

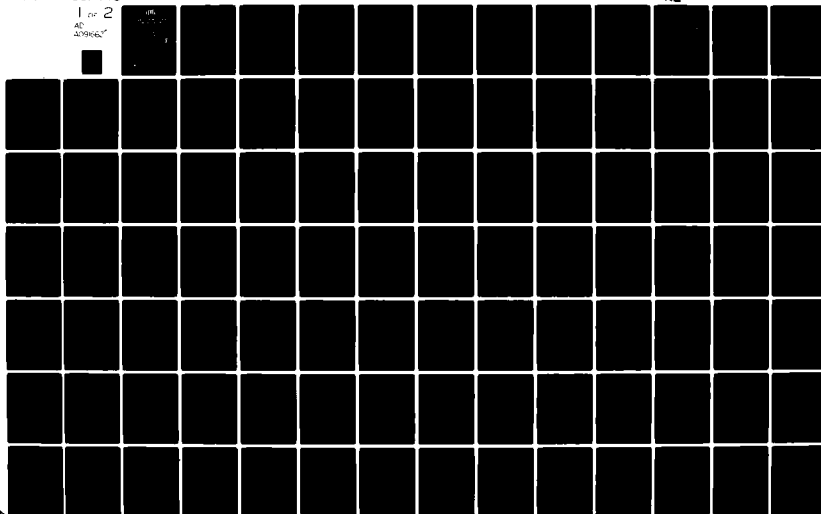
AD-A091 662

GTE PRODUCTS CORP NEEDHAM HEIGHTS MA COMMUNICATION S--ETC F/6 5/8
SPEECH OPTIMIZATION AT 9600 BITS/SECOND. VOLUME 1. SOFTWARE SIM--ETC(U)
SEP 80 A J GOLDBERG, L COSELL, S KWON DCA100-78-C-0064

UNCLASSIFIED

1 of 2
AD
A091662

NL





SOFTWARE SIMULATION

SDTIC
ELECT
NOV 18 1980

**SUBMITTED TO
DEFENSE COMMUNICATIONS AGENCY**

SEPTEMBER 30, 1980

Sylvania Systems Group
Communication Systems Division
GTE Products Corporation
77 A Street
Needham Heights, Mass. 02194 U.S.A.
Area Code 617 449-2000
TELEX: 92-2497

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

8011 17 159

12) 1-1-1

9) Final Report, Oct 78-Sep 80, 1

6) Speech Optimization at 9600
bits/second

15) DCA 100-78-C-0064

Volume 1
Software Simulation

11) 1-1-1

10) A.J./Goldberg L./Cose II S./Kwon L./Bergeron
R./Cheung

Submitted to
Defense Communications Agency

September 30, 1980

406155

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
	AD-A097-1062	
4. TITLE (and Subtitle)	5. TYPE OF REPORT & PERIOD COVERED	
9600 BPS Speech Optimization Study Volume 1	Final Report October 1978 - September 1980	
6. AUTHOR(s)	7. PERFORMING ORG. REPORT NUMBER	
A. J. Goldberg, L. Cosell, S. Kwon, L. Bergeron, and R. Cheung	DCA 100-78-C-0064	
8. PERFORMING ORGANIZATION NAME AND ADDRESS	9. CONTRACT OR GRANT NUMBER(s)	
GTE Sylvania 77 "A" Street Needham Heights, MA 02194		
10. CONTROLLING OFFICE NAME AND ADDRESS	11. REPORT DATE	
Defense Communications Agency Contract Management Division, Code 260 Washington, D.C. 20305	September 30, 1980	
12. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	13. NUMBER OF PAGES	
	Volume 1 - 144 pages	
	14. SECURITY CLASS. (of this report)	
	Unclassified	
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE	
16. DISTRIBUTION STATEMENT (of this Report)		
Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
speech coding, 9600 bps speech transmission, adaptive transform coder, adaptive bit allocation, discrete cosine transform, digital voice terminal, real-time speech coder		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)		
This report describes the design and development of a real-time adaptive transform coder that transmits high-quality speech over a 9600 bps channel with bit-error rates of up to 1% without significant loss of speech fidelity. The report presents the results of our FORTRAN simulations on the adaptive transform coder which maximized the quality of the transmitted speech. Important aspects of the ATC algorithm which are optimized (cont'd)		

DD FORM 1473 EDITION OF 1 NOV 68 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

next page

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (cont'd)

were specification and transmission of the side-band information, accuracy of the pitch and voicing decisions, and error-protection of the important transmission parameters. Also included is the system design, detailed documentation, and program listings of the MAP-300 real-time implementation of the optimized ATC speech coder. Finally, the report includes a description of analog equipment GTE built to interface the MAP-300 to telephone handsets and tape recorders and a description of digital circuits (RS 423 compatible) to interface the MAP-300 to a modem.

This report is bound in two volumes. Volume I contains a description of the ATC system and the results of the FORTRAN simulations. Volume II contains all the information on the real-time system including documentation for implementing the ATC system on the MAP, listing of the MAP software, and documentation for the hardware built by GTE.

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

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Table of Contents
for
Volume 1
Software Simulation

<u>Section</u>		<u>Page</u>
	LIST OF ILLUSTRATIONS	ii
	LIST OF TABLES	iii
1	SUMMARY OF PROGRAM	1-1
	1.1 Introduction	1-1
2	SIMULATION OF THE ATC ALGORITHM	2-1
	2.1 Introduction	2-1
	2.2 Basic Principles of ATC Operation	2-1
	2.3 Optimization and Modification of the ATC System	2-11
	2.3.1 Description of the Modified ATC System	2-14
	2.3.2 Basis Spectrum of the ATC System	2-15
	2.3.3 Bit Assignments Rule of the ATC System	2-23
	2.3.4 Quantization of Sideband and Mainband Information	2-24
	2.3.5 Reducing the Effects of Lowpass Filtering	2-27
	2.3.6 Bit Allocations to Sideband and Mainband	2-29
	2.3.7 Reducing Discontinuities at the Frame Boundary	2-31
	2.4 ATC System in the Presence of Random Channel Errors	2-34
	2.4.1 The Effects of Random Channel Errors on the Performance of the ATC System at 9.6 Kbps	2-25
	2.4.2 Tradeoff Analysis Between Data Rate and Channel Error Rate	2-25
	2.4.3 Application of BCH Code	2-37
	2.4.4 Selection and Protection of the Important Bits in the ATC System	2-42
	2.4.5 Summary	2-52
	2.5 FORTRAN Programs for the Simulation of the ATC System	2-54
	2.5.1 FORTRAN Program of the ATC Algorithm	2-54
	2.5.2 Task Building of the ATC Program	2-58
	2.6 Summary and Conclusions	2-62
APPENDICES		
A	Primitive BCH Codes	2-67
B	Operations in Galois Field	2-78
C	FORTTRAN Source Listings for the ATC Simulation	2-81

LIST OF ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
1.1-1	Speech Processing System Comprised of MAP Hardware and Speech Processor Interface	1-3
2.2-1	Discrete Cosine Transform Operation	2-3
2.2-2	Adaptive Transform Coder	2-5
2.2-3	Graphical Description of Vocoder Strategy for ATC	2-7
	(Cont.)	2-8
2.3-1	Block Diagram of the ATC Analyzer	2-13
2.3-2	Block Diagram of the ATC Synthesizer	2-14
2.3-3	Estimation of the Basis Function	2-16
2.3-4	A Simple Search Routine of Pitch	2-19
2.3.5	Generation of the Pitch Weighting Function	2-21
2.3-6	Probability Density Function of Discrete Cosine Transform Coefficient	2-26
2.3-7	DCT, Basis Spectrum, and Bits Assigned in ATC Coder	2-28
2.3-8	The Performance of an ATC System	2-32
2.4-1	The Performance of an ATC System Under Noisy Channel	2-36
2.4-2	ATC Performance Vs. Data Rate	2-38
2.4-3	Performance of the ATC System with Various Channel Impairments Upon Digital Data Stream	2-43
2.4-4	Bits Assignments of DCT Coefficients in Descending Order	2-45
2.4-5	Performance of the ATC System Vs. Number of Blocks Protected	2-50
2.4-6	Performance of the ATC System Vs. Channel Error Rates	2-51
2.5-1	Flow Diagram of the ATC FORTRAN Program	2-55
2.5-2	Example Operation of the ATC Program	2-60
2.5-3	Example Operation of the ATC Program	2-61
A1	Computation of σ_1 , σ_2 , σ_3	2-74
A2	Chien's Search Decoding Procedure	2-76

LIST OF TABLES

<u>Table</u>		<u>Page</u>
1.1-1	Optimized ATC System Specification	1-4
2.3-1	Bit Allocations to the Sideband Information	2-30
2.4-1	Performance of ATC System Under Various Channel Conditions	2-46
2.4-2	Performance of the ATC System with Various Channel Conditions	2-49
2.6-1	Optimized ATC System Specification	2-63
A	Generator Polynomials for Selected Primitive BCH Codes	2-69

Chapter I

Summary of Program

1.1 Introduction

Under the 9600 BPS Speech Optimization Study, GTE Sylvania simulated and implemented a full duplex Adaptive Transform Coder (ATC) speech digitization algorithm. The simulations were performed using FORTRAN computer programs while the implementation used a CSP, Inc. Map -300 floating point array processor with digital and audio input/output circuitry designed by GTE.

This study and implementation effort has resulted in a number of significant accomplishments in developing speech digitization algorithms. The most important of these include:

- a. The demonstration via FORTRAN simulation that ATC at 9600 bps can produce good quality speech having a Signal-to-Noise ratio (S/N) of about 17 dB.
- b. Establishment of a benchmark speech processing technique at 9600 b/s which indicates that high quality speech is possible at this data rate.
- c. The development of error coding techniques which will permit ATC to function at a bit error rate (BER) of 10^{-2} with little reduction in S/N using (63,45) BCH codes.
- d. The design and implementation of analog audio circuitry to permit speech to be input to and output from the CSP, Inc. MAP processor from microphones and tape recorders.
- e. The design and implementation of a digital transmission interface (RS 423 compatible) to the CSP, Inc. MAP processor so that data can be sent to a modem.

- f. Implementation of a real-time full duplex ATC speech digitizer on the CSP, Inc. MAP processor whose block diagram is shown in Figure 1.1-1. This digitizer performs its processing with floating point arithmetic and, does not compromise numerical accuracy.
- g. Real-time demonstration of ATC in the presence of 10^{-2} channel error rate without significant performance degradation.

The voice quality produced by the ATC simulation is the best of any technique operating at 9600 b/s now known to GTE. The technique, whose specifications are shown in Table 1.2-1, is numerically complex requiring the complete processing capability of the CSP, Inc. MAP-300 floating point processor. Thus, for ATC to be practical, either higher speed hardware must be built or the technique must be simplified.

The investigation and developments leading to the real-time ATC system proceeded in three phases. During the first phase the ATC algorithm originally proposed by Zelinski and Noll^{1,2} was investigated and the modifications proposed by Crochiere and Tribolet^{3,4} were incorporated to improve voice quality. Numerous FORTRAN simulations were conducted to optimize performance with respect to data rate, channel error performance and robustness to speaker and room noise. At the end of the first phase which lasted about 4 months the ATC algorithm was frozen and the real-time implementation begun.

Concurrent with the first phase was the design and fabrication of the digital and analog I/O interfaces to the CSP, Inc. MAP-300 processor. In addition to building our own units, GTE Sylvania, under separate subcontracts with BBN and Notre Dame University, built two additional units for incorporation with their MAP-300 speech processing systems.

SPEECH PROCESSING SYSTEM COMPRISED OF MAP HARDWARE AND SPEECH PROCESSOR INTERFACE

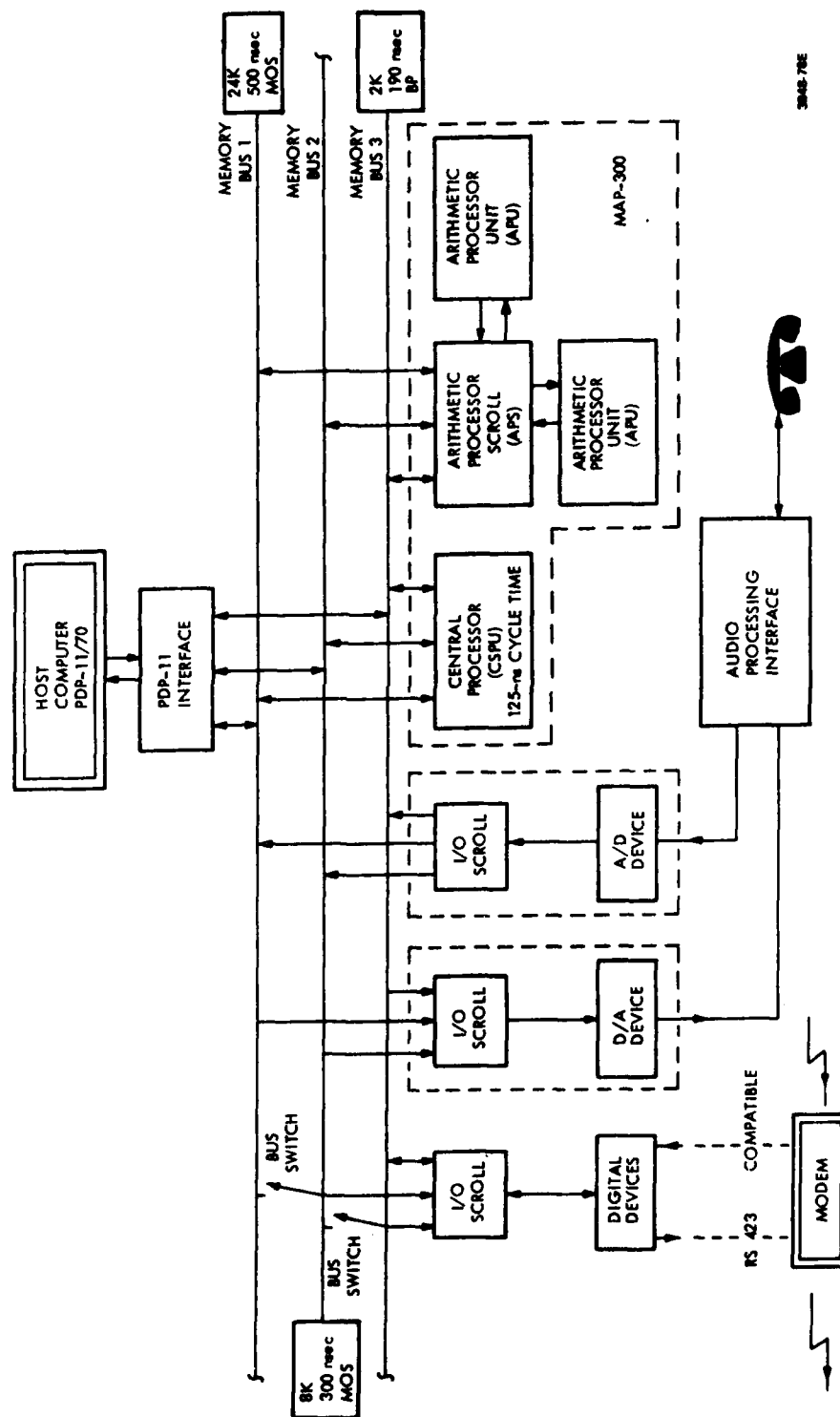


FIGURE 1.1-1

<u>PARAMETER</u>	<u>SPECIFICATION</u>
Input Bandwidth	0-3200 Hz
Sampling Rate	6400 Hz
Frame Rate	26.016/sec.
Number of Samples/Frame	246
Number of Samples Overlapped/Frame	10
Bits/Frame	369
Pitch	{ 6 if voiced 0 if unvoiced
Pitch Gain	{ 2 if voiced 0 if unvoiced
Voiced/Unvoiced	1
RMS Energy	5
DC BIAS	5
PARCOR 1	5
PARCOR 2	5
PARCOR 3	4
PARCOR 4	4
PARCOR 5	3
PARCOR 6	3
PARCOR 7	2
PARCOR 8	2
Parity Bits (Error Correction)	54
SYNC	1
DCT Coefficients	{ 267 voiced 275 unvoiced
Number of Error Control Blocks/Frame	3
Error Control Technique	(63,45) BCH

TABLE 1.1-1: OPTIMIZED ATC SILENCE SPECIFICATION

The third phase, the real-time implementation, began in February 1979 and continued until August 1980. During this time, test programs for the analog and digital I/O were developed and numerous software and hardware problems with the MAP-300 were resolved. Finally, in the summer of 1979, the first working modules of the ATC digitizer were operational on the MAP-300, and it was at this time that the scope of the software development project became apparent. The MAP-300, for all its speed was barely adequate to perform ATC with error control in a full duplex mode. Consequently, from August 1979 to the delivery of the ATC system a year later, considerable effort was placed on writing efficient MAP-300 real-time software.

The final ATC system, as delivered to DCA, indicates that a full duplex ATC speech processing system can operate on the MAP-300 processor in real-time.

Future speech digitization development at 9600 cannot ignore the ATC algorithm because even though the technique is complex, it shows that good quality speech is possible at this data rate. Thus, the ATC technique developed under this contract will serve as a benchmark or standard to compare all new 9600 b/s speech digitization algorithms.

This report is written in two volumes. Volume 1 contains documentation on the ATC simulations while Volume 2 contains documentation on the real-time software and hardware I/O circuitry.

Chapter 2

Simulation of the ATC Algorithm

2.1 Introduction

Adaptive Transform Coding (ATC) was originally proposed by Zelinsky and Noll^{1,2} and represents an efficient block-coding technique for speech digitization in the 8.0 to 16 K b/s range. Early simulations of the ATC algorithm at GTE Sylvania indicated that this technique was capable of producing better speech quality than any other technique at 9.6 kbps known to the company at the time. When DCA requested the study of new techniques at 9.6 kbps GTE Sylvania responded with the ATC algorithms as originally proposed by Zelinsky and Noll. Later articles by Tribolet and Crochiere^{3, 4} however, indicated that further improvements were possible in the algorithm, and after contract award, GTE decided, based on simulations conducted under its IR&D program, to develop this algorithm even though it was about 50% more complex than the original Zelinsky and Noll design.

In this Chapter we first discuss the theory of ATC operation. Then we discuss the simulation and optimization of this system and the need for error protection and correction for some critical transmission parameters. Finally we discuss the results of the simulation with and without error protective coding in the presence of channel errors as high as one error in 100 bits. (A BER of 10^{-2} .)

2.2 Basic Principles of ATC Operation

In its basic form, ATC consists of sending the largest cosine transform coefficients of a segment of data with each coefficient quantized according to an algorithm that gives the larger coefficients more bits than the smaller

coefficients. This ATC algorithm departs from earlier algorithms that not only had to send the amplitudes of the coefficients, but also had to send considerable information about which coefficients were quantized and how many bits were associated with each. This extra information could consume as much data capacity as the coefficient amplitudes themselves. Attempts at sending only specific coefficients or the use of a fixed-bit assignment generally reduced voice quality by creating waveform discontinuities at the frame boundaries and by spectrally distorting the signal between boundaries.

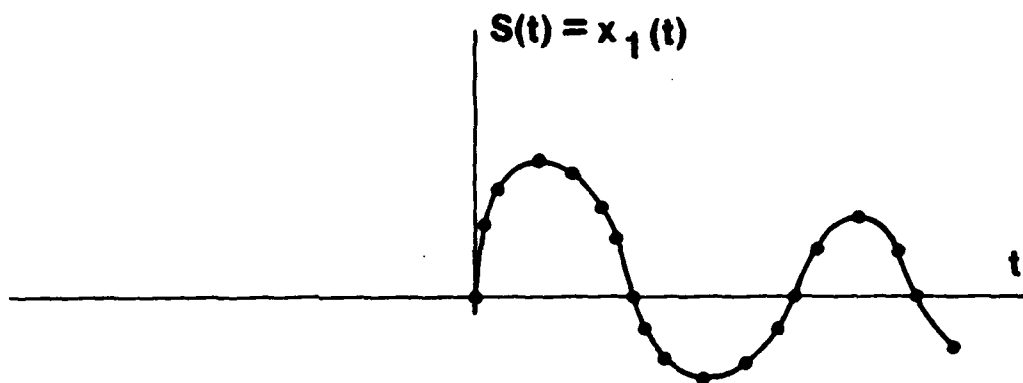
In ATC, however, information about which amplitude is sent and how many bits are allocated to each is contained in the basis spectrum, which requires from 1200 to 2400 b/s. This basis spectrum generally is information about the envelope of the transform coefficients being quantized. Its calculation can be performed by the smoothing of transform coefficients or by separate estimates involving least-square analysis.

To understand ATC, consider a sampled waveform segment shown in Figure 2.3-1(a). If this waveform is multiplied by $1/2$, delayed by half the sampling interval T , and reflected about $t=0$, it yields $x_2(t)$ whose Fourier transform is given by:

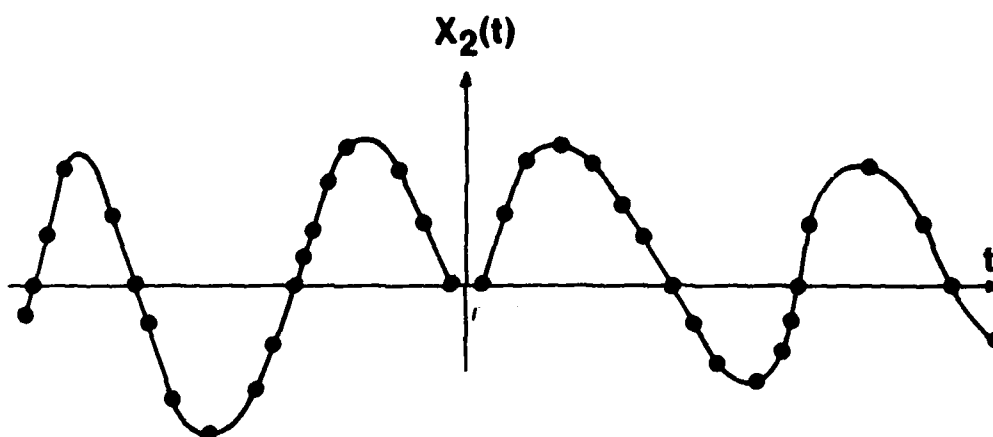
$$X_2(f) = \sum_{n=-(N-1)}^{N-1} x_2(nT) \exp(-j2\pi f(n+1/2)T) \quad (2.2-1)$$

If we sample the Fourier transform of $X_2(f)$ at frequencies $\frac{\pi m}{2NT}$, the discrete Fourier transform (DFT) becomes

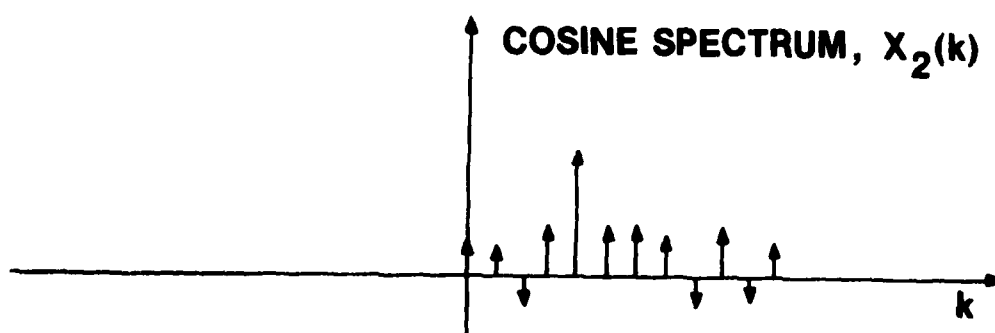
$$X_2\left(\frac{\pi m}{2NT}\right) = X_2(m) = \sum_{n=-(N-1)}^{N-1} x_2(nT) \exp(-j\frac{\pi m}{N}(n+1/2)) \quad (2.2-2)$$



(a) Original Waveform



(b) Reflected Waveform



6310-78E

(c) DFT Output

Figure 2.2-1: Discrete Cosine Transform Operation

Using symmetry properties of $X_2(nT)$, $X_2(m)$ shown in Figure 2.2-1(b) is real only and is given by

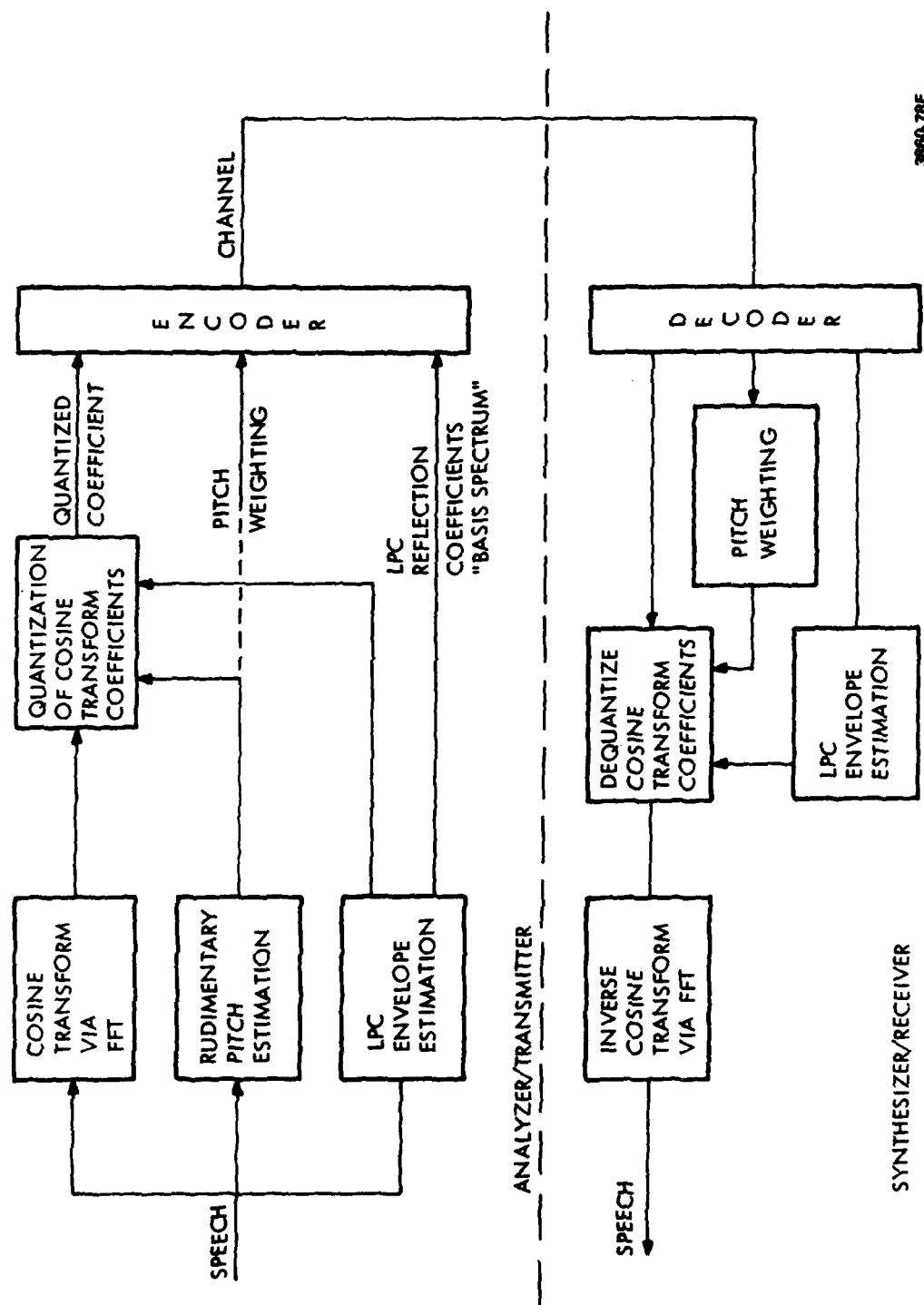
$$X_2(m) = \sum_{n=0}^{N-1} X_1(nT) \cos\left(\frac{\pi m}{2N} (2n+1)\right) \quad 0 \leq m \leq N-1 \quad (2.2-3)$$

Equation (2.2-3) is the cosine transform. This derivation shows that the Fast Fourier Transform (FFT) can be used to implement the cosine transform by delaying and reflecting the original waveform and then taking the FFT on a waveform twice as long as the original.

The most expensive implementation costs with the ATC algorithm are associated with the Discrete Cosine Transform (DCT) and Discrete Fourier Transform (DFT). Although the DCT cannot be employed directly, methods elaborated by Ahmed et al⁶ and Cooley et al⁷ use the DFT to compute the desired transform. Our FORTRAN simulations used the Cooley method for DCT calculation and a special FFT algorithm to lower simulation costs.

After calculation of the DCT coefficients, the basis spectrum (envelope of the cosine transform) can be estimated by making all the cosine transform coefficients positive and smoothing between peaks to efficiently send the envelope. We can quantize the amplitudes of every m th (m is typically 8) envelope sample and send those as the coefficients of the basis spectrum.

However, this original ATC algorithm, as proposed by Zelinski and Noll, suffers from a "burbling" characteristic at lower data rates. To reduce this distortion, Tribolet uses side transmission of pitch and spectral parameters obtained by Linear Predictive Coding (LPC)⁵ analysis. The side transmission of the LPC and pitch parameters does in fact remove the "burbling" sound and improve the overall signal-to-noise ratio. Figure 2.2-2 describes



3860-78E

Figure 2.2-2: Adaptive Transform Coder

the operation of this ATC digitizer.

The innovative solution to the basis spectrum calculation is formed from a least-square analysis of $x_2(t)$, that is, finding those predictor coefficients which minimize.

$$E = \sum_{n=0}^{N-1} \left[x_2(nT) - \sum_{i=1}^P a_i x_2[(n-i)T] \right]^2 \quad (2.2-4)$$

These predictor coefficients, or alternately reflection coefficients, carry information about the envelope since:

$$Y(f) = \text{FFT}(a_i) \quad (2.2-5)$$

and the envelope is then $Y^{-1}(f)$.

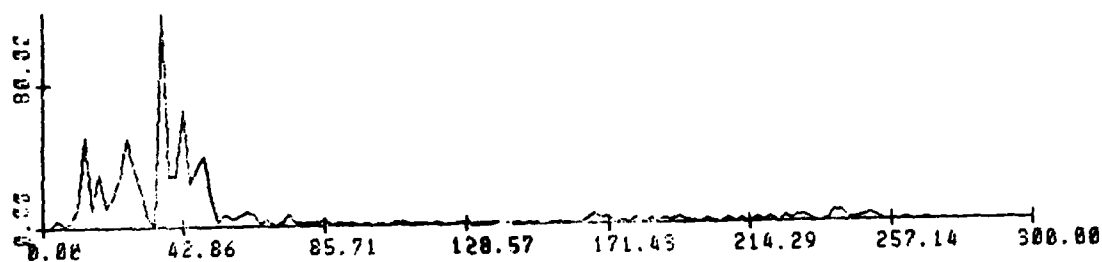
In addition to linear predictive modeling of the ATC spectrum, the Trib-olet approach uses a pitch excitation source. This accounts for the fine structure in the short-time spectrum, which is consistent with the known mechanisms of speech production. This scheme forces the assignment of transform bits to many pitch striations that otherwise would not be transmitted at all.

With reference to Figure 2.2-3, the ATC analysis is described as follows:

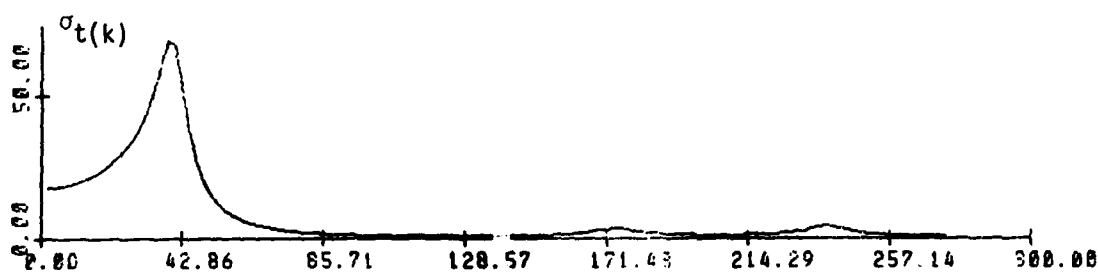
1. The input speech (Figure 2.2-3(a)) is Fourier transformed to yield a DCT spectrum (Figure 2.2-3(b)). This spectrum is squared, windowed, and inverse Fourier transformed to yield an autocorrelation function (i.e., pseudo-ACF) of the reflected speech waveform. The first $P+1$ values of this function are used to define a correlation matrix in the usual normal equation formulation sense. The solution of these equations (i.e., Levinson recursion) yields a prediction filter of



(a) Input Speech Samples

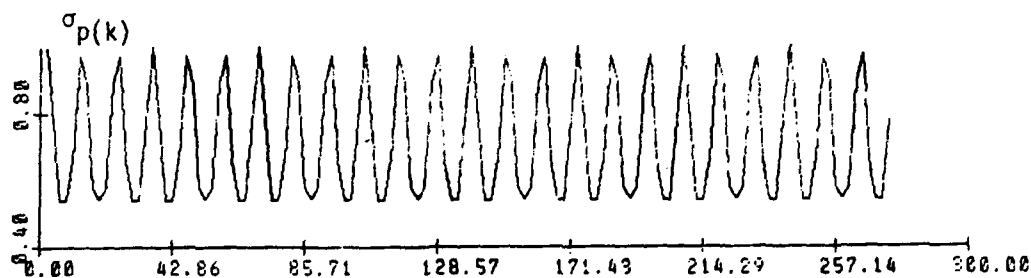


(b) DCT of Input Speech($\times 10^1$)

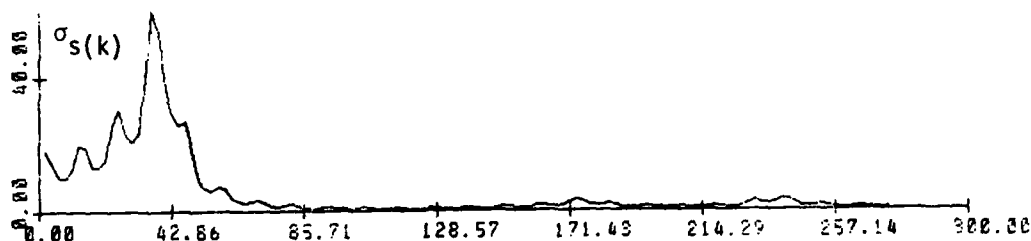


(c) LPC Spectrum($\times 10^1$)

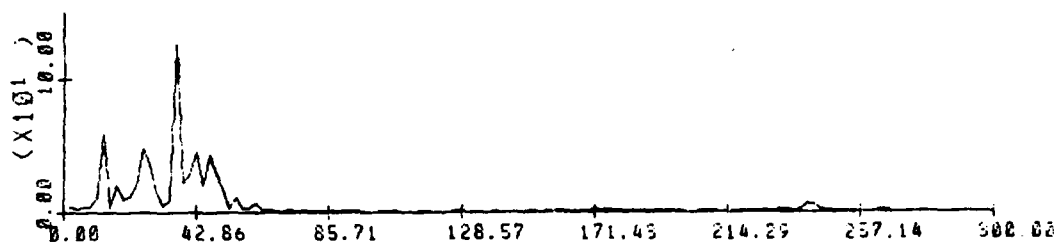
Figure 2.2-3: Graphical Description of Vocoder Strategy for ATC



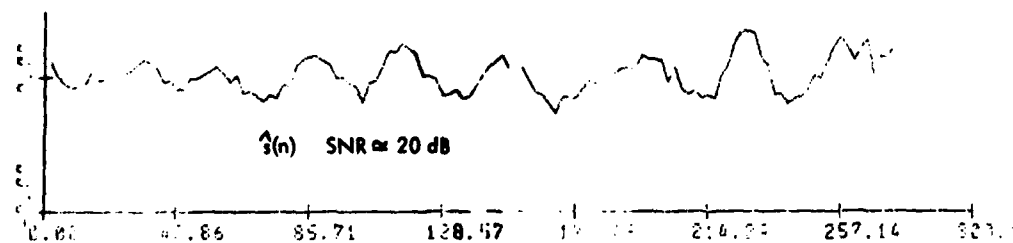
(d) Pitch-Weighting Spectrum($\times 10^1$)



(e) Basis Spectrum($\times 10^1$)



(f) Quantized DCT($\times 10^1$)



(g) Error Waveform-Original-Processed($\times 10^1$)

000-79E

Figure 2.2-3: Graphical Description of Vocoder Strategy for ATC (Cont.)

order P. The inverse spectrum of this filter yields a smoothed estimate of the DCT (Figure 2.2-3(c)) spectrum levels to be used in the adaptation of the quantizers.

2. A rudimentary estimate of the pitch value, M, is found in the pseudo-ACF after the second zero crossing beyond the P+1 ACF value. A corresponding gain factor, G, is also computed as the ratio of ACF(M)/ACF(0). With these two parameters, a pitch pattern is generated in the frequency domain (Figure 2.2-3(d)) and applied congruently with the LPC spectrum. This combination, yielding a linear prediction spectral fit to the DCT of the input speech, is called the basis spectrum (Figure 2.2-3(e)).
3. The computation to determine the number of bits to allocate for each transform then proceeds as follows:

Let σ_i be the amplitude of the i th term of the envelope of the basis spectrum. The B_i , the number of bits allocated to the i th cosine transform coefficient, is given by:

$$B_i = \left[B_f/N - (1/2N) \sum_{j=1}^N \log_2 \sigma_j^2 \right] + 1/2 \log_2 \sigma_i^2 \quad (2.2-6)$$

where

B_f = the total number of bits allocated to send the cosine transform coefficients per frame

N = the total number of cosine transform coefficients calculated per frame.

Note that the term in brackets is calculated once per frame. Fairly simple algorithms ensure that B_i is an integer value and that the sum of the integer B_i adds to B_f .

The cosine transform coefficients approximate a Gaussian probability density function. Optimum Gaussian quantizers derived by Max²⁰ can be used to encode each transform coefficient with B_i bits. Since many of the B_i 's will be zero, only larger coefficients are sent. However, GTE Sylvania's experience with the 9600-b/s ATC has shown that optimal quantizers can be developed that more closely match the transform distribution.

4. The receiver uses the basis spectrum information (LPC, M, G) to regenerate the DCT envelope, to generate the bit allocation using Equation (2.2-6), to decode the cosine transform coefficients (Figure 2.2-3(f)), and then to take the inverse cosine transform using the FFT. Frame boundary problems exist at all data rates since quantization of the transform coefficients causes the regenerated waveform to be slightly different than the original. By overlapping the frames slightly and by interpolating across the frame boundaries, these discontinuities can be smoothed.

The overall quality of this approach can be estimated from Figure 2.2-3(g), which shows the error waveform defined as:

$$e(n) = s(n) - \hat{s}(n) \quad (2.2-7)$$

The received waveform, $\hat{s}(n)$, has a high signal-to-noise ratio (~ 17 dB) for some speakers, even for erroneous pitch estimations made in the analyzer. In fact, GTE Sylvania has demonstrated through audio tapes that an eighth-order LPC predictor ($P = 8$), coupled with the rudimentary pitch extractor (and no voiced/unvoiced logic), yields consistently high-quality speech.

2.3 Optimization and Modification of the ATC System

The adaptive transform coding scheme shown in Figure 2.2-2 produces high quality synthesized speech above 9600 bps. In this scheme, the quality in objective signal-to-noise ratio and in subjective perceptual effects degrades by lowering the transmission data rate.

There are several sources which reduce the voice quality but solutions for most of these problems exist. The first one is the quantization noise caused by coarse quantization of the DCT coefficients at low data rates. This problem can be minimized by developing optimal quantizers from the distribution of actual DCT coefficients. The second and most severe degradation source is the reduction in bandwidth at low data rates. This effective lowpass filtering stems from the fact that only large DCT coefficients are being coded because there are not sufficient bits to send the smaller coefficients. The effects of lowpass filtering can be removed by the addition of random noise to the low energy frequency band. The third source of degradation are waveform discontinuities at the frame boundaries since the DCT coefficients are coarsely quantized and some low valued coefficients are not transmitted. These effects may be reduced by overlapping the frames slightly and by interpolating across the frame boundaries.

There are other areas for improvement in the ATC scheme. The first one is the trade-off of bits allocated to DCT coefficients and to side information within given transmission data rate. Another is the method for quantizing the side information. It can be shown that closer estimation (in the mean square error sense) of the basis function to actual DCT coefficients provides better performance of the ATC coder. In fact, the extreme case, where the basis spectrum equals the actual spectrum,

quantization of the DCT can be precise, eliminating lowpass effects or boundary problems as long as the sign bits of DCT coefficients are provided because DCT is a unitary transform.

In the following sections, the modified ATC system will be described. Also, those areas requiring further developments will be described and the possible improvements will be discussed.

2.3.1 Description of the Modified ATC System

The block diagram of the ATC analyzer and synthesizer are shown in Figure 2.3-1 and Figure 2.3-2, respectively. In this scheme, the input speech is buffered into blocks of data $\{v(n)\}$ which consist of a frame. This frame of speech data is overlapped slightly (about 10 samples) in order to reduce the frame boundary problems. The mean and variance of the input speech signal are calculated for the transformation of zero mean and unit variance. This mean and variance are quantized and sent to the receiver for the renormalization of the synthesized speech. The Discrete Cosine Transform is calculated on the zero mean and unit variance input. The DCT coefficients are then adaptively quantized to form mainband information and transmitted to the receiver. At the receiver, they are decoded and inverse transformed to reproduce the zero mean and unit variance speech signal. This signal is renormalized by the mean and variance to produce the synthesized speech.

In the meantime, the mean and variance of the signal are decoded and dequantized to renormalize the inverse transformed signal. In order to reduce the effects of signal discontinuities at the frame boundaries, the overlapped signals are interpolated.

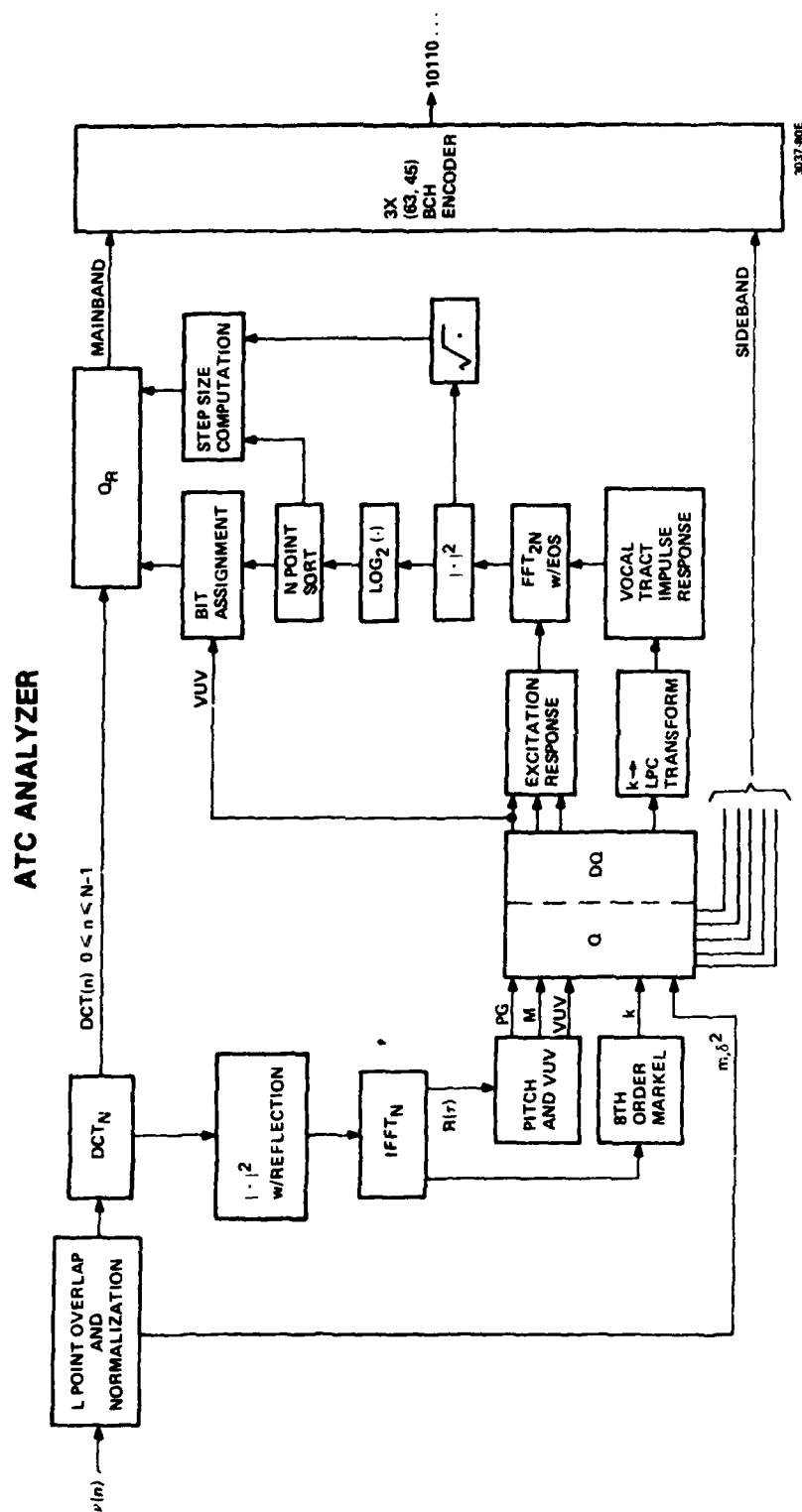


FIGURE 2.3-1 BLOCK DIAGRAM OF THE ATC ANALYZER

ATC SYNTHESIZER

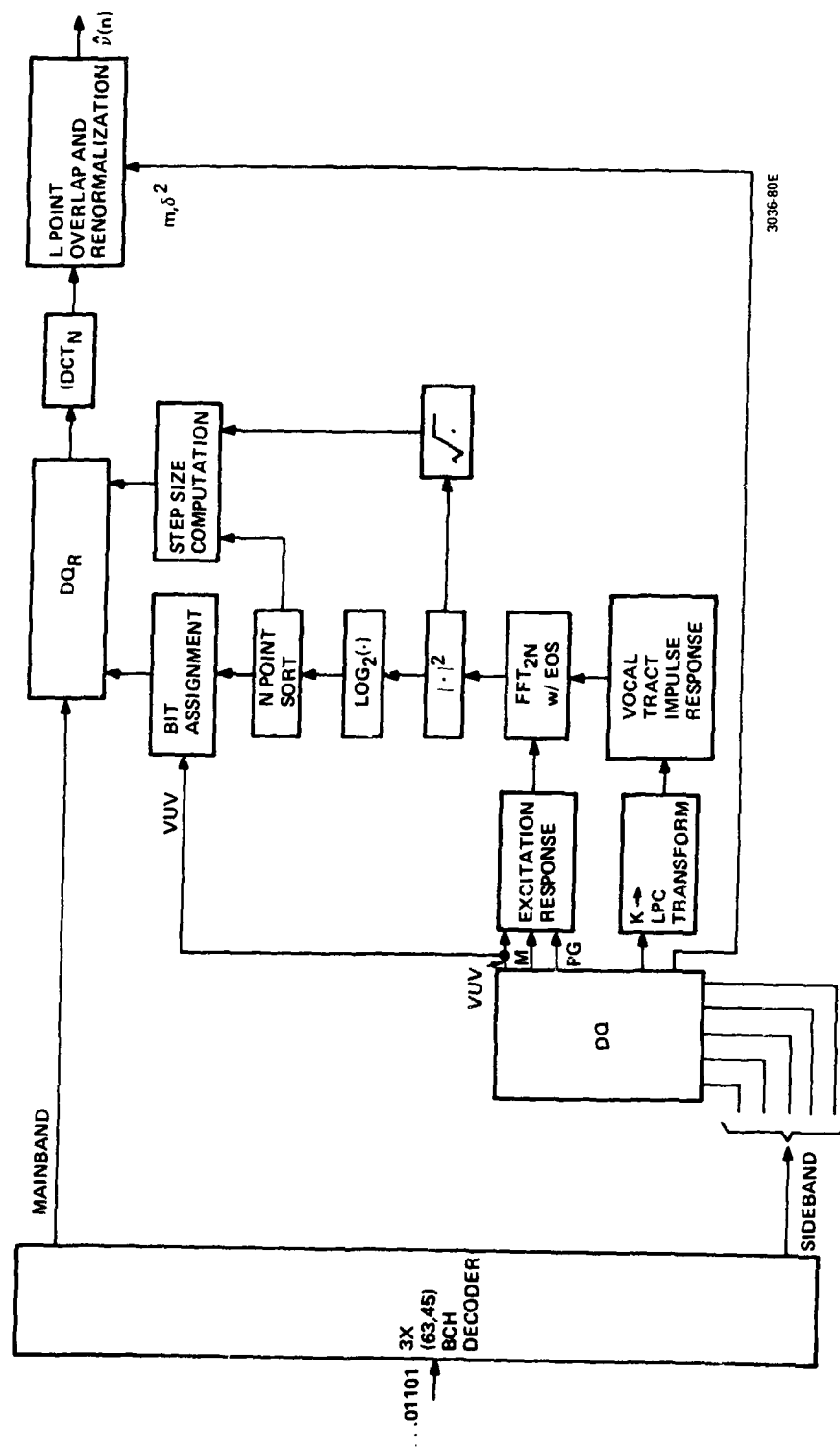


FIGURE 2.3-2 BLOCK DIAGRAM OF THE ATC SYNTHESIZER

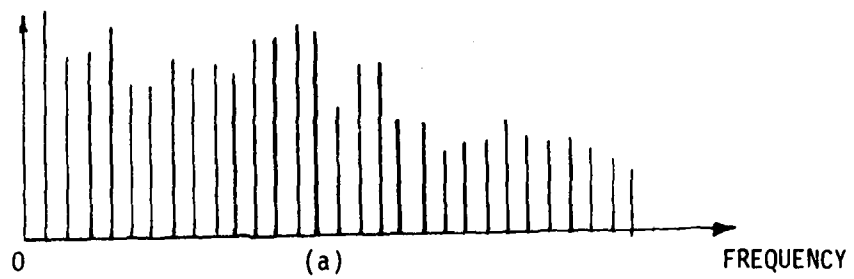
The adaptive quantizations and dequantizations of this scheme are based on the sideband information which the basis spectrum will be computed from. The bit assignments and step size computation will be determined by the optimum bit assignments rule from the basis spectrum. The sideband information includes the pitch gain (PG), pitch number (M), voiced/unvoiced decision, and the 8 PARCOR coefficients. The mean and variance of the input speech signals are also included in the sideband information. The data of the sideband and mainband are encoded by the three block of a (63,45) BCH code in order to reduce the effects of the channel errors. The channel errors occurring during the transmission through the noisy channel will be corrected by the decoder. The information of the mainband is fed to the synthesizer to reproduce the speech signal.

2.3.2 Basis Spectrum of the ATC System

The performance of the ATC system is heavily dependent on the generation of the basis spectrum from which the adaptive quantization and dequantization rule is derived. Two basic adaptation techniques have been proposed. The first technique, proposed by Zelinski and Noll^{1,2} is described in Figure 2.3-3.

After the calculation of DCT coefficients, the basis spectrum is estimated by making DCT coefficients positive and averaging between peaks to compress the DCT envelope. The amplitudes of the every m th (m is typically 8 to 16) sample of the envelope are quantized and sent to the receiver to represent the spectral levels at specified frequencies. These amplitudes are then geometrically interpolated (i.e., linearly interpolated in log amplitude) to form the basis spectrum. This simple, "non-speech

DCT SPECTRUM



SIDE INFORMATION



INTERPOLATED BASIS FUNCTION

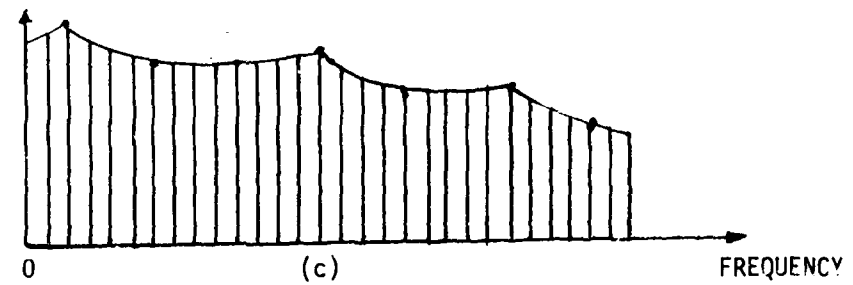


FIGURE 2.3-3 ESTIMATION OF THE BASIS FUNCTION

- (a) Actual squared amplitudes of the DCT coefficients
- (b) Averaged samples
- (c) Estimated basis spectrum obtained by interpolation

specific" algorithm is quite appropriate for speech transmission above 9.6 Kbps. However, the synthesized signal is degraded by a very perceptible "burbling" distortion as the data rate decreases below 9.6 Kbps.

Zelinski and Noll² suggested incorporating a form of voice-excited "fill-in" procedure similar to that used in voice excited vocoder technique. In their technique, low energy frequency bands, which receive no bits for encoding at the transmitter, are filled-in at the receiver with random noise in order to enhance the perceived speech quality. Some improvements have been reported, but the addition of random noise introduces some hoarseness to the synthesized speech. They adjust the amount of added random noise to optimize the speech quality, i.e., the problems of "bubbling" and "hoarseness" are reduced, but it is not sufficient to overcome the difficulties aforementioned at data rates below 9.6 Kbps. Tribolet and Crochiere³ proposed a more appropriate algorithm for bit rates below 9.6 Kb/s which is a "speech specific," adaptation algorithm, and takes full advantage of the known models and dynamics of the speech production mechanism in order to predict the DCT spectral levels. This algorithm is based on an all pole model of the formant structure of speech and a pitch model to represent the fine structure (pitch striations) in the speech spectrum^{13, 14}. The resulting algorithm is referred to as a "vocoder-driven" adaptation strategy due to the close relationship of this spectral estimate to a vocoder model.

The block diagram in the Figure 2.3-1 illustrates the implementation of the technique. First the DCT spectrum is squared and inverse transformed with an inverse DFT. This yields an autocorrelation-like function, the pseudo-ACF (Auto-Correlation Function). The first $P + 1$ values of this function are used to define a correlation matrix in the usual normal

equations formulation sense ¹³ . The solution of these equations yields an LPC filter of order P. The inverse spectrum, illustrated in Figure 2.2-3(c) yields an estimate of the formant structure of the DCT spectrum denoted as $\sigma_t(k)$.

The fine structure of the DCT spectrum is obtained from a pitch model. To obtain the pitch period, M, the pseudo-ACF is searched for a maximum. The pitch estimate taken from the rudimentary procedure suggested by Tribolet and Crochiere ^{3,4} has a definite bearing on the SNR of the processed speech. The use of this imperfect pitch value does not grossly affect the subjective voice quality. However, in order to derive the most impact from the use of a pitch weighting function, the original pitch extraction procedure has been modified. The flowchart of the search routine for pitch period, M, is shown in Figure 2.3-4. It consists of a simple search routine which commences after the appearance of the second zero crossing in the autocorrelation function. The pitch contour which results from this technique is more accurate than the original unconstrained approach with a corresponding increase in the cumulative SNR. This simple scheme has proven to be adequate for the development of the ATC system when the voice/unvoiced decision device is incorporated. The corresponding pitch gain, G, is the ratio of the pseudo-ACF at M over its value at the origin. With these two parameters, a pitch pattern $\sigma_p(k)$ is generated in the frequency domain as illustrated in Figure 2.2-3(d). The two spectral components $\sigma_t(k)$ and $\sigma_p(k)$ are multiplied and normalized to yield the final spectral estimate for $\sigma_s(k)$,

$$\sigma_s(k) = \sigma_t(k)\sigma_p(k) \quad k = 0, 1, 2, \dots, N-1 \quad (2.3-1)$$

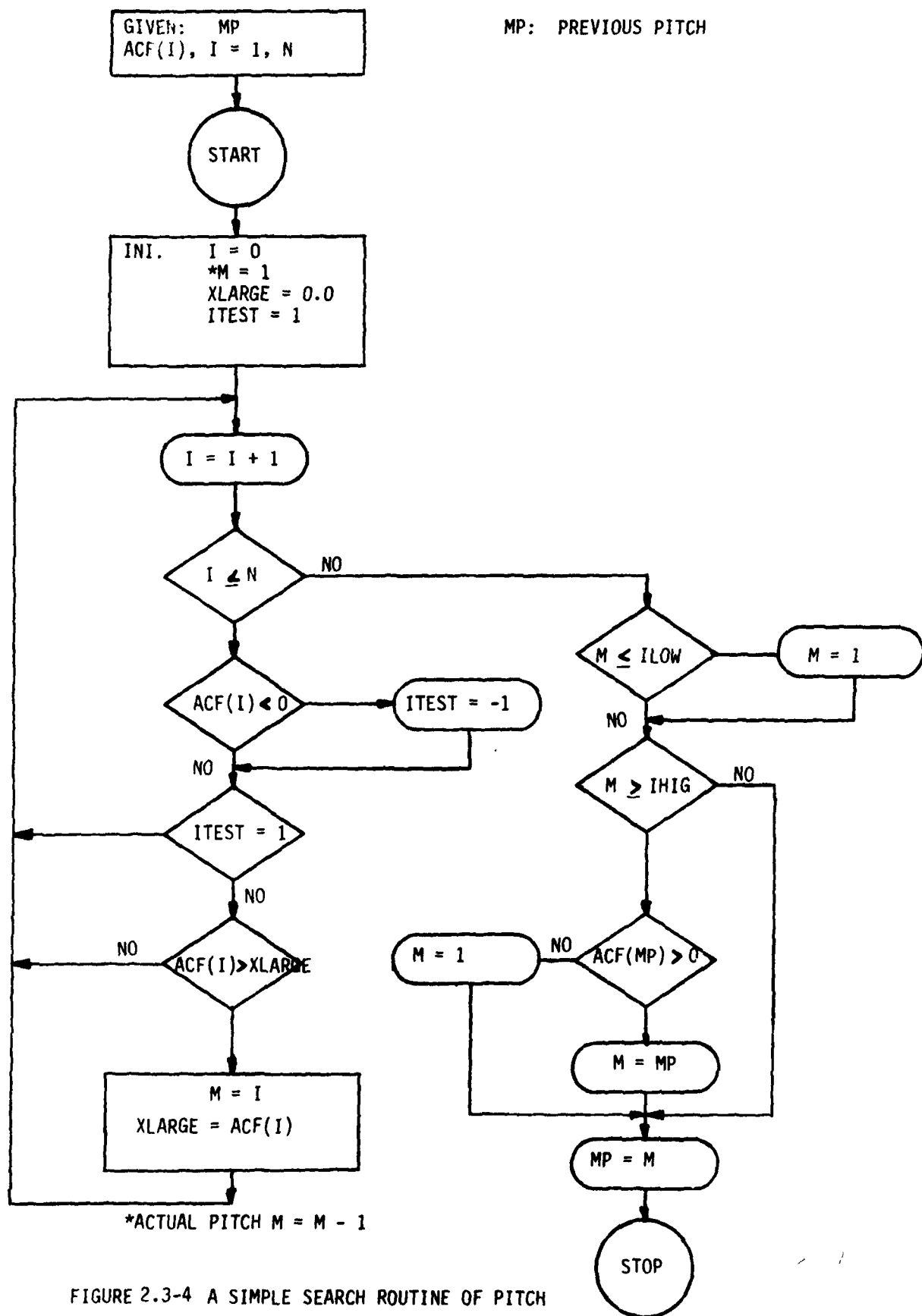


FIGURE 2.3-4 A SIMPLE SEARCH ROUTINE OF PITCH

This estimate, illustrated by Figure 2.2-3(e) is then used for the bit assignment and step-size adaptation algorithms as seen in Figure 2.3-1.

There are many ways of generating pitch weighting function in the frequency domain. In the model of GTE Sylvania, first the pitch gain is defined as

$$G = \text{ACF}(M) / \text{ACF}(0) \quad (2.3-2)$$

and a time domain pitch impulse train with exponentially decaying amplitudes is generated as

$$p(n) = \begin{cases} G^k & , \quad n = kM, \quad k = 0, 1, \dots, K, \quad K = \lfloor N/M \rfloor \\ 0 & \text{otherwise} \end{cases} \quad (2.3-3)$$

where N is the number of speech samples in a frame and $\lfloor \cdot \rfloor$ denotes the largest integer. This time domain signal $p(n)$ is transformed into a zero mean and unit power process $p_1(n)$ which is again transformed into the frequency domain as

$$P_1(k) = \sum_{n=0}^{N-1} p_1(n) e^{-j \frac{2\pi kn}{N}}, \quad k = 0, 1, \dots, N-1 \quad (2.3-4)$$

This periodic pitch weighting function $P_1(k)$, shown in Figure 2.3-5, when multiplied by the LPC spectrum, is adequate for the generation of the basis function in many cases. However, there are cases in which the pitch harmonics are not well preserved in the high frequency band for some voiced sounds, particularly for the fricative voiced sounds (V, Z). There are also many cases where the pitch harmonics of $P_1(k)$ are not

MODIFIED PITCH WEIGHTING FUNCTION FOR VDS-ATC

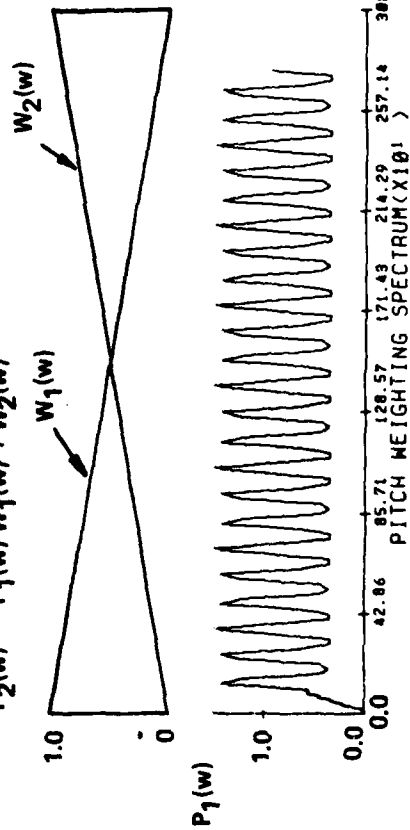
MODIFY THE PRIMARY IMPULSE RESPONSE, $p(n)$, TO YIELD A ZERO VALUED DC COMPONENT AND UNITY POWER

$$p_1(kn) = p(kn) - \sum_{k=0}^K \frac{pG^k}{N}, \quad K = \left\lceil \frac{N}{M} \right\rceil$$

$$P_1(w) = \text{DFT}_{2N} \left\{ p_1(kn) \right\}$$

THEN APPLY COMPLEMENTARY LINEAR WEIGHTING FUNCTIONS, $W_1(w)$ AND $W_2(w)$ SUCH THAT

$$P_2(w) = P_1(w) W_1(w) + W_2(w)$$



3031-80E

FIGURE 2.3-5 GENERATION OF THE PITCH WEIGHTING FUNCTION

matched to the actual spectrum of the input speech, particularly in the high frequency band. This fact can be explained from the errors of the pitch period in time domain. Most gross errors of the pitch period are caused by erroneous decisions of the pitch detection routine. However, a small amount of erroneous pitch period estimates may always exist because of the discrete sampling process in the time domain.

Therefore, we generated a pitch weighting function which is periodic in the low frequency band and close to unit amplitude in high frequency band. One such function can be generated as

$$P_2(k) = P_1(k) W_1(k) + W_2(k) \quad (2.3-5)$$

where the weighting functions $W_1(k)$ and $W_2(k)$ are shown in Figure 2.3-5. The pitch weighting function $P_2(k)$, of equation (2.3-5), when it is multiplied by the LPC spectrum, has proven to be an efficient for the estimation of the basis function.

The basis function of the ATC system will be the LPC spectrum in eq. (2.3-1) with or without multiplication of the pitch weighting function. Experiments have shown that a closer estimation (in the mean square error sense) of the basis function to the actual spectrum makes the ATC system perform better. In fact, in the extreme case, where the basis spectrum equals the DCT spectrum, quantization of the DCT parameter can be precise, eliminating all types of distortion as long as the sign bits of the DCT coefficients are provided. A post V/UV decision is made on the basis of the signal-to-noise ratios in frequency domain with or without multiplication of the pitch weighting function, i.e., if the signal-to-noise ratio of the basis function without multiplication of the pitch weighting function provides higher value than the one with pitch weighting function, it is better to make that frame as unvoiced.

2.3.3 Bit Assignments Rule of the ATC System

It has been shown that the basis function of the ATC system plays an important role on the performance of the ATC system. The choice of bit assignments also determines how accurately the DCT coefficients are encoded. Thus, it controls the distribution of the quantizing noise in the frequency domain. The optimum bit assignments rule (in the minimum mean square error criterion) for a stationary Gaussian-Markov process has been derived in eq. (2.2-6) from the rate distortion theory.^{15,16} It can be shown that the optimum bit assignments rule based on a minimum mean square error leads to a flat noise distribution in the frequency domain. It has been known that a flat noise in frequency domain is not the most desirable perceptual criterion. Tribolet and Crochiere⁴ modified the bit assignment rule of eq. (2.2-6) by multiplying a weighting function $W(k)$ that weights the importance of the noise in different frequency bands. They have suggested the weighting function to be

$$W(k) = \sigma_s^{2\gamma}(k), \quad k = 0, 1, \dots, N-1 \quad (2.3-6)$$

where γ is a parameter that can be experimentally varied from -1 to 0. So the value of γ is slowly varied between these two extremes ($-1 < \gamma < 0$), the noise spectrum will evolve from a flat distribution to the one that precisely follows the speech spectrum. Extensive experiments^{17,18,19} of noise shaping have shown that the noise spectrum which follows the spectrum of the speech in certain ways provides slightly higher subjective speech quality than the one of flat noise spectrum does. The value of $\gamma = -0.125$ was reported to give a good result⁴. However, when the data rate decreases below 8 Kb/s, the effects of the weighting function as well

as the optimum bit assignments rule cannot be described clearly. The performance of the ATC system may be optimized asymptotically at the low data rates by incorporating a simple limiter of the highest bit allocation. By adjusting the largest number of the bit allocation, the spectrums of the noise can be varied. The spectrum of noise will be flat for the large value of the limiter output, and the spectrum of the noise will follow the speech spectrum when the value of the limiter output is small (≈ 1). Experiments have shown that the maximum number of bit assignments of 5 provides a good result at the data rate 9.6 Kb/s.

2.3.4 Quantization of Sideband and Mainband Information

The quantization effects at high transmission data rates do not cause a perceptual loss of performance of the ATC system. However, at low data rates, these quantization effects constitute a major source of degradation of the synthesized speech.

Let P_j^2 be the j th actual spectrum and P_{sj}^2 be the j th spectrum from the side information. Then, the normalized DCT coefficient can be expressed as

$$x_j = P_j / P_{sj} \quad (2.3-7)$$

Let B_j be the number of bits assigned to j th DCT coefficient. Then P_j is a Gaussian distributed random variable if the samples of time domain signals are Gaussian distributed. The distribution of the normalized DCT coefficients is, however, not known and analytical derivation of this distribution function is too complicated to calculate. GTE Sylvania performed simulations to develop the distribution function and the results are shown

in Figure 2.3-6. Note that the distribution function of x_j lies in between the Gaussian and Laplace distribution. GTE Sylvania has written a computer program which calculates the characteristics of an optimum quantizer from the simulated distribution function by following the procedures of Max²⁰.

Let $\hat{x}_j$ be the quantized value of x_j , then the procedure of Max minimizes the mean square error, i.e.,

$$e = E|(\hat{x}_j - x_j)^2| \quad (2.3-8)$$

where E denotes the statistical expectation. GTE Sylvania has determined the distribution of x_j under the given conditions of B_j which is a function of the ATC coder and speech signals. The conditional distribution of x_j is slightly different from the distribution of Figure 2.3-6. GTE Sylvania has developed a computer program which generates an optimum quantizer from the conditional distribution of x_j . These procedures can be applied to develop quantizing tables for every system parameter where the minimum mean square error criterion is an adequate measure.

In the present ATC scheme, the LPC technique is used to calculate the basis spectrum with transmission requiring quantization of the PARCOR coefficients. In this case, the error criterion of eq. (2.3-8) is modified as

$$E = E|s(x_j) (\hat{x}_j - x_j)^2| \quad (2.3-9)$$

where $s()$ is the weight function derived from the sensitivity analysis of power spectrum with respect to PARCOR coefficient variation²¹. A large data base was used to accumulate the statistical information needed for optimal quantization of the PARCOR parameters.

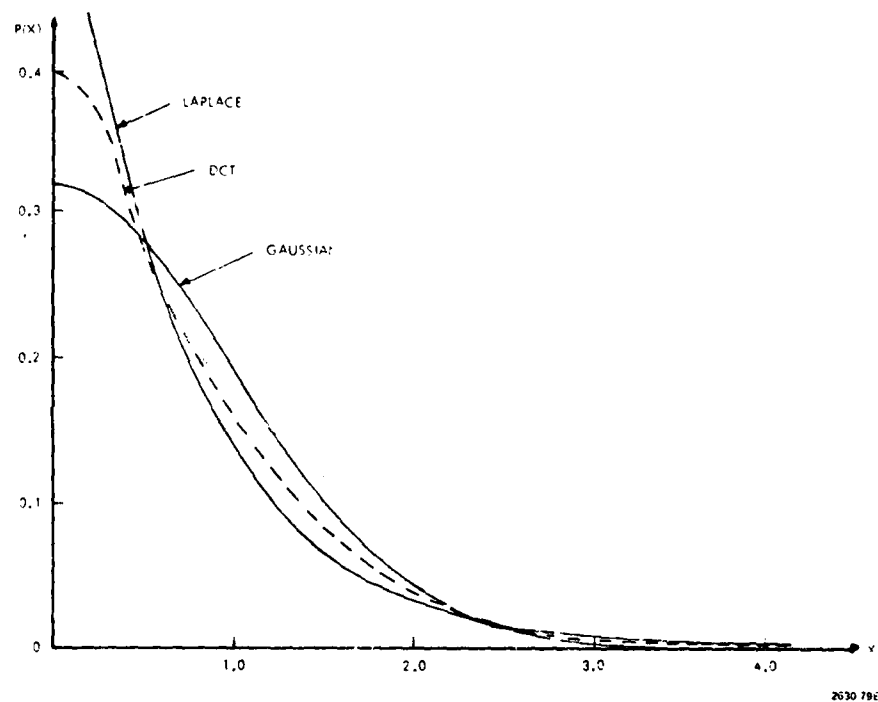


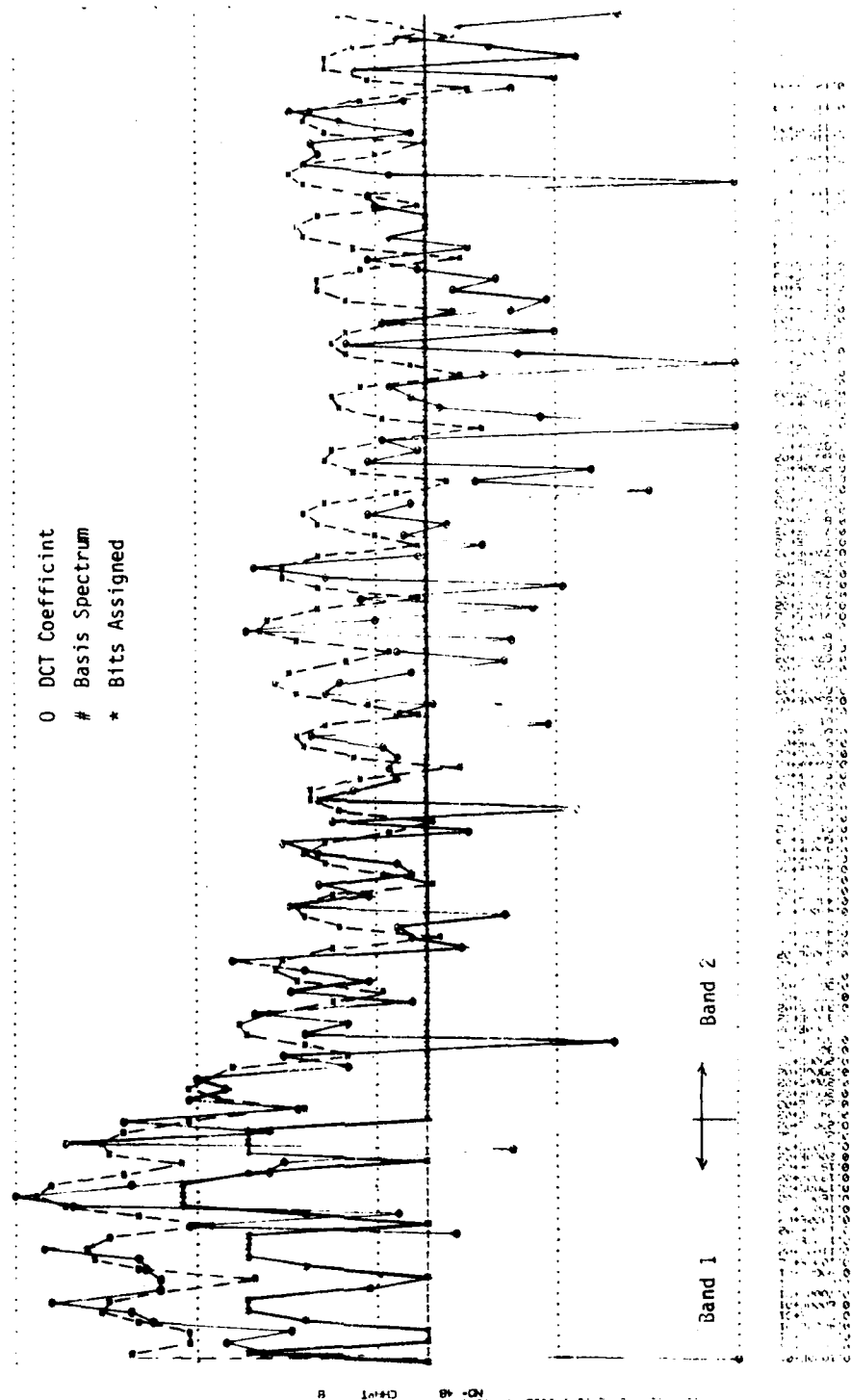
Figure 2.3-6 Probability Density Function of Discrete Cosine Transform Coefficient

The quantizer for each variable when optimized with the proper statistical error criterion, appears adequate for the ATC system over a variety of different speakers and acoustics noise conditions and data rates.

2.3.5 Reducing the Effects of Lowpass Filtering

The quality of the ATC coder degrades as the transmission data rate decreases. One of the major sources of this degradation is the low pass filtering effect which can be explained by examining Figure 2.3-7. This figure shows that no DCT coefficient is transmitted in frequency band 2. However, Figure 2.3-7 illustrates that the basis spectrum which is derived from LPC techniques closely follows the actual spectrum. The DCT coefficients of frequency band 1 are quantized from the LPC basis spectrum, where the phase (sign in this case) and amplitude information are modified from the LPC basis spectrum. This change results in an improvement over the LPC technique. In frequency band 2, the LPC basis spectrum cannot be modified since no bits are assigned. Zelinski and Noll ², who use a different basis spectrum, substitute the DCT coefficients in this frequency band 2 with the noise samples whose variances are derived from the side information. Some improvements have been reported at low data rates.

This technique perceptually adds some bandwidth to the ATC system, but introduces some hoarseness to the speech. This hoarseness arises from the destruction of pitch harmonics in the frequency domain, and can be reduced by using the LPC basis spectrum modified by the pitch weighting function of eq. (2.3-5). This basis function is shown in Figure 2.3-7 with "#" symbol. The optimized ATC coder with the "fill in" procedure produces high quality synthesized speech above the data rate 7200 b/s.



2629-79E

Figure 2.3-7: DCT, Basis Spectrum and Bits Assigned in ATC Coder

2.3.6 Bit Allocations to Sideband and Mainband

The performance of the ATC system depends on the several system devices, including quantizer characteristics, bit assignment rules, methods of estimating the basis function, bit allocations to the sideband and mainbands, etc. It has been shown that the estimation of the basis function plays an important role on the performance of the ATC system. However, it is desirable to allocate fewer bits for the generation of the basis function so that more bits remain for encoding DCT coefficients. Thus, tradeoff analyses were conducted in the area of DCT coefficient and LPC parameter quantization.

First, the performance of the ATC system was measured with the basis spectrums estimated by 10th order and 8th order LPC process (no quantization is applied to the PARCOR parameters). Both SNR measurements and informal listening tests have shown no significant differences. This is an important finding, since the quantization of this sideband information consumes a fair amount of the available data rate. Any conservation of bits in the LPC process can be used to improve or protect the transmission of the DCT coefficients.

Second, a large data base, comprised of 15 male and female speakers totalling 30,000 frames, was used to create relative frequency histograms for each PARCOR coefficient. The probability density functions were derived from this data and used with the technique described in section 2.3.4 to develop optimal quantizers.

The bit allocations strategies for the sideband information are shown in Table 2.3-1. Combining all the sideband information parameters, the total data rate is in the range of $35 \leq \text{bits/frame} \leq 45$. Since it is

<u>Parameter</u>	<u># of bits/frame</u>	
<u>PARCOR #</u>	<u># of Bits Tested</u>	<u># of bits Decided</u>
1	4, 5	5
2	3, 4, 5	5
3	3, 4	4
4	3, 4	4
5	2, 3	3
6	2, 3	3
7	2, 3	2
8	2, 3	2
pitch, M	6	6
pitch gain, G	2, 3	2
variance	5	5
sync	1	1
V/UV	1	1

TABLE 2.3-1 BIT ALLOCATIONS TO THE SIDEBAND INFORMATION

necessary to allocate bits for the error correcting code in order to reduce the effects of channel errors, the data rate may be increased to $85 \leq \text{bits/frame} \leq 95$. By allocating 7200 bits/second for the encoding of DCT coefficients, there are 2400 b/s left for the sideband information. This limits the frame updating rate to less than 30 frames/second which forces the frame size of 256 samples with a 6400 Hz sampling rate. In order to reduce the effects of the signal discontinuities at the frame boundaries, the frames are overlapped slightly (10 samples). Therefore, there are 369 bits per frame (246 samples) with 6400 sampling rate for the 9.6 Kb/s ATC system.

With the above constraints, the performance of the 9.6 Kbps ATC system was evaluated with various combinations of bit allocations to the sideband information parameters. As a result, the combination of the bits sequence shown in Table 2.3-1 was determined to be optimal with respect to objective measurements (SNR). The performance of the ATC system is plotted with respect to the data rate in Figure 2.3-8 with the sideband data rate shown in Table 2.3-1. The figure shows that the decision on the sideband data rate is adequate for the ATC system of 6800 b/s ~ 9600 b/s, since its performance is not sensitive to the changes of the data rate.

2.3.7 Reducing Discontinuities at the Frame Boundary

The adaptive transform coding scheme of Figure 2.2-2 produces noise-like "burbling" and "click" sounds at low data rates. This noise is generated at the frame boundaries by waveshape discontinuities in the time domain. The noise generated from these discontinuities cannot be entirely eliminated, but can be reduced by overlapping frames slightly and by interpolating across the frame boundaries.

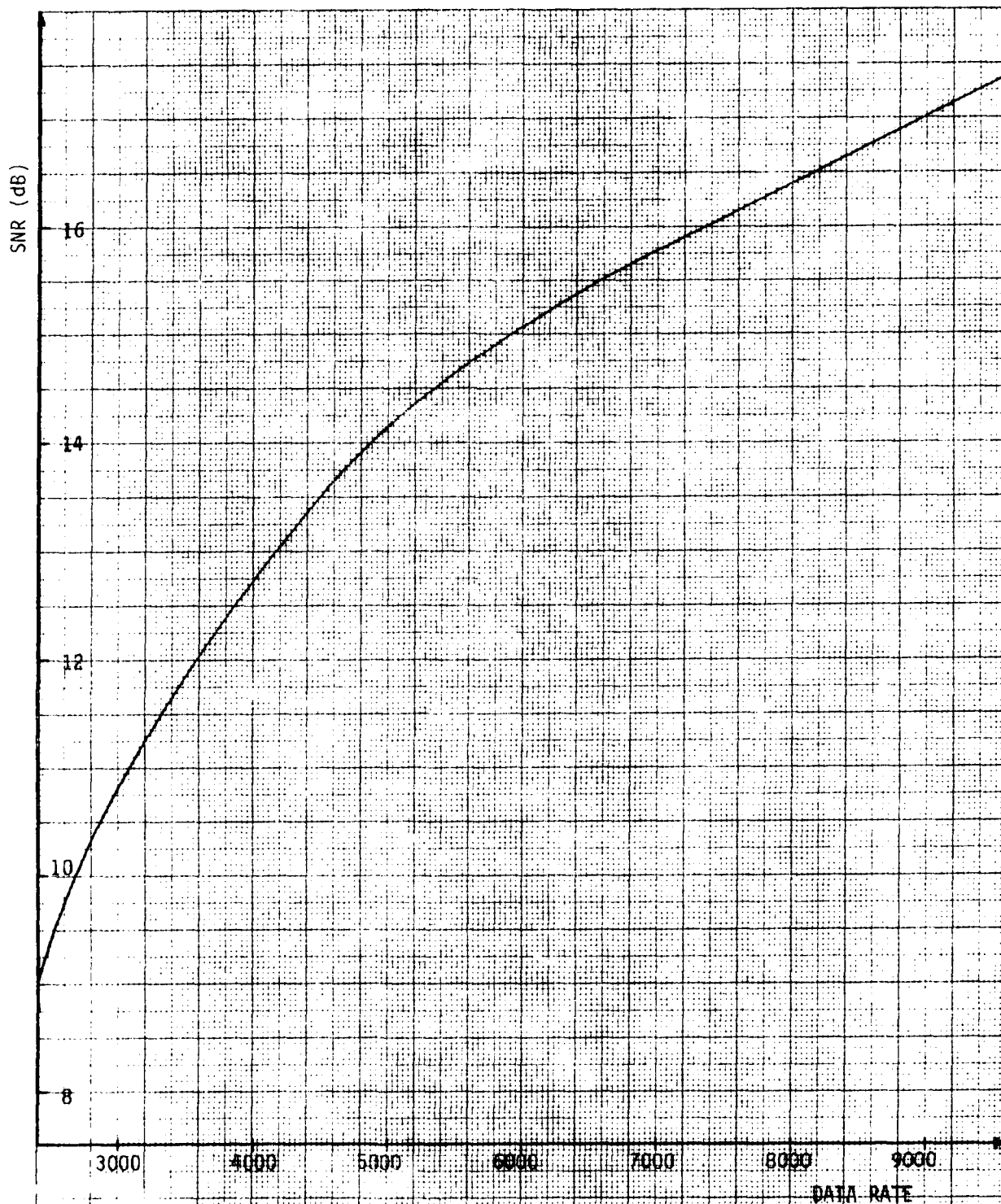


FIGURE 2.3-6 THE PERFORMANCE OF AN ATC SYSTEM

The frame size of the ATC system may be chosen as 128 samples/frame in the previous section. However, the frame size was increased to 256 samples to reduce the effects of signal discontinuities at the frame boundaries. The FORTRAN simulations of the ATC system at frame sizes of 128 and 256 samples revealed two beneficial findings. First, a larger frame size does not adversely affect the signal-to-noise ratio (SNR) but noticeably improves the subjective voice quality. Second, a larger frame size with pitch weighting is better than a smaller frame size with pitch weighting. Both these findings can be explained rather simply. The short term speech spectrum may not be stationary for a large frame size (246 samples), which may cause the synthesized spectrum to be smoothed more than it should be. However, since the frames are updated half as often, there are half as many frame discontinuities. In the ATC system, the frame discontinuities are the most obvious distortion and are lessened significantly with the 256 sample frame size. As the frame size increases, the resolution of the FFT increases as well due to an increase in the FFT order (N), i.e.,

$$\text{frequency resolution} = \frac{BW}{N}, \quad BW = \text{signal bandwidth} \quad (2.3-10)$$

A finer frequency resolution of the pitch weighting spectrum places more striations in the LPC basis spectrum. Hence, a more detailed sampling of the spectrum is achieved for larger frame sizes.

2.4 ATC System in the Presence of Random Channel Errors

Our FORTRAN simulations have shown that the ATC system produces high quality synthesized speech at 9600 bps if no channel errors are present. Error-free transmission, however, is not always possible since the transmitted signals are often affected by channel characteristics and by noise which may or may not vary in time. Additive Gaussian noise is the main source of signal corruption in many digital data transmission systems that will introduce random channel errors (error positions are independent of time). These channel errors may degrade speech quality, and the degradation of speech may depend on the positions of these channel errors.

In the following sections, the design of the ATC system will be examined and changed to optimize the performance of the system under the influence of random channel errors ranging from a bit error rate (BER) of 0 to 10^{-2} . First, the effects of random channel errors on the performance of the ATC system will be examined in section 2.4.1. Afterwards, the performance of the ATC system, as a function of data rate and channel error rate, will be provided in section 2.4.2. Then forward error correcting codes will be employed to reduce the effects of channel errors. The application of BCH codes, which are presently the most powerful random-error-correcting codes, will be presented in section 2.4.3. The performance of the ATC system is sensitive to the errors in the sideband information, since the bit assignments of the DCT coefficients depend on the sideband information. Some DCT coefficients are more important than the others in the sense of maintaining the system's performance with no channel errors. The selection and protection of the important bits in ATC system were made by analyzing the performance of the ATC system with various bits protected. Then, in section 2.4.4, we selected parameters

of BCH code used to protect these bits. Finally, the conclusions are summarized in section 2.4.5.

2.4.1 The Effects of Random Channel Errors on the Performance of the ATC System at 9.6 Kbps

The ATC algorithm was originally designed for the error-free channel. A noisy channel, however, will introduce errors in the received bits. The performance of the ATC coder given by its signal-to-quantization noise ratio (SNR) is plotted in Figure 2.4-1 with respect to the channel error rate. These plots are obtained by evaluating various types of speech totally about 30 sec.

Degradation of the synthesized speech due to the channel errors was not noticed in the informal listening tests when the channel error rate was lower than 10^{-3} . However, the quality of the speech as well as the SNR drops rapidly when the channel error rate is higher than 10^{-3} . It is, therefore, desirable to protect the system performance at the higher channel error rates: ($>10^{-3}$).

2.4.2 Tradeoff Analysis Between Data Rate and Channel Error Rate

The performance of the ATC system was evaluated in terms of the SNR with the transmission data rate varying from 7700 to 9600 bps. The results are plotted in Figure 2.4-2 together with the performance of the ATC system under the influence of random channel errors. As it is noted from this Figure, the performance of the system does not drop rapidly as the transmission data rate decreases, while the performance of the system drops rapidly when the channel error rate is higher than 10^{-3} . Since

M-210A-10 TO THE NEXT PETER 46 1017

SNR dB

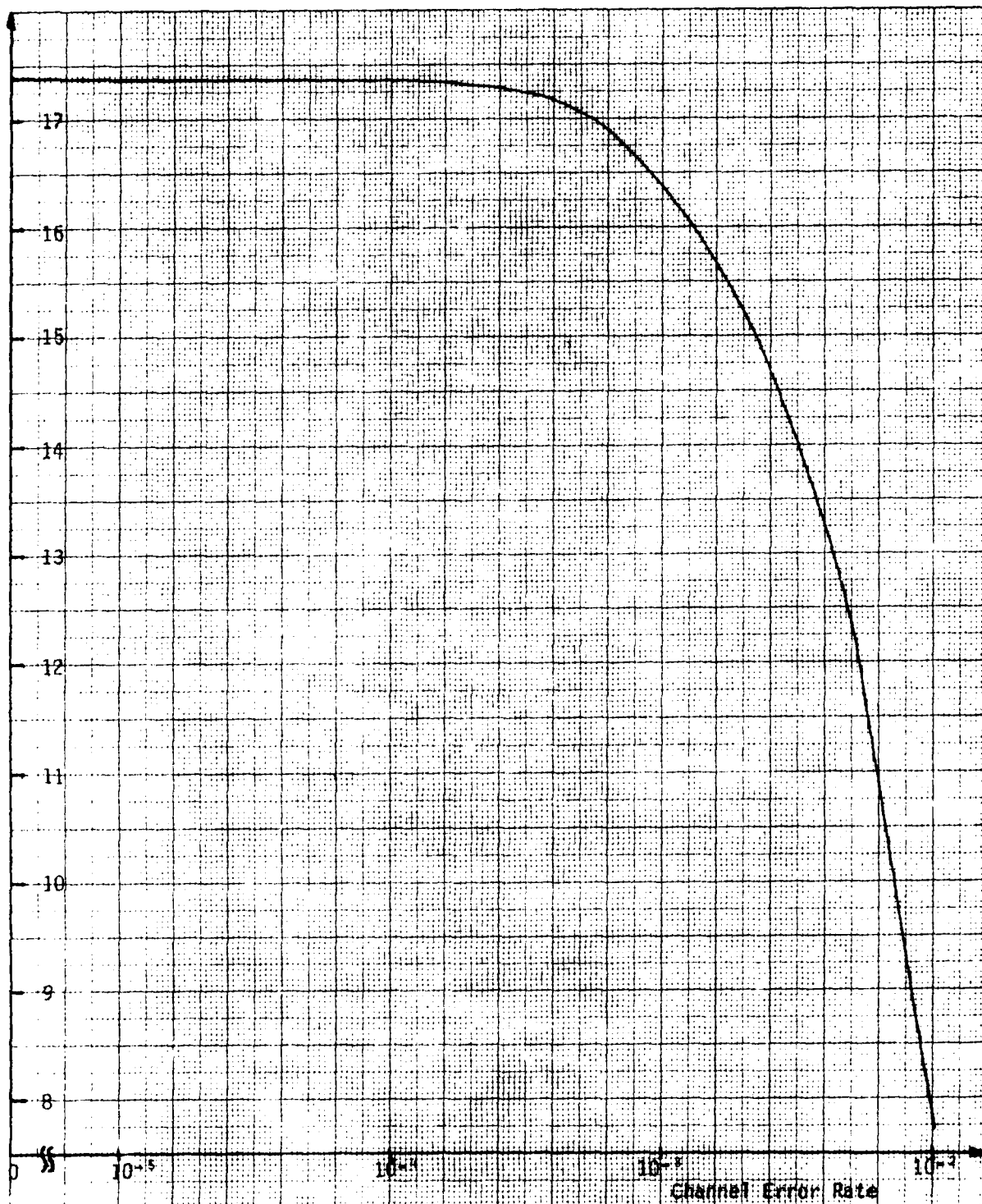


FIGURE 2.4-1: THE PERFORMANCE OF ATC SYSTEM UNDER NOISY CHANNEL.

some channel errors can be corrected by utilizing error correcting codes, the error rate of the information bits can be reduced depending on the channel characteristics and the selection of error correcting codes. However, additional information (parity bits) has to be sent to the receiver to correct channel errors. Therefore, reducing the channel errors requires reducing the source information rate. Figure 2.4-2 shows that the performance of the ATC system can be improved by using the error correcting codes when the channel error rate is higher than 10^{-3} since the performance of the system drops slowly when the source information rate decreases. However, error correcting codes are not advisable to reduce the effects of channel errors when the error rate is lower than 10^{-3} since the performance of the system does not drop rapidly as the channel error rate increases.

2.4.3 Application of BCH Code

There are many ways of utilizing error correction codes to reduce the effects of channel errors. The method of correcting errors depends on the application, i.e., data rate, channel error rate, complexity, cost, etc. Since the ATC coder is designed for real-time implementation, error correcting code must not require a large time delay for correcting channel errors. Block codes of short length are suited to the real-time implementation of ATC algorithm. These codes require no additional time delay to process the error correcting algorithm if the length of the block code is less than the number of bits received in a frame period. There are many types of block codes that can be properly used depending on the channel characteristics.

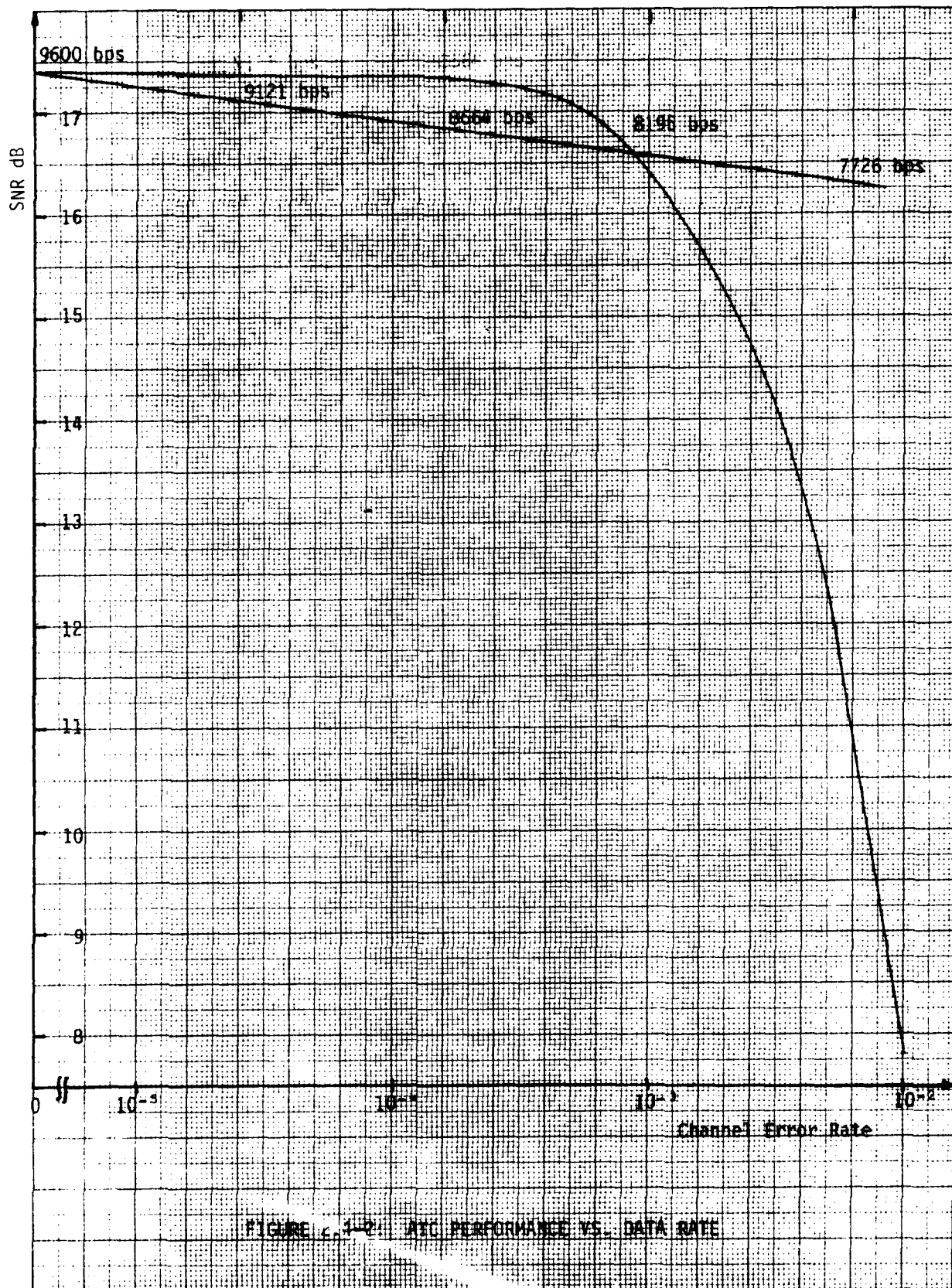


FIGURE 2-1-21 ATC PERFORMANCE VS. DATA RATE

Most real communication channels corrupt signals in many ways. The signals may be corrupted by the additive Gaussian noise and/or impulsive noise that produce random and burst errors, respectively. In other situations, the characteristics of the channel may vary in time (fading channel) or the channel may be selected at random from one ensemble of channels with widely different characteristics such as the switched telephone network. It is very hard to construct an error-control system which adapts to various types of channels. Since additive Gaussian noise is the main source that corrupt signals in many practical communication channels, the most practical error correcting code is the one which is capable of correcting random errors.

The Base-Chaudhuri-Hocquenhem (BCH) codes, that are a generalization of Hamming codes for correcting multiple errors. They are well known to be the most powerful random-error correcting codes, and a decoding algorithm that can be implemented with a reasonable amount of complexity has been devised for these codes.^{10, 11, 12} A more fundamental description of the BCH codes and their encoding, decoding algorithm are given in Appendix A. This appendix shows that with the block length of the code $n=2^m-1$ and mt parity checks, it is possible to correct any t or less errors in a primitive (n,k) BCH code where k is the number of information bits. The proper choice of m , n , t in primitive BCH code depends on the channel error rate, data rate, and the system's specifications. For the real-time implementation of ATC coder, the values of $t=3$ and $m=6, 7, 8$ are considered as reasonable choices.

The performance of random-error correcting BCH codes is expressed in terms of error-probability. Let $P(m,n)$ be the probability of m errors occurring in an n -bit block and β_m denote the probability of decoding an

error pattern of weight m correctly, then the probability of decoding the received code word erroneously may be expressed as

$$\begin{aligned}
 P &= 1 - \sum_{m=0}^n \beta_m P(m,n) \\
 &= \sum_{m=0}^n \alpha_m P(m,n)
 \end{aligned}
 \tag{2.4-1}$$

where $\alpha_m = 1 - \beta_m$ denotes the probability of erroneously decoding an error pattern of weight m . The parameter α_m is a function of the code and decoding algorithm. If a t error-correcting BCH code is employed and it is decoded with the Peterson decoding algorithm shown in Appendix A, the parameter α_m may be expressed as

$$\alpha_m = \begin{cases} 0 & 0 \leq m \leq t \\ 1 & t < m \leq n \end{cases}
 \tag{2.4-2}$$

and the probability of erroneously decoding the code word may be reduced from eq. (2.4-1) as

$$P_e = \sum_{m=t+1}^n p(m,n)
 \tag{2.4-3}$$

If the bit errors occur independently and at random with probability e , then the probability $p(m,n)$ can be expressed as

$$p(m,n) = \binom{n}{m} e^m (1-e)^{n-m}
 \tag{2.4-4}$$

where the probability $p(m,n)$ is simply the binomial distribution and P_e in eq. (2.4-3) is simply the tail of the distribution.

Let the channel error rate $e = 10^{-2}$ which is specified by the contract. Let the block length of the BCH code $n = 127$ and $t = 3$, then the probability of error occurring in the block can be written as

$$\begin{aligned}
 P_e &= \sum_{m=4}^{127} p(m, 127) \\
 &= 1 - p(0, 127) - p(1, 127) - p(2, 127) - p(3, 127) \\
 &= 0.0393 \quad (2.4-5)
 \end{aligned}$$

In this BCH code, the information rate may be expressed as

$$\begin{aligned}
 R &= k/n \\
 &= 106/127 \\
 &= 0.8346 \quad (2.4-6)
 \end{aligned}$$

where 16.54% of the data is used for the redundant parity checks. The information rate for the (127,106) BCH code from eq. (2.4-6) is a high 83.46%. However, the probability of error occurring in the block is also high since from eq. (2.4-5), it is expected to have one block in error for each 25 blocks. Let $n = 63$, $k = 45$, $t = 3$ ((63,45) BCH Code), then the probability of error occurring in the block of 63 bits will be

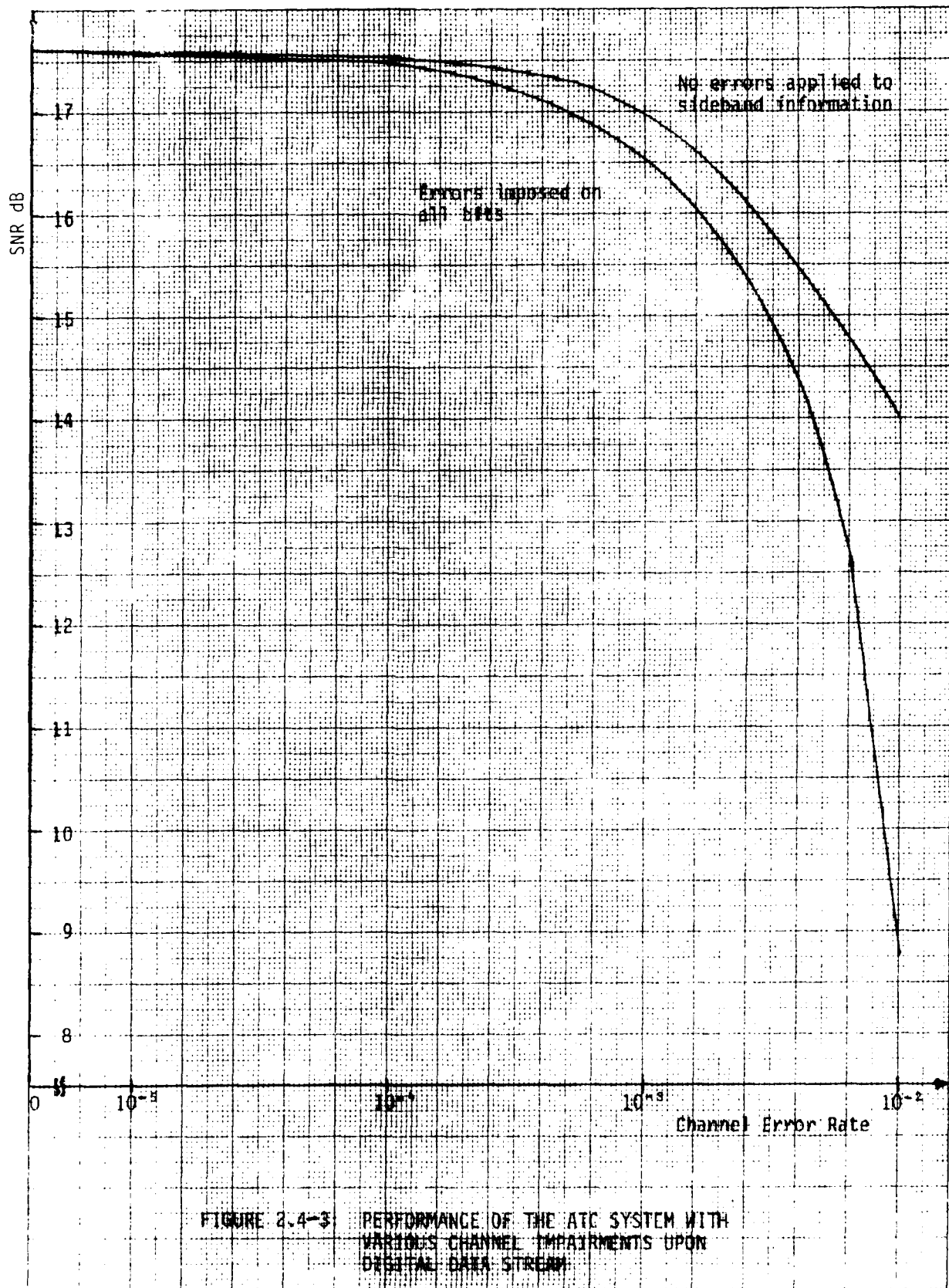
$$\begin{aligned}
 P_e &= \sum_{m=4}^{63} p(m, 63) \\
 &= 1 - p(0, 63) - p(1, 63) - p(2, 63) - p(3, 63) \\
 &= 0.003725 \quad (2.4-7)
 \end{aligned}$$

The information rate R is about 71.42%, which is lower than one of the (127,106) BCH code, but one erroneous block is expected out of 268 blocks, which turns out to be a proper choice in the following section.

2.4.4 Selection and Protection of the Important Bits in the ATC System

The performance of the ATC system was evaluated under various simulated channel error rates (random errors) to see the effects of channel errors. Two independent error generators were used on separate regions of the bit allocation strategies to isolate the most sensitive bits out of the 9600 bps system. One error rate, error rate A, was applied to the bit stream of the DCT coefficients. The other error rate, error rate B, was applied to the bits allocated to the sideband information parameters. The results are plotted in Figure 2.4-3. In this figure, the cumulative SNR does not degrade more than 9 dB when the channel errors are applied to every bit at the rate 10^{-2} . Although the processed sentence is generally intelligible, there are periods when concentrated errors lead to undesirable distortions (pops, clicks, etc.). On the other hand, a few bits of protection on the sideband information (42 bits/frame in a 369 bits/frame) lead to the improvement of SNR by 4.2 dB. Therefore, protection of the sideband information from the channel errors is necessary to minimize the reduction in SNR, since the performance of the ATC system is very sensitive to the errors in the sideband information.

To further improve the system's performance at error rates less than 10^{-2} , a primitive BCH code was applied for the partial protection of the bits related to the DCT coefficients as well as for the protection of sideband information. The protection



of all information is not considered practical because of software requirements (computation time and program size), and as Figure 2.4-5 shows, the protection of all information does not lead to the highest system performance at the error rates below 10^{-2} .

In the early simulations of ATC system with channel errors by Zelinski and Noll,^{1, 2} no channel errors are applied to the sideband information or to the most significant bit of each DCT coefficient. The performance of the system was shown to be insensitive to the changes of channel error rates up to 5%. Although the ATC system has been modified, the protection of the most significant bit from each DCT coefficient may lead to a good selection of important bits. We modified this scheme shown in Figure 2.4-4 (diagonal protections). In this figure, the quantized DCT coefficients (not the value of the quantized DCT, but the decimal number or address of the quantization tables which is ready to serialize for the transmission through the channel) are ordered in descending magnitude. The information on magnitude and the number of bits for each DCT is obtained from the sideband information. In the "diagonal protections," the bits to be protected are selected in the sequence of 1→6→10→14→17→ --- up to the desired number of bits. Another way of selecting important bits is the technique of "horizontal protections" where the bits are selected for protection in the horizontal direction in Figure 2.4-4 as 1→2→6→10→3→7→ up to the desired number of bits. In order to select the number of bits to be protected, two block lengths of BCH codes (i.e., (63,45) and 127,106) BCH code) have been applied to the ATC system with the selections of important bits described as "horizontal protections" and "diagonal protections." The number of bits to be protected as well as the performance of the ATC system are tabulated in Table 2.4-1 at the channel error rate

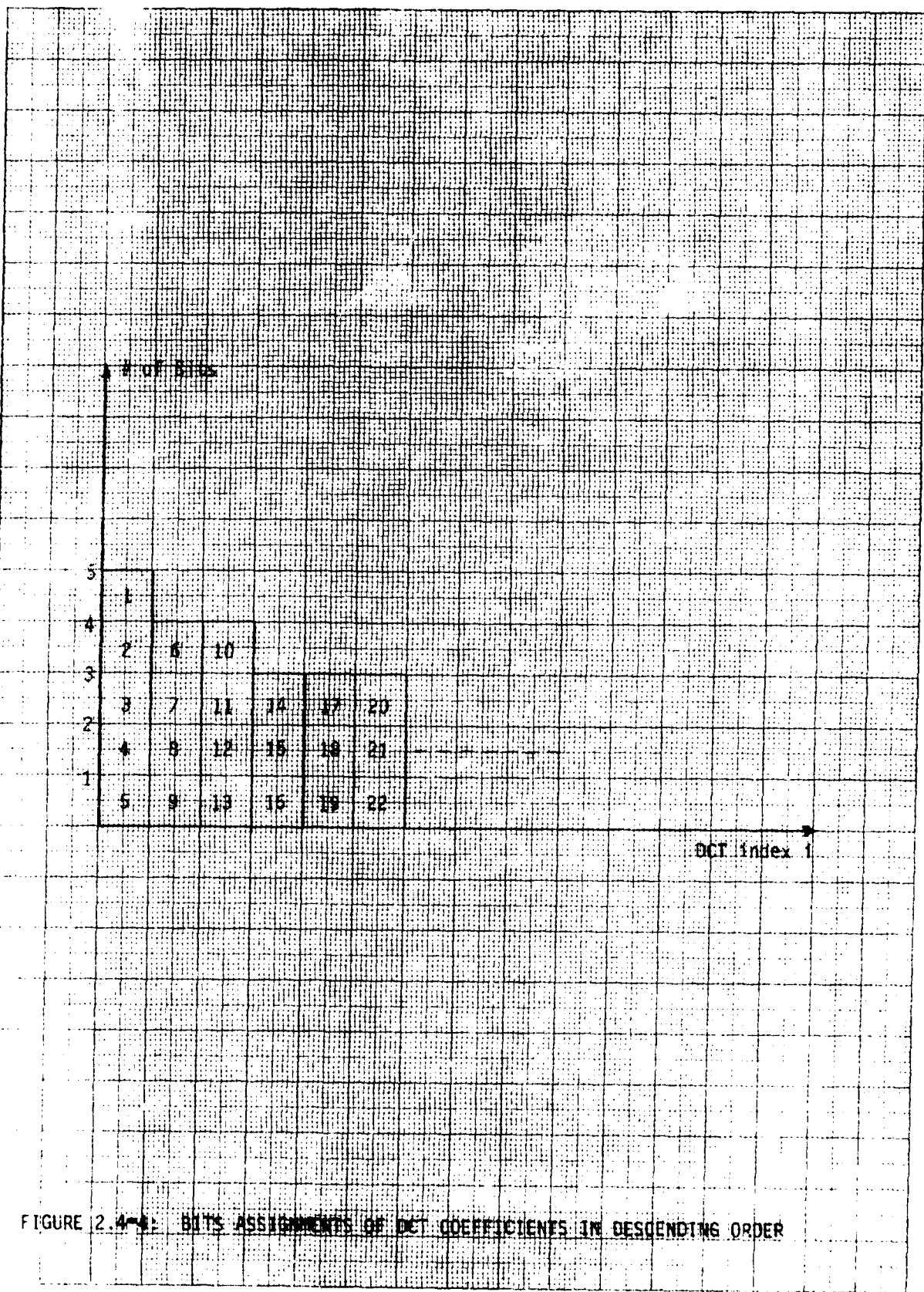


FIGURE 2.4-4: BITS ASSIGNMENTS OF DCT COEFFICIENTS IN DESCENDING ORDER

Program Name	Number of Bits Protection	Number of Bits Overhead	Name of BCH Code used 2 times	SNR at Channel Error Rate 0	SNR at Channel Error Rate at 10^{-7}
ATC	0	0	*	17.3 dB	8.8 dB
ATCD Diagonal Protection	45 x 2	18 x 2	(63, 45)	15.7	13.7
	90	36			
	106 x 2	23 x 2	(127, 106)	15.6	14.4
ATCH Horizontal Protection	212	42			
	45 x 2	18 x 2	(63, 45)	15.7	14.5
	90	36			
	106 x 2	23 x 2	(127, 106)	15.6	15.7
	212	42			
*Total Number of Bits per Frame = 360 Bits					
TABLE 2.4-1: PERFORMANCE OF ATC SYSTEM UNDER VARIOUS CHANNEL CONDITIONS					

0 and 10^{-2} . As it is noted from the table, "horizontal protection" performs better than "diagonal protection" by 1.3 dB when two blocks of a (127,106) BCH code are applied to the ATC system at the channel error rate 10^{-2} . Another observation is that the (127,106) BCH code improves performance over a (63,45) BCH code by 1.1 dB in "horizontal protection" at the channel error rate 10^{-2} . However, informal listening tests indicate that there are periods that contain a large amount of distortions (pops, clicks, etc.) which lead to major objectionable speech degradation at 9600 bps. The main source of this degradation are the frames of speech that have more than 3 errors which cannot be corrected by the system. The probability of more than 3 errors occurring in a block is 0.039 from eq. (2.4-4) when the (127,106) BCH code is employed for the channel error corrections of rate 10^{-2} . This probability is reduced to 0.0037 when the (63,45) BCH code is incorporated. Thus, to reduce the number of frames that contain a large amount of distortions, one should use the (63,45) BCH code rather than the (127,106) BCH code when the channel error rate is 10^{-2} .

Finally, the Table 2.4-1 indicates that it may be better to protect more than 90 bits out of a frame (369 bits/frame) to increase the SNR at the error rate 10^{-2} . To protect more bits, one must use more parity bits which reduces the number of bits for encoding speech signals. The trade-off analysis between the number of parity bits and error rates on the performance of the ATC system was investigated by using a (63,45) BCH code coupled with selecting the important bits by the technique of "horizontal protection."

To find out the best number of bits to be protected, the performances of the ATC system were evaluated at the several different channel

error rates by varying the number of blocks for (63,45) BCH code in a frame. The results are tabulated in Table 2.4-2. The performance of the system is plotted in Figure 2.4-5 when the channel error rate is fixed at several values and the number of blocks of a (63,45) BCH code is a variable from 0 to 5. The performance of the system is also plotted in Figure 2.4-6 when the number of blocks protected from channel errors is fixed at some values and the channel error rate varies from 0 to 10^{-2} . As it is noted from Figure 2.4-5, the performance of the ATC system improves rapidly as the number of blocks (or number of bits) to be protected increases at the channel error rate 10^{-2} . As the number of blocks reaches 3, the performance of the ATC system saturates, while the best performance (denoted by Δ in the Figure 2.4-5) is obtained when 4 blocks of a (63,45) BCH code are employed at an error rate of 10^{-2} . At the channel error rate 5×10^{-3} , the best performance is obtained when the 3 blocks of a (63,45) BCH code are applied to protect the important bits of the ATC system. At the channel error rate 10^{-3} , the protection of one block (45 bits) is sufficient to obtain the best performance of the system. It is not necessary to protect any bits in order to reduce the effects of channel errors when the error rate is lower than 5×10^{-4} .

For the real-time implementation of the ATC system, the recommendation was to use the 3 blocks of a (63,45) BCH code because of the computation time and the saturation of the ATC system's performance. This conclusion was reached from the Figure 2.4-6, since the performance of the ATC system, when 3 blocks of a (63,45) BCH code is employed to reduce the effects of channel errors, is consistent and high for the channel error rates below 10^{-2} . The degradation of the performance due to the channel errors of rate up to 10^{-2} is less than 0.76 dB. Informal listening tests indicate

# of Error Rate Blocks Protected	0.0	10^{-4}	5×10^{-4}	10^{-3}	5×10^{-3}	10^{-2}
0 (0*)	17.39 dB	17.34	17.07	16.37	12.35	7.80
1 (45)	17.13	17.11	16.95	16.75	15.01	13.33
2 (90)	16.84	16.82	16.73	16.64	15.01	15.19
3 (135)	16.59	16.58	16.56	16.53	15.27	15.83
4 (180)	16.30	16.30	16.30	16.28	15.19	15.95
5 (225)	15.96	15.96	15.96	15.96	15.92	15.81

* The number inside the parentheses indicate the number of bits protected

TABLE 2.4-2 PERFORMANCE OF THE ATC SYSTEM WITH VARIOUS CHANNEL CONDITIONS

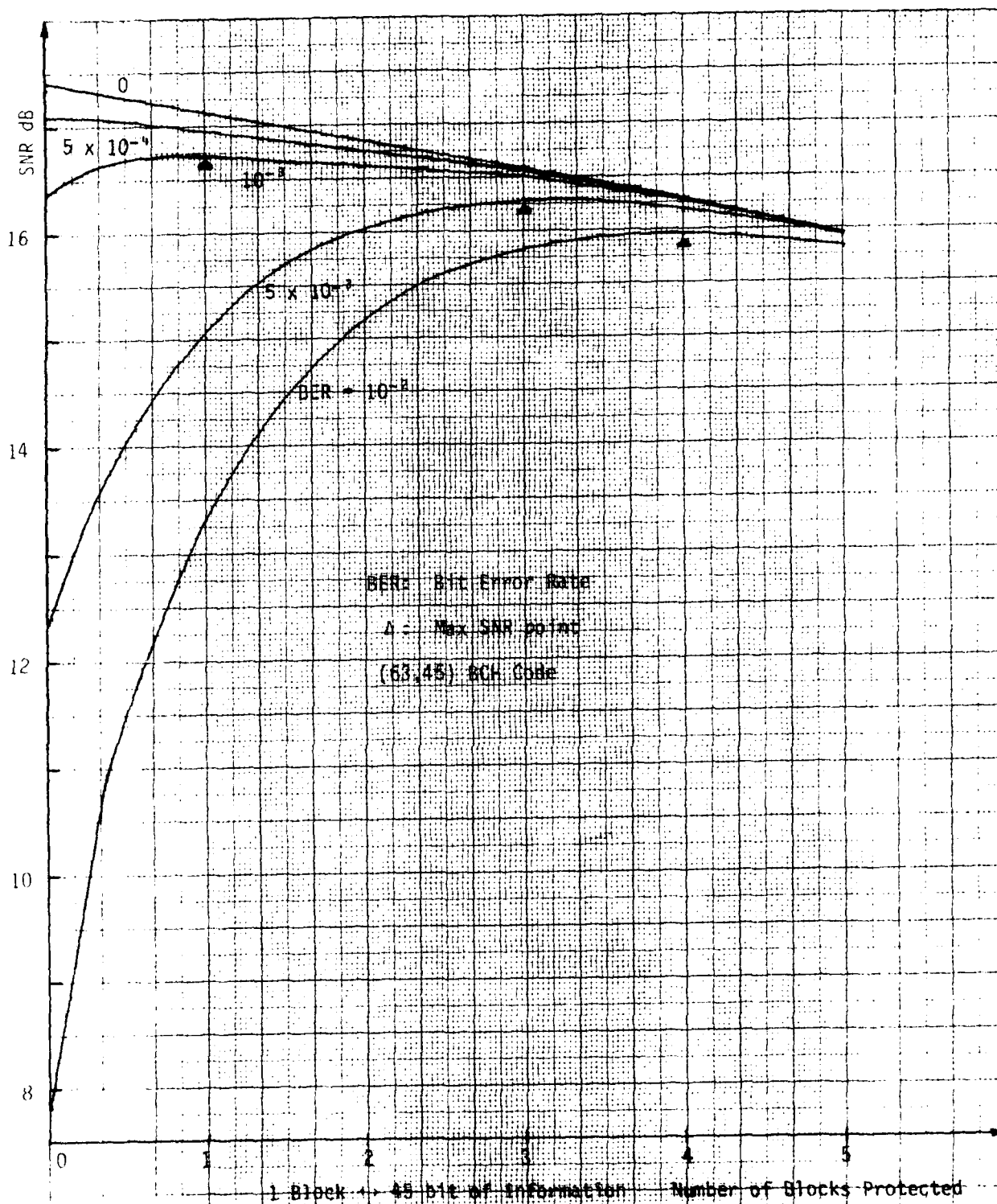


FIGURE 2.4-B: PERFORMANCE OF THE ATC SYSTEM VS. NUMBER OF BLOCKS PROTECTED

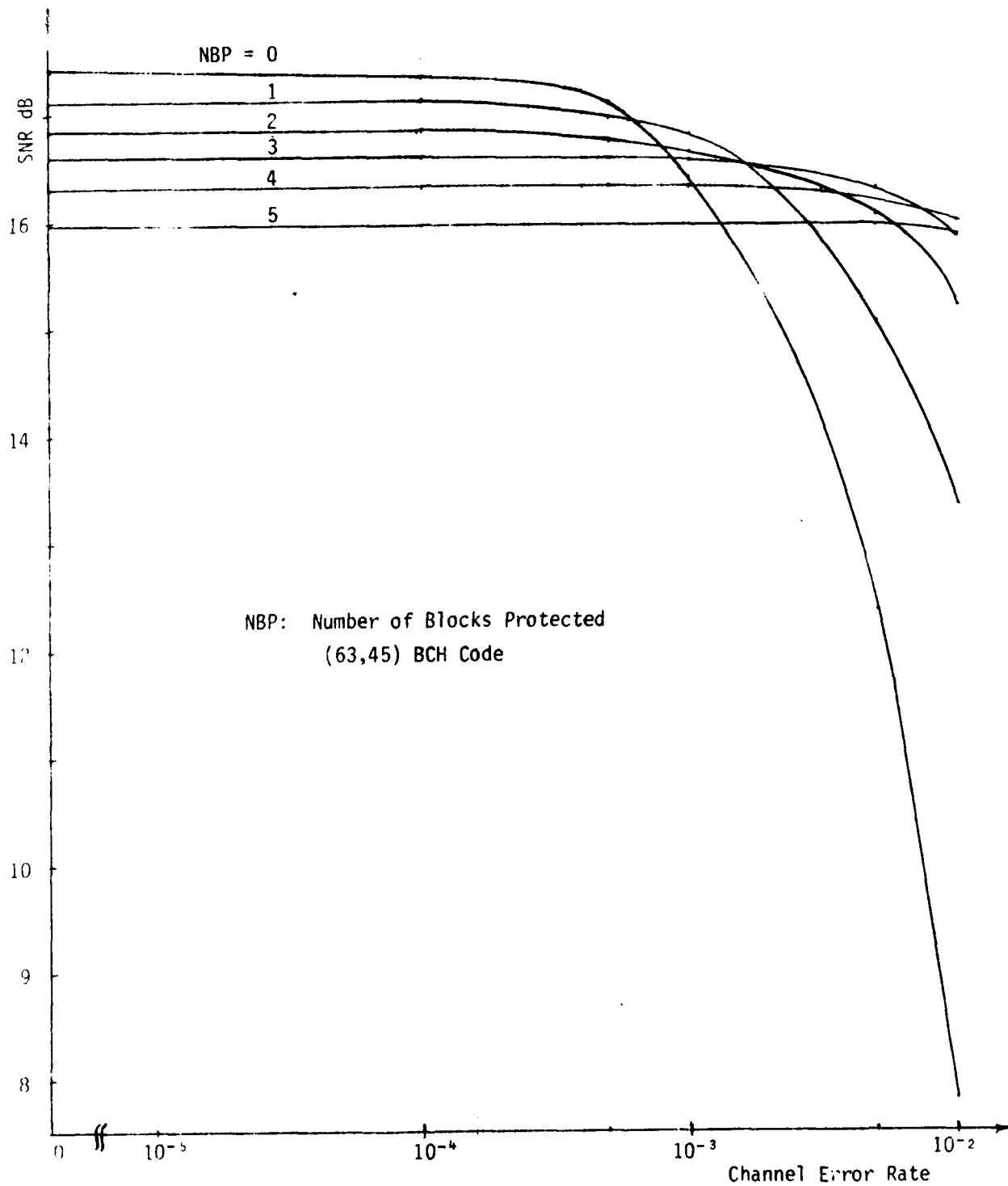


FIGURE 2.4-6: PERFORMANCE OF THE ATC SYSTEM VS. CHANNEL ERROR RATES

that the speech quality of 9600 bps is high and consistent in the presence of channel errors of rate up to 10^{-2} .

2.4.5 Summary

The ATC system had its transmission data sent through a Gaussian noise channel to investigate effects of random channel errors. The degradation of the speech quality or the signal-to-quantization noise ratio does not degrade significantly when the channel error rate is lower than 10^{-3} . However, the degradation of speech quality, when the channel error rate is higher than 10^{-3} , is so severe that one must reduce the effects of the channel errors. Since the performance of the ATC system has been insensitive with respect to the changes of the source information data rate, it was possible to employ the error correcting code to reduce the effects of the channel errors.

BCH codes were briefly described and were used to improve performance with channel errors. The selection and protection of the important bits in the ATC system were conducted by utilizing small block lengths (i.e., 63 or 127) of BCH code which correct up to 3 errors in the block. As a result, the selection of the important bits in ATC system were made by the technique of "horizontal protections." The (63,45) BCH code was also selected to reduce the number of periods which contain a large amount of signal distortion caused by a large number of burst errors (≥ 4) in the protected block.

The optimum performance of the ATC system was obtained for a given channel error rate. For example, the optimum performance of the ATC system (15.95 dB of SNR) was obtained when the 4 blocks of a (63,45) BCH code are incorporated to reduce the effects of the channel errors

of the rate 10^{-2} . However, we recommended using 3 blocks of a (63,45) BCH code for the protection of channel errors up to the rate 10^{-2} because of the saturation of the ATC system's performance and real-time computation capability.

Finally, based on the SNR performance, we have shown that the ATC system designed in this study produces a high and consistent quality of speech at the data rate 9600 bps in the presence of random channel errors up to the rate 10^{-2} .

2.5 FORTRAN Program for the Simulation of the ATC System

The ATC scheme developed in the previous sections is programmed in FORTRAN, and the simulations of the ATC scheme are performed by a PDP-11 computer with a RSX-11M operating system.

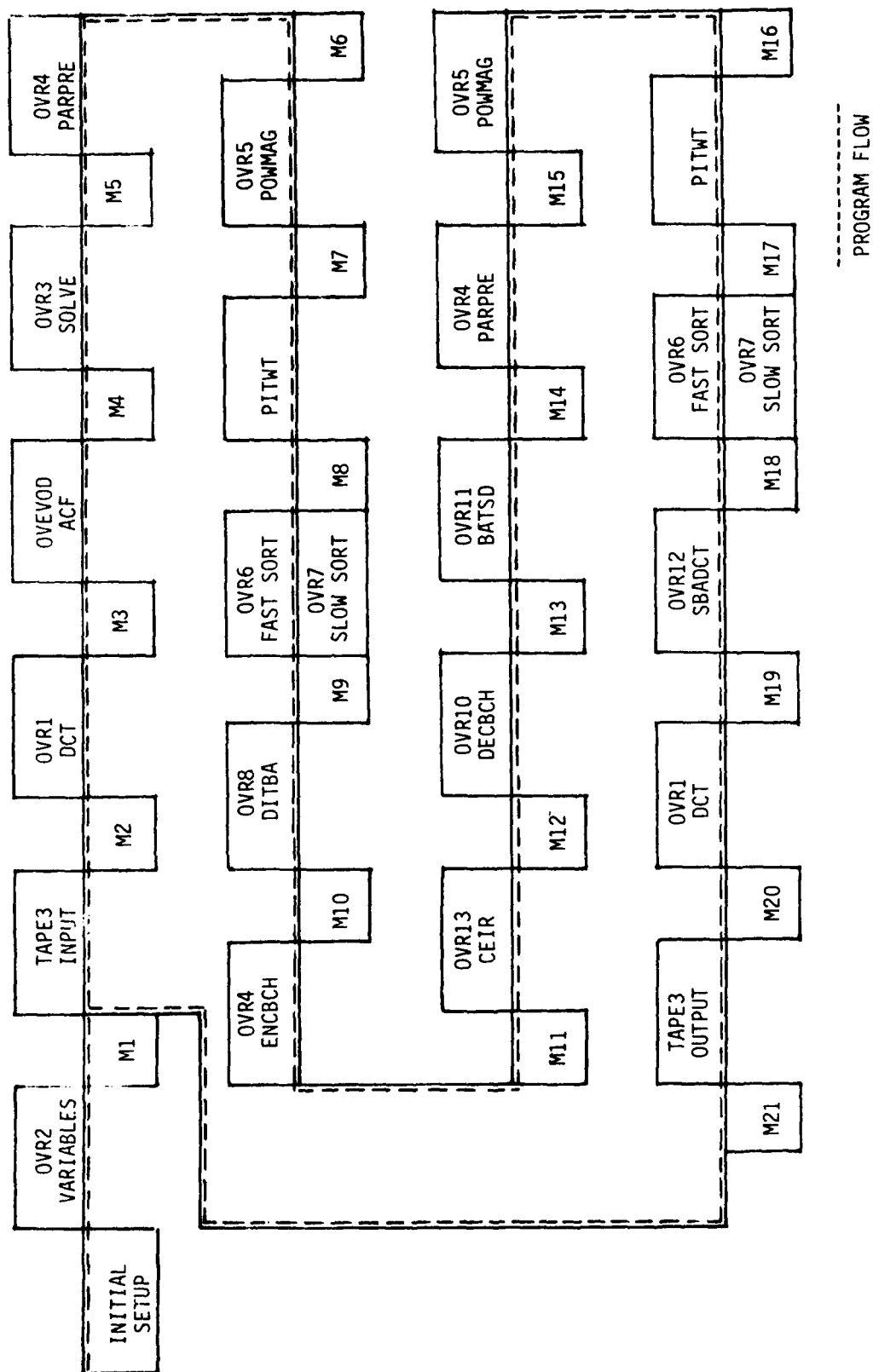
The FORTRAN program will be described first in section 2.5.1. The task building of the program from the source file and the operation of the program will be shown in section 2.5.2. Appendix C contains a source listing of all FORTRAN programs for the ATC simulation.

2.5.1 FORTRAN Program of the ATC Algorithm

The ATC algorithm was developed using the FORTRAN programs before the real-time implementation of the ATC scheme began on the MAP-300 of CSPI, Inc. The flow diagram of the algorithm is shown in Figure 2.5-1. The programs consist of the main routine (ATC 70) and 25 subroutines. The functional descriptions of the program will be given following the flow diagram of Figure 2.5-1.

First, the parameters of the ATC system are defined in the initial setup routine within ATC 70. The quantizer tables for the sideband parameters and DCT coefficients are also defined in this routine. The use of the pitch weighting function, the sorting techniques (fast but approximate or slow but exact), input speech sampling rate, simulation channel error rate, and the information of the input/output speech file (name of the file and the storage device of the system, etc.) are defined in the subroutine OVR2. The characteristics of the ATC system are defined here and the program is ready to execute over and over until it is terminated.

The number of frames that are processed by the program is updated in the main program (labeled M1 on Figure 2.5-1) and the input speech is read and buffered by the subroutine TAPE3. The mean and variance of the input



signal are calculated for the normalization in the main program at M2, and the discrete cosine transform (DCT) is performed on the normalized input signal in the subroutine OVR1. The vectors are shuffled in the main routine at M3 for the fast calculation of the pseudo autocorrelation function (ACF) of the input speech signal which is performed in the subroutine OVEVOD.

The pseudo-ACF is searched for a maximum to obtain the pitch period, M. The corresponding pitch gain, G, is the ratio of the pseudo-ACF at M over its value at the origin. The pitch period, M, and the pitch gain, G, are determined in the main routine at M4. The PARCOR coefficients are calculated from the normalized pseudo-ACF in the subroutine OVR3. The quantizations and dequantizations of the PARCOR coefficients and pitch period, pitch gain, DC bias and variance of the input speech signal are performed in the main program at M5. The LPC filter coefficients are calculated from the PARCOR coefficients in the subroutine OVR4, and the generation of the LPC excitation source is performed in the main program at M6. DFT is performed on the LPC excitation source to get the LPC basis spectrum in the subroutine OVR5, and this spectrum is normalized in the main routine at M7. The pitch weighting function is generated in the subroutine PITWT, and it is multiplied to the LPC basis spectrum to form the ATC basis spectrum in the main program at M8.

The bit assignments rule is derived from the basis spectrum and the quantizations of the DCT coefficients are performed. First, the basis spectrum is sorted in descending magnitude in the subroutine OVR6 (fast sorting routine) or OVR7 depending on the terminal input. The fast sorting routine may provide an approximate result of the slow but exact sorting routine OVR7. The bit assignments routine and the quantizations of the DCT coefficients are performed in the main program at M9. The quantized decimal

inputs are serialized into binary vector in the subroutine OVR8, and the encoding of a (63,45) BCH code is performed in the subroutine OVR9.

The encoded binary data is then transmitted through the simulated noisy channel in which the information of the transmitter may be altered due to the introduction of the channel errors. Simple tests are performed in the main routines at M10, M11, M12, and M13.

At the receiver side, the received sequence of binary data is fed to the decoder routine for the correction of errors if any in the subroutine OVR10. The sideband information is obtained first by unpacking the corrected binary vector in the subroutine OVR11. The sideband information, which consists of PARCOR coefficients, pitch period, pitch gain, mean and variance of the input signal, is dequantized in the main routine at M14. In order to generate the basis spectrum, LPC filter coefficients are calculated from the PARCOR coefficients in subroutine OVR4, and the time domain excitation source for the LPC spectrum is performed in the main routine at M15. The LPC spectrum is generated in the subroutine OVR5, and the pitch weighting function is calculated in the subroutine PITWT for the case of voiced sounds. The ATC basis spectrum is obtained by the multiplication of the LPC basis spectrum and pitch weighting function in the main program at M16 and M17. The basis spectrum is again sorted in descending magnitude in the subroutine OVR6 or OVR7. The bit assignments rule is exercised again from the sorted basis spectrum in the main program at M18. The mainband information is obtained by unpacking the received binary data in the subroutine OVR12. The dequantizations of the DCT coefficients are performed in the main program at M19, and the inverse DCT is performed to reproduce the time domain signal in the subroutine OVR1.

The time domain signal is renormalized by the mean and variance of the input signals and interpolated in order to reduce the effects of the

signal discontinuities at the frame boundaries in the main routine at M20. This reproduced signal is fed to the output device in the subroutine TAPE3, and the post analysis (measuring the signal-to-noise ratio) is performed in the main program at M21. These procedures are repeated until the desired number of frames are processed by the program.

2.5.2 Task Building of the ATC Program

A magnetic tape was sent to DCA containing all of the source files necessary to build the ATC program.

Also included was 10 frames of data with zero and one percent error rates, that are shown in Figure 2.5-2 and Figure 2.5-3, respectively.

The task module of the ATC program can be built as follows:

- I) CATC70.COMD is an indirect command file that compiles all source needed for taskbuilding the overlay ATC program. It also purges all old object files. It also spools the overlay descriptor language program.

*Note that ATC70 is compiled with the slash DE option which allows for printout to LUN 4 all diagnostics in the ATC main program. This requires a larger compiler partition and also requires assignment of LUN 4 to a system device upon installation of the main program. Therefore, if diagnostics are undesired, do not compile with the /DE option. However, there is a rather elegant set of diagnostics, not to be passed over in haste.

This program is invoked by typing (in MCR)

@ CATC.70

11) ATCOLA.CMD task builds the overlay descriptor program creating the task ATC70. If a map is desired, then add the LP option in the BUILD.CMD program.

This program is invoked by typing (in MCR)

TKB @ ATCOLA

The program executes by typing

RUN ATC70

as shown in Figure 2.5-2.

```

DO WE DO SERIALIZATION OF DATA(Y/N)? Y
DO WE INSERT ERRORS IN CHANNEL(Y/N)? Y
ERROR RATE IN E11.4=
DO WE ENCODE AND DECODE(Y/N)? Y
USE PITCH WEIGHTING(Y/N)? Y
USE FAST SORT(Y/N)? Y
SOURCE RATE AND XMIT DATA RATE IN 216=6400,9600
IS THE INPUT ON MAG. TAPE? N
IS THE OUTPUT GOING TO MAG TAPE? N
DEFAULT FILE NAME= NL:
DEFAULT FILE NAME= SPEECH.DAT
DO FRAMES= 10
TOTAL NUMBER OF BITS= 369
DO FRAME= 1
SN= 22.37 CSN= 22.37 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 2 SN= 13.77 CSN= 18.07 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 3 SN= 22.12 CSN= 19.42 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 4 SN= 16.59 CSN= 18.71 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 5 SN= 17.08 CSN= 18.39 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 6 SN= 15.99 CSN= 17.99 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 7 SN= 16.55 CSN= 17.78 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 8 SN= 19.24 CSN= 17.97 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 9 SN= 23.89 CSN= 18.62 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 10 SN= 23.21 CSN= 19.08 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 11 SN= 16.59 CSN= 18.86 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
DO FRAME=

```

FIGURE 2.5-2 EXAMPLE OPERATION OF THE ATC PROGRAM

```

DRUN ATC70
SHALL WE DO SERIALIZATION OF DATA(Y/N)?Y
SHALL WE INSEFT ERRORS IN CHANNEL(Y/N)?Y
ERROR RATE IN L11.4=
1.E-2
SHALL WE ENCCODE AND DECODE(Y/N)?Y
USE PITCH WEIGHTING(Y/N)? Y
USE FAST SORT(Y/N)? Y
SAMPLING RATE AND XMIT DATA RATE IN 2I6=6400,9600
IS THE INPUT ON MAG. TAPE? N
IS THE OUTPUT GOING TO MAG TAPE? N
OUTPUT FILE NAME= NL:
INPUT FILE NAME= SPEECH.DAT
NO FRAMES= 10
TOTAL NUMBER OF BITS= 369
INITIAL FRAME=5
FRAME= 5 SN= 22.12 CSN= 22.12 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 6 SN= 13.77 CSN= 17.94 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 7 SN= 22.25 CSN= 19.38 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 8 SN= 16.51 CSN= 18.66 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 9 SN= 17.39 CSN= 18.41 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 10 SN= 16.05 CSN= 18.02 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 11 SN= 16.55 CSN= 17.81 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 12 SN= 19.24 CSN= 17.99 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 13 SN= 23.89 CSN= 18.64 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 14 SN= 23.21 CSN= 19.10 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
FRAME= 15 SN= 16.59 CSN= 18.87 PITCH= 1 P.GAIN= 1.000 DCT BITS= 276
ATC70 DONE!

```

FIGURE 2.5-3 EXAMPLE OPERATION OF THE ATC PROGRAM

2.6 Summary and Conclusions

The ATC optimization studies resulted in an ATC system which does not degrade significantly with a BER of 10^{-2} at a data rate of 9600 b/s. The specifications for the optimized system are shown in Table 2.6-1. The actual quantization tables can be found in either the FORTRAN listing of Appendix D or in tables within section 3 of Volume 2 describing the real-time MAP software.

The voice quality produced by the 9600 b/s ATC simulations is the best of any technique now known to GTE. The technique, however, is numerically complex requiring the complete processing capability of the CSP, Inc. MAP-300 floating point processor. Thus, for ATC to be practical, either higher speed hardware must be built or the technique must be simplified.

Future speech digitization development at 9600 cannot ignore the ATC algorithm because even though the technique is complex, it shows that good quality speech is possible at this data rate. Thus, the ATC technique developed under this contract will serve as a benchmark or standard to compare all new 9600 b/s speech digitization algorithms.

<u>PARAMETER</u>	<u>SPECIFICATION</u>
Input Bandwidth	0-3200 Hz
Sampling Rate	6400 Hz
Frame Rate	26.016/sec.
Number of Samples/Frame	246
Number of Samples Overlapped/Frame	10
Bits/Frame	369
Pitch	{ 6 if voiced 0 if unvoiced
Pitch Gain	{ 2 if voiced 0 if unvoiced
Voiced/Unvoiced	1
RMS Energy	5
DC BIAS	5
PARCOR 1	5
PARCOR 2	5
PARCOR 3	4
PARCOR 4	4
PARCOR 5	3
PARCOR 6	3
PARCOR 7	2
PARCOR 8	2
Parity Bits (Error Correction)	54
SYNC	1
DCT Coefficients	{ 267 voiced 275 unvoiced
Number of Error Control Blocks/Frame	3
Error Control Technique	(63,45) BCH

TABLE 2.6-1: OPTIMIZED ATC SYSTEM SPECIFICATION

References:

- (1) R. Zelinsky and P. Noll, "Adaptive Transform Coding of Speech Signals", IEEE Trans. Acoust., Speech and Sign. Proc., Vol. ASSP-25, pp. 299-309, August 1977.
- (2) R. Zelinsky and P. Noll, "Approaches to Adaptive Transform Speech Coding at Low Bit Rates", IEEE Trans. Acoust., Speech and Sig. Proc., Vol. ASSP-27, pp. 89-95, February 1979.
- (3) J.M. Tribolet and R.E. Crochiere, "A Vocoder-Driven Adaptation Strategy for Low-Bit Rate Adaptive Transform Coding of Speech", Proc. 1978 Int. Conf. on Digital Signal Processing, Florence, Italy, pp. 638-642, September 1978.
- (4) J.M. Tribolet and R.E. Crochiere, "An Analysis/Synthesis Frame Work for Transform Coding of Speech", Proc. 1979 Int. Conf. on ASSP, Washington, D.C., pp. 81-84, April 1979.
- (5) B.S. Atal and S.L. Hanauer, "Speech Analysis and Synthesis by Linear Prediction of Speech Wave", J.A.S.A., Vol. 50, No. 2, 1971.
- (6) N. Ahmed, T. Natarajan, and K. Rao, "Discrete Cosine Transform", IEEE Trans. on Computers, Vol. C-23, pp. 90-93, 1974.
- (7) J. Cooley, P. Lewis, and P. Welch, "The Fast Fourier Transform Algorithm: Programming Considerations in the Calculation of Sine, Cosine, and Laplace Transforms", J. of Sound Vib., Vol. 12, pp. 315-337, July 1970.
- (8) M.J. Narasimha and A.M. Peterson, "On the Computation of the Discrete Cosine Transform", IEEE Trans. on Commun., Vol. COM-16, pp. 934-936, June 1978.
- (9) J. Mackhoul, "A Fast Cosine Transform in One and Two Dimensions," IEEE Trans. on Acoustics, Speech and Signal Processing, Vol. ASSP-28, No. 2, February 1980.

- (10) Bose, R. C. and D. K. Ray-Chandhuri, "On a Class of Error-Correcting Binary Group Codes," IEEE Trans. on Information and Control, Vol. IC-3, pp. 68-79, 1960.
- (11) Bose, R. C. and D. K. Ray-Chandhuri, "Further Results on Error-Correcting Binary Group Codes," IEEE Trans. on Information and Control, Vol. IC-3, pp. 279-290, 1960.
- (12) Hocquenghem, A., "Codes Correcteurs d'erreurs," Chiffres, Vol. 2, pp. 147-156, 1959.
- (13) J. D. Markel and A. H. Gray, Jr., "Linear Prediction of Speech," Springer-Verlay, N.Y., 1976.
- (14) L. R. Rabiner and R. W. Schafer, "Digital Processing of Speech Signals," Prentice-Hall Inc., New Jersey, 1978.
- (15) L. D. Davisson, "Rate-Distortion Theory and Application," Proc. IEEE, Vol. 60, pp. 800-808, 1972.
- (16) J. Huang and P. Schultheiss, "Block Quantization of Correlated Gaussian Random Variables," IEEE Trans. Communications Systems, Vol. CS-11, 1963, pp. 289-296.
- (17) J. Makhoul and M. Berouti, "Adaptive Noise Spectral Shaping and Entrophy Coding in Predictive Coding of Speech," IEEE Trans. on ASSP, Vol. ASSP-27, February 1979.
- (18) B. S. Atal and M. R. Schroeder, "Predictive Coding of Speech Signals and Subjective Error Criteria," IEEE Trans. on ASSP, Vol. ASSP-27, June 1979.
- (19) J. L. Flanagan, et al, "Speech Coding," IEEE Trans. on Commun., Vol. COM-27, No. 4, pp. 710-737, April 1979.

- (20) J. Max, "Quantizing for Minimum Distortion," IRE Transactions on Information Theory, Vol. IT-6, pp. 7-12, March 1960.
- (21) A. H. Gray, Jr., R. M. Gray, and J. D. Markel, "Comparison of Optimal Quantizations of Speech Reflection Coefficients," IEEE Trans. Acoust., Speech and Sig. Proc., Vol. ASSP-25, pp. 9-23, February 1977.

Appendix A Primitive BCH Codes

The BCH codes described in this appendix are cyclic codes that are well defined in terms of the roots of the generator polynomials [1]. These codes were discovered by Bose and Chaudhuri [2] - [3] and separately by Hocquenghem [4]. A binary (n,k) BCH code word consists of n symbols (bits in the binary case) where the first k bits are the information bits and the remaining $r = n-k$ bits are redundant parity checks. It is convenient to represent code words with polynomials as

$$f(x) = f_0 + f_1x + \dots + f_{n-1}x^{n-1}, \quad f_i = 0 \text{ or } 1 \quad (A1)$$

where each bit position is associated with a locator. If $f(x)$ is a code word, then

$$f_1(x) = f_1 + f_2x + \dots + f_{n-1}x^{n-2} + f_0x^{n-1} \quad (A2)$$

is also a codeword in a cyclic codes. In the primitive BCH code, which is the most convenient and powerful BCH code in theory and practice, the block length of the code may be defined as

$$n = 2^m - 1 \quad (A3)$$

and with mt parity checks, it can correct any set of t independent errors within the block of n bits, where m and t are arbitrary positive integers [5]. This code may be described conveniently with the aid of finite Galois field theory introduced in Appendix B.

Let α be a primitive element of the finite field $GF(2^m)$, then the primitive BCH code may be described as the set of polynomials such that

$$f(\alpha^i) = 0, i = 1, 3, 5, \dots, 2t - 1 \quad (A4)$$

It is known in coding theory that these polynomials consist of all multiples of a single polynomial $g(x)$, known as the generator polynomial. This polynomial also satisfies the equations as

$$g(\alpha^i) = 0, i = 1, 3, 5, \dots, 2t - 1 \quad (A5)$$

These generator polynomials are tabulated in Table A for the selected primitive BCH codes.

Encoding Procedures

Let the k information bits be represented by the polynomial $d(x)$ as

$$d(x) = \sum_{i=0}^{k-1} d_i x^i \quad (A6)$$

then, the code word of n bits may be expressed as

$$f(x) = x^{n-k} d(x) + r(x) \quad (A7)$$

where $r(x)$ is the remainder (parity check) obtained according to the following equation:

$$\frac{x^{n-k} d(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)} \quad (A8)$$

Block length n	k	t	Generator Polynomial
63	57	1	$g_1(x) = (6, 1, 0) = x^6 + x + 1$
	51	2	$g_3(x) = g_1(x) \cdot (6, 4, 2, 1, 0)$
	45	3	$g_5(x) = g_3(x) \cdot (6, 4, 2, 1, 0)$
127	120	1	$g_1(x) = (7, 3, 0) = x^7 + x^3 + 1$
	113	2	$g_3(x) = g_1(x) \cdot (7, 3, 2, 1, 0)$
	106	3	$g_5(x) = g_3(x) \cdot (7, 4, 3, 2, 0)$
255	247	1	$g_1(x) = (8, 4, 3, 2, 0) = x^8 + x^4 + x^3 + x^2 + 1$
	239	2	$g_3(x) = g_1(x) \cdot (8, 6, 5, 4, 2, 1, 0)$
	231	3	$g_5(x) = g_3(x) \cdot (8, 7, 6, 5, 4, 2, 0)$

TABLE A: GENERATOR POLYNOMIALS FOR SELECTED PRIMITIVE BCH CODES

where $g(x)$ is the generator polynomial of the code. Therefore, encoding can be performed by the following procedures:

- 1). Calculate $x^{n-k} d(x)$ by left shifting the information bits $n-k$ times
- 2). Calculate the remainder (parity bits) $r(x)$ from the division of $x^{n-k} d(x)$ by $g(x)$
- 3). Add the polynomial $x^{n-k} d(x)$ and $r(x)$ to form the code word

The procedures of 1) and 3) can be done simply by shifting and addition. However, the procedure of 2) is rather involved in computation if the actual division is performed to get the remainder. If the BCH code is specified and it is desired to speed up the processing time of 2), it is recommended to use a look-up table procedure for the calculation of the remainder from 2). The code word is then transmitted through the noisy channel, where the received code word may be altered depending on the introduction of channel errors.

Decoding Procedures

There are several algorithms for a decoding of BCH codes. Efficient decoding algorithms have been discovered for BCH codes [1] - [7]. The Berlekamp decoder is particularly attractive for powerful codes that provide for a good deal of error corrections (e.g., 10 or more). The Peterson algorithm, however, is more efficient for less powerful codes (e.g., the codes used in generalized burst trapping). In this decoding procedure, the problem of finding efficient solutions to the key decoding equation will be addressed by using the Peterson technique.

When a BCH code word $\{f(x)\}$ is transmitted over a noisy channel, this code word may be corrupted by the channel, and what is received $\{\gamma(x)\}$ can be different from the intended code word. Thus, the received word may be expressed as

$$\gamma(x) = f(x) + e(x) \quad (A9)$$

where $e(x)$ is the error polynomial which a decoder must compute to correct errors introduced by the channel. Let the received data be expressed in vector γ as

$$\gamma = [\gamma_0, \gamma_1, \dots, \gamma_{n-1}] \quad (A10)$$

or its associated polynomial $\gamma(x)$ by

$$\gamma(x) = \gamma_0 + \gamma_1 x + \dots + \gamma_{n-1} x^{n-1} \quad (A11)$$

Denote each of the error location numbers by β_j , $j = 1, 2, \dots, t$, then it is shown [1] that the power sums S_i can be expressed as

$$\begin{aligned} S_i &= \gamma(\alpha^i) \\ &= \sum_{j=1}^t \beta_j^i, \quad i = 1, 3, 5, \dots, 2t-1 \end{aligned} \quad (A12)$$

In order to find the error locations, the Peterson procedures consist of three steps:

Step 1: Compute the power sums S_i from the received sequence through the relations

$$S_i = \gamma(\alpha^i), \quad i = 1, 3, 5, \dots, 2t-1 \quad (A13)$$

$$S_{2i} = S_i^2$$

Step 2: Compute the symmetric functions σ_k , $k = 1, 2, \dots, t$ from the power sums S_i , i.e.,

$$\begin{aligned} \sigma(x) &= x^t + \sigma_1 x^{t-1} + \dots + \sigma_{t-1} x + \sigma_t \\ &= (x + \beta_1)(x + \beta_2) \dots (x + \beta_t) \end{aligned} \quad (A14)$$

and the σ_k 's may be obtained by the use of Newton's identities [1]

$$\underline{\sigma} = \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \vdots \\ \sigma_t \end{bmatrix} \quad (A15)$$

$$\begin{aligned} &= M_t^{-1} \underline{S} \\ &= \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ S_2 & S_1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & \dots & S_{t-1} \end{bmatrix}^{-1} \begin{bmatrix} S_1 \\ S_3 \\ \vdots \\ S_{2t-1} \end{bmatrix} \end{aligned}$$

If the determinant of M_t is singular, then reduce the error number t by 2 and proceed with it again.

Step 3: Find the error position locator β_j , $j = 1, 2, \dots, t$, which is the roots of the polynomial $\sigma(x)$ in eq. (A14).

An efficient algorithm for calculating the β_j 's from eq. (A14) has been developed by Chien [5], and all that remains to completely specify a binary BCH decoder is the computation of the coefficients of error locator polynomial, σ_j 's. As it is noted from eq. (A15), the calculation of the σ_j 's involved matrix inversion which can be expressed analytically for the case $t \leq 3$. The results are:

For $t = 1$,

$$\sigma_1 = S_1$$

For $t = 2$,

$$\sigma_1 = S_1$$

$$\sigma_2 = (S_3 + S_1^3)/S_1$$

For $t = 3$,

$$\sigma_1 = S_1$$

$$\sigma_2 = (S_1^2 S_3 + S_5)/(S_1^3 + S_3)$$

$$\sigma_3 = (S_1^3 + S_3) + S_1 \sigma_2$$

The calculation of the σ_i 's and the estimation of the error number are shown in Figure A1 for $t = 3$. The flowchart of Chien's search decoding

$$m_1(\alpha) = \alpha^6 + \alpha + 1$$

$$= 0$$

$$= \alpha^{6^3} + 1$$

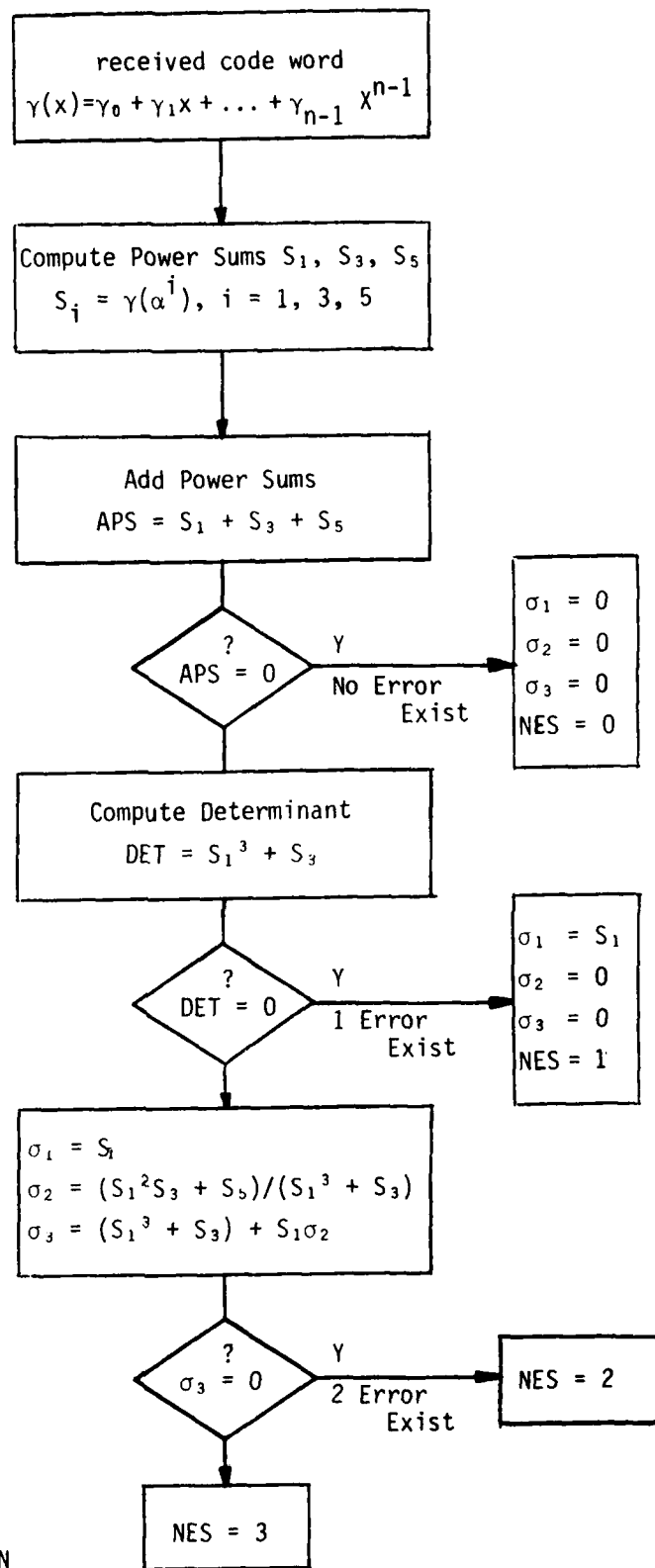


FIGURE A1: COMPUTATION
OF $\sigma_1, \sigma_2, \sigma_3$

3 or More Error Exist

procedure is shown in Figure A2. This flowchart is for $t = 3$, i.e., the decoding algorithm can correct errors up to 3. One interesting observation in this decoding procedure is that the correction of errors may be performed erroneously if the number of errors in the block is greater than 3. Hence, the corrections may introduce additional channel errors. In order to avoid these additional errors, error corrections are made only when the estimated error number (NES in Figure A1) equals to the measured error number (K in Figure A2). This procedure eliminates most of the additional errors when more than 3 errors exist in the received word. In other words, the detection of errors more than 3 (i.e., 4, 5, 6, ..., etc.) is feasible most of the cases. This fact contributes some improvements of the coder performance when the channel is very noisy (bit error rate $\approx 10^{-2}$).

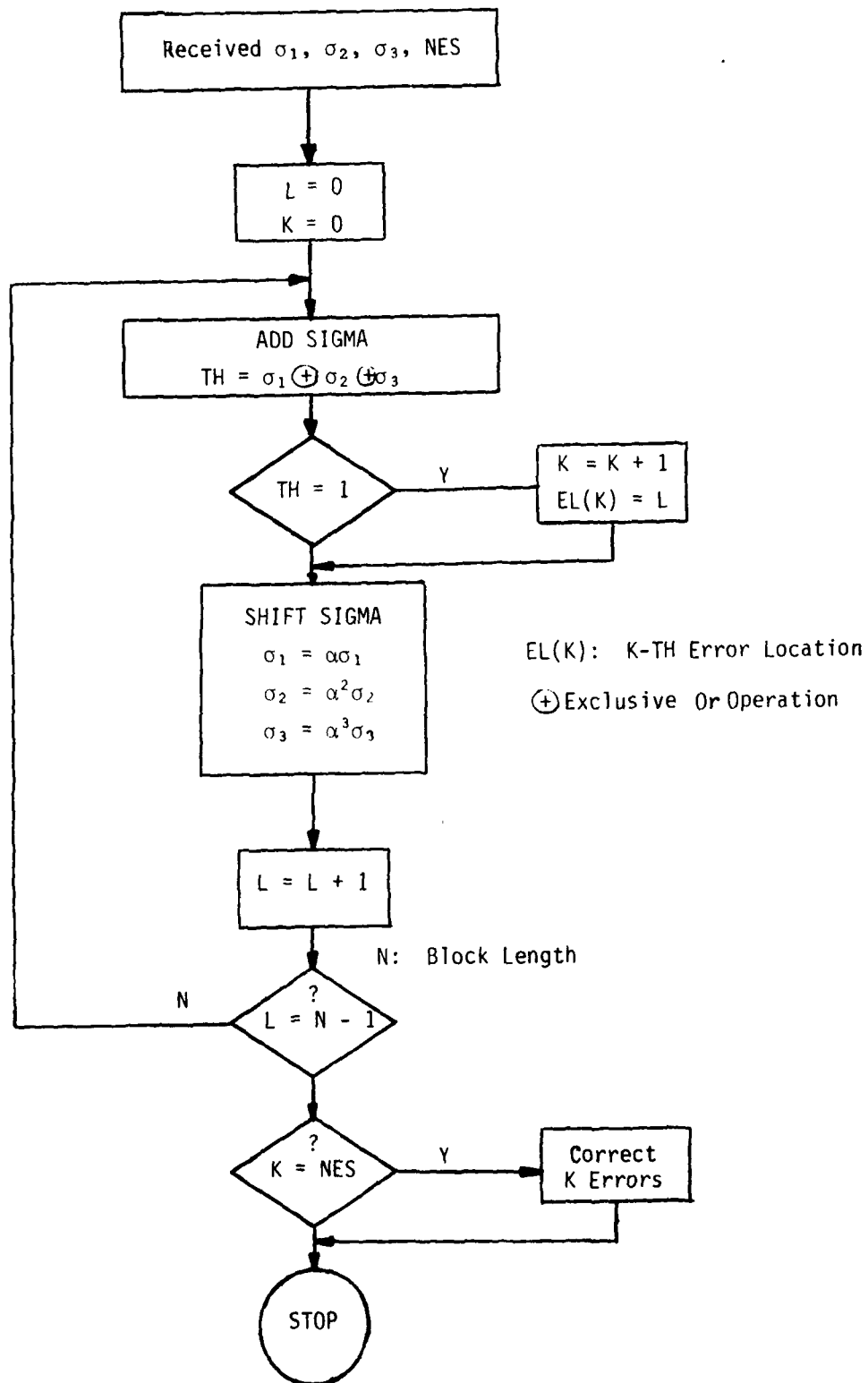


FIGURE A2: CHIEN'S SEARCH DECODING PROCEDURE

References for Appendix A

- [1] Peterson, W. W., Error-Correcting Codes, The MIT Press, Cambridge, MA, 1961.
- [2] Bose, R. C. and D. K. Ray-Chandhuri, "On a Class of Error-Correcting Binary Group Codes," IEEE Trans. on Information and Control, Vol. IC-3, pp. 68-79, 1960.
- [3] Bose, R. C. and D. K. Ray-Chandhuri, "Further Results on Error-Correcting Binary Group Codes," IEEE Trans. on Information and Control, Vol. IC-3, pp. 279-290, 1960.
- [4] Hocquenghem, A., "Codes Correcteurs d'erreurs," Chiffres, Vol. 2, pp. 147-156, 1959.
- [5] R. T. Chien, "Cyclic Decoding Procedures for Bose-Chandhuri-Hocquenghem Codes," IEEE Trans. on Information Theory, Vol. IT-10, pp. 357-363, 1964.
- [6] E. R. Berlekamp, Algebraic Coding Theory, New York, McGraw-Hill, 1968.
- [7] J. L. Massey, "Shift-Register Synthesis and BCH Decoding," IEEE Trans. on Information Theory, Vol. IT-15, No. 1, pp. 122-127, 1969.

Appendix B Operations in Galois Field

A Galois field is a finite set of elements that satisfy the axioms of a general field. Two operations (addition and multiplication) and their inverses are defined on the field elements. There is an identity element for each field element for both of the operations (0, 1) that is itself in the field. Also, both addition inverses and multiplication inverses are in the field. Finally, the rules of commutation and associativity are obeyed by the elements of the field.

Consider the following sixteen polynomials and their vector binary representations.

0	0000
1	0001
$1 + X$	0011
$1 + X + X^2$	0111
$1 + X + X^2 + X^3$	1111
X	0010
$X + X^2$	0110
$X + X^2 + X^3$	1110
X^2	0100
$X^2 + X^3$	1100
X^3	1000
$1 + X^3$	1001
$1 + X^2$	0101
$1 + X^2 + X^3$	1101
$X + X^3$	1010
$1 + X + X^3$	1011

As long as addition and multiplication of these polynomials is defined so that the axioms for the field are obeyed, then this will, in fact, be a Galois field of 2^4 elements ($GF(2^4)$).

Addition is defined to be modulo 2. Each element is its own additive inverse and addition and subtraction of elements are the same.

Multiplication must be defined so that the product of two elements does not take us out of the field. For this reason, multiplication in a Galois field is not ordinary multiplication of polynomials. Rather, multiplication is defined modulo an irreducible polynomial, the primitive polynomial of the Galois field. For our field $GF(2^4)$, the primitive polynomial is $1 + X + X^4$. To generate the 16 vectors in the field, all one needs to do is to divide X^m where $m = 0, 1, \dots, 14$ by the primitive polynomial.

0	-1		
1	0		
X	X		
X ²	X ²		
X ³	X ³		
X ⁴	1 + X	$1 + X + X^4 \mid X^4$	$\frac{1}{R[1 + X]}$
X ⁵	X + X ²		
X ⁶	X ² + X ³	$1 + X + X^4 \mid X + X^2$	$\frac{X}{R[X + X^2]}$
X ⁷	X ³ + X + 1		
X ⁸	X ² + 1		
X ⁹	X ³ + X		
X ¹⁰	X ² + X + 1		
X ¹¹	X ³ + X ² + 1		
X ¹²	X ³ + X ² + X + 1		
X ¹³	X ³ + X ² + 1		
X ¹⁴	X ³ + 1		
X ¹⁵	X ⁰ + 1		

It is now seen that the product of two binary vectors in the field is just the sum of their powers. The table repeats every fifteen powers so it is all done modulo 15.

$$\chi^i + \chi^j = \chi^{i+j} \pmod{15}$$

$$\frac{\chi^i}{\chi^j} = \chi^{i-j} \pmod{15}$$

APPENDIX C

FORTRAN Source Listings for the ATC Simulation

This appendix contains the FORTRAN source programs for the ATC simulation. The first page of this listing is a compile file which uses FORTRAN-IV PLUS to generate object files from the source files and which sends listings to the line printer. The second page of these programs is the overlay description language (ODL) needed to build the ATC task under the RSX-11M operating system. The remainder of the appendix includes the main program and subroutines. The order of the programs follows the order of the files as listed in the overlay description language on the second page.

PIP ATCOLA.CMD: X/PU
PIP BUILD.CMD: X/PU
PIP ATC70.ODL: X/PU
PIP ATC70.OBJ: X/DE
PIP OUR1.OBJ: X/DE
PIP OUR3.OBJ: X/DE
PIP TAPE3.OBJ: X/DE
PIP OUR3.OBJ: X/DE
PIP OUR4.OBJ: X/DE
PIP OUR5.OBJ: X/DE
PIP OUR6.OBJ: X/DE
PIP OUR7.OBJ: X/DE
PIP OUR8.OBJ: X/DE
PIP OUR9.OBJ: X/DE
PIP OUR10.OBJ: X/DE
PIP OUR11.OBJ: X/DE
PIP OUR12.OBJ: X/DE
PIP OUR13.OBJ: X/DE
PIP QUEVOD.OBJ: X/DE
PIP FASTF.OBJ: X/DE
PIP GF2POL.OBJ: X/DE
PIP GENTAB.OBJ: X/DE
PIP LOOKUP.OBJ: X/DE
PIP INALOK.OBJ: X/DE
PIP CHARTS.OBJ: X/DE
PIP GF2D1U.OBJ: X/DE
PIP GF2MUL.OBJ: X/DE
PIP GF2ADD.OBJ: X/DE
PIP ATCOLA.CMD/SP
PIP BUILD.CMD/SP
PIP ATC70.ODL/SP
F4P ATC70.ATC70-ATC70/DE
F4P TAPE3.TAPE3-TAPE3
F4P QUEVOD.QUEVOD-QUEVOD
F4P FASTF.FASTF-FASTF
F4P OUR1.OUR1-OUR1
F4P OUR2.OUR2-OUR2
F4P OUR3.OUR3-OUR3
F4P OUR4.OUR4-OUR4
F4P OUR5.OUR5-OUR5
F4P OUR6.OUR6-OUR6
F4P OUR7.OUR7-OUR7
F4P OUR8.OUR8-OUR8
F4P OUR9.OUR9-OUR9
F4P OUR10.OUR10-OUR10
F4P GF2POL.GF2POL-GF2POL
F4P GENTAB.GENTAB-GENTAB
F4P LOOKUP.LOOKUP-LOOKUP
F4P INALOK.INALOK-INALOK
F4P CHARTS.CHARTS-CHARTS
F4P GF2D1U.GF2D1U-GF2D1U
F4P GF2MUL.GF2MUL-GF2MUL
F4P GF2ADD.GF2ADD-GF2ADD
F4P OUR11.OUR11-OUR11
F4P OUR12.OUR12-OUR12
F4P OUR13.OUR13-OUR13
-4P

```

      NAME ENTAB
      Q=(S,U,V,W,X,Y,Z
      .
      .FCIR QUR3
      .FCIR QUR4
      .FCIR QUR5
      .FCIR QUR6
      .FCIR QUR7
      .FCIR QUR8
      .FCIR QUR9
      .FCIR QUR10-GF2POL-LOOKUP-INULOK-CHARTS-*(GENTAB)
      .FCIR QUR11
      .FCIR QUR12
      .FCIR QUR13
      .FCIR CNTRL1-*(QA,GF2ADD,BB,TAPE3)
      .FCIR QUEUGD-*(FASTF)
      .FCIR GF2MUL-*(GF2DIV)
      .END

```

```

J:
K:
L:
M:
N:
O:
P:
Q:
R:
S:
T:
U:
V:
W:
X:
Y:
Z:
AA:
BB:

```


AD-A091 662

GTE PRODUCTS CORP. NEEDHAM HEIGHTS MA COMMUNICATION S--ETC F/G 5/8
SPEECH OPTIMIZATION AT 9600 BITS/SECOND. VOLUME 1. SOFTWARE SIM--ETC(U)
SEP 80 A J GOLDBERG, L COSELL, S KWON DCA100-76-C-0064

UNCLASSIFIED

NL

2 of 2

AD

Approved

END
DATE
FILMED
12 80
DTIC

1 0.16391E-01.0.33470E-01.0.51313E-01.0.69970E-01.
1 0.89376E-01.0.11219E+00.0.13700E+00.0.16288E+00.
1 0.15972E+00.0.21809E+00.0.25131E+00.0.28606E+00.
1 0.32430E+00.0.36803E+00.0.41824E+00.0.47689E+00.
1 0.54371E+00.0.61470E+00.0.69616E+00.0.77133E+00.
1 0.84345E+00.0.92970E+00.0.10190E+01.0.11218E+01.
1 0.12403E+01.0.13426E+01.0.14432E+01.0.15411E+01.
1 0.16443E+01.0.17866E+01.0.19332E+01.0.1E+21.
1 0.15141E+00.0.31624E+00.0.40542E+00.0.50895E+00.
1 0.59219E+00.0.65272E+00.0.70486E+00.0.75395E+00.
1 0.81878E+00.0.87625E+00.0.94152E+00.0.10123E+01.
1 0.10847E+01.0.11630E+01.0.12381E+01.0.13059E+01.
1 0.13727E+01.0.14357E+01.0.14930E+01.0.15468E+01.
1 0.16017E+01.0.16527E+01.0.16951E+01.0.17323E+01.
1 0.17683E+01.0.18035E+01.0.18365E+01.0.18668E+01.
1 0.18951E+01.0.19267E+01.0.19616E+01.0.1E+21.
1 0.31538E+00.0.45224E+00.0.56230E+00.0.65753E+00.
1 0.74494E+00.0.82867E+00.0.91130E+00.0.99458E+00.
1 0.10795E+01.0.11578E+01.0.12608E+01.0.13531E+01.
1 0.14519E+01.0.15544E+01.0.17491E+01.0.1E+21.16X0.0.
1 0.40927E+00.0.57217E+00.0.69580E+00.0.80120E+00.
1 0.89689E+00.0.98763E+00.0.10764E+01.0.11652E+01.
1 0.12547E+01.0.13464E+01.0.14408E+01.0.15388E+01.
1 0.16420E+01.0.17540E+01.0.18155E+01.0.1E+21.16X0.0.
1 0.55688E+00.0.74670E+00.0.89149E+00.0.10220E+01.
1 0.11526E+01.0.12967E+01.0.14823E+01.0.1E+21.24X0.0.
1 0.57331E+00.0.75317E+00.0.90932E+00.0.10407E+01.
1 0.11725E+01.0.13190E+01.0.15053E+01.0.1E+21.24X0.0.
1 0.56750E+00.0.87517E+00.0.11666E+01.0.1E+21.28X0.0.
1 0.72199E+00.0.98497E+00.0.12430E+01.0.1E+21.28X0.0/
PARCOR(J) DELQUANTIZER DECISIONS
DATA PDEL/
1 0.80431E-02.0.24740E-01.0.42199E-01.0.60427E-01.
1 0.79512E-01.0.10024E+00.0.12414E+00.0.14986E+00.
1 0.17599E+00.0.20368E+00.0.23451E+00.0.26811E+00.
1 0.30401E+00.0.34440E+00.0.39157E+00.0.44480E+00.
1 0.50698E+00.0.57740E+00.0.65201E+00.0.74030E+00.
1 0.80236E+00.0.88455E+00.0.97486E+00.0.10631E+01.
1 0.11805E+01.0.13001E+01.0.13851E+01.0.15023E+01.
1 0.15805E+01.0.17030E+01.0.18651E+01.0.19972E+01.
1 0.87097E-01.0.26319E+00.0.36928E+00.0.46156E+00.
1 0.55635E+00.0.62403E+00.0.68141E+00.0.73631E+00.
1 0.79160E+00.0.84537E+00.0.90653E+00.0.97650E+00.
1 0.10480E+01.0.11214E+01.0.12027E+01.0.12736E+01.
1 0.13833E+01.0.14072E+01.0.14643E+01.0.15197E+01.
1 0.15740E+01.0.16295E+01.0.16759E+01.0.17142E+01.
1 0.17504E+01.0.17683E+01.0.18208E+01.0.18521E+01.
1 0.18814E+01.0.19108E+01.0.19426E+01.0.19807E+01.
1 0.23813E+00.0.39243E+00.0.51204E+00.0.61256E+00.
1 0.70251E+00.0.70737E+00.0.86938E+00.0.95262E+00.
1 0.10365E+01.0.11236E+01.0.12129E+01.0.13085E+01.
1 0.14075E+01.0.15162E+01.0.16545E+01.0.18436E+01.15X0.0.
1 0.31411E+00.0.50442E+00.0.63952E+00.0.75167E+00.
1 0.85074E+00.0.94305E+00.0.10332E+01.0.11205E+01.
1 0.12094E+01.0.13000E+01.0.13239E+01.0.14888E+01.
1 0.15880E+01.0.16951E+01.0.18129E+01.0.19500E+01.15X0.0.
1 0.44510E+00.0.66865E+00.0.82476E+00.0.95821E+00.
1 0.10859E+01.0.12194E+01.0.13739E+01.0.15907E+01.24X0.0.
1 0.46200E+00.0.58453E+00.0.84172E+00.0.97631E+00.
1 0.11550E+01.0.12400E+01.0.13900E+01.0.15206E+01.24X0.0.
1 0.39739E+00.0.73761E+00.0.10127E+01.0.13204E+01.28X0.0.
1 0.57866E+00.0.86541E+00.0.11045E+01.0.13815E+01.28X0.0/
DATA DCTHR/
1 0.12475E+00.0.25019E+00.0.37706E+00.0.50625E+00.
1 0.63881E+00.0.77594E+00.0.91907E+00.0.10699E+01.

SECRET

```

<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
NOW QUANTIZATION OF DECIMAL DATA IS DECIDED.
DECODING AND ENCODING, AND CHANNEL ERROR INSERTION RATE
FOR LINE SIMULATION

```

```

7302 WRITE(S,7302)
7303 FORMAT(1H,'SHALL WE DO SERIALIZATION OF DATA(Y,N)?')
7304 READ(S,2601,END=999,ERR=999)SER
7305 IF(SER.EQ.NO)GO TO 7307
7306 IF(SER.EQ.YES)GO TO 7307
7307 WRITE(S,7313)
7308 FORMAT(1H,'SHALL WE INSERT ERRORS IN CHANNEL(Y,N)?')
7309 READ(S,2602,END=999,ERR=999)ERRDEC
7310 IF(ERRDEC.EQ.NO)GO TO 7303
7311 WRITE(S,7300)
7312 FORMAT(1X,'ERROR RATE IN E11.4=')
7313 READ(S,7301)PREA
7314 CONTINUE
7315 WRITE(S,7304)
7316 FORMAT(1H,'SHALL WE ENCODE AND DECODE(Y,N)?')
7317 READ(S,2602,END=999,ERR=999)ENCODEC
7318 IF(ENCODEC.EQ.NO)GO TO 7303
7319 WRITE(E11.4)
7320 *****
7321 OUR2 WILL SET UP FRAME CONSTANTS, SAMPLING RATE AND BIT ALLOCATIONS
7322 CALL OUR2
7323 *****
7324 BEGIN ANALYSIS/SYNTHESIS
7325 *****
7326 TYPE 808,NBRPF
7327 BTLTH=BT.LTH
7328 FORMAT(1X,'TOTAL NUMBER OF BITS=',2X,14)
7329 LP3=LPCM+1
7330 IURF=0
7331 ICUSUB=0
7332 ICOUNT=0
7333 DO 1002 I=1,NP
7334 YY(I)=0.0
7335 WRITE(S,2005)
7336 FORMAT(1H,'INITIAL FRAME=')
7337 READ(S,1101)INITFR
7338 FORMAT(16)
7339 NOW REWIND THE TAPE
7340 IST=1
7341 NSKIP=1
7342 CALL TAPE3(6)
7343 *****
7344 INITIALIZE RANDOM NUMBER GENERATOR
7345 IPN=0
7346 JRN=0
7347 DO 1003 I=1,49
7348 CALL RANDU(IPN,JRN,YDR)
7349 *****
7350 CONTINUE
7351 BTLTH=BT.LTH
7352 ICOUNT=ICOUNT+1
7353 IF(ICOUNT.LT.16)GO TO 67
7354 CALL CLOSE(4)
7355 CALL ASSIGN(4,'GDS')
7356 PAUSE 'CHANGE IN LUN4 IS MADE, "RESUME"'
7357 ICOUNT=ICOUNT

```

[illegible]

```

9920 WRITE(4,922)(1,COS(ARGX(I-1)),I=1,LTH)
9921 FORMAT(1X,4(1X,'13.0'),E15.8,2X))
9922 FORMAT(1X,4(1X,'13.0'),E15.8,2X))
9923 FORMAT(1X,4(1X,'13.0'),E15.8,2X))
9902 WRITE(4,902)(1,DC1(I),I=1,LTH)
      FORMAT(1X,4(1X,'13.0'),E15.8,2X))

      COMPUTE PSEUDO AUTOCORRELATION FUNCTION

      FIRST DO EVEN ODD REFLECTION AROUND SYMMETRIC SIGNAL
      AND SCALE BY 2
      DCT1(1)=XR(1)/2.0
      DCT2(1)=0.0
      XI(1)=XR(2)/2.0
      XI(LTH+1)=XR(LTH2)/2.0
      MLTH2=LTH2+2
      DO 90 I=2,LTH
        J=2*1-1
        DCT1(1)=XR(J)/2.0
        DCT2(1)=XR(MLTH2-J)/2.0
        XI(1)=XR(J+1)/2.0
        XI(LTH+1)=XR(MLTH2-J-1)/2.0
      DO 91 I=1,LTH
        XR(I+LTH)=DCT1(I)
        XR(I)=DCT1(I)
      NOTE DCT1 AND DCT2 ARE JUST TEMPORARY SHUFFLE VECTORS
      WRITE(4,903)(1,XR(I),I=1,LTH)
      WRITE(4,921)(1,XI(I),I=1,2*MLTH)
      CALL QUBOD(LTH2,2,2)
      CALL EUBOD(LTH2,XP,XI,2,2)
      WRITE(4,920)(1,XR(I),I=1,LTH)
      WRITE(4,921)(1,XI(I),I=1,LTH)
      SAVE PSEUDOCORRELATION

      DO 625 I=1,LP1
        R(I)=XR(I)
      WRITE(4,903)(1,R(I),I=1,LP1)
      FORMAT(1X,5(1X,'12.0'),E15.8,2X))

      FIND THE LARGEST PSEUDOCORRELATION
      XLTH2=LTH2
      XLARGE=0.
      M=1
      DO 95 I=LP1+1,LTH
        IF(XR(I).LT.XLARGE)GO TO 95
        M=I
        XLARGE=XR(I)
      CONTINUE
      15 AND 94 ARE THE SMALLEST AND LARGEST ALLOWABLE
      VALUES INTO THE PITCH QUANTIZER ROUTINES
      IF(M.LT.15)M=15
      IF(M.GT.94)M=94
      IF(PHASE.EQ.NO)IPDEC=0
      IF(IPDEC.EQ.0)M=1
      IF(M-1 THEN THE FRAME IS NOT PITCH WEIGHTED
      THEREFORE THE 6 BITS FOR PITCH AND 2 BITS FOR
      PITCH MAGNITUDE CAN BE ADDED TO DCT BIT ALLOCATION
      IF(M.EQ.1)BTLTH=BTLTH+8

      GET PITCH GAIN

```



```

QTG=I-1
QUANTIZE PITCH,M
QUANTIZE THE PITCH IN 6 BITS OR 64 LEVELS
WITH THE HIGHEST VALUE BEING 94 AND LOWEST 15
IBPT=0
IF(IPDEC.EQ.0)GO TO 87
ITR=M-15
IBPT=ITK
IF(ITR.LE.47)GO TO 9191
IBPT=(ITR-46)/2
IBPT=47+ITR1
ITR=(ITR/2)*2
CONTINUE
DTM=ITR+15
GTM=IBPT
WRITE(4,927)GTM,QTG
FORMAT(1X,'GTM=',16,' AND QTG=',16)
DEQUANTIZE PARCOR(J),J=1,...LPON,G,M
DEQUANTIZE PARCOR(J)
DO 751 J=1,LPON
K=(J-1)*32+1
I=OTPAR(J)+K
REMOVE BIAS
DTPAR(J)=PDPC(I)+1,0
WRITE(4,928)(J,DTPAR(J),J=1,LPON)
FORMAT(1X,4(1X,' DTPAR(',12,')=' ,E15.8,2X))
DEQUANTIZE PITCH GAIN,G
I=GTG+1
DTG=PDPC(I)
EXPAND THE PITCH GAIN
DTG=DTGX2
DEQUANTIZE PITCH,M
IF(IPDEC.EQ.0)DTM=1
CONTINUE
WRITE(4,929)DTM,DTG
FORMAT(1X,'DTM=',E15.8,' AND DTG=',E15.8)
RECREATE LP COEFFICIENTS
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CALL OUR4
CALL PARPRE(LPON,DTPAR,DTA)
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
WRITE(4,931)(J,DTRAJ),J=1,LPON)
FORMAT(1X,4(1X,' DTRA(',12,')=' ,E15.8,2X))
REGENERATE RESIDUAL ENERGY
U=1.-DTPAR(1)*X2
DO 870 I=2,LPON
U=U*(1.-DTPAR(1))*2)
ENG=SQRT((2.*XINSTAGE)*XU)
WRITE(4,932)ENG
WRITE(5,6000)I
FORMAT(1X,16)
FORMULATE LPC BASIS SPECTRUM
XR(1)=1.
XI(1)=0.
DO 628 I=1,LPON
XR(I+1)=DTAL(I)
DO 629 I=1,LPON,XI(I)

```

```

XZ(1)=0.
FORALL THE PITCH WEIGHTING SPECTRUM
NOTICE WE ARE CONSTRUCTING A ZERO DC
PITCHED SIGNAL TO USE IN THE FREQUENCY DOMAIN

DTM=DTM-1
DO 98 I=1,LTH2
XI(I)=0.
EGR=0.
K1=LTH/IFIX(DTM)
DCPIT=0.
DO 99 K=1,100
J=(K-1)*DTM+1
IF(J.GT.LTH)GO TO 1020
TEMP=DTGX(K-1)
DCPIT=DCPIT+TEMP
XI(J)=TEMP
DCPIT=DCPIT-FLOAT(LTH)
DO 9797 I=1,LTH
XI(I)=XI(I)-DCPIT
EGR=EGR+XI(I)**2
CONTINUE
EGR=1./SQRT(EGR)
WRITE(4,91)'K,EGR
FORMAT('I,X',K=' ',I3,' AND EGR=',E15.8)
OVERLAY 5 DETERMS THE MAGNITUDE SPECTRUM OF THE TWO REAL SIGNALS XR,XI
WHICH ARE BOTH LOADED INTO ONE FFT
CALL FOURFAG(XR,XI,NSTAGE,LTH2)
CALL OURS

<<<<<<<<<<<<<<<<<<<<<<<<<<<<
DO 630 I=1,LTH
DCT1(I)=ENG/XR(I)
WRITE(4,907)(1,DCT1(I),I=1,LTH)
FORMAT('I,X,4(IX','DCT1(',I3,',',E15.8,2X))
PIWT=1 IS USED TO APPROPRIATELY ADJUST
THE FREQUENCY DOMAIN PITCH WEIGHTING FUNCTION.
IT LOOPASS FILTERS IT FOR FREQUENCIES BELOW
1.5KHZ,SIMULATING A POLE IN THE DRIVING FUNCTION
IT ALSO UNCORRELATES THE FREQUENCIES SIMULATING
COMPONENTS FOR THE HIGH FREQUENCIES SIMULATING
A WHITE NOISE STIMULUS IN THE HIGH FREQUENCY PART OF THE SPECTRUM.
IF THE FRAME IS UNPITCHED THEN JUMP AROUND IT.
IF(DTH.EQ.0)GO TO 1009
CALL PIWT(EGR,LTH,XI,PWEIT)
GO TO 1011
DO 1010 I=1,LTH
PWEIT(I)=EGRX(I)
PWEIT(I)=1.
CONTINUE
WRITE(4,905)(1,PWEIT(I),I=1,LTH)
FORMAT('I,X,4(IX,'PWEIT(',I3,',',E15.8,2X))

COMPLETE BASIS SPECTRUM

DO 102 I=1,LTH
DCT1(I)=DCT1(I)*PWEIT(I)
WRITE(4,908)(1,DCT1(I),I=1,LTH)
FORMAT('I,X,4(IX,'DCT1(',I3,',',E15.8,2X))
DCT1(I)=1.E-7

ORDER DCT MAX-TO-MIN

```

```

<<<<<<<<<<<<<<<<<<<<<<
OVERLAYS 6,7 ARE JUST SUPTING ALGORITHMS TO SORT THE
DCT COEFFICIENTS INTO THEIR DESCENDING ORDER
IF(TYPSRT.EQ.YES)CALL OVER
IF(TYPSRT.EQ.NO)CALL OVR?
<<<<<<<<<<<<<<<<<<<<<<
WRITE(4,909)((I,DCT2(I),I=1,LTH)
FORMAT('1X,4(1X,'DT2(',13,')=',E15.8,2X))
WRITE(4,910)((I,IORDR(I),I=1,LTH)
FORMAT('1X,6(1X,'IORDR(',13,')=',13,2X))
COMPRESS BASIS SPECTRUM

DO 4 I=1,LTH
IF(DCT2(I).EQ.0)DCT2(I)=10E-10
DCT2(I)=ALOG(DCT2(I))/DLG2
WRITE(4,909)(I,DCT2(I),I=1,LTH)
INITIAL DCT BIT ASSIGNMENT

TYPE 603,BTLTH
FORMAT(IX,'NUMBER OF BITS FOR DCT-',IX,F11.4)
SLOG=0.
DO 104 I=1,LTH
SLOG=SLOG+DCT2(I)
CLEAR OUT BIT-ASSIGNMENT ARRAY
DO 105 I=1,LTH
IBIT(I)=0
CTEMP=(BTLTH-SLOG)/LTH
DO 105O I=1,LTH
BITS=CTEMP+DCT2(I)
IF(BITS.LE.0.0)GO TO 1051
LAST-I
!TOT-LAST
WRITE(4,930)BITS,CTEMP,!TOT
FORMAT(/IX,'BITS=',E15.8,' \ CTMP=',E15.8,' * !TOT=',13)
SLOG=0.0
FORM SUM-OF LOGS UP TO LAST SAMPLE
DO 109 I=1,!TOT
SLOG=SLOG+DCT2(I)
CTEMP=(BTLTH-SLOG)/!TOT
TOTAL ASSIGNAABLE BITS
IBITSL=BTLTH
DO 110 I=1,!TOT
K=I
BITS=CTEMP+DCT2(I)
ROUND UPWARDS
IF(BITS.LT.0.0)BITS=0
IBITS=BITS+.5
SET BIT-ASSIGNMENT & CLAMP # 5 BITS
IF((BITS.GE.5)IBITS=5
IBIT(I)=IBITS
REMAINING BITS=LAST VALUE-NEXTEST ASSIGNMENT
IBITS=-IBITS-IBITS
IF(IBITSL.GT.0)GO TO 110
EITHER NONE REMAINING OR TOO MANY IN LAST ASSIGNMENT
IBIT(!)-IBIT(!)+IBITSL
GO TO 111
CONTINUE
FINAL DCT BIT ASSIGNMENT

ALL LTH SAMPLES USED.TRY TO SCALE UPWARDS
DO 160 I=1,LTH

```

```

C      IF(IBITSL-LE,0)GO TO 111
        IBITI=IBIT(I)+1
        CLAMP @ 5 BITS/SAMPLE
        IF(IBITI.GT.5)GO TO 160
        IBIT(I)=IBITI
        IBITSL=IBITSL-1
    160 CONTINUE
    111 CONTINUE
D      WRITE(4,911)(I,IBIT(I),I=1,LTH)
D911   FORMAT(/1X,6(1X,'IBIT(',13,')=' ,13,2X))
        QUANTIZE DCT(I)

DO 113 I=1,LTH
J=IGORDR(I)
IGTDCT=0
NUMBIT=IBIT(I)
LEVEL=2**NUMBIT
IF(NUMBIT.LE.0)LEVEL=0
NL2=LEVEL/2
K=NUMBIT*16
DO 115 L=1,NL2
K=K+1
IF(ABS(DCT(J)).LE.DCTTHR(K)*DCT1(J))GO TO 114
CONTINUE
IGTDCT=L-1
IF(DCT(J).LT.0)IGTDCT=IGTDCT+NL2
GTDC1(J)=IGTDCT
GTDC1(I)=IGTDCT
WRITE(4,912)(I,GTDCT(I),I=1,LTH)
D912   FORMAT(/1X,4(1X,'GTDCT(',13,')=' ,13,2X))
        IF(SER.EQ.NO)GO TO 1902

<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>>>>>
NOW BEGIN DECIMAL TO DIGITAL CONVERSION AND ERROR
PROTECTION TO ALLOW FOR MODEM TRANSMISSION

CALL OURS(NBRPF,NBRPF)
SUBROUTINE DITER(NBRPF)
THESE BITS IN THE INBA VECTOR WILL BE
TRANSMITTED THROUGH A CHANNEL WHICH IS NOT ERROR FREE

IF(ENDEC.EQ.NO)GO TO 4449
<<<<<<<<<<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>>>>>>
THEN ENCODE IN AN MACH(63,45) CODE
THE TOTAL NUMBER OF BITS ERROR PROTECTED IS 3x45
DO 446 NDB=1,NBPS
CALL OURS
CALL ENCDBCH
CONTINUE
IF(ERRDEL.EQ.NO)GO TO 4450
446
4449
<<<<<<<<<<<<<<<<<<<<<<<<<<
SIMULATE CHANNEL WITH DESIRED ERROR RATE OF PREA
WRITE(5,1905)(INBA(I),I=1,NBRPF)
CALL OUR13(NBRPF,PREA,IRN,JRN,NBRB,3)
CALL CEIR( )

```

```

4450 IF(VNDEC.EQ.ND)GO TO 445
    <<<<<<<<<<<<<<<<<<<<<<<<<<<<
NOW DECODE AND DETECT ERROR LOCATION
WRITE(S,1905)(INBA(I),I=1,NBRPF)
FORMAT(1X,63(11))
NFRSK=0
DO 448 NDB=1,NEPB
CALL DECBC
CALL QVR10
IF(NDB.EQ.1.AND.NES.GE.4)NFRSK=1
CONTINUE
WRITE(S,1905)(INBA(I),I=1,NBRPF)
CONTINUE
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
NOW CREATE DECIMAL DATA FROM THE BINARY DATA
READ SIDE INFORMATION FROM BINARY VECTOR
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CALL QVR11
CALL BATSD(MEPB)
<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CONTINUE
NO 1900 J=1,LPCN
QRPAR(J)=QTTPAR(J)
GRG-QTG
QRH-GTH
QRDC-GTDC
QRVAR-QtUAR
DO 1901 I=1,LTH
QRCT(I)=GTDC(I)
C1901
DEQUANTIZE PARCOR(J),J=1,...LPCN,G,M,DCBIAS,UAR
DEQUANTIZE PARCOR(J)
DO 1751 J=1,LPCN
K=(J-1)*32+1
I=QRPAR(J)+K
REMOVE BIAS
DRPAR(J)=-PDEC(I)+1,0
WRITE(4,940)(J,DRPAR(J),J=1,LPCN)
FORMAT(/1X,4(1X,'DRPAR(',12,')=',E15.8,2X))
DEQUANTIZE PITCH GAIN,G
I=GRG+1
DRG=PDEC(I)
DEQUANTIZE PITCH,M
IF(IPDEC.EQ.0)GO TO 9193
ITR=GRM
IF(ITR.LE.47)GO TO 9192
ITR1=GRM-47
ITR=(ITR1-1)*2+48
DPM=ITR+15
GO TO 9194
DPI=1
CONTINUE
WRITE(4,941)DPM,DRG
FORMAT(/1X,'DPM=',E15.8,' AND DRG=',E15.8)
DEQUANTIZE DC BIAS,DCBIAS
I=GRDC+1

```

```

DRDC=DCDEC(1)
WRITE(4,942)DRDC,APDC
FORMAT(/1X,'DEDC=',E15.8,5X,'QRDC=',I5)
DEQUANTIZE VARIANCE,VAR
I=GRVAR+
DRVAR=VARDEC(1)
WRITE(4,943)DRVAR
FORMAT(/1X,'DRVAR=',E15.8)
RECREATE LP COEFFICIENTS
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CALL OUR4
CALL PARPRE(LPCN,DRPAR,DRA)
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
REGENERATE RESIDUAL ENERGY
U=1.0-DRPAR(1)*X2
DO 1870 I=2,LPCN
U=UX(I,0-DRPAR(1)*X2)
ENG=SORT((2.0*XINSTAGE)*U)
WRITE(4,932)ENG
FORMULATE LPC BASIS SPECTRUM
XR(1)=1.
XI(1)=0.
DO 1628 I=1,LPCN
XR(I+1)=DRA(I)
XI(I+1)=0.
DO 1629 I=LPCN+2,LTH2
XR(I)=0.
FORMULATE THE PITCH WEIGHTING SPECTRUM
AGAIN WE WILL CONSTRUCT A ZERO DC
BIASED PITCHED SIGNAL
DRM=DRM-1
DO 198 I=1,LTH2
XR(I)=0.
XI(I)=0.
KI=LTH/IFIX(DRM)
DCPIT=0.0
EGR=0.
DO 199 K=1,100
J=(K-1)*XDRM+1
IF(J.GT.LTH)GO TO 11020
TEMP=DRGXX(K-1)
DCPIT=DCPIT+TEMP
XI(J)=TEMP
DCPIT=DCPIT/FLOAT(LTH)
DO 9798 I=1,LTH
XI(I)=XI(I)-DCPIT
EGR=EGR+XI(1)*X2
CONTINUE
EGR=1./SORT(EGR)
WRITE(4,917)K,EGR
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
CALL OUR5
CALL POWTAG(XR,XI,NSTAGE,LTH2)
<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<
DO 1630 I=1,LTH
DCI(I)=ENG/XR(I)
WRITE(4,907)(I,DCI(I),I=1,LTH)

```

```
C
IF(DRM.EQ.0)GO TO 11009
CALL PITMT(EGR,LTH,XI,PWEIT)
GO TO 11011
C
11009 DO 11010 I=1,LTH
PWEIT(1)=EGRXSORT(XR(1)*XR(I)+XI(I)**XI(I))
PWEIT(I)=EGRXXI(I)
PWEIT(I)-1.0
CONTINUE
WRITE(4,906)(I,PWEIT(I),I=1,LTH)
C
COMPLETE BASIS SPECTRUM
DO 1102 I=1,LTH
DC(1,I)=DCT1(I)*PWEIT(I)
WRITE(4,908)(I,DCT1(I),I=1,LTH)
ORDER DCT MAX-TO-MIN
<<<<<<<<<<<<<<<<<<<<<<<<<
DCT1(I)=1./E-7
IF(TYPSRT.EQ.YES)CALL OUR6
IF(TYPSRT.EQ.NO)CALL OUR7
<<<<<<<<<<<<<<<<<<<<<<<<<
WRITE(4,909)(I,DCT2(I),I=1,LTH)
WRITE(4,910)(I,IORDR(I),I=1,LTH)
C
COMPRESS BASIS SPECTRUM
DO 14 J=1,LTH
IF(DCT2(J).EQ.0)DCT2(J)=10E-10
DCT2(J)=ALOG(DCT2(J))/LOG2
WRITE(4,909)(J,DCT2(J),J=1,LTH)
INITIAL DCT BIT ASSIGNMENT
SLOG=0.
DO 1104 I=1,LTH
SLOG=SLOG+DCT2(I)
CLEAR OUT BIT ASSIGNMENT ARRAY
DO 1105 I=1,LTH
IBIT(I)=0
CTEMP=(BTLTH-SLOG)/LTH
DO 1105O I=1,LTH
BITS=CTEMP+DCT2(I)
IF(BITS.LE.0.0)GO TO 11051
LAST-I
LAST POSITIVE ENTRY IS LAST SAMPLE TO BE USED
ITOT=LAST
WRITE(4,930)BITS,CTEMP,ITOT
SLOG=0.0
FORM SUM-OF-LOGS UP TO LAST SAMPLE
DO 1109 I=1,ITOT
SLOG=SLOG+DCT2(I)
CTEMP=(BTLTH-SLOG)/ITOT
TOTAL ASSIGNABLE BITS
IBITSL=BTLTH
DO 1110 I=1,ITOT
K=1
BITS=CTEMP+DCT2(I)
ROUND UPWARDS
IF(BITS.LT.0.0)BITS=0
IBITS=BITS+.5
SET BIT ASSIGNMENT & CLAMP @ 5 BITS
IF(IBITS.GE.5)IBITS=5
IBIT(I)=IBITS
REMAINING BITS-LAST VALUE-NEXT ASSIGNMENT
```

[illegible]

[illegible]

```

0913 FORMAT(/1X,4(1X,'NOUT(',I3,',')=.16,2X))
      ICSUB IS USED IN THE SIGNAL TO NOISE
CALCULATION. IT CORRESPONDS TO THE NUMBER
OF FRAME WHICH HAD MORE THAN 3 ERRORS
IN THE FIRST BLOCK.
      IF(NFRSK.EQ.0)GO TO 1065
      ICSUB=ICSUB+1
      GO TO 1066

      <<<<<<<<<<<<<<<<<<
>>>>>>>>>>>>>>>>>>>
      COMPUTE S/N RATIO

      CONTINUE
      SUM1=0.
      SUM2=0.
DO 3605 I=1,NTOTO
      XNIN=FLOAT(NIN(I))
      SUM1=SUM1+XNIN*XNIN
      SUM2=SUM2+(XNIN-Float(NOUT(I)))**2
      SNR=10.*XLOG10(SUM1/SUM2)
      FRAME=ICOUNT-ICSUB-INITFR+1-IVARF
      IBTLTH=INT(BLTH)
      CSN=((FRAME-1)/FRAME)*CSN+(1./FRAME)*SNR
      WRITE(5,255)ICOUNT,SNR,CSN,M.G,IBTLTH
      FORMAT(1X,'FRAME=',I4,2X,'SN=',FS,2,2X,'PITCH=',I3
1,2X,'P.GAIN=',FS,3,2X,'ICT BITS=',I4/)
      CONTINUE
      GO TO 1001
      CONTINUE
      STOP 'ATC70 DUNE 1'
END
SUBROUTINE PITMT(EGR,LTH,XI,PWEIT)
DIMENSION XI(1),PWEIT(1)
STR=0.0
DO 1010 I=1,LTH
      T1=EGR*XI(I)
      T2=FLOAT(I-1)/FLOAT(LTH)
      T3=1.-T2
      XI(I)=T1*T3+T2
      IF(XI(I).LE.0.5)XI(I)=0.5
      STR=STR+XI(I)**2
CONTINUE
STR=SQRT(FLOAT(LTH)/STR)
DO 1011 I=1,LTH
      PWEIT(I)=XI(I)*STR
RETURN
END

```

```

SUBROUTINE CUP1
  SUBROUTINE DCTSUB(LTH,YR,YI,DCT,INIT1)
  THIS PROGRAM DOES THE FORWARD AND INVERSE FAST
  DISCRETE COSINE TRANSFORMS. IF THE VALUE OF INIT1
  IS 1 THEN THE FORWARD DCT IS DONE. IN THIS CASE
  LTH DCT COEFFICIENTS ARE RETURNED ALONG WITH 2LTH
  MAGNITUDE SPECTRUM COEFFICIENTS TO BE USED
  IN THE AUTOCORRELATION FUNCTION ESTIMATION.

  IF THE VALUE OF INIT1 IS 2 THEN THE INVERSE
  DCT IS PERFORMED AND LTH VALUES OF A DISCRETE TIME SIGNAL ARE
  RETURNED.
  COMMON/BLOCK1/INIT1,DCT(256),LTH,NSTAGE
  COMMON/SORT/DCT1(256),DCT2(256),IDRDR(256),YR(512),YI(512)
  LTH1=LTH/2
  LTH2=LTH+2
  LTH3=LTH+2
  LTH4=2*LTH
  LTHALF=LTH1/2
  PI=4.0*ATAN(1.0)
  PI2=2*PI/LTH
  PI3=PI/2/8
  PI6=2*PI/3
  IF (INIT1.EQ.2) GO TO 4000
  NOW RESHAFFLE BY FIRST ADDING LTH ZEROES
  AND THEN RESHAFFLING TO DO THE FFT
  DO 10 I=1,LTH
    YI(LTH+I)=0.0
    YI(1+LTH)=0.0
  DO 11 I=1,LTH
    YI(1)=YR(2*I-1)
    YI(2*LTH+1-I)=YR(2*I)
  DO 12 I=1,2*LTH
    YR(1)=YI(1)
    YI(1)=0.0
  NOW SEPARATE INTO EVEN AND ODD COMPONENTS AND SCALE BY 2
  DO 13 J=1,LTH
    YR(J)=YR(2*I-1)/2.0
    YI(J)=YR(2*I)/2.0
  CALL EVDOD(LTH,YR,YI,1,1)
  DO 35 I=1,LTH
    DCT(1)=2*(COS((2*I-2)*PI/3)*YR(2*I-1)+SIN((2*I-2)*PI/3)*YI(2*I-1))
    DCT2P=2*(COS((2*I-1)*PI/3)*YR(2*I)+SIN((2*I-1)*PI/3)*YI(2*I))
    YR(2*I-1)=DCT(1)**2
    YR(2*I)=DCT2P**2
    YI(2*I-1)=0.0
    YI(2*I)=0.0
  GO TO 5000

  NOW WE WILL DO THE INVERSE DCT
  CONTINUE
  TYPE=505,INIT1,LTH,NSTAGE
  FORMAT(1X,14,1X,14,1X,14)
  WRITE(5,934)(I,DCT(1),I=1,LTH)
  FORMAT(1X,4(1X,DRDCT2(',13,')=','E15.8,2X))
  YI(1)=DCT(1)/2.0
  YI(1)=0.0
  YR(LTH/2+1)=COS(PI6*LTH/2)*DCT(LTH/2+1)+SIN(PI6*LTH/2)*
  YI(LTH/2+1)/2.0
  YI(LTH/2+1)=SIN(PI6*LTH/2)*DCT(LTH/2+1)-COS(PI6*LTH/2)
  YI(DCT(LTH/2+1))/2.0

```

```

DO 410 K=2,LTH/2
YR(K)=(COS(PI6*(K-1))*XDT(K)+SIN(PI6*(K-1))*XDT(LTH+2-K))/2.0
YI(K)=(-SIN(PI6*(K-1))*XDT(K)-COS(PI6*(K-1))*XDT(LTH+2-K))/2.0
YR(LTH+2-K)=YR(K)
YI(LTH+2-K)=-1.0*YI(K)

```

410

```

DO 420 K=1,LTH/4+1
J=K-1
Q1=(YI(K)+YI(LTH/2+2-K))/2.
Q2=(YI(K)-YI(LTH/2+2-K))/2.
Q3=(YR(K)+YR(LTH/2+2-K))/2.
Q4=(YR(K)-YR(LTH/2+2-K))/2.

```

```

QTEMP1=SIN(2*PI*XJ/LTH)*Q4
QTEMP2=SIN(2*PI*XJ/LTH)*Q1
QTEMP3=COS(2*PI*XJ/LTH)*Q1
QTEMP4=COS(2*PI*XJ/LTH)*Q4

```

```

YR(K)=Q3-QTEMP1-QTEMP3
YI(K)=Q2+QTEMP4-QTEMP2
YR(LTH/2+2-K)=Q3+QTEMP1+QTEMP3
YI(LTH/2+2-K)=Q2+QTEMP4-QTEMP2-Q2
WRITE(S,934)(I,YR(I),I=1,LTH)
WRITE(S,934)(I,YI(I),I=1,LTH)
CALL FFTF(NTAGE-2,2,1,1)
DO 430 N=1,LTH/2
DCT(2*N-1)=YR(N)
DCT(2*N)=YI(N)
YI(N)=0.0
YI(LTH/2+N)=0.0

```

426

430

```

DO 445 I=1,LTH/2
YR(2*I-1)=DCT(I)
YR(2*I)=DCT(LTH-I+1)
RETURN
END

```

430

445

5000

OURC.FTN 27-NOV-79

```
SUBROUTINE OUR2
COMMON/MTAPE0/NIN(256),NOUT(256)
COMMON/MTAPE1/NTKIP,IST,NTOTI,NTUPS,NTOTO
COMMON/MTAPE2/NEID,NEKR,NFILE,NINS,NOUTS
COMMON/MTAPE3/NBF(1324),NBUF(1324)
COMMON/MTAPE4/LST,IBEG
COMMON/MTAPE5/MASK,ISU(2),IOATT,IOSUC,IEALN,IORDJ
COMMON/MTAPE6/IEEUF,IOEUF,IOEZF,IOPLB,MT0(C,MT1/6),DSW
COMMON/MTAPE6/APPEND
COMMON/SORT/DCT1(256),DCT2(256),IORDR(256),XR(S12),XI(S12)
NOW INSERT COMMON BLOCKS FOR THE OVERLAYS
COMMON/BLOCK1/INITI,DCT(256),LTH,NSTAGE
COMMON/BLOCK2/G(8),R(11),U,PARCOR(8)
COMMON/BLOCK3/DTPAR(8),DTA(8),LPON
EQUIVALENCE (DCT,DRODCT),(DTPAR,DRPAR),(DTA,DRA)
COMMON/BLOCK4/LTH2
COMMON/BLOCK5/LTH3,LTH4,NP,XN,LP1,ARG
1,DLOG2,ONS,ICOUNT,IREF,NBPF,NBPPF
2,P1,NSTAG,BLTH,PLATE,TYPSRT,NEPB
LOGICAL X1,PLATE,YES,NO,TYPSRT
DATA YES,'Y',
DATA NO,'N',/
```

INITIALIZATION

```
WRITE(S,2342)
FORMAT(1H,'NUMBER OF BITS/FRAME TOTAL=')
READ(S,101,END=999,ERR=999)NBPF
WRITE(S,2345)
FORMAT(1H,'NUMBER OF BITS/FRAME FOR DCT=')
READ(S,101,END=999,ERR=999)NBITS
BLTH=NBITS
WRITE(S,2606)
FORMAT(1H,'USE PITCH WEIGHTING(Y/N)? ')
READ(S,2601,END=999,ERR=999)PLATE
FORMAT(A1)
TYPE 2607
FORMAT(1H,'USE FAST SORT(Y/N)? ')
READ(S,2601,END=999,ERR=999)TYPSRT
```

```
*****CONSTANTS
*****
```

THE FRAME LENGTH IS LTH AND IS 256

LTH=256

WE WILL USE PITCH WEIGHTING PLATE

PLATE=YES

WE WILL USE FASTF LINKED LIST SORT TYPSRT

TYPSRT=YES

WE WILL SET THE NUMBER OF STAGES FOR THE DFT TO

BE LOG(256)+1

NSTAGE=9

SET UP CONSTANTS

OF FFT STAGES,STAGE

WRITE(S,2343)

FORMAT(1H,'NUMBER OF FFT STAGES=')

READ(S,101,END=999,ERR=999)NSTAGE

NSTAG=NSTAGE+1

VERY ACCURATE VALUE FOR P1=7.14159...

P1=4.0*ATAN(1.0)

FRAME SIZE,LTH

2342

2345

2606

2601

2607

2343

READ (5,101,END=9999,ERR=9999) IREC
RETURN
CALL EXIT
END

9999

```

PROGRAM SOLV,FTN
COMPUTES THE PREDICTOR COEFFICIENTS OF A WAVEFORM
GIVEN THE NORMALIZED AUTOCORRELATION COEFFICIENTS

SUBROUTINE CURC
COMMON/BLOCK2/A(8),R(11),U,PARCOR(8)
COMMON/BLOCK3/DTPAR(8),DTPA(8),N
DIMENSION B(40)
SUBROUTINE SOLVE(A,R,N,U,PARCOR)
A(1)=-R(2)/R(1)
PARCOR(1)=-A(1)
U=1+A(1)*R(2)
TYPE 999,I,U
FORMAT(1X,'U(',12,')=' ,E15.8)
DO 30 I=2,N
W=R(I+1)
DO 10 M=1,I-1
B(M)=A(I-M)
W=W+B(M)*R(M+1)
AK=-W/U
DO 20 M=1,I-1
A(M)=A(M)+AK*B(M)
A(I)=AK
PARCOR(I)=-AK
U=U+AK*W
TYPE 999,I,U
CONTINUE
TYPE 998,(I,PARCOR(I),I=1,N)
FORMAT(1X,'PARCOR(',12,')=' ,E15.8)
TYPE 997,(I,A(1),I=1,N)
FORMAT(1X,'A(',12,')=' ,E15.8)
RETURN
END

```

00000

0

D
D999

10

20

D
30

D998

D
D997

```

SUBROUTINE OUR4
SUBROUTINE PARPRE(N,PARCOR,A)
CONTIN/BLOCK3/PARCOR(8),A(8),N
DIMENSION AP(50)
A(1)=PARCOR(1)
DO 120 I=2,N
IM1=I-1
DO 110 J=1,IM1
AP(J)=A(J)-PARCOR(I)*A(I-J)
AP(I)=PARCOR(I)
DO 140 J=1,I
A(J)=AP(J)
CONTINUE
TYPE 997,(I,A(I),I=1,N)
FORMAT(1X,'A',12,' ',E15.8)
RETURN
END

```

C

110

140

120

D997

```

SUBROUTINE OURS
SUBROUTINE FOURAG(XR,XI,NSTAGE,LTH2)
PROGRAM TO DETERMINE POWER SPECTRUM OF TWO REAL SIGNALS
WE WILL USE THE PROPERTIES OF EVEN ODD
SEPARATION TO OBTAIN THE 2 POWER SPECTRUMS
X(N)=XR(N)+XI(N)
COMMON/SORT/DCT1(256),DCT2(256),IORDR(256),XR(512),XI(512)
COMMON/BLOCK1/INIT,DCT(256),LTH,NSTAGE
LTH3=LTH2+2
IP=LTH2/2
CALL FASTF(NSTAGE,1,1,2)
XR(1)=ABS(XR(1))
XI(1)=ABS(XI(1))
XR(IP+1)=ABS(XR(IP+1))
XI(IP+1)=ABS(XI(IP+1))
DO 30 I=2,IP
  QRTMP=2*XR(1)*XR(LTH3-I)
  QITMP=2*XI(1)*XI(LTH3-I)
  XRTEMP=QRTMP+XR(LTH3-I)*XR(LTH3-I)
  XITEMP=QITMP+XI(LTH3-I)*XI(LTH3-I)
  XR(1)=XRTEMP/2.0
  XI(1)=XITEMP/2.0
  XR(LTH3-I)=XR(1)
  XI(LTH3-I)=XI(1)
CONTINUE
RETURN
END

```

```

SUBSLAU SORTS 128 DCT COEFFICIENTS INTO THEIR
DECREASING ORDER AND PASSES BACK A 16 BIT WORD
TO THE MAIN PROGRAM ATC70.
WRITTEN JULY 3, 1979 BY MIH

SUBROUTINE OURS
  SUBROUTINE SORT4
    DIMENSION QUANT(4),P(4)
    COMMON/SORT/DCT1(256),DCT2(256),IORDR(256),XR(512),XI(512)
    COMMON/BLOCK1/INIT1,DCT(256),LTH,NSTAGE
    INTEGER P
    REAL YMIMR(1024)
    EQUIVALENCE (YMIMR(1),XR(1)),(YMIMR(513),XI(1))
    THESE QUANTAL LEVELS ARE 4 EQUAL MASS POINTS FROM THE GAMMA
    DISTRIBUTED DCT1 COEFFICIENTS. THEY EACH REPRESENTS 25 PERCENT
    OF THE DISTRIBUTION.
    DATA QUANT/0.713,2.50,7.50,1.0E20/
    DO 1 L=1,4
      P(L)=0
    DO 2 I=1,512
      YMIMR(I)=0
    DO 5 J=1,LTH
      K=J-1
    DO 3 LEVEL=1,4
      IF(DCT1(J).LT.QUANT(LEVEL)) GO TO 4
    CONTINUE
    YMIMR(J+(LEVEL-1)*LTH)=FLOAT((J-1-P(LEVEL))) * 256. + FLOAT(K)
    P(LEVEL)=J
    IND=LTH
    DO 6 LEVEL=1,4
      ISKIP=0
      MARK=P(LEVEL)
      MARK=MARK-ISKIP
      IF(MARK.LT..75) GO TO 6
      NOTE WE TAKE OUT THE UPPER 15TH BIT TO AVOID
      INTEGER OVERFLOW IN TESTING FOR THE LOWER EIGHT
      BITS. THIS IS NOT NECESSARY WHEN
      WE DIVIDE BY 256 TO OBTAIN THE UPPER BITS.
      X=YMIMR(MARK+(LEVEL-1)*LTH)
      IF(X.GT.32767.0)X=X-32768.
      DCT2(IND)=DCT1(IND+(IFIX(X),255)+1)
      IORDR(IND)=IAND(IFIX(X),255)+1
      IND=IND-1
    ISKIP=IAND(IFIX((YMIMR(MARK+(LEVEL-1)*LTH))/256),255)+1
    GO TO 7
  CONTINUE
  RETURN
  END

```

```

PROGRAM NAME: SORTN.FTN  ORIGINATED: 13-SEP-79
                        UPDATED: 13-SEP-79
PERFORMS SORT OF DCT1 INTO DCT2 IN MAX-TO-MIN ORDER
W/ INDEX ORDER RETURNED IN IORDR

SUBROUTINE OUR7
SUBROUTINE SORTN(LTH)
COMMON/SORT/DCT1(256), DCT2(256), IORDR(256), XR(512), XI(512)
COMMON/BLOCK1/INIT1,DCT(256),LTH,NSTAGE
DO 202 I=1,LTH
  DCT2(I)=DCT1(I)
  IORDR(I)=I
  LTH1=LTH-1
DO 100 J=1,LTH1
  JP1=J+1
  IF(DCT2(J).GE.DCT2(JP1))GO TO 100
  TEMP=DCT2(J)
  DCT2(J)=DCT2(JP1)
  DCT2(JP1)=TEMP
  ITEMP=IORDR(J)
  IORDR(J)=IORDR(JP1)
  IORDR(JP1)=ITEMP
GO TO 1015
CONTINUE
RETURN
END

```

202
1015

100

```

SUBROUTINE CURB(NBPF,NBPT)
SUBROUTINE DITBA(NBPF)
ROUTINE DITBA IS DIGITAL TO BINARY CONVERSION
ROUTINE
NBPF IS THE NUMBER OF BITS PER FRAME
COMMON/BLOCK1/INT1(1),DCT(256),LTH,NSTAGE
COMMON/SORT/DCT1(256),DCT2(256),JORDR(256),XR(512),XI(512)
COMMON/DITBA/ GTDLT(256),GTPAR(8),GTDC,GTUAR,QTU,QTG
COMMON/BLOCK6/IBIT(256),IPDEC,INBA(500)
DIMENSION INB(6)
INTEGER GTDCT,GTPAR,GTDC,GTUAR,QTU,QTG
TYPE 57,NBPF,IPDEC
FORMAT(1X,'NBPF=',14,3X,'IPDEC=',14)
DO 10 I=1,NBPF
INBA(I)=0
XI ARRAY CONTAINS THE PROTECTED BITS OF DCT
NBPT IS NUMBER OF BITS OF DCT COEFFICIENTS TO BE PROTECTED
IF(IPDEC.EQ.1)NBPT=NBPT-8
SELECT THE DCT BITS TO BE PROTECTED
INP=0
IBST=IBIT(1)
FORMAT(1X,'IT1=',13,1X)
CONTINUE
DO 100 I=1,LTH
IAB=IBIT(I)
WRITE(4,16)I,IBST,IAB
FORMAT(1X,'I=',13,2X,'IBST=',13,2X,'IAB=',13,2X)
IF(IAB.LT. IBST)GO TO 130
IF(IAB.EQ.0)GO TO 100
INP=INP+1
IT1=2XX(IAB-1)
WRITE(4,15)IT1
IT2=IT1-1
IT3=GTDCT(I)
GTDCT(I)=IAND(IT3,IT2)
IBIT(I)=IBIT(I)-1
I1=I13/IT1
XI(INP)=FLOAT(I1)+0.1
IF(INP.GE.NBPT)GO TO 120
CONTINUE
IBST=IBST-1
GO TO 110
CONTINUE

CONVERT PARCOR INTO BINARY VECTOR
AND LOAD INTO INBA
INB IS THE TEMPORARY BINARY VECTOR
ALLOCATE 5,5,4,4,3,2,2 BITS TO THE PARCORS
NBA=0
DO 20 I=1,8
IDT=6-(I+1)/2
CALL DBCONV(GTPAR(1),IDT,INB)
DO 30 J=1,IDT
NBA=NBA+1
INBA(NBA)=INB(J)
CONTINUE
CONVERT VARIANCE INTO BINARY VECTOR
CALL DBCONV(GTUAR,5,INB)
DO 40 J=1,5
NBA=NBA+1
INBA(NBA)=INB(J)
NOW CONVERT VOICED INTO BINARY VECTOR
NBA=NBA+1

```

```

INBA(NBA)=IPDEC
PROTECT 3 BITS FROM DCBIAS
IBDC=GTDC
DO 41 I=1,3
  IPD=5-I
  IT1=2**IPD
  IT2=IT1-1
  II=IBDC/IT1
  IBDC=IAND(IBDC,IT2)
  NBA=NBA+1
  INBA(NBA)=II
  CONTINUE
41 GTDC=IBDC
NOW GENERATE BINARY PITCH AND PITCH GAIN
IF(IPDEC.EQ.0)GO TO 200
CALL DBCONJ(GTM,6,INB)
DO 210 J=1,6
  NBA=NBA+1
  INBA(NBA)=INB(J)
  CALL DBCONJ(GTS,2,INB)
  INBA(NBA+1)=INB(1)
  INBA(NBA+2)=INB(2)
  NBA=NBA+2
  CONTINUE
210 NOW LOAD XI INTO THE BINARY ARRAY
ISTP=1
IEDP=45-NBA
IEDP1=IEDP
DO 310 NEB=1,3
  IF(IEDP.GT.0) GO TO 400
  IEDP=45
  NBA=63
  IF(IEDP1.EQ.0)GO TO 310
  INBA(64)=INBA(46)
  INBA(46)=0
  NBA=64
  IEDP=44
  IF(IEDP1.EQ.-1)GO TO 310
  INBA(65)=INBA(47)
  INBA(47)=0
  NBA=65
  IEDP=43
  GO TO 310
CONTINUE
DO 300 I=ISTP,IEDP
  NBA=NBA+1
  INBA(NBA)=XI(I)
  NBA=NEBAS3
  ISTD=IEDP+1
  IEDP=IEDP+45
  CONTINUE
LOAD THE REMAINING TWO BITS OF THE DCBIAS
CALL DBCONJ(GTDC,2,INB)
DO 599 J=1,2
  NBA=NBA+1
  INBA(NBA)=INB(J)
599 NOW LOAD THE REMAINING THE UNPROTECTED
  BITS FROM THE GTDC
DO 500 I=1,LTH
  IDT=IBIT(I)
  IF(IDT.LE.0)GO TO 500

```

```

CALL DECON(QTDT(1),IDT,INB)
DO 600 J=1,IDT
  NBR=NBR+1
  INB(NBR)=INB(J)
  CONTINUE
  TYPE 808,NBR
  FORMAT(1X,'NBR IN SER. ROUTINE=',14)
  WRITE(4,909)(1,INB(I),I=1,NBRPF)
  FORMAT(1X,8(2X,13,2X,1,2X,13))
  RETURN
END

```

500
600
7
0808
0909

00000000

```

DECONVERT TAKES A DECIMAL NUMBER AND RETURNS
ITS BINARY EQUIVALENT
SUBROUTINE DECON(IX,LIB,INB)
  IX IS THE DECIMAL NUMBER
  LIB IS THE NUMBER OF BITS TO BE ALLOCATED
  INB IS THE BINARY VECTOR EQUIVALENT
  SUBROUTINE DECON(IX,LIB,INB)
  DIMENSION INB(1)
  IY=IX
  DO 10 I=1,LIB
    IR=LIB+1-I
    INB(IR)=MOD(IY,2)
    IY=IY/2
  RETURN
END

```

10


```

SUBROUTINE OUR10
ED3BCH,FTN
ENCODING, DECODING TEST OF BCH CODE
MARCH 20, 1979
LENGTH OF BCH CODE
POLYNOMIALS ARE ORDERED IN DESCENDING POWER SERIES
LBOH=2**MBOH-1
THIS ROUTINE CORRECT 3 ERRORS
COMMON/BLOCK6/INA(256),IPDEC,INBA(500)
COMMON/DEBCH/NE,KT
COMMON/BLOCK3/ICOFF1(64),ICOFF3(64),ICOFF5(64)
COMMON/BLOCK1/ICOFF(64),IORDR(64)
DIMENSION NEAL(6),INB(7),INC(7)
DIMENSION ISD1(9),ISD3(9),ISD5(9)

READ INBA VECTOR INTO INA VECTOR IN BLOCK OF 63
DO 10 I=1,63
KT1=(KT-1)*63+I
INA(I)=INBA(KT1)
DECODING ROUTINE

*****
FIRST SET UP ALL OF THE COEFFICIENT
AND POWER TABLES TO BE USED BY THE SUBROUTINES
IF WE HAVE ALREADY GENERATED THE TABLES FOR THE
DECODING ROUTINE DO NOT DO IT AGAIN
IF(KT.GE.2)GO TO 111
MBOH=6
LBOH=2**MBOH-1
CALL GENTAB(MBOH)

CALCULATE POWER SUMS
R(ALPHAX1),R(ALPHAX3),R(ALPHAX5)
CONTINUE
CALL GF2POL(MBOH,INA,ISD1,IP1,
21SD3,IP3,ISD5,IP5)
IPSIG1=IP1
WRITE(5,101)(ISD1(I),I=1,MBOH)
WRITE(5,202)(ISD3(I),I=1,MBOH)
WRITE(5,303)(ISD5(I),I=1,MBOH)
FORMAT(1X,'S1-',1811)
FORMAT(1X,'S3-',1811)
FORMAT(1X,'S5-',1811)
WRITE(5,102)IP1,IP3,IP5
FORMAT(16,1X,16,1X,16)
CHECK ERROR RANGE
IF(IP1.EQ.-1.AND.IP3.EQ.-1.AND.IP5.EQ.-1)GO TO 1599
GO TO 50
NO CHANNEL ERROR
CONTINUE
WRITE(5,1600)
FORMAT(1X,'NO CHANNEL ERROR')
RETURN
CONTINUE
CORRECT CHANNEL ERROR

CALCULATE SIGMA(1),I=1,3
CALCULATE DET(3),I.E.,S1**3+S3
CALL MODPOL(IP1,3,LBOH,IEMOP1)
CALL INALOK(MBOH,IEMOP1,INA)
CALL GF2ADD(INA,MBOH,ISD3,MBOH,INA,ND)
CALL LOOKUP(MBOH,INA,IPDEN0)

```

```

C      IF(ISA0.EQ.2)WRITE(5,303)(INB(1),I=1,6)
      NES=3
      IF(IPDENQ.CT.-1) GO TO 70
      NES=1
      ONLY ONE ERROR OCCUR
      GO TO 80
      CONTINUE
      CALCULATE SIGMA(2) AND SIGMA(3)
      CALL MODPOL(IP1,2,LBCH,INUMOP)
      CALL MODPOL(INUMOP,IP3,LBCH,INUMOP)
      CALL INALOK(MBCH,INUMOP,INB)
      CALL GF2ADD(INB,MBCH,ISDE,MBCH,ISD3,NC)
      CALL LOOKUP(MBCH,ISD3,IPSIG2)
      IF(ISA0.EQ.2)WRITE(5,303)(INC(1),I=1,MBCH)
      CALL MODPOL(IPSIG2,IPDENQ,LBCH,IPSIG2)
      CALL INALOK(MBCH,IPSIG2,ISD3)
      WRITE(5,606)(ISD3(1),I=1,6)
      FORMAT(1X,'SIGMA2=',18I1)
      IF(ISA0.EQ.2.AND.ISA1.EQ.1)WRITE(5,404)I,(ISD3(J),J=1,MBCH)
      FORMAT(1X,'N=',13,3X,8I1)

C      IF(ISA0.EQ.2)WRITE(5,202)(ISD3(1),I=1,MBCH)
      CALCULATE SIGMA(3)
      CALL MODPOL(IPSIG2,IP1,LBCH,IPSIG3)
      CALL INALOK(MBCH,IPSIG3,INB)
      CALL GF2ADD(INA,MBCH,INB,MBCH,ISDE,NC)
      CALL LOOKUP(MBCH,ISD5,IPSIG3)
      WRITE(5,707)(ISD5(1),I=1,6)
      FORMAT(1X,'SIGMA3=',18I1)
      IF(IPSIG3.EQ.-1)NES=2
      CONTINUE
      TYPE 505,NES
      FORMAT(5X,'NUMBER OF ERRORS =',I3)
      CORRECT NES ERROR BY CHIEN'S SEARCH METHOD

      IPSIG1,IPSIG2,IPSIG3 ARE THE EXPONENTS OF SIGMA1,SIGMA2,SIGMA3
      ISD1,ISD3,ISD5, ARE THE VECTOR POWER SUMS 1,3,5
      NEST=0
      DO 11 11=1,LBCH
      111=11-1
      IF(111.EQ.0)111=LBCH
      IF(NES.EQ.1.AND.IPSIG1.EQ.0) GO TO 33
      IF(NES.EQ.1)GO TO 44
      CALL GF2ADD(ISD1,MBCH,ISD3,MBCH,INB,NC)
      CALL GF2ADD(ISD5,MBCH,INB,MBCH,INC,NC)
      IF(ISA0.EQ.2)WRITE(5,303)(INL(1),I=1,6)
      CALL LOOKUP(MBCH,INC,IPSUM)
      IF(ISA0.EQ.2.AND.ISA1.EQ.1)WRITE(5,303)(INC(1),I=1,MBCH)
      IF(INC(1).EQ.1)GO TO 44
      IF(IPSUM.NE.0) GO TO 44
      CORRECT ERROR
      CONTINUE
      KTT=(KT-1)*63+111
      NEST=NEST+1
      NEEL(NEST)=KTT
      WRITE(5,1700)111
      FORMAT(10X,'CORRECTED ERROR LOCATION=',13)
      CONTINUE
      SHIFT ISU1,ISU3,ISU5
      FIRST MULTIPLY SIGMA1 BY ALPHA
      CALL MODPOL(IPSIG1,1,LBCH,IPSIG1)
      CALL INALOK(MBCH,IPSIG1,ISD1)
      IF(NES.EQ.1)GO TO 11
      MULTIPLY SIGMA2 BY ALPHA SQUARED

```

```

C 1808 CALL MODMUL(IPSIG2,2,LBCH,IPSIG2)
C 36 CALL INALOK(MBCH,IPSIG2,ISD3,
C 11 TYPE 808,IPSIG3 BY ALPHA CUBED
C 36 FORMAT(SX,IPSIG3,13)
C 11 CALL MODMUL(IPSIG3,3,LBCH,IPSIG3)
C 36 CALL INALOK(MBCH,IPSIG3,ISDS)
C 11 WRITE(5,707)(ISD5(I),I=1,6)
C 11 CONTINUE
C 36 CHECK ERROR STATUS
C 11 TYPE 506,NES,NEST
C 36 FORMAT(SX,NES=,13,3X,NEST=,13)
C 11 IF(NES.EQ.NEST)GO TO 888
C 36 NES=4
C 11 RETURN
C 36 CORRECT ERRORS
C 11 CONTINUE
C 36 DO 72 I=1,NES
C 11 KTI=NEEL(I)
C 36 TYPE 5060,KTI
C 11 FORMAT(SX,ERROR LOCATION =,15)
C 36 TYPE 507,INBA(KTI)
C 11 FORMAT(SX,OLD INBA VALUE BEFORE CORRECTIONS=,13)
C 36 INBA(KTI)=IBOR(INBA(KTI),1)
C 11 TYPE 508,INBA(KTI)
C 36 FORMAT(SX,INBA VALUE AFTER CORRECTION=,13)
C 11 CONTINUE
C 36 RETURN
C 36 END
C 36 SUBROUTINE MODPOW(IEXP1,MULT,IMOD,IEXP2)
C 36 EXP1 IS THE EXPONENT TO BE MULTIPLIED
C 36 MULT IS THE MULTIPLIER OF THE EXPONENT
C 36 IMOD IS THE INTEGER THAT IT IS ALL MODULUED TO
C 36 EXP2 IS THE RETURNED EXPONENT
C 36 ITEMP=MOD((MULT*IEXP1),IMOD)
C 36 IF(IEXP1.EQ.-1) ITEMP=-1
C 36 IEXP2=ITEMP
C 36 RETURN
C 36 END
C 36 SUBROUTINE MODMUL(IEXP1,IEXP2,IMOD,IEXP3)
C 36 EXP1 AND EXP2 WILL BE ADDED AND MODULUED BY IMOD
C 36 ITEMP=MOD((IEXP1+IEXP2),IMOD)
C 36 IF(IEXP1.EQ.-1.OR.IEXP2.EQ.-1) ITEMP=-1
C 36 IEXP3=ITEMP
C 36 TYPES,IEXP1,IEXP2,IEXP3
C 36 FORMAT(2X,15,15,15)
C 36 RETURN
C 36 END
C 36 SUBROUTINE MODSUB(IEXP1,IEXP2,IMOD,IEXP3)
C 36 IEXP2 WILL BE SUBTRACTED FROM IEXP1 AND MODULUED IMOD
C 36 ITEMP=MOD((IEXP1-IEXP2+IMOD),IMOD)
C 36 IF(IEXP1.EQ.-1) ITEMP=-1
C 36 IEXP3=ITEMP
C 36 RETURN
C 36 END
C 36 SUBROUTINE UNPACK(MBCH,ISUM,IVEC)
C 36 DIMENSION IVEC(1)
C 36 DO 10 J=1,MBCH
C 36 INORM=2X(MBCH-J)
C 36 ISHIFT=(ISUM/INORM)
C 36 IT=IAND(1,ISHIFT)
C 36 IVEC(J)=IT
C 36 FORMAT(1X,18IT=,12)
C 36 CONTINUE
C 36 WRITE(5,25)ISUM

```

```
225 FORMAT(1X,'SUM=',I4)  
226 WRITE(5,26)(IUEC(J),J=1,MBCH)  
227 FORMAT(1X,'VECTOR=',5(11,1X))  
228 RETURN  
229 END
```

```

SUBROUTINE GF2POL(MBCH, INA, ISD1, IEXP1,
2ISD3, IEXP3, ISDS, IEXPS)
  DIMENSION ISD3(1), ISDS(1), ISD1(1), INA(1)
  IORD DETERMINES WHICH OF THE FOUR SUMS IS GENERATED,
  S(1), S(3), S(5), ...
  ISD IS THE BINARY VALUED VECTOR CORRESPONDING TO THE
  GF(2) FIELD
  IEXP IS THE EXPONENT CORRESPONDING TO THE VECTOR IN THE FIELD
  IBLCK=(2**MBCH)-1
  IF(INA(1).EQ.0) GO TO 12
  IPOWER1=1
  IPOWER3=3
  IPOWER5=5
  CALL INLOK(MBCH, IPOWER1, ISD1)
  CALL INLOK(MBCH, IPOWER3, ISD3)
  CALL INLOK(MBCH, IPOWER5, ISDS)
  ISD1(MBCH)=IEOR(INA(2), ISD1(MBCH))
  ISD3(MBCH)=IEOR(INA(2), ISD3(MBCH))
  ISDS(MBCH)=IEOR(INA(2), ISDS(MBCH))
  GO TO 19
12 DO 13 I=1, MBCH-1
   ISD1(I)=0
   ISD3(I)=0
   ISDS(I)=0
   CONTINUE
13 ISD1(MBCH)=INA(2)
   ISD3(MBCH)=INA(2)
   ISDS(MBCH)=INA(2)
   DO 20 I=2, IBLCK-1
    CALL CHAPTS(MBCH, ISD1, ISD3, ISDS)
    ISD1(MBCH)=IEOR(INA(I+1), ISD1(MBCH))
    ISD3(MBCH)=IEOR(INA(I+1), ISD3(MBCH))
    ISDS(MBCH)=IEOR(INA(I+1), ISDS(MBCH))
    CONTINUE
20 BEFORE YOU GO GET ITS POWER
   CALL LOOKUP(MBCH, ISD1, IEXP1)
   CALL LOOKUP(MBCH, ISD3, IEXP3)
   CALL LOOKUP(MBCH, ISDS, IEXPS)
   RETURN
   END

```

000000

12

13

19

30

```

SUBROUTINE LOOKUP(MBCH,IVEC,IEXP)
THIS SUBROUTINE RETURNS THE POWER OF THE ELEMENT
IN THE GF(2) FIELD CORRESPONDING TO ITS BINARY VALUED VECTOR -1
NOTICE THAT THE ZEROED VALUE VECTOR TRAPS TO THE EXPONENT -1
IVEC IS THE BINARY VALUED VECTOR
IEXP IS THE POWER OF THE FIELD ELEMENT
COMMON/BLUC31/ICOFF(64),IPOWER(64)
DIMENSION IVEC(1)
INDEX=0
DO 10 I=1,MBCH
INDEX=((2**X(MBCH-I))*IVEC(I))+INDEX
CONTINUE
IEXP=IPOWER(INDEX+1)
RETURN
END

```

```

SUBROUTINE INALOK(MBCH, IPOWER, IVEC)
SUBROUTINE INALOK RETURNS THE BINARY VALUED VECTOR IVEC
FROM THE GF2 FIELD CORRESPONDING TO THE POWER OF THE ELEMENT OF THE FIELD
IPOWER DETERMINES WHICH OF THE 6 BIT VECTORS IS INDEXED
IVEC IS THE VECTOR FROM THE FIELD
COMMON/BLDC31/ ICDEFF(64), IEXP(64)
DIMENSION IVEC(9)
ITEMP=ICDEFF(IPOWER+1)
CALL UNPACK(MBCH, ITEMP, IVEC)
DO 10 I=1, MBCH
IVEC(I)=ICDEFF((MBCH*(IPOWER+1))+I)
CONTINUE
RETURN
END

```

C 100

C 9

C 10

```

SUBROUTINE CHARTS(MBCH, ISD1, ISD3, ISD5)
THIS SUBROUTINE ROUTINE THE APPROPRIATELY ALTERED
VECTOR TO THE POWER SUM
ISD IS JUST IVE*(ALPHA**I/TREE).
COMMON/BLOC30/ICOEFF1(64),ICOEF3(64),ICOEF5(64)
COMMON/BLOC31/ICOEF1(64),ICOEF3(64)
DIMENSION ISD1(1),ISD3(1),ISD5(1)
INDEX1=0
INDEX3=0
INDEX5=0
ICOUNT=2**MBCH
DO 10 I=1,MBCH
INDEX1=((2**((MBCH-I))*ISD1(1))+INDEX1)
INDEX3=((2**((MBCH-I))*ISD3(1))+INDEX3)
INDEX5=((2**((MBCH-I))*ISD5(1))+INDEX5)
CONTINUE
ITEMP1=ICOEF1(INDEX1+1)
ITEMP3=ICOEF3(INDEX3+1)
ITEMP5=ICOEF5(INDEX5+1)
CALL UNPACK(MBCH,ITEMP1,ISD1)
CALL UNPACK(MBCH,ITEMP3,ISD3)
CALL UNPACK(MBCH,ITEMP5,ISD5)
DO 51 I=1,MBCH
ISD5(I)=ICOEF5((MBCH*(INDEX5))+I)
CONTINUE
DO 52 I=1,MBCH
ISD1(I)=ICOEF1((MBCH*(INDEX1))+I)
CONTINUE
DO 54 I=1,MBCH
ISD3(I)=ICOEF3((MBCH*(INDEX3))+I)
CONTINUE
RETURN
END

```

C 33

3

10

C 51
C 41
C 52
C 43
C 54

[illegible]

```

100 SUM(J+1)=ITEMP1
101 ITEMP2=IORDP(J)
102 IORDP(I)=IORDP(J+1)
103 IORDP(J+1)=ITEMP2
104 GO TO 1015
105 CONTINUE
106 WRITE(5,910)(I,IORDP(I),I=1,ICOUNT)
107 FORMAT(1X,6(1X,IORDP(I),13,1),13,2X))
108 WRITE(5,909)(I,SUM(I),I=1,ICOUNT)
109 FORMAT(1X,4(1X,I,1X,I,13,1),13,2X))
110
111 FORMAT(1X,'I,VEC=',1811)
112 THIS SUBROUTINE GENERATES FIVE DIFFERENT CHARTS
113 THE CHARTS ARE APPROPRIATELY WEIGHTED BY ALPHA*KKJ.
114 THE SUM OF THE VECTOR CORRESPONDING TO THIS WEIGHTING IS STORED
115 IN THE CHART AND IS THEN UNPACKED INTO ITS MBCH VECTOR AT
116 A LATER TIME.
117
118 INDEX1=0
119 INDEX3=0
120 INDEX5=0
121 ICOUNT=2*MBCH
122 DO 80J1=1,3
123 J=(J1-1)*X2+1
124 DO 17 ISUN2=1,ICOUNT
125 ISUM1=ISUN2-1
126 IEXP1=IORDP(ISUM1+1)
127 DO 13 I=1,MBCH
128 CALL MODUL(IEXP1,J,ICOUNT-1,IEXP2)
129
130 INSTEAD LET'S STORE THE SUM
131 ITEMP(ISUN2)=ICOEFF(IEXP2+2)
132 ITEMP((ISUM1*MBCH)+I)=ICOEFF(IMBCH*(IEXP2+1))+I)
133 CONTINUE
134 CONTINUE
135 IF(J.EQ.1)GO TO 21
136 IF(J.EQ.2)GO TO 22
137 IF(J.EQ.3)GO TO 23
138 IF(J.EQ.4)GO TO 24
139 DO 38 I=1,ICOUNT
140 DO 38 I=1,ICOUNT*MBCH
141 ICOEF5(I)=ITEMP(I)
142 CONTINUE
143 GO TO 15
144 DO 34 I=1,ICOUNT
145 DO 34 I=1,ICOUNT*MBCH
146 ICOEF1(I)=ITEMP(I)
147 CONTINUE
148 GO TO 15
149 DO 35 I=1,ICOUNT
150 DO 35 I=1,ICOUNT*MBCH
151 ICOEF2(I)=ITEMP(I)
152 CONTINUE
153 GO TO 15
154 DO 36 I=1,ICOUNT
155 DO 36 I=1,ICOUNT*MBCH
156 ICOEF3(I)=ITEMP(I)
157 CONTINUE
158 GO TO 15
159 DO 37 I=1,ICOUNT
160 DO 37 I=1,ICOUNT*MBCH
161 ICOEF4(I)=ITEMP(I)
162 CONTINUE
163 CONTINUE
164 CONTINUE
165

```

RETURN
END

```

BATSD.FIN
MARCH 13, 1979
CONVERT INPUT BINARY VECTOR INTO DECIMAL SIDE INFORMATION
SUBROUTINE OVR11
SUBROUTINE BATSD(NEPB)
COMMON/DITER/OTDCT(256),OTPAR(8),OTDC,OTVAR,OTM,OTG
COMMON/BLOCKS/IBIT(256),IPDEC,INBA(500)
DIMENSION INB(6)
INTEGER OTDCT,OTPAR,OTDC,OTVAR,OTM,OTG
READ PARCOR(1),1=1,8
NUV=IPDEC
NEPA=3
NBA=0
DO 10 I=1,8
  IDT=6-(I+1)/2
  DO 20 J=1,IDT
    NGA=NB+1
    INB(J)=INBA(NBA)
    CALL BIDCONU(INB,IDT,OTPAR(1))
    CONTINUE
  DO 30 J=1,5
    NBA=NB+1
    INB(J)=INBA(NBA)
    CALL BIDCONU(INB,5,OTVAR)
    READ U/UU
    NBA=NB+1
    IPDEC=INBA(NBA)
    READ DCBIAS
    OTDC=0
    IF(NEPB.LE.0)GO TO 60
    DO 40 I=1,NEPB
      IT=5-I
      ITT=2*IT
      NBA=NB+1
      OTDC=OTDC+INBA(NBA)*ITT
    CONTINUE
    ADD REMAINDER OF DCBIAS
    NDCCR=5-NEPB
    IF(NDCCR.EQ.0)GO TO 80
    NDCPC=NEPB*63
    IF(NEPB.EQ.0.AND.NUV.EQ.0)NDCPC=34
    IF(NEPB.EQ.0.AND.NUV.EQ.1)NDCPC=42
    DO 70 I=1,NDCCR
      IP=NDCPC+1
      INB(I)=INBA(IP)
    CONTINUE
    CALL BIDCONU(INB,NDCCR,IBPG)
    OTDC=OTDC+IBPG
  CONTINUE
  IF(IPDEC.EQ.0)RETURN
  DO 50 J=1,5
    NBA=NB+1
    INB(J)=INBA(NBA)
    CALL BIDCONU(INB,6,OTM)
    NBA=NB+1
    INB(1)=INBA(NBA+1)
    IF(INBA.GT.45)INB(1)=INBA(NBA+18)
    NBA=NB+1
    INB(2)=INBA(NBA)
    IF(INBA.GT.45)INB(2)=INBA(NBA+18)
    CALL BIDCONU(INB,2,OTG)

```

```

END
SUBROUTINE BDCUNU TO TAKE A BINARY VECTOR TO A DECIMAL NUMBER
SUBROUTINE BDCUNU(INB,LIB,IY)
  DIMENSION INE(1)
  IY=0
  IF(LIB.LE.0)RETURN
  DO 10 I=1,LIB
    IT=2** (I-1)
    IY=IY+INB(LIB+1-I)*IT
  RETURN
END

```

2

10

```

C      CBADCT.FTM
C      THIS RETURNS THE DECIMAL VALUE OF THE DCT COEFFICIENTS
C      FROM THE CODED AND (PACKED BINARY VECTOR INBA
C      SUBROUTINE QUP12(NBPF)
C      SUBROUTINE SBADCT(NBPF)
C      COMMON/SORT/CT1(256),DCT2(256),IORDR(256),XR(512),XI(512)
C      COMMON/BACK1/INIT1,DCT(256),LTH,NSTAGE
C      COMMON/BLOCK6/IBIT(256),IPDEC,INBA(500)
C      DIMENSION INB(6)
C      IBIT(1),1-1,LTH ARE BIT ASSIGNMENTS VECTOR
C      DCT(1),1-1,LTH WILL BE DCT QUANTIZER LEVEL IN REAL FORMAT.
C      THEN THE DCT WILL BE INTEGERIZED AND PASSED TO ORDCT
C      NEPB=3
C      NBPF IS THE TOTAL NUMBER OF BITS PER FRAME PLUS THE PARITY BITS
C      NBPF=NBPF+3*18
C      TYPE 707,NBPF,NBPF
C      FORMAT(1X,'NBPF=',14,3X,'NEPB=',1,14)
C      WRITE(4,808)(1,INBA(1),1-1,NBPF)
C      FORMAT(1X,'8(2X,13,2X,',2X,13))
C      NUU=IPDEC
C      NEA=34+NEPB
C      IF(NUU.EQ.1)NBA=NEA+8
C      NEPT=3X45-3-34
C      IF(NUU.EQ.1)NBPT=NEPT-8
C      INI XI(1),DCT(1),1-1,LTH
C      DO 10 I=1,LTH
C      DCT(I)=0.1
C      XI(I)=FLOAT(1BIT(I))+0.1
C      IF(NEPB.LE.0)GO TO 120
C      REMOVE REDUNDANT BIT FROM INBA(I) VECTOR
C      NF1=NBPF
C      DO 40 J=1,3
C      NF1=NF1-18
C      NI1=J*45+1
C      DO 20 I=NI1,NF1
C      INBA(I)=INBA(I+18)
C      CONTINUE
C      WRITE(4,909)(1,INBA(1),1-1,NF1)
C      FORMAT(1X,'8(2X,13,2X,',2X,13))
C      FIND NEW BIT ASSIGNMENTS AND READ SCRAMBLED DATA
C      INP=0
C      IBST=IBIT(1)
C      CONTINUE
C      DO 100 I=1,LTH
C      IAB=IBIT(I)
C      IF(IAB.LT.IBST)GO TO 130
C      IF(IAB.EQ.0)GO TO 100
C      INP=INP+1
C      NBA=INBA+1
C      ITT=2X(IAB-1)
C      IBIT(I)=IBIT(I)-1
C      DCT(IORDR(I))=DCT(IORDR(I))+FLOAT(INBA(NBA)*ITT)
C      IF(INP.GE.NEPT)GO TO 130
C      CONTINUE
C      IBST=IBST-1
C      GO TO 110
C      CONTINUE
C      READ DCBIAS
C      NBA=90
C      REMEMBER TO MOVE NBA POINTER 2 BITS TO ACCOUNT
C      FOR THE DC BIAS BITS YOU ALREADY PLUCKED OUT
C      NBA=NBA+2
C      IBIT(1),1-1,LTH ARE REDUCED BIT ASSIGNMENT
C      READ DCT(1),1-1,LTH FROM INBA AND XI

```

```

C      CONVERT INBA INTO DCT
DO 500 I=1,LTH
  IDT=IBIT(I)
  IF(IDT.LE.0)GO TO 500
  READ INPUT VECTOR
  DO 600 J=1,IDT
    NBS=NBA+1
    INB(J)=INBA(NBA)
    CALL BDCONJ(INB,IDT,IY)
    DCT(IORDR(I))=DCT(IORDR(I))+FLOAT(IY)
  CONTINUE
  TYPE911,NBA
  FORMAT(IX,'NBA IN DESER. ROUTINE =',I4)
  NOW SORT DCT INTO AN ORDERED DCT2 ARRAY
  DO 505 I=1,LTH
    DCT2(I)=DCT(IORDR(I))
  DO 510 I=1,LTH
    IBIT(I)=IFIX(XI(I))
  RETURN
END
SUBROUTINE BDCONJ(INB,LIB,IY)
  DIMENSION INB(1)
  IY=0
  IF(LIB.LE.0)RETURN
  DO 10 I=1,LIB
    IT=2*(I-1)
    IY=IY+INB(LIB+1-I)*IT
  RETURN
END

```

```

MARCH 16, 1979
CHANNEL ERROR SIMULATION ROUTINE
SUBROUTINE OUR13(NBRPF,PROB,IRN,JRN,NERB,NEPB)
SUBROUTINE CETR(INBA,NBRPF,PROB,IRN,JRN,NERB,NEPB)
COMMON/BLOCK6/IBIT(255),IPDEC,INBA(500)
COMMON/SA/ICOUNT,IPR34
DIMENSION NERB(6)
TYPE BUS,NBRPF,NEPB
FORMAT(1X,I4,2X,I4)
WRITE(5,809)(INBA(I),I=1,NBRPF)
FORMAT(1X,63(11))
NEPB1=NEPB+1
DO 50 I=1,NEPB1
  NERB(I)=0
  XMIT INPUT BINARY VECTOR
  IF(NERB.LE.0)GO TO 40
  DO 30 J=1,NEPB
    DO 10 I=1,63
      ISU1=NERB(J)
      IP=I+63X(J-1)
      IN=INBA(IP)
      INERB=NERB(J)
      CALL RANERR(IM,PROB,IRN,JRN,INERB)
      INBA(IP)=IN
      NERB(J)=INERB
      IF(ISU1.NE.NERB(J))WRITE(5,100)ICOUNT,I
    CONTINUE
  CONTINUE
  CONTINUE
  RETURN
  ISTP=NEPB*63+1
  DO 20 I=ISTP,NBRPF
    ISU1=NERB(NEPB1)
    IM2=INBA(I)
    INERB2=NERB(NEPB1)
    CALL RANERR(IM2,PROB,IRN,JRN,INERB2)
    INBA(I)=IM2
    NERB(NEPB1)=INERB2
    IF(ISU1.NE.NERB(NEPB1))WRITE(5,100)ICOUNT,I
  CONTINUE
  FORMAT(1X,'FR=',I4,2X,'ERR LC=',I3)
  WRITE(5,809)(INBA(I),I=1,NBRPF)
  RETURN
END
RANERR_FTN
SUBROUTINE RANERR(IX,PROB,IRN,JRN,NER)
CALL RANDU(IRN,JRN,YOR)
IF(YOR.GE.PROB)RETURN
IX=IEOR(IX,1)
NER=NER+1
RETURN
END

```

```

SUBROUTINE OVERD(LTH1,LTH2,LTH3,LTH4,IMARK)
SUBROUTINE EVGDD(LTH1,XR,XI,ITWID,IMARK)
WE WILL TAKE A REAL SIGNAL X AND DO AN N/2 PT
COMPLEX FFT ON X BY BREAKING THE SIGNAL INTO ITS EVEN AND ODDPOINTS.
THEN THE PROPERTIES OF EVEN-
ODD SEPARATION WILL BE USED TO GET XR,XI THE PEAL
AND IMAGINARY COMPONENTS OF THE FFT OF THE ORIGINAL SIGNAL

ITWID JUST DETERMINES WHETHER THE FORWARD FFT IS DONE
OR THE INVERSE FFT. IF ITWID EQUALS ONE THE FORWARD FFT IS RETURNED.
COMMON/SORT/DCT1(256),DCT2(256),IDPDR(256),XR(512),XI(512)
DIMENSION XR(1),XI(1)
LTH=2*LTH1
LTH2=2*LTH+1
LTH3=LTH+2
LTH4=LTH+2
NSTAGE IS THE NUMBER OF STAGES FOR THE FFT
PI=4.*XATAN(1./2.)
PI2=2*PI/LTH
PI3=PI/2/4
NOW BREAK INTO EVEN AND ODD COMPONENTS
ALSO SCALE BY2 TO ADJUST OUTPUT
DO 11 J=1,LTH1
XR(J)=XR(2*XJ-1)/2.0
XI(J)=XR(2*XJ)/2.0
CONTINUE
NOW LOAD INTO FFT AND THEN RESHUFFLE
CALL FASTF(NSTAGE,1,1,2)
XTMP1=2.0*(XR(1)+XI(1))
XR(LTH+1)=2.0*(XR(1)-XI(1))
XTMP2=2.0*(XR(1)-XI(1))
XR(1)=XTMP1
XI(1)=0.0
XI(LTH+1)=-2.0*(XI(LTH+1))
IF(IMARK.EQ.2)GO TO 12
XR(LTH+1)=XTMP2
XR(LTH1+LTH+1)=XR(LTH+1)
XI(LTH1+LTH+1)=0.0
XI(LTH1+LTH+1)=-1.0*(XI(LTH+1))
DO 25 K=2,LTH+1
J=K-1
Q1=(XI(K)+XI(LTH2-K))
Q2=(XI(K)-XI(LTH2-K))
Q3=(XR(K)+XR(LTH2-K))
Q4=(XR(K)-XR(LTH2-K))
OTEMP1=((SIN(PI/2*K))*Q4)
OTEMP2=((SIN(PI/2*K))*Q1)
OTEMP3=((COS(PI/2*K))*Q1)
OTEMP4=((COS(PI/2*K))*Q4)
XR(K)=Q3-OTEMP1+OTEMP3
XR(LTH2-K)=Q3+OTEMP1-OTEMP3
XI(K)=Q2-OTEMP4-OTEMP2
XI(LTH2-K)=-1.0*(Q2+OTEMP4+OTEMP2)
IF(IMARK.EQ.2)GO TO 25
XR(LTH3-K)=XR(K)
XR(LTH3+K)=XR(LTH2-K)
XI(LTH3-K)=-1.0*(XI(K))
XI(LTH1+K)=-1.0*(XI(LTH2-K))
CONTINUE
IF(ITWID.EQ.1)GO TO 30

```

QUESTION ()

(1)

11

12

6

C

K

NON DO COMPLEX CONJUGATE AND NORMALIZE
BY LTH FOR THE INVERSE FFT.

ALTH=-1.0/ALTH
DO 26 I=1,LTH
XP(I)=XR(I)*ALTH
XI(I)=XI(I)/ALTH
CONTINUE
RETURN
END

```

PASTF.FTN      14-FEB-78
SUBROUTINE PASTF(PI,TYPE,IS,IMARK)
  NP = LUG NUMBER OF SAMPLES, MAX=10.
  JTYPE=1 IF BEST TRANSFORM, JTYPE=2 IF REVERSE TRANSFORM.
  IS = 1 IF COEFFICIENT TABLE SET-UP REQUIRED.
  IS = 2 IF COEFFICIENT TABLE BY-PASS REQUIRED.
  COMMON/SORT/DCT1(256),DCT2(256),IDR(256),X(S12),Y(S12)
  COMMON/BLOCK/INIT1,DCT1(56),LTH,NESTAGE
  DIMENSION S(256),KX(11)
  REAL*8 PIHAF
  EQUIVALENCE (KX(1),Y10),(KX(2),K9),(KX(3),K8),(KX(4),K7)
  EQUIVALENCE (KX(5),K6),(KX(6),K5),(KX(7),K4),(KX(8),K3),(KX(9),K2)
  EQUIVALENCE (KX(10),K1)
  N=2000
  IF(IMARK.EQ.2)GO TO 1200
  WRITE(5,934)(I,X(I),I=1,4)
  WRITE(5,934)(I,Y(I),I=1,4)
  FORMAT(1X,4(1X,'FT('',13,'')=',E15.8,2X))
  N4=N/4
  GO TO (1,3),IS
  S(N4)=1.
  IF(N4-1)3,3,5
  N8=N/8
  PI=4.*ATAN(1.0)
  PIHAF=PI/2.
  S(N8)=DSIN(PIHAF/2.)
  IF(N8-1)3,3,6
  N16=N/16
  S(N16)=DSIN(PIHAF/4.)
  N16=N8*N16
  S(N16)=DCDS(PIHAF/4.)
  IF(N16-1)3,3,7
  N=N8-2
  LX=8
  DO 10 L=3,11
    I=N4/LX
    S(I)=DSIN(PIHAF/LX)
    IC=N4-I
    S(IC)=DCDS(PIHAF/LX)
    KMAX=LX/2-2
    LX=LXX2
    DO 10 K=1,KMAX
      KI=(2K+1)*I
      KID=N4-KID
      S(KI)=S(I)*S(KIDC)+S(IC)*S(KID)
      PX=1
      DO 100 M=1,NP
        INCR=PX
        NINCL=-INCR
        PX=PX+X2
        JMAX=N/16
        IDEL=2XJMAX
        DO 100 J=1,JMAX
          NINCL=NINCL+INCR
          DO 100 I=J,N,IDEL
            IJMAX=1+IJMAX
            XTEN=X(I)-X(IJMAX)
            YTEM=Y(I)-Y(IJMAX)
            X(I)=X(I)+X(IJMAX)
            Y(I)=Y(I)+Y(IJMAX)
            IF(J-1)50,50,51
            X(IJMAX)=XTEN
            Y(IJMAX)=YTEM
          50
        51
      100
    10
  3
  10
  3
  50

```

```

51 GO TO 100
52 IF (NANGL-N4)53,52,54
53 GO TO (60,61),JTYPE
X(IJ*AX)=YTEM
Y(IJ*AX)=-XTEM
GO TO 100
54 X(IJ*AX)=-YTEM
Y(IJ*AX)=XTEM
GO TO 100
55 SN=S(NANGL)
NCOS=N4-NANGL
CS=S(NCOS)
GO TO 75
56 NSIN=2*N4-NANGL
SN=S(NSIN)
NCOS=N4-NSIN
LS=-S(NCOS)
GO TO (90,91),JTYPE
75 X(IJ*AX)=XTEM*CS+YTEM*SN
X(IJ*AX)=-XTEM*SN+YTEM*CS
GO TO 100
91 X(IJ*AX)=XTEM*LS-YTEM*SN
Y(IJ*AX)=XTEM*SN+YTEM*LS
CONTINUE
REORDER RESULTS IN NATURAL ORDER
KX(1)=N
DO 22 L=2,NP
KX(L)=KX(L-1)/2
DO 24 L=NP,9
KX(L+1)=1
IJ=1
DO 30 J1=1,K1,1
DO 30 J2=J1,K2,K1
DO 30 J3=J2,K3,K2
DO 30 J4=J3,K4,K3
DO 30 J5=J4,K5,K4
DO 30 J6=J5,K6,K5
DO 30 J7=J6,K7,K6
DO 30 J8=J7,K8,K7
DO 30 J9=J8,K9,K8
DO 30 J1=J9,K10,K9
IF (IJ-J1)28,29,29
XTEM=X(IJ)
X(IJ)=X(J1)
X(J1)=XTEM
YTEM=Y(IJ)
Y(IJ)=Y(J1)
Y(J1)=YTEM
GO TO (30,31),JTYPE
31 X(IJ)=X(IJ)/N
Y(IJ)=Y(IJ)/N
IJ=IJ+1
30 K-TH SPECTRAL VALUE IS STORED IN X(K+1),Y(K+1),K=0,N-1.
IF (IMARK.EQ.2)RETURN
WRITE(S,934)(I,X(I),I=1,4)
WRITE(S,934)(I,Y(I),I=1,4)
RETURN
END

```

```

GFZADD,FTI
ADDITION OVER GF(2)
POLYNOMIAL AX: MUST BE ORDERED IN DESCENDING POWER SERIES
SUBROUTINE GFZADD(ITA,NA,INB,NB,IPC,INC)
DIMENSION ITA(1),INB(1),INC(1)
NC=NA
IF(NB.GT.1)NC=NB
DO 10 I=1,NC
IC=NC+1-I
IRA=NC+1-I
IRB=NB+1-I
ITA=0
ITB=0
IF(IRA.GT.0)ITA=ITA+IRA)
IF(IRB.GT.0)ITB=ITB+IRB)
INC(IC)=IEOR(ITA,ITB)
CONTINUE
RETURN
END

```

```

GF2M1.FTN
MULTIPLICATION OVER GF(2)
N=NF.NB<NF
SUBROUTINE GF2M1(INA,NA,INB,NB,INC,NC,INF,NF)
DIMENSION IAT(17),INA(1),INB(1),INC(1),INF(1)
NCC=NA+NB-1
III VECTOR C
DO 10 I=1,NCC
IAT(I)=0
MULTIPLY A AND B
DO 20 I=1,NA
DO 30 J=1,NB
IC=I+J-1
IT=IAND(INA(I),INB(J))
IAT(IC)=IEOR(IAT(IC),IT)
CONTINUE
CONTINUE
CALL GF2DIV(IAT,NCC,INF,NF,INC,NC)
RETURN
END

```

```

GF2DIV.FTN
FINITE FIELD DIVISION
DIVISOR VECTOR B IS DESTROYED IN COMPUTATION IF NOT NORMALIZED
SUBROUTINE GF2DIV(INA,NA,INB,NB,ITC,NC)
DIMENSION INA(1),INB(1),INC(1)
NORMALIZE VECTOR B
NC=NB-1
NBP=NB
DO 11 I=1,NB
IF(INB(I).EQ.1)GO TO 22
NBP=NBP-1
IF(NBP.LE.0)GO TO 11
DO 33 J=1,NBP
INB(J)=INB(J)+1
CONTINUE
VECTOR B=0
WRITE(5,100)
FORMAT(1X,'DIVISOR=0')
RETURN
CONTINUE
IF(NA.GE.NBP)GO TO 10
INA(1) IS THE ANSWER
DO 20 I=1,NC
IR=NC+1-I
INC(IR)=0
IF(1.GT.NA)GO TO 20
ITA=NA+1-I
INC(IR)=INA(ITA)
CONTINUE
RETURN
CONTINUE
INI VECTOR C
ACTUAL NA MAY BE SMALLER THAN NBP
DO 30 I=1,NBP
INC(I)=INA(I)
NAP=NBP
CONTINUE
CHECK C(1)=1
IF(INC(1).EQ.0)GO TO 222
START DIVISION
DO 50 I=1,NBP
INC(I)=IEOR(INC(I),INB(I))
CONTINUE
NAP=NAP+1
IF(NAP.GT.NA)GO TO 333
SHIFT ONE BIT LEFT
DO 60 I=1,NBP-1
INC(I)=INC(I+1)
INC(NBP)=INA(NAP)
GO TO 111
CONTINUE
INSERT 0 IF NBP NOT EQUAL TO NB
IF(NBP.NE.NB)GO TO 777
DO 555 I=1,NC
INC(I)=INC(I+1)
RETURN
CONTINUE
DO 773 I=1,NC
IT=0
IR=NC+1-I
IEC=NBP+1-I
IF(1.GT.NET)GO TO 773
IT=INC(IEC)
INC(IR)=IT

```

RETURN
END

```

SUBROUTINE TAPE3(10)
IMPLICIT INTEGER(A-Z)
COMMON/TAPE0/NIN(256),NOUT(256)
COMMON/TAPE1/NSKIP,IST,NTOTI,NTUPS,NTOTO
COMMON/TAPE2/NEPR,NFILE,NINS,NOUTS
COMMON/TAPE3/NBF(1324),NBUF(1324)
COMMON/TAPE4/LST,IBEG
DIMENSION ICARD(64)
GO TO (700,900,900,999,1000,1001,1002,1003),10

C
700 INPUT DATA
   IST=IST+NTUPS
   IF(1324-GE.1ST) GO TO 200
100 IST=1ST-1024
   NSKIP=NSKIP+1
300 KOVER=LST+NTOTI-1325
   IF(KOVER.GT.0) GO TO 305
   DO 5000 I=1,NTOTI
3000 NIN(I)=NBF(LST+I-1)
   LST=LST+NTUPS
   RETURN
305 IPEP=LST-1024
   DO 5001 I=1,300
5001 NBF(IPEP+I-1)=NBF(LST+I-1)
   LST=IPEP
   IF(NINS.EQ.0) GO TO 2100
   DISK INPUT
   DO 5010 I=1,16
   READ(2,END=2001,ERR=6000)(ICARD(J),J=1,64)
   K=64*(I-1)+300
   DO 5010 J=1,64
5010 NBF(K+J)=ICARD(J)
   IF(NERR.NE.0) GO TO 6000
   GO TO 200
   NEND=1
2001 GO TO 200
2100 CALL TTN(NBF(301),NEND,NERR)
   GO TO 200

C
600 CONTINUE
C
5005 OUTPUT DATA
   DO 5005 I=1,NTOTO
   NBUF(1BEG+I-1)=NOUT(I)
   1BEG=1BEG+NTOTO
   KON=1025-13EG
   IF(KON.GT.0) GO TO 90
   IF(NOUTS.EQ.0) GO TO 2002
   DO 6010 I=1,16
   K=64*(I-1)
   DO 6011 J=1,64
6011 ICARD(J)=NBUF(K+J)
6010 WRITE(3)(ICARD(J),J=1,64)
   GO TO 2003
2002 CALL TOUT(NBUF,NEPR)
2003 LRESID=-KON
   DO 5006 I=1,LRESID
5006 NBUF(I)=NBUF(1024+I)
   1BEG=LRESID+1
   RETURN
90
C
INITIALIZE
   IF(NINS+NOUTS).LE.1) CALL ATTACH
   IF(NINS.EQ.0) CALL RANDO
   IF(NOUTS.EQ.0) CALL FIND1

```

```

913      IF(NFILE.EQ.0) GO TO 955
          IF(NINQ.EQ.0) CALL FSRCH(NFILE,NERR)
          IF(NERR.NE.0) GO TO 6000
          DO 912 J=1,NSKIP
          IF(NINS.EQ.0) GO TO 3000
          DISK INPUT
          DO 5011 I=1,16
          READ(2,END=3001,ERR=6000)(ICARD(JJ),JJ=1,64)
          K=64X(1-1)+300
          DO 5011 JJ=1,64
          NBF(K+JJ)=ICARD(JJ)
          GO TO 912
          NEND=1
          GO TO 912
3000      CALL TIN(NBF(301),NEND,NERR)
          IF(NERR.NE.0) GO TO 6000
          912 CONTINUE
          GO TO 955
999      CONTINUE
          IBEG=1
          CALL EDFSH(NERR)
          RETURN
995      CONTINUE
          IBEG=1
          LST=1ST+300
          1ST=1ST-NTUPS
          RETURN
          END OF FILE
C
1000      IF(NOUTS.EQ.0) GO TO 2010
          CALL CLOSE(3)
          GO TO 2011
2010      CALL EDFSH(NERR)
          IBEG=1
          RETURN
          NEND=0
          IF(NINS.EQ.0) GO TO 4020
          REWIND 2
          GO TO 4011
          CALL RAND0
          NERR=0
          CALL FSRCH(NFILE,NERR)
          IF(NERR.NE.0) GO TO 6000
          DO 950 J=1,NSKIP
          IF(NINS.EQ.0) GO TO 4000
          DISK INPUT
          DO 5012 I=1,16
          READ(2,END=4001,ERR=6000)(ICARD(J1),J1=1,64)
          K=64X(1-1)+300
          DO 5012 J2=1,64
          NBF(K+J2)=ICARD(J2)
          GO TO 950
          NEND=1
          GO TO 4002
          CALL TIN(NBF(301),NEND,NERR)
          IF(NERR.NE.0) GO TO 6000
          IF(NEND.NE.0) GO TO 2000
          950 CONTINUE
          LST=1ST+300
          1ST=1ST-NTUPS
          RETURN
          NERR=16384
          RETURN
          IF(NINS.EQ.1) CALL CLOSE(2)
          NEND=0
          RETURN

```

```

1003 CALL INFO
      RETURN
1000 TYPE 5001
1001 FORMAT(IX,'INPUT FILE ERROR')
      NERR=1
      RETURN
      END

PROGRAM FSIO.FTN TO MOVE MAG TAPES
AND WRITE SPEECH FOR REAL-TIME I/O USING QIO

SUBROUTINE ATTACH
IMPLICIT INTEGER(A-Z)
COMMON/TAPES/END,NERR,NFILE,NINS,NOUTS
COMMON/TAPES/MASK,ISW(2),IOATT,IOSUC,IEALN,IORAD,
IOALB,IEVER,IOSPF,IEEOF,IOEPF,IOPLB,MT0(6),MT1(6),DSW
DATA IOATT,IOSUC,IEALN/0,0,1,0,0,0/
DATA IORAD,IOALB,IEVER,IOSPF,IOPLB/0,0,0,0,0,0/
DATA IOSPF,IEEOF,IOEPF/0,0,0,0,0,0/
DATA MASK/0,0,0,0,0,0/
DATA MT0/0,0,0,0,0,0,0,0,0,0,0,0/
DATA MT1/0,0,0,0,0,0,0,0,0,0,0,0/
IF(NINS.NE.0) GO TO 1
CALL ASNLIN(2,'MT',0,DSW)
IF(DSW.EQ.1)GO TO 10
WRITE(5,100)
FORMAT(IX,'MT0: ATTACH UNSUCCESSFUL')
NERR=1
RETURN
100
10 CALL MT0IO(IOATT,2,1,0,ISW,0,DSW)
IF(IOSUC.EQ.1)AND(MASK,ISW(1)))GO TO 1
IF(IAND(IEALN,MASK).NE.IAND(MASK,ISW(1)))GO TO 11
IF(NOUTS.NE.0) GO TO 2
CALL ASNLIN(3,'MT',1,DSW)
IF(DSW.EQ.1) GO TO 20
WRITE(5,101)
FORMAT(IX,'MT1: ATTACH UNSUCCESSFUL')
NERR=1
RETURN
101
20 CALL MT0IO(IOATT,3,1,0,ISW,0,DSW)
IF(IOSUC.EQ.1)AND(MASK,ISW(1)))GO TO 2
IF(IAND(IEALN,MASK).NE.IAND(MASK,ISW(1)))GO TO 12
RETURN
2
END

SUBROUTINE RUND0
IMPLICIT INTEGER(A-Z)
COMMON/TAPES/END,NERR,NFILE,NINS,NOUTS
COMMON/TAPES/MASK,ISW(2),IOATT,IOSUC,IEALN,IORAD,
IOALB,IEVER,IOSPF,IEEOF,IOEPF,IOPLB,MT0(6),MT1(6),DSW
1 CALL MT0IO(IORAD,2,1,0,ISW,0,DSW)
IF(IOSUC.EQ.1)AND(MASK,ISW(1)))GO TO 1
WRITE(5,902)
FORMAT(IX,'MT0: BUSY')
NERR=1
RETURN
902
1
END

SUBROUTINE RUND1
IMPLICIT INTEGER(A-Z)
COMMON/TAPES/END,NERR,NFILE,NINS,NOUTS
COMMON/TAPES/MASK,ISW(2),IOATT,IOSUC,IEALN,IORAD,
IOALB,IEVER,IOSPF,IEEOF,IOEPF,IOPLB,MT0(6),MT1(6),DSW
1 CALL MT0IO(IORAD,3,1,0,ISW,0,DSW)
IF(IOSUC.EQ.1)AND(MASK,ISW(1)))GO TO 1
WRITE(5,902)

```

```

902 NERR=1
1   FORMAT(1X,'MT1: BUSY'//)
   C   RETURN
   C   END

1   SUBROUTINE TINCBUF,NEND,NERR)
   C   IMPLICIT INTEGER(A-Z)
   C   COMMON/MTAPES/MASK,ISW(2),IOATT,IOSUC,IEALN,IORAD,
   C   IOALB,IEVER,IOSPF,IEEOF,IOEUF,IORLB,MT0(6),MT1(6),DSW
   C   NEND=0
   C   NERR=0
   C   CALL GETADR(MT0,BUF)
   C   CALL MTGIO(IOALB,2,1,0,ISW,MT0,DSW)
   C   IF(IOSUC.EQ.IAND(MASK,ISW(1)))GO TO 1
   C   IF(IAND(IEEOF,MASK).EQ.IAND(MASK,ISW(1)))NEND=1
   C   IF(IAND(IEVER,MASK).EQ.IAND(MASK,ISW(1)))NERR=1
   C   RETURN
   C   END

1   SUBROUTINE TOUT(NBUF,NERR)
   C   IMPLICIT INTEGER(A-Z)
   C   COMMON/MTAPES/MASK,ISW(2),IOATT,IOSUC,IEALN,IORAD,
   C   IOALB,IEVER,IOSPF,IEEOF,IOEUF,IORLB,MT0(6),MT1(6),DSW
   C   NERR=0
   C   CALL GETADR(MT1,NBUF)
   C   CALL MTGIO(IOALB,3,1,0,ISW,MT1,DSW)
   C   IF(IOSUC.EQ.IAND(MASK,ISW(1)))GO TO 1
   C   IF(IAND(IEVER,MASK).EQ.IAND(MASK,ISW(1)))NERR=1
   C   RETURN
   C   END

1   SUBROUTINE FSRCH(NFILE,NERR)
   C   IMPLICIT INTEGER (A-Z)
   C   COMMON/MTAPES/NBF(1324),NBUF(1324)
   C   COMMON/MTAPES/MASK,ISW(2),IOATT,IOSUC,IEALN,IORAD,
   C   IOALB,IEVER,IOSPF,IEEOF,IOEUF,IORLB,MT0(6),MT1(6),DSW
   C   NERR=0
   C   FILE=NFILE-1
   C   IF(FILE.LE.0) RETURN
   C   DO 1 1-1.FILE
   C   CALL GETADR(MT0,NBF(301))
   C   CALL MTGIO(IOALB,2,1,0,ISW,MT0,DSW)
   C   IF(IOSUC.EQ.IAND(MASK,ISW(1)))GO TO 2
   C   WRITE(5,100)NFILE
   C   FORMAT(1X,'FILE',14,' NOT FOUND'//)
   C   NERR=1
   C   RETURN
   C   MT0(1)=1
   C   CALL MTGIO(IOSPF,2,1,0,ISW,MT0,DSW)
   C   RETURN
   C   END

1   SUBROUTINE EDFSH(NERR)
   C   IMPLICIT INTEGER(A-Z)
   C   COMMON/MTAPES/NBF(1324),NBUF(1324)
   C   COMMON/MTAPES/MASK,ISW(2),IOATT,IOSUC,IEALN,IORAD,
   C   IOALB,IEVER,IOSPF,IEEOF,IOEUF,IORLB,MT0(6),MT1(6),DSW
   C   NERR=0
   C   CALL GETADR(MT1(1),NBUF(1))
   C   CALL MTGIO(IOALB,3,1,0,ISW,MT1,DSW)
   C   IF(IOSUC.EQ.IAND(MASK,ISW(1)))GO TO 1
   C   IF(IAND(IEEOF,MASK).EQ.IAND(MASK,ISW(1)))GO TO 2
   C   NERR=1
   C   WRITE(5,1000)ISW(1)
   C   FORMAT(1X,'FILE SEARCH ERROR' 09//)

```

```

2      RETURN
      CALL GETADR(MT1(1),NBUF(1))
      CALL MTQIO(IORL,B,3,1,0,ISJ,MT1,DSJ)
      IF(IOSUC.EQ.IAND(MASK,ISJ(1)))GO TO 1
      IF(IAND(IEOF,MASK).EQ.IAND(MASK,ISJ(1))) GO TO 3
      NERR=1
      RETURN
3      MT1(1)=1
      CALL MTQIO(IOSPF,3,1,0,ISJ,MT1,DSJ)
      RETURN
      END
C      SUBROUTINE EDRAW(NERR)
      IMPLICIT INTEGER(A-Z)
      COMMON/TAPES/MASK,ISJ(2),IOPAT,IOSUC,IORL,IORAD,
1      IOLB,IEVER,IOSPF,IEEOF,IOEOF,IORL,MT0(6),MT1(6),DSJ
      NERR=0
      DO 1 I=1,2
      CALL MTQIO(IOEOF,3,1,0,ISJ)
      IF(IOSUC.EQ.IAND(MASK,ISJ(1)))GO TO 2
      NERR=1
      RETURN
2      MT1(1)=1
      CALL MTQIO(IOSPF,3,1,0,ISJ,MT1,DSJ)
      IF(IOSUC.EQ.IAND(MASK,ISJ(1)))RETURN
      NERR=1
      RETURN
      END
D      SUBROUTINE INFO.FTN
      SUBROUTINE INFO
      IMPLICIT INTEGER(A-Z)
      COMMON/TAPES/MIN(256),NOUT(256)
      COMMON/TAPES1/NSKIP,IST,NTOT1,NTUPS,NTOTO
      COMMON/TAPES2/NEID,NERR,NFILE,NINS,NOUTS
      COMMON/TAPES3/NBF(1324),NBUF(1324)
      LOGICAL X1,Y,ANS
      DIMENSION EXTIN(3),EXTOUT(3)
      DATA EXTIN/1,'N','P',/
      DATA EXTOUT/0,'U','T',/
      DATA Y/'Y',/
      DATA NEND,NERR,NFILE/0,0,0,/
      DATA NSKIP,IST/1,1/
      GET I/O INFORMATION FOR SPEECH HANDLER
      APPEND=,FALSE.
      TYPE 1
      FORMAT(1H$, 'IS THE INPUT ON MAG. TAPE? ')
      READ(5,2)ANS
      FORMAT(A1)
      IF(ANS.NE.Y)NINS=1
      TYPE 3
      FORMAT(1H$, 'IS THE OUTPUT GOING TO MAG TAPE? ')
      READ(5,2)ANS
      IF(ANS.NE.Y)NOUTS=1
      TYPE 4
      IF(NOUTS.NE.0) GO TO 5
      TYPE 4
      FORMAT(1H$, 'APPEND DATA? ')
      READ(5,2)ANS
      IF(ANS.EQ.Y) APPEND=,TRUE.
      IF(NOUTS.EQ.0) GO TO 15L

```

```

150 151 100 101 155 113 14 152 39 1600 151 1601 1603 1602 103
RXS11 SUPPORTED FILE
TYPE 150
FORMAT(1H$, 'OUTPUT FILE NAME= ')
CALL FILEN(3,EXTOUT)

BEGINNING OF INPUT

IF(NINS.NE.0) GO TO 155
TYPE 100
FORMAT(1H$, 'MT FILE NO.=(13) ')
READ(5,101)NFILE
FORMAT(13)
GO TO 14
GO TO 14
TYPE 13
FORMAT(1H$, 'INPUT FILE NAME= ')
CALL FILEN(2,EXTIN)
NFILE=1
CALL TAPE3(3)
IF(NERR.NE.0) RETURN
IF(APPEND)CALL TAPE3(4)
RETURN
END

SUBROUTINE FILEN(UNIT,EXT)

THIS SUBROUTINE ACCEPTS THE NAME OF
FORM THE TTY DEVICE 5
DEFAULT DEVICE
UNLESS SPECIFIED IN INPUT STRING
UNIT=UNIT NUMBER
EXT = LOGICAL*1 BUFFER OF EXTENSION

IMPLICIT INTEGER(A-Z)
LOGICAL*1 INSTR,DOT,BLNK,EXT
DIMENSION INSTR(40)
DIMENSION EXT(3)
DATA BLNK,DOT,' ','.'/

INPUT FILE
READ (5,99)(INSTR(I),I=1,40)
FORMAT(40H1)
CHECK FOR END OF LINE
DO 1600 I=40,1,-1
J=1
IF(INSTR(I).NE.BLNK)GO TO 1601
TYPE 151
FORMAT(1H$, ' > ')
GO TO 152
DO 1602 I=1,J
IF(INSTR(I).NE.BLNK) GO TO 1602

BLNK DISCOVERED-COLLAPSE LINE BY 0

DO 1603 K=1,J-1
INSTR(K)=INSTR(K+1)
INSTR(J)=BLNK
J=J-1
GO TO 1601
CONTINUE
DO 103 I=1,J
IF(INSTR(I).EQ.DOT) GO TO 25
INSTR(J+1)=DOT
INSTR(J+2)=EXT(1)
INSTR(J+3)=EXT(2)

```

```

INSTR(J+4)=EXT(3)
J=J+4
CALL SCAN(INSTR,J)
CALL ASSIGN(UNIT,INSTR,J)
RETURN
END

SUBROUTINE SCAN(BUF,LTH)
IMPLICIT INTEGER(A-Z)
LOGICAL X1 BUF,DEVICE
DIMENSION BUF(1),DEVICE(4)
DATA DEVICE/'S','Y','Q','/'
DO 1 I=1,LTH
IF (BUF(I).EQ.DEVICE(4))RETURN
LTH=LTH+4
DO 2 I=LTH,5,-1
BUF(I)=BUF(I-4)
DO 3 I=1,4
BUF(I)=DEVICE(I)
RETURN
END

```

25

2

1

2

3