SUB-BAND CODING OF SPEECH: A SPLIT-BAND ADPCM CODER WITH ENTROPY ENCODING

by

VICTOR SHKAWRYTKO

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Master of Science
in
Electrical Engineering

ISBN 0-315-34047-9

SUB-BAND CODING OF SPEECH: A SPLIT-BAND ADPCM CODER WITH ENTROPY ENCODING

BY

VICTOR SHKAWRYTKO

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

I hereby declare that I am the sole author of this publication.

I authorize the University of Manitoba to lend this publication to other institutions or individuals for the purpose of scholarly research.

VICTOR SHKAWRYTKO

I further authorize the University of Manitoba to reproduce this publication by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

VICTOR SHKAWRYTKO

The University of Manitoba requires the signatures of all persons using or photocopying this publication. Please sign below, and give address and date.

ABSTRACT

A reduction in the memory space required to store sub-band coded (Split-Band ADPCM) speech is achieved using entropy encoding. Two methods of entropy encoding used to reduce the average word length of Split-Band ADPCM digital output symbols are presented in this study. Huffman coding (a minimum redundancy technique) is compared to a sub-optimal approach developed by Paul D. Shaft, which is useful when source statistics are not accurately specified. Both marginal and conditional probability estimates are employed in the coding procedures. The Split-Band ADPCM encoder implementation is a variation on current designs and employs off-the-shelf switched-capacitor filters for the band-splitting function.

The average code word length resulting from Huffman coding was found to be within 2% of the calculated entropy (the theoretical minimum). Actual space savings of 14.2% for marginal coding and 29.9% for conditional coding were obtained. Shaft coding performed comparably in both the marginal and conditional cases (within 3% of Huffman coding) with savings of 12% and 27.9%, respectively.

ACKNOWLEDGEMENTS

The author expresses his deep appreciation to Professor G. O. Martens for his interest, guidance and support during the course of this work. The author wishes to thank Mr. Mark Jarmasz, Mr. Paul Soble and Mr. Rick Smegal for the many useful discussions he had with them. Thanks are also extended to Ms. Renée Martens for providing a voice sample for analysis.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter I

INTRODUCTION

Advances in digital technology have spawned the development of various digital systems for communication purposes [1]. Since speech is clearly the most natural form of human communication, it is not surprising that the digitization of speech signals has been and continues to be a major component of digital communications research.

Efficient representation of digitized speech not only provides savings in channel bandwidth (or storage requirements) in conventional systems such as telephony, but also facilitates many new and innovative approaches to communication. Among these are voice response from computers, 'store and forward' voice messageing, aids to the visually handicapped, instructional aids, emergency alert systems, etc. In short, any application where voice is the desired mode of communication can exploit this technology. The desirability for speech as a communication medium has spurred on years of research in the field of coding digitized speech.

The underlying goal in efficient speech coding is to achieve maximal speech quality at a minimal data rate with the simplest (or least expensive) coder configuration [2]. Coder complexity tends to be proportional to the efficiency

of the coding technique and therefore the channel utilization. Complex coders are typically expensive so that in the design of digital speech systems one must balance coding efficiency with coder cost, trading off efficiency for complexity.

Current advances in large scale and very large scale integration (LSI and VLSI) have shifted the emphasis in speech coding to more complex schemes since more powerful signal processing hardware is becoming available. An indicator of this trend is the appearance of so-called 'specialty' processors, such as the single-chip signal processors by Bell Labs (DSP), Texas Instruments (TMS 320-10) and Nippon Electric (μPD7720). Using these devices alongside the current higher speed microprocessors now makes the implementation of highly efficient speech coders at reasonable costs possible.

Efficient coding of speech involves the exploitaton of various speech signal characteristics. These include time domain and frequency domain signal characteristics [2]. Waveform coding is a broad category of speech coding techniques which involves facsimile reproduction of an input waveform through the use of time and/or frequency domain charateristics of the signal [2]. This most often results in a tolerably distorted signal replica which satisfies the requirements of a given application while achieving an overall reduction in the output data rate of the coder. Subband coding is a speech coding technique which falls into

the broader category of Waveform coding. It requires a relatively complex hardware structure but produces a high quality representation of the speech signal (according to certain noise preference criteria) at moderate bit-rates [3]. A subset of sub-band coders, an approach known as Split-Band ADPCM, is the focus of this thesis.

Entropy coding (or variable-length coding) is an approach to encoding quantized data with a variable word-length format to compress the amount of data produced by the digitization process. This technique relies on a nonuniform probability distribution of the quantized data points [4]. Short code words are assigned to the most likely data points while longer code words are assigned to less probable quantizer outputs. This yields an average data rate that is lower than the uncoded data rate, provided that the probability distribution of the quantizer output is non-uniform. An optimal code word assignment procedure (lowest average code word length) is that due to D. A. Huffman [5].

The use of variable-length coding to decrease the average data rate of a Split-Band ADPCM coder is described in this thesis. Basic speech signal characteristics are presented initially, in Chapter II, followed by an overview of Waveform Coders, including ADPCM and Sub-band coding, in Chapter III. The type of 'noiseless' entropy coding discussed above is presented in Chapter IV where two procedures for variable-length coding are described: Huffman coding and a pro-

cedure developed by P. D. Shaft [6]. A Split-Band ADPCM coder implemented for this study is presented in Chapter V, including a description of the coder implementation and coder performance measurements. The coder used was implemented in hardware and speech data were collected and entropy coded on a microcomputer. The analysis/coding procedures are described in Chapter VI and the results of performing entropy coding on the Split-Band ADPCM coder output are presented and discussed in Chapter VII. Two variable-length coding procedures are compared: Huffman coding and Shaft coding. Conditional coding (using conditional probabilities) is also implemented for both variable-length schemes.

Conclusions and recommendations for further work are presented in the concluding chapter, Chapter VIII.

Chapter II

SPEECH SIGNAL CHARACTERISTICS

2.1 OVER VIEW OF SPEECH PRODUCTION

Figure 2.1 shows a simplified speech production model typically used in describing speech [7,8]. This model consists of two linearly separable sections, the sound source and the vocal tract filter. The sound source consists of two separate, alternately selectable sources : a) a pulse source representing the periodic excitation of the vocal tract by the vocal chords (during vowels and dipthongs) called voiced

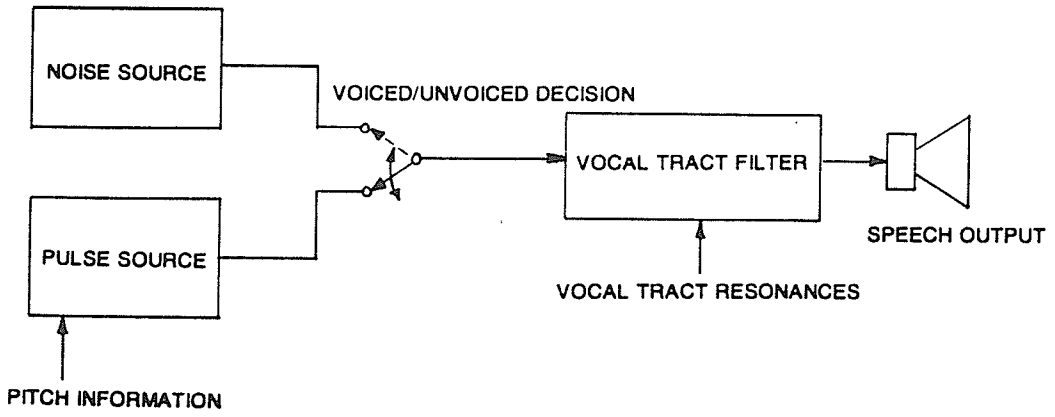


Figure 2.1: Speech Production Model

excitation, and b) a noise source representing the excitation of the vocal tract by air flowing past a constriction (typical of fricatives) called unvoiced excitation. The vocal tract filter section is a model of the resonant filtering action of the vocal tract which spectrally shapes the excitation (sound source). This model has been used to parameterize the speech production process.

Parameterization of the speech production process has been exploited in encoding procedures known as speech source coding techniques [2]. These source coding techniques rely on mathematically modelling speech generation in order to mimic vocal tract operation. Key parameters of the model are extracted from real speech, and then used in a mathematical analog to regenerate speech at will. The primary information-carrying parameters of the vocal process model are considered to be pitch, voiced/unvoiced decision, and pole frequencies of the modulating vocal tract filter [8]. This approach has been successfully used in a variety of voice coders (vocoders) employing analog components initially and later digital hardware. Although efficient from a bit-rate point of view, these approaches have been typically costly and complex.

A second class of speech coders is termed waveform coders. Waveform coders strive for facsimile reproduction of the input waveform. Several properties of the speech signal are used in the design of Waveform coders. These include

power and amplitude distributions, spectral characteristics and certain temporal waveform properties.

Some basic speech waveform properties are discussed below.

2.2 BASIC SPEECH WAVEFORM PROPERTIES

Among the most basic properties of speech waveforms is that speech is bandlimited and thus may be sampled at a finite rate [2]. This bandlimiting is partially due to the process of speech production, but more typically bandlimiting is performed by voice transmission circuits. A typical conservative sampling rate of 8 kHz is used in telephony applications where the signal is bandlimited in the range of 300 to 3.2 kHz by conventional voice circuits.

A second basic property of speech which can easily be observed on an oscilloscope is the dual nature of the speech excitation as shown by the speech production model of Figure 2.1 Intense, quasi-periodic segments of voiced speech are observed which contrast with lower amplitude noise-like passages corresponding to unvoiced sounds [2]. Properties of this dual source excitation are reflected in the short-term signal statistics of the speech waveform.

Characteristics of the speech waveform can be observed in both the time domain and the frequency domain.

Time domain characterisitics are measured using the probability density function (PDF) and the autocorrelation function (ACF) of the signal. The PDF of speech (representing the characteristics of speech amplitude) is in general non-uniform, with a high probability of zero and near-zero amplitudes and a significant probability of very high amplitudes [2]. Both long-term and short-term PDFs of speech have been modelled, the former as the sum of Gaussian and Laplace functions (or by the Gamma function) and the latter by simply a Gaussian model [8].

The quasi-periodic nature of speech is easily shown using the ACF. The ACF (which represents the correlations among amplitude samples of a waveform) shows large adjacent-sample correlations in Nyquist sampled speech when averaged over long segments (55 seconds) [2]. Short-term ACFs of speech (20 milliseconds) show secondary peaks due to the quasi-periodicity [2]. These correlations indicate that redundancy is present in the signal and that data compression can be achieved by removing this redundancy.

In the frequency domain, the power spectral density (PSD) is typically used to characterize the speech waveform. A long-term averaged PSD of speech shows a predominantly low-pass spectrum. Unvoiced segments of speech can have high-pass spectra while voiced segments can show local resonances (called formants) despite their globally low-pass nature [8]. The time course of these formants as well as the high

frequency waveform components are recognized as important factors in speech intelligibility and must therefore be adequately represented.

Chapter III

WAVEFORM CODING TECHNIQUES

As mentioned in the previous chapters, Waveform coders are used to replicate a digitized waveform through the exploitation of certain waveform properties to reduce the coder output data rate.

Waveform coders can be classified according to the type of signal characteristics a particular coder utilizes. Specifically, two classes of Waveform coders are defined: i) time domain Waveform coders and ii) frequency domain Waveform coders. Although the sub-band coder investigated in this study is regarded as a member of the latter category, this designation is somewhat inaccurate, since sub-band coding tends to be on the threshold between the two classes and has been classified by some as an intermediate technique [9].

Split-Band ADPCM sub-band coding employs both time domain and frequency domain speech characteristics. Hence, both classes of coders are described in this chapter, followed by a description of the Split-band ADPCM approach. A detailed discussion of Waveform coders can be found in [2] and [10].

3.1 TIME DOMAIN WAVEFORM CODERS

Pulse Code Modulation (PCM)

All digital signal processing techniques, including Waveform coding, require that the signal be digitized. This involves the conversion of the analog input signal into a discrete-time (sampled at uniform intervals) and discrete-amplitude (quantized to one of a finite number of levels) representation. In this form, the signal can be subjected to a variety of mathematical operations in order to arrive at a desired output sequence and ultimately a desired output signal. Digitization is also referred to as Analog-to-Digital (A/D) conversion.

The term Pulse Code Modulation (PCM) is used to describe one of the most commonly used forms of A/D conversion. The following steps are involved in PCM [10]:

1. As with all sampled signals, the input signal is bandlimited and then sampled at the Nyquist frequency. If W is the bandwidth of the input signal, a sampling rate of 2W (at least) is used.
2. For a given number of bits per quantum level, r, the input signal is quantized to one of 2^r levels where each level is represented by a distinct binary number.
3. To decode, the binary words are mapped back into amplitude levels at each sampling interval and then in-

terpolated by a low pass filter to reconstruct the continuous waveform.

Recalling that speech amplitudes exhibit a wide dynamic range with both low amplitude segments and large amplitude excursions, the noise associated with approximating the signal with a finite number of quantum levels comes strongly into play. If the levels are uniformly distributed over the range of the quantizer, then the low amplitude segments are quantized with only a few levels. This degrades the signal representation by increasing the quantization noise during low amplitude passages. Increasing the number of quantizer levels improves the signal-to-quantizing noise ratio but also increases the output data rate. A non-uniform quantization scheme can alleviate the quantizing noise problem somewhat by providing a greater number of quantizing levels for the low amplitudes and fewer levels for the less frequent larger amplitude excursions. Such non-uniform quantizers are used in telephony and are classified according to the formula used to determine the (logarithmic) level distribution curve (μ-law and A-law quantizers).

A second approach to improving the signal-to-quantizing noise ratio is to vary the quantizer step-sizes dynamically according to the variance of the input signal. One such approach is known as Adaptive PCM and is discussed in a later section.

Differential Approaches

Since the variance of the input signal affects the signal-to-quantization noise ratio, reduction of the input signal variance can improve the performance of a uniform quantizer.

One way to decrease the variance of a signal to be quantized is to apply differential encoding to the signal. Differential encoding involves quantization of the difference between an input sample and an estimate of that sample based on past input estimates. The following technique is such an approach.

Differential Pulse Code Modulation (DPCM)

Nyquist rate sampled speech exhibits large adjacent sample correlation [8]. If this correlation is greater than 50 % then a differential encoding scheme can be effectively employed.

Figure 3.1 shows a Differential PCM block diagram. The input/output expressions are given by

$$d_n = X_n - \hat{X}_n' = X_n - \sum_{i=1}^{N} a_i \hat{X}_{n-i} \qquad (3.1)$$

where $\hat{X}_n'$ is the predicted value of X_n, based on N weighted samples of $\hat{X}_n$. Jayant has shown that the variance of d_n is smaller than the variance of the input signal [10]. Hence, the required dynamic range for the quantizer can be lower or

fewer bits can be used to quantize the difference signal. The estimate $\hat{X}_n'$ is a best estimate in a minimum mean squared error sense if the a_i are computed using the first N samples of the speech ACF [11, 8]. Since these a_i are fixed, only long-term average ACF values are used.

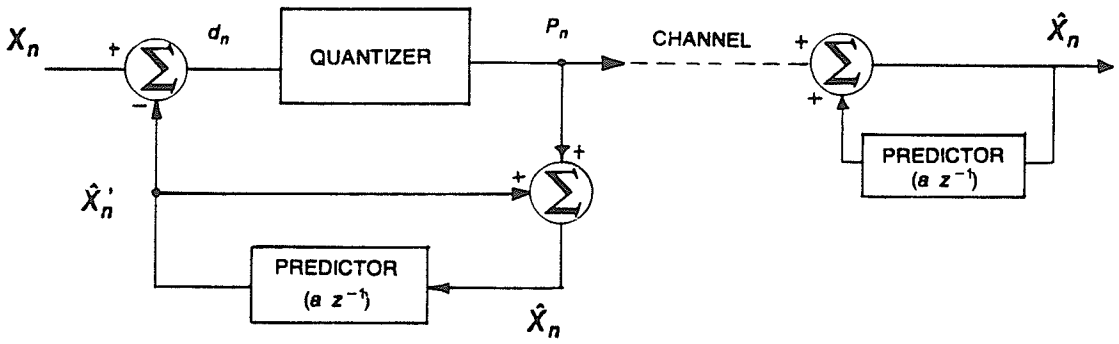


Figure 3.1: Differential PCM Block Diagram

Drawbacks of a fixed set of such a_i are apparent. Long-term ACFs do not accurately reflect short-term speech statistics. Hence, the coder will not perform optimally over a range of inputs. A more effective approach, adaptive DPCM, is discussed later.

A simplified form of differential coding known as Delta Modulation is often used where inexpensive coder hardware is

required. In Delta Modulation, the speech signal is over-sampled at typically 4 to 10 times the Nyquist rate and then coarsely quantized into a one-bit value. This technique uses simple, inexpensive hardware and generates good quality speech at moderate bit rates (16-64 kbits/sec) [10].

Adaptive Approaches

The error signal will have less redundancy than the input signal if the adjacent sample correlations are high. The better the estimate, the lower the variance of the error signal will be, resulting in less redundancy in the error signal.

Matching quantizer step-size (quantizer levels) to the input signal variance can be achieved using adaptive techniques. Adaptive techniques must somehow measure input signal activity, usually from past values of the coder output, and modify the step-size accordingly. Input signal power for speech coders varies slowly enough so that one or two sample adaption schemes are possible [2]. This implies that the coder requires a one or two sample memory and that a new step-size must be computed at each sampling instant. Adaptive PCM is such an approach. The quantizer step-size is varied according to one or two past values of the coder output. The inverse process is performed at the decoder and the original waveform is reproduced. (Note: Since past outputs are used, both the encoder and decoder have the same

information for varying the step-size. No side information is needed at the decoder).

Adaptive Differential PCM

Ideally, a priori knowledge of the input to a coder would allow two forms of adaption in an ADPCM coder: i) an adaptive quantizer matched to the PDF of the input signal to maintain a fixed signal-to-quantizing noise ratio, and ii) an adaptive predictor matched to the ACF of the input signal to remove redundancy and to further decrease the data rate. Since very little a priori knowledge of the signal is available, local estimates of input signal parameters must be computed. These parameters may be applied either to quantizer adaption or predictor adaption. A combination of these two is also possible, although much more complex.

A schematic block diagram of an ADPCM (Adaptive Quantizer) coder is shown in Figure 3.2. It contains a fixed first order predictor and step size adaption logic which operates on the most recent quantizer output.

If the output of an r-bit uniform quantizer is given by

$$y_n = P_n \frac{\Delta_n}{2} \quad where \pm P_n = 1, 3, \cdots 2^r - 1 \tag{3.2}$$

the step size Δ_n is

$$\Delta_n = \Delta_{n-1} M(|P_{n-1}|) \tag{3.3}$$

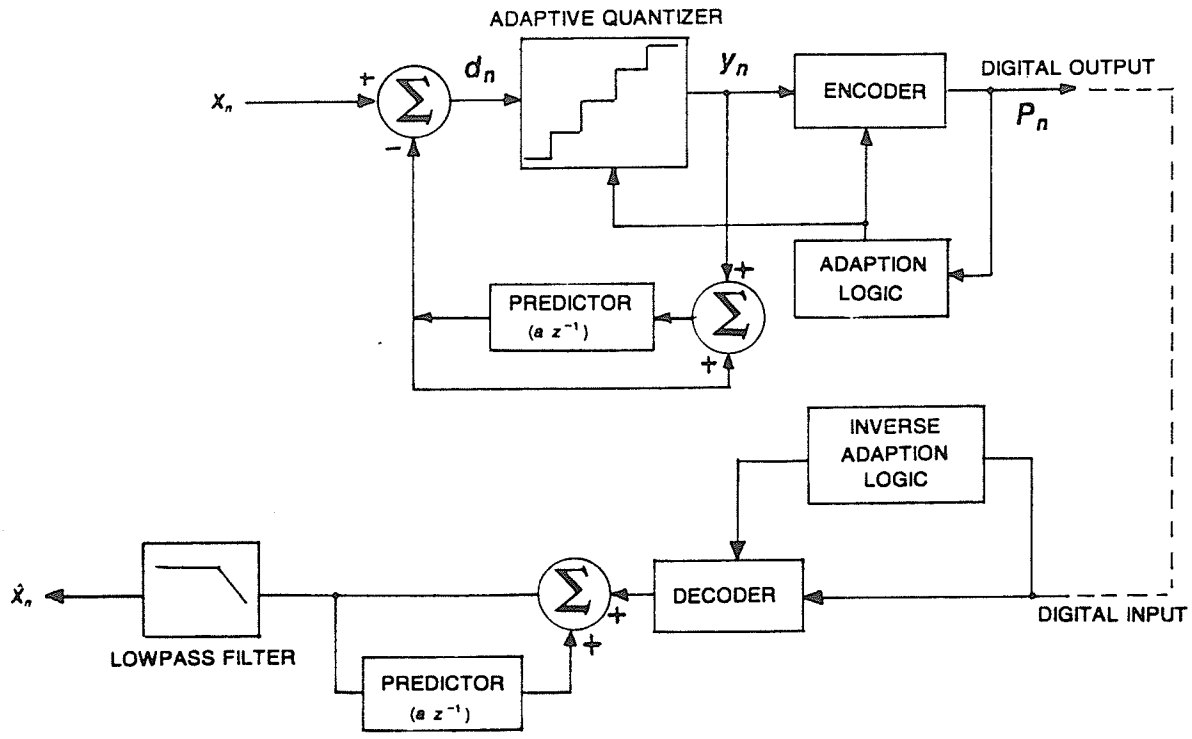


Figure 3.2: Block Diagram of Adaptive DPCM Coder (Adaptive Quantizer)

where M( ) is a time-invariant function of the quantizer level magnitude. Table 3.1 shows adaption multipliers associated with a 3-bit quantizer [12]. The design of such multipliers is based on the maximization of the signal-to-quantizing noise power averaged over an entire test signal [12].

This type of adaptive quantizer ADPCM coder is used in the Split-Band ADPCM coder implemented for this study. Details of the specific adaption scheme used in this coder are given in Chapter V.

TABLE 3.1

Step-Size Adaption Multipliers of a 3-Bit Quantizer

Quantizer Output Word ($\|P_{n-1}\|$)	Multiplier Value $M(\|P_{n-1}\|)$
111 or 000	2
110 or 001	5/4
101 or 010	7/8
100 or 011	7/8

ADPCM with adaptive prediction is a second possible approach. The technique involves computation of the short-term ACF of the speech segment to be quantized, implying some encoding delay. The coefficients derived from the ACF are used in the predictor and are transmitted to the receiver as side information along with the quantizer output. The transmission of the predictor coefficients does not require a great deal of channel bandwidth since updates are infrequent and the coefficients can be encoded coarsely [2].

3.2 FREQUENCY DOMAIN WAVEFORM CODERS

Split-Band ADPCM is a subcategory of frequency domain coders (specifically sub-band coders). Frequency domain coders are coders which segment the speech signal according to frequency and encode each segment separately. In this way, it is possible to vary the overall quantizing noise spectrum to obtain perceptually optimum coding by adjusting individual band quantizers. As well, bits can be allocated as required and may even be dynamically reallocated to encode only active sub-bands with a minimum number of bits.

Frequency domain coders vary in complexity from the simpler sub-band coders to more complex Adaptive Transform Coders which use block transformations. The emphasis in this section will be on sub-band coding techniques. Adaptive Transform Coding is briefly described for completeness.

Sub-Band Coding

Sub-band coding [3] involves partitioning the speech spectrum into typically four to eight sub-bands using band-pass filters. Each of these bands is then low-pass translated, sampled at the Nyquist rate for that band, and then digitally encoded. Each band is independently encoded according to perceptual criteria specific to that band. Decoding involves the translation of the digital information back into band signals, again through modulation and band-

pass filtering, and then the summation of the band signals to recover a replica of the original signal. Figure 3.3 is

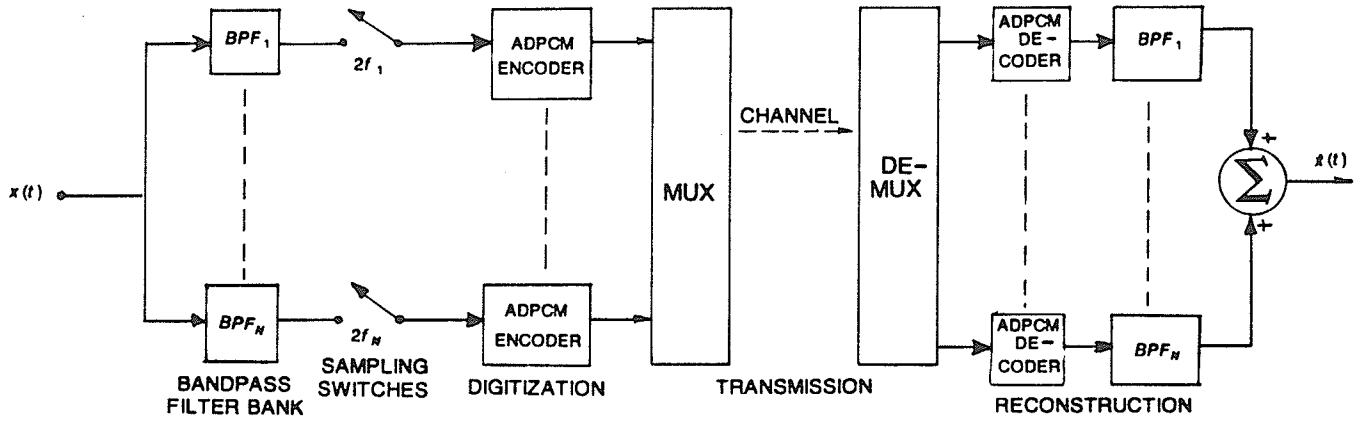


Figure 3.3: Sub-band Coder Block Diagram

a block diagram of a sub-band coder. Various schemes of band-pass and low-pass translation exist, but a common approach is to use integer-band sampling to eliminate the need for modulators [13].

The main design considerations in sub-band coding are [13]:

1. selection of the number of sub-bands and their bandwidths,
2. the implementation of the band-pass filters,

3. selection of the type of band encoders and the number of bits per band.

The emphasis in this study is placed on a sub-band coder which encodes 7 kHz bandwidth speech at 64 kbits/sec. A type of sub-band coder employing two bands and using ADPCM for encoding each band is implemented. A description of this type of coder is given in the following section.

Split-Band ADPCM

If a 7 kHz bandwidth signal is split into two 3.5 kHz bands and each band is sampled at 8 kHz, then a 64 kbits/sec bit rate is achieved if 4 bits/sample are used to encode each band. This can be realized using ADPCM encoders on each band. The target of 64 kbits/sec data rate is not a justification for use of two sub-bands or ADPCM. In fact, various combinations of coders and band partitionings are possible with sub-band coding. The particular configuration employed in this study is documented in [14] and [15]. The relative simplicity and ease of implementation makes this configuration a suitable choice for this study.

One aspect which is investigated is an alternate filter-bank implementation. Traditional implementations have either been analog filter banks [14] or digital quadrature mirror filters [15]. An alternate, switched-capacitor filter bank implementation with off-the-shelf integrated cir-

cuit Reticon switched-capacitor filters is used in this study. The filter bank implementation and other implementation details are presented in Chapter V.

Adaptive Transform Coding

Transform coding involves the transformation of windowed segments of speech data. The transformation yields a set of coefficients which are encoded (quantized) and transmitted to the decoder. Upon decoding, the quantized coefficients are inverse-transformed and concatenated to produce a replica of the original waveform.

Time-to-frequency block transformations are of interest in speech processing since such transformations allow coding noise to be controlled in the frequency domain [2]. As well, time-varying properties of speech can be accomodated in the frequency domain through adaptive techniques if the speech production process is modelled as a linear, time-invariant filter model, as is typically the case [8]. An example of such a transform is the Discrete Cosine Transform which has been found to be particularly suited to speech coding. A summary of the adaption strategies, quantization strategies and noise shaping techniques in ATC is given in [2].

Chapter IV

NOISELESS SOURCE CODING

Data compression is merely the removal of redundancy from a signal [16]. Redundancy in a signal may result from a characteristic of the signal source or it may be introduced by some operation on the signal. Nonetheless, redundancy removal yields a more efficient use of signal processing, transmission or storage resources in a signal handling system.

A class of operations used for redundancy removal is the Information-Preserving Transformation, which is reversible [16]. This class employs a mapping procedure to reversibly decrease the correlation between the message symbols of a source. The result is a set of code words with, on average, fewer bits than the original message alphabet.

A review of some information theoretic definitions is presented below followed by a discussion of Huffman and Shaft coding.

4.1 INFORMATION

Information is formally defined in relation to the probability of an event, E, occuring. Specitically, if the probability of event E is P(E) then the information resulting from the occurrence of E is defined as

$$I(E) = \log_2 \frac{1}{P(E)} = -\log_2 P(E) \text{ bits} \qquad (4.1)$$

The units of (4.1) are determined by the base of the logarithm.

4.2 DISCRETE INFORMATION SOURCE

Given that a source S is emitting a sequence of statistically independent source symbols from a fixed and finite alphabet $S = \{s_1, s_2, s_3, .. s_q\}$, then the source can be completely described by the source alphabet and the probabilities with which the symbols occur, $P(s_1)$, $P(s_2)$, $P(s_3)$,...$P(s_q)$. The source is considered discrete since the source alphabet is not continuous. A zero-memory source is defined as a a source which emits symbols which are statistically independent.

4.3 ENTROPY AND REDUNDANCY

The information associated with each symbol emitted from a zero-memory source is given by

$$I(S) = -\log_2 P(S) \text{ bits} \tag{4.2}$$

and is called the self-information or the marginal information of that symbol. An average amount of information per symbol is defined as

$$H(S) = \sum_{i=1}^{q} P(s_i)\, I(s_i) \text{ bits} \tag{4.3}$$

and is known as the entropy (H(S)) of the source S. H(S) is therefore

$$H(S) = -\sum_{i=1}^{q} P(s_i) \log_2 P(s_i) \text{ bits} \tag{4.4}$$

The redundancy of the source is simply defined as

$$R(S) = 1 - H(S) \tag{4.5}$$

4.4 MARKOV SOURCES AND CONDITIONAL ENTROPY

As mentioned in Chapter II, an important feature of the speech signal is the adjacent sample correlations inherent in the speech production process. A zero-memory source is therefore too restrictive a model for speech. A more gener-

alized source, called a Markov source, is defined to account for adjacent sample correlations.

For a Markov source of order m, each source symbol s_i depends on m preceeding symbols and the source is entirely specified by the source alphabet and a set of conditional probabilities. Initially one must consider the Markov source as being in a 'state' which is defined by the occurrence of a sequence of m symbols. This state has a probability of occurrence $P(s_{j_1}, s_{j_2}, \cdots s_{j_m})$ and is one of q^m possible states. A conditional symbol probability is defined as $P(s_i/s_{j_1}, s_{j_2}, \cdots s_{j_m})$ where i = 1,2, 3...q and j_p = 1, 2, 3...q such that the occurrence of the symbol s_i depends only on the preceeding state of m symbols. One can combine the above probabilities and define the probability of a joint event of the state $(s_{j_1}, s_{j_2}, \cdots s_{j_m})$ and the present symbol s_i occuring $P(s_{j_1}, s_{j_2}, \cdots s_{j_m}, s_i)$ which is simply the product of the state probability and the conditional symbol probability.

The information provided by an mth order Markov source is (given the source is in state $(s_{j_1}, s_{j_2}, \cdots s_{j_m})$)

$$I(s_i/s_{j_1}, s_{j_2}, \cdots s_{j_m}) = -\log_2 P(s_i/s_{j_1}, s_{j_2}, \cdots s_{j_m}) \qquad (4.6)$$

Therefore, the entropy of the source for state $(s_{j_1}, s_{j_2}, \cdots s_{j_m})$ is

$$H(S/s_{j_1}, \cdots s_{j_m}) = -\sum_{i=1}^{q} P(s_i/s_{j_1}, \cdots s_{j_m}) \log_2 P(s_i/s_{j_1}, \cdots s_{j_m}) \qquad (4.7)$$

When averaged over q^m possible states, the entropy of the mth order source is

$$H(S) = \sum_{i=1}^{q^m} P(s_{j_1}, \cdots s_{j_m})\, H(S/s_{j_1}, \cdots s_{j_m})$$

$$= \sum_{j=1}^{q^m} P(s_{j_1}, \cdots s_{j_m}) \sum_{i=1}^{q^m} H(s_i/s_{j_1}, \cdots s_{j_m})$$

$$= -\sum_{j=1}^{q^m}\sum_{i=1}^{q^m} P(s_{j_1}, \cdots s_{j_m})\, P(s_i/s_{j_1}, \cdots s_{j_m}) \log_2 P(s_i/s_{j_1}, \cdots s_{j_m}) \qquad (4.8)$$

Let subscript J represent the previous state of the Markov source and subscript N represent the present state. Rewriting H(S)

$$H(S) = -\sum_{j=1}^{q^m}\sum_{i=1}^{q^m} P_J\, P_{N/J} \log_2 P_{N/J} \qquad (4.9)$$

This is referred to as the conditional entropy of the source.

4.5 REDUNDANCY REMOVAL

Information-Preserving Transformations

An information-preserving transformation is a reversible mapping of a message sequence into an output sequence of fewer bits [16]. Reversibility allows the input to be entirely reconstructed from the output sequence; hence the information-preserving nature of the transformation. The greater the correlations between message symbols, the greater the redundancy present in the message and the lower the information content. Through redundancy removal, which is based upon a knowledge of the source statistics, the mapping

of the input sequence results in a nearly random output sequence. One may consider this a flattening of the source PDF.

Minimum-Redundancy Codes (Huffman Codes)

Typical signal representations consist of a finite alphabet (such as an r-bit A/D converter with 2^r output states). Knowing the statistics of the finite alphabet source allows one to construct a code which is optimal in a minimum average code word length sense [5].

Huffman coding is known as minimum-redundancy coding or optimal coding. Optimality in this case means a set of code words with the shortest possible average length. The properties of the broader class of codes to which Huffman codes belong are briefly described below.

Uniquely Decodable and Instantaneous Codes

Two restrictions placed on the Huffman coding procedure are that no two code words shall have identical arrangements of digits and that once the begining of the message is known, no other indicators are needed to determine where a particular code word begins or ends [5]. The first restriction implies that the code constructed is uniquely decodable [17]. Unique decodability states further that any distinct sequence of source symbols must result in a distinct sequence of code words.

The second restriction implies that the code is instantaneous; no reference to succeeding code words is required to decode any code word in a sequence [17]. A necessary and sufficient condition for a code to be instantaneous is that no complete code word is a prefix of any other code word. The prefixes of the code word 0111, for example, are 0111, 011, 01 and 0.

Three additional restrictions define the framework for the construction of an optimal code. A basic requirement is that the most probable source symbols be assigned the shortest code words. The two other restrictions deal with ensuring that the fewest possible digits are assigned to each code word.

Consider an ensemble of N messages. Let P(i) be the probability of the ith message. It follows that

$$\sum_{i=1}^{N} P(i) = 1 \qquad (4.10)$$

Let the length of the ith message be L(i), corresponding to the number of coding digits assigned to it. The average message length is therefore

$$L_{av} = \sum_{i=1}^{N} P(i)\, L(i) \qquad (4.11)$$

Minimum-redundancy or optimality is defined here as a code with the lowest possible L_{av} for an ensemble of N messages and for a given number of coding digits, r.

The procedure used to generate an optimum binary code is given in Chapter VI. A description of this procedure is also available in [5, 17-20].

Alternate Redundancy Removing Codes

Shaft Encoding

In comparison to the Huffman encoding scheme, an alternate variable-length approach, developed by Shaft [6], is implemented for this study. The approach is intuitive and is applicable to situations where source statistics are only partially known or are time-varying. Speech falls into this category of sources [2, 8]. The performance of a coding scheme varies according to the degree of error in the specification of the source statistics (Huffman coding is known to be fairly robust with respect to errors in the source symbol probabilities [21]). The coding procedure of Shaft is described in Chapter VI.

Some properties of the Shaft code are:

1. Minimum length = n'
2. Maximum length = $n' + 2^{(r - n' + 1)} - 2$

where n' is the fewest number of bits to be assigned to represent any source symbol. Thus, if r-n' increases, the minimum length code word decreases while the maximum length code word increases. As a result, the performance bounds are well defined: r-n' bits/symbol at best and

$n' + 2^{(r - n' + 1)} - 2 - r$ bits/symbol more than the uncoded case, at worst.

For the Shaft encoding scheme, the average number of bits per symbol can be computed using equation (4.11).

Fixed-Length Encoding

Prior to a description of fixed-length redundancy-removing codes, a motivation for their use is presented.

The use of Huffman codes can pose practical problems when transmitted. Due to their variable length, any error in a code word can cause loss of code word synchronization at the receiver. Long sequences of errors at the receiver output can result from a single channel error. Only through careful code selection can this problem be decreased, but still not eliminated.

Variable-length input-to-fixed-length output coding or just variable input coding performs a type of mapping opposite to Huffman coding: a sequence of source symbols are mapped into one of a number of fixed-length code words. The sequence length varies, hence the 'variable input' designation. The approach described by Cohn and Melsa in [22] applied to an ADPCM speech digitizer, consists of accepting a sequence of source symbols until a message is formed. This message is then associated with a corresponding code word (channel symbol). Symbols are read from the source until a

member of the message alphabet is recognized and then an appropriate code word is generated and the encoder is reset. One can see that this is one form of run-length encoding [23]. It can be shown [22] that this type of coding has an average minimum-bits-per-symbol property similar to Huffman codes. Because of the fixed-length nature of the code, long strings of errors are avoided during transmission. On the other hand, a single error in a channel symbol can result in a string of errors at the receiver, depending on the length of run which the channel symbol represents.

4.6 FIXED-RATE CHANNELS: BUFFER CONSIDERATIONS

Typically, channels used in digital communications operate at a fixed rate i.e. channel symbols leave the transmitter and arrive at the receiver at equal, uniformly-spaced time intervals. In order to accomodate fixed-rate transmission, variable-length and variable input codes must be buffered prior to transmission or after reception i.e. variable-length codes must be formatted into fixed length channel symbols and variable input codes must buffer source symbols until a message is assembled.

Jelinek has shown [24] that codes designed to minimize the probability of buffer overflow in variable-length codes are not minimum-average-length code words. This indicates that Huffman codes are non-optimal in a minimum-buffer-overflow sense. Since Shaft codes are variable-length

codes, the problem of buffer overflow and also buffer exhaust (when the buffer empties and no symbols are available for transmission) is also present.

Run-length codes also require buffering of both source and received symbols in order for a fixed rate of transmisson in the channel to be possible. Jelinek and Schneider discuss variable-length-to-block coding and buffer considerations in [25].

Buffer considerations are important in communication system design and especially source coding techniques. In the evaluation of the coder implemented for this study, emphasis is placed on measuring sub-band coder performance and the bit-rate reduction associated with entropy encoding. The need for appropriate buffer design is recognized, but is not undertaken.

Chapter V

A SPLIT-BAND ADPCM CODER

Sub-band coding was discussed in Chapter III. In this chapter a simplified sub-band coder using two sub-bands and ADPCM quantizers is described. The configuration implemented for the purpose of this study is described in the context of previous work, as reported in the literature [14,15], on 'commentary grade' speech and music encoding.

5.1 CODER CONFIGURATIONS

As described briefly in Chapter III, Split-Band ADPCM involves the splitting of an input signal into two frequency bands followed by the digitization of each band with a separate ADPCM coder. Figure 5.1 shows the encoding/decoding blocks of such a coder. The rationale behind this or any sub-band coding scheme is to allow individual segments of the input signal spectrum to be quantized in the most perceptually advantageous manner so that bits are allocated where required and quantization noise is 'shaped' in a perceptually acceptable way. As Figure 5.1 shows, two filters, one low-pass and one band-pass, perform the band-splitting function. The filters are followed by two ADPCM encoders, a transmission channel and two ADPCM decoders. Finally, the

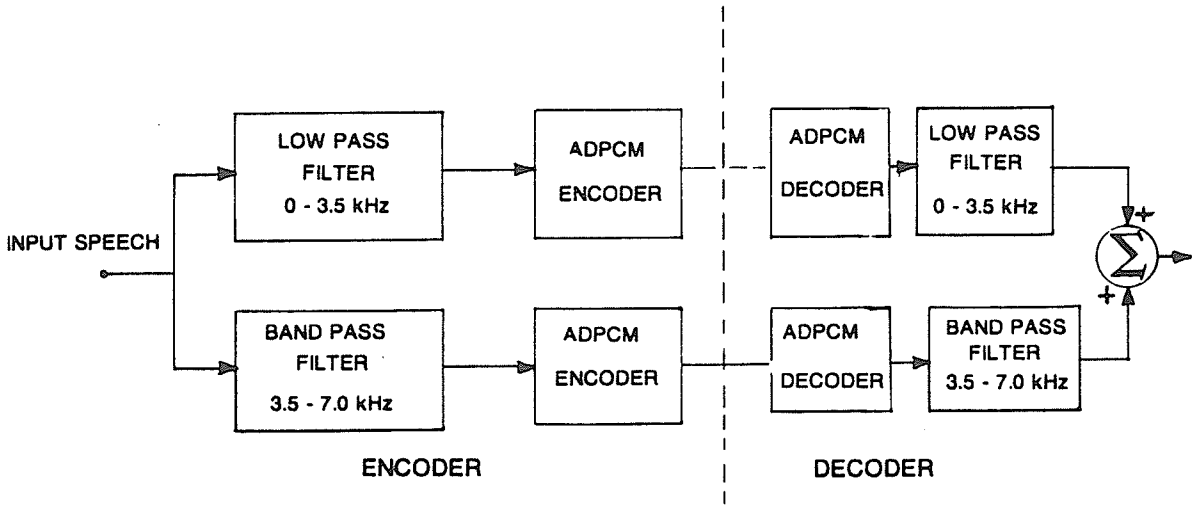


Figure 5.1: Split-Band ADPCM Coder Block Diagram

outputs of the ADPCM decoders are filtered by a filter bank similar to that of the encoder stage and the filter outputs are summed to form a reconstructed replica of the original input waveform.

Band-Splitting Filters

The filter bank in Figure 5.1 can be implemented in various ways. In the evolution of this coder structure, analog filters were initially used by Johnston and Goodman to perform the band-splitting [14]. However, a significant phase difference was encountered between the two filters of the filter bank. The measured frequency response of the coder in [14] is shown in Figure 5.2. The response is affected

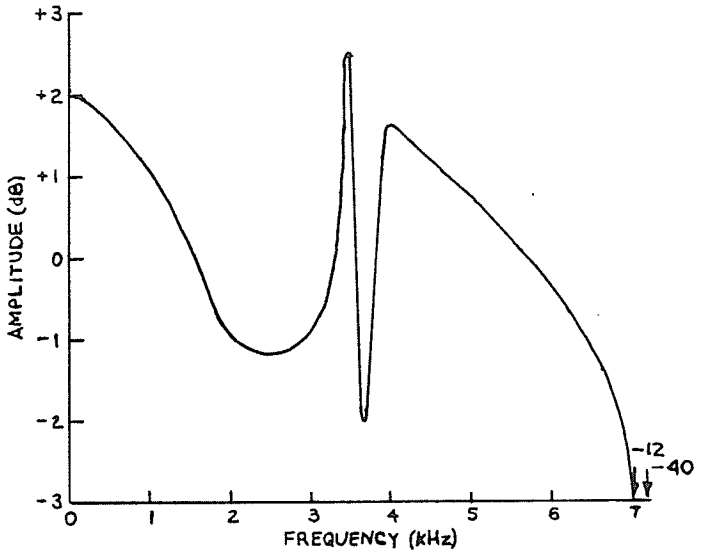


Figure 5.2: Frequency Response of Split-Band Filter Bank [14]

most significantly around the cross-over point between the two bands, where the phase difference has the greatest effect. As well, the use of two different types of coders (ADPCM and APCM) contributed to the different delays in each band.

As an improvement to the design in [14], an all-digital version of the coder was developed by Johnston and Crochiere [15]. An all-digital approach affords better overall performance (due to the degree of control over the coder parameters) and employs a Quadrature-Mirror Filter (QMF) bank as well as two ADPCM encoders. Quadrature-Mirror Filters are a special filter implementation using Finite Impulse Response (FIR) filters designed with fewer taps than are typically required to obtain the desired response. They provide a frequency response with only +/- 0.2 dB ripple at the transition between bands.

QMF banks, if constructed carefully, allow an efficient all-digital filter bank implementation. These filters do require significant computational power to implement the processing algorithms. QMFs are described in detail in [15] and [26].

As an alternative to digital filter banks, the implementation described herein employs Reticon switched-capacitor filters arranged as in Figure 5.3.

Switched-capacitor filters are a type of sampled-data filter (discrete-time but continuous amplitude) which can be easily and efficiently integrated into single-chip devices. No external components are required to determine the filter corner frequency since it is controlled entirely by an input clock (the cutoff is proportional to the clock frequency). Switched-capacitor filters are very stable since only ratios of capacitor values are used to determine the filter characteristic.

Band-splitting is achieved using an overall low-pass filter at 7 kHz followed by low-pass and high-pass filters in parallel to split the band around 3.5 kHz. This particular configuration was selected to impart as much common delay as possible to the filters in each band. Specifically, rather than constructing a band-pass filter from a cascade of high-pass and low-pass filters which involves two stages and more delay than a single filter, the configuration in Figure 5.3 was used.

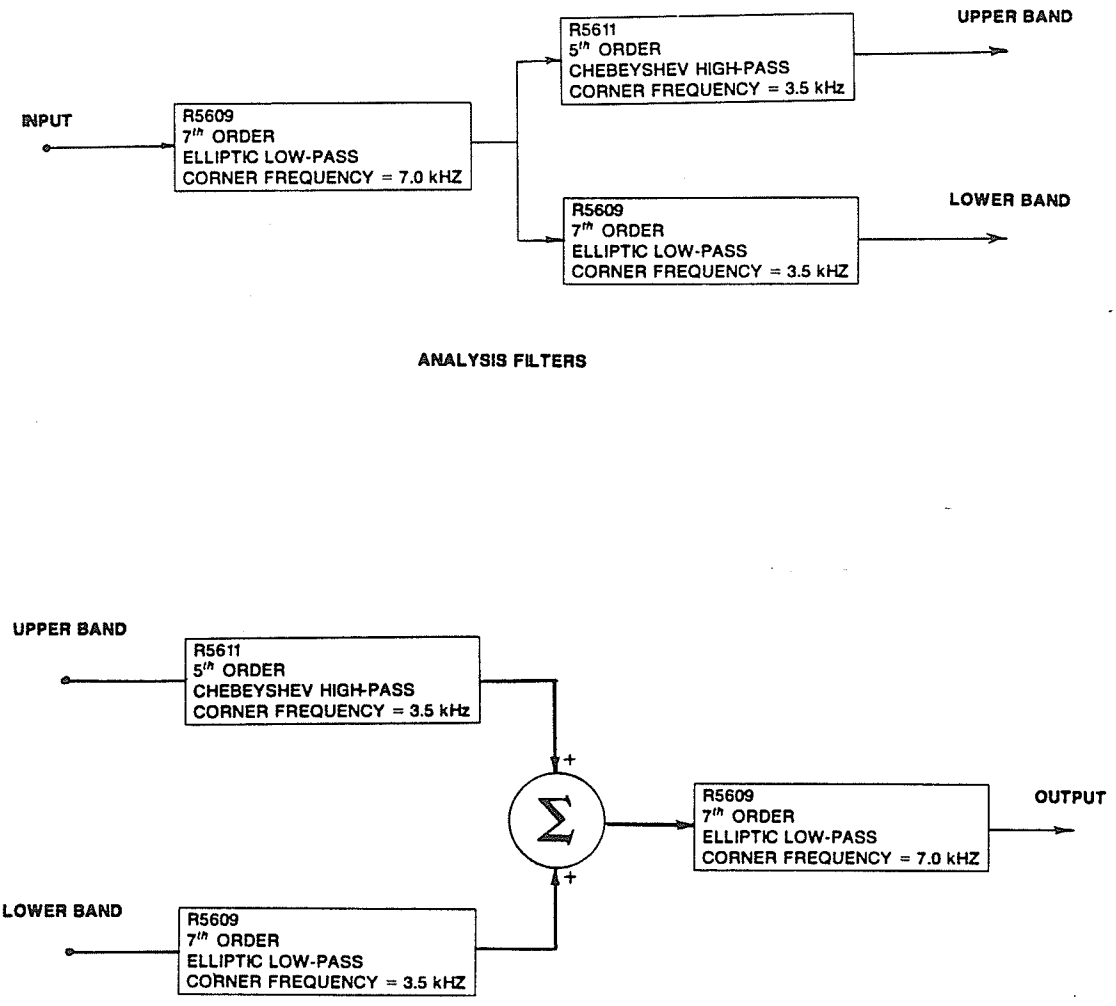


Figure 5.3: Reticon Switched-Capacitor Filter Bank Implementation

Specifications of the Reticon devices along with the clock frequencies used to set the cutoffs of the filter banks are given in Table 5.1.

TABLE 5.1

Reticon Switched-Capacitor Filter Specifications

Filter Type:	R5609	R5611
	Low-pass 7^{th} order Elliptic 75 dB stop-band rejection +/- 0.5 dB pass-band ripple	High-pass 5^{th} order Chebeyshev 30 dB/oct roll-off < 0.6 dB pass-band ripple
Dynamic Range	75 dB	80 dB
Clock/Corner Freq. Ratio	100	500
Clock Freq. used		
Upper Band	700 kHz	1750 kHz
Lower Band	350 kHz	----

ADPCM Coders

As with the filter implementations, various coder implementations have been employed in two-band sub-band coders. Again, the objective is to encode each band with perceptually minimal distortion by selecting an appropriate coding scheme for a particular band. Johnston and Goodman employed two robust adaptive quantizers implemented using voltage controlled amplifiers (ADPCM for the lower band and APCM for the upper band) [14]. Johnston and Crochiere implemented two ADPCM coders, again with adaptive quantizers [15]. This approach involved the optimization of the step-size multipliers to include music as well as speech. The first order predictor coefficients for both ADPCM coders were also selected to accomodate both speech and music signals.

The ADPCM coders used in this study are the commercially available OKI Semiconductor ADPCM codecs (MSM5218). The OKI

MSM5218 is a single chip ADPCM codec which can perform both ADPCM analysis and synthesis. An 8-bit input sample (a sample of up to 12 bits is possible) is converted to either 3 or 4 bits of ADPCM data at the output by the analysis stage of the device The A/D conversion is performed by an OKI MSM5204ARS 8-bit CMOS A/D converter. The data can be stored or transmitted and then passed to the synthesis stage of an identical device to be decoded back into the original signal. For convenience, 8 bits are used at the input and 4 bits at the output. Sampling rates of up to 8 kHz are possible and since the Nyquist rate for a 3.5 kHz bandwidth is 7kHz, the highest rate is employed.

The OKI device consists (internally) of an adaptive quantizer and a first order fixed predictor as described in Chapter III (Figure 3.1) with one exception. The coefficient associated with the predictor is, in this case, unity. This represents an ideal integrator (or accumulator) in the feedback loop. Use of an integrator yields less-than-optimal prediction, that is the error signal variance is not entirely minimized [7]. Speech is a pseudo-stationary process and therefore one cannot expect to achieve minimum error signal variance even with an optimal fixed predictor. Hence, the integrator implementaton is an acceptable alternative to the optimal predictor. As well, the first-order optimal fixed predictor is only slightly more advantageous than the ideal integrator [7].

Recalling equation (3.3), the step size in the adaptive quantizer changes according to the relationship

$$\Delta_n = \Delta_{n-1} M(|P_{n-1}|) \qquad (5.1)$$

More specifically, in the case of the OKI device [27] the relationship is described by

$$\Delta_n = \Delta_{n-1} 1.1^{M(P_{n-1})} \qquad (5.2)$$

TABLE 5.2

Step-Size Multiplier Function

4-Bit Code	$M(P_{n-1})$
1111	+8
1110	+6
1101	+4
1100	+2
1011	-1
1010	-1
1001	-1
1000	-1
0000	-1
0001	-1
0010	-1
0011	-1
0100	+2
0101	+4
0110	+6
0111	+8

where the function $M(|P_{n-1}|)$ is tabulated in Table 5.2. Thus, for a given past output quantizer value, an exponent is selected to compute the next step size. The step size is quantized to 49 values with a minimum of 16 and a maximum of

TABLE 5.3

Quantized Step-Sizes (Δ_n)

No.	Step-size	No.	Step-size	No.	Step-size
1	16	18	80	35	408
2	17	19	88	36	449
3	19	20	97	37	494
4	21	21	107	38	544
5	23	22	118	39	598
6	25	23	130	40	658
7	28	24	143	41	724
8	31	25	157	42	796
9	34	26	173	43	876
10	37	27	190	44	963
11	41	28	209	45	1060
12	45	29	230	46	1166
13	50	30	253	47	1282
14	55	31	279	48	1411
15	60	32	307	49	1552
16	66	33	337		
17	73	34	371		

1552 as shown in Table 5.3. The actual output value q_n is determined by solving the equation

$$q_n = (1 - 2B_3)(\Delta_n B_2 + \frac{1}{2}\Delta_n B_1 + \frac{1}{4}\Delta_n B_0 + \frac{1}{8}\Delta_n) \qquad (5.3)$$

(in the 4-bit case) where the B_n are the values of the powers of 2 in the quantizer output P_n . To reconstruct the original input value at the nth instant the equation

$$\hat{X}_n = \hat{X}_{n-1} + q_n \qquad (5.4)$$

is used. The initial conditions used in the OKI device are $\hat{X}_0 = 0$ and $\Delta_1 = 16$.

Complete schematics of the coder and the microprocessor interface are given in Appendix C.

5.2 CODER PERFORMANCE

The performance of the Split-Band ADPCM coder described in this study was assessed using frequency response and SNR measurements.

Measured Frequency Response

For comparison, see the frequency response of the analog filters used by Johnston and Goodman shown in Figure 5.2 where the maximum deviation from a flat response is seen to be +/- 2 dB. The measured frequency response of the Split-

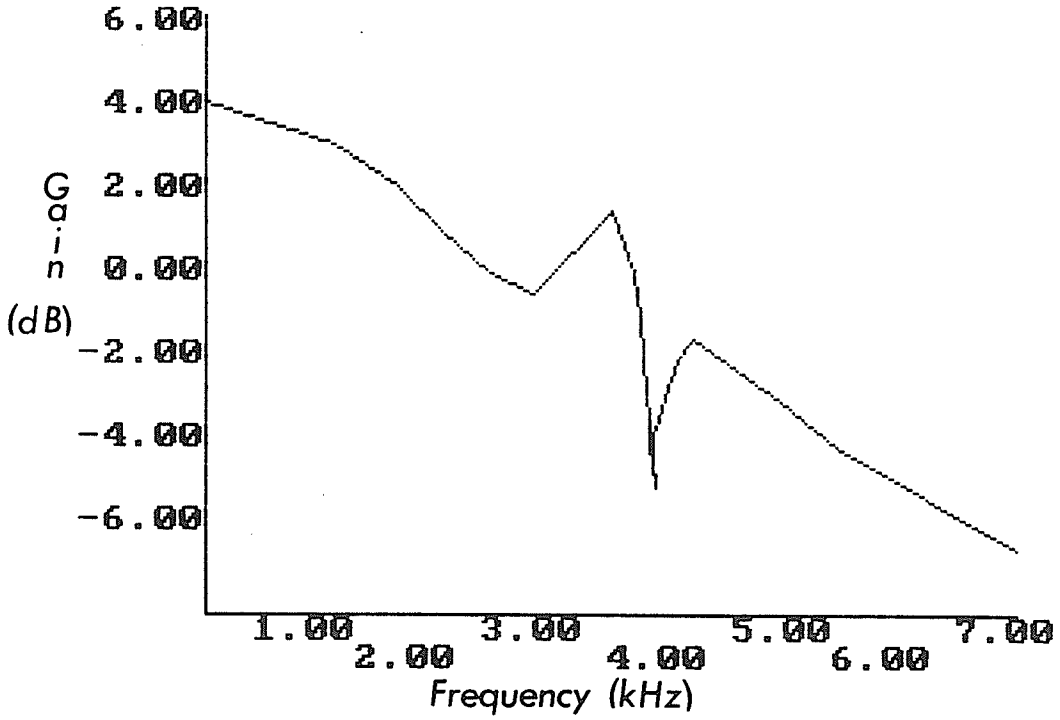


Figure 5.4: Measured Frequency Response of the Split-Band ADPCM Coder

Band ADPCM coder used in this study is given in Figure 5.4. Two features are apparent from this figure: the deviation in the frequency response is +/- 4.5 dB and the upper frequency band is significantly attenuated. Ideally, the upper and lower bands should be equalized so that the response is as uniform as possible. Attenuation of the lower band serves to improve the audio quality of the output signal somewhat (using an informal listening test) but is found to have a negligible effect on the relative frequency probability measurements. All data were encoded using the filter bank with the above frequency response.

SNR Measure

The signal-to-noise ratio of the Split-Band ADPCM Coder was measured using a sinusoid at the input to the coder, a Hewlett-Packard 3400A True RMS meter and an SAE 1800 Parametric Equalizer configured as a tunable notch filter connected to the coder output. The input sinusoid was progressively swept through the frequency range of 100 Hz-8000 Hz. At each measurement frequency, the RMS value of the coder output was measured as well as the output of the notch filter which was tuned to the frequency of the input sinusoid. The ratio of the RMS coder output and the RMS notch filter output was taken to obtain the SNR (in dB). The SNR versus frequency of the Split-Band ADPCM Coder is given in Figure 5.5. The SNR of the Split-Band ADPCM Coder is similar to

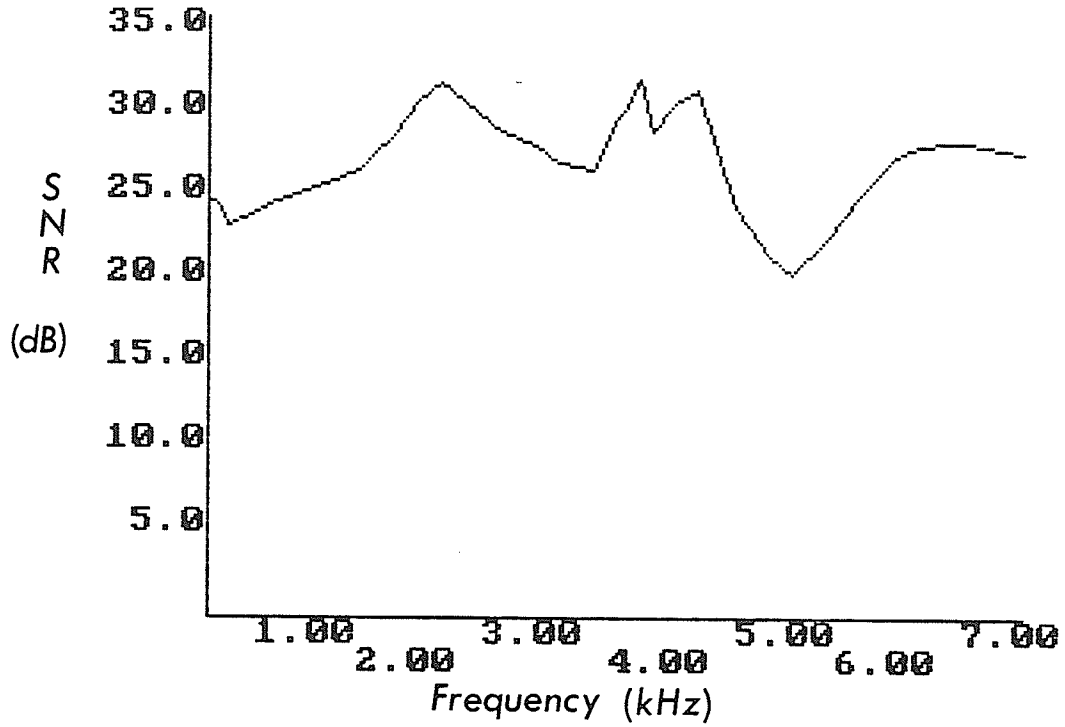


Figure 5.5: SNR vs. Frequency of the Split-Band ADPCM Coder

the analog filter-based implementation by Johnston and Goodman [14]. The analog filtered system is on average 10 dB higher in the lower band, but comparable to the Split-Band ADPCM Coder in the upper frequencies.

An important consideration in the use of off-the-shelf switched-capacitor filters is the clock residue present at the output of the filters. Since switched-capacitor filters are sampled-data devices, sampling residue at the filter output in the form of sampling clock related noise is present. This noise, when fed into a second sampled-data filter that uses a clock not synchronized to the first filter (as in the filter bank implemented in this study), is aliased into the audio band by the second filter and results in au-

dible frequencies harmonically related to the two clocks. This problem was encountered when the Reticon devices were used, since the high-pass filter clock was derived separately from the low-pass filter clocks. Clock synchronization or a single filter bank design with only one input clock would alleviate the problem of sampling clock residue. (Note: Passive RC low-pass filters are used after each output to minimize sampling residue, although complete elimination of it is not achieved in this way.)

Chapter VI

ANALYSIS TECHNIQUES

6.1 RELATIVE FREQUENCY ANALYSIS

A requirement of any variable-length coding scheme is the knowledge of the a priori probabilities of the source symbols to be encoded. Since the actual statistics of a given speech passage are rarely known in advance, measurements of the speech statistics must be performed. Although it is unlikely that an exact evaluation of the probabilities associated with a digitized speech passage can be made without extremely long observations, estimates of these values can be obtained using relative frequency analysis techniques.

The relative frequency interpretation of probability is well known and states that if an experiment is repeated n times and an event E occurs n_e times then the probability P(E) is closely approximated by $\frac{n_e}{n}$, provided n is sufficiently large. In asking how large is a sufficiently large number of trials one must employ the law of large numbers [28], specifically Bernoulli's and Borel's theorems.

Given an event E, in an experiment, which occurs with probability p, one can define the following random variable

$$x_i = \begin{cases} 1 & \text{if } E \text{ occurs at the } i^{th} \text{ trial} \\ 0 & \text{otherwise} \end{cases} \qquad (6.1)$$

The sample mean of this random variable can be defined as

$$\bar{X}_n = \frac{X_1 + \cdots X_n}{n} \tag{6.2}$$

Bernoulli's theorem is then given by

$$P\left\{|\bar{X}_n - p| < \epsilon\right\} \geq 1 - \frac{1}{4\,n\,\epsilon^2} \tag{6.3}$$

where epsilon is a non-negative constant. The theorem states that one can calculate the probability of the sample mean being within epsilon of the actual probability. Therefore, by selecting an epsilon and a desired degree of certainty, the number of trials required to satisfy these criteria can be calculated. This then becomes a lower bound on the 'sufficient' number of trials mentioned above. Borel's Theorem is a stronger statement of Bernoullli's Theorem in that it states that the sample mean tends to the true probability with probability 1.

Two additional factors affect the required number of trials in the experiment. The value of epsilon obtained using Bernoulli's Theorem represents an 'error' within which the actual probability values can be calculated with a selected certainty. This implies that the measured probability estimates must be greater than or equal to epsilon and that two adjacent probabilities must not be closer than twice epsilon in order for the probability estimates to be unambiguously resolved. The above conditions depend directly on the number of trials (n) and the source symbol statistics (which

are not known a priori). An initial guess for a maximum epsilon can be made: epsilon must be smaller than 1/N, where N is the number of distinct source symbols This assumes that the source symbols are equiprobable. Measurements of the symbol probabilities can be used to further adjust the number of trials, based on the smallest measured distance between adjacent probability estimates.

For probability estimates whose magnitudes are less than epsilon, i.e. the magnitude of the error is greater than the magnitude of the measurement, one can adjust these values according to the following rule: all non-zero values of less than epsilon are assumed to be equiprobable and are assigned equal values. These values, when summed, constitute the difference between the sum of the remaining probability estimates (those greater than epsilon) and 1. Thus, the ambiguity of these estimated values is arbitrarily removed and the worst case for coding purposes is imposed. The only factor which must then be considered is how many adjusted values one is willing to accept. This decision can be based on the maximum number of trials which can be realistically handled by the processing system employed and the characteristics of the random source being measured.

For this study, relative frequency analysis was performed on the speech data acquired from the sub-band coder. Each 4-bit sample was treated as a source symbol and counted. The total number of samples was also counted and then the

ratio of source symbol count to total count was taken, yielding an estimate of the probability of occurrence of the given source symbol.

Based on speech sample sizes used in published studies [4, 29], approximately 220 seconds of speech were collected and a corresponding value of epsilon was calculated based on a 99% certainty value. Adjustments based on epsilon were made to the probability estimates which were less than epsilon (18% of the marginal values and 46% of the conditional values). Differences between probability estimates were not used to vary epsilon since the time required to collect and process the data had become prohibitive. It was determined that doubling the number of data points, for example, would not effect the results significantly but would increase collection and processing times dramatically.

The probabilities measured using the above approach represent the marginal probabilities of the different source symbols. The conditional probabilities used in the conditional form of coding are obtained by extending the above procedure. The computation of conditional probabilities is discussed in a later section. Histograms of the marginal and conditional probability estimates are given in Appendix B.

6.2 AVERAGE LENGTH CRITERION

The degree to which a given approach compresses data must ultimately be measured by applying the coding technique to raw data and observing the resulting reduction. More typically for a variable-length code the average length of the code is specified. The average length (as mentioned in Chapter 4) is given by the sum

$$L_{av} = \sum_{i=1}^{\alpha} P(i)\, L(i) \qquad (6.4)$$

Substituting in the relative frequency representation of the probability yields the equation

$$L_{av} = \sum_{i=1}^{\alpha} \frac{n_i}{N}\, l_i \qquad (6.5)$$

where l_i is the length associated with the i^{th} probability estimate

For the purposes of this study, the average length is used as the figure of merit.

The average length is calculated according to equation (6.5) using the symbol probabilities and the symbol lengths generated by the coding programs. One can show that the average length corresponds directly to the reduction in storage due to coding for both the marginal and conditional cases. This is shown in Appendix A.

6.3 CONDITIONAL PROBABILITY CALCULATIONS

Since the Split-Band ADPCM coder output can be modelled as a Markov source, due to large adjacent speech sample correlations, conditional coding can be used in an attempt to further decrease the average word-length of the entropy encoding. In this study, the coder output is modelled as a first order Markov source in accordance with the discussion in Chapter IV. The conditional relative frequencies are calculated using the formulations below. Initially, one must recall a definition of probability which states

$$P(i/j) = \frac{P(i,j)}{P(j)} \tag{6.6}$$

Recall that in Chapter IV, the probability of the occurrence of a joint event (the present symbol and the past, conditional state) is defined for a Markov source. Thus, by calculating the relative frequency of the joint event and dividing by the marginal relative frequency of the conditional state, the conditional probability is computed. The resulting conditional probability is given by

$$P(i/j) = \frac{n_{i,j}}{n_j} \tag{6.7}$$

6.4 HUFFMAN CODING PROCEDURE

Following the basic provision that the most probable symbols are coded with the shortest code words, the Huffman procedure involves the following steps:

1. The source symbols are rank ordered according to probability.
2. The least probable symbol (the Nth symbol, for example) is combined with the N-1 symbol to create a new composite symbol whose probability is the sum P(N)+P(N-1).
3. A new source with one fewer symbols is then constructed and the rank ordering procedure is applied again.
4. The process of combining the least probable source symbols and then rank ordering a new smaller source is repeated until only one symbol remains.
5. In order to assign sequences of digits to each source symbol one must determine how many times an original source symbol has been combined to produce a new, composite source symbol; this corresponds to the number of digits to be assigned to the original source symbol.
6. The process of assigning actual digits to the source symbols requires that the path traced through the series of composite symbols (as followed in the previous step to determine the code word length) be re-

versed, beginning at the source containing a single symbol and tracing backward to the original symbols; each time a composite symbol is split into two, a digit is assigned to the least probable symbol while the opposite digit is assigned to the second-least probable symbol.

The above process is best described through an example.

EXAMPLE 6-1

Given a source $S=\{s_1, s_2, s_3, \ldots s_8\}$ with the following probability assignments

$$s_1=0.55$$
$$s_2=0.01$$
$$s_3=0.06$$
$$s_4=0.30$$
$$s_5=0.01$$
$$s_6=0.01$$
$$s_7=0.05$$
$$s_8=0.01$$

Huffman coding procedes as follows:

1. Step 1: The symbols are rank ordered in decreasing probability:

$$s_1=0.55$$

$$s_4=0.30$$

$$s_3=0.06$$

$$s_7=0.05$$

$$s_2=0.01$$

$$s_5=0.01$$

$$s_6=0.01$$

$$s_8=0.01$$

2. Step 2: The two lowest probabilities are combined to make a new symbol and a new source containing one less symbol

$$s_1=0.55$$

$$s_4=0.30$$

$$s_3=0.06$$

$$s_7=0.05$$

$$s_2=0.01$$

$$s_5=0.01$$

$$s_9= s_8+s_6=0.02$$

3. Step 3: Steps 1 and 2 are continued until only one symbol remains

Step 1	Step 2	Step 1	Step 2
$s_1=0.55$	s_1	$s_1=0.55$	s_1
$s_4=0.30$	s_4	$s_4=0.30$	s_4
$s_3=0.06$	s_3	$s_3=0.06$	s_3
$s_7=0.05$	s_7	$s_7=0.05$	s_7
$s_9=0.02$	s_9	$s_9=0.02$	$s_{11}=s_{10}+s_9$
$s_2=0.01$	$s_{10}=s_5+s_2$	$s_{10}=0.02$	
$s_5=0.01$			

Step1	Step 2	Step 1	Step 2
$s_1=0.55$	s_1	$s_1=0.55$	s_1
$s_4=0.30$	s_3	$s_4=0.30$	s_4
$s_3=0.06$	s_7	$s_{12}=0.09$	$s_{13}=s_{12}+s_3$
$s_7=0.05$	$s_{12}=s_{11}+s_7$	$s_3=0.06$	
$s_{11}=0.04$			

Step 1	Step 2	Step 1	Step 2
$s_1=0.55$	s_1	$s_1=0.55$	$s_{15}=s_{14}+s_1$
$s_4=0.30$	$s_{14}=s_{13}+s_4$	$s_{14}=0.45$	
$s_{13}=0.15$			

4. Step 4: To determine the word lengths (and subsequently the actual code words) the composite source symbols must be decomposed into their original source symbol combinations. This is most easily accomplished by drawing a tree structure with each composite symbol as a node and its two components as

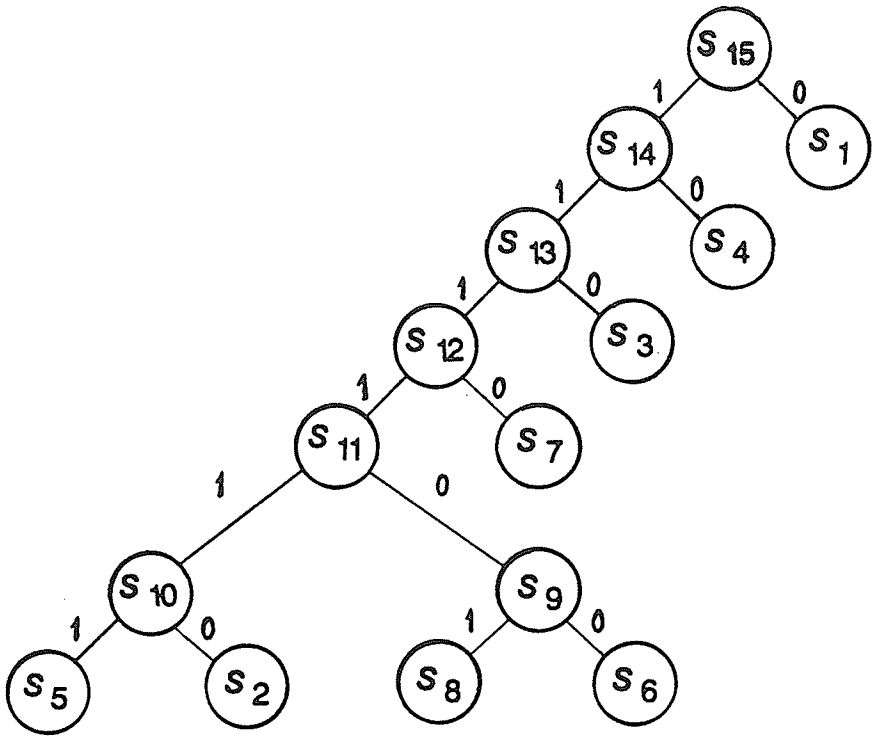


Figure 6.1: Huffman Code Tree Structure (from EXAMPLE 6-1)

branches (as shown in Figure 6.1). Each branch has associated with it a 1 or a 0 depending one whether the branch is to the left or right. To obtain a particular code word, the tree is followed from the top, through the branches until the symbol is reached. Each time a branch is encountered, the value of the branch (1 or 0) is recorded and added to the symbol code word. For example, to obtain the code word for s_7 one passes through the following symbols: starting at s_{15}--> s_{14} (1), s_{13} (1), s_{12} (1), s_7 (0). Therefore, the code word for s_7 is 1110 and the code word length is 4.

Performing the above operation results in the following code words:

Symbol	Code Word	Length(l)	Probability(p)	p x l
s_1	0	1	0.55	0.55
s_4	10	2	0.30	0.60
s_3	110	3	0.06	0.18
s_7	1110	4	0.05	0.20
s_2	111110	6	0.01	0.06
s_5	111111	6	0.01	0.06
s_6	111100	6	0.01	0.06
s_8	111101	6	0.01	0.06
				====

Average Length = 1.77 bits

(The units of average length are bits/symbol.)

For comparison, the entropy is calculated:

$$H(S) = - p_1\log_2 p_1 - p_2\log_2 p_2 \ldots - p_8\log_2 p_8$$

$$= 0.474 + 0.066 + 0.244 + 0.521 + 0.066 + 0.066 + 0.216 + 0.066$$

$$= 1.72 \text{ bits/symbol}$$

Therefore, the maximum possible reduction in bit-rate is

$$(1 - 1.72/3)\ 100 = 42.7\ \%$$

while Huffman coding achieves

$$(1 - 1.77/3)\ 100 = 41.0\ \%.$$

In this case, Huffman coding provides a reduction in code word length within 3 % of the entropy.

6.5 SHAFT CODING PROCEDURE

An example of a non-optimal source encoding algorithm is one developed by Paul D. Shaft [6]. This approach is of interest since it addresses a problem encountered in Huffman coding: source statistics are not always available or measurements of the source are not completely accurate. The Shaft approach uses an intuitive algorithm which demonstrates good performance over a range of inputs [6]. The algorithm involves one variable parameter which is adjusted to obtain a minimum average code word length. The procedure for 'Shaft' coding is as follows for a source with 2^r symbols:

1. As in Huffman coding, source symbols are rank ordered in decreasing probability.
2. The variable parameter n' is selected; n' must be less than r, otherwise the code words become simply r bits long.
3. The entire list of source symbols is divided into two sections, one section which is $2^{n'-1}$ symbols long and the other which contains the remaining symbols.
4. The section of $2^{n'-1}$ symbols is assigned a 0 while the remaining symbols are assigned a 1.

5. The $2^{n'-1}$ symbols are then divided in half with each half being assigned zeros and ones respectively; this continues until words are n' bits long (all the last digits in the $2^{n'-1}$ levels have alternate ones and zeros).
6. The above sequence of subdivision followed by digit assignment is repeated until the final $2^{n'-1}$ symbols have been assigned.

In the following example the procedure is illustrated. One should note that each subdivision differs in length from the successive subdivision by one bit except for the final two subdivisions. This is the case since the final subdivision requires no 0 prefix (assignment of a 0 to the $2^{n'-1}$ levels and a 1 to the remaining levels) since there are no levels remaining to assign 1s to.

EXAMPLE 6-2

Using the same source symbols and probability assignments as in Example 6-1 the following steps are performed in Shaft coding:

1. Step 1: As with Huffman coding the source symbols are rank ordered in decreasing probability

$s_1=0.55$

$s_4=0.30$

$s_3=0.06$

$s_7=0.05$

$s_2=0.01$

$s_5=0.01$

$s_6=0.01$

$s_8=0.01$

2. Step 2: The parameter n' must be selected. In order to obtain the value of n' which yields the lowest average code word length, n' is varied, in this case, from 1 to 2 (since three bits are used). The value $2^{n'-1}$ is computed, which establishes how each column is to be subdivided.

   n'=1 a) $2^{n'-1} = 1$

   b) subdivide the above column into an upper group of 1 symbol,

c) assign a 0 to the upper group and a 1 to the remaining symbols,

d) divide the upper group into halves and assign 0s to upper halves and 1s to the lower halves continuing until all upper levels have been assigned n' bits,

e) repeat b) through d) until the final subdivision assignments have been made;

		l	p	l x p
	s_1 0	1	0.55	0.55
1st subdivision	----			
	s_4 1 0	2	0.30	0.60
2nd subdivision	------			
	s_3 1 1 0	3	0.06	0.18
3rd subdivision	--------			
	s_7 1 1 1 0	4	0.05	0.20
4th subdivision	----------			
	s_2 1 1 1 1 0	5	0.01	0.05
5th subdivision	----------			
	s_5 1 1 1 1 1 0	6	0.01	0.06
6th subdivision	----------			
	s_6 1 1 1 1 1 1 0	7	0.01	0.07
7th subdivision	----------			
	s_8 1 1 1 1 1 1 1	7	0.01	0.07
				====

Average length = 1.78 bits

n'=2 $2^{n'-1} = 2$

		l	p	l x p
	s_1 0 0	2	0.55	1.10
	s_4 0 1	2	0.30	0.60
1st subdivision	------			
	s_3 1 0 0	3	0.06	0.18
	s_7 1 0 1	3	0.05	0.15
2nd subdivision	--------			
	s_2 1 1 0 0	4	0.01	0.04
	s_5 1 1 0 1	4	0.01	0.04
3rd subdivision	----------			
	s_6 1 1 1 X 0	4	0.01	0.04
	s_8 1 1 1 X 1	4	0.01	0.04
				====

Average length = 2.19 bits

Note: The final two code words contain X where normally a 0 prefix would be included if further subdivisions were possible. This prefix would serve to differentiate the current subdivided group from the succeeding group. Since there is no succeeding group, the prefix is dropped to decrease the code word length.

The value of n' which yields the lowest average length is 1. From Example 6-1

H(S)= 1.72 bits/symbol

Therefore, the maximum possible reduction in bit-rate is

$$(1 - 1.72/3)\ 100 = 42.7\ \%$$

while Shaft coding (with n'=1) achieves

$$(1 - 1.78/3)\ 100 = 40.7\ \%$$

which compares favourably with Huffman coding (41.0 %) and is within 3.5 % of the entropy.

6.6 CONDITIONAL SHAFT AND HUFFMAN CODING

Conditional entropy encoding involves the use of conditional probabilities to generate a variable-length code. Since conditional probabilities involve two entities, the present symbol and the conditional state, the resulting noiseless source code contains sets of variable-length symbols, one for each conditional state. Therefore, for 4-bit ADPCM and a first order Markov model, one obtains 2^4 or 16 sets of 16 variable-length codes. This yields a total of 256 code words (16x16) and is referred to as 4x4 conditional coding. Both conditional Shaft and conditional Huffman coding are reported in this study.

6.7 EXPERIMENTAL TEST SET-UP

Facilities to evaluate the performance of the Split-Band ADPCM coder were constructed around a University of Manitoba microcomputer (BB-II System) and the Millenium 9508S Microsystem Emulator emulating a Motorola 6802 microprocessor.

System Hardware

The hardware of the development station includes a Z80-based microcomputer with two 8 inch floppy disc drives and built-in keyboard and monitor. The computer runs under the CP/M disc operating system.

The Milllenium 9508S is a microprocessor emulator which can be configured to emulate one of many commercially available microprocessors. The system consists of a main processor and an emulation pod which connects to external hardware via a 40 pin connector, replacing the actual microprocessor. Communication between the 9508S and the BB-II System is via an RS232 type connection. The emulator/coder interface consists of a wire-wrapped circuit board containing a MC6821 Peripheral Interface Adaptor (PIA) and address decoding logic. As well, a socket and some support hardware connected to the PIA in a fashion identical to a MC6802 microprocessor serve as the connector to the emulator. The OKI ADPCM devices are connected to the peripheral ports of the PIA such

that one port receives the ADPCM data during encoding while the other port sends data to the ADPCM devices during the decoding process.

System Software

Several software programs have been developed on the BB-II in Motorola 6802 and Intel 8080 assembly language, and the Pascal programming language. Assembly language utility routines were initially written to facilitate file transfers between the BB-II and the 9508S. This allowed 6802 assembly language programs to be downloaded into the 9508S and executed. These routines were also used to acquire speech data and transfer them to the BB-II to be stored on disc for further processing.

The data acquisition system configuration is illustrated in Figure 6.2. The process of data acquisition is as follows:

1. A tape recorded version of the input speech is fed into an amplifier and then passed to the band-splitting filter bank.
2. The resulting two signals are converted to 4-bit ADPCM data by the OKI devices and passed to the PIA parallel interface.
3. At each sampling instant the emulator processor (executing a machine language program) loads a new data point and stores it in an internal memory location.

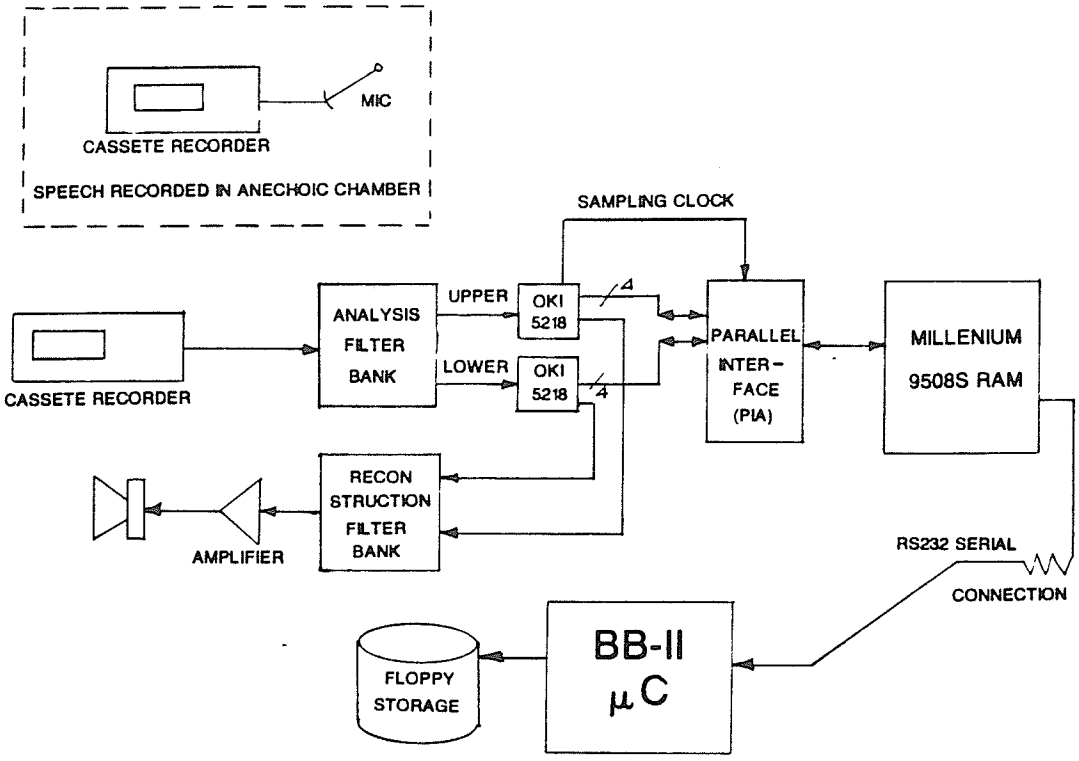


Figure 6.2: Data Acquisition System Configuration

4. When the speech passage is completed, the emulator is halted and the speech data which are stored in the emulator is then available for transfer to the BB-II system.

Assembly language routines for acquiring the coder output data are written in 6802 assembly language largely due to the availablity of that particular emulator pod. An emulator was incorporated into the system since it contains 64 K bytes of Random Access Memory (used as a data buffer) and has utility routines onboard for data transfers via its RS-232 port. Due to the limited memory available in the em-

ulator, the speech was encoded one sentence at a time and then compiled into one large record. To facilitate the transfer of encoded speech data to and from the disc files of the BB-II, Intel 8080 assembly language routines were written and executed on the BB-II.

6.8 ANALYSIS OF DATA

The acquired speech data, after being transferred to a disc file on the BB-II, were operated on by several programs written in the Pascal programming language. Data was partitioned into 4-bit and 8-bit catagories. The 4-bit category consists of an upper and lower channel designation. The 8-bit category involves the 4x4 conditional coding of the upper and lower bands. The entropy for both data catagories was computed as well as the reduction in storage requirements due to coding. The process of generating codes from the marginal values is illustrated in Figure 6.3. Each of the coding procedures as well as the entropy calculation yields a value in bits/symbol, an average length. A similar process is used to analyze and code the data in the conditional case. This process is depicted in Figure 6.4. Computer programs for performing the above calculations are listed in Appendix D.

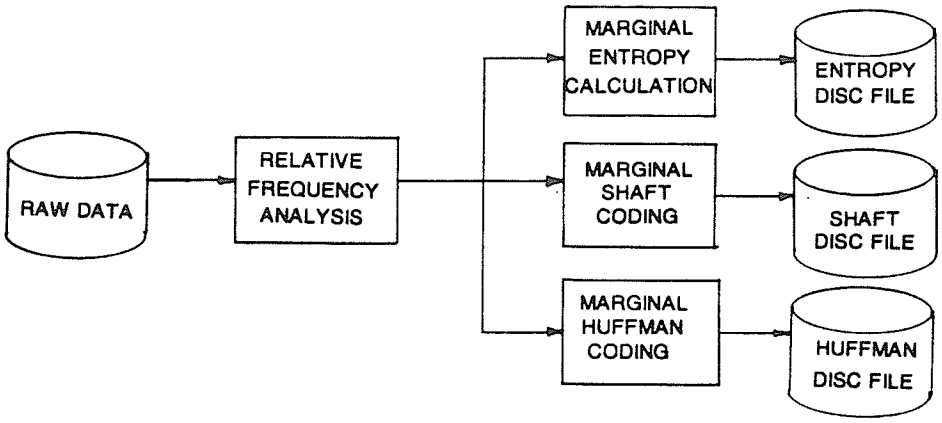


Figure 6.3: Marginal Coding Calculations

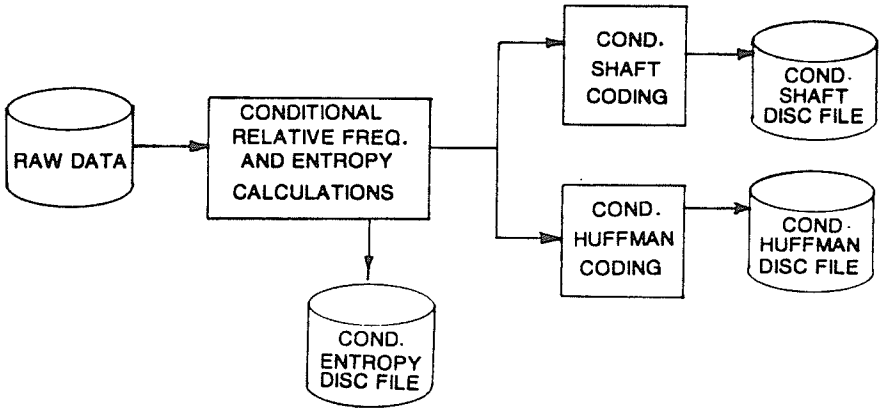


Figure 6.4: Conditional Coding Calculations

Chapter VII

RESULTS AND DISCUSSION

Two sets of ten syllabically balanced Harvard sentences (as specified by the IEEE Standards of Speech Quality Measurements [30]) spoken by one male and one female speaker as well as numerous speech passages obtained from a television broadcast were analyzed. Approximately 220 seconds of continuous speech (comprised of 4 male and 5 female speakers) were used as the input signal to the coding system. As described in the previous chapter, the data were collected and stored on disc on the BB-II microcomputer, where they were subsequently analyzed and entropy coded. As was also discussed, the data were partitioned into two catagories and several subcatagories prior to analysis. A summary of the

TABLE 7.1

Summary of Results of Entropy Encoding Split-Band ADPCM Speech

			$\dot{L}_{av}$(bits)	Saving(%)	H(S)(bits)	Max. Saving(%)
4-Bits	Huffman		3.43	14.2	3.37	15.8
	Shaft		3.52	12.0	3.37	15.8
8-Bits	Huffman	Conditional	5.61	29.9	5.54	30.8
	Shaft	Conditional	5.77	27.9	5.54	30.8

* Both upper and lower bands have equal values.

results of the coding procedures is given in Table 7.1

Marginal Entropy Coding

An average length within 2 % of the entropy is achieved using marginal Huffman coding. This result is consistent with other investigations reported in the literature involving entropy coding of both ADPCM and ADM encoded speech and television signals [4,29,and 31]. As well, space savings of approximately 14% are within the ranges reported in two of the aforementioned studies which used speech as the input signal [4, 29].

Shaft coding performed comparably in the 4-bit case (average length within 5 % of the entropy).

Conditional Entropy Coding

The conditional entropy calculation indicated that nearly twice the savings in space are possible using conditional coding. This is in fact the case for both Huffman and Shaft encodings. As well, the relationships between the entropy, Huffman coding and Shaft coding observed in the marginal case are preserved in the conditional case: Huffman coding is within 2% of the entropy and Shaft coding is within 5% of the entropy. Also, the Shaft average length remained within 3% of the Huffman average length.

In observing the histograms of the conditional probability estimates (given in Appendix B) it is noted that numerous zero entries are encountered throughout the sets of estimates. Initial calculations of entropy and average code word lengths using probability estimates which were not adjusted in the manner described in Chapter VI resulted in incorrect values (average lengths were less than the entropy - an impossible result). After adjustment, the relationship between the entropy and the average lengths complied with theoretical predictions. Based on the number of values lower than epsilon (46%), it appears that a greater number of trials must be performed before conditional probabilities can be accurately measured. In comparison, Bocci and Lo-Cicero used 150 seconds of speech in their analysis of entropy coding of Adpative Delta Modulated speech [29] while Papamichalis used approximately one hour of speech from 58 speakers (under a variety of sound quality conditions) [32]. Papamichalis was measuring the the conditional probabilities of linear prediction speech parameters.

An interesting property of this sub-band coder which the results indicate is that the entropy of both coder channels is essentially equal. Observation of the actual signals from the upper and lower bands indicated a significant difference in the energies of the two bands. This is further complicated by the uneven frequency response of the filter banks. Independent variation of the levels of the two bands

did not seem to affect the entropy values. On the other hand, overall input signal level did seem to affect the observed statistics of the coder outputs.

Chapter VIII

CONCLUSIONS AND RECOMMENDATIONS

An investigation into the application of entropy coding to a Split-Band ADPCM coder with a switched-capacitor filter bank has been presented. The main objective of this study was to determine the degree to which entropy coding (specifically Huffman and Shaft coding) reduces the amount of memory space required to store Split-Band ADPCM coded speech. Entropy coding of ADPCM and ADM coded speech has been reported in the literature [4, 29, 31], but the application of entropy coding to Split-Band ADPCM coded speech has not yet been addressed. Therein lies the uniqueness of this thesis.

As described in Chapter VII, the amount of space required to store Split-Band ADPCM coded speech is reduced by 14.2% when 4-bit Huffman coding is employed. A different variable-length coding scheme developed by P. D. Shaft performs comparably, providing a saving of 12%. Conditional coding yields a doubling in the space savings for both variable-length schemes. These values agree with those reported in other studies involving entropy coding of digitized speech.

Use of relative frequency analysis to estimate the coder output source symbol probabilities required the adjustment of the probability estimates according to a simple rule.

The adjustment is based on the use of Bernoulli's Theorem to calculate the number of trials required for a probability estimate to be within epsilon of the true value. The values which are less than epsilon are assigned worst case (equi-probable) values. This results in slightly higher but theoretically valid values for entropy and Huffman and Shaft average lengths computed using the adjusted probabilities.

The use of switched-capacitor filters for band-splitting proved to be a viable alternative to analog approaches. The configuration used for this study suffered from two major problems: poor frequency response, especially at the transition between sub-bands and audible clock residues. Both of these problems can be diminished by designing filters specifically for the band-splitting task. Cascading off-the-shelf filters (such as the Reticon devices) will not realize a filter function with the type of accuracy required for this application. By designing for the task, a transfer function as accurate as the existing analog Split-Band ADPCM filter banks can likely be realized. As well, by selecting appropriate component values in the design, the aliased clock residue problem can be eliminated by using a single input clock.

In considering that OKI CMOS single-chip devices exist to perform ADPCM coding and that switched-capacitor filters are easily integrated, the incorporation of such devices into single-chip Split-Band ADPCM coders and decoders appears to

be a realistic objective. Such devices could be readily incorporated into voice systems which currently use smaller bandwidth, less sophisticated encoding schemes such as PCM in telephony. This would result in an improvement in signal bandwidth at little or no cost in data rate, since a standard 64 kbit/sec data rate is maintained. For speech storage systems, such as store-and-forward meassaging, the further application of entropy coding to stored passages of speech could provide an additional space saving of up to 30%, based upon the results of this study.

Further study in the area of entropy coded sub-band coding may include the following topics:

1. Simulation of the entire sub-band coder may be performed with the incorporation of various quantizer structures (with emphasis on the effect of these structures on coder output symbol statistics) such as:
   a) first and second order weighted fixed predictors,
   b) first and second order adaptive predictors.
2. The following transmission requirements of variable-length and fixed length entropy encoding schemes maybe studied:
   a) transmit and receive buffering,
   b) performance in the presence of channel errors of various rates,

c) data multiplexing, framing and demultiplexing.

3. Entropy coding of multi-band sub-band coders (4, 6, and 8 bands) could be performed to observe the statistical behavior of the individual sub-bands alone and relative to each other.

4. Further investigation of the probability adjustment procedure for relative frequency analysis appears warranted. Adjustments based on smallest adjacent probability differences could be implemented in addition to the simple scheme described in this study.

Appendix A

AVERAGE CODE-WORD LENGTH AND REDUCTION OF BIT-RATE

Definitions:

N = total number of source samples processed.

r = number of bits per uncoded symbol.

n_i = total number of occurrences of the i^{th} symbol in the record of N data points.

$n_{i,j}$ = total number of occurrences of the i^{th} symbol preceeded by the j^{th} state (joint event).

p_i^m = i^{th} symbol marginal probability.

$p_{i/j}^c$ = i^{th} symbol conditional probability given the j^{th} conditional state.

l_i = length of the encoded i^{th} symbol (using marginal probabilities).

$l_{i,j}$ = length of the encoded i^{th} symbol associated with the j^{th} preceeding state.

L_{av}^m = average length of the code words encoded using marginal probabilities.

L_{avj}^c = average length of the encoded symbols associated with the j^{th} conditional state. There are 2^r such symbols.

L_{av}^{total} = average length of the L_{avj}^cs over all j states (2^r).

It is proved below that for probabilities calculated using relative frequency analysis, the average length for coded data using Huffman or Shaft encoding directly corresponds to the amount of space which is saved if the coding procedure is directly applied to the data. This is true for both marginal and conditional coding procedures.

Marginal Coding

In the marginal case the probability of an i^{th} symbol is (using relative frequency analysis)

$$p_i^m = \frac{n_i}{N} \tag{A.1}$$

The average length is defined as

$$L_{av}^m = \sum_{i=1}^{2^r} p_i^m \, l_i \tag{A.2}$$

Substituting (A.1) into (A.2)

$$L_{av}^m = \sum_{i=1}^{2^r} \frac{n_i}{N} \, l_i$$

$$= \frac{1}{N} \sum_{i=1}^{2^r} n_i \, l_i \tag{A.3}$$

The total amount of space required to store the encoded version of the data is given by

$$S_{total}^m = \sum_{i=1}^{2^r} n_i \, l_i \tag{A.4}$$

Therefore, from (A.3) and (A.4)

$$S_{total}^m = N \; L_{av}^m \tag{A.5}$$

Conditional coding

In the conditional case, two probabilities are defined:

$$marginal : p_i^m = \frac{n_i}{N} \tag{A.6}$$

and

$$conditional : p_{i/j}^c = \frac{n_{i,j}}{n_j} \tag{A.7}$$

The average length of the conditionally coded symbols for the conditional state j is given by

$$L^{c}_{avj} = \sum_{i=1}^{2^r} p^{c}_{i/j}\, l_{i,j}$$

$$= \sum_{i=1}^{2^r} \frac{n_{i,j}}{n_j}\, l_{i,j} \qquad \text{(A.8)}$$

The total average length is the combination of the conditional average lengths given by

$$L^{total}_{av} = \sum_{j=1}^{2^r} p^{m}_{j}\, L^{c}_{avj}$$

$$= \sum_{j=1}^{2^r} \frac{n_j}{N} \sum_{i=1}^{2^r} \frac{n_{i,j}}{n_j}\, l_{i,j}$$

$$= \frac{1}{N} \sum_{j=1}^{2^r} \sum_{i=1}^{2^r} n_{i,j}\, l_{i,j} \qquad \text{(A.9)}$$

The space required to store the i^{th} symbol given the j^{th} conditional state is

$$s_{l,j} = \sum_{i=1}^{2^r} n_{i,j}\, l_{i,j} \qquad \text{(A.10)}$$

The total space required for all j states is

$$S^{c}_{total} = \sum_{j=1}^{2^r} s_{i,j}$$

$$= \sum_{j=1}^{2^r} \sum_{i=1}^{2^r} n_{i,j}\, l_{i,j} \qquad \text{(A.11)}$$

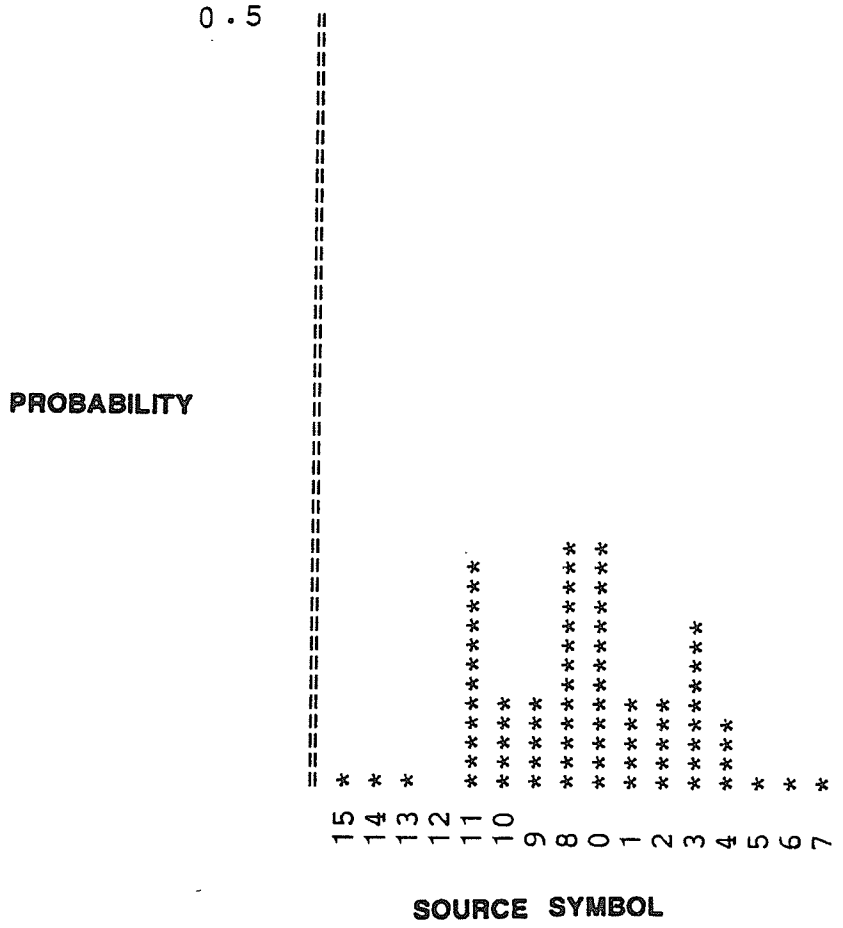
Therefore, from (A.10) and (A.11)

$$S^{c}_{total} = N\, L^{\ total}_{av} \qquad \text{(A.12)}$$

Appendix B

HISTOGRAMS

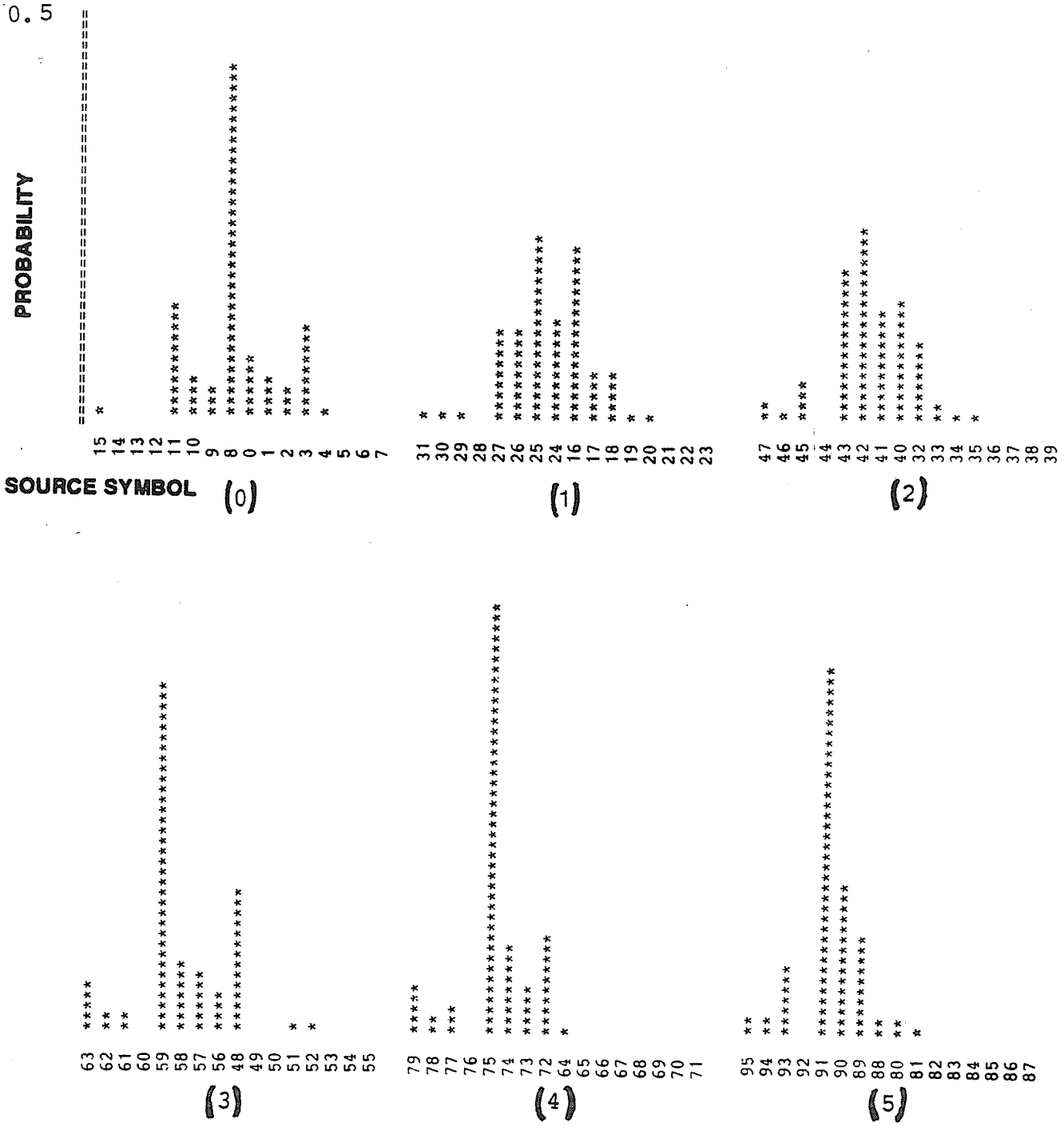
Marginal Probability Histogram



111 **
110 **
109 ********
108
107 **
106 ****************
105 ****************
104 ***
96 **
97
98
99
100
101
102
103
(6)
127 ****
126 **
125 ********
124
123 ********************************
122 *****
121 ****************************
120 ***
112 **
113
114
115
116
117
118
119
(7)
143 *
142
141
140
139 *******
138 ***
137 ****
136 ******
128 ************************************
129 ***
130 ****
131 ****
132 **********
133
134
135 *
(8)
159
158
157
156
155 **
154 ****
153 *****
152 ********************
144 **********
145 **********
146 **********
147 ******
148 ***
149 *
150 *
151 *
(9)
175
174
173
172
171 *
170 *
169 **
168 **********
160 **********
161 **********
162 **********
163 *********
164 *****
165 ****
166 **
167 **
(10)
191
190
189
188
187 **
186
185
184 **********
176 ******
177 *******
178 *******
179 **********
180 *****
181 **
182 **
183 *****
(11)
207
206
205
204
203
202
201
200
192
193
194
195
196
197
198
199
(12)

Conditional Probability Histograms

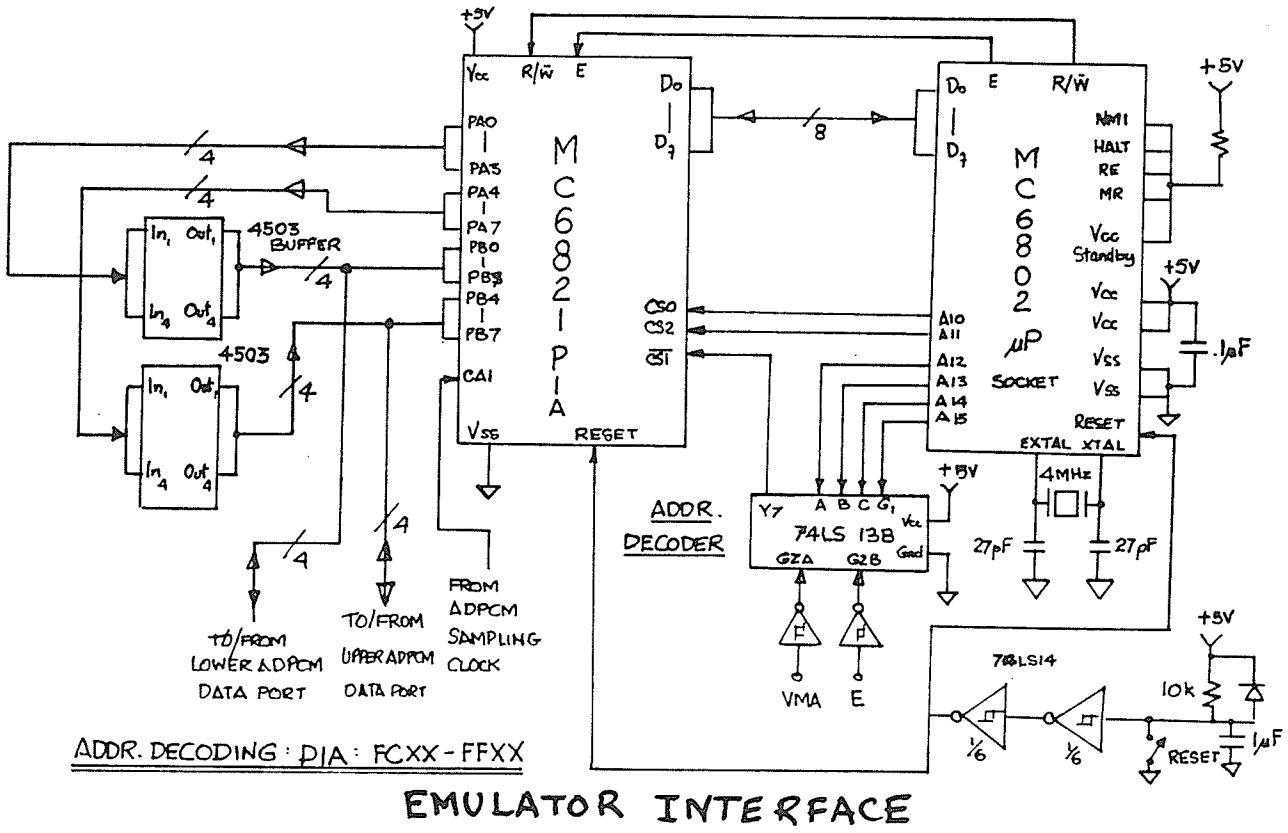
(by Conditional State depicted in brackets)

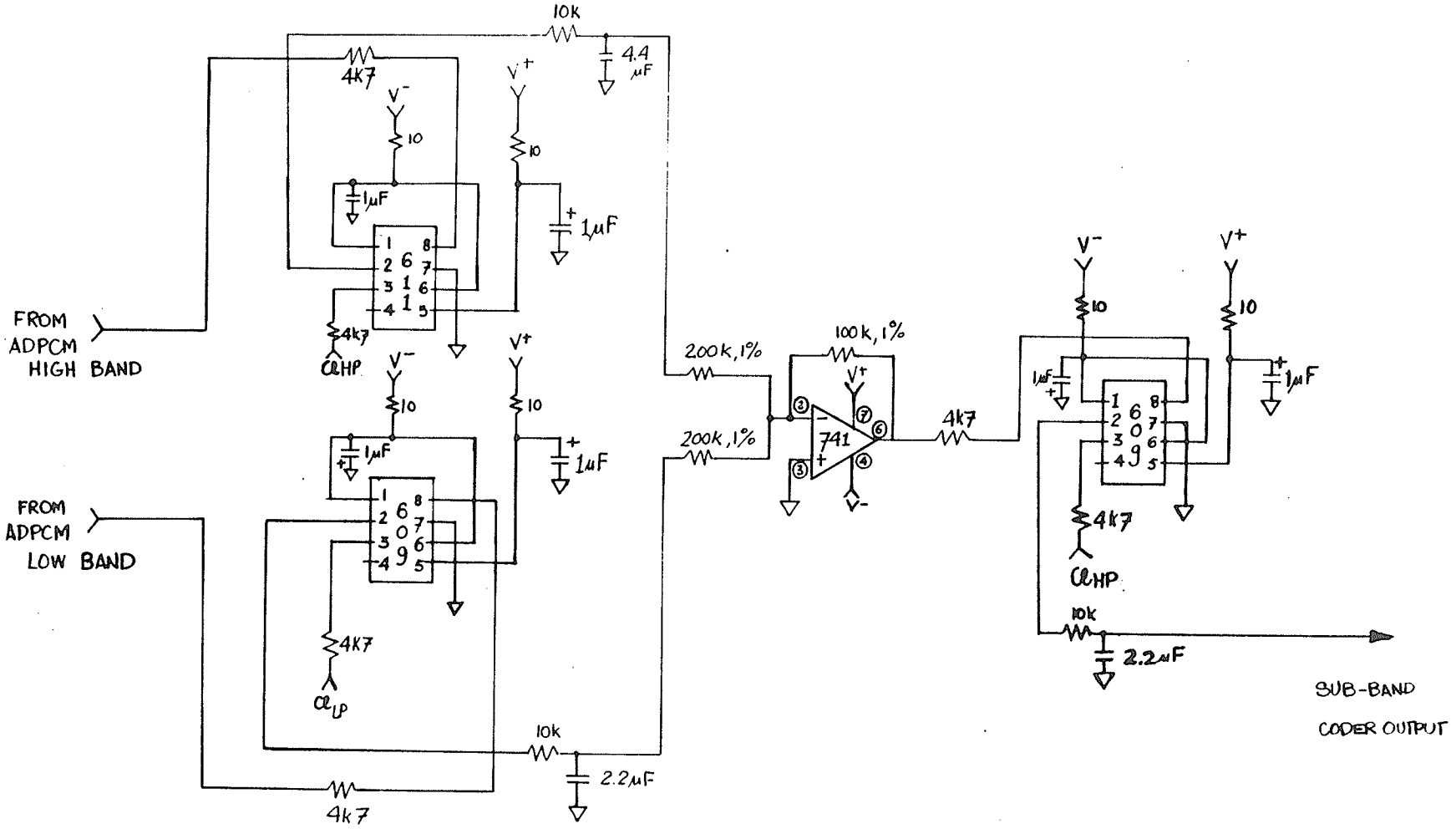


223
222
221
220
219
218
217 *
216 **
208 **
209 **********
210 ********************
211 ********************************
212 **********
213 *******
214 **
215 **
(13)
239
238
237
236
235
234
233
232 *
224 ***
225 **********
226 **********
227 ********************************
228 **********
229 *********
230 **
231 **
(14)
255
254
253
252
251
250
249
248 **
240 **
241 **********
242 *****
243 ********************************
244 ********
245 ********
246 **
247 ****
(15)

Appendix C

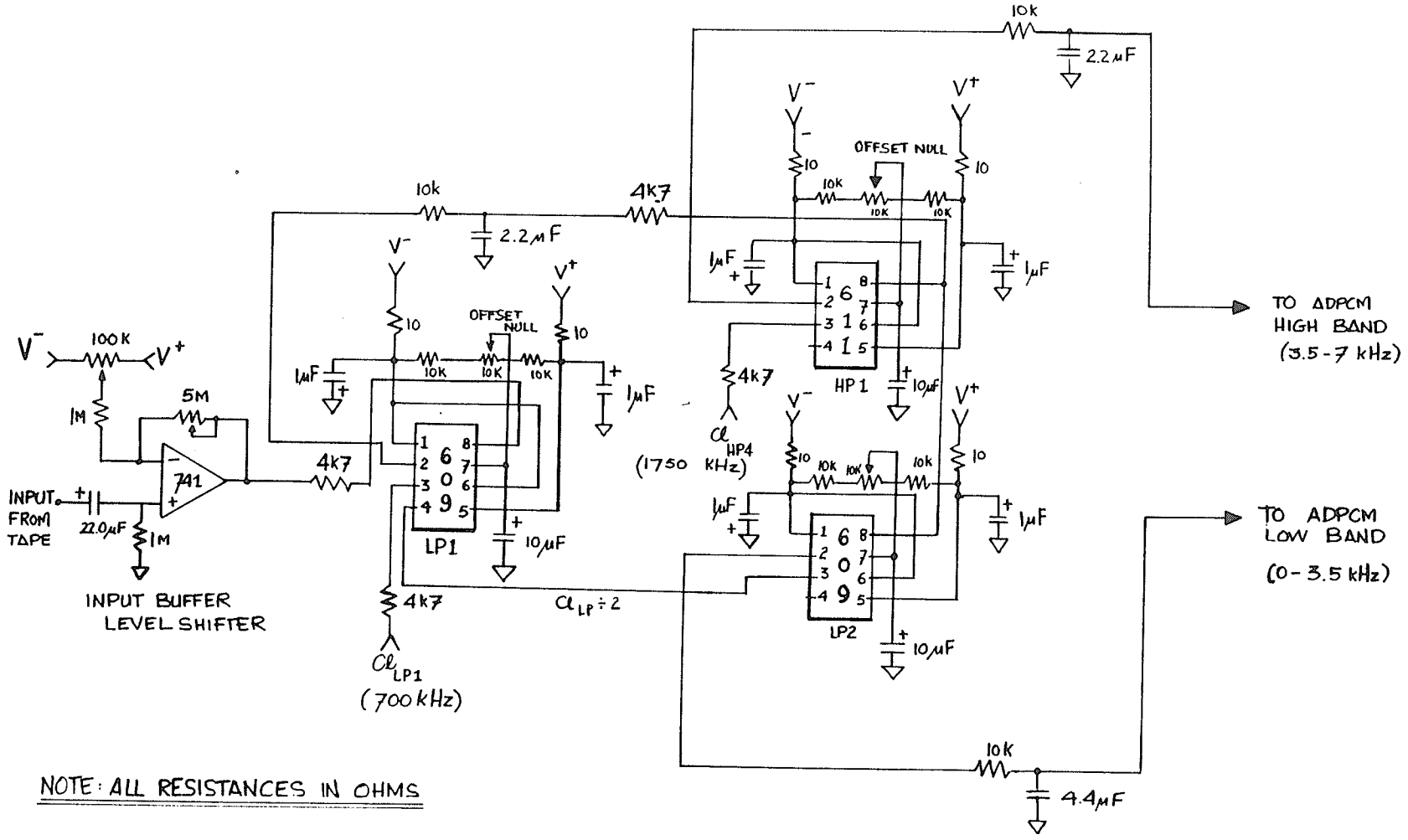
HARDWARE DIAGRAMS





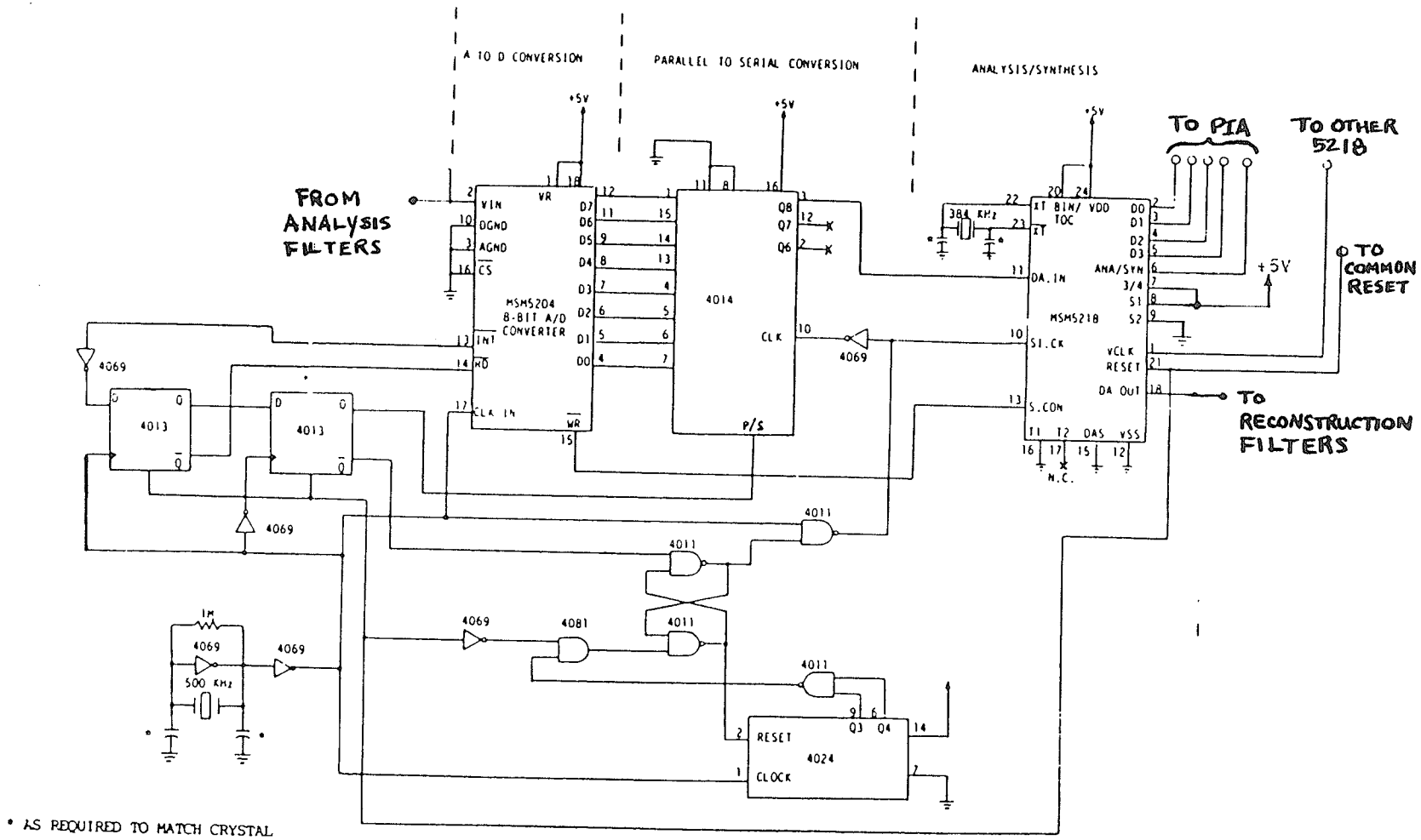
NOTE: ALL RESISTANCES IN OHMS

RECONSTRUCTION FILTERS



NOTE: ALL RESISTANCES IN OHMS

BAND SPLITTING FILTERS



* AS REQUIRED TO MATCH CRYSTAL

SCHEMATIC DIAGRAM OF MSM5218 ANALYSIS/SYNTHESIS SYSTEM
WITH 8-BIT A/D CONVERTER

Appendix D

SOFTWARE LISTINGS

```
*****************************************************************
{THIS PROGRAM CALCULATES THE RELATIVE FREQUENCY PROBABILITY}
{ESTIMATES OF AN INPUT ASCII-CODED HEX DATA FILE}
program rfreq5(input,output,infile,outfile1);
type
    datanum = string[1];
    hexnum = string[2];
    newstr = string[3];
    filenam = string[13];
    dual = record pattern: 0..$ff;
                  probability: real;
           end;

var
   name: array[1..30] of filenam;
   prob1: array[0..$ff] of real;
   prob2: array[0..15] of real;
   newprob1: array[0..$ff] of dual;
   newprob2: array[0..15] of dual;
   ent,total2,total1,temp,sumtotal: real;
   dot,n,position: integer;
   answer:hexnum;
   upper,lower,both,finished:boolean;
```

```
    data: array[1..2] of string[1];
    data0: char;
    dummy: char;
    data1: char;
    data2: char;
    str1: newstr;
    str2: hexnum;
    str3: hexnum;
    pointr: integer;
    count: integer;
    name1: filenam;
    name2: filenam;
    i: integer;
    j: integer;
    ji: integer;
    enn: integer;
    twotoenn:integer;
    counter: integer;
    code: integer;
    index: integer;
    infile: text;
    outfile,outfile1,outfile2,outfile3: text;
const
    cr = #$0d;
    lf = #$0a;

label loop,loop1,loop2;
```

```
begin
      upper:=false;
      lower:=false;
      both:=false;
      write(cr,lf);
      reset(input);
      write(cr,lf,'-- Relative Frequency Calculation  --',cr,lf);
      write(cr,lf);
      reset(input);
      write('Enter file(s) to be processed ',cr,lf);
      n:=1;
      finished:=false;
      while finished <> true do begin
            while not eoln(input) do begin
                  read(input,name[n]);
            end;
            write(cr,lf);
            reset(input);
            if name[n] = '!' then begin
               finished:=true;
               n:=n-1;
               end;
            n:=n+1;
            end;
      write(cr,lf,'Enter result file -- ');
      reset(input);
      while not eoln(input) do begin
```

```
      read(input,name2);
      end;
assign(outfile1,name2);
{here's where the work starts}
write(cr,lf,'Enter number of bits/symbol -- ');
reset(input);
while not eoln(input) do begin
      read(input,enn);
      end;
twotoenn:=round(exp(enn*ln(2)));
write(cr,lf);
write(output,cr,lf,'Please select one of the following
options:');
write(output,cr,lf);
write(output,' Upper nibble - U',cr,lf);
write(output,' Lower nibble - L',cr,lf);
write(output,cr,lf,'( B for both nibbles) ==> ');
reset(input);
while not eoln(input) do begin
      read(input,answer);
      end;
   if answer = 'U' then upper:=true;
   if answer = 'u' then upper:=true;
   if answer = 'L' then lower:=true;
   if answer = 'l' then lower:=true;
   if answer = 'B' then both:=true;
   if answer = 'b' then both:=true;
write(output,cr,lf);
```

```
write(cr,lf,'          ***** Working ******',cr,lf);
total1:=0;
total2:=0;
count:=0;
counter:=0;
for counter:=0 to 255 do
      begin
      prob1[counter]:=0.0;
      newprob1[counter].pattern:=0;
      newprob1[counter].probability:=0.0;
      end;
for counter:=0 to 15 do begin
      prob2[counter]:=0.0;
      newprob2[counter].pattern:=0;
      newprob2[counter].probability:=0.0;
      end;

for counter:=1 to n-1 do begin
      name1:=name[counter];
      assign(infile,name1);
      reset(infile);
      write(name1,cr,lf);
      if both = true then begin
         while not eof(infile) do begin
               while not eoln(infile) do begin
         loop: read(infile,data0);
               if data0 = ' ' then goto loop;
               data1:=data0;
```

```
            read(infile,data0);
            data2:=data0;
            str1:='$'+data1+data2;
            str2:='$'+data1;
            str3:='$'+data2;
            val(str1,index,code);
            prob1[index]:=prob1[index]+1;
            val(str2,index,code);
            prob2[index]:=prob2[index]+1;
            val(str3,index,code);
            prob2[index]:=prob2[index]+1;
            total1:=total1+1;
            total2:=total2+2;
            end;
            readln(infile);
            end;
end
else begin
if lower = true then read(infile,dummy);
      while not eof(infile) do begin
    loop1:  read(infile,data0);
            if data0 = ' ' then goto loop1;
            if data0 = cr then goto loop1;
            if data0 = lf then goto loop1;
            read(infile,dummy);
            read(infile,dummy);
            data1:=data0;
    loop2:  read(infile,data0);
```

```
                    if data0 = ' ' then goto loop2;
                    if data0 = cr then goto loop2;
                    if data0 = lf then goto loop2;
                    read(infile,dummy);
                    read(infile,dummy);
                    data2:=data0;
                    str1:='$'+data1+data2;
                    str2:='$'+data1;
                    str3:='$'+data2;
                    val(str1,index,code);
                    prob1[index]:=prob1[index]+1;
                    val(str2,index,code);
                    prob2[index]:=prob2[index]+1;
                    val(str3,index,code);
                    prob2[index]:=prob2[index]+1;
                    total1:=total1+1;
                    total2:=total2+2;
                    end;
        end;
        n:=n-1;
        close(infile);
end;
write(cr,lf);
if enn > 4 then begin
for i:=0 to twotoenn-1 do begin
    prob1[i]:=prob1[i]/total1;
    newprob1[i].probability:=newprob1[i].probability/total1;
    end;
```

```
end
else begin
for i:=0 to twotoenn-1 do begin
    prob2[i]:=prob2[i]/total2;
    newprob2[i].probability:=newprob2[i].probability/total2;
    end;
end;
{Rank ordering routine}
write(cr,lf,'Rank Ordering',cr,lf);
if enn <= 4 then begin
pointr:=twotoenn-1;
i:=twotoenn-1;
while pointr >= 0 do
begin
     if prob2[pointr] = 0.0 then
     begin
          newprob2[i].pattern:=pointr;
          i:=i-1;
     end;
     pointr:=pointr-1;
end;
pointr:=0;
while pointr < twotoenn do
begin
     i:=0;
     while i < twotoenn do
     begin
         if prob2[i] > newprob2[pointr].probability then
```

```
            begin
                newprob2[pointr].probability:=prob2[i];
                newprob2[pointr].pattern:=(i);
            end;
            i:=i+1;
        end;
        prob2[newprob2[pointr].pattern]:=0;
        pointr:=pointr+1;
end;
end
else begin
pointr:=twotoenn-1;
i:=twotoenn-1;
while pointr >= 0 do
begin
        if prob1[pointr] = 0.0 then
        begin
            newprob1[i].pattern:=pointr;
            i:=i-1;
        end;
        pointr:=pointr-1;
end;
pointr:=0;
while pointr < twotoenn do
begin
        i:=0;
        while i < twotoenn do
        begin
```

```
            if prob1[i] > newprob1[pointr].probability then
            begin
               newprob1[pointr].probability:=prob1[i];
               newprob1[pointr].pattern:=(i);
            end;
            i:=i+1;
      end;
      prob1[newprob1[pointr].pattern]:=0;
      pointr:=pointr+1;
end;
end;
write(cr,lf,'Writing to file',cr,lf);
rewrite(outfile1);
writeln(outfile1,enn);
if enn > 4 then begin
writeln(outfile1,total1);
for count:=0 to twotoenn-1 do begin
      write(outfile1,newprob1[count].probability,' ',
      newprob1[count].pattern,cr,lf);
      write(newprob1[count].probability,'  ',
      newprob1[count].pattern,cr,lf);
      end;
end
else begin
writeln(outfile1,total2);
for count:=0 to twotoenn-1 do begin
      write(outfile1,newprob2[count].probability,' ',
      newprob2[count].pattern,cr,lf);
```

```
            write(newprob2[count].probability,'  ',
            newprob2[count].pattern,cr,lf);
            end;
        end;
        write(cr,lf);
        close(outfile1);
        write(cr,lf);
        write('finished---- total data points processed = ');
        if enn > 4 then write(total1) else write(total2);
        write(cr,lf);

        write(#$07);

end.
```

```
*****************************************************************
program hufmn(input,output,infile,outfile);
{THIS PROGRAM CALCULATES THE AVERAGE WORDLENGTH OF A SOURCE}
{WHEN IT IS HUFFMAN ENCODED. THE INPUT FILE MUST CONTAIN AS THE}
{FIRST RECORD THE TOTAL NUMBER OF POSSIBLE SOURCE SYMBOLS (FOR }
{EIGHT BITS IT IS 256, FOR EXAMPLE). THE PROGRAM CONSTRUCTS A TREE}
{AND TRACES THROUGH IT TO DETERMINE THE LENGTHS OF EACH CODEWORD.}
{THE AVERAGE WORDLENGTH IS THEN COMPUTED AND DISPLAYED.}
{THE MAXIMUM NUMBER OF SOURCE SYMBOLS ALLOWED IS 256.}

type
    treerec= record
                    leftchild: integer;
                    rightchild: integer;
                    parent: integer;
             end;

   alphatree= record
                    probability: real;
                    leaf: integer;
                    number:integer;
              end;

   forestree= record
                    weight: real;
                    pattern:integer;
                    root: integer;
              end;
   filename= string[13];
```

```
    hexnum = string[2];

var
    tree : array [1..512] of treerec;
    alphabet : array [1..256] of forestree;
    forest : array [1..256] of forestree;
    length: array [1..256] of integer;
    n,x,limit,lastree,lastree1: integer;
    enn,total1:integer;
    total:real;
    prob,newsum:real;
    filenam1: filename;
    filenam2: filename;
    infile: text;
    outfile: text;
    answer:hexnum;
    lastnode,ending,symlength,zeros: integer;
const
      cr= #$0d;
      lf= #$0a;

label again;


procedure lightones(var least,second: integer);

var
   i:integer;

begin
```

```
      if forest[1].weight <= forest[2].weight then
         begin least:=1; second:= 2 end
      else
         begin least:=2; second:=1 end;

      for i:=3 to lastree do begin
          if forest[i].weight < forest[least].weight then
             begin second:=least; least:=i end
          else if forest[i].weight < forest[second].weight then
             second:=i;
          end;

end {lightones};

procedure print;

     var
        n: integer;
    begin
    for n:=1 to lastree do
    begin
         write(lst,forest[n].weight,' ',forest[n].root,' ',
         forest[n].pattern,cr,lf);
    end;
    n:=1;
    while n <= lastnode do
       begin
            write(lst,#$0d,#$0a);
            write(lst,tree[n].leftchild,' ',tree[n].parent,' ',
            tree[n].rightchild,' ');
```

```
            {writeln(outfile,#$0d,#$0a);
            writeln(outfile,tree[n].leftchild,' ',tree[n].parent,' ',
            tree[n].rightchild,' ');}
            n:=n+1;
        end;
end{print};

function create(lefttree,righttree:integer):integer;
        begin
            lastnode:=lastnode+1;
            tree[lastnode].leftchild:=forest[lefttree].root;
            tree[lastnode].rightchild:=forest[righttree].root;
            tree[lastnode].parent:=0;
            tree[forest[lefttree].root].parent:=lastnode;
            tree[forest[righttree].root].parent:=lastnode;
            create:=lastnode;
        end {create};

procedure Huff;

    var
       i,j,t:integer;
       newroot:integer;
    begin
        while lastree > 1 do begin
            lightones(i,j);
            newroot:=create(i,j);
            forest[i].weight:=forest[i].weight+forest[j].weight;
            forest[i].root:=newroot;
```

```
                forest[j]:=forest[lastree];
                lastree:=lastree-1;
          end;

   end{huff};

procedure tracetree;
var
   i,j: integer;
begin
     for i:=1 to 256 do
         begin
         length[i]:=0;
         end;
     i:=1;
     while i <= lastree1 do
           begin
           j:=i;
           while tree[j].parent <> 0 do
           begin
                 length[i]:=length[i]+1;
                 j:=tree[j].parent;
           end;
           i:=i+1;
           end;
 end{tracetree};

begin
     again:
```

```
reset(input);
write(cr,lf);
write(output,'Huffman Wordlength Calculator',#$0d,#$0a);
write(output,'Input Filename ==> ');
     while not eoln(input) do begin
      read(input,filenam1);
      end;
assign(infile,filenam1);
write(output,#$0d,#$0a);
write(output,'Output Filename ==> ');
reset(input);
while not eoln(input) do
      begin
      read(input,filenam2);
      end;
assign(outfile,filenam2);
reset(infile);
rewrite(outfile);
readln(infile,enn);
readln(infile,total);
write(output,#$0d,#$0a);
n:=1;
lastree:=0;
while not eof(infile) do
      begin
      readln(infile,forest[n].weight,forest[n].pattern);
      forest[n].root:=n;
      alphabet[n]:=forest[n];
```

```
        {write(forest[n].weight,' ',forest[n].root,' ',
        forest[n].pattern,cr,lf);}
        lastree:=lastree+1;
        if forest[n].weight = 0.0 then lastree:=lastree-1;
        n:=n+1;
        end;
        write(lastree,cr,lf);
write(output,'HUFFING',#$0d,#$0a);
for n:=1 to lastree do begin
tree[n].leftchild:=0;
tree[n].rightchild:=0;
tree[n].parent:=0;
end;
lastnode:=lastree;
ending:=lastree;
limit:=lastree;
lastree1:=lastree;
huff;
write(output,#$0d,#$0a);
writeln(outfile,#$0d,#$0a);
tracetree;
x:=1;
newsum:=0;
while x <= limit do
        begin
        newsum:=(alphabet[x].weight*length[x])+newsum;
        x:=x+1;
        end;
```

```
      write(cr,lf,((newsum/enn)*100),'% space used',cr,lf);
      total1:=round((1-(newsum/enn))*100);
      write(output,'The average wordlength for source ',filenam1,
      ' is ',newsum,' bits/symbol',cr,lf);
      write(cr,lf,'A total of ',total1,'% is saved thru coding  ',
      cr,lf);
      writeln(outfile,#$0d,#$0a,'The average wordlength for source ',
      filenam1,' is ',newsum,' bits/symbol',cr,lf);
      writeln(outfile,cr,lf,'A total of ',total1,'% is saved thru codi
      cr,lf);
      close(outfile);
      close(infile);
      write(#$07);
      write(cr,lf,'Do you wish to go again? (Y/N) ');
      reset(input);
      while not eoln(input) do begin
            read(input,answer);
            end;
      if answer = 'Y' then goto again;
      if answer = 'y' then goto again;

end.
```

```
*****************************************************************
{THIS PROGRAM ACCEPTS A DATA FILE WHICH HAS BEEN PROCESSED BY}
{RFREQ.COM AND CONTAINS THE RANK ORDERED RELATIVE FFREQUENCIES}
{OF A SOURCE.}
{THE LENGTH OF THE SHAFT CODEWORD IS COMPUTED FOR EACH SOURCE SYMBOL}
{AND THEN THE AVERAGE LENGTH OF THE SHAFT ENCODED DATA IS COMPUTED}
{FOR VALUES OF n' FROM 1 TO 8.}
{THE MINIMUM AVERAGE LENGTH IS THEN STORED IN THE OUTFILE}
program shaft(input,output,infile,outfile);
type
    hexnum = string[2];
    filenam = string[13];
    dual = record pattern: 0..$ff;
                  probability: real;
                  length: integer;
           end;

var
   prob: array[0..$ff] of real;
   newprob: array[0..$ff] of dual;
   total,total2: integer;
   total1,total3:real;
   nprime: integer;
   newn: integer;
   n: integer;
   primeval: integer;
   count: integer;
   i: integer;
```

```
    j: integer;
    t: integer;
    counter: integer;
    laverage: real;
    name1: filenam;
    name2: filenam;
    index: integer;
    infile: text;
    outfile: text;
    answer: hexnum;

const
    cr = #$0d;
    lf = #$0a;

label  again;

begin
     again:
     write(cr,lf);
     reset(input);
     write('-- Shaft Code Average Codeword Length Calculation  --',
     cr,lf);
     write('Enter data file to be processed ==> ');
     while not eoln(input) do
           begin
           read(input,name1);
           end;
     assign(infile,name1);
```

```
write(cr,lf);
reset(input);
write('Enter result file ==> ');
while not eoln(input) do
      begin
      read(input,name2);
      end;
{write(cr,lf,'Enter the value of n" ==>');
reset(input);
while not eoln(input) do
      begin
      read(input,nprime);
      end;}
assign(outfile,name2);
rewrite(outfile);
reset(infile);
readln(infile,n);
readln(infile,total1);
write(cr,lf,'          ***** Shafting ******',cr,lf);
total3:=(total1*n)/8;
write('total space required (uncoded) = ',total3,' bytes',cr,lf)
write(outfile,'total space required (uncoded) = ',total3,' bytes
cr,lf);
total:=0;
count:=0;
counter:=0;
while counter < 256 do
      begin
```

```
        prob[counter]:=0.0;
        newprob[counter].pattern:=0;
        newprob[counter].length:=0;
        newprob[counter].probability:=0.0;
        counter:=counter+1;
        end;
        i:=0;
while not eof(infile) do
begin
     readln(infile,newprob[i].probability,newprob[i].pattern);
     i:=i+1;
end;
{compute total number of subdivisions}
for nprime:=1 to 7 do
     begin
     primeval:=round(exp((nprime-1)*ln(2)));
     t:=round(exp(n*ln(2))/exp((nprime-1)*ln(2)));
     i:=0;
     count:=1;
     newn:=nprime;
     while count < t do
            begin
            j:=0;
            while j < primeval do
                begin
                     newprob[(i+j)].length:=newn;
                     j:=j+1;
                end;
```

```
            i:=i+primeval;
            newn:=newn+1;
            count:=count+1;
            end;
    j:=0;
    while j < primeval do
        begin
            newprob[(i+j)].length:=newprob[((i-1)+j)].length;
            j:=j+1;
        end;
        i:=0;
        laverage:=0.0;
        for i:=0 to 255 do
            begin
            laverage:=laverage+(newprob[i].probability*
            newprob[i].length);
            end;
        total2:=round((1-(laverage/n))*100);
        if laverage > n then total2:=0;
        write(outfile,cr,lf,'********SHAFT CODE********',cr,lf);
        write(outfile,cr,lf,'Average Length for n" = ',nprime,
        ' is ',laverage,cr,lf);
        write(cr,lf,'Average Length of Shaft Code',cr,lf);
        write('n" = ',nprime,'  Lav = ',laverage,cr,lf);
        write('percent space saved is = ',total2,'%',cr,lf);
        write(outfile,'percent space saved is = ',total2,'%',cr,
end;
```

```
        close(infile);
        close(outfile);
        write(cr,lf);
        write('--- finished ----');
        write(cr,lf);
        write('Do you wish to go again? (Y/N) ');
        reset(input);
        while not eoln(input) do begin
              read(input,answer);
              end;
        if answer = 'Y' then goto again;
        if answer = 'y' then goto again;

end.
```

```
*****************************************************************
{THIS PROGRAM COMPUTES THE CONDITIONAL ENTROPY, THE MARGINAL PROB.
AND THE CONDITIONAL PROB. OF AN INPUT FILE}
{THE MEASURED PROBABILITIES WHICH ARE SMALLER THAN EPSILON
ARE ADJUSTED AS DESCRIBED IN THE TEXT}


program ncon(input,output,infile,outfile1,outfile2);
type
    hexnum = string[2];
    newstr = string[3];
    filenam = string[13];
    dual = record pattern: 0..$ff;
                  probability: real;
           end;

var
   name: array[1..30] of filenam;
   prob1: array[0..$ff] of real;
   tpr1: array[0..$ff] of real;
   tpr2: array[0..15] of real;
   prob2: array[0..15] of real;
   newprob1: array[0..$ff] of dual;
   newprob2: array[0..15] of dual;
   ent,totalC,totalM,temp,sumtotal1,sumtotal2,fixup,epsilon,probtot:
   real;
   dot,n,position,epscnt: integer;
   answer:hexnum;
   upper,lower,both,trans,finished:boolean;
```

```
    data: char;
    name2,name3,name4: filenam;
    dummy: char;
    data1: char;
    data2: char;
    str1: newstr;
    str2: hexnum;
    str3: hexnum;
    pointr: integer;
    count: integer;
    name1: filenam;
    i: integer;
    j: integer;
    ji,ij,k: integer;
    counter: integer;
    code: integer;
    index: integer;
    infile: text;
    outfile,outfile1,outfile2,outfile3: text;
const
    cr = #$0d;
    lf = #$0a;

label loop,loop1;

begin
     upper:=false;
     lower:=false;
```

```
both:=false;
trans:=false;
write(cr,lf);
reset(input);
write(cr,lf,'-- Conditional Entropy Calc [Rel Freq. Output
and Epsilon Compensation]   --',cr,lf);
write(cr,lf);
reset(input);
write('Enter epsilon ==> ');
while not eoln(input) do begin
      read(input,epsilon);
      end;
write(cr,lf);
reset(input);
write('Enter file(s) to be processed ',cr,lf);
n:=1;
finished:=false;
while finished <> true do begin
      while not eoln(input) do begin
            read(input,name[n]);
      end;
      write(cr,lf);
      reset(input);
      if name[n] = '!' then begin
         finished:=true;
         n:=n-1;
         end;
      n:=n+1;
```

```
      end;
reset(input);
write(cr,lf,'Enter conditional result DTA file -- ');
while not eoln(input) do begin
      read(input,name2);
      end;
write(cr,lf);
reset(input);
write(cr,lf,'Enter marginal result DTA file -- ');
while not eoln(input) do begin
      read(input,name4);
      end;
write(cr,lf);
reset(input);
write(cr,lf,'Enter result ENT file -- ');
while not eoln(input) do begin
      read(input,name3);
      end;
write(cr,lf);
{here's where the work starts}
write(cr,lf);
write(output,cr,lf,'Please select one of the following options:'
write(output,cr,lf);
write(output,' Upper nibble - U',cr,lf);
write(output,' Lower nibble - L',cr,lf);
write(output,' Both nibbles -B',cr,lf);
write(output,cr,lf,' ==> ');
reset(input);
```

```
while not eoln(input) do begin
      read(input,answer);
      end;
   if answer = 'U' then upper:=true;
   if answer = 'u' then upper:=true;
   if answer = 'L' then lower:=true;
   if answer = 'l' then lower:=true;
   if answer = 'B' then both:=true;
   if answer = 'b' then both:=true;
write(output,cr,lf);
write(output,' Transpose nibbles? (Y/N)',cr,lf);
reset(input);
while not eoln(input) do begin
      read(input,answer);
      end;
   if answer = 'Y' then trans:=true;
   if answer = 'y' then trans:=true;
write(output,cr,lf);
write(cr,lf,'         ***** Working ******',cr,lf);
totalM:=1;
totalC:=0;
count:=0;
counter:=0;
for counter:=0 to 255 do
      begin
      prob1[counter]:=0.0;
      newprob1[counter].pattern:=0;
      newprob1[counter].probability:=0.0;
```

```
        end;
    for counter:=0 to 15 do begin
        prob2[counter]:=0.0;
        newprob2[counter].pattern:=0;
        newprob2[counter].probability:=0.0;
        end;

    for counter:=1 to n-1 do begin
        name1:=name[counter];
        assign(infile,name1);
        reset(infile);
        write(name1,cr,lf);
        if both = true then begin
           read(infile,data);
           str2:='$'+data;
           val(str2,index,code);
           prob2[index]:=prob2[index]+1;
           while not eof(infile) do begin
                 while not eoln(infile) do begin
                 data1:=data;
           loop: read(infile,data);
                 if data = ' ' then goto loop;
                 data2:=data;
                 if trans = true then begin
                 str1:='$'+data2+data1;
                 end else str1:='$'+data1+data2;
                 str2:='$'+data2;
                 val(str1,index,code);
```

```
            prob1[index]:=prob1[index]+1;
            val(str2,index,code);
            prob2[index]:=prob2[index]+1;
            totalM:=totalM+1;
            totalC:=totalC+1;
            end;
            readln(infile);
            end;
end
else begin
if upper = true then begin
      read(infile,data);
      read(infile,dummy);
      read(infile,dummy);
      end
      else begin
      read(infile,dummy);
      read(infile,data);
      read(infile,dummy);
      read(infile,dummy);
      end;
      str2:='$'+data;
      val(str2,index,code);
      prob2[index]:=prob2[index]+1;
      while not eof(infile) do begin
            data1:=data;
   loop1:   read(infile,data);
            if data = ' ' then goto loop1;
```

```
                    if data = cr then goto loop1;
                    if data = lf then goto loop1;
                    data2:=data;
                    read(infile,dummy);
                    read(infile,dummy);
                    if trans = true then begin
                    str1:='$'+data2+data1;
                    end
                    else str1:='$'+data1+data2;
                    str2:='$'+data2;
                    val(str1,index,code);
                    prob1[index]:=prob1[index]+1;
                    val(str2,index,code);
                    prob2[index]:=prob2[index]+1;
                    totalM:=totalM+1;
                    totalC:=totalC+1;
                    end;
        end;
        n:=n-1;
        close(infile);
end;
assign(outfile2,name2);
rewrite(outfile2);
assign(outfile3,name4);
rewrite(outfile3);
writeln(outfile3,4);
writeln(outfile3,totalM);
writeln(outfile2,8);
```

```
writeln(outfile2,totalC);
write(cr,lf);
sumtotal1:=0;
sumtotal2:=0;
for j:=0 to 15 do begin
    for i:=0 to 15 do begin
        ij:=(i*16)+j;
        prob1[ij]:=(prob1[ij]*totalM)/(prob2[i]*totalC);
        tpr1[ij]:=prob1[ij];
        end;
    end;
for i:=0 to 15 do begin
    prob2[i]:=prob2[i]/totalM;
    tpr2[i]:=prob2[i];
    end;

{Rank ordering routine}
write(cr,lf,'Rank Ordering');
pointr:=15;
i:=15;
while pointr >= 0 do
begin
     if prob2[pointr] = 0.0 then
     begin
          newprob2[i].pattern:=pointr;
          i:=i-1;
     end;
     pointr:=pointr-1;
```

```
end;
pointr:=0;
while pointr < 16 do
begin
    i:=0;
    while i < 16 do
    begin
        if prob2[i] > newprob2[pointr].probability then
        begin
           newprob2[pointr].probability:=prob2[i];
           newprob2[pointr].pattern:=(i);
        end;
        i:=i+1;
    end;
    prob2[newprob2[pointr].pattern]:=0;
    pointr:=pointr+1;
end;
{Adjust for values less than or equal to epsilon}
probtot:=0.0;
j:=0;
for i:=0 to 15 do begin
    if newprob2[i].probability > epsilon then begin
       probtot:= probtot+newprob2[i].probability;
       j:=j+1;
       end;
end;
if j >0 then begin
   if j <> 16 then begin
```

```
        probtot:=(1.0-probtot)/(16-j);
        while j < 16 do begin
              newprob2[j].probability:=probtot;
              tpr2[newprob2[j].pattern]:=probtot;
              j:=j+1;
        end;
    end;
end;
{Rank ordering by conditional state}
for j:=0 to 15 do begin
            pointr:=15;
            for i:=0 to 15 do begin
                ji:=(j*16)+i;
                if prob1[ji] = 0.0 then begin
                   newprob1[j*16+pointr].pattern:=ji;
                   pointr:=pointr-1;
                end;
            end;
end;
for j:=0 to 15 do begin
    for pointr:=0 to 15 do begin
            for i:=0 to 15 do begin
                ji:=(j*16)+i;
                if prob1[ji] > newprob1[j*16+pointr].probability
                then
                begin
                   newprob1[j*16+pointr].pattern:=ji;
                   newprob1[j*16+pointr].probability:=prob1[ji];
```

```
                end;
            end;
            prob1[newprob1[j*16+pointr].pattern]:=0.0;
    end;
end;
{Epsilon compensation}
probtot:=0.0;
for j:=0 to 15 do begin
    k:=0;
    for i:=0 to 15 do begin
        if newprob1[j*16+i].probability > epsilon then begin
           probtot:= probtot+newprob1[j*16+i].probability;
           k:=k+1;
        end;
    end;
    if k > 0 then begin
       if k <> 16 then begin
          probtot:=(1.0-probtot)/(16-k);
          while k < 16 do begin
                newprob1[j*16+(k)].probability:=probtot;
                tpr1[newprob1[j*16+(k)].pattern]:=probtot;
                k:=k+1;
          end;
       end;
    end;
end;
{ENTROPY CALC}
write(cr,lf,'Computing Entropy',cr,lf);
```

```
ent:=0;
for j:=0 to 15 do begin
    for i:=0 to 15 do begin
        ji:=(j*16)+i;
        temp:=tpr1[ji];
        if tpr1[ji] = 0.0 then temp:=1;
        ent:= ent - (tpr2[j]*temp*ln(temp)/ln(2));
    end;
end;
write('the entropy is ',ent);

for i:= 0 to 15 do begin
    sumtotal2:=sumtotal2+tpr2[i];
    end;
write(cr,lf,'The sum of the Marginal Probabilities is  ',
sumtotal2);
for i:=0 to 255 do begin
    sumtotal1:=sumtotal1+newprob1[i].probability;
    end;
write(cr,lf,'The sum of the Conditional Probabilities is  ',
sumtotal1);


assign(outfile1,name3);
rewrite(outfile1);
write(outfile1,cr,lf,'The conditional entropy (4x4) is: ',ent,
' bits/symbol');
if both = true then write(outfile1,cr,lf,'(Both nibbles)');
if lower = true then write(outfile1,cr,lf,'(Lower nibbles)');
```

```
if upper = true then write(outfile1,cr,lf,'(Upper nibbles)');
close(outfile1);
write(cr,lf,'Conditional Probabilities',cr,lf);
for count:=0 to 255 do begin
     write(outfile2,newprob1[count].probability,' ',
     newprob1[count].pattern,cr,lf);
     end;
write(cr,lf);
close(outfile2);
write(cr,lf,'Marginal Probabilities',cr,lf);
for count:=0 to 15 do begin
     write(outfile3,newprob2[count].probability,' ',
     newprob2[count].pattern,cr,lf);
     end;
close(outfile3);
write(cr,lf);
epscnt:=0;
for i:=0 to 255 do begin
    if newprob1[i].probability <= epsilon then epscnt:=epscnt+1;
    end;
write(cr,lf,'The number of conditional values < = epsilon = ',
epscnt);
write(cr,lf,'This is ',epscnt*100/256,' % of the total');
epscnt:=0;
for i:=0 to 15 do begin
    if newprob2[i].probability <= epsilon then epscnt:=epscnt+1;
    end;
write(cr,lf,'The number of marginal values < = epsilon = ',
```

```
        epscnt);
        write(cr,lf,'This is ',epscnt*100/16,' % of the total');
        write('finished---- total data points processed = ',totalM);
        write(cr,lf);

        write(#$07);


end.
```

```
*****************************************************************
{THIS PROGRAM CALCULATES THE  MARGINAL ENTROPY OF A SOURCE.}
{THE VALUE OF THE MARGINAL ENTROPY IS STORED IN THE OUTFILE.}
program entropy2(input,output,infile,outfile);
type
    hexnum = string[2];
    newstr = string[3];
    filenam = string[13];
var
   prob: array[0..$ff] of real;
   total: real;
   data: char;
   data1: char;
   data2: char;
   str1: newstr;
   ent: real;
   i,n,twoton:integer;
   count: integer;
   counter: integer;
   code: integer;
   name1: filenam;
   name2: filenam;
   index: integer;
   infile: text;
   outfile: text;
   total1:integer;
   answer:hexnum;
 const
```

```
   cr = #$0d;
   lf = #$0a;


label loop,next,again;

begin
     again:
     reset(input);
     write(cr,lf,' -- Entropy Calculation -- Use a DTA file -- ',cr,l
     write('Enter data file to be processed ==> ');
     while not eoln(input) do
           begin
           read(input,name1);
           end;
     assign(infile,name1);
     write(cr,lf);
     reset(input);
     write('Enter result file ==> ');
     while not eoln(input) do
           begin
           read(input,name2);
           end;
     assign(outfile,name2);
     {write(output,cr,lf,'Enter number of bits/symbol ==> ');
     reset(input);
     while not eoln(input) do begin
           read(input,n);
           end;}
```

```
reset(infile);
rewrite(outfile);
readln(infile,n);
readln(infile);
twoton:=round(exp(n*ln(2)));
total:=0;
count:=0;
counter:=0;
while counter < 256 do
      begin
      prob[counter]:=0.0;
      counter:=counter+1;
      end;
{while not eof(infile) do
begin
     while not eoln(infile) do
     begin
          loop:
          read(infile,data);
          if data = ' ' then goto loop;
          data1:=data;
          str1:='$'+data1;
          if n > 4 then
             begin
             read(infile,data);
             data2:=data;
             str1:='$'+data1+data2;
             end;
```

```
            val(str1,index,code);
            if index <= twoton then begin
               prob[index]:=prob[index]+1.0;
               total1:=total1+1;
               end;
            total:=total+1.0;
      end;
      readln(infile);
end;

if total= 0 then total:=1;
if total1 = 0 then total1:=1;}
i:=0;
while not eof(infile) do begin
       readln(infile,prob[i]);
       i:=i+1;
       end;
ent:=0;
while count < 256 do
begin
      if prob[count] <= 0 then goto next;
      prob[count]:=(prob[count])*(ln(prob[count])/ln(2.0));
      prob[count]:=(-1.0)*prob[count];
      next:
      ent:=ent+Prob[count];
      count:=count+1;
end;
write(outfile,'The Entropy of the source:',name1,' is ',ent,
```

```
        ' bits');
        close(infile);
        close(outfile);
        write(cr,lf);
        write('Finished---- the Entropy of the source: ',name1,' is ',
        ent,' bits');
        write(cr,lf,'Do you wish to go again? (Y/N) ');
        reset(input);
        while not eoln(input) do begin
              read(input,answer);
              end;
        if answer = 'Y' then goto again;
        if answer = 'y' then goto again;

end.
```

REFERENCES

1. J. L. Flanagan, "Computers that talk and listen: man-machine communication by voice," Proc. IEEE, vol. 64, no. 4, pp. 405-415, April 1976

2. J. L. Flanagan, et al, "Speech coding," IEEE Trans. Commun., vol. COM-27, no. 4, pp. 710-737, April 1979

3. R. E. Crochiere, S. A. Webber and J. L. Flanagan, "Digital coding of speech in sub-bands," Bell Syst. Tech. J., vol. 55, no. 8, pp. 1069-1085, Oct. 1976

4. K. Virupaksha and J. B. O'Neal, "Entropy-coded adaptive differential pulse-code modulation (DPCM) for speech," IEEE Trans. Commun., vol. COM-22, pp. 777-787, June 1974

5. D. A. Huffman, "A method for construction of minimum redundancy codes," Proc. IRE, vol. 40, pp. 1098-1101, Sept. 1952

6. P. D. Shaft, "A source encoding algorithm for quantized data," IEEE Trans. Commun.,vol. COM-22, pp.867-869, June 1974

7. J. L. Flanagan, Speech Analysis, Synthesis and Perception, New York:Springer-Verlag, 1972

8. L. R. Rabiner and R. W. Schafer, Digital Processing of Speech Signals, Englewood Cliffs, New Jersey:Prentice-Hall, 1978

9. J. N. Holmes, "A survey of methods for digitally encoding speech signals," Radio and Electronic Engineer, vol. 52, no. 6, pp. 267-276, June 1982

10. N. S. Jayant, "Digital coding of speech waveforms: PCM, DPCM, and DM Quantizers," Proc. IEEE, vol. 62, no. 5, pp. 611-632, May 1974

11. J. Makhoul, "Linear prediction: a tutorial review," Proc. IEEE, vol. 63, no. 4, pp. 561-580, April 1975

12. P.Cumminsky, N. S. Jayant and J. L. Flanagan, "Adaptive quantization in differential PCM coding of speech," Bell Syst. Tech. J., vol. 52, no. 7, pp. 1105-1119, Sept. 1973

13. R. E. Crochiere, "On the design of sub-band coders for low-bit-rate speech communication," Bell Syst. Tech. J., vol. 56, no. 5, pp. 747-770, May-June 1977

14. J. D. Johnston and D. J. Goodman, "Digital transmission of commentary-grade (7 kHz) audio at 56 or 64 kbits/s," IEEE Trans. Commun., vol. COM-28, no. 1, pp. 136-138, Jan. 1980

15. J. D. Johnston and R. E. Crochiere, "An all-digital "commentary grade" subband coder," J. Audio Eng. Soc., vol. 27 no. 11, pp.855-865, Nov. 1979

16. H. Blasbalg and R. Van Blerkom, "Message compression," IRE Trans. Space Electron. Telemetry, vol. 8, pp. 228-238, Sept. 1962

17. N. Abramson, Information Theory and Coding, New York:McGraw-Hill, 1963

18. R. G. Gallager, Information Theory and Reliable Communications. New York: Wiley, 1968

19. R. M. Fano, Transmission of Information: a Statistical Theory of Communication, New York: MIT Press, 1961

20. F. M. Reza, An Introduction to Information Theory. New York: McGraw-Hill, 1961

21. R. W. Hamming, Coding and Information Theory, Englewood Cliffs, New Jersey:Prentice-Hall, 1980

22. D. L. Cohn and J. L. Melsa, "The residual encoder-an improved ADPCM system for speech digitization," IEEE Trans. Commun., vol. COM-23, no. 9, pp. 935-941, Sept. 1975

23. S. W. Golomb, 'Run-length encodings," IEEE Trans. Inform. Theory, vol. IT-12, pp. 399-401, July 1966

24. F. Jelinik, "Buffer overflow in variable length coding of fixed rate sources," IEEE Trans. Inform Theory, vol. IT-14, no. 3, pp. 490-501, May 1968

25. F. Jelinik and K. S. Schneider, "On variable-length-to-block coding," IEEE Trans. Inform Theory, vol. IT-18, no. 6, pp. 756-774, Nov. 1972

26. R. E. Crochiere and L. R. Rabiner, Multirate Digital Signal Processing, Englewood Cliffs, New Jersey:Prentice-Hall, 1983

27. General Explanation of OKI ADPCM Speech Synthesis LSI (MSM5205RS), Tokyo: OKI Electric Ltd.,1982

28. A. Papoulis, Probability, Random Variables and Stochastic Processes, New York:McGraw-Hill, 1965

29. P. M. Bocci and J. L. LoCicero, "Bit rate reduction of digitized speech using entropy techniques," IEEE Trans. Commun., vol. COM-31,no. 3, pp. 424-430, March 1983

30. "IEEE recommended practice for speech quality measurements," IEEE Trans. Audio Electroacoust.,vol. AU-17, pp. 225-246, Sept. 1969

31. Shri K. Goyal and J. B. O'Neal, "Entropy coded differential pulse-code modulation systems for television," IEEE Trans. Commun., vol. COM-23, pp. 660-666, June 1975

32. P. E. Papamichalis, "Markov-Huffman coding of LPC parameters," IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-33, pp. 451-453, April 1985