carrier frequency f_c . This bandpass signal is called the *modulated* signal $s(t)$, and the baseband source signal is called the *modulating* signal $m(t)$ [Ref. 5: p. 204].

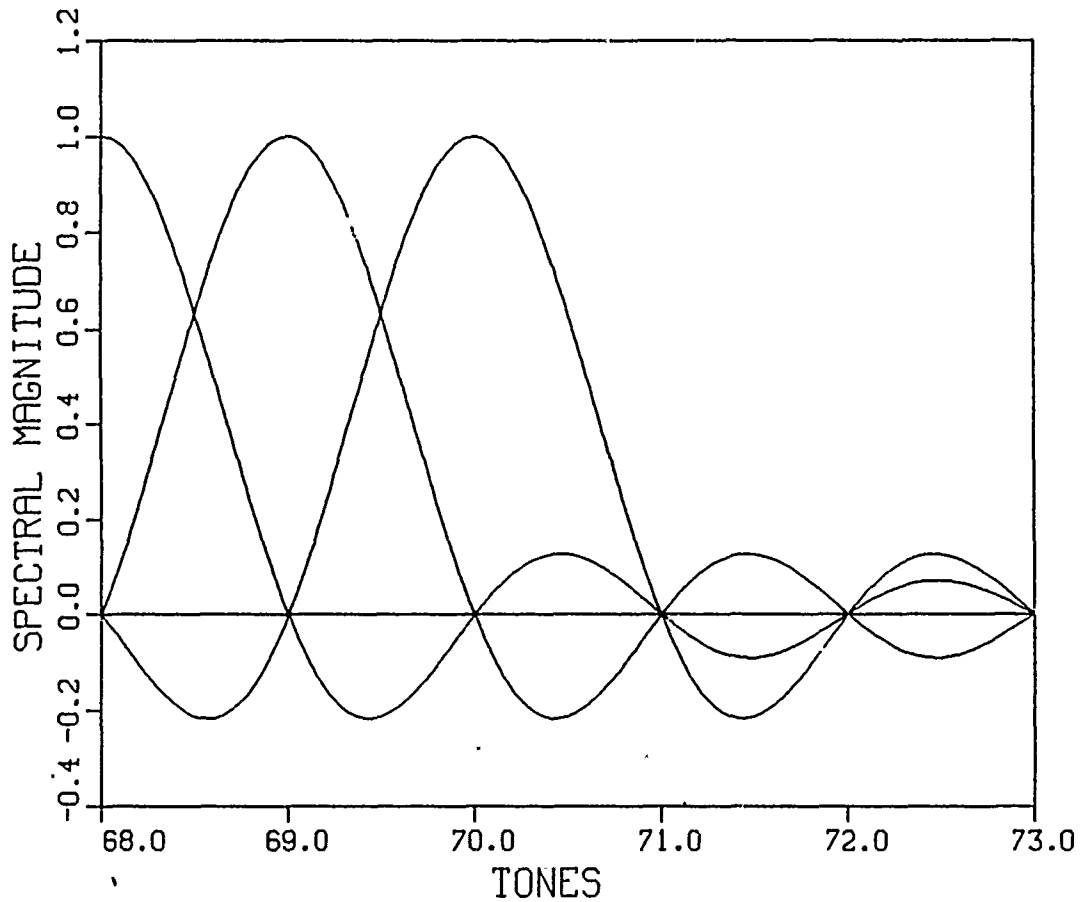


Figure 3. Orthogonal tone spacing.

The MFM signal can uniquely accomplish modulation in a number of ways. The signal $x_i(n)$, as defined in (2), can be modulated in amplitude, frequency, and phase, by translating the message into changes in A_{ik} , k and ϕ_{ik} respectively. Note, any combination of the modulation types is also possible, such as changing amplitude and phase to produce quadrature amplitude modulation (QAM). The signal, $s(t)$, is called a bandpass signal. However, MFM can be bandpass or baseband and through multiplication with a carrier frequency $x_i(n)$ can be translated to any frequency band desired. The signal sets considered in this thesis are bandpass and modulated using quadrature phase shift keying (QPSK) and differential quadrature phase shift keying (DQPSK).

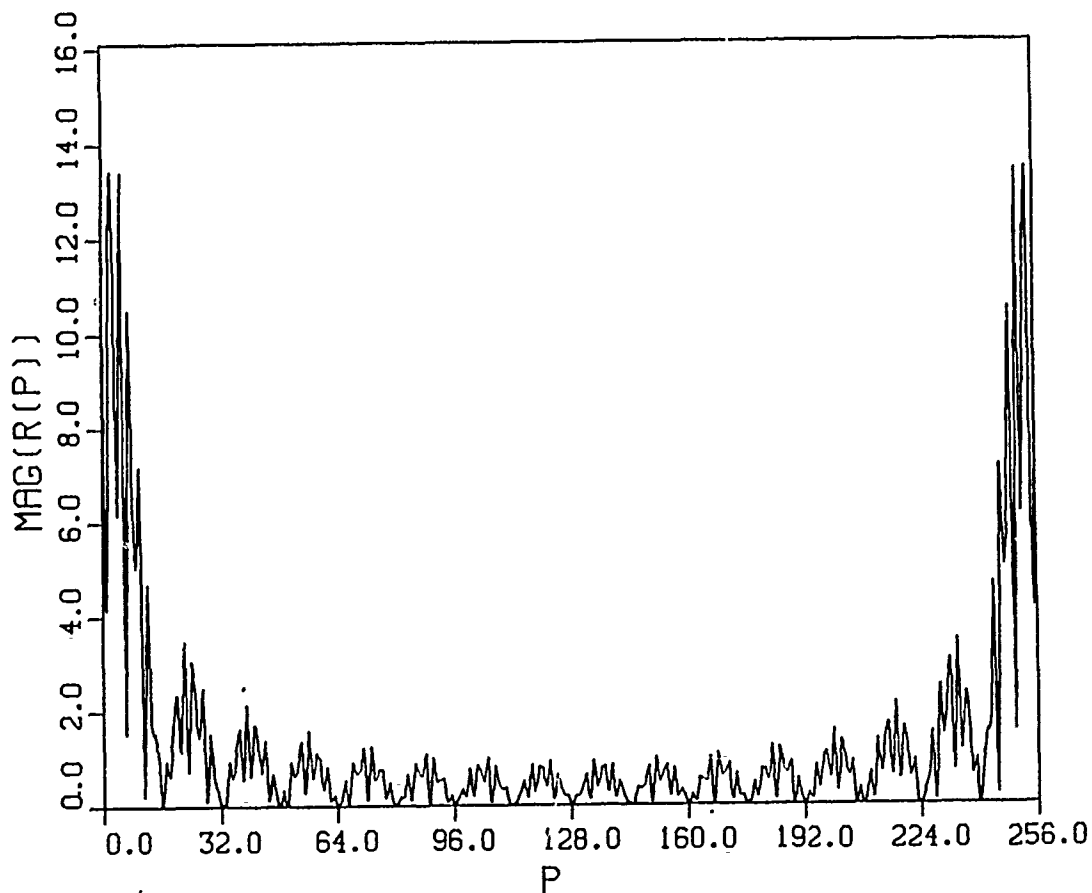


Figure 4. ACF of a white bandpass sequence (after Ref. 1: p. 16.)

(1) *QPSK*. Conventional QPSK converts a digital input into four modulation voltage levels (symbols) to determine the phase of the transmitter output. To minimize the probability of symbol error the phases are spaced at multiples of $\pi/2$. A plot of the complex envelope of one tone of $x_i(n)$ is shown in Figure 5. The angle ϕ can take on the values of $\pm \pi/4$ and $\pm 3\pi/4$.

Encoding MFM with QPSK is accomplished by loading the complex frequency domain array with the appropriate phase information. For example, if the digital input is '0110...'. The first symbol, '01', would be loaded into the frequency bin, k , as $\text{Re}[X(k)] = -A_k/\sqrt{2}$ and $\text{Im}[X(k)] = A_k/\sqrt{2}$, where A_k is the amplitude of tone k . Likewise, the second symbol '10' would be loaded into the second bin, $k+1$, as $\text{Re}[X(k+1)] = -A_{k+1}/\sqrt{2}$ and

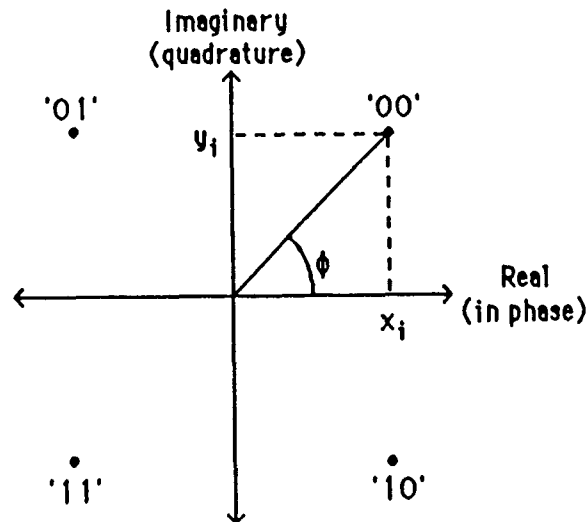


Figure 5. QPSK signal constellation.

$\text{Im}[X(k+1)] = A_{k+1}/\sqrt{2}$. This would continue until all the tones in the baud were filled or there were no more symbols. As mentioned earlier, the discrete time domain signal is produced by taking the IDFT of the complex frequency array. Each successive baud is encoded similarly.

Decoding QPSK directly into bits is accomplished easily as follows. Assuming a coherent receiver, decoding requires evaluating the polarity of the real and imaginary components of each frequency bin. Notice in Figure 5 that the symbol mapping uses Gray encoding. This reduces the probability of bit error because errors caused by AWGN are likely to occur when the adjacent symbol is selected for the transmitted symbol; thus, the symbol error will contain only one bit error. Gray encoding also allows decoding straight into bits, with the right bit determined by the polarity of the real component and the left bit by the polarity of the imaginary component. The digital signal is obtained by successively decoding each tone of each baud.

(2) *DQPSK*. QPSK requires strict phase coherent regeneration of the sampling frequency to eliminate phase ambiguity. This results in a complex synchronization design or distribution of the sampling clock frequency to each receiver. DQPSK provides a practical solution to the phase uncertainty problem

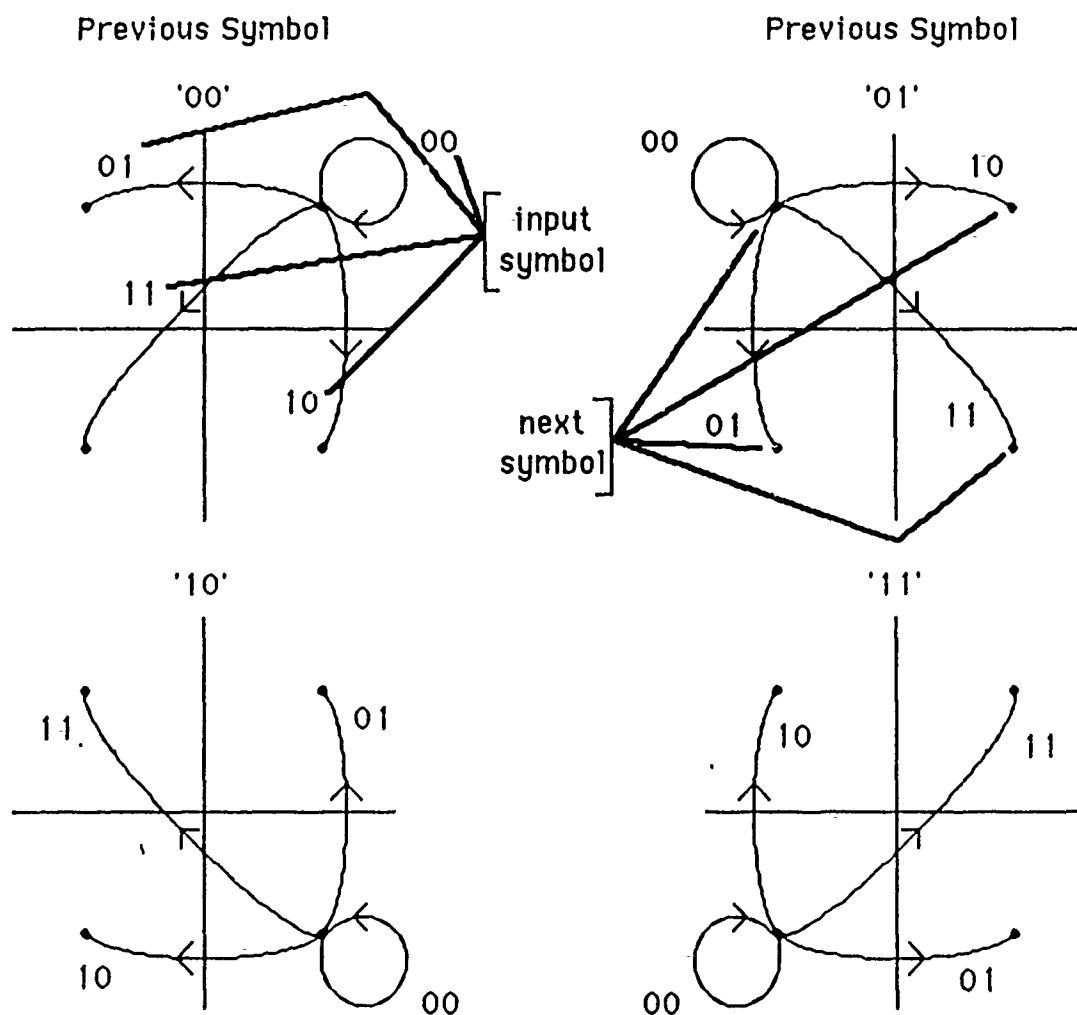


Figure 6. DQPSK encoding scheme.

but at a cost of approximately 2.3 dB more in SNR in order to obtain the same probability of bit error as coherent QPSK [Ref. 6: p. 442]. Encoding MFM with DQPSK is similar to QPSK; however, DQPSK translates the original symbol set into a second "differential" symbol set, which is then encoded as QPSK. Translated symbols are generated based on the input symbol and the previous translated symbol. Figure 6 shows this translation. Notice that, regardless of the previous symbol, an input of '00' generates a new symbol in the same quadrant as the previous symbol. An input of '01' rotates the new symbol $+\pi/2$ radians from the previous, '11' rotates π radians, and '10' rotates $-\pi/2$. In the receiver,

decoding is performed by determining the phase difference between successive pairs of tones, using complex arithmetic. The phase difference, $\Delta\phi$, is found from

$$e^{j\Delta\phi} = e^{j\phi_i}(e^{j\phi_{i-1}})^* = e^{j\phi_i} e^{-j\phi_{i-1}} = e^{j(\phi_i - \phi_{i-1})}, \quad (11)$$

and in the absence of noise will be 0 , $\pm \pi/2$, or π radians. To realign the complex signal to the original constellations, $e^{j\Delta\phi}$ is rotated by $+\pi/4$ radians and the in-phase and quadrature components are decoded as with QPSK.

III. SYSTEM DEVELOPMENT

Multi-Frequency Modulation joins the basic methods of digital communications theory and signal processing. The theory behind MFM is not new [Ref. 7], however, the ability to implement it inexpensively is linked to the recent development of inexpensive DSP chips for personal computers. The advent of packet switching in data communications also makes MFM a preferred choice for the MODEM because of its "packet" like format.

A. SYSTEM DESCRIPTION

An MFM communications system is shown in Figure 7. Information, denoted by $m(n)$, is modulated into a frequency band that will propagate over the available channel. The receiver converts the noise-corrupted signal, $r(t)$, to an estimate of the source information, $\tilde{m}(n)$. Theoretically, $m(n)$ is any signal that can be represented in a digital format. To be practically useful, the conversion process must be real time. However, due to signal conversion speeds available for this research project, the input information had to be restricted to stored data files. The ideal transmitter output, $x(t)$, is described for the l^{th} baud by (1).

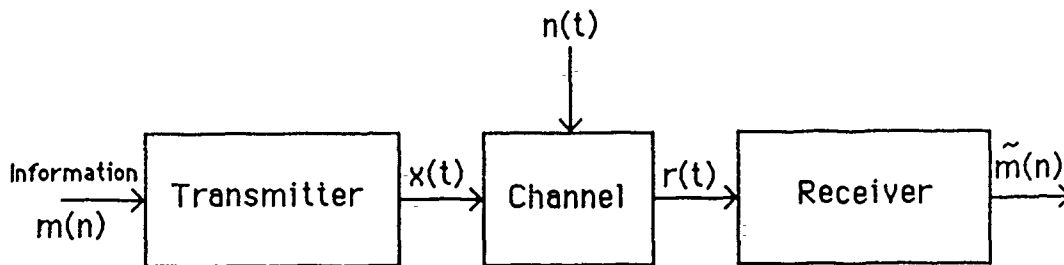


Figure 7. MFM Communication System.

B. BLOCK DESCRIPTION

1. Transmitter

The transmitter is subdivided into functional blocks, shown in Figure 8. As mentioned above, $m(n)$ has been digitized and stored in a file. Therefore, the input is a string of binary digits. Processing is performed on a baud-by-baud

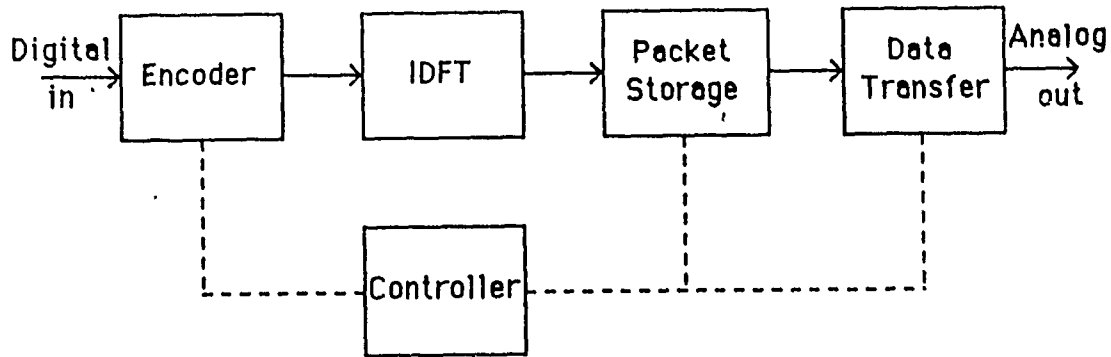


Figure 8. Transmitter Functional Block Diagram.

basis, until the end of the data file, or maximum packet length has been obtained, whichever comes first.

The encoder converts input symbols into complex values stored in the frequency domain array. The value of the symbols depends on the type of modulation. For example, QPSK symbols are two bits long and are encoded as previously discussed. The discrete signal, produced by computing the IDFT of the complex-valued frequency domain array, is loaded into the packet storage area. The controller determines the parameters of the packet based on the modulation type, baud, and message size; it then sequences the input data through the transmitter one baud at a time. Once the message has been processed, the entire stored digital signal, $x(n)$, is transferred out at the selected rate, f_x samples per second, through a D/A converter. Depending on the channel's frequency response, filtering of the output may be desired to remove the high frequency amplitude discontinuities introduced by the D/A converter.

2. Receiver

The receiver, shown in Figure 9, demodulates the MFM signal by reversing the transmitter process. Data acquisition is the process that samples the analog signal at f_x samples per second and converts it to digital format with an A/D converter. Though not shown in Figure 9, filtering of the input is recommended to bandlimit input noise and to reject out-of-band interference. The converted data sequence is stored until all data is acquired. Again, this is due to

time limitations in the signal processing algorithms available in the system used in this research task. The stored real values are accessed one baud at a time to perform a k_x point DFT. The first half of the resulting complex values are decoded to obtain the amplitude and phase modulation information. As in the transmitter, the controller sequences the data through the system.

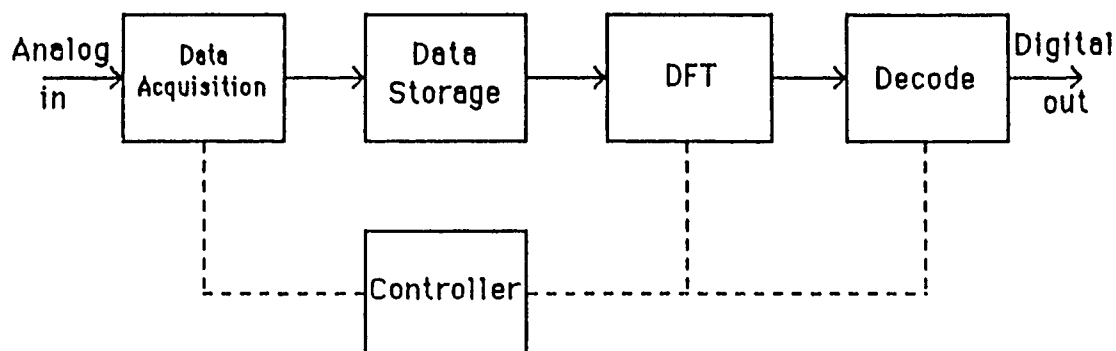


Figure 9. Receiver Functional Block Diagram.

Notice the delay from input to output is the time it takes to acquire the entire packet and process the first baud. In contrast, real time processing must complete the k_x point DFT in the same or less time than it takes to fill up one baud length buffer. Processing can alternate between buffers, and data will flow continuously through the system at the sampling rate. The storage requirements and processing time using $f_x = 61440$ Hz, $L = 30$, and $k_x = 1024$ samples are given in Table 1 for a system with a real time and non-real time DFT capability and for the non-real time system used in this research. The AT implementation processing time highlights the need for a high speed vector processor to perform the DFT computations.

3. Synchronization

In digital communications, various degrees of synchronization are required. Typical degrees are:

- Carrier synchronization,
- Bit or symbol synchronization,
- Word or frame synchronization.

Table 1. STORAGE REQUIREMENTS AND PROCESSING TIME

	Real time	Non real time	AT implementation
Storage required (# of samples)	2048	30720	30720
Time required for processing 1024 samples (msec)	≤ 16.7	> 16.7	10×10^3

These have slightly different meanings and forms depending on the system. For the MFM system, carrier and frame synchronization may be required.

Carrier synchronization is required if the modulation scheme is coherent, such as QPSK. This means generating an f_x at the receiver in frequency and phase coherence with the transmitted f_x . Though the packet does not contain the f_x harmonic, every tone is harmonically related to it. The packet can be constructed to have a pilot tone separated from the modulation tones from which f_x can be derived. For example, the modulation tones could range from k_i to k_{i+j} , giving a band of $j + 1$ consecutive tones. The pilot tone, k_p , would then be placed several tones away to ensure minimum band interference from the message during its extraction.

As mentioned previously, MFM, as implemented, is in the form of a transmission packet. To acquire the packet the receiver must know when to start sampling. This is accomplished by frame synchronization. Typically, unique words are inserted to mark the start of each frame. In MFM, the unique word is called a "synchronization baud", and it is added at the beginning of each packet. This baud is generated similarly to other bauds, except the tones and phases are predetermined. Acquisition of the received signal starts after successful detection of the synchronization baud. [Ref. 5: pp. 511-512, 8: pp. 293-295]

IV. SYSTEM IMPLEMENTATION

In the previous chapters the tools and framework for developing MFM packet signals were established. One actual realization of this form of communications will now be presented. Though the IBM Personal Computer (PC) was used, MFM can be implemented in a variety of ways. The PC, however, is ideally suited for MFM implementation for the following reasons:

- Increased I/O channel maximum throughput rate of approximately 100 KHz using direct memory access (DMA).
- Digital signal processing allows encoding, decoding, modulation, demodulation, and channel equalization.
- Signal processing algorithms are available in high level languages.
- Packet construction is easily modified to conform to various channels.
- External hardware is easily interfaced.
- Cost is low.

Successful data acquisition and decoding of (1) was the first and most important goal of this thesis. Other goals affecting software and hardware development were the following:

- Maximize packet size.
- Develop synchronization circuitry.
- Transmit/receive ASCII files.
- Develop software for testability and flexibility.
- Develop software for statistical testing.

A. SIGNAL PARAMETERS

Software and hardware implementations can be easily modified to receive any packet construction. However, all software conforms to the signal parameters in Appendix A, established between NPS and NOSC for an acoustic application. This is a bandpass signal in the band from 16-20 KHz using a clock rate of 61440 Hz. All baud lengths are powers of two, allowing utilization of

hardware/software Fast Fourier Transform (FFT) algorithms to speed frequency transformation processing.

Various baud sizes are available to give greater flexibility in adapting the packet to test specific channel parameters. In DQPSK, where the information is represented by the phase difference between adjacent tones, channel phase distortion affects the shorter baud more, due to their larger Δf . The maximum tone-to-tone phase error introduced by a linear phase channel is shown in Table 2.

Table 2. PHASE ERROR FROM A LINEAR CHANNEL

k_x	256	512	1024	2048	4096
Δf (Hz)	240	120	60	30	15
Phase error (degrees)	25.6	12.8	6.4	3.2	1.6

Obviously the longer baud would be preferred when differential coding is between adjacent tones. However, if the channel introduces time-related distortion, like noise bursts, electrical glitches, or propagation fluctuations, a short duration baud is desired, and differential coding should be between the same tones on adjacent bauds.

B. SOFTWARE

1. Transmitter

Generation of the MFM signal has been accomplished in previous research at NPS. Utilizing previous hardware and core software, this project expanded the transmitter software to provide encoding of 16-QAM and QPSK, encoded data files using DQPSK, constructed maximum size packets, generated a synchronization baud, and provided greater flexibility in initializing the DMA. Actual transmitter hardware will not be discussed; it has been the subject of

previous research [Ref. 9]. Two new transmitter programs, TRANSMIT and XMITMES, will be discussed.

a. TRANSMIT

The program TRANSMIT provides maximum flexibility in constructing and encoding signal packets. It is an excellent tool for basic system testing and can be used as a training aid to demonstrate various digital modulation schemes. A procedural flow diagram of TRANSMIT is shown in Figure 10. SelectBaud is a user interactive procedure for establishing the parameters of a 16-20 KHz packet and for choosing the encoding scheme to be utilized. Parameters k_x and L are selected. From k_x , the lower and upper bandlimits, k_1 and k_2 , are set. This generates a bandlimited packet with $k_2 - k_1 + 1$ tones. The data storage requirement is $k_x L$, representing the total number of samples in the packet which are clocked out. The encoding scheme determines the path for encoding the complex frequency domain array.

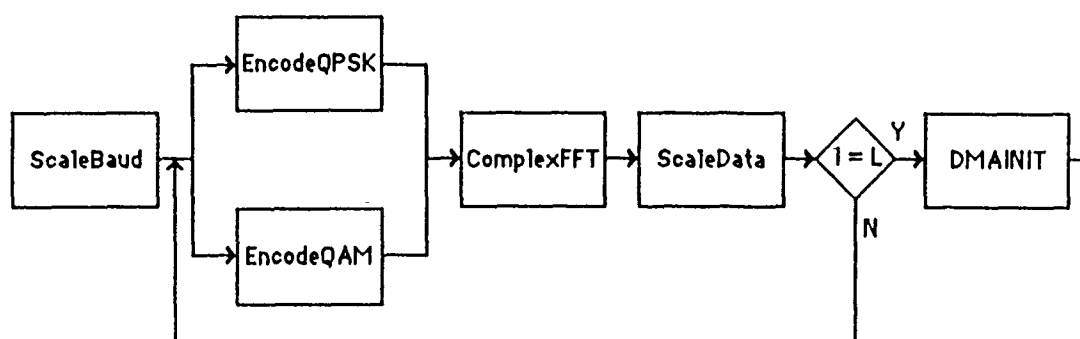


Figure 10. TRANSMIT algorithm.

EncodeQPSK begins by displaying a four symbol QPSK constellation used as a reference in selecting the symbols over the band. The symbols for the K tones in the baud can be selected in a variety ways:

- Symbols for all tones are randomly selected from a random generator.
- The symbol for each tone is selected by the user.
- Individual tones may be removed from the band.

This last feature allows construction of a baud with an arbitrary number of tones within the band determined by k_1 and k_2 . Selected symbols for the band are loaded into the complex frequency domain array with their complex conjugate image frequency. EncodeQAM is functionally the same as EncodeQPSK, except the symbols conform to a 16-QAM constellation [Ref. 1: p.27].

To obtain the real discrete time-domain sequence of the encoded baud, ComplexFFT computes the inverse FFT. ComplexFFT consumes the majority of the processing time in the program due to the complex arithmetic operations required, thus restricting the overall throughput of the system. For real time processing, the FFT algorithm must be accomplished by a hardware signal processor.

Each value in the time domain sequence is represented as a real data type, occupying six bytes of memory. ScaleData converts these values down to a one byte format acceptable to the D/A converter and places them into a packet storage buffer. EncodeQPSK, ComplexFFT, and ScaleData are executed for each baud, until all L baud have been processed. To transmit the packet out of the computer, DMAINIT transfers samples at f_x samples per second over the DMA channel to D/A converter [Ref. 9,10].

b. XMITMES

To demonstrate the suitability of MFM for transferring information from a source to a sink, the program XMITMES was written to transmit an ASCII file encoded using DQPSK. Affixed to the beginning of the packet is a synchronization baud. As shown in Figure 11, XMITMES has an even simpler structure than TRANSMIT, because all tones in the band are encoded using DQPSK.

The synchronization baud is a predetermined sequence generated by SyncBaud. This baud is constructed as are all other baud, except k_x is fixed at 256, and tones 68 to 83 are encoded with the same random symbol pattern regardless of the packet construction or input message. This synchronization sequence occupies the first 256 values in the packet buffer and therefore is the first to be clocked out of the computer.

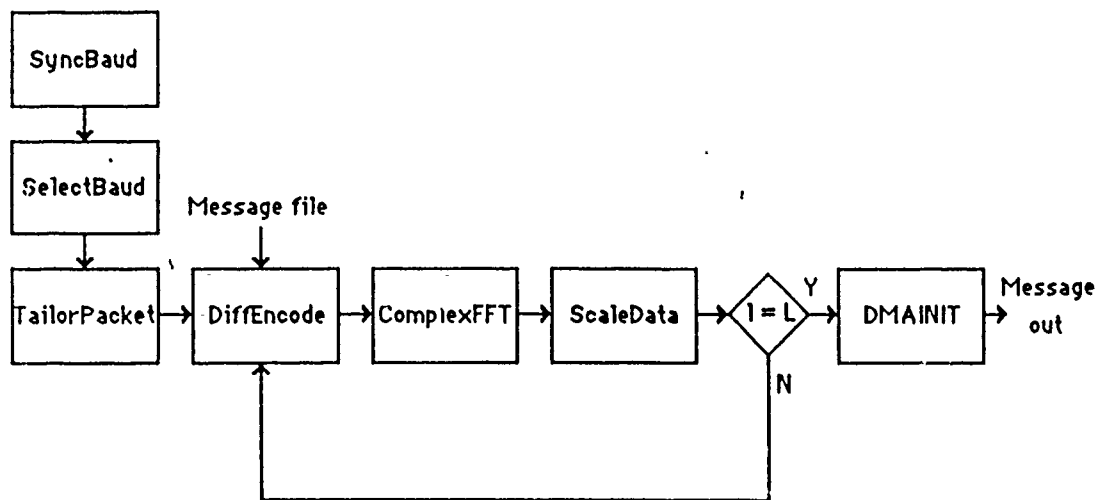


Figure 11. XMITMES algorithm.

As messages can be of various sizes, TailorPacket sets the maximum number of baud required for encoding. This is determined by dividing the number of characters in the message by the number of characters that can be encoded.

DiffEncode encodes the message file into the complex frequency domain array. It reads one character at a time; then breaks the eight bit character into four 2-bit symbols. The symbols are DQPSK encoded and stored in the frequency array. Once encoded, processing and signal output by ComplexFFT, ScaleData, and DMAINIT are the same as in TRANSMIT.

2. Receiver

Like the transmitter, the receiver requires hardware to interface the computer with the channel and processing software to demodulate the MFM signal packet. The channel interface is a high-speed data acquisition board, model DASH-16F, manufactured by MetraByte Corporation [Ref. 11]. Also at the receiver, synchronization circuitry is required to detect the beginning of the packet. The synchronization circuitry is discussed in Section C. Implementation of the processing software assumes synchronization and a receiver f_x the same as at the transmitter. Two receiver programs, RECEIVE and RECMES will be discussed.

a. RECEIVE

The program RECEIVE, shown in Figure 12, processes and displays only the first baud of a packet. It is an excellent tool to experiment with the DASH-16F's software drivers and it provides a quick indication of the overall system performance. Two procedures are available for data input, GetData and AcquireData. GetData reads in the time domain sequence file generated in the transmitter, eliminating all data acquisition hardware in the transmitter and receiver; thus ensuring perfect synchronization and accurate evaluation of all encoding, decoding, and processing software. The computational SNR of the system is determined under this condition, as the only noise present is from round off and truncation errors introduced in processing.

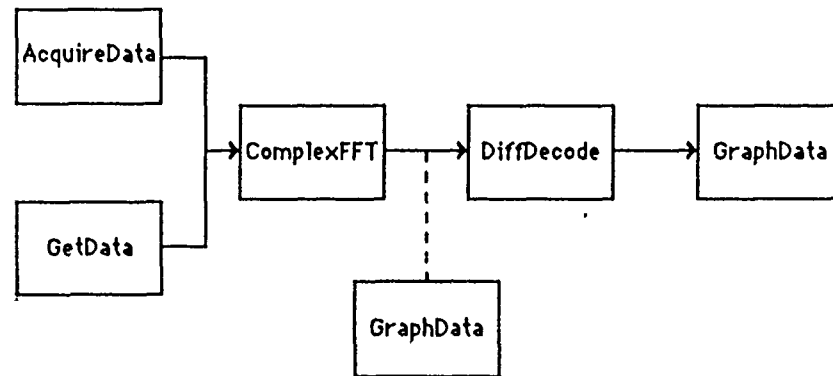


Figure 12. RECEIVE algorithm.

Analog data acquisition is performed by the procedure AcquireData. It initializes and controls the DASH-16F using procedures written by Quinn-Curtis [Ref. 12]. AcquireData allocates memory to store the sampled values transferred from the board using the DMA controller. Direct Memory Access is the only data transfer mode capable of transferring data to memory at the required f_s , without disruption by other interrupt processes in the computer. Other important initialization parameters are triggering source and the number of samples to be collected. The A/D may be triggered from two sources, a programmable interval timer or an external trigger source. The programmable interval timer divides either a 1 MHz or 10 MHz clock to derive the sampling rate of the trig-

ger. Since this method cannot produce an arbitrary f_x , external triggering is used. After initialization, conversions take place on the positive transition of every trigger until the specified number of samples have been acquired and transferred to memory. Collected data is then converted into a format acceptable for further processing.

DiffDecode determines the encoded symbols by differentially decoding the complex frequency array transformed from the sampled data by ComplexFFT. The output of the program is the frequency spectrum plot in Figure 13. The lower plot is the spectral response of the $k_x/2$ tones in the baud; the upper plot represents only tones from k_1 to k_2 . The color on the original display indicates the phase quadrant of a given tone. Graphing the data in this fashion provides quick qualitative analysis of the frequencies and their phase information.

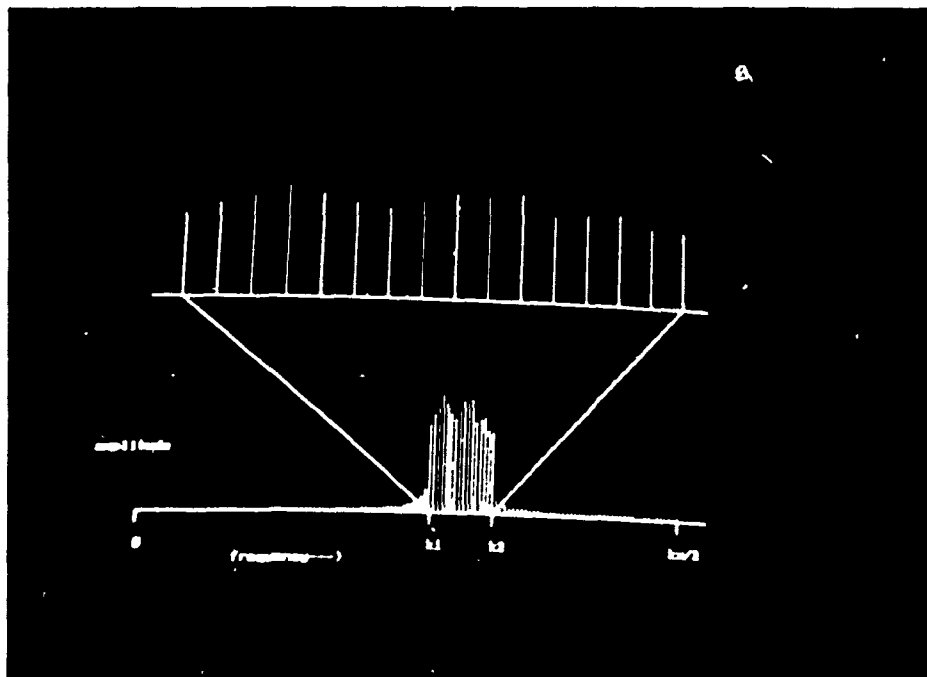


Figure 13. Baud magnitude spectrum.

b. RECMES

RECMES, shown in Figure 14, demodulates the ASCII encoded transmission produced by XMITMES. It differs from RECEIVE in that it can

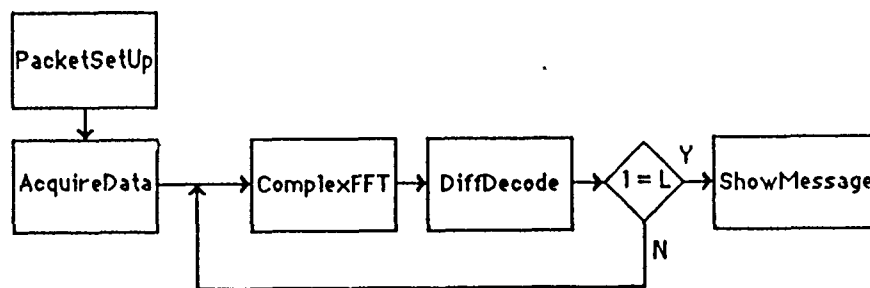


Figure 14. RECMES algorithm.

process a multiple baud packet. The user interactive procedure, PacketSetUp, tailors processing to the expected receive packet, using the inputs k_x and L . AcquireData samples and stores a memory segment of data regardless of the packet size, Lk_x . However, once stored, ConvertData, ComplexFFT, and DiffDecode process only L baud of the data. DiffDecode combines four differentially decoded symbols into one byte, representing the ordinal number of an ASCII character. To reconstruct the message, the characters are transferred to text file MESSAGE.DAT until processing is complete. For convenience, ShowMessage displays the recovered message.

C. HARDWARE

Synchronization of DQPSK MFM is obtained from a hardware correlator that is external to the host receiver computer. The 128 point correlator, illustrated in Figure 15, provides the data acquisition board with sampling triggers synchronized with respect to time of arrival of the packet. Using only the polarity of the analog input, it functions as a matched filter to the last half of the 256 point synchronization baud. This type of correlator is referred to as a polarity coincidence correlator(PCC).

The hard limiter used to obtain the polarity information in the analog input is a fast, high precision, high gain, operational amplifier. During positive and negative portions of the input the output is +5 Vdc and 0 Vdc respectively. This unipolar signal is synchronized to the receiver's f_x as it is clocked through a 128 point serial shift register. To minimize bit instability, due to the shift register input being asynchronous to f_x , the hard limiter slew rate should be as large as

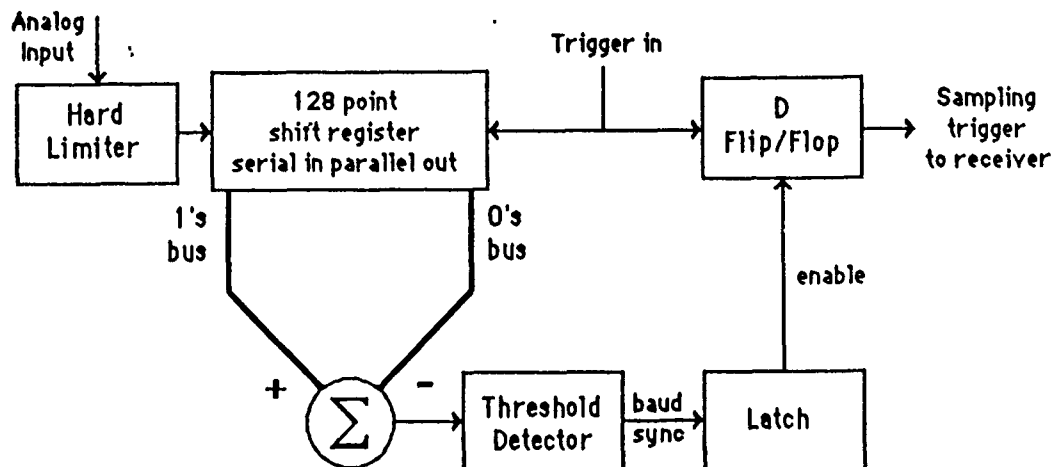


Figure 15. Polarity coincidence correlator.

possible. For example, during testing, 15 bits out of 128 were unstable using an MC1747 general purpose operational amplifier with a slew rate of 0.7 V/ μ sec. This was reduced to 2 bits using the faster LM318 with a slew rate of 70 V/ μ sec.

The 1's and 0's buses store the time-reversed polarity sequence of the synchronization baud. The voltage on the buses represent the correlation between the stored sequence and the sequence in the shift register. As the sequences come into alignment, the 1's bus voltage increases while the 0's decreases. This inverse property is combined by a differential amplifier to give the total correlation. When both sequences match, the 1's bus voltage is +5 Vdc and the 0's is 0 Vdc, giving a maximum differential voltage of 5 Vdc into the threshold detector. The threshold detector generates a synchronization trigger on detection of the correlation peak. A latch is set, enabling the D flip-flop to pass sampling triggers for data acquisition at the packet's beginning as required for demodulation.

During design and testing the correlator progressed from a 16 to 128 point shift register. With each shift register output bus connected through a resistor voltage divider network, progressive testing was necessary to ensure power supply current ratings were not exceeded. Maximal-length sequences were used as test correlation sequences due to their unique two-valued autocorrelation functions that are easily determined by finite-field arithmetic [Ref. 8: pp. 368-375].

V. PERFORMANCE EVALUATION

Initial testing of the MFM system was concerned with quickly determining the quality of the demodulated signals. Figure 13 provided an excellent representation of the demodulated and decoded signal and was used throughout system development. This plot uniquely displays the frequency domain magnitude and phase. Since QPSK symbols are at least 90 degrees apart each color (on the original display) represents a specific quadrant in the complex frequency plot. For example, a green line indicates a tone whose phase is in the second quadrant and whose magnitude is represented by the length of the line. Although each color represents a broad decision region in phase, it is still clear a phase shift of at least 180 degrees occurs across the 4 KHz band. This shift is primarily due to the linear phase Butterworth filter used to simulate the system channel, and it contributed to the decision to develop a DQPSK encoded signal.

A. SYSTEM FREQUENCY RESPONSE

After development of all hardware and software, testing revealed significant phase fluctuations across the frequency band. The system phase response, plotted in Figure 16, was determined by subtracting the transmitted phase from the received phase for each tone. Ideally the response should be smooth. With DQPSK encoded from tone-to-tone, phase differences introduced by the system will result in reduced tolerance to additive noise. For example, notice the phase difference of 0.4 radians between tones 296 and 297. When noise is added, decoding the symbol in error will occur at a lower noise level than it would if the phase difference was zero; thus a higher SNR is required for low error rate decoding when the channel introduces tone-to-tone phase fluctuations. The tone-to-tone phase fluctuations were substantially reduced, as shown in Figure 17, by decreasing the magnitude of the encoded tones in the transmitter which moved the operating range of the D/A converter to a more linear region. Further analysis of the effect of the D/A and A/D converters on phase shift is recommended in order to be able to specify them properly in future designs.

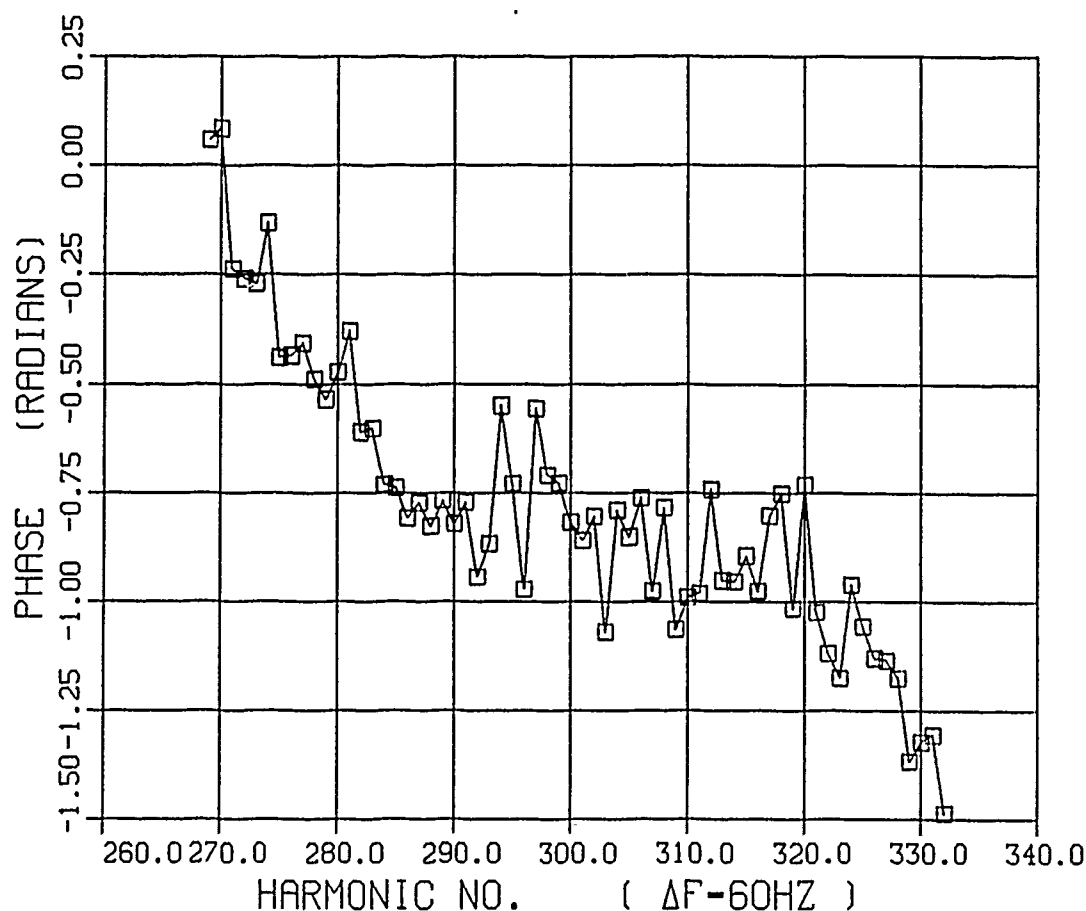


Figure 16. Initial system response.

A second important feature of the system phase response is its average slope across the band. The phase slope is affected by synchronizatio timing as shown in Figure 18. When the sequence through the correlator was delayed 2 bits, or $2/61440 = 32.55 \mu\text{secs}$, relative to the analog input to the receiver, thus delaying data acquisition accordingly, the phase slope changes as predicted by the time delay Fourier transform theorem;

$$x(t - T_d) \leftrightarrow X(f)e^{-j\omega T_d}. \quad (12)$$

This shows the phase slope in radians for a signal delayed by T_d seconds, is $2\pi T_d$ radians/Hz. For example, at $f = \Delta f/k_i = (60)(300) = 18000 \text{ Hz}$ and

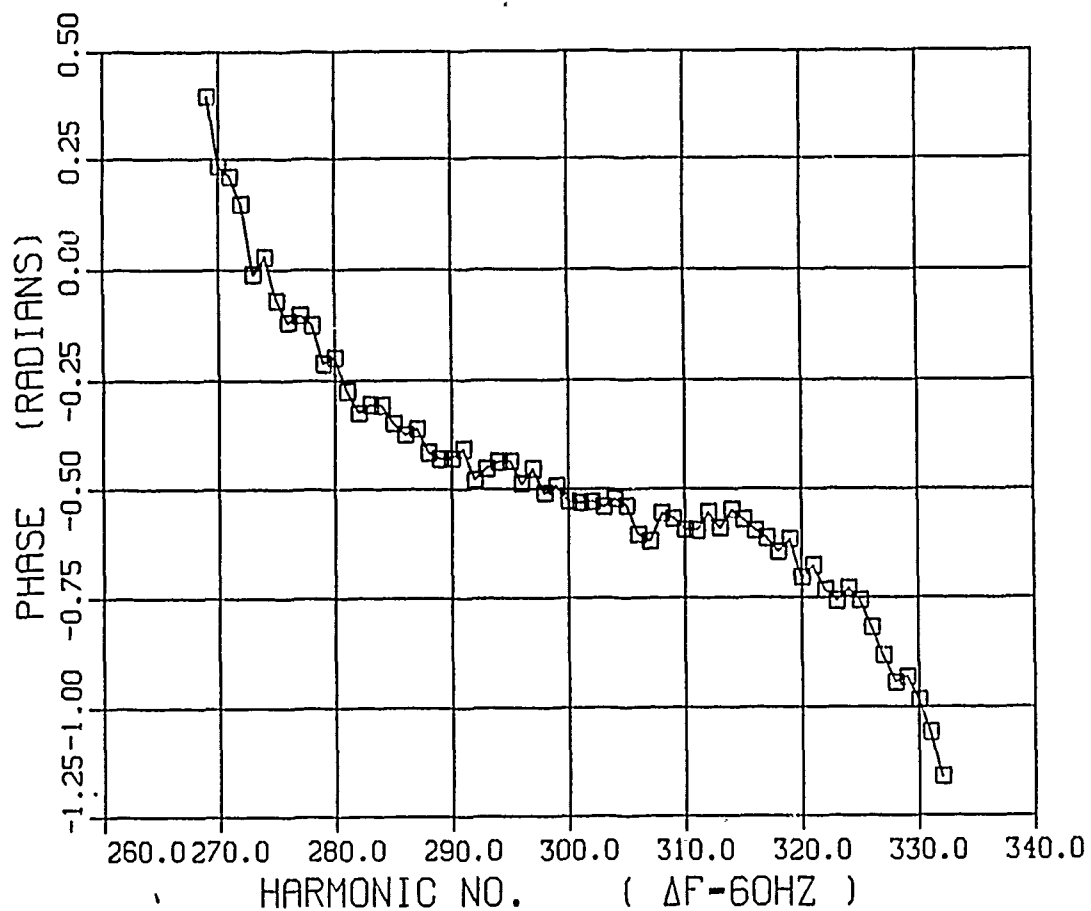


Figure 17. Improved system response.

$T_d = 32.55 \mu\text{secs}$, the phase shift is 3.7 radians. This compares closely to the actual phase shift of tone 300 in Figure 17 and Figure 18.

In summary, system frequency response analysis identified phase shifts introduced by the system hardware and by the synchronization timing. The magnitude of the encoded tones and the synchronization sequence delay were selected experimentally in order to minimize tone-to-tone phase fluctuations and provide the comparatively flat response shown in Figure 18 for all subsequent testing.

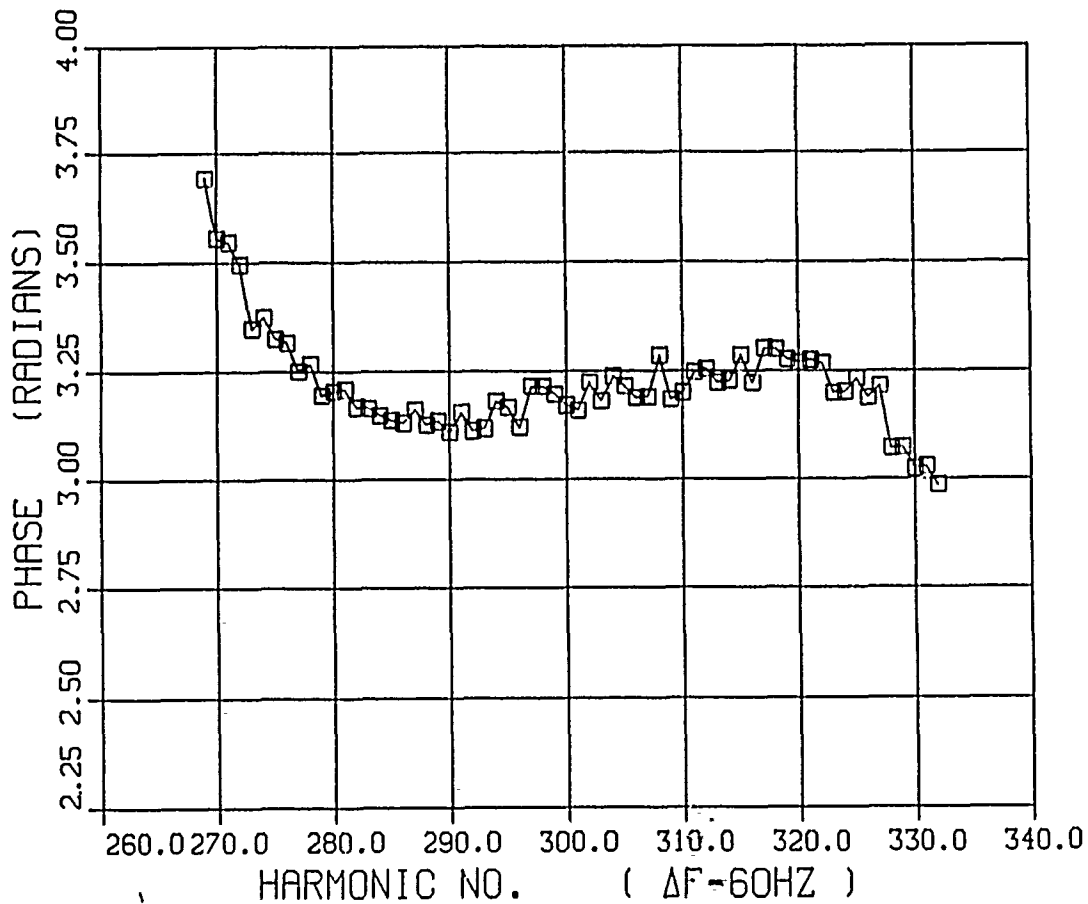


Figure 18. System response with 2-bit synchronization delay.

B. SNR PERFORMANCE

The performance of the system when corrupted by additive noise is determined by its output signal-to-noise ratio (SNR). For QPSK-encoded MFM the mean of the real and imaginary parts of the $2K$ coefficients of the DFT represent the received signal amplitude, and their variance represents the noise power. The output SNR is defined as the ratio of the square of the mean to the variance of each for these $2K$ coefficients [Ref. 1: p. 25].

Figure 19 shows the system SNR, which is the output SNR when there is no additive noise in the channel, versus the frequency spacing, Δf , of the five baud types. As would be expected, DQPSK performs better with a smaller Δf because the phase difference between adjacent tones is smaller when Δf is smaller.

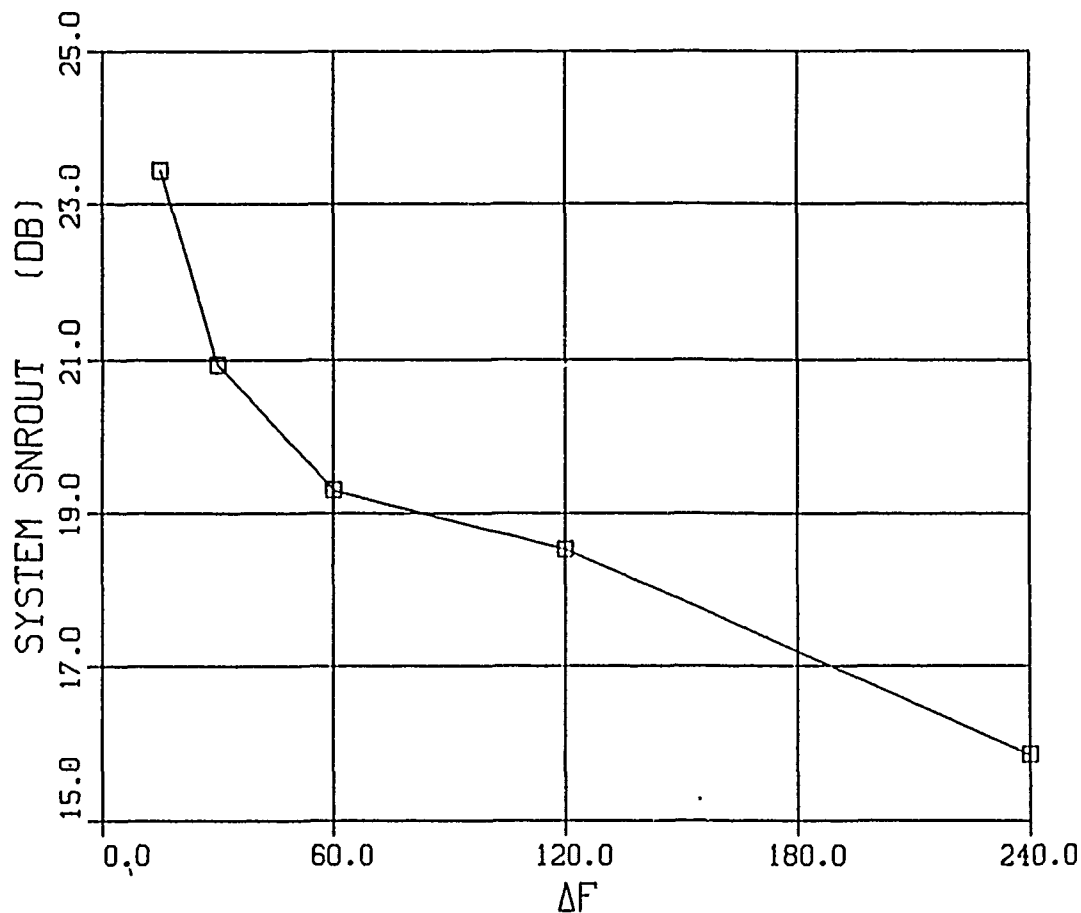


Figure 19. Maximum SNR output.

In Figure 20, the SNR out of the receiver is shown versus input SNR which was varied by adding noise to the analog signal. Theoretically, the output SNR should equal the narrowband input SNR in QPSK [Ref. 1: pp. 24-25]. The reduction in output SNR at higher input SNRs is due to each baud approaching its maximum output SNR as set by system noise and shown in Figure 19. Table 3 lists the bit errors at various SNR input levels. There were approximately 2500 bits transmitted for each entry. This clearly indicates the worsening effect of the channel on bauds with greater tone spacing Δf at high SNRs.

A direct comparison of the MFM signal with commercial modems is difficult due to the wide variety of techniques that are used to achieve high-speed data transfer. However, with a bandwidth efficiency of 2 bits/s/Hz, a DQPSK MFM

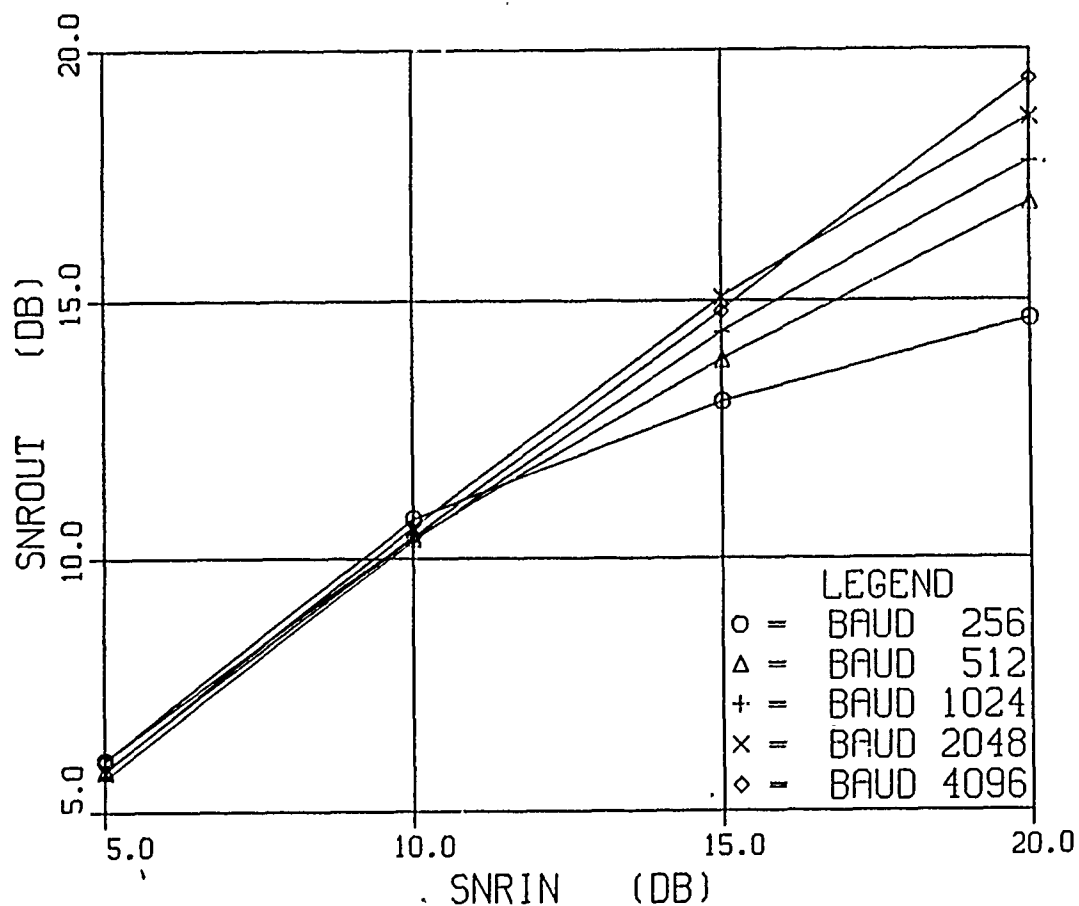


Figure 20. SNR performance.

signal can achieve a throughput of 6000 bits per second on a standard switched telephone line having a bandwidth of 3 KHz. Though not implemented in this project, several techniques could be utilized to increase the bandwidth efficiency, such as increasing the constellation size and/or using data compression algorithms as in commercial modems [Ref. 13, 14]. Using these techniques MFM could achieve a bandwidth efficiency of 4-8 Bits/s/Hz, which compares favorably to current high speed modems.

Table 3. BIT ERRORS IN 2500 BITS TRANSMITTED VS BAUD TYPE AND SNR.

Δf k_x	SNR (dB)			
	5	10	15	20
240 256	165	35	18	10
120 512	154	16	3	0
60 1024	148	17	0	0
30 2048	140	9	0	0
15 4096	127	10	0	0

VI. CONCLUSIONS AND FURTHER STUDY

MFM is well suited to the signal processing environment of the personal computer. As implemented in this thesis, the packet signal provides a high speed, low bit error data transfer link between two industry standard computers. Synchronization hardware allows the link to use asynchronous data.

The theory and properties of MFM have been discussed for a white bandpass packet signal. Packet construction and various modulation formats have been shown to be easily adapted to a given channel. The system was developed and implemented for testability and to be easily modified to accommodate a variety of applications. However, the data storage requirements and signal conversion speeds were unique to this project due to available hardware. As implemented, this system would require a multiple packet transmission to achieve practical operation.

Analysis of the system's phase response led to adjustments in the encoded tone magnitudes and synchronization timing. This substantially improved the system's phase response. SNR test results indicate a smaller Δf baud has superior performance in a linear phase channel. For Δf less than 60 Hz, bit errors are acceptably low for input SNRs greater than approximately 15 dB. Exhaustive testing is required to compare bit error performance with conventional digital modulation schemes.

Areas of further study should focus toward increasing the system bandwidth efficiency and implementation of hardware signal processing for real-time operation. In fact, current research at NPS is directed at increasing the throughput speed of the system described. Improved bandwidth efficiency can be obtained through data compression algorithms and developing receiver software to decode the 16-QAM encoded signal generated by TRANSMIT. Finally to provide a greater flexibility in adapting MFM to channels with significant uncompensated phase distortion, the option should be developed to encode DQPSK on a baud-to-baud basis.

APPENDIX A. DESIGN PARAMETERS

**Table 4. DESIGN PARAMETERS FOR A 1/15TH SECOND SIGNAL PACKET
IN A 16-20KHZ BANDPASS CHANNEL.**

Baud length (sec)	ΔT	1/240	1/120	1/60	1/30	1/15
No. of Bauds	L	16	8	4	2	1
Tone spacing	Δf	240	120	60	30	15
Lowest Harmonic	k_1	68	135	269	537	1073
Lowest tone freq	f_1	16320	16200	16140	16110	16095
Highest Harmonic	k_2	83	166	332	664	1328
Highest tone freq	f_2	19920	19920	19920	19920	19920
Samples per Baud	k_x	256	512	1024	2048	4096
Sampling freq	f_x	61440	61440	61440	61440	61440

APPENDIX B. OPERATING INSTRUCTIONS

A. PRELIMINARIES

1. Hardware setup

a. System setup

Setup system as shown in Figure 21. All signal/trigger connections should be accompanied with a ground lead to minimize interference.

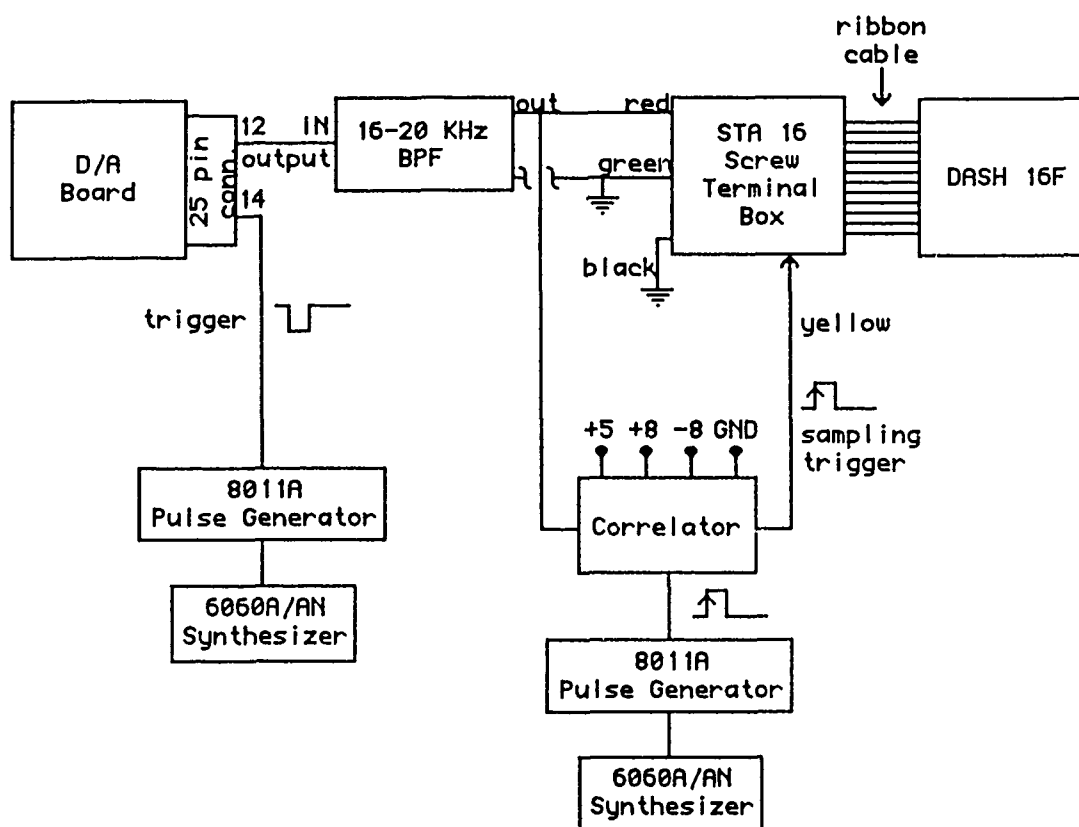


Figure 21. System interconnection diagram.

b. DASH-16F switch setting:

CHAN CNFG8
A/DBIP
DMA1

GAIN (12.5v) -----dipswitch A, B--ON; C, D, USER--OFF
 Address (300h) -----dipswitch 9, 8--OFF; 7, 6, 5, 4--ON

c. Trigger Requirements

The D/A converter board uses negative logic as shown in Figure 22. The pulse width, PW, must be less than one μsec to ensure proper signal output. This is due to DMA request and acknowledge hardware. All hardware on the correlator and DASH-16F is positive edge-triggered. The trigger magnitude should be approximately 5.3V peak, due to CMOS shift register chips in the correlator.

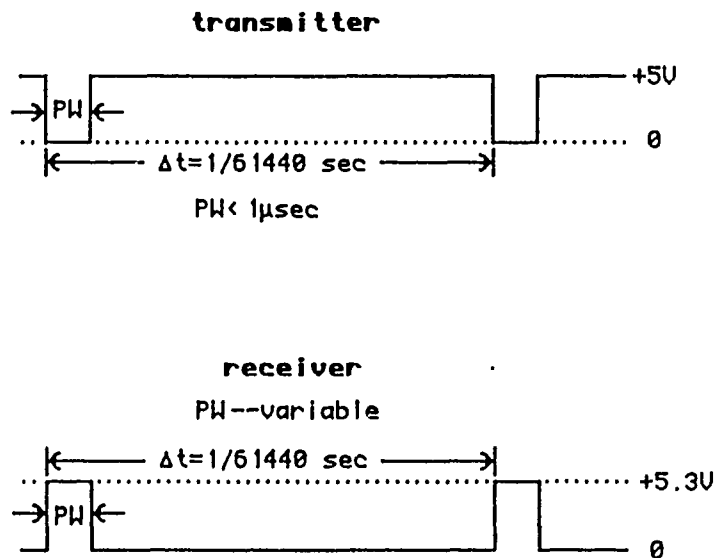


Figure 22. Trigger specifications.

d. Power Supplies

+8Vdc supply -----HP6216A 30v 500ma (or suitable substitute)
 -8Vdc supply -----HP6216A 30v 500ma (or suitable substitute)
 +5Vdc supply -----HP6216A 30v 500ma (or suitable substitute)

2. Software setup

a. Transmitter

1. Convert DMAINIT.ASM and DMASTOP.ASM to BINARY files [Ref. 15: pp. 91-93].
2. Place files DMAINIT.BIN, DMASTOP.BIN, COMPFIT.INC, and FIT87B2.INC in the same directory as TRANSMIT.PAS AND XMITMES.PAS.

3. Compile to disk TRANSMIT.PAS and XMITMES.PAS.

b. Receiver

1. Compile TP4D16.PAS to disk to create TP4D16.TPU.
2. Place TP4D16.TPU in the same directory as RECEIVE.PAS and RECMES.PAS.
3. Compile to disk RECEIVE.PAS and RECMES.PAS.

B. MESSAGE TRANSMISSION PROCEDURE

1. As necessary conduct preliminary equipment setup.
2. Reset correlator (removes triggers to DASH-16F).
3. Run XMITMES until "Ready to Transmit".
4. Run RECMES (match baud size used in XMITMES).
5. When receiver prompts "Ready to acquires", press enter key on transmitter.
6. Demodulated message is displayed on receiver screen and stored in file MESSAGE.DAT.

TRANSMIT and RECEIVE are similarly executed. However TRANSMIT would require manual loading of the synchronization baud in Table 5 by individual loading each tone with the appropriate symbol.

Table 5. SYNCHRONIZATION BAUD SYMBOL SEQUENCE

tone	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
symbol	3	3	2	2	4	1	2	1	2	3	4	3	1	4	3	1

APPENDIX C. TESTING PROCEDURES

A. RESPONSE TESTING PROCEDURE

1. Conduct preliminary equipment setup.
2. Run XMITMES and exit.
3. Transfer file XMITDAT.DAT to the receiver computer. (Not required for subsequent response testing if transmitted message and packet construction are unchanged.)
4. Transmit message IAW message transmission procedure. (RECMES generates output file RECSTAT.DAT used by RESPONSE and STATS.)
5. Run RESPONSE. (Ensure baud size is correctly edited into RESPONSE.)
6. Graph output file RESPbaud.DAT.

B. SNR TESTING PROCEDURE

1. Conduct preliminary equipment setup.
2. Add random noise source and true RMS voltmeter as shown in Figure 23.

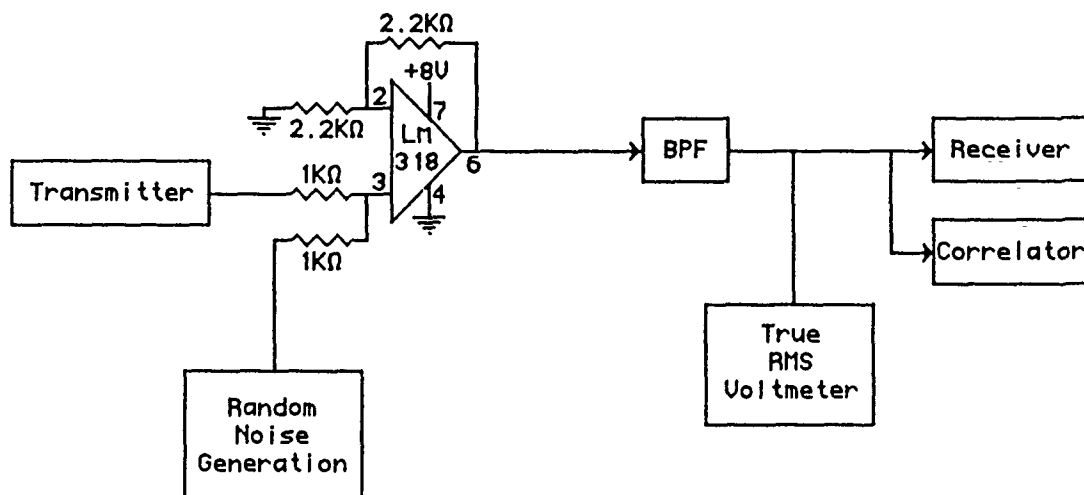
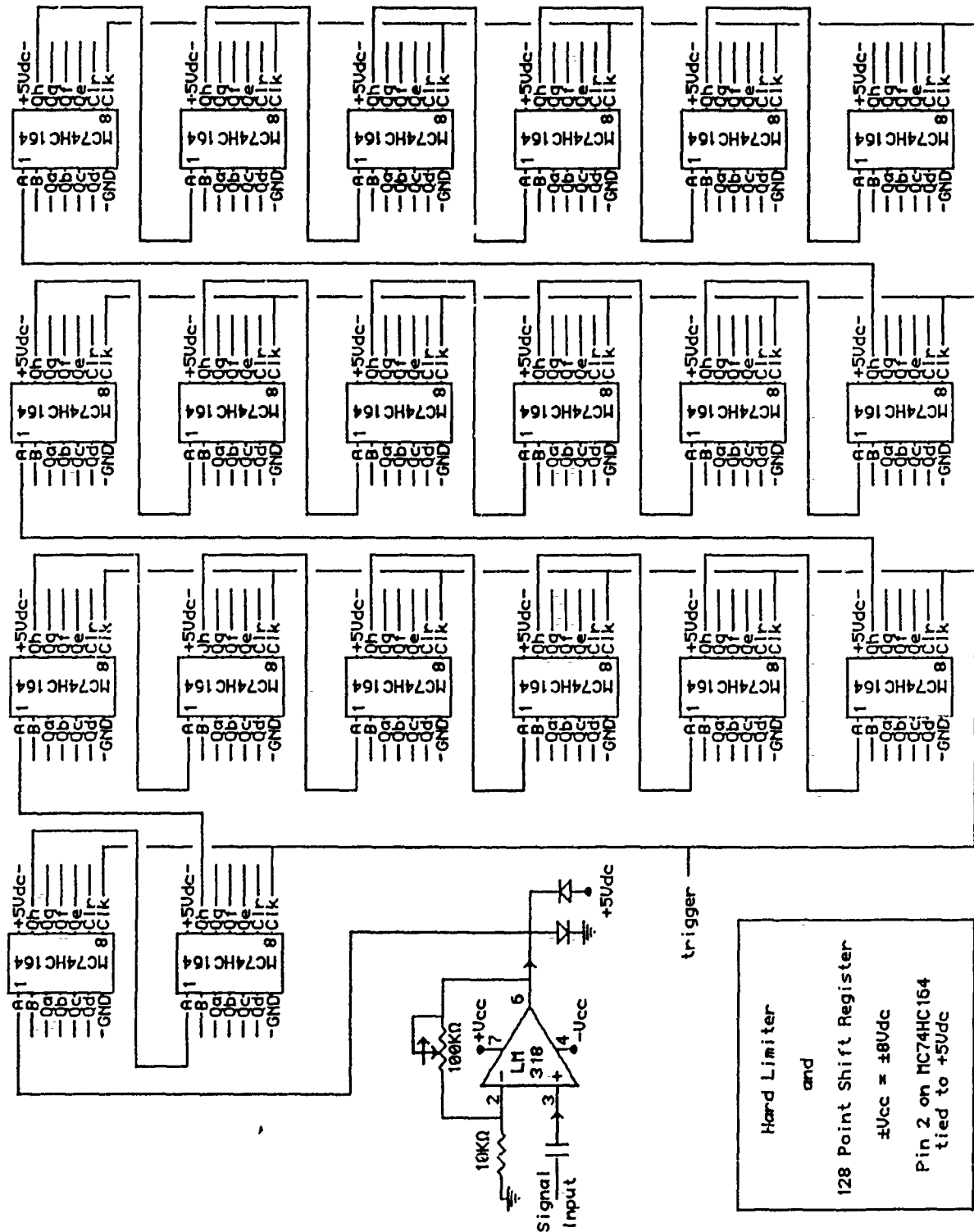


Figure 23. Test equipment interconnection.

3. Set SNR level.
4. Conduct steps 1-3 in the Response testing procedure.
5. Run Statistics (results are in file STATbaud.DAT).

APPENDIX D. CORRELATOR SCHEMATIC



APPENDIX E. TRANSMIT

```

program TRANSMIT;
(*Transmits variable length packets up to 61440. The length is
determined by BCSTARRAY and can be take to a full segment of memory.
No sync baud is used. All external files are assumed to be in the same
directory*)

type
  TNvector = array[0..4095] of real;
  TNvectorPtr = ^TNvector;
  BCSTARRAY = array[-28673..32767] of byte;  (*stores output samples*)

var
  kx,          (*baud size*)
  k1,          (*lowest tone in band*)
  k2,          (*highest tone in band*)
  NUMBAUDS,    (*number of bauds in packet*)
  BAUDCOUNT,  (*current baud being processed*)
  BYTECOUNT,  (*number of bytes to be transferred by DMA*)
  I            : integer;
  XREAL,       (*complex freq/time domain arrays*)
  XIMAG        : TNvectorPtr;
  QAM,         (*encoding scheme*)
  INVERSE      : boolean;  (*direction of FFT*)
  ERROR        : byte;     (*status of ComplexFFT*)
  BCST         : BCSTARRAY; (*packet storage buffer*)
  THEFILE      : file of byte; (*time sequence output file*)
  ANSWER       : char;     (*input variable*)

(*$I FFT87B2.INC*)
(*$I COMPPFFT.INC*)

(*-----*)
procedure SelectBaud;
(*SelectBaud establishes QPSK or QAM encoding kx, k1, k2, and the number
of baud in the packet*)

var
  ANSWER      : integer;
  MODANSWER   : char;

begin
  window(20,10,80,20);

  (*Select encoding scheme*)
  repeat
    clrscr;
    writeln('Select modulation desired');
    writeln;
    writeln('  A --- QPSK');
    writeln('  B --- 16 QAM');
    readln(MODANSWER);
  until MODANSWER in ['A','B'];

```

```

until MODANSWER in ['a','A','b','B'];
if MODANSWER in ['b','B'] then
    QAM := true;

(*Select buad size*)
kx:=0;
repeat
    clrscr;
    if kx < 0 then writeln('TRY AGAIN');
    writeln('What is the length of the bauds (kx)?');
    writeln('i.e. 256, 512, 1024, 2048, 4096');
    readln(ANSWER);
    case ANSWER of
        256: kx:=256;
        512: kx:=512;
        1024: kx:=1024;
        2048: kx:=2048;
        4096: kx:=4096;
    end; (*case ANSWER*)
    if kx = 0 then
        kx := -1;
    until kx > 0;

(*Set bandlimits*)
case kx of
    256:begin
        k1:=68; k2:=83;
        end;
    512:begin
        k1:=135; k2:=166;
        end;
    1024:begin
        k1:=269; k2:=332;
        end;
    2048:begin
        k1:=537; k2:=664;
        end;
    4096:begin
        k1:=1073; k2:=1328;
        end;
end; (*case kx*)

(*Select number of baud*)
write('How many bauds do you desire to transmit? ');
readln(NUMBAUDS);
window(1,1,80,25);
end; (*SelectBaud*)

(*-----*)
(*-----*)
procedure DisplayQAM;
(*DisplayQAM shows the 16-QAM constellation in the upper portion of the
screen. A window is set at the bottom of the screen for further
interaction with the user.*)

begin
    clrscr;
    gotoXY(11,1);

```

```

write(chr(218));
for I:= 12 to 64 do
  write(chr(196));
write(chr(191));
for I:= 2 to 19 do
  begin
    GotoXY(11,I); write(chr(179));
    GotoXY(65,I); write(chr(179));
  end;
GotoXY(11,20);
write(chr(192));
for I := 12 to 64 do
  write(chr(196));
write(chr(217));

window(12,2,63,19);
write('This program encodes a 16 QAM multifrequency');
writeln(' signal. ');
writeln('The vector space is shown for one frequency. ');
writeln('      ,chr(179),');
writeln('      *      *      ,chr(179),      *      *      ');
writeln('      4      3      ,chr(179),      2      1      ');
writeln('      ,chr(179),');
writeln('      *      *      ,chr(179),      *      *      ');
writeln('      8      7      ,chr(179),      6      5      ');
writeln('      ,chr(179),');
writeln('      _____ ,chr(179), _____ ');
writeln('      *      *      ,chr(179),      *      *      ');
writeln('      12     11     ,chr(179),      10     9      ');
writeln('      ,chr(179),');
writeln('      *      *      ,chr(179),      *      *      ');
writeln('      16     15     ,chr(179),      14     13     ');
writeln('      ,chr(179),');

(*active window*)
window(12,21,80,25);
end; (*DisplayQAM*)
(*-----*)
(*-----*)
procedure DisplayQPSK;
(*DisplayQPSK shows the QPSK constellation in the upper portion of the
screen. A window is set at the bottom of the screen for further
interaction with the user.*)

begin
  clrscr;
  gotoXY(11,1);
  write(chr(218));
  for I:= 12 to 64 do
    write(chr(196));
  write(chr(191));
  for I:= 2 to 19 do
    begin
      GotoXY(11,I); write(chr(179));
      GotoXY(65,I); write(chr(179));
    end;
end;

```



```

        9..12 :TEMPI:=-30.0;
        13..16:TEMPI:=-90.0;
    end;(*case VECTORARRAY*)
    case VECTORARRAY[J] of
        0      :TEMPR:= 0.0;
        1,5,9,13 :TEMPR:= 90.0;
        2,6,10,14:TEMPR:= 30.0;
        3,7,11,15:TEMPR:=-30.0;
        4,8,12,16:TEMPR:=-90.0;
    end;(*case VECTORARRAY*)

(*Load complex conjugate image*)
    XREAL^[J]:= TEMPR;
    XREAL^[kx-J]:= TEMPR;
    XIMAG^[J]:= TEMPI;
    XIMAG^[kx-J]:= -TEMPI,
    end;(*J:=k1*)
end;(*EncodeData*)
(*-----*)
begin
    repeat
        clrscr;
        writeln(' Select one of the following for baud ',BAUDCOUNT,' ');
        writeln;
        writeln('R      Randomly select all ',k2 - k1 + 1,' vectors');
        writeln('I      Individually select vectors');
        readln(RESPONSE);
    until RESPONSE in ['r','R','i','I'];
    fillchar(VECTORARRAY,sizeof(VECTORARRAY),0);
    case RESPONSE of
        'r','R' :for I:= k1 to k2 do
            VECTORARRAY[I]:=random(16)+1;
        'i','I' :begin
            I:= k1;
            while I <= k2 do
                begin
                    repeat
                        clrscr;
                        writeln('      Vector Selection Menu');
                        writeln('#      vector desired');
                        writeln('17      random vector');
                        writeln('18      no vector');
                        write('19      no more tones      Tone ',I,' ');
                        readln (ANSWER);
                    until ANSWER in [1..19];
                    case ANSWER of
                        1..16: VECTORARRAY[I]:= ANSWER;
                        17: VECTORARRAY[I]:= random(16)+1;
                        18:;(*nop*)
                        19: I:= k2;
                    end;(*case ANSWER*)
                    I:=I+1;
                end;(*while I*)
            end;(*'i','I'*)
        end;(*case RESPONSE*)
    end;
end;

```

```

    EncodeData;
end; (*SelectQAM*)
(*-----*)
(*-----*)
procedure SelectQPSK;
(*SelectQPSK selects and encodes the symbols for each tone in the band
set by k1 and k2. All symbols can be randomly or manually selected.*)

var
    RESPONSE    : char;
    ANSWER,I    : integer;
    PHASEARRAY  : array[0..4095] of integer;

(*-----*)
procedure EncodeData;
(*EncodeData loads the frequency domain arrays XREAL and XIMAG with the
QPSK symbols in VECTORARRAY and their complex conjugate image.*)

var
    J            : integer;
    TEMPR,TEMPI : real;

begin
    fillchar(XREAL^,sizeof(XREAL^),0);
    fillchar(XIMAG^,sizeof(XIMAG^),0);
    for J:= k1 to k2 do
        begin
            case PHASEARRAY[J] of
                0      : TEMPI:= 0.0;
                1,2    : TEMPI:= 80.0;
                3,4    : TEMPI:= -80.0;
            end; (*case PHASEARRAY*)
            case PHASEARRAY[J] of
                0      : TEMPR:= 0.0;
                1,4    : TEMPR:= 80.0;
                2,3    : TEMPR:= -80.0;
            end; (*case PHASEARRAY*)
            XREAL^[J] := TEMPR;
            XREAL^[kx-J] := TEMPR;
            XIMAG^[J] := TEMPI;
            XIMAG^[kx-J] := -TEMPI;
        end; (*J:=k1*)
    end; (*EncodeData*)
(*-----*)
begin
    repeat
        clrscr;
        writeln(' Select one of the following for baud ',BAUDCOUNT,'. ');
        writeln;
        writeln('R      Randomly select all ',k2 - k1 + 1,' phases');
        writeln('I      Individually select phases');
        readln(RESPONSE);
    until RESPONSE in ['r','R','i','I'];
    fillchar(PHASEARRAY,sizeof(PHASEARRAY),0);
    case RESPONSE of
        'r','R': for I:= k1 to k2 do

```

```

        PHASEARRAY[I]:=random(4)+1;
    'i','I':begin
        I:= k1;
        while I <= k2 do
            begin
                repeat
                    clrscr;
                    writeln('      Phase Selection Menu');
                    writeln('#      phase desired');
                    writeln('5      random phase');
                    writeln('6      no phase');
                    write('7      no more tones      Tone ',I,' ');
                    readln (ANSWER);
                until ANSWER in [1..7];
                case ANSWER of
                    1..4: PHASEARRAY[I]:= ANSWER;
                    5: PHASEARRAY[I]:= random(4)+1;
                    6:; (*nop*)
                    7: I:= k2;
                end; (*case ANSWER*)
                I:=I+1;
            end; (*while I*)
        end; (*'i','I'*)
    end; (*case RESPONSE*)
    EncodeData;
end; (*SelectQPSK*)
(*-----*)
procedure ScaleData;
(*ScaleData converts each real value in array XREAL down to a byte and
stores the byte in packet storage buffer BCST. INDEX sets the location
in the buffer of each byte*)

var
    INDEX,J,TEMP    : integer;
    DATA           : byte;

begin
    for J := 0 to kx-1 do
        begin
            TEMP := round(XREAL^[J] + 126);
            if TEMP < 0 then
                TEMP := 0;
            DATA := TEMP;
            INDEX := J + (BAUDCOUNT - 1) * kx - 28673;
            BCST[INDEX] := DATA;
            write(THEFILE,DATA);
        end; (*for J*)
    end; (*ScaleData*)
(*-----*)
(*-----*)
procedure Dmainit(var BCST : BCSTARRAY;
                  BYTECOUNT : integer);
(*Assembly language procedure used to initialize and unmask the DMA for
data transfer. The source must be converted to a BIN file.*)

```

```

external 'DMAINIT5.BIN';
(*-----*)
procedure Dmstop; (*Masks DMA, stopping data transfer*)
external 'DMASTOP.BIN';
(*-----*)
procedure TINPUT;
(*This procedure reads in binary file, HOUTFA1.DAT, into an array,
  performs an FFT using ComplexFFT and displays the results
  with GraphData*)

var
  NUMPTS : integer;
(*-----*)
procedure GetData;
(*GetData reads time domain sequence from file HOUTFA1.DAT into XREAL*)

var
  THEFILE : file of byte;
  DATA   : byte;
  I,
  TEMP    : integer;
  rdata   : real;

begin
  assign(THEFILE, 'HOUTFA1.DAT');
  reset(THEFILE);
  new(XREAL);
  new(XIMAG);
  fillchar(XREAL^, sizeof (XREAL^), 0);
  fillchar(XIMAG^, sizeof (XIMAG^), 0);
  for I := 0 to kx-1 do
    begin
      read(THEFILE, DATA); (*read file one byte at a time*)
      TEMP := DATA;
      temp := temp and 255; (*reduces number of bits used*)
                          (*to represent the input data*)

      XREAL^[I] := (TEMP - 126); (*loads XREAL array*)
    end; (*for I*)
  close(THEFILE);
end; (*GetData*)
(*-----*)
(*-----*)
procedure PutData;
(*Writes frequency domain arrays XREAL and XIMAG to file TFFTOUT.OUT*)

var
  I      : integer;
  OUTFILE : text;

begin
  assign(OUTFILE, 'TFFTOUT.DAT');
  rewrite(OUTFILE);
  for I := 0 to kx-1 do
    writeln(OUTFILE, I, ' ', XREAL^[I], ' ', XIMAG^[I]);

```

```

        close(OUTFILE);
end; (*PutData*)
(*-----*)
procedure GraphData;
(*GraphData plots the kx/2 points of complex array XREAL and XIMAG.
  Symbols are decoded and represented by the color of each tones
  spectral line.*)

var
    I,J,                (*loop indices*)
    KXDIV2,
    COLOR,
    X,XZ,
    Y,YZ    : integer;

begin
    GraphColorMode;
    palette(2);
    Graphbackground(0);
    draw(30,10,30,170,1);    (*y-axis*)
    draw(30,170,300,170,1);  (*x-axis for full spectrum*)
    draw(30,155,300,155,1);
    draw(30,80,300,80,1);    (*x-axis for zoom spectrum*)

    (*every other dot equals one bin at 256*)
    draw(166,170,166,174,1); (*x-axis scale marks*)
    draw(196,170,196,174,1); (*x-axis scale marks*)
    draw(30,113,26,113,1);    (*y-axis scale marks*)
    draw(30,23,26,23,1);      (*y-axis scale marks*)
    I:= 23;
    (*Grouping of 4 horizontal lines to identify background color
    used for one of the symbols*)
    repeat
        draw(31,I,300,I,0);
        draw(31,I+1,300,I+1,1);
        draw(31,I+2,300,I+2,2);
        draw(31,I+3,300,I+3,3);
        I:=I+12;
    until I > 75;
    KXDIV2 := round(kx / 2);

    (*Decode symbols and assign color*)
    for I := 0 TO KXDIV2 do
        begin
            if (XREAL^[I] >= 0) and (XIMAG^[I] > 0) then
                COLOR := 0;
            if (XREAL^[I] < 0) and (XIMAG^[I] > 0) then
                COLOR := 1;
            if (XREAL^[I] < 0) and (XIMAG^[I] <= 0) then
                COLOR := 2;
            if (XREAL^[I] >= 0) and (XIMAG^[I] <= 0) then
                COLOR := 3;

            (*zoom spectrum*)

            if (I >= k1) and (I <= k2) then

```

```

begin
  XZ := round(((I * 4096.0) / kx) - 1040);
  YZ := 80 - round(0.5*(sqrt(sqr(XREAL^[ I]) + sqr(XIMAG^[ I] ))));
  draw(XZ,80,XZ,YZ,COLOR);
end;

(*full spectrum*)

X := round(I/KXDIV2 * 256 +30);
Y :=170-round(20*ln(sqrt(sqr(XREAL^[ I]+1)+
                        sqr(XIMAG^[ I]+1)))/ln(10));
draw(X,170,X,Y,COLOR);
end;
end; (*GraphData*)
(*-----*)
begin
  INVERSE := false;    (*sets forward FFT*)
  ERROR := 0;
  GetData;
  NUMPTS := kx;
  writeln('Performing FFT');
  ComplexFFT (NUMPTS,INVERSE,XREAL,XIMAG,ERROR);
  writeln('Error = ',ERROR,' hit the enter key'); readln;

  PutData;

  GraphData;
end; (*Tinput*)
(*-----*)
(*-----*)

begin
  repeat
    clrscr;
    assign(THEFILE,'HOUTFA1.DAT');
    rewrite(THEFILE);
    new(XREAL);
    new(XIMAG);
    INVERSE:=true;
    QAM:=false;

    SelectBaud;

    if QAM then
      DisplayQAM
    else
      DisplayQPSK;

  (*Packet contruction loop*)
  for baudcount := 1 to numbauds do
    begin
      if QAM then
        SelectQAM
      else
        SelectQPSK;
      writeln('Performing IFFT');
    end
  end
end

```

```

        ComplexFFT(kx, INVERSE, XREAL, XIMAG, ERROR);
        ScaleData;
    end; (*for BAUDCOUNT*)

    dispose(XREAL);
    dispose(XIMAG);
    close(THEFILE);
    BYTECOUNT := NUMBAUDS*kx-1;
    writeln('Press return to transmit'); readln;

    Dmainit(BCST, BYTECOUNT);

    repeat
        writeln('Transmit some more? (*yes or no*) ');
        readln(ANSWER);
    until ANSWER in ['n', 'N', 'y', 'Y'];
    dmastop;
    window(1, 1, 80, 25);
    until ANSWER in ['n', 'N'];
    Tinput;
end.

```

APPENDIX F. XMITMES

```

program XMITMES;
(*Transmits a synchbaud and message from file 'MESSAGE.DAT'. The message
is encoded using QPSK. 'MESSAGE.DAT' is a text file. It should already
exist before using this program. Output is used to collect data for
TESTING*)

const
    FIRST_ELEMENT = -28929;

type
    TNvector = array[0..4095] of real;
    TNvectorPtr = ^TNvector;
    (*Sync + 61440*)
    BCSTARRAY = array[FIRST_ELEMENT..32767] of byte;

var
    kx,
    k1,k2,I,w,
    NUMBAUDS,MAXNUMBAUDS,
    BAUDCOUNT,BYTECOUNT,
    SYMBOLCOUNT,MAXNUMCHAR,
    MESSAGESIZE      : integer;
    MAGNITUDE,
    CHARACTERS_PER_BAUD : real;
    XREAL,XIMAG      : TNvectorPtr;
    INVERSE           : boolean;
    TEMPBYTE,ERROR    : byte;
    BCST              : BCSTARRAY;
    BYTEFILE          : file of byte;
    TESTFILE          : text;
    ANSWER,
    NEXTCHAR          : char;

(*$I FFT87B2. INC*)
(*$I COMPFFT. INC*)

(*-----*)
procedure SyncBaud;
(*Process the synchronization baud and stores the 256 point time domain
sequence at the beginning of the packet storage area.*)

var
    SYNCREAL, SYNCIMAG : TNvectorPtr;
    J, TEMP            : integer;
    SYNCDATA           : byte;
    SYNCMAG            : real;

begin
    new(SYNCREAL);
    new(SYNCIMAG);
    SYNCMAG:= MAGNITUDE;

```

```

(*load synchronization symbols*)
fillchar(SYNCREAL^,sizeof(SYNCREAL^),0);
fillchar(SYNCIMAG^,sizeof(SYNCIMAG^),0);
SYNCREAL^[ 68]:= -SYNCMAG; SYNCIMAG^[ 68]:= -SYNCMAG;
SYNCREAL^[ 69]:= -SYNCMAG; SYNCIMAG^[ 69]:= -SYNCMAG;
SYNCREAL^[ 70]:= -SYNCMAG; SYNCIMAG^[ 70]:=  SYNCMAG;
SYNCREAL^[ 71]:= -SYNCMAG; SYNCIMAG^[ 71]:=  SYNCMAG;
SYNCREAL^[ 72]:=  SYNCMAG; SYNCIMAG^[ 72]:= -SYNCMAG;
SYNCREAL^[ 73]:=  SYNCMAG; SYNCIMAG^[ 73]:=  SYNCMAG;
SYNCREAL^[ 74]:= -SYNCMAG; SYNCIMAG^[ 74]:=  SYNCMAG;
SYNCREAL^[ 75]:=  SYNCMAG; SYNCIMAG^[ 75]:=  SYNCMAG;
SYNCREAL^[ 76]:= -SYNCMAG; SYNCIMAG^[ 76]:=  SYNCMAG;
SYNCREAL^[ 77]:= -SYNCMAG; SYNCIMAG^[ 77]:= -SYNCMAG;
SYNCREAL^[ 78]:=  SYNCMAG; SYNCIMAG^[ 78]:= -SYNCMAG;
SYNCREAL^[ 79]:= -SYNCMAG; SYNCIMAG^[ 79]:= -SYNCMAG;
SYNCREAL^[ 80]:=  SYNCMAG; SYNCIMAG^[ 80]:=  SYNCMAG;
SYNCREAL^[ 81]:=  SYNCMAG; SYNCIMAG^[ 81]:= -SYNCMAG;
SYNCREAL^[ 82]:= -SYNCMAG; SYNCIMAG^[ 82]:= -SYNCMAG;
SYNCREAL^[ 83]:=  SYNCMAG; SYNCIMAG^[ 83]:=  SYNCMAG;

(*complex conjugate image*)
for J := 68 to 83 do
begin
    SYNCREAL^[ 256-J]:= SYNCREAL^[ J];
    SYNCIMAG^[ 256-J]:= -SYNCIMAG^[ J];
end;(*for J*)

ComplexFFT(256,INVERSE,SYNCREAL,SYNCIMAG,ERROR);

(*scale data/load time sequence*)
for J := 0 to 255 do
begin
    TEMP:=round(SYNCREAL^[ J] + 126);
    if TEMP < 0 then
        TEMP:=0;
    SYNCDATA:=TEMP;
    BCST[ J+FIRST_ELEMENT]:=SYNCDATA;
end;(*for J*)
dispose(SYNCREAL);
dispose(SYNCIMAG);
end;(*SyncBaud*)
(*-----*)
(*-----*)
procedure SelectBaud;
(*SelectBaud establishes kx, k1, and k2*)

var
    ANSWER : integer;

begin
    kx:=0;
    (*select baud size*)
    repeat
        if kx < 0 then writeln('TRY AGAIN');
        writeln('What is the length of the bauds (kx)?');

```

```

        writeln('i.e. 256, 512, 1024, 2048, 4096');
        readln(ANSWER);
        case ANSWER of
            256: kx:=256;
            512: kx:=512;
            1024: kx:=1024;
            2048: kx:=2048;
            4096: kx:=4096;
        end; (*case ANSWER*)
        if kx = 0 then kx := -1;
    until kx > 0;

    (*set tone limits*)
    case kx of
        256: begin
            k1:=68; k2:=83;
        end;
        512: begin
            k1:=135; k2:=166;
        end;
        1024: begin
            k1:=269; k2:=332;
        end;
        2048: begin
            k1:=537; k2:=664;
        end;
        4096: begin
            k1:=1073; k2:=1328;
        end;
    end; (*case kx*)
end; (*SelectBaud*)
(*-----*)
(*-----*)
procedure TailorPacket;
(*TailorPacket sets the maximum number of baud required to encode the
message*)

begin
    MESSAGE_SIZE:= filesize(BYTEFILE);
    writeln('Message is ',MESSAGE_SIZE,' bytes. ');

    (*kx/2 is the number of bit/baud for QPSK. kx/2-2 is the number
    for DQPSK. Each character is 8 bits*)
    CHARACTERS_PER_BAUD:=(kx/8 - 2)/8;

    (*61440/kx is maximum number of bauds possible*)
    MAXNUMCHAR:= trunc(61440.0/kx * CHARACTERS_PER_BAUD);
    if MESSAGE_SIZE > MAXNUMCHAR then
        begin
            writeln('Message is too large. The last ',
                MESSAGE_SIZE - MAXNUMCHAR,
                ' characters will not be transmitted. ');
            MESSAGE_SIZE:=MAXNUMCHAR;
        end;

    MAXNUMBAUDS:=trunc(MESSAGE_SIZE / CHARACTERS_PER_BAUD);

```

```

(*ensure last few characters are included*)
if frac(MESSAGE_SIZE / CHARACTERS_PER_BAUD) > 0.0 then
    MAXNUMBAUDS:=MAXNUMBAUDS + 1;

repeat
    writeln;
    writeln('Enter number of ',kx,' bauds to process. ',
            MAXNUMBAUDS,' is the maximum. ');
    readln(NUMBAUDS);
until NUMBAUDS in [ 1..MAXNUMBAUDS];
end; (*TailorPacket*)
(*-----*)
(*-----*)
procedure DiffEncode;
(*DiffEncode differential encodes symbols on a tone-to-tone basis.
  BYTEFILE is read from one byte at a time. The byte is isolated into
  2-bit groups and stored in BITS. BITS is then used to DQPSK encode the
  frequency domain arrays XREAL and XIMAG. Bytes partially encoded are
  carried over into the next baud by global variable TEMPBYTE.*)

var
    J      : integer;
    BITS   : byte;

begin
    fillchar(XREAL^,sizeof(XREAL^),0);
    fillchar(XIMAG^,sizeof(XIMAG^),0);

    (*first tone of every baud set to pi/2*)
    XREAL^[k1]:= MAGNITUDE;
    XIMAG^[k1]:= MAGNITUDE;

    if SYMBOLCOUNT = 0 then
        read(BYTEFILE,TEMPBYTE);

    (*break apart character*)
    for J:= (k1 + 1) to k2 do
        begin
            SYMBOLCOUNT:=SYMBOLCOUNT + 1;
            if frac(SYMBOLCOUNT / 4) = 0.25 then
                BITS:= (TEMPBYTE and $C0) shr 6;
            if frac(SYMBOLCOUNT / 4) = 0.5 then
                BITS:= (TEMPBYTE and $30) shr 4;
            if frac(SYMBOLCOUNT / 4) = 0.75 then
                BITS:= (TEMPBYTE and $0C) shr 2;
            if frac(SYMBOLCOUNT / 4) = 0.0 then
                begin
                    BITS:= TEMPBYTE and $03;
                    if not EOF(BYTEFILE) then
                        read(BYTEFILE,TEMPBYTE)
                    else
                        TEMPBYTE:=$40;(*fill character*)
                end;
            if (BITS < 0) and (BITS > 3) then
                writeln('Bits not assigned properly');
        end;
    end;

```

```

(*differential encode*)
  case BITS of
    0: begin XREAL^[J] := XREAL^[J-1];
             XIMAG^[J] := XIMAG^[J-1]; end;
    1: begin XREAL^[J] := -XIMAG^[J-1];
             XIMAG^[J] := XREAL^[J-1]; end;
    2: begin XREAL^[J] := XIMAG^[J-1];
             XIMAG^[J] := -XREAL^[J-1]; end;
    3: begin XREAL^[J] := -XREAL^[J-1];
             XIMAG^[J] := -XIMAG^[J-1]; end;
  end; (*case BITS*)
end; (*for J*)

(*complex conjugate image*)
for J:= k1 to k2 do
  begin
    XREAL^[kx - J] := XREAL^[J];
    XIMAG^[kx - J] := -XIMAG^[J];
    writeln(TESTFILE,BAUDCOUNT:4,J:5,trunc(XREAL^[J]):6,
            trunc(XIMAG^[J]):6);
  end;
end; (*DiffEncode*)
(*-----*)
(*-----*)
procedure ScaleData;
(*ScaleData converts each real value in XREAL down to a byte and stores
the byte in the packet storage buffer, BCST. INDEX establishes the
location in the buffer of each byte in the packet.*)

var
  INDEX,J,TEMP   : integer;
  DATA          : byte;

begin
  for J := 0 to kx-1 do
    begin
      TEMP := round(XREAL^[J] + 126);
      if TEMP < 0 then
        TEMP := 0;
      DATA := TEMP;
      (*256 is added to INDEX to start message bauds
      after the sync baud*)
      INDEX := J+(BAUDCOUNT-1)*kx+FIRST_ELEMENT+256;
      BCST[INDEX] := DATA;
    end; (*for J*)
  end; (*ScaleData*)
  (*-----*)
  (*-----*)
  procedure Dmainit(var BCST : BCSTARRAY;
                    BYTECOUNT : integer);
  (*Assembly language procedure used to initialize and unmask the DMA for
  data transfer. The source code must be converted to a BIN file.*)

  external 'DMAINIT5.BIN';
  (*-----*)

```

```

procedure Dmastop;
(*Masks DMA, stopping data transfer.*)
external 'DMASTOP.BIN';
(*-----*)
(*-----*)
begin
    clrscr;

    (*contains hex values to be encoded and transmitted*)
    assign(BYTEFILE,'MESSAGE.DAT');
    reset(BYTEFILE);

    (*Output file of encoded symbols. Used for system testing*)
    assign(TESTFILE,'XMITDAT.DAT');
    rewrite(TESTFILE);

    INVERSE:=true;
    repeat
        writeln('Enter magnitude of tones.(must be > 0.0) ');
        readln(MAGNITUDE);
    until MAGNITUDE > 0.0;

    writeln('Loading sync baud. ');
    SyncBaud;

    SelectBaud;
    TailorPacket;

    SYMBOLCOUNT:=0;
    TEMPBYTE:=$00;
    writeln('Number of bauds is ',numbauds);
    new(XREAL);
    new(XIMAG);

    for baudcount := 1 to numbauds do
        begin
            DiffEncode;
            writeln('Performing IFFT ',BAUDCOUNT,' ',
                NUMBAUDS-BAUDCOUNT,' left');
            ComplexFFT(kx,INVERSE,XREAL,XIMAG,ERROR);
            ScaleData;
        end;(*for BAUDCOUNT*)

    dispose(XREAL);
    dispose(XIMAG);
    close(BYTEFILE);
    close(TESTFILE);
    BYTECOUNT := 256 + NUMBAUDS*kx - 1;

    repeat
        writeln('Press return to transmit');readln;
        Dmainit(BCST,BYTECOUNT);
        repeat
            writeln('Transmit some more? (*yes or no*) ');
            readln(ANSWER);
        until ANSWER in ['n','N','y','Y'];

```

```

        dmastop;
until ANSWER in ['n','N'];

(*  reset(TESTFILE);
    while not EOF(TESTFILE) do
        begin
            while not EOLN(TESTFILE) do
                begin
                    read(TESTFILE,NEXTCHAR);
                    write(NEXTCHAR);
                end;
            readln(TESTFILE);
            writeln;
        end;
    close(TESTFILE);*)
end.

```

APPENDIX G. DMAINIT

```

codeseg segment
    public dmainit
    assume cs:codeseg

;procedure DMAINIT ( BLKADDRESS : XMITPOINTER;
;                   BYTECOUNT : INTEGER);
;
;this procedure initializes dma channel 3 and sets the
;parameters to output the array bcst by passing the address
;of the array start on the stack. BYTECOUNT is the number
;of bytes to transfer and is pushed on the stack by the
;calling program

        dma      equ      0
        dmapage  equ      80h

dmainit proc near
    push    bp
    mov     bp,sp
    les     di,dword ptr[bp+6] ;use bp to address stack
    mov     al,5bh             ;move add of bcst into es:di
    out     dma+11,al          ;dma chan 3 single mode, read, autoinit
    out     dma+12,al          ;reset first/last ff
    mov     ax,es               ;calc high order 4 bits of buffer area
    mov     cl,4
    rol     ax,cl
    push    ax                  ;save ax for dma start addr
    and     al,0fh
    out     dmapage+2,al        ;store in ch 3 dma page reg
    pop     ax
    and     al,0f0h
    add     ax,di               ;get page offset
    out     dma+6,al            ;output waveform buffer start addr
    mov     al,ah
    out     dma+6,al
    mov     ax,[bp+4]           ;output dma byte count
    out     dma+7,al
    mov     al,ah
    out     dma+7,al
    mov     al,3                ;unmask ch 3 to start
    out     dma+10,al
    pop     bp
    ret     6                   ;pop 6 bytes off stack for addr of bcst

dmainit endp
codeseg ends

```

APPENDIX H. DMASTOP

```
codeseg segment
    public dmastop
    assume cs:codeseg

;procedure DMASTOP;
;
;this procedure stops dma channel 3

    dma      equ      0
    dmapage  equ      80h

dmastop proc    near
    push     bp
    mov      al,7      ;mask ch 3 to stop
    out      dma+10,al
    pop      bp
    ret

dmastop endp
codeseg ends
```

APPENDIX I. RECEIVE

```

program RECEIVE;
(*This program acquires an analog signal, stores the raw data in a memory
buffer, converts the data to TYPE real, performs an FFT using COMPPFFT,
differential decodes between adjacent tones and displays the result in
the frequency domain with the phase quadrant represented by color.
Data input can also be read in from the time domain sequence file
HOUTFA1.DAT generated by program TRANSMIT*)

uses Graph, Crt, tp4dl6;

(*$I-*)
(*$R-*)

const
    kx = 256;      (*set baudlength*)
    kxml = 255;

type
    TNvector = array[0..kxml] of real; (*TYPE for real and imaginary
                                         data for FFT routing*)
    TNvectorPtr = ^TNvector;          (*Pointer for FFT data array
                                         which allows dynamic allocation
                                         of memory*)

var
    INVERSE      : boolean;
    XREAL, XIMAG : TNvectorPtr;
    ERROR        : byte;
    NUMPTS,
    k1, k2       : integer;
    ANSWER       : char;

(*$I FFT87B2. INC*)
(*$I COMPPFFT. INC*)
(*-----*)
(*-----*)
procedure AcquireData;
(*AcquireData initializes Metrobyte DASH -16F data acquisition board.
Using TTOOLS procedures D16_init and D16_ainm. Data transfer is
controlled by the DMA controller and initialized by D16_ainm, and
disabled by D16_dma_int_disable. TTOOLS procedures are external
procedures included by 'uses' tp4dl6.*)

const max_buffer = 1000;

var i:      integer;
    rate:   real;
    cnt_num, mode, cycle, trigger,
    base_adr, err_code, int_level, dma_level,
    board_num, chanlo,
    op_type, status, next_cnt, err_code_s :      integer;

```

```

    dmaPointer: pointer;
    datavector: ^integer;
    ad_data, chan_data: array[0..max_buffer] of integer;

begin
    clrscr;

    (*allocates memory on the heap and returns a pointer to the start
    of the buffer*)
    GetDMABuffer(max_buffer,dmaPointer,err_code);

    (*This statement assigns a generic pointer to a variable of a
    specific pointer type, i.e. ^integer, so that the pointer can be
    passed to the dl6_ainm routine.*)
    datavector := dmaPointer;
    board_num := 0; int_level := 7; dma_level := 1;
    base_adr := $300;

    (*initialize the driver*)
    dl6_init(board_num,base_adr,int_level,dma_level,err_code);
    chanlo := 0;
    cycle:=0; (*0 - one sweep of the DMA 1 - autoinitialize*)
    trigger:=0; (*0 - external 1 - internal*)
    cnt_num:=kx; (*# of samples*)
    rate := 10000.0; (*used for internal trigger*)
    mode := 2; (*DMA mode*)

    writeln('Ready to acquire');
    (*collects kx analog values using DMA and stores in a buffer*)
    dl6_ainm(board_num,chanlo,mode,cycle,trigger,cnt_num,rate,
            datavector^,err_code);

    (*status indicates the progress of acquisition. When all
    samples have been acquired status=0*)
    status := 11;

    (*wait until all data acquired*)
    repeat
        dl6_dma_int_status(board_num,op_type,status,next_cnt, err_code_s);
    until status = 0;
    writeln('Data received');

    if err_code <> 0 then
        dl6_print_error(err_code)
    else
        begin
            writeln('Processing data');

            (*converts left justified data, returns the true binary value
            of the sampled data*)
            dl6_convert_data(2047,cnt_num,datavector^,ad_data[0],
                            chan_data[0],0,err_code);

            new(XREAL);
            new(XIMAG);
            fillchar(XREAL^,sizeof (XREAL^),0);
            fillchar(XIMAG^,sizeof (XIMAG^),0);

```

```

    for I:= 0 to kxml do
        begin
            XREAL^[ I] := ad_data[ I] /5;
        end;
    end;

    (*stop DMA operation if in autoinitialization*)
    dl6_dma_int_disable(board_num,err_code);

    (*frees memory allocated with GetDMABuffer*)
    FreeDMABuffer(max_buffer,dmaPointer,err_code);

end;(*Acquire*)
(*-----*)
(*-----*)
procedure GetData;
(*GetData loads XREAL with the real time domain sequence generated by
program TRANSMIT. Only one baud is loaded*)

var
    THEFILE      : file of byte;
    DATA        : byte;
    I,
    TEMP         : integer;

begin
    assign(THEFILE,'HOUTFA1.DAT');
    reset(THEFILE);
    new(XREAL);
    new(XIMAG);
    fillchar(XREAL^,sizeof (XREAL^),0);
    fillchar(XIMAG^,sizeof (XIMAG^),0);
    for I := 0 to kxml do
        begin
            read(THEFILE,DATA);          (*read file one byte at a time*)
            TEMP := DATA;                (*puts byte into integer variable*)
            XREAL^[ I] := (TEMP - 126);   (*loads XREAL array*)
        end; (*for I*)
    close(THEFILE);
end; (*GetData*)
(*-----*)
(*-----*)
procedure DiffDecode;
(*DiffDecode differentially decodes complex frequency domain arrays XREAL
and XIMAG. Four decoded symbols are recombined into a byte and
transferred to file BYTESOUT.DAT.*)

var
    I,
    SYMBOLCOUNT : integer;
    TEMPREAL,TEMPIMAG : real;
    BITS,TEMPBYTE   : byte;
    TEMPCHAR        : char;
    OUTFILE         : TEXT;

begin

```

```

assign(OUTFILE, 'BYTESOUT.DAT');
rewrite(OUTFILE);
SYMBOLCOUNT:=0;
TEMPBYTE:=0;
for I := k1 to (k2 - 1) do
  begin
    (*Complex multiply two adjacent tones, I and the complex conjugate of
    I+1. This will give the phase difference between the two tones.
    The answer is in rectangular notation*)
    TEMPREAL:=XREAL^[ I] * XREAL^[ I+1] + XIMAG^[ I] * XIMAG^[ I+1];
    TEMPIMAG:=XREAL^[ I] * XIMAG^[ I+1] - XREAL^[ I+1] * XIMAG^[ I];

    (*Complex multiply (TEMPREAL + j TEMPIMAG) and (1+j). This rotates
    the differential vector 45 degrees. XREAL and XIMAG are used to
    store the results. This eliminate the original data*)
    XREAL^[ I]:=(TEMPREAL - TEMPIMAG)/80;
    XIMAG^[ I]:=(TEMPREAL + TEMPIMAG)/80;

    if (XREAL^[ I] >= 0) and (XIMAG^[ I] > 0) then
      BITS:=$00;
    if (XREAL^[ I] < 0) and (XIMAG^[ I] > 0) then
      BITS:=$01;
    if (XREAL^[ I] < 0) and (XIMAG^[ I] <= 0) then
      BITS:=$03;
    if (XREAL^[ I] >= 0) and (XIMAG^[ I] <= 0) then
      BITS:=$02;
    SYMBOLCOUNT := SYMBOLCOUNT + 1;

    (*fill TEMPBYTE with four symbols*)
    if frac(SYMBOLCOUNT / 4) = 0.25 then
      TEMPBYTE := (BITS shl 6);
    if frac(SYMBOLCOUNT / 4) = 0.5 then
      TEMPBYTE := (BITS shl 4) or TEMPBYTE;
    if frac(SYMBOLCOUNT / 4) = 0.75 then
      TEMPBYTE := (BITS shl 2) or TEMPBYTE;
    if (frac(SYMBOLCOUNT / 4) = 0.0) then
      begin
        TEMPBYTE := BITS or TEMPBYTE;
        TEMPCHAR := chr(TEMPBYTE);
        write(OUTFILE,TEMPCHAR);
        TEMPBYTE:=0;
      end; (*if frac*)
    end; (*for I*)

    TEMPCHAR := chr(TEMPBYTE);
    write(OUTFILE,TEMPCHAR); (*puts two 0 at end of each baud*)
    close(OUTFILE);

    XREAL^[ k2]:=0;
    XIMAG^[ k2]:=0;
  end; (*DiffDecode*)
  (*-----*)
  (*-----*)
  procedure GraphData(XR, XI:TNVectorPtr);
  (*GraphData graphs complex arrays XR and XI. Two scales are used:Full
  scale, 0 to kx/2 tones and Zoom scale, k1 to k2*)

```

```

var
  grDriver,grMode,ErrCode,
  I,J,                      (*loop indices*)
  k1,k2,
  MXDIV2,
  COLOR,
  X,XZ,
  Y,YZ                      : integer;

begin
  k1 := round(kx * 67.0 / 256.0);
  k2 := round(kx * 83.0 / 256.0);
  grDriver := EGA;
  grMode := EGAHi;
  initgraph(grDriver,grMode,'c: TP4 GRAPHICS');
  ErrCode := GraphResult;
  if ErrCode <> grOK then
    begin
      writeln('Graphics error: ', GraphErrorMsg(ErrCode));
      writeln('Program aborted...');
      Halt(1);
    end;(*if ErrCode*)
  SetBKColor(black);
  SetColor(white);

  (*axis construction*)
  line(60,300,600,300);    (*x-axis for full spectrum*)
  line(68,140,600,140);   (*x-axis for zoom spectrum*)
  line(332,300,332,308);  (*x-axis scale marks*)
  line(392,300,392,308);  (*x-axis scale marks*)
  line(60,300,60,308);
  line(572,300,572,308);
  line(96,140,332,300);
  line(576,140,392,300);

  MXDIV2 := round(kx / 2);

  for I := 0 to MXDIV2 do
    begin
      if (XR^[I] >= 0) and (XI^[I] > 0) then
        SetColor(lightmagenta);
      if (XR^[I] < 0) and (XI^[I] > 0) then
        SetColor(lightgreen);
      if (XR^[I] < 0) and (XI^[I] <= 0) then
        SetColor(lightrd);
      if (XR^[I] >= 0) and (XI^[I] <= 0) then
        SetColor(yellow);

      (*zoom spectrum*)

      if (I > k1) and (I <= k2) then
        begin
          XZ := round(2*((I * 4096.0 / kx) - 1040));
          YZ := 140 - round(sqrt(sqr(XR^[I]) + sqr(XI^[I])));
          line(XZ,140,XZ,YZ);
        end;
      end;
    end;
  end;

```

```

        end;

        (*full spectrum*)

        X := round(2*(I/MXDIV2 * 256 + 30));
        Y := 300 - round(sqrt(sqr(XR^[I]) + sqr(XI^[I])));
        line(X,300,X,Y);
    end;

    (*graph labels*)
    setcolor(lightgray);
    outtextXY(60,320,'0');
    outtextXY(331,320,'k1');
    outtextXY(391,320,'k2');
    outtextXY(565,320,'kx/2');
    outtextXY(150,330,'frequency-->');
    outtextXY(20,250,'amplitude');
    readln;
    outtextXY(300,330,'Press return to continue. ');

    readln;
    CloseGraph;
end; (*GraphData*)
(*-----*)
(*-----*)
begin (*main body*)
    INVERSE := false;          (*sets forward FFT*)
    ERROR := 0;
    NUMPTS := kx;
    k1 := round(kx * 67.0 / 256.0 + 1);
    k2 := round(kx * 83.0 / 256.0);

    repeat
        writeln('Enter 1 to sample data');
        writeln('      2 to read data from an ASCII file');
        readln(ANSWER);
    until ANSWER in ['1','2'];

    if ANSWER = '1' then
        (*AcquireData samples input analog signal*)
        AcquireData
    else
        GetData;
        (*GetData reads a file of discrete sample produced in the transmitter.
        GetData is used for testing Graphics and Decoding procedures without
        the need for external hardware and Data acquisition board
        initialization*)

        writeln('Computing FFT');
        ComplexFFT (NUMPTS,INVERSE,XREAL,XIMAG,ERROR);
        DiffDecode;
        (* writeln('Error = ',ERROR,' hit the enter key');readln;*)
        GraphData(XREAL, XIMAG);
    end.

```

APPENDIX J. RECMES

```

program RECEIVER;
(*Acquires the signal. Stores it in a memory buffer. Differential decodes
  between tones. Maximum number of bauds are received. The number of
  bauds processed is a user input*)

uses Graph, Crt, tp4d16;

(*$I-*)
(*$R-*)

const MAX_BUFFER = 65500;

type
  TNvector = array-0..4096- of real; (*TYPE for real and imaginary data
                                     for FFT routing*)
  TNvectorPtr = ^TNvector;          (*Pointer for FFT data array which
                                     allows dynamic allocation of memory*)

var
  INVERSE          : boolean;
  X_AL, XIMAG      : TNvectorPtr;
  ERROR, TEMPBYTE : byte;
  J,
  k1,k2,kx,ANSWER,
  ERR_CODE, BAUDCOUNT,
  SYMBOLCOUNT,
  NUMBAUDS,
  MAXNUMBAUDS      : integer;
  MAGNITUDE,
  PHASE            : real;
  DATAVECTOR      : ^integer;
  DMAPOINTER       : pointer;
  TESTFILE,
  OUTFILE          : TEXT;

(*$I FFT87B2.INC*)
(*$I COMPFFT.INC*)
(*-----*)
(*-----*)
procedure PacketSetUp;

begin
  repeat
    clrscr;
    if kx < 0 then writeln('TRY AGAIN');
    writeln('Enter baud size ');
    readln(ANSWER);
    case ANSWER of
      256: kx:= 256;
      512: kx:= 512;
      1024: kx:=1024;
    end;
  until kx > 0;
end;

```

```

        2048: kx:=2048;
        4096: kx:=4096;
    end; (*case*)
    if kx = 0 then kx := -1;
until kx > 0;
MAXNUMBAUDS := trunc((MAX_BUFFER/2)/kx);

repeat
    writeln;
    writeln('Enter number of ', kx, ' bauds to process. ',
        MAXNUMBAUDS, ' is the maximum. ');
    readln(NUMBAUDS);
until NUMBAUDS in [1..MAXNUMBAUDS];

    k1 := round(kx * 67.0 / 256.0 + 1);
    k2 := round(kx * 83.0 / 256.0);
end; (*PacketSetUp*)
(*-----*)
(*-----*)
procedure AcquireData;
(*AcquireData initializes Metrobyte DASH-16F data acquisition board,
using TTOOLS procedure D16_int and D16_ainm. Data transfer is
controlled by the DMA controller and initialized by D16_ainm and
disabled by D16_dma_int_disable. TTOOLS procedures are external
procedures included by 'uses' tp4d16.*)

var
    RATE: real;
    I, CNT_NUM, MODE, CYCLE, TRIGGER,
    BASE_ADR, INT_LEVEL, DMA_LEVEL,
    BOARD_NUM, CHANLO,
    OP_TYPE, STATUS, NEXT_CNT, ERR_CODE_S : integer;

begin
    BOARD_NUM := 0; INT_LEVEL := 7; DMA_LEVEL := 1;
    BASE_ADR := $300;

    D16_init(BOARD_NUM, BASE_ADR, INT_LEVEL, DMA_LEVEL, ERR_CODE);

    CHANLO := 0;
    CYCLE:=0; (*0-one sweep of the DMA 1-autoinitialize*)
    TRIGGER:=0; (*0 - external 1 - internal*)
    CNT_NUM:=trunc(MAX_BUFFER / 2); (*# of samples*)
    RATE := 10000.0; (*used for internal trigger*)
    MODE := 2; (*DMA mode*)
    writeln('Ready to acquire');

    D16_ainm(BOARD_NUM, CHANLO, MODE, CYCLE, TRIGGER, CNT_NUM,
        RATE, DATAVECTOR^, ERR_CODE);

    STATUS := 11;

    (*status indicates the progress of acquisition. When all
    samples have been acquired status=0*)
    repeat
        D16_dma_int_status(BOARD_NUM, OP_TYPE, STATUS, NEXT_CNT, ERR_CODE_S);

```

```

until STATUS = 0;
writeln('Data received');

if ERR_CODE <> 0 then
    D16_print_error(ERR_CODE);
    D16_dma_int_disable(BOARD_NUM,ERR_CODE);

end;(*Acquire*)
(*-----*)
(*-----*)
procedure ConvertData;
(*ConvertData separates channel and acquired data. CHAN_DATA is not used.
  Acquired data is stored in XREAL.*)

var
    AD_DATA: array[0..4096] of integer;
    I,CHAN_DATA,ERR_CODE,
    SEGMENTPART,OFFSETPART: integer;
    NEWDATAVECTOR      : ^integer;
    TEMPPOINTER        : pointer;

begin
    writeln('Processing baud ',BAUDCOUNT);
    fillchar(XREAL^,sizeof (XREAL^),0);
    fillchar(XIMAG^,sizeof (XIMAG^),0);
    SEGMENTPART:=seg(DATAVECTOR^);
    OFFSETPART:=ofs(DATAVECTOR^) + 2 * kx * (BAUDCOUNT - 1);
    TEMPPOINTER:=ptr(SEGMENTPART,OFFSETPART);
    NEWDATAVECTOR := TEMPPOINTER;
    d16_convert_data(2047,kx,NEWDATAVECTOR^,AD_DATA[ 0] ,
                    CHAN_DATA,0,ERR_CODE);

    for I:= 0 to (kx - 1) do
        begin
            XREAL^[ I] := AD_DATA[ I] /5;
        end;
    end;(*ConvertData*)
    (*-----*)
    procedure DiffDecode;
    (*DiffDecode differentially decodes complex frequency domain arrays XREAL
      and XIMAG. Four decoded symbols are recombined into a byte and
      transferred to file BYTESOUT.DAT.*)

    var
        I                : integer;
        TEMPREAL,TEMPIMAG : real;
        BITS              : byte;
        TEMPCHAR          : char;

    begin
        for I := k1 to (k2 - 1) do
            begin
                (*Complex multiply two adjacent tones, I and the complex conjugate of I+1.
                  This will give the phase difference between the two tones. The answer

```

```

is in rectangular notation*)
    TEMPREAL:=XREAL^[ I] * XREAL^[ I+1] + XIMAG^[ I] * XIMAG^[ I+1];
    TEMPIMAG:=XREAL^[ I] * XIMAG^[ I+1] - XREAL^[ I+1] * XIMAG^[ I];

(*Complex multiply (TEMPREAL + j TEMPIMAG) and (1+j). This rotates the
differential vector pi/4 radians. XREAL and XIMAG are used to store
the results. This eliminate the original data*)
    XREAL^[ I]:=(TEMPREAL - TEMPIMAG)/80;
    XIMAG^[ I]:=(TEMPREAL + TEMPIMAG)/80;

(*decode*)
    if (XREAL^[ I] >= 0) and (XIMAG^[ I] > 0) then
        BITS:=$00;
    if (XREAL^[ I] < 0) and (XIMAG^[ I] > 0) then
        BITS:=$01;
    if (XREAL^[ I] < 0) and (XIMAG^[ I] <= 0) then
        BITS:=$03;
    if (XREAL^[ I] >= 0) and (XIMAG^[ I] <= 0) then
        BITS:=$02;
        SYMBOLCOUNT := SYMBOLCOUNT + 1;

    (*fill TEMPBYTE with four symbols*)
    if frac(SYMBOLCOUNT / 4) = 0.25 then
        TEMPBYTE := (BITS shl 6);
    if frac(SYMBOLCOUNT / 4) = 0.5 then
        TEMPBYTE := (BITS shl 4) or TEMPBYTE;
    if frac(SYMBOLCOUNT / 4) = 0.75 then
        TEMPBYTE := (BITS shl 2) or TEMPBYTE;
    if (frac(SYMBOLCOUNT / 4) = 0.0) then
        begin
            TEMPBYTE := BITS or TEMPBYTE;
            TEMPCHAR := chr(TEMPBYTE);
            write(OUTFILE,TEMPCHAR);
            TEMPBYTE:=0;
        end; (*if frac*)
    end; (*for I*)

    XREAL^[ k2]:=1.0;
    XIMAG^[ k2]:=1.0;

end; (*DiffDecode*)
(*-----*)
procedure Showmessage;
(*Showmessage reads in decoded message*)

var
    NEXTCHAR: char;

begin
    writeln;
    writeln('The message transmitted is..');
    assign(OUTFILE,'MESSAGE.DAT');
    reset(OUTFILE);
    while not EOF(OUTFILE) do
        begin
            while not EOLN(OUTFILE) do

```

```

        begin
            read(OUTFILE,NEXTCHAR);
            write(NEXTCHAR);
            end;(*while not EOLN*)
        readln(OUTFILE);
        writeln;
        end;(*while not EOF*)
    close(OUTFILE);
end;(*Showmessage*)
(*-----*)

begin (*main body*)
    GetDMABuffer(MAX_BUFFER,DMAPOINTER,ERR_CODE);

    DATAVECTOR := DMAPOINTER; (*This statement assigns a generic pointer
                                to a variable of a specific pointer type,
                                i.e. ^integer, so that the pointer can be
                                passed to the dl6_ainm routine.*)

    new(XREAL);
    new(XIMAG);

    INVERSE := false;                (*sets forward FFT*)
    ERROR := 0;
    kx:=0;

    PacketSetUp;

    (*received message output file*)
    assign(OUTFILE,'MESSAGE.DAT');
    rewrite(OUTFILE);

    (*RECSTAT file shows the computed real and imaginary values of the
    received signal and its decoded representation*)
    assign(TESTFILE,'RECSTAT.DAT');
    rewrite(TESTFILE);
    (*  writeln(TESTFILE,'Baudlength is',kx:5);*)

    SYMBOLCOUNT:=0;
    TEMPBYTE:=0;

    AcquireData;  (*AcquireData samples input analog signal*)

    for BAUDCOUNT := 1 to NUMBAUDS do
        begin
            ConvertData;
            ComplexFFT (kx,INVERSE,XREAL,XIMAG,ERROR);
            (*  writeln(TESTFILE,'Baud':11,'Tone':5,'Real':10,
                'Imaginary':10,'Mag':10,'Phase':8);*)
            for J:=k1 to k2 do
                begin
                    MAGNITUDE:=20*ln(sqrt(sqr(XREAL^[J])+
                    sqr(XIMAG^[J]))) / ln(10);

                    if (XREAL^[J] >= 0) then

```

```

        PHASE:=arctan(XIMAG^[J] / XREAL^[J]);
        if (XREAL^[J] < 0) and (XIMAG^[J] > 0) then
            PHASE:=Pi + arctan(XIMAG^[J] / XREAL^[J]);
        if (XREAL^[J] < 0) and (XIMAG^[J] <= 0) then
            PHASE:=arctan(XIMAG^[J] / XREAL^[J]) - Pi;

        writeln(TESTFILE,'R',BAUDCOUNT:4,J:5,
                XREAL^[J]:10:4,XIMAG^[J]:10:4,
                MAGNITUDE:10:4,PHASE:8:4);

    end;

DiffDecode;

for J:=k1 to k2 do
    begin
        MAGNITUDE:=sqrt(sqr(XREAL^[J])+sqr(XIMAG^[J]));
        writeln(TESTFILE,'D',BAUDCOUNT:4,J:5,
                XREAL^[J]/MAGNITUDE:10:4,
                XIMAG^[J]/MAGNITUDE:10:4);
    end;
end;

writeln(OUTFILE,chr(TEMPBYTE));

close(OUTFILE);
close(TESTFILE);
dispose(XREAL);
dispose(XIMAG);
FreeDMABuffer(MAX_BUFFER,DMAPOINTER,ERR_CODE);
Showmessage;
(* writeln('Error = ',ERROR,' hit the enter key');readln;*)
end.

```

APPENDIX K. RESPONSE

```
program RESPONSE;
(*RESPONSE outputs the channels phase response, by subtracting
the transmitted phase from the received phase for each tone.
Prior to running, set NUMTONES and K1MINUS1 for the proper
baud parameters then compile and run the program*)

var
  MAG,PHASE,
  XREAL,XIMAG :array[1..256] of real;
  K1MINUS1,
  NUMTONES,K,
  I,BAUD,Kx,Kr :integer;
  XMITPHASE,
  XR,XI       :real;
  CODE        :char;
  XMITDATA,
  RECDATA,
  CHANNELDATA :text;

begin
  assign(XMITDATA,'xmitdat1.024');
  reset(XMITDATA);

  assign(RECDATA,'recstat.dat');
  reset(RECDATA);

  assign(CHANNELDATA,'resp1024.dat');
  rewrite(CHANNELDATA);

  (*NUMTONES and K1MINUS1 should be set for the same baud parameters
  as xmitdat*.*** and recstat.dat*)
  NUMTONES:=64;
  K1MINUS1:=268;

  (*get transmitted phase and received phase*)
  while not EOF(XMITDATA)do
    begin
      for I:=1 to NUMTONES do
        begin
          readln(XMITDATA,BAUD,Kx,XREAL[ I ],XIMAG[ i ] );
          readln(RECDATA,CODE,BAUD,Kr,XR,XI,MAG[ I ],PHASE[ I ] );
        end;

        (*ensure tones match*)
        if Kx <> Kr then
          begin
            writeln('Kx <> Kr');
            readln;
          end;

        (*convert transmitted symbol into phase*)
```

```

for I:=1 to NUMTONES do
  begin
    if (XREAL[I] >= 0.0) and (XIMAG[I] >= 0.0) then
      XMITPHASE:= pi/4;
    if (XREAL[i] < 0.0) and (XIMAG[i] >= 0.0) then
      XMITPHASE:= 3*pi/4;
    if (XREAL[I] >= 0.0) and (XIMAG[I] < 0.0) then
      XMITPHASE:= -pi/4;
    if (XREAL[I] < 0.0) and (XIMAG[I] < 0.0) then
      XMITPHASE:= -3*pi/4;

    (*subtract transmitted phase from received phase*)
    PHASE[I]:=PHASE[I] - XMITPHASE;

    (*phase differences may jump be 2*pi radiar This helps
      but does not work in every case. Output file may have
      to be edited to 2*pi jumps*)
    if PHASE[I] > 2*pi then
      PHASE[i]:=PHASE[I] - 2*pi;
    if PHASE[i] < -2 then
      PHASE[i]:=PHASE[i] + 2*pi;

    (*actual tone numbers*)
    K:=I + K1MINUS1;

    writeln(CHANNELDATA,BAUD: 4,K: 4,MAG[I]: 10: 4,PHASE[I]: 10: 4);
    readln(RECDATA);
  end;
end;
close(XMITDATA);
close(RECDATA);
close(CHANNELDATA);

erd.

```

APPENDIX L. STATISTICS

program Statistics;
 (*to run this program the transmitter program Xmitmes must be run using
 MESSAGE.DAT. The file produced, XMITDAT.DAT, must be transferred to the
 receiver computer. Then transfer the data between computers using
 Xmitmes and Receive. Receive produces the file RECSTAT.DAT. This
 program uses XMITDAT.DAT and RECSTAT.DAT to compute the mean and
 variance for each quadrant, and the SNR for each quadrant, baud and
 the overall SNR for the packet.*)

uses CRT;

type
 STATARRAY = array[1..125] of real;

var
 Kx,K,NUMBAUDS,
 I,J,BAUD,
 TONEX,TONER,
 Q1,Q2,Q3,Q4,
 BITERROR1,BITERROR2,BITERROR3,BITERROR4 : integer;

TEMPR,TEMPI,
 SNRINDB,SNRIN,
 SNROUTDB,SNROUT,
 MEANR,MEANI,VARR,VARI: real;

CODE : char;

XD,RD,SD : text;

XR,XI,RR,RI : array[1..16,1..128] of real;
 BTM,BTV,BTQ : array[1..16] of real;
 SR1,SR2,SR3,SR4,
 SI1,SI2,SI3,SI4 : STATARRAY;

STATMATIC : array[1..16,1..4,1..5] of real;

(*-----*)

procedure Stat(NPTS: integer;
 X: STATARRAY;
 var XMEAN, XVAR : real);

(*input NPTS: number of points to be used to compute the mean
 X: the array of data

output XMEAN: the mean of the data in array X
 XVAR: the variance of the data in array X*)

var
 SUM: real;
 N : integer;

```

begin
  (*Compute mean*)
  SUM:= 0.0;

  for N:= 1 to NPTS do
    SUM:= SUM + X[N];

  if NPTS > 0 then
    XMEAN:= SUM / NPTS
  else
    XMEAN:=1.0e-10;

  (*Compute variance*)
  SUM:= 0.0;

  for N:=1 to NPTS do
    SUM:= SUM + sqr(X[N] - XMEAN);

  if NPTS > 1 then
    XVAR:= SUM / (NPTS - 1)
  else
    XVAR:=1.0e-10;

end; (*Stat*)
(*-----*)
begin (*main body*)
  repeat
    write('Enter baud size. ');
    readln(Kx);
  until (Kx mod 256 = 0) and (Kx >= 256) and (Kx <= 4096);

  write('Enter input SNR in DB ');
  readln(SNRINDB);
  SNRIN := exp(SNRINDB / 10 * ln(10));

  K := trunc(Kx / 16);

  assign(XD,'XMITDAT.DAT');
  reset(XD);
  assign(RD,'RECSTAT.DAT');
  reset(RD);
  case Kx of
    256:begin assign(SD,'STAT256.DAT'); NUMBAUDS:= 16; end;
    512:begin assign(SD,'STAT512.DAT'); NUMBAUDS:= 8; end;
    1024:begin assign(SD,'STAT1024.DAT'); NUMBAUDS:= 4; end;
    2048:begin assign(SD,'STAT2048.DAT'); NUMBAUDS:= 2; end;
    4096:begin assign(SD,'STAT4096.DAT'); NUMBAUDS:= 1; end;
  end; (*case Kx*)
  rewrite(SD);

  (*read in transmitted and received data*)
  for I:= 1 to NUMBAUDS do
    begin
      for J:= 1 to K do
        readln(RD);

```

```

for J:= 1 to K do
begin
  readln(XD, BAUD, TONEX, XR[ I,J] , XI[ I,J] );
  readln(RD, CODE, BAUD, TONER, RR[ I,J] , RI[ I,J] );
  if TONEX <> TONER then
  begin
    clrscr;
    writeln('XMITDAT.DAT and RECSTAT.DAT are in error. ');
    Halt;
  end;
end; (*for J*)
end; (*for I*)
close(XD);
close(RD);

(*decode transmitted data*)
for I:= 1 to NUMBAUDS do
  for J:= 1 to (K - 1) do
  begin
    TEMPR:= XR[ I,J] * XR[ I,J+1] + XI[ I,J] * XI[ I,J+1];
    TEMPI:= XR[ I,J] * XI[ I,J+1] - XR[ I,J+1] * XI[ I,J];
    XR[ I,J]:= TEMPR - TEMPI;
    XI[ I,J]:= TEMPR + TEMPI;
  end; (*for J*)

(*store statistics for each quadrant and baud compute statistics*)
BITERROR1:=0; BITERROR2:=0; BITERROR3:=0; BITERROR4:=0;
for I:= 1 to NUMBAUDS do
begin
  Q1:=0; Q2:=0; Q3:=0; Q4:=0;
  for J:= 1 to (K - 1) do
  begin
    if (XR[ I,J] >= 0.0) and (XI[ I,J] > 0.0) then
    begin
      Q1:=Q1 + 1;
      SR1[ Q1]:= RR[ I,J];
      SI1[ Q1]:= RI[ I,J];
      if RR[ I,J] < 0.0 then
        BITERROR1:= BITERROR1 + 1;
      if RI[ I,J] <= 0.0 then
        BITERROR1:= BITERROR1 + 1;
    end;
    if (XR[ I,J] < 0.0) and (XI[ I,J] > 0.0) then
    begin
      Q2:=Q2 + 1;
      SR2[ Q2]:= RR[ I,J];
      SI2[ Q2]:= RI[ I,J];
      if RR[ I,J] >= 0.0 then
        BITERROR2:= BITERROR2 + 1;
      if RI[ I,J] <= 0.0 then
        BITERROR2:= BITERROR2 + 1;
    end;
    if (XR[ I,J] < 0.0) and (XI[ I,J] <= 0.0) then
    begin
      Q3:=Q3 + 1;
      SR3[ Q3]:= RR[ I,J];

```

```

        SI3[Q3]:= RI[I,J];
        if RR[I,J] >= 0.0 then
            BITERROR3:= BITERROR3 + 1;
        if RI[I,J] > 0.0 then
            BITERROR3:= BITERROR3 + 1;
        end;
    if (XR[I,J] >= 0.0) and (XI[I,J] <= 0.0) then
        begin
            Q4:=Q4 + 1;
            SR4[Q4]:= RR[I,J];
            SI4[Q4]:= RI[I,J];
            if RR[I,J] < 0.0 then
                BITERROR4:= BITERROR4 + 1;
            if RI[I,J] > 0.0 then
                BITERROR4:= BITERROR4 + 1;
            end;
        end;
    end; (*for J*)

```

(*STATMATIC is a 3 dimensional array. The first index is the baud number. The second is the quadrant. The third is the statistics for each quadrant.

```

STATMATIC[I,J,1] = the mean of the real parts
STATMATIC[I,J,2] = the variance of the real parts
STATMATIC[I,J,3] = the mean of the imaginary parts
STATMATIC[I,J,4] = the variance of the imaginary parts
STATMATIC[I,J,5] = the number of points transmitted in a given quadrant*)

```

```

BTM[I]:=0.0;
BTV[I]:=0.0;
BTQ[I]:=0.0;
if Q1 > 1 then
    begin
        Stat(Q1,SR1,MEANR,VARR);
        Stat(Q1,SI1,MEANI,VARI);
        STATMATIC[I,1,1]:=MEANR;
        STATMATIC[I,1,2]:=VARR;
        STATMATIC[I,1,3]:=MEANI;
        STATMATIC[I,1,4]:=VARI;
        BTM[I]:=BTM[I] + Q1*(abs(MEANR)+abs(MEANI));
        BTV[I]:=BTV[I] + (Q1-1)*(VARR+VARI);
        BTQ[I]:=BTQ[I] + 2*Q1;
    end;
STATMATIC[I,1,5]:=Q1;

```

```

if Q2 > 1 then
    begin
        Stat(Q2,SR2,MEANR,VARR);
        Stat(Q2,SI2,MEANI,VARI);
        STATMATIC[I,2,1]:=MEANR;
        STATMATIC[I,2,2]:=VARR;
        STATMATIC[I,2,3]:=MEANI;
        STATMATIC[I,2,4]:=VARI;
        BTM[I]:=BTM[I] + Q2*(abs(MEANR)+abs(MEANI));
        BTV[I]:=BTV[I] + (Q2-1)*(VARR+VARI);
        BTQ[I]:=BTQ[I] + 2*Q2;
    end;

```

```

        end;
        STATMATRIC[ I,2,5]:=Q2;

    if Q3 > 1 then
        begin
            Stat(Q3,SR3,MEANR,VARR);
            Stat(Q3,SI3,MEANI,VARI);
            STATMATRIC[ I,3,1]:=MEANR;
            STATMATRIC[ I,3,2]:=VARR;
            STATMATRIC[ I,3,3]:=MEANI;
            STATMATRIC[ I,3,4]:=VARI;
            BTM[ I]:=BTM[ I] + Q3*(abs(MEANR)+abs(MEANI));
            BTV[ I]:=BTV[ I] + (Q3-1)*(VARR+VARI);
            BTQ[ I]:=BTQ[ I] + 2*Q3;
        end;
        STATMATRIC[ I,3,5]:=Q3;

    if Q4 > 1 then
        begin
            Stat(Q4,SR4,MEANR,VARR);
            Stat(Q4,SI4,MEANI,VARI);
            STATMATRIC[ I,4,1]:=MEANR;
            STATMATRIC[ I,4,2]:=VARR;
            STATMATRIC[ I,4,3]:=MEANI;
            STATMATRIC[ I,4,4]:=VARI;
            BTM[ I]:=BTM[ I] + Q4*(abs(MEANR)+abs(MEANI));
            BTV[ I]:=BTV[ I] + (Q4-1)*(VARR+VARI);
            BTQ[ I]:=BTQ[ I] + 2*Q4;
        end;
        STATMATRIC[ I,4,5]:=Q4;
    end; (*for I*)

    (*output statistics to a file*)
    writeln(SD,'*** SNRIN= ',SNRIN:10:7,' or ',SNRINDB:10:7,' DB ***');
    writeln(SD);
    writeln(SD,' Baud length is ',Kx);

    SR2[1]:=0.0;
    SR2[2]:=0.0;
    SR2[3]:=0.0;
    for I:= 1 to NUMBAUDS do
        begin
            writeln(SD);writeln(SD);writeln(SD);writeln(SD);
            writeln(SD,'DFT statistics for baud #: ',I:4);writeln(SD);
            for J:= 1 to 4 do
                begin
                    (*real output data*)
                    writeln(SD);writeln(SD);
                    writeln(SD,'Given the phase is in quadrant ',J);
                    writeln(SD);
                    if STATMATRIC[ I,J,5] > 1 then
                        begin
                            writeln(SD,'Number of points = ':35,
                                STATMATRIC[ I,J,5]:10:5);
                            writeln(SD,'Mean of real parts = ':35,
                                STATMATRIC[ I,J,1]:10:5);
                        end;
                end;
            end;
        end;
    end;

```

```

        writeln(SD,'Variance of real parts = ':35,
                STATMATRIC[I,J,2]:10:5);
        SNROUT:=sqr(STATMATRIC[I,J,1]) / STATMATRIC[I,J,2];
        SNROUTDB:= 10.0 * ln(SNROUT) / ln(10.0);
        writeln(SD,'SNROUT of real part = ':35,
                SNROUT:10:5,' = ', SNROUTDB:10:5,' DB');
        writeln(SD,'SNROUT/SNRIN of real part = ':35,
                (SNROUT / SNRIN):10:5);

        (*imaginary output data*)
        writeln(SD);
        writeln(SD,'Mean of imag parts = ':35,
                STATMATRIC[I,J,3]:10:5);
        writeln(SD,'Variance of imag parts = ':35,
                STATMATRIC[I,J,4]:10:5);
        SNROUT:= sqr(STATMATRIC[I,J,3]) / STATMATRIC[I,J,4];
        SNROUTDB:= 10 * ln(SNROUT) / ln(10.0);
        writeln(SD,'SNROUT of imag part = ':35,
                SNROUT:10:5,' = ', SNROUTDB:10:5,' DB');
        writeln(SD,'SNROUT/SNRIN of imag part = ':35,
                (SNROUT / SNRIN):10:5);

        end (*if Statmatric*)

    else
        begin
            writeln(SD,'Quadrant has less then 2 statistic
                        points ');
            writeln(SD,'and is not used in the overall statics. ');
        end; (*else*)
    end; (*for J*)

    writeln(SD);writeln(SD);writeln(SD);
    writeln(SD,'Overall (Real + Imag) statistics for Baud #: ',I:4);

    SNROUT:= sqr(BTM[I]/BTQ[I]) / (BTV[I]/(BTQ[I]-1));

    SNROUTDB:= 10 * ln(SNROUT) / ln(10.0);
    writeln(SD,'Overall Baud SNROUT = ':35,SNROUT:10:5,' = ',
            SNROUTDB:10:5,' DB');
    writeln(SD,'Overall baud gain (SNROUT / SNRIN) = ':35,
            (SNROUT/SNRIN):10:5);

    (*store baud means and variances*)
    SR2[1]:= SR2[1] + BTM[I];
    SR2[2]:= SR2[2] + BTV[I];
    SR2[3]:= SR2[3] + BTQ[I];
    end; (*for I*)
    writeln(SD);writeln(SD);writeln(SD);writeln(SD);
    writeln(SD,'*** TOTAL OVER ALL ',NUMBAUDS,' BAUDS ABOVE BAUD ',
            Kx,' ***');
    writeln(SD);writeln(SD);
    SNROUT:=sqr(SR2[1]/SR2[3]) / (SR2[2] / (SR2[3] - 1));
    SNROUTDB:=10 * ln(SNROUT) / ln(10.0);
    writeln(SD,'Overall SNROUT = ':35,SNROUT:10:5,' = ',
            SNROUTDB:10:5,' DB');

```

```

writeln('Overall SNROUT = ':35,SNROUT:10:5,' = ',
        SNROUTDB:10:5,' DB');
writeln(SD,'Overall gain (SNROUT / SNRIN) = ':35,
        (SNROUT/SNRIN):10:5);

writeln(SD);writeln(SD);
writeln(SD,'Total possible statistical points ',(Kx/8-2)*NUMBAUDS:3:0,
        '. Actual number of statistical points ',SR2[3]:3:0);

writeln(SD);
writeln(SD,'BIT ERRORS':35);
writeln(SD,'Quadrant 1 = ':30,BITERROR1);
writeln(SD,'2 = ':30,BITERROR2);
writeln(SD,'3 = ':30,BITERROR3);
writeln(SD,'4 = ':30,BITERROR4);
BITERROR1:= BITERROR1 + BITERROR2 + BITERROR3 + BITERROR4;
writeln(SD,'Total Bit errors = ':35, BITERROR1);
writeln('Total Bit errors = ':30, BITERROR1);
close(SD);
end.(*main body*)

```

LIST OF REFERENCES

1. Paul H. Moose, "Theory of multi-frequency modulation (MFM) digital communications," Technical Report No. NPS 62-89-019, Naval Postgraduate School, Monterey, May 1989
2. Robert D. Strum and Donald E. Kirk, *First Principles of Discrete Systems and Digital Signal Processing*, Addison-Wesley Publishing Company, Massachusetts, 1988
3. Deborah E. DeFrank, "The design and operation of a multiple channel frequency synthesizer and modulation support circuit," Master's Thesis, Naval Postgraduate School, Monterey, California, 1981
4. Bernard Sklar, *Digital Communications Fundamentals and Applications*, Prentice Hall, Englewood Cliffs, New Jersey, 1988
5. Leon W. Couch II, *Digital and Analog Communication Systems*, Macmillan Publishing Company, New York, 1987
6. Tri T. Ha, *Digital Satellite Communications*, Macmillan Publishing Company, New York, 1986
7. S. B. Weinstein and P. M. Ebert, "Data transmission by frequency division multiplexing using the discrete fourier transform," *IEEE Trans. Comm.*, COM-19, pp. 628-634, October 1971
8. Roger E. Ziemer and Roger L. Peterson, *Digital Communications and Spread Spectrum Systems*, Macmillan Publishing Company, New York, 1985
9. Robert D. Childs, "High speed output interface for a multifrequency quaternary phase shift keyed signal generated on an industry standard computer", MSEE Thesis, Naval Postgraduate School, Monterey, California, December 1988.

10. *Microprocessor and Peripheral Handbook*, v.1, pp. 2-234 to 2-252, Intel Corporation, 1988
11. *DASH-16/16F Manual*, MetraByte Corporation, 1986
12. *Turbo Pascal Data Acquisition and Control Tools for Metrabyte Das-8 and Das-16*, pp. 5-30, Quinn-Curtis
13. John H. Humphrey and Gary S. Smock, "High-speed modems," *Byte*, pp. 102-113, June 1988
14. G. P. Lang and F. M. Longstaff, "A leech lattice modem," *IEEE Journal on Selected Areas in Communications*, Vol. 7, No. 6, pp. 968-973, June 1989
15. Peter Norton and John Socha, *Peter Norton's Assembly Language Book for the IBM PC*, Prentice Hall Press, New York, 1986

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Professor P. H. Moose, Code 62Me Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5000	6
5.	Commander Naval Ocean Systems Center Attn: Mr. Darrell Marsh (Code 624) San Diego, CA 92151	3
6.	LCDR Robert Childs Commander Submarine Force U. S. Pacific Fleet Pearl Harbor, HA 96860-6550	1
7.	LT Terry Gantenbein SOAC 90020 Commanding Officer Naval Submarine School Box 700 Groton, CT 06349-5700	1