Implementation of Multi-Frequency Modulation with Trellis Encoding and Viterbi Decoding Using a Digital Signal Processing Board

by

John W. Wisniewski
June 1991

Thesis Advisor: P.H. Moose

Approved for public release; distribution is unlimited.
Multi-Frequency Modulation has been the topic of several papers at NPS. In past systems the majority of time required for the generation of the MFM signal was due to the software routine used to implement the FFT. In this report a Digital Signal Processor was used to generate the FFT. The use of Trellis coding and Viterbi decoding on a Digital Signal Processor was also investigated. Assembly language programs for three encoder/decoder systems were developed. The first uses a 16 QAM signal, the second uses a 2/3 rate convolutional encoder and Viterbi decoder and the third uses the CCITT V.32 convolutional encoder and Viterbi decoder.
Implementation of Multi-Frequency Modulation
with Trellis Encoding and Viterbi Decoding
Using a Digital Signal Processing Board

by

John W. Wisniewski
Lieutenant, United States Navy
B.S.E.E., Virginia Military Institute

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1991

Author: John W. Wisniewski

Approved by: P. H. Moose, Thesis Advisor

T. T. Ha, Second Reader

Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering
ABSTRACT

Multi-Frequency Modulation has been the topic of several papers at NPS. In past systems the majority of time required for the generation of the MFM signal was due to the software routine used to implement the FFT. In this report a Digital Signal Processor was used to reduce the time needed to generate the FFT. The use of Trellis coding and Viterbi decoding on a Digital Signal Processor was also investigated. Assembly language programs for three encoder/decoder systems were developed. The first uses a 16 QAM signal, the second uses a 2/3 rate convolutional encoder and Viterbi decoder and the third uses the V.32 convolutional encoder and a Viterbi decoder.
DISCLAIMER

Some of the terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each occurrence of a trademark, all trademarks appearing in this thesis are listed below following the name of the firm holding the trademark.

Ariel Corporation .......... Ariel, PC-56, PC-56D, BUG-56
Borland International, Inc. ........ Turbo Pascal
The Math Works, Inc. ................. Matlab
Microsoft Corporation ............. Microsoft C
Motorola, Inc. ......................... Motorola

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 logical 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

4

A. MULTI-FREQUENCY MODULATION  
4

B. TRELLIS CODED MODULATION  
6

C. VITERBI DECODER  
10

## III. SYSTEM DEVELOPMENT

13

A. ENCODERS  
16

1. 16 QAM Encoder  
16

2. 2/3 Rate Convolutional Encoder  
16

3. V.32 Convolutional Encoder  
18

B. GENERATING THE MFM SIGNAL  
19

C. DECODERS  
20

1. 16 QAM DECODER  
20

2. 2/3 Rate Code Decoder  
21

3. V.32 DECODER  
22

## IV. SYSTEM OPERATION

24

## V. CONCLUSIONS

26
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>FFT</td>
<td>28</td>
</tr>
<tr>
<td>B</td>
<td>IFFT</td>
<td>32</td>
</tr>
<tr>
<td>C</td>
<td>SINE COSINE GENERATOR</td>
<td>36</td>
</tr>
<tr>
<td>D</td>
<td>16 QAM DATA FILES</td>
<td>37</td>
</tr>
<tr>
<td>E</td>
<td>16 QAM ENCODER</td>
<td>38</td>
</tr>
<tr>
<td>F</td>
<td>16 QAM BOUNDARY FILE</td>
<td>41</td>
</tr>
<tr>
<td>G</td>
<td>16 QAM DECODER</td>
<td>42</td>
</tr>
<tr>
<td>H</td>
<td>2/3 RATE CODE DATA FILES</td>
<td>47</td>
</tr>
<tr>
<td>I</td>
<td>2/3 RATE CODE ENCODER</td>
<td>48</td>
</tr>
<tr>
<td>J</td>
<td>2/3 RATE CODE BOUNDARY FILE</td>
<td>51</td>
</tr>
<tr>
<td>K</td>
<td>2/3 RATE CODE VITERBI DECODER</td>
<td>52</td>
</tr>
<tr>
<td>L</td>
<td>V.32 DATA FILES</td>
<td>62</td>
</tr>
<tr>
<td>M</td>
<td>V.32 ENCODER</td>
<td>63</td>
</tr>
</tbody>
</table>

vi
I. INTRODUCTION

There has long been an interest in frequency division multiplexing as a means of combatting impulsive noise, avoiding equalization and making fuller use of the bandwidth available. The principle for Multi-Frequency Modulation (MFM) was first used over thirty years ago in the Collins Kineplex system. Since that time MFM has been used under many names:

- multiplexed Quadrature Amplitude Modulation (QAM)
- orthogonal Frequency Division Multiplexing (FDM)
- dynamically assigned QAM
- multicarrier modulation

A parallel or multiplexed data system offers the potential to reduce some of the problems of serial systems. In order to increase the data rate in a serial system you must utilize higher order modulation or decrease the symbol interval, increasing the bandwidth, at the risk of degrading the performance of the system. In a parallel data system several sequential streams of data are transmitted simultaneously. In a classical parallel data system there are N non-overlapping subchannels with each data element occupying only a small portion of the available bandwidth.

With the continuing development of Digital Signal Processing technology there has been renewed interest in MFM for possible use in:

- General Switched Telephone Network (GSTN)
- 60-108 Khz Frequency-division Multiplexed (FDM) group band
• Cellular radio

• High speed data for transmission on the high-rate digital subscriber line (HDSL)

Multi-Frequency Modulation (MFM) can easily be implemented on a computer due to the minimal requirements of hardware to construct a transmitter or receiver. The primary components of a MFM transmitter/receiver are D/A, A/D converters and a method for computing an FFT. MFM has been the topic of several papers and projects at NPS using various techniques for data acquisition and modulation. One of the primary problems with implementing an MFM system in past projects, has been the time required to perform an FFT using current software routines. With the continuing development of Digital Signal Processing (DSP) chips there exist a great number of DSP boards complete with D/A, A/D converters, filters, and memory available for industry standard computers. This provides the capability to purchase an off the shelf item that has all the necessary components to implement an MFM transmitter/receiver. Utilizing a DSP board for the computation of the FFT's required for MFM provides a tremendous performance gain over the software routines that were used in previous projects. Once the programs are downloaded, the performance of the DSP board is independent of the performance of the host computer.

General purpose Digital Signal Processor chips, such as the Motorola DSP 56001 (which was used in the development of the programs included), have an architecture ideally suited to the rapid calculation of FFTs. The 56001 is a fixed point Digital Signal Processor which has three internal memory and address busses which allow the execution of instructions with
parallel data moves to the X and Y memory locations. This allows the rapid execution of complex computations. There are also 256 locations of X and Y Rom that contain Mu-law and A-law expansion tables as well as a full four quadrant sine wave table.

The first objective of this project is to investigate the use of a DSP for implementing an MFM system and reduce the time required to generate a MFM signal. The second objective was to investigate the use of trellis coding and Viterbi decoding using a Digital Signal Processor.
II. THEORY

A. MULTI-FREQUENCY MODULATION

A Multi-Frequency Modulated signal has a packet structure and is comprised of time and frequency slots. (Figure 1)

![Diagram of MFM Signal Packet](from Ref. 1: p.3)

The following definitions will be used for discussing MFM: [Ref. 1: pp 5-6]

- \( K \): Number of MFM tones
- \( T \): Packet length in seconds
- $\Delta T$: Baud length in seconds
- $k_x$: Baud length in number of samples
- $L$: Number of bauds per packet
- $\Delta t$: Time between samples in seconds
- $f_s=1/\Delta t$: Sampling or clock frequency for D/A and A/D conversion in Hz
- $\Delta f=1/\Delta T$: Frequency spacing (minimum) between MFM tones
- $\Phi_{ik}$: Symbol set. Phase of the $k^{th}$ tone in the $l^{th}$ baud
- $A_{ik}$: Amplitude of the $k^{th}$ tone in the $l^{th}$ baud

Each MFM packet consists of $L$ bauds of $K$ tones. The packet construction for the $l^{th}$ baud is given by:[Ref 1: pp 6-7]

$$x_l(u) = \sum_k x_{lk}(u) \quad (1)$$

where, the analog representation of each tone during the $l^{th}$ baud is given by:

$$x_{lk}(u) = A_{lk} \cos\left(2\pi k f_u + \Phi_{lk}\right) \quad ; 0 \leq u \leq \Delta T \quad (2)$$

The time $u$, is referenced from the beginning of the baud. The time at the beginning of the baud is defined as $t_0$ and the time at any given point in the packet is $t=t_0+l\Delta T+u$.

By sampling (1) and (2) at intervals $\Delta t=1/f_s$, a discrete time sampled version for the $l^{th}$ baud can be found:

$$x_l(n) = \sum_k x_{lk}(n) \quad (3)$$
where the discrete time version of each tone during the $l^{th}$ baud is given by:

$$x_{lk}(n) = A_{lk}\cos(2\pi kn/k_x + \phi_{lk}) ; 0 \leq n \leq k_x - 1$$

(4)

The value, $n$, is the discrete time referenced to the beginning of the baud. From the Sampling Theorem the maximum frequency must be less than $f_x/2$. There are a maximum of $k_x/2$ tones available spaced at intervals of $\Delta f$ between dc and $f_x/2 - \Delta f$. The value of $k$ is in the range from 0 and $k_x/2$, where $k$ is the harmonic number.

By taking the Discrete Fourier Transform (DFT) of (3):

$$X_1(k) = \sum_{k} X_{lk}(k)$$

(5)

The discrete time signal (3) can be generated by the $k_x$ point IDFT of (5):

$$x_l(n) = \text{IDFT} \{X_1(k)\}$$

(6)

It is easily seen that the $l^{th}$ baud is generated by taking the IDFT of a complex valued array of length $k_x$. The values in the array, $x_l(n)$ are the discrete time samples of the analog transmit signal. The generation of the $l^{th}$ baud is completed by clocking the $k_x$ samples out at $f_x$ samples/sec. The entire packet is completed by an L fold repetition of the procedure.

B. TRELLIS CODED MODULATION

Trellis coded modulation evolved as a combination of coding and modulation techniques for digital transmission over band limited channels. The primary advantage for using trellis coded modulation over other coding schemes is that significant coding gains can be achieved without compromising bandwidth efficiency [Ref 2:pp 5-12].
modulation schemes employ redundant nonbinary modulation in combination with a finite-state encoder to govern the selection of modulation signals and to generate coded signal sequences. A simple four state scheme can improve the reception of a digital transmission by as much 3db in additive white gaussian noise. If more complex coding schemes are used gains of 6db or more may be achieved. In order to achieve the potential gains of a trellis coded signal a soft decision decoder, such as a Viterbi decoder, must be utilized in the receiver. These gains can also be achieved without reducing the information rate or increasing the bandwidth as required by conventional error correction schemes.

For low to medium data rates (≤ 4800 bits/s), signaling methods that use independent symbol-by-symbol transmission are adequate to provide acceptably low error rates over voice grade circuits. When data rates increase in speed (≥ 9600 bits/s) the same is not true, QAM and optimal signal sets fail to provide acceptably low error rates.[Ref 3:pp 648-649]

There are many factors which comprise linear and nonlinear distortion on a voiceband channel including phase jitter, frequency offset and additive noise. It was shown in that, relative to the uncoded system, trellis codes with four and eight states provide marked improvement in performance with respect to additive noise, second and third harmonic distortion, phase jitter, impulse noise and other channel impairments.[Ref 3:pp 649-651]

For an uncoded system the binary data transmit rate is equal to m/T bits/sec, where m is the number of bits/symbol and l/T the symbol rate. In a conventional QAM system (uncoded) there are 2^s discrete symbol points (amplitude and/or phase levels) with each successive symbol transmitted
independently. The error performance depends on the minimum distance between the signal points (the larger the distance, the lower the error rate). The minimum distance at the transmitter is limited by the average power allowed on the circuit and the choice of the signal points. Even with the use of in-phase and quadrature components the minimum distance \( d_{\text{min}} \) between points decreases with the increase in the number of symbols, \( m \), and constant average power. This results in a degradation at higher transmission rates assuming that there is a constant symbol rate.

The objective of coding is to increase the effective minimum distance between the signals without increasing the average power. One method of accomplishing this is with the use of convolutional encoder. In a convolutional encoder there are \( m \) current bits, and \( v \) past bits used to develop the codeword. The \( v \) past bits define the state of the encoder and are operated on to produce \( m+j \) bits, the rate of the code is described as \( m/(m+j) \). The \( m+j \) bits require \( 2^{m+j} \) discrete channel symbols. Using a convolutional encoder the minimum distance between symbols is no longer the measure of error performance, performance is now a measure of the minimum distance between the allowed transition of symbols from one state to another.\[\text{Ref 3:p 649}\] A convolutional encoder using a shift register to provide two past bits for a 2/3 rate convolutional code is shown in Figure 2. The allowed transition between states is shown in Figure 3. A convolutional code can be described as an \((n,k,m)\) code, where \( n \) is the number of encoded bits, \( k \) the number of information bits and \( m \) the number of past bits used for encoding.
The input and output relationship are depicted by the branches on the trellis diagram. The upper branch depicts the transition with $x_0$ set to "0", the lower branch depicts the transition with $x_0$ set to "1". Each node of the trellis diagram represents one of four states created by the past bits $s_1$ and $s_2$.

In order to achieve the optimum decoding gains from the use of convolutional encoders the decoder must use a trace back routine to find
the most probable path through the trellis. The rules for bit to symbol mapping for coded systems:[Ref 3:pp 650]

1. All parallel transitions in the trellis structure receive maximum possible Euclidean distance in the signal constellation.

2. All transitions diverging from a merging into a trellis state receive maximum possible Euclidean distance.

C. VITERBI DECODER

The Viterbi decoder utilizes a maximum likelihood sequence estimation method to decode the incoming data stream. A predetermined measure is used to determine the symbol sequence which is closest to the received symbol sequence. At any time, k, the shortest path, called the survivor, entering each state (node) of the trellis is retained. To proceed to time k+1, all time k survivors are extended by computing the metrics (lengths) of the extended path segments based on the calculated branch metrics, dependent on the branch symbols in the trellis and the value of the received sample.[Ref 3:pp 648-649] An example of a trellis and a path through the trellis is shown in Figure 4. The metrics of the remaining paths are computed and the shortest path retained. The shortest length into each state, the (k+1) survivor is retained. The number of survivors never exceeds the number of states in the trellis.

The basic operations required in a soft decision Viterbi decoder are:[Ref 3:p 651]

- computation of branch metrics, for an additive white gaussian noise channel, are proportional to \((r-x_i)^2\) where \(r\) is the received sample and \(x_i\) is the noise free symbol associated with the message.

- addition of branch metrics and survivors to determine the survivors for each state.
comparison along the extended path metrics to determine the survivor for each state.

It was shown that the Viterbi algorithm was equivalent to a dynamic programming solution to the problem of finding the shortest path through a weighted graph [Ref 3:pp 317-321]. The decoder must produce an estimate \( \hat{v} \) of the codeword \( v \) based on the received sequence \( r \). For an information sequence of length \( L \), the trellis must contain \( L+m+1 \) levels or time units to decode the sequence. An \( (n,k,m) \) code has an information sequence of length \( K1 \), and is encoded into a codeword of length \( N=n(L+m) \). A maximum likelihood decoder (MLD) for a discrete memoryless channel (DMC) chooses
\( \hat{v} \) as the codeword \( v \) which maximizes the log-likelihood function \( \log P(r|v) \). Since for a DMC:

\[
P(z|v) = \prod_{i=0}^{L-1} P(x_{i} | v_{i}) - \prod_{i=0}^{N-1} P(x_{i} | v_{i})  \tag{7}
\]

it follows that log-likelihood function is formed by the summation of the branch metrics:

\[
\log P(z|v) = \sum_{i=0}^{L-1} \log P(x_{i} | v_{i}) - \sum_{i=0}^{N-1} \log P(x_{i} | v_{i})  \tag{8}
\]

where \( P(r_{i}|v_{i}) \) is a channel transition probability. This is a minimum error probability decoding rule when all code words are equally likely. The log-likelihood function \( \log P(r|v) \) are called the metric of path \( v \) and is denoted \( M(r|v) \). A partial path metric is formed by summing up the partial path metrics for \( j \) branches and is expressed by:

\[
M([r|v],j) = \sum_{i=0}^{j-1} M(x_{i} | v_{i})  \tag{9}
\]

The final survivor \( \hat{v} \) in the Viterbi algorithm is the maximum likelihood path;

\[
M(r|\hat{v}) \geq M(r|v), \quad \text{all } v \neq \hat{v}  \tag{10}
\]

The path is then traced back to determine the symbol transmitted.[Ref. 4:pp 316-318]
III. SYSTEM DEVELOPMENT

All the programs used in this thesis were run on an Ariel PC-56 DSP board. The PC-56 DSP board is an eight bit card that can be used in any industry standard computer. The primary components of the PC-56 are a 20 Mhz Motorola DSP 56001, 16k of external memory, a 24 bit bidirectional interrupt-driven port with a header for external I/O, a TLC32040 14 bit analog interface chip with built in input and output filters gain section. An Ariel PC-56D was also used for comparison, it contains a 27 Mhz Motorola DSP 56001, 64k of external memory and a NEXT compatible DB-15 port. For actual implementation of this system a DSP board with dual A/D and D/A converters is required.

The structure of the Motorola DSP 56001 makes it ideally suited for high speed communications. In the DSP chip there are 256 locations of 24 bit x and y memory that occupy the lowest 256 locations of the DSP address space. The locations from 256-511 are allocated for the on-chip ROM. The onboard ROM contains Mu-law and A-law expansion tables as well as a full four quadrant sine table. There are 512 locations of 24 bit high speed program RAM (PRAM) on the chip.[Ref 5]

A feature that makes the DSP 56001 desirable for use in this system is that it offers several addressing modes. It implements three types of arithmetic for addressing, linear, modulo and reverse carry. For each of the address registers R0-R7 there is an offset register NO-N7 and a modifier register M0-M7. The offset register contains the values to increment and decrement the address register. The modifier register
defines the type of address arithmetic to be used. For modulo arithmetic
the contents of the modifier register Mn specify the base modulus.[Ref
6:pp 5-2,5-4] The DSP 56001 also provides three different addressing
modes, register direct, address register indirect, and post or pre
increment/decrement.

The basic configuration of an MFM transmitter and receiver are shown
in Figure 5 and Figure 6. For the purpose of this project three types of
modulators were used in the investigation for use on a DSP board. In
previous systems implemented at NPS [Ref 1] and [Ref 7] the complex
conjugate of the input data was loaded into the image frequencies of the
IFFT. This resulted in only real data being generated by the IFFT. For
this project complex data was loaded into all available locations to
generate both real and complex data. The signal may be modulated using
any type of modulation scheme. For the purpose of this thesis the three
different modulators used a 16 QAM signal, a 2/3 rate convolutional
encoder with an eight PSK signal and a 4/5 rate code with a 32 QAM signal
using the CCITT modem standard V.32 convolutional encoder. The source
code for the V.32 encoder and Viterbi decoder was included in an example
manual from Motorola [Ref 8:pp A1-C2] for use on their simulator and was
modified for use on the Ariel board to generate a Multi-Frequency
Modulated signal.
Figure 5 MFM Transmitter

Figure 6 MFM Receiver
A. ENCODERS

Of the 256 bins available in the IFFT array, only 201 message symbols are carried in each baud. The other 55 bins are loaded with zeros to allow for filtering of the signal. Once the 256 bins are filled, the IFFT subroutine is called to generate the values for the MFM signal. At this point the values would be clocked out through the D/A converters.

1. 16 QAM Encoder

At initialization of the system the message to be transmitted is downloaded from the host computer as well as the programs for encoding (QAM16EN) the signal and the data for the look-up table (16QAMRE.DAT and 16QAMIM.DAT). The data is read in using four bit increments. The value read in is moved into an offset register and used to determine the coordinates of the points on the constellation in Figure 7. The real values of the constellation are stored in the \( x \) memory and the corresponding imaginary values are stored in the \( y \) memory. Once the constellation values are determined the real and complex values are stored in memory. Once one hundred samples have been read into memory the routine installs 55 zeros at the center of the array then reads the remainder of the 256 symbols. Once all 256 samples are stored the IFFT routine is called to generate the time domain signal.

2. 2/3 Rate Convolutional Encoder

At initialization of the system the message to be sent as well as the encoder (23ENCOD) and the data files (23REAL.DAT and 23IMAG.DAT) are loaded into memory. In the 2/3 rate encoder two bits are read in from the input message and using the convolutional encoder of Figure 2 a third bit
is generated. The eight PSK constellation in Figure 8 is used to transmit the message. A look up table is used to find the points on the constellation. After the points on the constellation are found, the real
and imaginary values are loaded into an array. After the first 100 samples are loaded into the array, 55 zeros are loaded into the array to allow for filtering and the remainder of symbols are read. Once all 256 values are loaded the IFFT is performed and the values are stored ready to be clocked out.

3. V.32 Convolutional Encoder

At initialization the message, encoder program (MFMENCOD) and data files (QAMREAL.DAT and QAMIMAG.DAT) are downloaded to the DSP board. For the V.32 encoder after the four bit symbol is read, the convolutional encoder Figure 9 is used to generate a fifth redundant bit.

![Figure 9 V.32 Convolutional Encoder (from Ref. 8:p 2-4)](image)

The 32 QAM constellation in Figure 10 is used for the five bit signal. A look up table is used to determine the real and imaginary values for the corresponding point on the constellation. As in the 16 QAM encoder and
the 2/3 rate code encoder, 201 encoded symbols and 55 zeros are loaded into the IFFT array and the IFFT is performed.

B. GENERATING THE MFM SIGNAL

When all 256 bins have been loaded the program calls the routine to perform the IFFT. The routine used to perform the IFFT was generated by using the relationship $x(n) = (FFT[X(k)]^4)/N$. The program to generate the FFT was included with the Ariel DSP package\(^1\). In order to speed up

\[\text{Figure 10 32 QAM Constellation} \quad (\text{from Ref. 8: p 2-3})\]

\(^1\)Unless otherwise specified, all software (C) COPYRIGHT 1989 by Ariel Corporation, Highland Park, NJ. All rights reserved. You may use the software provided by Ariel Corporation on any one computer; copy the object code into any machine readable form for your use. You may modify the software provided by Ariel and/or merge or incorporate it into any general use software program of your development except for a program of similar nature to the Ariel product. You may freely reproduce any such program of your development; however the merged or incorporated part of the Ariel provided software will continue to be subject to all other provisions of this agreement.
the routine as much as possible the values used in the look up table were conjugated and scaled by 256 to save from doing them in the modulator. By using the prescaled and conjugated values in the encoder constellations the IFFT routines require no more time than the FFT routine used in the decoder.

After computing the IFFT the real and complex values are stored in memory. By utilizing a DSP board with dual D/A converters the real and imaginary values can be clocked out at the sampling frequency \( f_x \). The design of the system may be altered to either allow transmission of each 256 symbol packet or store the entire message and transmit at the end of the message.

C. DECODERS

After using a quadrature receiver to recover the inphase and quadrature components, the signals are sampled at the same frequency, \( f_x \), as used in the encoder. These sampled values are used to determine the transmitted symbols.

1. 16 QAM DECODER

At initialization the boundary data file BOUN16.D and the decoder program QAM16DEC must be loaded onto the DSP board. To decode the 16 QAM signal the constellation was partitioned as shown in Figure 11. The magnitude of each received signal value is compared to the boundaries in the first quadrant. Once a bounded area is found the actual values are then used to determine the correct quadrant. The data file BOUN16.D is indexed and the corresponding symbol is read out.
2. 2/3 Rate Code Decoder

The use of a convolutional encoder requires the use of a soft decoder such as a Viterbi Decoder to recover the transmitted symbol. The boundary data file BOUN23.D and the Viterbi decoder program MFM23DEC are first loaded into memory and run. After computing the FFT to recover the transmitted symbols, a received point is read out of the array and the quadrant of the constellation (Figure 8) that it lies in is found. The data file BOUN23.D holds the four closest points, one in each state, to the quadrant that the received point lies in. A boundary file is used to help speed up the decoder routine, instead of computing the distance to all points in the constellation only the point closest in each state is used for computing the euclidean distance to the received point. Using the
received point and the four values found in BOUN23.D the euclidean distance to each point is computed. The accumulated distance table is updated with the distance to each state. The smallest accumulated distance is found and the trellis is traced back 16 time periods to find the most likely transmitted point. After finding the most likely point it must be decoded. The bit Y2 is discarded and using the relation Y1=X1@S1 the value of X1 is recovered by using past state information and X1=Y1@S1. The value of Y0 is output as X0. The values are stored in memory.

3. V.32 DECODER

The use of a convolutional encoder in the V.32 system requires that a Viterbi Decoder must also be used to recover the transmitted symbol. The boundary file BOUND.D and the program file MFMDECOD are first loaded into memory. The constellation must first be partitioned in such a way that the partition equally divides the distance between the four symbols in the same state. The partition used for the state 110 is shown in Figure 12. By using this method the partitions for all eight states may be imposed on the constellation resulting in 52 separate partitions as shown in Figure 13. For each partition there are eight points, one from each state, that is closest to the boundary. These eight points for each of the 52 boundaries are stored in the data file BOUND.D that is loaded into the processor at initialization. Once a bounded area has been found, the eight points are used to determine the euclidean distance from the received point to the points read from BOUND.D. The distances calculated
are used to update the accumulated distance to each state. The state with the smallest accumulated distance is found and the trellis is traced back 16 time periods to find the input state at the end of the most likely path. The closest point in that state to the input point at that time period is found and output. The most significant bit is masked off. The differential encoding used on the two most significant bits must be decoded by:

\[ Q_1^n = Y_1^n \oplus Y_1^{n-1} \quad (11) \]

\[ Q_2^n = (Q_1^n \oplus Y_1^{n-1}) \oplus Y_2^{2n-1} \oplus Y_2^n \quad (12) \]

Once the two most significant bits have been decoded the symbol is output to memory.
IV. SYSTEM OPERATION

The Digital Signal Processing boards used to run the programs that were developed did not have dual D/A, A/D converters required to fully implement an MFM system. The programs as listed, do not fully implement a MFM system. There are no subroutines included for the clocking out of samples through D/A converters in the encoders and no sampling routines in the decoders. The encoding programs read in the data to be transmitted from memory, encode the data and store the values created by the IFFT in memory. The same locations are used to store all values generated, it is assumed that the values will be clocked out once available freeing up these locations for the next baud. The values would be clocked out through the D/A converters to generate a signal. In the decoder programs, the data is read out of locations were the sampled values would be stored. The same locations are used for all sampled values, it is assumed each baud will be decoded prior to receiving the next baud. If a convolutional encoder is used, a decision must be made as to whether the next baud should be delayed to allow decoding of the previous baud, or transmit the baud when available and store the values in the decoder for off line decoding.

Once the modulation technique is chosen the appropriate constellation data files must be loaded for the look up tables used in the decoder. These files are located in Appendices D, H, L. Once the appropriate data files are loaded the encoder program and message file are loaded into memory and the encoder is run. The programs developed were run out of
Ariel's BUG-56 monitor-debugger, they can also be adapted to run as subroutines in a Microsoft C program. The monitor-debugger allowed easy access to all the chips registers and functions making running and debugging the programs simpler.

To use the decoder the appropriate boundary values must be loaded. Once the boundary values are loaded the decoder program is loaded and run. The output is stored in memory.

All of the programs listed are written in assembly language for the DSP 56001 and must be compiled into a loadable file using the Motorola assembler. All of the data files are properly formatted to be read into the correct memory locations. If the programs are run out of the BUG-56 monitor-debugger, a macro can be created to load all the files needed for an encoder or decoder using one instruction.
V. CONCLUSIONS

The time required for the encoder/decoder for a 256 sample baud is shown in Table I. The Turbo Pascal routine used in past projects.

Table I Sample Times for Programs Using 256 Sample Baud

<table>
<thead>
<tr>
<th>PROGRAMS</th>
<th>TIME (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>20 Mhz 56001 27 Mhz 56001</td>
</tr>
<tr>
<td>4096 pt complex fft</td>
<td>.023 sec .017 sec</td>
</tr>
<tr>
<td>256 pt complex fft</td>
<td>1 .7</td>
</tr>
</tbody>
</table>

MFM PROGRAMS (for 256 sample baud)

<table>
<thead>
<tr>
<th>PROGRAMS</th>
<th>TIME (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 QAM Encoder</td>
<td>2.7 2.0</td>
</tr>
<tr>
<td>16 QAM Decoder</td>
<td>2.7 2.0</td>
</tr>
<tr>
<td>2/3 convolutional encoder</td>
<td>3.2 2.4</td>
</tr>
<tr>
<td>2/3 rate code</td>
<td>10.8 8.0</td>
</tr>
<tr>
<td>Viterbi decoder</td>
<td>10.8 8.0</td>
</tr>
<tr>
<td>V.32 encoder (w/o diff enc)</td>
<td>3.6 2.7</td>
</tr>
<tr>
<td>V.32 encoder (w/o diff enc)</td>
<td>15.8 11.7</td>
</tr>
<tr>
<td>V.32 encoder</td>
<td>3.7 2.8</td>
</tr>
<tr>
<td>V.32 Viterbi decoder</td>
<td>16.9 12.5</td>
</tr>
</tbody>
</table>
required 22 seconds to perform a 4096 point complex FFT on an 8 Mhz AT computer with a coprocessor. [Ref. 9:p 18] It is readily apparent that the Digital Signal Processor has a tremendous performance advantage over normal software routine. Using Matlab benchmarks a Sun Sparc station required .34 seconds and a 20 Mhz 386 with a coprocessor required 1.16 seconds for a 4096 point complex FFT.

Multi-Frequency Modulation can be easily implemented on a Digital Signal Processor Board. The programs for implementing the 16 QAM system required the same amount of time to encode and to decode. There is little penalty for using a trellis encoder but the Viterbi decoder takes over three times longer than the encoder for the 2/3 rate code and almost four times as long for the V.32 decoder.

The use of DSP boards in implementing MFM shows much improvement over existing techniques. Suggestions for future research include running the MFM programs on DSP boards with the required dual D/A, A/D converters. A determination must be made as to whether the decoder should store the entire message or delay transmission of a baud until the previous baud is decoded. Another alternative is to use more than one processor, one to encode/decode and another to clock out/sample the data.

The versatility and speed of Digital Signal Processors make them ideally suited for use for a Multi-Frequency Modulated System. They offer much greater performance than is currently available using software routines and are can easily perform other functions by simply loading a new program. The continuing development of more DSP compatible products show that there will be greater uses of DSP technology in the future.
APPENDIX A. FFT

This program originally available on the Motorola DSP bulletin board. It is provided under a DISCLAIMER OF WARRANTY available from Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.

Radix 2, In-Place, Decimation-In-Time FFT (fast).

Last Update 18 Aug 88 Version 1.0

fft macro points,data,odata,coef
fft ident 1,0

Radix 2 Decimation in Time In-Place Fast Fourier Transform Routine

Complex input and output data
Real data in X memory
Imaginary data in Y memory
Normally ordered input data
Normally ordered output data
Coefficient lookup table
-Cosine values in X memory
-Sine values in Y memory

Macro Call - fft points,data,odata,coef

points number of points (16-32768, power of 2)
data start of data buffer
odata start of output data buffer
coef start of sine/cosine table

Alters Data ALU Registers
x1 x0 y1 y0
a2 a1 a0 a
b2 b1 b0 b

Alters Address Registers
r0 n0 m0
r1 n1 m1
n2
r4 n4 m4
r5 n5 m5
r6 n6 m6

28
move #data,r0 ;initialize input pointer
move #points/4,n0 ;initialize input and output pointers offset
move n0,n4 ;
move n0,n6 ;initialize coefficient offset
move #points-1,m0 ;initialize address modifiers
move m0,m1 ;for modulo addressing
move m0,m4
move m0,m5

; Do first and second Radix 2 FFT passes, combined as 4-point butterflies

move x:(r0)+n0,x0
tfr x0,a x:(r0)+n0,y1
do n0,_twopass
tfr y1,b x:(r0)+n0,y0
add y0,a x:(r0),x1
add x1,b r0,r4
add a,b (r0)+n0
sub b,a b,x:(r0)+n0
tfr x0,a ,x0 y:(r0),b
sub y0,a x0,x:(r0)
add y0,b x0,x:(r0)
sub a,b b,x:(r0)
tfr x0,a ,x0 y:(r4),b
add y0,a y:(r0)+n0,y0
add y0,b x0,x:(r0)+n0
add b,a y:(r0)+x0
sub a,b a,y:(r4)+n4
tfr x0,a b,y:(r4)+n4
sub y0,a x1,b
sub y1,b x:(r0)+n0,x0
add a,b x:(r0)+n0,y1
sub b,a b,y:(r4)+n4
tfr x0,a ,y:(r4)+

_twopass

; Perform all next FFT passes except last pass with triple nested DO loop

move #points/8,n1 ;initialize butterflies per group
move #4,n2 ;initialize groups per pass
move #-1,m2 ;linear addressing for r2

29
move #0,m6 ;initialize C address modifier for reverse carry (bit-reversed) addressing

do #@cvi(\log(points)/\log(2)-2.5),_end_pass ;example: 7 passes
for 1024 pt. FFT
move #data,r0 ;initialize A input

pointer
move r0,r1
move n1,r2
move r0,r4
output pointer
move (r1)+n1 ;initialize A

pointer
move r1,r5
output pointer
move #coef,r6 ;initialize B input

pointer
lua (r2)+,n0 ;initialize pointer

offsets
move n0,n4
move n0,n5
move (r2)- ;butterfly loop

count
move x:(r1),x1 y:(r6),y0 ;lookup -sine and -cosine values
move x:(r6)+n6,x0 y:(r0),b ;update C pointer, preload data
mac x1,y0,b y:(r1)+,y1
macr -x0,y1,b y:(r0),a

do n2,_end_grp
do r2,_end_bfy
subl b,a x:(r0),b ,y:(r4) ;Radix 2 DIT butterfly kernel
mac -x1,x0,b x:(r0)+,a ,y:(r5)
macr -y1,y0,b x:(r1),x1 y:(r6),y0
subl b,a b,x:(r4)+ y:(r0),b
mac x1,y0,b a,x:(r5)+ y:(r1)+,y1
macr -x0,y1,b y:(r0),a

_end_bfy
move (r1)+n1
subl b,a x:(r0),b ,y:(r4)
mac -x1,x0,b x:(r0)+n0,a ,y:(r5)
macr -y1,y0,b x:(r1),x1 y:(r6),y0
subl b,a b,x:(r4)+n4 y:(r0),b
mac x1,y0,b x:(r6)+n6,x0 y:(r1)+,y1
macr -x0,y1,b a,x:(r5)+n5 y:(r0),a

_end_grp
move nl,bl
lsr b n2,al ;divide butterflies per group by two
ls1 a b1,nl ;multiply groups per pass by two
move al,n2

_end_pass

; Do last FFT pass
;
move #2,n0 ;initialize pointer offsets
move n0,n1
move #points/4,n4 ;output pointer A offset
move n4,n5 ;output pointer B offset
move #data,r0 ;initialize A input pointer
move #odata,r4 ;initialize A output pointer
move r4,r2 ;save A output pointer
lua (r0)+,rl ;initialize B input pointer
lua (r2)+n2,r5 ;initialize B output pointer
move #0,m4 ;bit-reversed addressing for output ptr. A
move m4,m5 ;bit-reversed addressing for output ptr. B
move #coef,r6 ;initialize C input pointer
move (r5)-n5 ;predecrement output pointer
move x:(rl),x1 y:(r6),y0
move x:(r5),a y:(r0),b

do n2,_lastpass
mac x1,y0,b x:(r6)+n6,x0 y:(r1)+n1,y1 ;Radix 2 DIT butterfly
kernel
macr -x0,y1,b a,x:(r5)+n5 y:(r0),a ;with one butterfly per
group
subl b,a x:(r0),b b,y:(r4)
macr -x1,x0,b x:(r0)+n0,a a,y:(r5)
macr -y1,y0,b x:(r1),x1 y:(r6),y0
subl b,a b,x:(r4)+n4 y:(r0),b

_lastpass
move a,x:(r5)+n5
endm
APPENDIX B. IFFT

This program uses the same routine used in the program FFT. The IFFT is calculated using the relationship IFFT=FFT[X(k)]/N. It is assumed that the values used in the constellation are conjugated and prescaled by N. If the values to be used are not conjugated a routine must be added to the beginning of this routine to conjugate the values. The division by N can also be accomplished by using left shifts, but will slow down the program.

Radix 2, In-Place, Decimation-In-Time IFFT (fast).

iff 

iff 

Radix 2 Decimation in Time In-Place Inverse Fast Fourier Transform Routine

Complex input and output data
Real data in X memory
Conjugated and prescaled imaginary data in Y memory
Normally ordered input data
Normally ordered output data
Coefficient lookup table
-Cosine values in X memory
-Sine values in Y memory

Macro Call - ifft 

points number of points (16-32768, power of 2)
data start of data buffer
odata start of output data buffer
coef start of sine/cosine table

Alters Data ALU Registers
x1 x0 y1 y0
a2 a1 a0 a
b2 b1 b0 b

Alters Address Registers
r0 n0 m0
r1 n1 m1
n2
r4 n4 m4
r5 n5 m5
; r6  n6  m6
;
; Alters Program Control Registers
; pc   sr
;
; Uses 6 locations on System Stack
;
; Latest Revision - 18 Aug-88
;
move  #data,r0 ;initialize input pointer
move  #points/4,n0 ;initialize input and output pointers offset
move  n0,n4
move  n0,n6 ;initialize coefficient offset
move  #points-1,m0 ;initialize address modifiers
move  m0,m1 ;for modulo addressing
move  m0,m4
move  m0,m5
;
; Do first and second Radix 2 IFFT passes, combined as 4-point butterflies
;
move  x:(r0)+n0,x0
move  y:(r0)+n0,y0
add  y0,a  x:(r0),x1  ;ar+cr
add  x1,b  r0,r4  ;br+dr
add  a,b   (r0)+n0  ;ar'=(ar+cr)+(br+dr)
subl b,a   b,x:(r0)+n0  ;br'=(ar+cr)-(br+dr)
tfr x0,a   a,x0  
sub  y0,a   a,y:(r0)+n0,
sub  y0,b   x0,x:(r0)
add  a,b   y:(r0)+n0,x0  ;cr'=(ar-cr)+(bi-di)
sabl b,a   b,x:(r0)  ;dr'=(ar-cr)-(bi-di)
tfr x0,a   a,x0  
add  y0,a   a,y:(r0)+n0,y0  ;bi+di
add  y0,b   x0,x:(r0)+n0
add  a,b   y:(r0)+n0,x0  ;ai'+(ai+ci)+(bi+di)
sabl b,a   a,y:(r4)+n4  ;bi'=(ai+ci)-(bi+di)
tfr x0,a   b,y:(r4)+n4
sub  y0,a   x1,b  
sub  y1,b   x:(r0)+n0,x0  ;ai-ci
add  a,b   x:(r0)+n0,y1  ;dr-br
subl b,a   b,y:(r4)+n4  ;ci'=(ai-ci)+(dr-br)
tfr x0,a   a,y:(r4)+n4

_perfallnext IFFT passes except last pass with triple nested DO loop

;
move #points/8,n1 ; initialize butterflies per group
move #4,n2 ; initialize groups per pass
move #l,m2 ; linear addressing for r2
move #m6 ; initialize C address modifier for
; reverse carry (bit-reversed) addressing

do @cvl(@log(points)/@log(2)-2.5),_end_pass ; example: 7 passes
for 1024 pt. IFFT
move #data,r0 ; initialize A input

pointer
move r0,r1
move n1,r2
move r0,r4

pointer
move (r1)+n1 ; initialize A output

pointer
move r1,r5 ; initialize B input

pointer
move #coef,r6 ; initialize B output

pointer
lua (r2)+,n0 ; initialize C input

offsets
move n0,n4
move n0,n5
move (r2)- ; butterfly loop count

values
move x:(r1),x1 y:(r6),y0 ; lookup -sine and -cosine

data
move x:(r6)+n6,x0 y:(r0),b ; update C pointer, preload

mac x1,y0,b
macr -x0,y1,b

do n2,_end_grp

kernel
mac -x1,x0,b x:(r0)+,a
macr -y1,y0,b x:(r1),x1
subl b,a b,x:(r4)+
mac x1,y0,b y:(r0),b
macr -x0,y1,b a,x:(r5)+

_end_bfy

_end_grp

move (r1)+n1
subl b,a x:(r0),b b,y:(r4) ; Radix 2 DIT butterfly

mac -x1,x0,b x:(r0)+,a
macr -y1,y0,b x:(r1),x1
subl b,a b,x:(r4)+n4 y:(r0),b
mac x1,y0,b x:(r6)+n6,x0 y:(r1)+,y1
macr -x0,y1,b a,x:(r5)+n5 y:(r0),a

move n1,b1

34
lsr  b  n2,a1  ;divide butterflies per group by two
lsr  a  b1,n1  ;multiply groups per pass by two
move  a1,n2
_end_pass
;
; Do last IFFT pass
;
move  #2,n0  ;initialize pointer offsets
move  n0,n1
move  #points/4,n4  ;output pointer A offset
move  n4,n5  ;output pointer B offset
move  #data,r0  ;initialize A input pointer
move  #odata,r4  ;initialize A output pointer
move  r4,r2  ;save A output pointer
lua  (r0)+,r1  ;initialize B input pointer
lua  (r2)+n2,r5  ;initialize B output pointer
move  #0,m4  ;bit-reversed addressing for output ptr. A
move  m4,m5  ;bit-reversed addressing for output ptr. B
move  #coef,r6  ;initialize C input pointer
move  (r5)-n5  ;predecrement output pointer
move  x:(r1),x1  y:(r6),y0
move  x:(r5),a  y:(r0),b
do  n2,_lastpass
mac  x1,y0,b  x:(r6)+n6,x0  y:(r1)+n1,y1  ;Radix 2 DIT butterfly
kernel
macr  -x0,y1,b  a,x:(r5)+n5  y:(r0),a  ;with one butterfly per
  group
subl  b,a
neg  b
move  b,y:(r4)
move  x:(r0),b
neg  a
move  a,y:(r5)
mac  -x1,x0,b  x:(r0)+n0,a
macr  -y1,y0,b  x:(r1),x1  y:(r6),y0
subl  b,a  b,x:(r4)+n4  y:(r0),b
_lastpass
move  a,x:(r5)+n5
endm
APPENDIX C. SINE COSINE GENERATOR

; This program originally available on the Motorola DSP bulletin board.
; It is provided under a DISCLAIMER OF WARRANTY available from
; Motorola DSP Operation, 6501 Wm. Cannon Drive W., Austin, Tx., 78735.

; Sine-Cosine Table Generator for FFTs.
; Last Update 25 Nov 86 Version 1.2

sincos macro points,coef
sincos ident 1,2

; sincos - macro to generate sine and cosine coefficient
; lookup tables for Decimation in Time FFT
; twiddle factors.

; points - number of points (2 - 32768, power of 2)
; coef - base address of sine/cosine table
; negative cosine value in X memory
; negative sine value in Y memory

; Latest revision - 25-Nov-86

pi equ 3.141592654
freq equ 2.0*pi/@cvf(points)

org x:coef
count set 0
dup points/2
dc -@cos(@cvf(count)*freq)
count set count+1
endm

org y:coef
count set 0
dup points/2
dc -@sin(@cvf(count)*freq)
count set count+1
endm

endm ;end of sincos macro

endm

; Latest revision - 25-Nov-86

pi equ 3.141592654
freq equ 2.0*pi/@cvf(points)

org x:coef
count set 0
dup points/2
dc -@cos(@cvf(count)*freq)
count set count+1
endm

org y:coef
count set 0
dup points/2
dc -@sin(@cvf(count)*freq)
count set count+1
endm

endm ;end of sincos macro

36
APPENDIX D. 16 QAM DATA FILES

The following two data files are prescaled (full scale constellation values divided by 256) and conjugated. Both data files must be loaded into the DSP prior to running QAM16ENC. They are properly formatted to be read into memory. They occupy the first 16 locations of x and y memory.

Data file '16QAMRE.DAT'

HX
00000000,0000000F
FFF800,FFE800,FFF800,FFE800
000800,000800,001800,001800
FFF800,FFF800,FEE800,FEE800
000800,001800,000800,001800

Data file '16QAMIM.DAT'

HY
00000000,0000000F
000800,000800,001800,001800
000800,001800,000800,001800
FFF800,FEE800,FFF800,FEE800
FFF800,FEE800,FFF800,FEE800
FFF800,FEE800,FFF800,FEE800
This program is an encoder for a 16 QAM system. A symbol consisting of four bits is read into the encoder and using a look up table outputs the values for the points on the constellation and are loaded into an array. The data files 16QAMRE.DAT and 16QAMIM.DAT must be loaded prior to running 16QAMEN. Once the 256 values are loaded into an array an IFFT is performed to generate the MFM signal.

The following registers are modified:
- r2
- r3
- r4
- r5

The following Data ALU Registers are modified:
- a0 a1 a
- x0 x1 y0
- b0 b1 b

Define memory locations to be used in the program

```
start equ $100 ; starting program location
startifft equ $350 ; starting program location for IFFT
input equ 40 ; starting location of input bits
output equ 50 ; starting location of output bits
coef equ 128 ; location of coefficients for IFFT
points equ 256 ; number of points for IFFT
data equ 768 ; location of input data for IFFT
odata equ 1280 ; location of output data for IFFT
locate equ $9C4 ; location of message to be sent (hex)

sincos points,coef
opt mex
```
org p:start
move #locate,r3
begin
move #data,r4
do #201,cod
move #output,r5
jsr readbit
move r4,xO
move #data+100,a
cmp xO,a ;check to see if 100 symbols have been read
jseq outzero ;output 55 zeros, read remaining symbols
jsr outbit
code
org p:startifft
ifft points,data,odata,coef ;macro call to perform IFFT
;jmp begin
nop
swi

; The subroutine readbit reads the input value into memory one bit at a time
readbit
move #input+3,r2
move y:(r3)+,a ;read input symbol
move #>$1,xO
do #4,loop ;symbol is read in one bit at a time
and xO,a a,xl
move a1,x:(r2)-
move xl,a
asr a
loop
rts

; The subroutine outbit outputs the values stored by readbit. This four bit symbol is used as an index into a lookup table.
outbit
move #input,r2
clr b
clr a
addl b,a
do #4,loop2 ;symbol read out one bit at a time
move x:(r2)+,b0
addl b,a
loop2
move a0,r5 ;symbol is used to index into lookup table
nop
move x:(r5),x0 ;get real constellation value
move y:(r5),y0 ;get imaginary constellation value

39
move xO,x:(r4) ;load real value into IFFT array
move yO,y:(r4)+ ;load imag value into IFFT array
rts

outzero
do #55,endzero ;subroutine loads zeros in center
move #0,xO ;55 locations of IFFT array to
move xO,x:(r4) ;allow for filtering
nop
move xO,y:(r4)+
endzero
nop
rts
APPENDIX F. 16 QAM BOUNDARY FILE

The following data file BOUN16.d contains the symbol data for each bounded area used in the 16 QAM decoder. It must be loaded into the DSP prior to running 16QAMDEC.

Data file 'BOUN16.D'

HX
00000200,0000020F
00000C,000008,000000,000004
00000D,00000A,000001,000006
00000E,000009,000002,000005
00000F,00000B,000003,000007
APPENDIX G. 16 QAM DECODER

; This program is a decoder for 16 QAM system. After computing the FFT
; of the received data the points are read out of the array one at a time
; and compared with boundaries on the constellation to determine the
; received point. The decoded symbols are then stored in memory. It is
; assumed that the sampled input data is loaded into the input data
; location (1280) of the FFT. If the sampled data is stored elsewhere a
; routine to load the FFT array will need to be added.

; The following registers are modified:

; r0  r4
; r1  r5
; r2  r6

; The following Data ALU registers are modified:

; a  b
; x0  x1
; y0  y1

qam16dec

ident 1,1
page 132,66,3,3,0
opt nomd,nomex,loc,nocex,mu,cex

include 'sincos' ;include macro for sine cosine values
include 'fft' ;include macro for calculating FFT

org 1:$0000

location dsm 16 ;storage locations for look up table
input  dsm 16 ;storage locations for input data

endlong equ *

org x:endlong
storr6 ds 1
org x:512
boundary1 ds 4 ;reserves four locations for boundary pts.
boundary2 ds 4 ;reserves four locations for boundary pts.
boundary3 ds 4 ;reserves four locations for boundary pts.
boundary4 ds 4 ;reserves four locations for boundary pts.

startfft equ $100 ;starting location for fft program

42
start equ $250 ;starting location for decoder program
points equ 256 ;number of points for FFT
coef equ 1024 ;location of coefficients for FFT
data equ 1280 ;location of input data for FFT
odata equ 1536 ;location of output data for FFT

;Full scale values for 16 QAM constellation.
four equ $200000
three equ $180000
two equ $100000
one equ $080000
zero equ $000000
mone equ $F80000
mtwo equ $F00000
mthree equ $e80000
mfour equ $e00000

sincos points,coef
opt mex

org p:startfft
fft points,data,odata,coef ;Macro call for FFT

opt cex
org p:start
jsr initialize
do #201,_endrun
jsr readdata
jsr findbound
_endrun
nop
swi

;This initialization routine initializes register and modifiers
;as well as clearing the memory.
;The constellation is also loaded into memory here.
initialize
move #$fff$m0 ;reset register to linear addressing
move #$fff$m1 ;reset register to linear addressing
move #$fff$m2 ;reset register to linear addressing
move #$fff$m4 ;reset register to linear addressing
move #$fff$m5 ;reset register to linear addressing
move #15,m6 ;set register for modulo 15 addressing
move #0,r1
cir b #0,r0
cir a r0,r5
do #50,clrmem
move a,x:(r0)+ b,y:(r5)+ ;clear first 50 memory locations
clrmem
move #input,b1
move b1,x:storr6

;Now load full scale values of the constellation in the table locations.

move #location,r0
move r0,r4
move #mone,a
move #mone,b
move a,x:(r0)+ b,y:(r4)+ ;-1 -1
move #mthree,b
move b,x:(r0)+ a,y:(r4)+ ;-3 -1
move #mthree,a
move a,x:(r0)+ b,y:(r4)+ ;-1 -3
move #mthree,a
move a,x:(r0)+ b,y:(r4)+ ;-3 -3
move #mone,a
move #one,a
move a,x:(r0)+ b,y:(r4)+ ; 1 -1
move #mthree,b
move a,x:(r0)+ b,y:(r4)+ ; 1 -3
move #three,b
move #mone,a
move b,x:(r0)+ a,y:(r4)+ ; 3 -1
move #mthree,a
move b,x:(r0)+ a,y:(r4)+ ; 3 -3
move #mthree,a
move #three,b
move #mone,x0
move #one,y1
move x0,x:(r0)+ y1,y:(r4)+ ;-1 1
move x0,x:(r0)+ y1,y:(r4)+ ;-1 3
move a,x:(r0)+ y!,y:(r4)+ ;-3 1
move a,x:(r0)+ b,y:(r4)+ ;-3 3
move #one,a
move #one,x0
move #one,y1
move x0,x:(r0)+ a,y:(r4)+ ; 1 1
move b,x:(r0)+ a,y:(r4)+ ; 3 1
move x0,x:(r0)+ y1,y:(r4)+ ; 1 3
move b,x:(r0)+ y1,y:(r4)+ ; 3 3
move #odata,x0
move #$eff,y0
move x0,y:$424 ;store location for input to decoder
move y0,y:$425 ;store location for output of decoder
rts

; The subroutine readdata reads in the data from the output of the FFT.
; The values are read out one point at a time. The values are compared to
; boundaries on a partitioned constellation to determine the received point.
readdata
move y:$424,r0
nop
move r0,y0
move #odata+100,a ;check to see if first 100 values have been read

read
cmp y0,a
jseq _delzero ;delete 55 zeros
move x:stor6,r6
move x:(r0),a ;input data is loaded into storage location
move y:(r0)+,b ;for decoding
move a,x:(r6)
move b,y:(r6)+
move r6,x:stor6
move r0,y:$424
rts

; subroutine _delzero removes the zeros installed in the encoder
_delzero
move #odata+155,r0
nop
rts

; The subroutine findbound compares the value that has been read out of the array to the boundaries on the constellation to decode the point. First the magnitude alone is used to find the correct bounded area then the signed values are used to determine the correct quadrant is used to increment the boundary pointer to find the correct point.

findbound
move x:-(r6),a ; real value is stored in a
move #boundary1,r2 ; load starting position for bounded values
move #two,y0
cmpm y0,a y:(r6),b ; compare mag of real value to two
; imaginary value is stored in b
jgt bigtwo ; x>2
cmpm y0,b x:(r2),xO ; compare mag of imag value to two
jlt continue ; x<2,y<2, load r2 with boundary1 and continue
move #four,x1
cmpm x1,b #boundary3,r2 ; compare magnitude of imag value
nop
move x:(r2),x0
jmp continue ; x<2,y>2 and y<4, load r2 with boundary4

bigtwo
 cmpm y0,b #boundary2,r2 ; x>2 y<2
 nop

move x:(r2),x0
jlt continue ;x>2 and x<4, y<2 load boundary2 and continue

cmpm x1,b #boundary4,r2
nop
move x:(r2),x0
jmp continue ;x>2, y<4 load boundary4 and continue

; This part of the routine finds the correct quadrant and updates the pointer to the correct point.

continue
clr a x:(r6),x1
cmp x1,a y:(r6),y1
jgt negx
cmp y1,a #3,n2
jgt posxnegy
posxposy
jmp outputdata ;output is in first quadrant

posxnegy
move x:(r2)+n2,x0 ;update r2 by 3, fourth quadrant
jmp outputdata

negx
cmp y1,a #1,n2
jgt negxnegy
negxposy
move x:(r2)+n2,x0 ;update r2 by 1, second quadrant
jmp outputdata

negxnegy
move #2,n2
nop
move x:(r2)+n2,x0 ;update r2 by 2, third quadrant

outputdata
move y:$425,r0
move x:(r2),a
nop
move a1,y:(r0)+ ;move decoded symbol to output location
move r0,y:$425
rts
APPENDIX H.  2/3 RATE CODE DATA FILES

The following two data files are prescaled (full scale constellation values divided by 256) and conjugated. Both data files must be loaded into the DSP prior to running 23ENCOD. They are in the proper format to be loaded into memory.

Data file '23REAL.DAT'

HX
00000000,00000007
001000,FFF000,000800,FFF800
FFF800,000800,FFF000,001000

Data file '23IMAG.DAT'

HY
00000000,00000007
FFF800,000800,FFF000,001000
FFF000,001000,FFF800,000800
APPENDIX I. 2/3 RATE CODE ENCODER

; This program is an encoder for a 2/3 rate convolutional code. A symbol
; consisting of two bits is read into the encoder, combinational
; logic is used to generate the third bit then a look up table is used
; to output the values from a point on the constellation. The data files
; 23REAL.DAT and 23IMAG.DAT must first be loaded in memory prior to
; 23ENCOD. The 256 values are loaded into an array and an IFFT is
; performed to generate the MFM signal.

; The following registers are modified;
; r0  r3
; r1  r4
; r2  r5

; The following Data ALU registers are modified;
; a  a0  a1
; b  b0
; x0  x1
; y0  y1

23encod
ident 1,1
page 132,54
opt nomd,nomex,loc

include 'sincos'
include 'ifft'

start equ $100 ;starting program location
startifft equ $350 ;starting program location for IFFT
points equ 256 ;number of points for IFFT
coef equ 128 ;location of coefficients for IFFT
data equ 768 ;location of input data for IFFT
odata equ 1280 ;location of output data for IFFT
locate equ $9C4 ;starting location for message to be sent
input equ 40 ;storage locations for input bits
output equ 50 ;storage locations for encoded bits
statement equ 60 ;storage for past bits

sincos points,coef
opt mex

org p:start
move #locate,r3
begin
  move  #data,r4
  do    #201,code
  move  #output,r5
  jsr   readbit
  jsr   encode
  move  r4,x0
  move  #data+100,a
  cmp   x0,a      ;check to see if 100 symbols have been read
  jseq  outzero   ;output 55 zeros, read remaining symbols
  jsr   outbit

code
  org   p:startifft
  ifft  points,data,odata,coef;macro call to perform IFFT

; begin

nop
swi

; The subroutine readbit reads two bits from the message and stores them
; in memory.
;
readbit
  move  #input+1,r2
  move  >$1,x0
  do    #2,loop
      and  x0,a      a,x1
  move  a1,x:(r2)-
  move  x1,a
  asr   a

loop
  rts

; The subroutine encode performs the convolutional encoding to generate
; a redundant bit.
;
encode
  move  #input,r0
  move  #output,r5
  move  #statemem,r1
  move  x:(r0)+,x0     ;read bit x0
  move  x:(r0)-,x1     ;read bit x1
  move  x:(r1)+,a      ;read past bit s1
  move  x:(r1)-,b      ;read past bit s2
  eor   x1,a           ;x1 eor s1
  eor   a,y1           ;x1 eor s1 eor s2 = y2
  eor   y1,b
  move  x0,y:(r5)      ;store x0
  move  x:(r1)+,a      ;read past bit s1
  move  x:(r1)-,x0     ;read past bit s2
eor x1,a ;xl eor sl = y1
move x0,x:(r1)+ ;move s2 to sl
move x1,x:(r1) ;move xl to sl
move a,x:(r0)+ ;temp store yl
move b,x:(r0) ;temp store y2
rts

; The subroutine outbit reads the three bits out of memory to form a
; symbol. The symbols are reconstructed by reading the bits one at a
; time added to to an empty register and shifted left until three bit
; symbol is formedA lookup table is used to get the values for the point
; on the constellation.

outbit move #input,r2
clr b
clr a y:(r5),b0
addl b,a
do #2,loop2
move x:(r2)+,b0
addl b,a

loop2
move a0,r5 ;move symbol to r5
nop

;use value in r5 to get points off the constellation
;move x:(r5),x0
move y:(r5),y0
;move constellation points to IFFT array
;move x0,x:(r4)
move y0,y:(r4)+
nop
rts

; subroutine outzero puts 55 zeros in IFFT array for filtering
;
outzero
do #55,endzero
move #0,x0
move x0,x:(r4)
nop
move x0,y:(r4)+
endzero
nop
rts
APPENDIX J. 2/3 RATE CODE BOUNDARY FILE

The following data file BOUN23.D contains the points on the constellation used to find the minimum distances to states after the proper quadrant is found.

Data file 'BOUN23.D

HX
00000200,000020F
000040,000042,000044,000047
000041,000042,000044,000046
000041,000043,000045,000046
000040,000043,000045,000047
APPENDIX K. 2/3 RATE CODE VITERBI DECODER

; This program is a Viterbi Decoder for a 2/3 rate convolutional encoder.
; There is a 16 time period delay which will approach the maximum possible
; gain for this type of encoder. There are 64 locations needed in memory
; (16 past time periods x 4 states = 64).
;
; The following registers are modified:
; r0  r5
; r1  r6
; r2  r7
; r4
;
; The following Data ALU registers are modified:
; a  b  x0  y0
; a1 b1 x1 y1
;
; The following register modifiers are used:
; m1  n0
; m5  n2
; m6

mfm23dec
ident 1,1
page 132,66,3,3,0
opt nomd,nomex,loc,nocex,mu,cex

include 'sincos'
include 'fft'

org 1:$0000

period dsm 64 ;64 storage locations
location dsm 8 ;constellation points
input  dsm 16 ;past 16 input bits
tables dsm 4 ;accumulated distance
temp   dsm 4 ;temp storage for distance table

endlong equ *
    org x:endlong
storr6 ds 1
ynow  dsm 3 ;input bits
    org y:endlong
ypast dsm 2 ;past bits
    org x:512
boundary1 ds 16 ;storage for boundary data
startfft equ $100 ;starting location for FFT
points  equ 256 ;number of points for FFT
data   equ 1280 ;location of sampled data for FFT
don e   equ 1536 ;output location for FFT
start  equ $250 ;starting location for decoder program

; Define full scale constellation values.
two    equ $100000
one    equ $080000
zero   equ $000000
mone   equ $F80000
mtwo   equ $F00000

large   equ .9
small   equ .1
offset  equ $000000 ;can be used to distort data

sincos points,coef
opt mex

org p:startfft
fft points,data,odata,coef
opt cex
org p:start
jsr initialize
do #201,_endrun
jsr readdata
jsr findmindist
jsr accumdist
jsr traceback
jsr outputdata

_endrun
nop
swi

;this initialization routine initializes register and modifiers
;as well as clearing the memory. The constellation is also loaded
;into memory here. The accumulated distance array is set so that
;state zero starts out at a value of zero and all others start out
;larger, forcing the paths to merge at the zero states.
initialize
move #$ffff,m0 ;sets linear addressing
move #63,m1   ;sets modulo 63 addressing
move #$fffF,m2 ;sets linear addressing
move #$fffF,m4 ;sets linear addressing
move #63,m5   ;sets modulo 63 addressing
move #15,m6   ;sets modulo 16 addressing
move #0,r1
clr  b  #$0, r0
clr  a  r0, r5
do #256, clrmem
    move  a, x:(r0)+  b, y:(r5)+
clrmem
    move  #tables+1, r7
    move  #$400000, al
    rep #3
    move  a1, x:(r7)+
move  #input, b1
move  b1, x: stor r6
move  #odata-16, r0
move  r0, r2
move  #0, x0
move  #0, y0
do  #16, clrreg
    move  x0, x:(r0)+
move  y0, y:(r2)+
clrreg

; Now load full scale values of the constellation in the table locations.
;
move  #location, r0  ; Real   Imag
move  r0, r4
move  #two, a
move  #one, y1
move  #mtwo, b
move  #mone, y0
move  #one, x1
move  #mone, x0
move  a, x:(r0)+  y1, y:(r4)+  ; 2   1
move  x1, x:(r0)+  a, y:(r4)+  ; -2  -1
move  x0, x:(r0)+  a, y:(r4)+  ; 1   2
move  b, x:(r0)+  y1, y:(r4)+  ; -1  -2
move  b, x:(r0)+  y0, y:(r4)+  ; -1  2
move  x1, x:(r0)+  b, y:(r4)+  ; 1  -2
move  x0, x:(r0)+  b, y:(r4)+  ; 1  -2
move  a, x:(r0)+  y0, y:(r4)+  ; 2  -1
move  #odata, x0
move  #$eff, y0
move  x0, y:$424
move  y0, y:$425
rts

; readdata reads in the data from the output of the FFT. The data is read
; in as complex points on the constellation.
readdata
  move y:$424,r0
  nop
  move r0,y0
  move #odata+100,a
  cmp y0,a
  jseq _delzero
  move x:storr6,r6
  move #offset,x0
  move x:(r0),a
  add x0,a y:(r0)+,b
  add x0,b a,x:(r6)
  move b,y:(r6)+
  move r6,x:storr6
  move r0,y:$424
  rts

_delzero
  move #odata+155,r0
  nop
  rts

; the minimum distance is found to the closest point in every state and
; stored. The values are stored so that indexing is made easier, state
; 0,2,3,1. This will greatly reduce the number of cycles needed later.
; a smoothing function is used to accumulate distances in the accumulated
; table so this minimum distance is multiplied by .1.
;
; The subroutine findmindist finds the quadrant of the received point
; which is used as a pointer into the boundary table to read the four
; closest points, one in each state.
;
findmindist
  move #boundaryl,r2
  clr a x:-(r6),xl
  cmp xl,a y:(r6),yl
  jgt negx ;x<0
  cmp yl,a #12,n2
  jgt posxnegy
  posxposy
  jmp findist

posxnegy
  move x:(r2)+n2,x0
  jmp findist

negx
  cmp yl,a #4,n2
  jgt negxnegy

negxposy
  move x:(r2)+n2,x0
  jmp findist
negxnegy
  move #8,n2
  nop
  move x:(r2)+n2,x0

; finds the distance from the received point to the points read from the boundary table.

; findist
move x:(r2)+,r0
move #tables,r4
move x:(r0),a
sub x1,a y:(r0),b
sub y1,b a,x0
mpy x0,x0,a b,y0
mac y0,y0,a x:(r2)+,r0
move #small,x0 a,y0
mpy x0,y0,a
move x:(r0),a a,y:(r4)+
sub x1,a y:(r0),b
sub y1,b a,x0 y:(r4)+,y0
mpy x0,x0,a b,y0
mac y0,y0,a x:(r2)+,r0
move #small,x0 a,y0
mpy x0,y0,a
move x:(r0),a a,y:(r4)-
sub x1,a y:(r0),b
sub y1,b a,x0y:(r4)-,y0
mpy x0,x0,a b,y0
mac y0,y0,a x:(r2)+,r0
move #small,x0 a,y0
mpy x0,y0,a
move a,y:(r4)

; the accumulated distance routine adds the smallest distance from the previously computed table for all pathes going into a state and does this for all four states. Since only certain transitions are allowed the calculations are done in a specific order to reduce delay.

; accumdist
clr a #tables,r0
move #$7fffffff,a1
move r0,r4

56
move  #temp,r2
move  #1,m0
move  m0,m4
move  #1,n1
move  n1,n5
move  r1,r5

;find minimum distance to state zero
do  #2,statezero
move  x:(r0),x0  y:(r4),b
add  x0,b
cmp  b,a
tge b,a  r0,r3
tge b,a  r4,r7
move  x:(r0)+,x0  y:(r4)+,b
statezero
move  r3,x:(r1)+n1
move  a,x:(r2)+  y:(r4)+,b
clr  a  r7,y:(r5)+n5
move  #$7fffffff,a1

;find minimum distance to state two
do  #2,statetwo
move  x:(r0),x0  y:(r4),b
add  x0,b
cmp  b,a
tge b,a  r0,r3
tge b,a  r4,r7
move  x:(r0)+,x0  y:(r4)+,b
statetwo
move  r3,x:(r1)+n1
move  a,x:(r2)+  y:(r4)+,b
clr  a  r7,y:(r5)+n5
move  #tables+2,r4
move  r4,r0
move  x:(r1)-n1,a
clr  a  x:(r1)-,b
move  #$7fffffff,a1
move  r1,r5

;find minimum distance to state one
do  #2,stateone
move  x:(r0),x0  y:(r4),b
add  x0,b
cmp  b,a
tge b,a  r0,r3
tge b,a  r4,r7
move x:(r0)+,x0    y:(r4)-,b
stateone
move r3,x:(r1)+n1
move a,x:(r2)+
clr a    r7,y:(r5)+n5
move #$7fffffff,a1

;find minimum distance to state three
do #2,statethree
move x:(r0),x0    y:(r4),b
add x0,b
cmp b,a
tge b,a    r0,r3
tge b,a    r4,r7
move x:(r0)+,x0    y:(r4)-,b
statethree
move r3,x:(r1)+n1
move a,x:(r2)+
clr a    r7,y:(r5)+n5
move #$7fffffff,a1
move (r4)+

;now move new accumulated distances into the accumulated distance
;table from the temporary table
;also find the min distance state and store in r4 which is no longer used

move    #$fffffff,m0
move    #$fffffff,m4
move    #temp,r3
move    #tables,r0
move    #large,x1
move    #2,n0
do #3,endtable
move x:(r3)+,x0
mpy    x1,x0,a
cmp a,b    a,x:(r0)+n0
tge a,b    r0,r4
endtable

move    #tables+1,r0
do #4,endtablex
move x:(r3)+,x0
mpy    x1,x0,a
cmp a,b    a,x:(r0)+n0
tge a,b    r0,r4
endtablex

;store in r0 instead of r4

move    r4,r0
move    #4,n1
move (r0)-n0
rts

;the traceback routine now goes back through every time period starting
;with the current time period and finds the state from which the path
;came from one time period previous. At the end of this search, the
;last state found will also point to the path at that state, which is the
;output of the trellis.

traceback

;find the displacement from the pointer to table and store value in n4

move #tables,n0
move (r1)-n1
lua (r0)-n0,n5
move r1,r5
do #15,endtrace
move (r1)-n1
move x:(r5+n5),r0
move r1,r5
lua (r0)-n0,n5
endtrace
move #location,r0
move y:(r5+n5),a
rts

;the output data routine unscrambles the path order and finds one
;of the two points on the constellation corresponding to the output state
;which is closest to the original input at that time period.

outputdata
move a,b
move #$b1,x0
cmp x0,a #$b2,y0
teq y0,b
cmp y0,a #$b3,x0
teq x0,b
cmp x0,a #$b1,y0
teq y0,b
move #$b5,x0
cmp x0,a #$b7,y0
teq y0,b
cmp y0,a
teq x0,b
move b,r2
move #tables,n2
move x:storr6,r6
lua (r2)-n2,n3
move n3,a
asl a
asl a
move a,n0
move r6,r3
lua (r0)+n0,r4
move #>7fffffff,xl
move r4,r0
do #4,endout
move x:(r3),a y:(r6),b
move x:(r0)+,x0 y:(r4)+,y0
sub x0,a
sub y0,b a,x0
mpy x0,x0,a b,y0
mac y0,y0,a
tfr a,b x1,a
cmp x1,b
tit b,a r0,r7
move a,xl
endout
clr a (r7)-
move #location,n0
move r7,r0
move #$f,al
lua (r0)-n0,r7
move r7,x0
and x0,a
jsr convoldec
move y:$425,r0
nop
move a0,y:(r0)+
move r0,y:$425
rts

;The subroutine convoldec decodes the received symbol by using
;combinational logic.

convoldec
move #ynow+2,r0
move #>$1,x0
move #ypast,r7
do #3,loop
and x0,a a,x1
move a1,x:(r0)-
move x1,a
asr a
loop
move x:(r0)+,a y:(r7)+,y0 ;read s1
move x:(r0)+,b y:(r7)-,y1 ;read y0 and s2
move x:(r0)-,a y:(r7)-,y1 ;read yl

nop

eor y0,a

move y1,y:(r7)+

move a,y:(r7)-

move b,x:(r0)-

move a,x:(r0)+

clr a

clr b

do #2,loop2

move x:(r0)-,b0

addl b,a

loop2

rts
APPENDIX L. V.32 DATA FILES

The following data files are prescaled (full scale constellation values divided by 256) and conjugated. Both data files must be loaded into the DSP prior to running MFMENG. They are properly formatted to be read into memory.

Data file 'QAMREAL.DAT'

HX
00000000,0000001F
FFE000,000000,000000,002000
002000,000000,000000,FFE000
FFF000,FFF000,001000,001000
001000,001000,FFF000,FFF000
FFE800,000800,FFE800,000800
001800,FFF800,001800,FFF800
000800,FFE800,000800,000800
FFF800,001800,FFF800,FFF800

Data file 'QAMIMAG.DAT'

HY
00000000,0000001F
FFF800,001800,FFF800,FFF800
000800,FFE800,000800,000800
FFE800,000800,FFF800,000800
001600,FFF800,001800,FFF800
001000,001000,FFF000,FFF000
FFF000,FFF000,001000,001000
FFE000,000000,000000,002000
002000,000000,00C000,FFE000

62
APPENDIX M. V.32 ENCODER

; This is a an encoder for V.32 standard with differential encoding.
; The encoder can also be used with the differential removed by commenting
; out the call to diff.

mfmencod
ident 1,1
page 132,54
opt nomd,nomex,loc

include 'sincos'
include 'ifft'

org x:$40
statemem ds 3 ;set 3 locations for p:ist states
input ds 4 ;set up 4 locations for input bits

org y:$40
ylpast ds 2 ;set up two locations for diff encoder
output ds 1 ;output of convolutional encoder

start equ $100 ;starting location for encoder
startifft equ $350 ;starting location for IFFT
points equ 256 ;number of points for IFFT
odata equ 1280 ;output of IFFT
data equ 768 ;input data for IFFT
coeef equ 128 ;location for coefficients for IFFT
locate equ $9C4 ;starting location for message

sincos points,coef
opt mex

org p:start
move #ylpast,r5
move #statemem,r3
move #locate,r6

begin
move #data,r7 ;reads 201 message symbols
do #201,code
move #output,r4
jsr readbit
jsr diff
jsr encode
move r7,x0
move #data+100,a
cmp x0,a
jseq outzero
jsr outbit

code
org p:startiff
iff points,data,odata,coef
jmp begin

nop
swi

; subroutine readbit reads in the four bit symbol one bit at a time
readbit
move #input+3,r2
move y:(r6)+,a
move #>$1,x0
do #4,loop
and x0,a
move a1,x:(r2)-
move xl,a
asr a
loop
rts

; subroutine diff differentially encodes the two most significant bits

diff
move #input,r1
move x:(r1)+,a
move x:(r1)-,b
eor y0,a
x0,a
eor y1,b
x1,a
move b,x:(r1)+
b,y:(r5)+
move a,x:(r1)
a,y:(r5)-
rts

; subroutine encode convolutionally encodes the four bits to generate a fifth bit

encode
move #input,r0
move #output,r4
move #statemem,r1
move x:(r0)+,xl
move x:(r1)+,a
move a,y:(r4)
and xl,a x:(r0),x0
move x:(r1)-,b
eor x0,b a,y0
eor y0,b b,y1
move b,x:(r1)+ y:(r4),b
and y1,b x0,a
move (r1)+
eor x1,a x:(r1),x0
eor x0,a y:(r4),y1
move b,y0
eor y0,a y1,x:(r1)-
move a,x:(r1)+
rts

; subroutine outbit reads the five bit symbol and uses it to index into the lookup table to get the values of the point on the constellation
outbit move #input,r2
clr b
clr a y:(r4),b0
addl b,a
do #4,loop2
move x:(r2)+,b0
addl b,a
loop2
move a0,r4
nop
move x:(r4),x0
move y:(r4),y0

; move the values read from the constellation to the input array for the IFFT
move x0,x:(r7)
move y0,y:(r7)+
nop
rts

; subroutine outzero loads 55 zeros into the IFFT array for filtering
outzero
do #55,endzero
move #0,x0
move x0,x:(r7)
nop
65
move
endzero
nop
rts
x0,y:(r7)+
APPENDIX N. V.32 BOUNDARY FILE

This data file BOUND.D contains the eight closest points (one in each state) to each of the 52 bounded areas used in the partitioned constellation. It must be loaded prior to using MFMDIFDE.

Data file 'BOUND.D'

HX
00000200,0000039F
00000212,0000086,000008B,000008D
00000213,0000095,000009A,000009E
00000214,0000086,0000089,000008F
00000215,0000095,000009A,000009E
00000216,0000086,0000089,000008F
00000217,0000095,000009A,000009E
00000218,0000086,000008B,000008D
00000219,0000095,000009A,000009E
0000021A,0000086,000008B,000008D
0000021B,0000095,000009A,000009E
0000021C,0000086,0000089,000008F
0000021D,0000095,000009A,000009E
0000021E,0000086,0000089,000008F
0000021F,0000095,000009A,000009E
00000220,0000097,0000099,000009E
00000221,0000097,0000099,000009E
00000222,0000086,000008B,000008D
00000223,0000096,000009A,000009D
00000224,0000084,000008B,000008D
00000225,0000094,000009A,000009D
00000226,0000086,0000089,000008F
00000227,0000095,0000099,000009E
00000228,0000086,0000089,000008F
00000229,0000097,0000099,000009E
0000022A,0000097,0000099,000009E
0000022B,0000084,000008B,000008D
0000022C,0000096,000009A,000009D
0000022D,0000085,000008A,000008D
0000022E,0000095,000009A,000009D
0000022F,0000085,000008A,000008D
00000230,0000085,0000088,000008F
00000231,0000095,000009A,000009E
00000232,0000086,0000089,000008E
00000233,0000097,0000099,000009E
00000234,0000086,000008B,000008D
00000235,0000097,0000099,000009E
00000236,0000097,0000099,000009E
00000237,0000085,000008A,000008D
00000238,0000085,000008A,000008D
00000239,0000094,000009A,000009D
0000023A,0000085,0000088,000008F
0000023B,0000095,0000099,000009E
0000023C,0000086,0000089,000008E
APPENDIX 0. V.32 VITERBI DECODER

;This program is a Viterbi Decoder for V.32. There is a 16
;time period delay which will approach the maximum possible
;gain for this type of encoder. If the differential encoder
;was not used in the encoder than the call to diff must be
;commented out.
;
;There are 128 memory locations allocated for path memory (8 states x 16
;time periods = 128 locations). The full scale constellation values are
;loaded into memory during the initialization routine.

mfmdecod
    ident 1,1
    page 132,66,3,3,0
    opt nomd,nomex,loc,nocex,mu,cex

    include 'sincos'
    include 'fft'

    org 1:$0000

    period dsm 128 ;128 locations for path memory
    location dsm 32 ;32 locations for constellation points
    input dsm 16 ;16 locations for input points
    tables dsm 8 ;8 locations for accumulated distance table
    temp dsm 8 ;8 temp locations for distances
    endlong equ *
    org x:endlon
    stor6 ds 1
    ynow ds 4 ;4 locations for input bits
    org y:endlon
    ypast ds 2 ;2 past bits for differential decoder

    ; 13 boundary tables with 8 points in each of the 4 quadrants
    org x:512
    boundry1 ds 32
    boundry2 ds 32
    boundry3 ds 32
    boundry4 ds 32
    boundry5 ds 32
    boundry6 ds 32
    boundry7 ds 32
    boundry8 ds 32

70
boundry9 ds 32
boundry10 ds 32
boundry11 ds 32
boundry12 ds 32
boundry13 ds 32

startfft equ $100 ; starting location for FFT routine
points equ 256 ; number of points for FFT
coef equ 1024 ; location of FFT coefficients
data equ 1280 ; location of sampled data
odata equ 1536 ; output data from FFT
start equ $250 ; starting location of decoder

; Load in full scale constellation values
four equ $200000
three equ $180000
two equ $100000
one equ $080000
zero equ $000000
mone equ $F80000
mtwo equ $F00000
mthree equ $E80000
mfour equ $E00000

large equ .9
small equ .1
offset equ $000000

sincos points, coef
opt mex

org p:startfft
fft points, data, odata, coef
opt cex
org p:start
jsr initialize
do #217, _endrun
jsr readdata
jsr findmindist
jsr accumdist
jsr traceback
jsr outputdata
_endrun
nop
swi

; this initialization routine initializes register and
; modifiers as well as clearing the memory. The constellation
; is also loaded into memory here. The accumulated distance
; array is set so that state zero starts out at a value of
; zero and all others start out larger, forcing the paths
; to merge at the zero states.

; initialize
move #$ffff, m0
move #127, m1
move #$ffff, m2
move #$ffff, m4
move #127, m5
move #15, m6
move #0, r1
clr b #$0, r0
clr a r0, r5
do #256, clrmem
move a, x:(r0)+ b, y:(r5)+
clrmem
move #tables+1, r7
move #$400000, al
rep #7
move a, x:(r7)+
move #input, b1
move b1, x:stor r6
move #odata-16, r0
move r0, r2
move #0, x0
move #0, y0
do #16, _clrreg
move x0, x:(r0)+
move y0, y:(r2)+

_clrreg

; Now load full scale values of the constellation in the table
; location.

move #location, r0
move r0, r4
move #mfour, a
move #one, b
move a, x:(r0)+ b, y:(r4)+ ; Real Imag
move #zero, a
move #mthree, b
move a, x:(r0)+ b, y:(r4)+ ; -4 1
move #one, b
move a, x:(r0)+ b, y:(r4)+ ; 0 -3
move #four, a
move #mone, b
move a, x:(r0)+ b, y:(r4)+ ; 0 1
move a, x:(r0)+ b, y:(r4)+ ; 4 1
move a, x:(r0)+ b, y:(r4)+ ; 4 -1

72
<table>
<thead>
<tr>
<th>Move</th>
<th>Instruction</th>
<th>Destination</th>
<th>Source</th>
<th>Value 1</th>
<th>Value 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move</td>
<td>#zero, a</td>
<td>b, y:(r4)</td>
<td>0</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#three, b</td>
<td>b, y:(r4)</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y:(r4)</td>
<td>-4</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#mone, b</td>
<td>b, y:(r4)</td>
<td>-2</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>y1, y:(r4)</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#two, a</td>
<td>y1, y:(r4)</td>
<td>2</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#one, b</td>
<td>b, y:(r4)</td>
<td>-2</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(0)</td>
<td>y1, y:(r4)</td>
<td>2</td>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>y1, y:(r4)</td>
<td>-2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#three, b</td>
<td>b, y:(r4)</td>
<td>-2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>y1, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#two, a</td>
<td>y1, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#one, a</td>
<td>b, y:(r4)</td>
<td>-3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>y0, y:(r4)</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#three, a</td>
<td>y0, y:(r4)</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>y0, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#one, a</td>
<td>b, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#zero, b</td>
<td>b, y0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#four, b</td>
<td>b, y:(r4)</td>
<td>1</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#mthree, x0</td>
<td>y0, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>y0, y:(r4)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#four, b</td>
<td>b, y0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y0</td>
<td>1</td>
<td>-4</td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>#mone, a</td>
<td>b, y0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Move</td>
<td>a, x:(r0)</td>
<td>b, y0</td>
<td>-1</td>
<td>-4</td>
<td></td>
</tr>
</tbody>
</table>

73
move #three,x0
move x0,x:(r0)+ y0,y:(r4)+ ; 3  0
move a,x:(r0)+ y0,y:(r4)+ ; -1  0
move #four,b
move a,x:(r0)+ b,y:(r4)+ ; -1  4
move #odata,x0
move #$eff,y0
move x0,y:$424
move y0,y:$425
rts

; readdata reads in the data from the output of the FFT.
; readdata
move y:$424,r0
nop
move r0,y0
move #odata+100,a
cmp y0,a
jseq _delzero
move x:storr6,r6
move #>offset,x0
move x:(r0),a
add x0,a y:(r0)+,b
add x0,b a,x:(r6)
move b,y:(r6)+
mover6,x:storr6
move r0,y:$424
rts

_delzero
move #odata+155,r0
nop
rts

; the minimum distance is found to the closest point in every
; state and stored. The values are stored so that indexing is
; made easier, state 0,2,3,1,4,7,6,5. This will greatly reduce
; the number of cycles needed later. A smoothing function is
; used to accumulate distances in the accumulated table so
; this minimum distance is multiplied by .1.
;
; The subroutine findmindist compares the received points to boundaries
; on the constellation. Once a bounded area is found the closest eight
; points are read out of the boundary data file and used to update the
; path distances.
; findmindist

move x:-(r6),a
move #one,x0
cmpm x0,a y:(r6),b
jgt bigone ;x>1
cmpm x0,b #boundary1,r2
jlt continue ;x<1, y<1, load r2 with boundary 1
move #two,x1
jmp continue ;and continue

bigone move #two,x1

cmpm x1,a
jgt bigtwo ;x>2, jmp to that case
cmpm x0,b #boundary2,r2
jlt continue ;x>1 and x<2, y<1 load boundary2
jmp continue ;and continue

cmpm x1,b #boundary5,r2
jlt continue ;x>1, y<2 load boundary 5 and
continue

bigtwo

cmpm x0,b #boundary3,r2
jlt continue ;x>2 and y<1 so load boundary3
;jump continue
abs a #two,y0
abs b a,x1
sub y0,a b,y1
sub x0,b
jmp continue
cmpm a,b y1,b
jgt greatery1
cmp y0,b #boundary7,r2
jmp continue

greatery1

sub y0,b x1,a
sub x0,a
jmp continue
cmpm a,b x1,a
jgt greatery2
cmp y0,a #boundary10,r2
jmp continue
cmp y0,b #boundary11,r2
jmp continue
cmp y0,a #boundary8,r2
jmp continue

greatery2

move y1,b
move #boundary9,r2
jmp continue

move #boundary12,r2
jmp continue

move #boundary13,r2
continue
clr a x:(r6),x1
cmp x1,a y:(r6),y1
jgt negx
cmp y1,a #24,n2
jgt posxnegy

posxposy
jmp findist

posxnegy
move x:(r2)+n2,x0 ;update r2 by 24
jmp findist

negx
cmp y1,a #8,n2
jgt negxnegy

degxposy
move x:(r2)+n2,x0 ;update r2 by 8
jmp findist

degxnegy
move x:(r2)+n2,x0
move x:(r2)+n2,x0

;The subroutine findist finds the euclidean distance between the received
;point and the eight points read out of the boundary table. The x and y
;coordinates are subtracted, squared and added. The square root is not
;performed.

findist
move x:(r2)+,r0
move #tables,r4
move x:(r0),a
sub x1,a y:(r0),b
sub y1,b a,x0
mpy x0,x0,a b,y0
mac y0,y0,a x:(r2)+,r0
move #small,x0 a,y0
mpy x0,y0,a
move x:(r0),a a,y:(r4)+
sub x1,a y:(r0),b
sub y1,b a,x0 y:(r4)+,y0
mpy x0,x0,a b,y0
mac y0,y0,a x:(r2)+,r0
move #small,x0 a,y0
mpy x0,y0,a y:(r4)+,b
move x:(r0),a a,y:(r4)-
sub x1,a y:(r0),b
sub y1,b a,x0 y:(r4)-,y0
mpy x0,x0,a b,y0
mac y0,y0,a x:(r2)+,r0
move #small,x0 a,y0
mpy x0,y0,a
move x:(r0), a  a, y:(r4)+
sub x1, a  y:(r0), b
sub y1, b  a, x0
mp y x0, x0, a  b, y0
mac y0, y0, a  x:(r2)+, r0
move #small, x0  a, y0
mp y x0, y0, a
move x:(r0), a  a, y:(r4)+
sub x1, a  y:(r0), b
sub y1, b  a, x0  y:(r4)+, y0
mp y x0, x0, a  b, y0
mac y0, y0, a  x:(r2)+, r0
move #small, x0  a, y0
mp y x0, y0, a
move x:(r0), a  a, y:(r4)+
sub x1, a  y:(r0), b
sub y1, b  a, x0  y:(r4)+, y0
mp y x0, x0, a  b, y0
mac y0, y0, a  x:(r2)+, r0
move #small, x0  a, y0
mp y x0, y0, a
move x:(r0), a  a, y:(r4)-
sub x1, a  y:(r0), b
sub y1, b  a, x0
mp y x0, x0, a  b, y0
mac y0, y0, a  x:(r2)+, r0
move #small, x0  a, y0
mp y x0, y0, a
move a, y:(r4)

; the accumulated distance routine adds the smallest distance
; from the previously computed table for all paths going into
; a state and does this for all eight states.

accumdist

clr a  #tables, r0
move  #$7fffffff, al
move  r0, r4
move  #temp, r2
move  #3, m0
move  m0, m4
move  #2, nl
move  n1, n5

77
move    r1,r5

;Distances in the accumulated distance table are added to distances in the
;path table and compared for the four paths. This is done by incrementing
;through a specially ordered path table.

;find minimum distance to state zero
do    #4,statetwo
move    x:(r0),x0  y:(r4),b
add     x0,b
cmp b,a
tge b,a    r0,r3
tge b,a    r4,r7
move x:(r0)+,x0  y:(r4)+,b
statezero
move r3,x:(r1)+
move a,x:(r2)+  y:(r4)+,b
clr a    r7,y:(r5)+n5
move #$7fffffff,a1

;find minimum distance to state two
do    #4,statetwo
move    x:(r0),x0  y:(r4),b
add     x0,b
cmp b,a
tge b,a    r0,r3
tge b,a    r4,r7
move x:(r0)+,x0  y:(r4)+,b
statetwo
move r3,x:(r1)+
move a,x:(r2)+  y:(r4)+,b
clr a    r7,y:(r5)+n5
move #$7fffffff,a1

;find minimum distance to state four
do    #4,statetwo
move    x:(r0),x0  y:(r4),b
add     x0,b
cmp b,a
tge b,a    r0,r3
tge b,a    r4,r7
move x:(r0)+,x0  y:(r4)+,b
statetwo
move r3,x:(r1)+
move a,x:(r2)+  y:(r4)+,b
clr a    r7,y:(r5)+n5
move #$7fffffff,a1

;find minimum distance to state six
do #4, state six
move x:(r0), x0 y:(r4), b
add x0, b
cmp b, a
tge b, a r0, r3
tge b, a r4, r7
move x:(r0)+, x0 y:(r4)-, b

state six
move r3, x:(r1)- n1
move a, x:(r2)+
move r7, y:(r5)
move #tables+4, r4
move r4, r0
move x:(r1)- n1, a
clr a x:(r1)-, b
move #$7fffff, a1
move r1, r5

; find minimum distance to state one
do #4, state one
move x:(r0), x0 y:(r4), b
add x0, b
cmp b, a
tge b, a r0, r3
tge b, a r4, r7
move x:(r0)+, x0 y:(r4)+, b

state one
move r3, x:(r1)+ n1
move a, x:(r2)+ y:(r4)+, b
clr a r7, y:(r5)+ n5
move #$7fffff, a1

; find minimum distance to state three
do #4, state three
move x:(r0), x0 y:(r4), b
add x0, b
cmp b, a
tge b, a r0, r3
tge b, a r4, r7
move x:(r0)+, x0 y:(r4)-, b

state three
move r3, x:(r1)+ n1
move a, x:(r2)+ y:(r4)+, b
clr a r7, y:(r5)+ n5
move #$7fffff, a1
move (r4)+

; find minimum distance to state five
do #4, state five
move x:(r0), x0 y:(r4), b
add x0, b
cmp b,a
tge b,a r0,r3
tge b,a r4,r7
move x:(r0)+,x0 y:(r4)-,b

statefive
move r3,x:(r1)+n1
move a,x:(r2)+ y:(r4)-,b
clr a r7,y:(r5)+n5
move #$7ffffff,al

;find minimum distance to state seven
do #4,stateseven
move x:(r0),x0 y:(r4),b
add x0,b
cmp b,a
tge b,a r0,r3
tge b,a r4,r7
move x:(r0)+,x0 y:(r4)+,b

stateseven
move r3,x:(r1)+
move a,x:(r2)+ y:(r4)+,b
clr b r7,y:(r5)+
move #$7fffffff,bl

;now move new accumulated distances into the accumulated
distance table from the temporary table also find the min
;distance state and store in r4 which is no longer used

move #$fffffff,m0
move #$fffffff,m4
move #temp,r3
move #tables,r0
move #large,x1
move #2,n0
do #4,endtable
move x:(r3)+,x0
mpy x1,x0,a
cmp a,b a,x:(r0)+n0
tge a,b r0,r4

endtable
move #tables+1,r0
do #4,endtablex
move x:(r3)+,x0
mpy x1,x0,a
cmp a,b a,x:(r0)+n0
tge a,b r0,r4

endtablex
;
;store in r0 instead of r4
;
move r4,r0

80
move #8,n1
move (r0)-n0
rts

;the traceback routine now goes back through every time period
;starting with the current time period and finds the state
;from which the path came from one time period previous. At
;the end of this search, the last state found will also point
;to the path at that state, which is the output of the
;trellis.

traceback

;find the displacement from the pointer to table and store
;value in n4

move #tables,n0
move (r1)-n1
lua (r0)-n0,n5
move r1,r5
do #15,endtrace
move (r1)-n1
move x:(r5+n5),r0
move r1,r5
lua (r0)-n0,n5
endtrace
move #location,r0
move y:(r5+n5),a
rts

;the output data routine unscrambles the path order and finds
;one of the four points on the constellation corresponding to
;the output state which is closest to the original input at
;that time period.

outputdata
move a,b
move #>$b1,x0
cmp x0,a #>$b2,y0
teq y0,b
cmp y0,a #>$b3,x0
teq x0,b
cmp x0,a #>$b1,y0
teq y0,b
move #>$b5,x0
cmp x0,a #>$b7,y0
teq y0,b
cmp y0,a
teq x0,b
move z,r2
move #/tables,n2
move x:storr6,r6
lua (r2)-n2,n3
move n3,a
asl a
asl a
move a,n0
move r6,r3
lua (r0)+n0,r4
move #>$7fffff,x1
move r4,r0
do #4,endout
move x:(r3),a   y:(r6),b
move x:(r0)+,x0 y:(r4)+,y0
sub x0,a
sub y0,b    a,x0
mpy x0,x0,a  b,y0
mac y0,y0,a
tfr a,b    x1,a
cmp x1,b
tlt b,a    r0,r7
move a,xl

endout
clr a (r7)-
move #location,n0
move r7,r0
move #5f,al
lua (r0)-n0,r7
move r7,x0
and x0,a
jsr diff
move y:$425,r0
nop
move a0,y:(r0)+
move r0,y:$425
rts

;The subroutine diff differentially decodes the two most significant bits.
;Each bit is stored in its own memory and the bits are decoded using
;Q1n = Y1n EOR Y1n-1, Q2n = (Q1n AND Y1n-1) EOR Y2n-1 EOR Y2n. The four
;bit symbol is formed and output.
;
diff
move #ynow+3,r0
move #>$1,x0
move #ypast,r7
do #4,diffloop1
and x0,a    a,x1
move a1,x:(r0)-
move x1,a
asl a

82
diffloop1
    move x:(r0)+,a  y:(r7)+,y0
    move x:(r0)+,a  y:(r7)-,y1
    move x:(r0)-,b  a,y:(r7)+
    move           b,y:(r7)-
    eor y0,a       a,x0
    eor y1,b       a,x1
    and y0,a       b,y1
    eor y1,a       x1,b
    move           b,x:(r0)+
    move           a,x:(r0)-
    clr a
    clr b
    do #4,diff2
    move           x:(r0)+,b0
    addl b,a

diff2
    rts
LIST OF REFERENCES


84
BIBLIOGRAPHY


INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Cameron Station
   Alexandria, Virginia  22304-6145
   2

2. Library, Code 52
   Naval Postgraduate School
   Monterey, California  93943-5002
   2

3. Department Chairman, Code EC
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California  93943-5100
   1

4. Professor P. H. Moose, Code EC/Me
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California  93943-5100
   6

4. Professor G. A. Myers, Code EC/Mv
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California  93943-5100
   1

5. Professor T. T. Ha, Code EC/Ha
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California  93943-5100
   1

   RD#7 Box 7102
   Moscow, Pennsylvania  18444
   1