NASA CR 114278

CODING SYSTEMS STUDY FOR HIGH DATA RATE TELEMETRY LINKS

By: K.S. Gilhousen
J.A. Heller
I.M. Jacobs
A.J. Viterbi

January 1971

Prepared under Contract No. NAS2-6024 by
LINKABIT CORPORATION
San Diego, California

for

AMES RESEARCH CENTER

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

1.0 INTRODUCTION

This report presents the results of a study of coding systems for high data rate links. The emphasis throughout is on convolutional codes. This is because high performance decoders exist for this class of codes, which are practical to implement at multi-megabit data rates.

The bit error rate, in the 10^{-3} to 10^{-8} range, vs. E_b/N_0 performance of many coding system configurations has been studied, through simulation and analytical techniques. Special attention has been paid to the sensitivity of performance to decoder parameters which affected complexity and cost significantly. These decoder parameters include

a) Code constraint length

b) Code rate (bandwidth expansion)

c) Data rate

d) Speed factor and buffer size (sequential decoding)

e) Path memory length, metric representation, and decision output selection (Viterbi decoding)

f) Receiver quantization.

A technique for obtaining and maintaining node synchronization and resolving phase ambiguities has been devised and analyzed. This technique is quite simple to implement. Simulation indicates that performance is more than adequate for the systems under consideration.

The report is divided into four major sections, plus the introduction. Each section deals with a distinct type of decoder or combination of decoders. Section 2 is concerned with Viterbi decoders. The important tradeoffs between performance and complexity are discussed here. The section concludes with a discussion of several methods of implementing Viterbi decoders at various data rates, and their relative complexity and cost.

Section 3 treats sequential decoding. Several techniques which either simplify or improve the performance of the Fano algorithm are discussed and evaluated. The implementation subsection emphasizes high speed rate 1/2, hard decision sequential decoding.

Section 4 evaluates a simple scheme for providing different levels of coding for data with different error rate requirements.

Finally, an interesting method of predecoding high rate received data using a Viterbi decoder, and shunting occasional difficult data to a sequential decoder, is discussed in Section 5. This technique holds out the possibility of efficiently decoding very high rate data using a relatively slow sequential decoder.

2.0 VITERBI DECODER STUDY

2.1 Introduction and Fundamentals

2.1.1 Code Representation. A convolutional encoder is a linear finite-state machine consisting of a K-stage shift register and n linear algebraic function generators. The input data, which is usually though not necessarily binary, is shifted along the register b bits at a time. An example with K=3, n=2, b=1 is shown in Fig. 2.1.

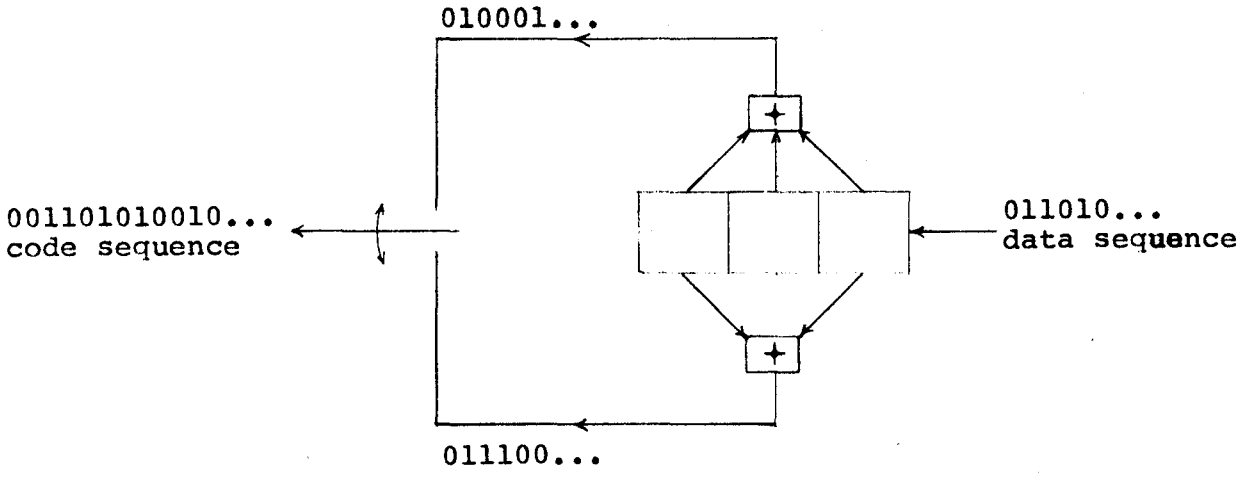


Fig. 2.1 Convolutional Coder for K=3, n=2, b=1

The binary input data and output code sequences are indicated on the diagram. The first three input bits, 0, 1, and 1, generate the code outputs 00, 11, and 01 respectively. We shall pursue this example to develop various representations of convolutional codes and their properties. The techniques thus developed will then be shown to generalize directly to any convolutional code.

It is traditional and instructive to exhibit a convolutional code by means of a tree diagram as shown in Figure 2.2.

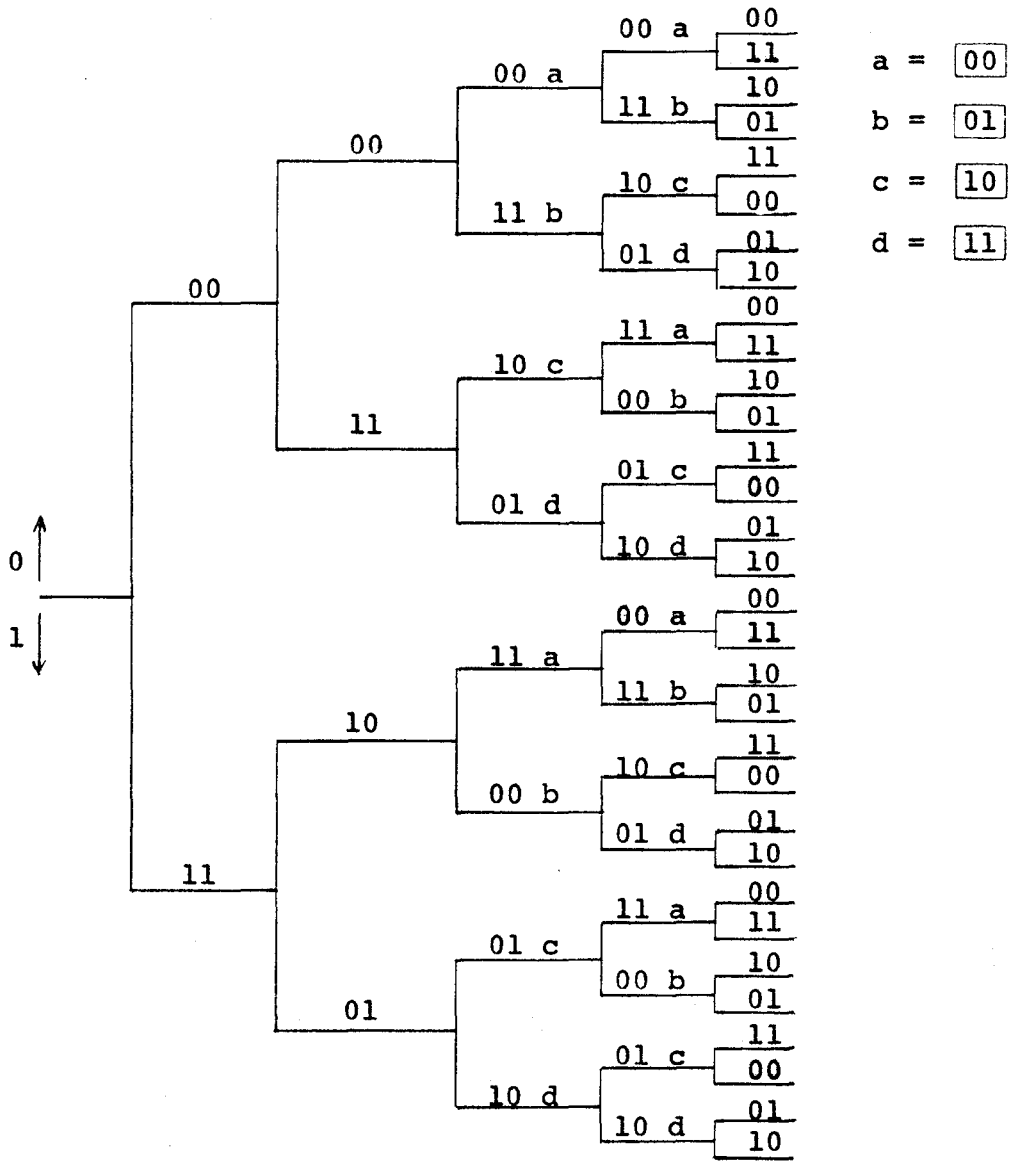


Figure 2.2. Tree Code Representation for Coder of Figure 2.1.

If the first input bit is a zero the code symbols are those shown on the first upper branch, while if it is a one the output code symbols are those shown on the first lower branch. Similarly, if the second input bit is a zero we trace the tree diagram to the next upper branch, while if it is a one we trace the diagram downward. In this manner all thirty-two possible outputs for the first five inputs may be traced.

From the diagram it also becomes clear that after the first three branches the structure becomes repetitive. In fact, we readily recognize that beyond the third branch the code symbols on branches emanating from the two nodes labelled "a" are identical, and similarly for all correspondingly labelled pairs of nodes. The reason for this is obvious from examination of the encoder. As the fourth input bit enters the coder at the right, the first data bit falls off on the left end and no longer influences the output code symbols. Consequently, the data sequences 100xy... and 000xy... generate the same code symbols after the third branch and, as is shown in the tree diagram, both nodes labelled "a" can be joined together.

This leads to redrawing the tree diagram as shown in Figure 2.3. This has been called a trellis diagram since a trellis is a tree-like structure with remerging branches. We adopt the convention here that code branches produced by a "zero" input bit are shown as solid lines and code branches produced by a "one" input bit are shown dashed.

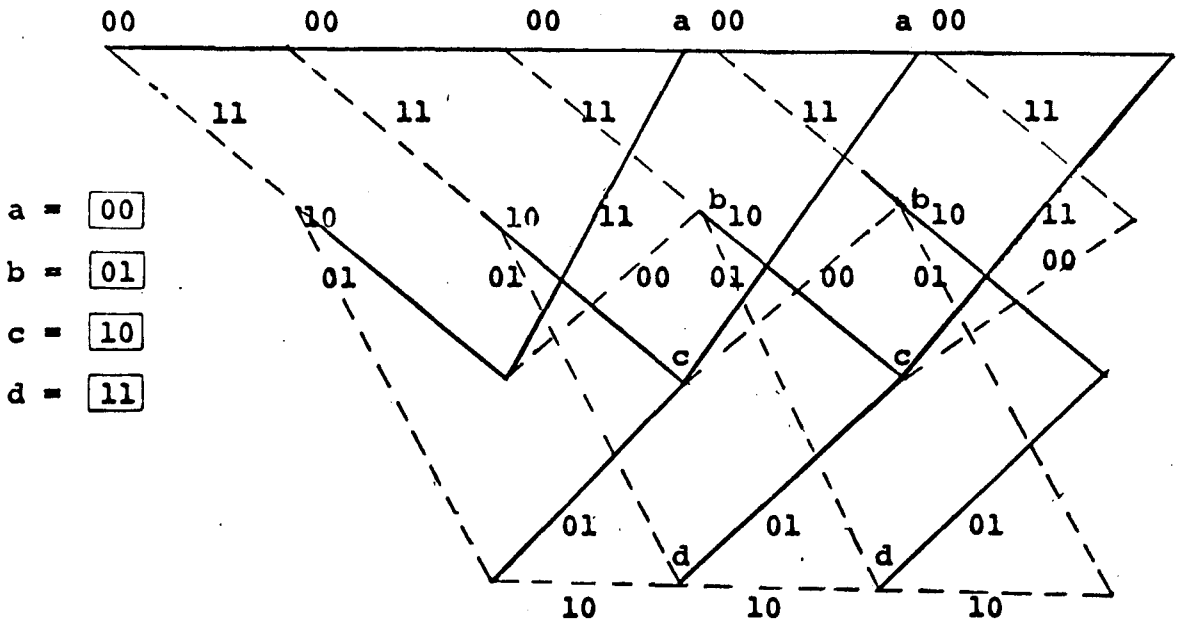


Figure 2.3 Trellis Code Representation for Coder of Figure 2.1.

The completely repetitive structure of the trellis diagram suggests a further reduction in the representation of the code to the state-diagram of Figure 2.4. The "states" of the state-diagram are labelled according to the nodes of the trellis diagram. However, since the states correspond merely to the last two input bits to the coder we may use these bits to denote the nodes or states of this diagram.

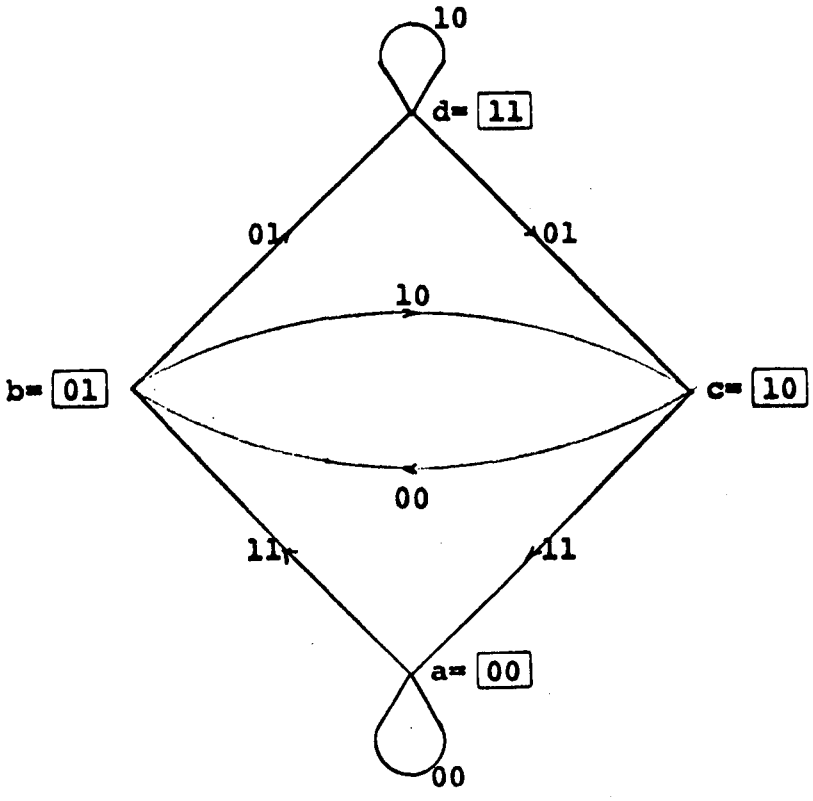


Figure 2.4. State-Diagram Representation for Coder of Figure 2.1.

We observe finally that the state-diagram can be drawn directly by observing the finite-state machine properties of the encoder and particularly the fact that a four-state directed graph can be used to represent uniquely the input-output relation of the eight-state machine. For the nodes represent the previous two bits while the present bit is indicated by the transition branch; for example, if the encoder (machine) contains 011,

this is represented in the diagram by the transition from state b = 01 to state d = 11 and the corresponding branch indicates the code symbol outputs 01.

2.1.2 The Viterbi Decoding Algorithm for the Binary Symmetric Channel (Hard Decision Outputs of a Gaussian Channel). On a binary symmetric channel, errors which transform a channel code symbol 0 to 1 or 1 to 0 are assumed to occur independently from symbol to symbol with probability p. If all input (message) sequences are equally likely, the decoder which minimizes the overall error probability for any code, block or convolutional, is one which examines the error-corrupted received sequence $y_1y_2\ldots y_j\ldots$ and chooses the data sequence corresponding to the transmitted code sequence $x_1x_2\ldots x_j\ldots$ which is closest to the received sequence in the sense of Hamming distance; that is the transmitted sequence which differs from the received sequence in the minimum number of symbols.

Referring first to the tree diagram, this implies that we should choose that path in the tree whose code sequence differs in the minimum number of symbols from the received sequence. However, recognizing that the transmitted code branches remerge continually, we may equally limit our choice to the possible paths in the trellis diagram of Figure 2.3. Examination of this diagram indicates that it is unnecessary to consider the entire received sequence (which conceivably could be thousands or millions of symbols in length) at one time in deciding upon the

most likely (minimum distance) transmitted sequence. In particular, immediately after the third branch we may determine which of the two paths leading to node or state "a" is more likely to have been sent. For example, if 010001 is received, it is clear that this is at distance 2 from 000000 while it is at distance 3 from 111011 and consequently we may exclude the lower path into node "a". For, no matter what the subsequent received symbols will be, they will effect the distances only over subsequent branches after these two paths have remerged and consequently in exactly the same way. The same can be said for pairs of paths merging at the other three nodes after the third branch. We shall refer to the minimum distance path of the two paths merging at a given node as the "survivor". Thus it is necessary only to remember which was the minimum distance path from the received sequence (or survivor) at each node, as well as the value of that minimum distance. This is necessary because at the next node level we must compare the two branches merging at each node level, which were survivors at the previous level for different nodes; e.g., the comparison at node "a" after the fourth branch is among the survivors of comparison at nodes "a" and "c" after the third branch. For example, if the received sequence over the first four branches is 01000111, the survivor at the third node level for node "a" is 000000 with distance 2 and at node "c" it is 110101, also with distance 2. In going from the third node level to the fourth the received sequence agrees precisely with the survivor from "c" but has distance 2 from the survivor

from "a". Hence the survivor at node "a" of the fourth level is the data sequence 1100 which produced the code sequence 11010111 which is at (minimum) distance 2 from the received sequence.

In this way, we may proceed through the received sequence and at each step preserve one surviving path and its distance from the received sequence, which is more generally called metric. The only difficulty which may arise is the possibility that in a given comparison between merging paths, the distances or metrics are identical. Then we may simply flip a coin as is done for block code words at equal distances from the received sequence. For even if we preserved both of the equally valid contenders, further received symbols would affect both metrics in exactly the same way and thus not further influence our choice.

This decoding algorithm was first proposed by Viterbi (Ref. 8) in the more general context of arbitrary memoryless channels. Another description of the algorithm can be obtained from the state-diagram representation of Figure 2.4. Suppose we sought that path around the directed state-diagram, arriving at node "a" after the kth transition, whose code symbols are at a minimum distance from the received sequence. But clearly this minimum distance path to node "a" at time k can be only one of two candidates: the minimum distance path to node "a" at time k-1 and the minimum distance path to node "c" at time k-1. The comparison is performed by adding the new distance accumulated in the kth transition by each of these paths to their minimum distances (metrics) at time k-1.

It appears thus that the state-diagram also represents a system diagram for this decoder. With each node or state, we associate a storage register which remembers the minimum distance path into the state after each transition as well as a metric register which remembers its (minimum) distance from the received sequence. Furthermore, comparisons are made at each step between the two paths which lead into each node. Thus four comparators must also be provided.

We defer the question of truncating the trellis and thereby making a final decision on all bits beyond L branches prior to the given branch until we have some additional properties of convolutional codes.

2.1.3 Distance Properties of Convolutional Codes. We continue to pursue the example of Figure 2.1 for the sake of clarity; in the next section, we shall easily generalize results. It is well known that convolutional codes are group codes. Thus there is no loss in generality in computing the distance from the all zeros code word to all the other code words, for this set of distances is the same as the set of distances from any specific code-word to all the others.

For this purpose, we may again use either the trellis diagram or the state-diagram. We first of all redraw the trellis diagram in Figure 2.5 labelling the branches according to their distances from the all zeros path. Now consider all the paths that merge with the all zeros for the first time at some arbitrary node "j".

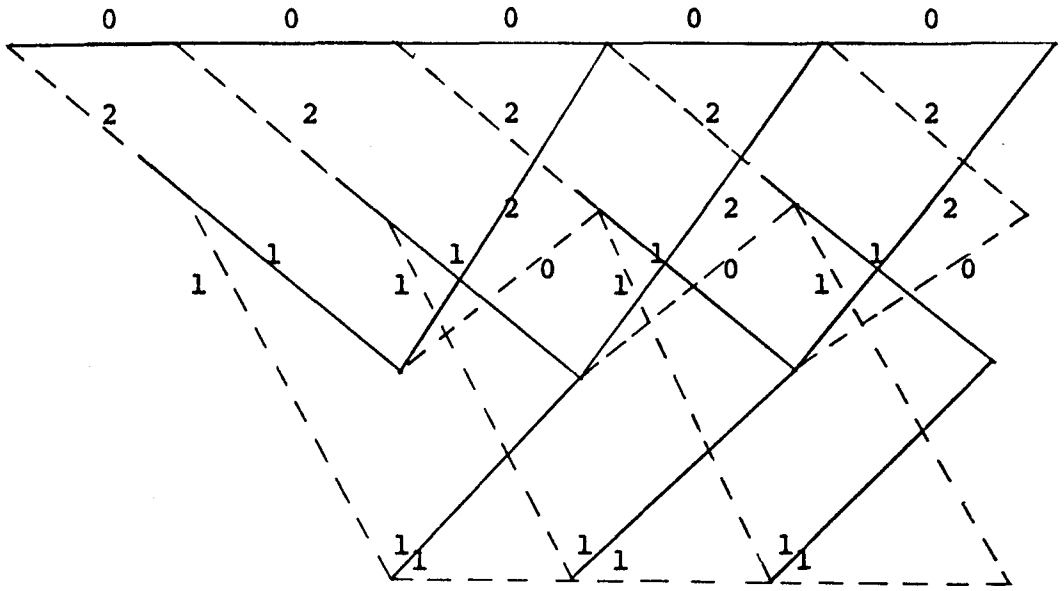


Figure 2.5 Trellis Diagram Labelled with Distances from all Zeros Path

It is seen from the diagram that of these paths, there will be just one path at distance 5 from the all zeros path and this diverged from it three branches back. Similarly there are two at distance 6 from it; one which diverged 4 branches back and the other which diverged 5 branches back, and so forth. We note also that the input bits for distance 5 path are 00..0100 and thus differ in only one input bit from the all zeros, while the distance 6 paths are 00..01100 and 00..010100 and thus differs in 2 input bits from the all zeros path. The minimum distance, sometimes called the minimum "free" distance, among all paths is thus seen to be 5. This implies that any pair of channel errors can be corrected, for two errors will cause the received

sequence to be at distance 2 from the transmitted (correct) sequence but it will be at least at distance 3 from any other possible code sequence. It appears thus that the distance of all paths from the all zeros (or any arbitrary) path can be so determined from the trellis diagram.

2.1.4 Generalization to Arbitrary Convolutional Codes. The generalization of these techniques to arbitrary binary-tree (b=1) convolutional codes is immediate. That is, a coder with a K stage shift register and n modulo-two adders will produce a trellis or state-diagram with 2^{K-1} nodes or states and each branch will contain "n" code symbols. The rate of this code is then

$$R = \frac{1}{n} \quad \frac{\text{bits}}{\text{code symbol}}$$

The example pursued in the previous sections had rate R=1/2. The primary characteristic of the binary-tree codes is that only two branches exit from and enter each node.

If rates other than 1/n are desired, we must make b>1, where b is the number of bits shifted into the register at one time. An example for K=2, b=2, n=3 and consequently rate R=2/3 is shown in Figure 2.6 and its state-diagram is shown in Figure 2.7.

It differs from the binary-tree codes only in that each node is connected to four other nodes, and for general "b" , it will be connected to 2^b nodes. Still all the preceding techniques including the trellis and state-diagram analysis are still applicable. It must be noted, however, that the minimum distance

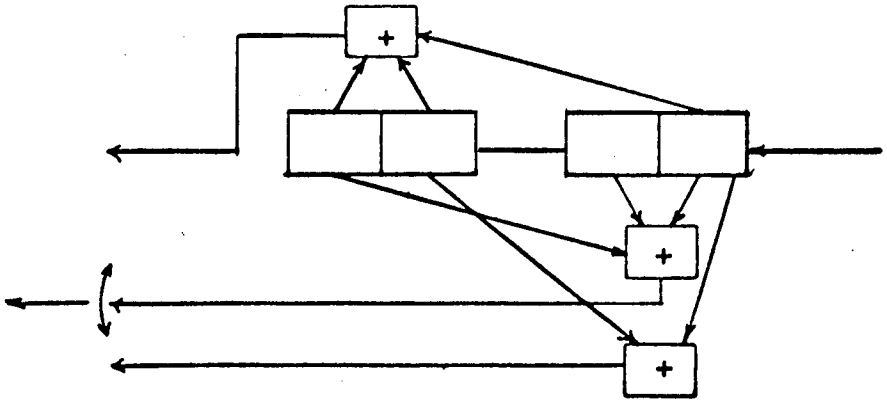


Fig. 2.6 Coder for K=2, b=2, n=3, R=2/3

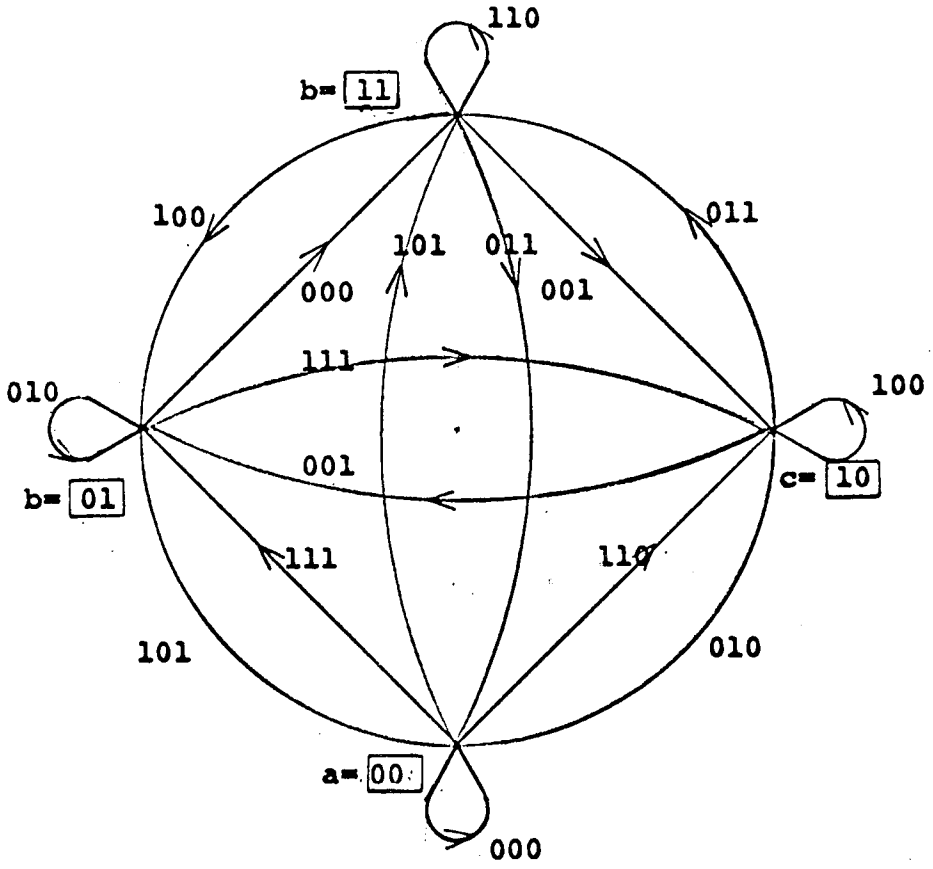


Fig. 2.7 State-Diagram for Code of Fig. 2.6.

decoder must make comparisons among all the paths entering each node at each level of the trellis and select one survivor out of four (or out of 2^b in general).

2.1.5 Systematic, Nonsystematic, and Catastrophic Convolutional Codes. The term systematic convolutional code refers to a code on each of whose branches one of the code symbols is just the data bit generating that branch. Thus a systematic coder will have its stages connected to only n-1 adders, the nth being replaced by a direct line from the first stage to the commutator. Figure 2.8 shows an R=1/2 systematic coder for K=3.

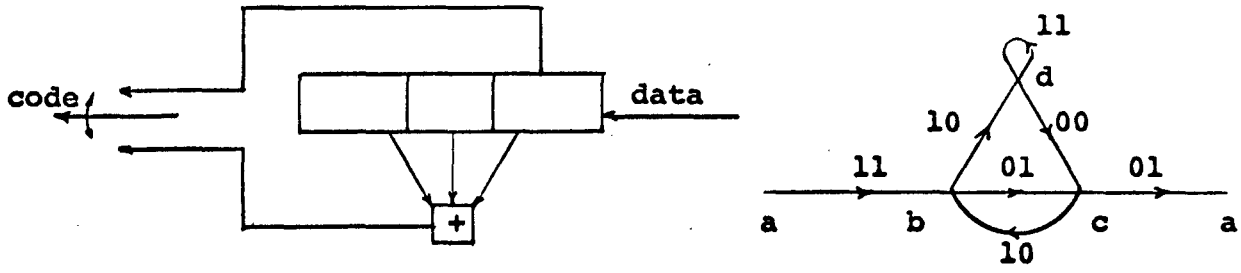


Figure 2.8 Systematic Convolutional Coder for K=3, R=1/2

It is well known that for group block codes, any nonsystematic code can be transformed into a systematic code which performs exactly as well. This is not the case for convolutional codes. The reason for this is that the performance of a code

on any channel depends largely upon the relative distance between codewords and particularly on the minimum free distance d, which is the minimum number of ones of any closed path through node "a". Eliminating one of the adders results in a reduction of "d". For example, the maximum minimum free distance systematic code for K=3 is that of Figure 2.8 and this has d=4, while the nonsystematic K=3 code of Figure 1.1 has minimum free distance d=5. Table 2.1 shows the maximum minimum free distance for systematic and nonsystematic codes for K=2 through 5.

Maximum Minimum Free Distance

K	Systematic	Nonsystematic
2	3	3
3	4	5
4	4	6
5	5	7

Table 2.1 Comparison of Systematic and Nonsystematic R=1/2 Codes

For large constraint lengths the results are even more widely separated.

A catastrophic error is defined as the event that a finite number of channel symbol errors causes an infinite number of data bit errors to be decoded. Massey and Sain (Ref. 9) have shown that a necessary and sufficient condition for a convolutional

code to produce catastrophic errors is that all of the adders have tap sequences, represented as polynomials, with a common factor.

In terms of the state-diagram it is easily seen that catastrophic errors can occur if, and only if, any closed loop path in the diagram has a zero weight (i.e., the exponent of D for the loop path is zero). To illustrate this, we consider the example of Figure 2.9.

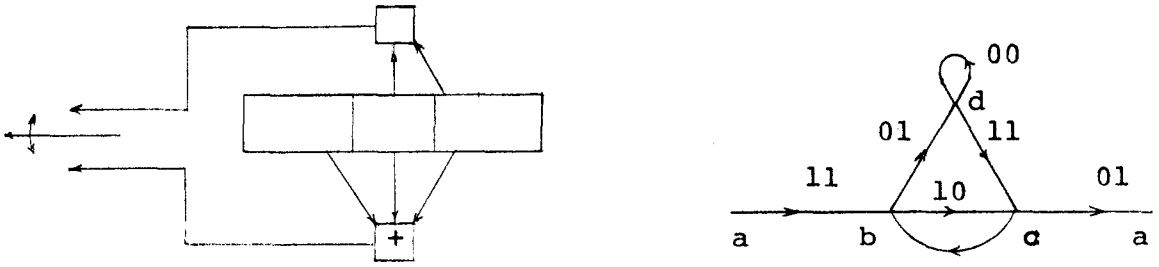


Figure 2.9 Coder Displaying Catastrophic Error Propagation

Assuming that the all zeros is the correct path, the incorrect path a b d d ... d c a has exactly 6 ones, no matter how many times we go around the self loop a. Thus, for a BSC, for example, four channel errors may cause us to choose this incorrect path or consequently make an arbitrarily large number of bit errors (equal to two plus the number of times the self loop is traversed).

We observe also that for binary-tree (R=1/n) codes, if each adder of the coder has an even number of connections, then the self loop corresponding to the all ones (data) state will have zero weight and consequently the code will be catastrophic.

The only advantage of a systematic code is that it can never be catastrophic, since each closed loop must contain at least one branch generated by a nonzero data bit and thus having a nonzero code symbol. Still, it can be shown that only a small fraction of nonsystematic codes is catastrophic (in fact, $1/(2^n-1)$ for binary-tree R=1/n codes). We note further that if catastrophic errors are ignored, nonsystematic codes with even smaller free distance than those of Table 2.1 exist.

2.1.6 Generalization of Viterbi Decoder to the Additive White Gaussian Noise Channel. Figure 2.10 exhibits a communication system employing a convolutional code. The convolutional encoder is precisely the device studied in the preceding sections.

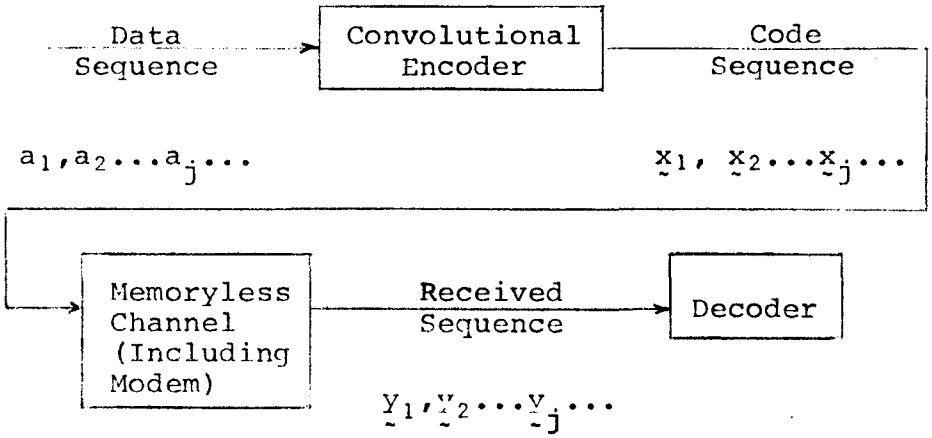


Fig. 2.10 Communication System Employing Convolutional Codes

The data sequence is generally binary (a_j=0 or 1) and the code sequence is divided into subsequences where $\underset{\sim}{x}_j$ represents the n code symbols generated just after the input bit a_j enters the coder, that is, the symbols of the jth branch. In terms of the example of Fig. 1.1, a_3=1 and $\underset{\sim}{x}_3$=01. The channel output or received sequence is similarly denoted. $\underset{\sim}{y}_j$ represents the n symbols received when the n code symbols of $\underset{\sim}{x}_j$ were transmitted. This model includes the BSC wherein the $\underset{\sim}{y}_j$ are binary n-vectors each of whose symbols differs from the corresponding symbol of $\underset{\sim}{x}_j$ with probability p and is identical to it with probability 1-p.

For completely general channels it is well known that if all input data sequences are equally likely, the decoder which minimizes the error probability is one which compares the conditional probabilities, also called likelihood functions, $P(\underset{\sim}{y}|\underset{\sim}{x}^m)$, where $\underset{\sim}{y}$ is the overall received sequence and $\underset{\sim}{x}^m$ is one of the possible transmitted sequences, and decides in favor of the maximum. This is called a <u>maximum likelihood decoder</u>. The likelihood functions are given or computed from the specifications of the channel. Generally it is more convenient to compare the quantities $\ln P(\underset{\sim}{y}|\underset{\sim}{x}^m)$, called the <u>log-likelihood functions</u>, and the result is unaltered since the logarithm is a monotonic function of its (always positive) argument. It is easily shown that for the BSC maximizing the log-likelihood function is equivalent to minimizing the Hamming distance, as we have done in previous sections.

We now consider the practical channel of primary interest:

namely, the additive white Gaussian noise (AWGN) channel with biphase PSK modulation. The modulator and optimum demodulator (correlator or integrate-and dump filter) for this channel are shown in Fig. 2.11.

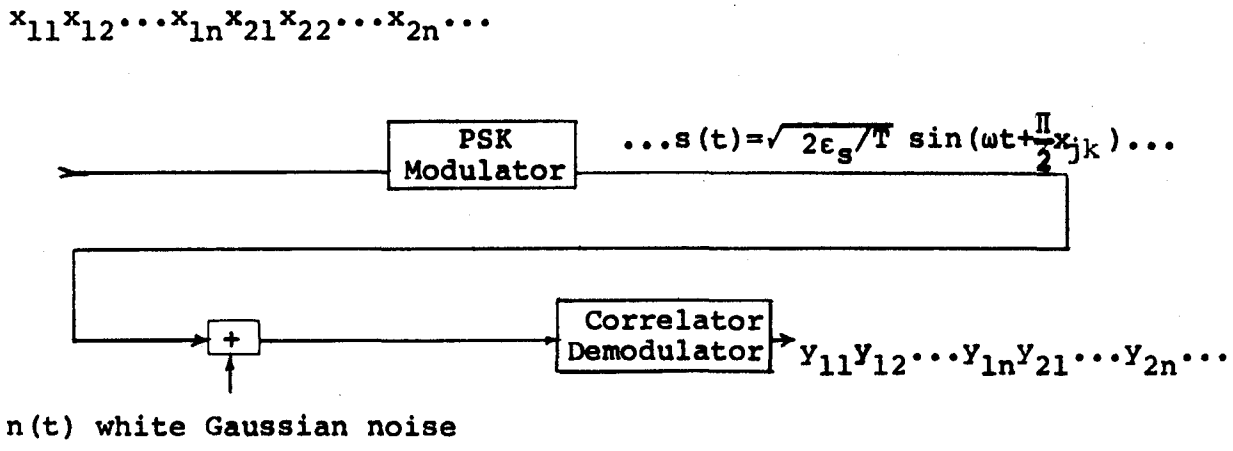


Fig. 2.11 Modem for Additive White Gaussian Noise PSK Modulated Memoryless Channel

We use the notation that x_{jk} is the kth code symbol for the jth branch. Each binary symbol (which we take here for convenience to be ±1) modulates the carrier by ± Π/2 radians for T seconds. The transmission rate is, therefore, 1/T symbols/second or b/nT= R/T bits/second. ε_s is the energy transmitted for each symbol. The energy per bit is, therefore, $\varepsilon_b = \varepsilon_s/R$. The white Gaussian noise is a zero mean random process of one-sided spectral density N_0 watts/Hz, which affects each symbol independently. It is

readily shown that the channel output symbol y_{jk} is a Gaussian random variable whose mean is $\sqrt{\varepsilon_s}\, x_{jk}$ (i.e., $+\sqrt{\varepsilon_s}$ if $x_{jk} = +1$ and $-\sqrt{\varepsilon_s}$ if $x_{jk} = -1$) and whose variance is $N_0/2$. Thus the conditional probability density (or likelihood) function of y_{jk} given x_{jk} is

$$p(y_{jk}|x_{jk}) = \frac{\exp\left[-(y_{jk} - \sqrt{\varepsilon_s}\, x_{jk})2/N_0\right]}{\sqrt{\pi N_0}}$$

The likelihood function for the jth branch of a particular code path is $\underset{\sim}{x}_j^{(m)}$

$$p\left(\underset{\sim}{y}_j | \underset{\sim}{x}_j^{(m)}\right) = \prod_{k=1}^{n} p\left(y_{jk} | x_{jk}^{(m)}\right)$$

since each symbol is affected independently by the white Gaussian noise and thus the log-likelihood function for the jth branch is

$$\ln p\left(\underset{\sim}{y}_j \mid \underset{\sim}{x}_j^{(m)}\right) = \sum_{k=1}^{n} \ln p\left(y_{jk} \mid x_{jk}^{(m)}\right)$$

$$= \frac{-1}{N_0} \sum_{k=1}^{n} y_{jk} - \sqrt{\varepsilon_s}\, x_{jk}^{(m)} \quad 2 - \tfrac{1}{2}\ell n \Pi / N_0$$

$$= \frac{2\sqrt{\varepsilon_s}}{N_0} \sum_{k=1}^{n} y_{jk} x_{jk}^{(m)} - \frac{\varepsilon_s}{N_0} \sum_{k=1}^{n} x_{jk}^{(m)} \quad 2$$

$$\frac{-1}{N_0} \sum_{k=1}^{n} y_{jk}^{2} - \frac{1}{2} \ell n \Pi / N_0$$

$$= C \sum_{k=1}^{n} y_{jk} x_{jk}^{(m)} - D$$

where C and D are independent of m, where we have used the fact that $x_{jk}^{(m)\ 2} = 1$. Similarly, the log-likelihood function for any path is the sum of the log-likelihood functions for each of its branches.

We have thus shown that the maximum likelihood decoder for the memoryless AWGN biphase modulated channel is one which forms the inner product between the received (real number) sequence and the code sequence (consisting of ± 1's) and chooses the path corresponding to the greatest.

Thus the metric for this channel is the inner product as contrasted with the distance* metric used for the BSC.

For convolutional codes the structure of the code paths was described in Sections 2.1.1 - 2.14. In Section 2.1.2 the optimum decoder was derived for the BSC. It now becomes clear that if we substitute the inner product metric $\sum y_{jk} x_{jk}^{(m)}$ for the distance metric $\sum d_{jk}^{(m)}$, used for the BSC, all the arguments used in Section 2.1.2 for the latter apply equally to this Gaussian Channel. In particular, the Viterbi decoder has a block diagram represented by the code state-diagram. At step j the stored metric for each state (which is the maximum of the metrics of all the paths leading to this state at this time) are augmented by the branch metrics for branches emanating from this state. The comparisons are performed among all pairs of (or in general sets of 2^b) branches entering each state and the maxima are selected as the new most likely paths. The history (input data) of each new survivor must again be stored and the decoder is now ready for step j+1.

2.1.7 Metric Quantization, Path Memory Truncation, and Other System Considerations. As we have just shown, the optimum metric for the biphase modulated AWGN channel is the inner product (or correlation) metric. However, since the y_{ij} are real numbers, a practical digital implementation requires quantization prior to

---

* Actually it is easily shown that maximizing an inner product is equivalent to minimizing the Euclidean distance between the corresponding vectors.

forming the metric.

In particular, if we quantize the y_{ij} to Q levels symmetric about zero, then the biphase AWGN channel is converted to a binary-input Q-output symmetric channel. Generally we choose $Q=2^q$ so that each received symbol can be represented by a "q" bit word. The optimum metric in this case is the log-likelihood function of this new binary-input Q-output channel. However, it has been found by simulation that nearly equivalent performance is obtained if the inner-product metric is used with y_{ij} replaced by $Q(y_{ij})$, where $Q(y_{ij})$ is an integer between 0 and Q-1 corresponding to the quantizer output level for an input y_{ij}. In fact, extensive simulation has shown that using this metric with 8-level quantization causes a performance degradation which is equivalent to a reduction of E_b/N_0 by less than 1/4 db for any given error probability level. On the other hand, quantization to 2 levels (which amounts to reducing the AWGN to a BSC) causes an effective reduction of E_b/N_0 by approximately 2 db.

Another major problem which arises in the implementation of a Viterbi decoder is the length of the path history which must be stored. In our previous discussion we ignored this important point and therefore implicitly assumed that all past data would be stored. A final decision can be made by forcing the coder into a known (all zeros) state, but this is totally impractical for long data sequences, for it requires storage of the entire trellis memory for each state. Suppose we truncate the path memories after L bits (branches) have been accumulated, by

comparing all 2^K metrics for a maximum and deciding on the bit corresponding to that path (out of 2^K) with the highest metric L branches forward. If L is several times as large as K, the additional bit errors introduced in this way are very few. It can be shown that the additional error probability due to path truncation, based on the largest path metric L branches beyond where the decision is to be made, is of the order of a block coding error for a code of block length L bits. Both theory and simulation then indicate that by making L four to five times as large as the code constraint length K, we can ensure that such additional errors have only a slight effect on the overall bit error probability.

Of course, basing the decision upon the maximum metric L branches forward may require a costly implementation to compare all 2^K state metrics. Other decision techniques, based on majority polling and metric overflow monitoring, are much less costly and appear to yield the same or better performance when L is increased slightly.

Cost and complexity of implementation of a Viterbi decoder depends strongly on constraint length, K, quantization, and speed. It depends much less strongly on path memory size, L, and the path truncation decision technique. In particular, the cost rises exponentially with K, but of course, the performance also improves with increasing K. Typically for a rate 1/2 code on an 8-level quantized AWGN, the required E_b/N_0 for $P_B \approx 10^{-5}$ is reduced by about 0.4 db per unit increase of K in the range

between 3 and 8. The cost of increasing L is only linear, but it is not justified on the basis of performance beyond L ≈ 5K. The cost of finer quantization depends strongly on the data speed requirements. The performance improvement from 2 level to 8 level quantization is nearly 2 db in E_b/N_0 but there is less than 0.25 db to be gained by using more than 8 levels.

For sufficiently low data speeds, all of the metric calculations and comparisons can be done serially, thus significantly reducing cost and complexity. At very high speeds, where digital gate speeds are only a few times faster than the received symbol rates, all metric computations and comparisons must be made in parallel. In intermediate speed regions, serial-parallel combinations may be possible.

Detailed consideration of Viterbi decoder implementation and system designs will be treated in Section 2.4.

2.2 Rate 1/2 Convolutional Codes and Viterbi Decoders. In this section, the performance of rate 1/2 Viterbi decoders is examined in detail. Bit error rate vs. E_b/N_0 obtained both by simulations and analysis are presented for optimum codes of constraint lengths 3 through 8. Particular attention is paid to the sensitivity of performance to the decoder parameters which influence complexity and cost. Also of interest is the ability of a Viterbi decoder to withstand demodulator imperfections, and its usefulness in communicating system quality information.

Computer simulation of Viterbi decoders is a useful technique for evaluating performance down to a bit error rate of about 10^{-4}

to 10^{-5}, depending on code constraint length. Simulations at lower error rates require prohibitively long computer runs to obtain meaningful data. Fortunately, an upper bound on both event and bit error rates has been derived which is very tight for error rates of about 10^{-5} and lower. A combination of the simulations and the numerically evaluated upper bound presented here provides a complete picture of Viterbi decoder performance over a wide range of error rates.

2.2.1 Good Convolutional Codes. One obvious criterion for selecting codes is bit error probability. Unfortunately, obtaining bit error probability through simulation is too time consuming to be used as a method of sifting through a large number of convolutional codes. A much more useful measure of a code is its minimum free distance. As defined previously, the free distance between two code words is the Hamming distance between them from the state in the trellis at which they diverge (the point at which the information bits begin to differ), to the state where they remerge (after K-1 identical information bits). A set of large free distances between the correct code path and the competing incorrect paths is desirable with Viterbi decoding. This is because the greater the free distance, the more channel errors must occur in order for an incorrect path to look more likely than the correct path.

The minimum free distance, d_f, is the smallest value of free distance between the correct path and any other path. Since

the codes under consideration are linear codes, the set of distances from any codeword to all other codewords is the same as the set of distances from the all zeros codeword to all other codewords. Thus, d_f is the minimum of the weight of all codewords from the point at which they diverge from, until the point at which they remerge to, the all zeros path. Often, but not always, the minimum weight path corresponds to an information sequence with a single 1 in it. The codeword associated with this sequence diverges from the all zeros path where the information 1 occurs, and remerges K-1 branches later. This, of course, is the shortest length over which two distinct paths can be diverged.

Using the algebraic properties of linear group codes, an upper bound on the minimum free distance of a convolutional code, as a function of constraint length, has been found (Ref. 1,2). For rate 1/n nonsystematic codes, the bound is

$$d_f \leq \min_h \frac{2^{h-1}}{2^h-1} \quad [\ (K+h-1)n\ ]$$

This bound provides a target value of d_f which can be used when searching for good codes. If a code is found with a d_f which satisfies the bound with equality, it is immediately known that no code exists with a larger minimum distance. Of course, maximizing minimum free distance does not necessarily minimize decoder error probability. The number of codewords having the minimum distance, as well as the distribution of codewords at distances

somewhat greater than d_f, are also important. After preselecting codes based on minimum free distance, these other factors are useful in final code selection. Simulations and numerical code evaluation indicates that choosing codes with maximum minimum free distance, taking into account the number of paths at this distance, and if necessary, slightly larger distances yields codes with minimum error probabilities with Viterbi decoding.

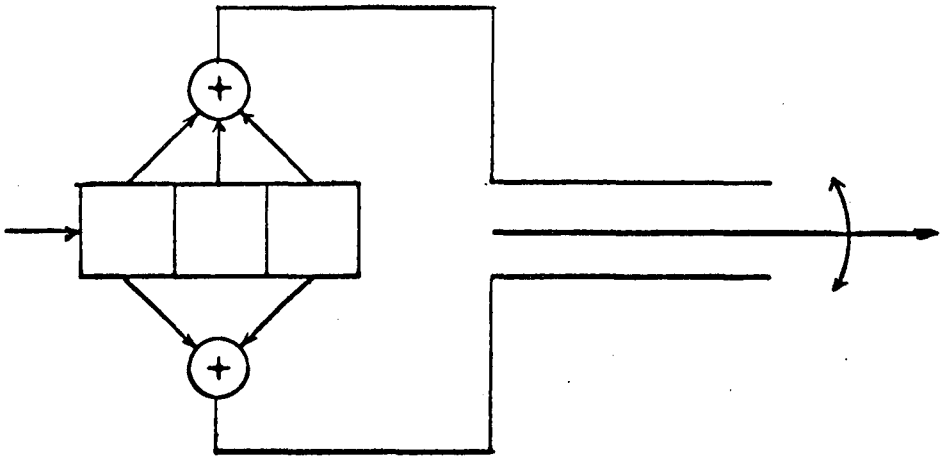
The optimum rate 1/2 codes for K=3 through 8 were found by Odenwalder (Ref. 3). They are tabulated in Table 2.2.1. For each constraint length the table shows the optimum code generators, the actual d_f for the code, the number of errors n_e in all of the codewords at the minimum distance, and the upper bound value on minimum free distance d_f*.

2.2.2 Numerical Code Performance Bound. One of the two principal tools used in evaluating the performance of convolutional codes in this study has been an upper bound on error probability related to the convolutional code transfer function. The bound is extremely tight for high E_b/N_0 (low decoder error rates) where computer simulation is impractical, due to the prohibitively long times required to collect significant data.

It has been shown (Refs 5,6) that a union bound on the performance of a convolutional code on memoryless channels can be obtained from the directed-graph state diagram of the coder. For example, the optimum constraint length K=3, rate 1/2 coder is shown in Figure 2.2.1. The states correspond to the con-

K	Code Generators	d_f	n_e	d_f*
3	111 101	5	1	5
4	1111 1101	6	2	6
5	11101 10011	7	4	8
6	111011 110001	8	6	9
7	1111001 1011011	10	36	10
8	11111001 10100111	10	2	10

Table 2.2.1 Optimum Rate 1/2 Codes. d_f is the code minimum free distance, n_e is the number of bit errors in paths at distance d_f, d_f* is the upper bound on minimum free distance.



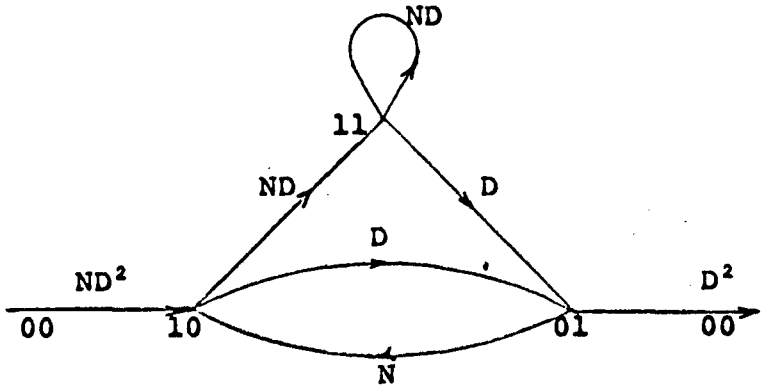


Fig. 2.2.1 Code and State Diagrams for K=3 Code

tents of all but the first stage of the coder register, when a new information bit has just entered the first stage. The exponent of D(0, 1, or 2) is the weight of the (two symbol) vector output at this time, and the exponent of N(0 or 1) indicates whether a 0 or 1 information bit has just entered the coder.

Regarding the all zeros node as both the input and output of the graph, the transfer function of any path through the tree is defined as the product of the branch transfer functions along that path. For example, the transfer function of the path corresponding to the information sequence 10100 is

$$T_{10100} = (ND^2)(D)(N)(D)(D^2) = N^2D^6 \qquad (2.2.1)$$

The transfer function of the graph is the sum of the transfer function of all paths starting and ending in the all zeros state. The general form of this transfer function is

$$T(N,D) = D^{d_f} f_1(N) + D^{d_f+1} f_2(N) + \ldots + D^{d_f+i} f_{i+1}(N) + \cdots \qquad (2.2.2)$$

Here d_f is the minimum free distance of the code. Notice that the exponent in the path transfer function (Eq. 2.2.1) is the weight of the code symbols on the particular path through the graph of Fig. 2.2.1. Therefore, with N=1, the terms in the transfer function T(1,D) are of the form

$$D^{d_f+i} f_{i+1}(1)$$

where $f_{i+1}(1)$ is just the number of paths at distance d_f+i.

For the unquantized, additive white Gaussian noise channel with PSK modulation, the error probability between the all zeros (correct) path, and another path which diverges from and returns to the all zeros path, is bounded by (Ref. 4)

$$P_2 < e^{-dE_s/N_0}$$

where d is the weight of the competing path. E_s is the code symbol energy and N_0 is the noise spectral density. For example, if the competing path corresponded to the information sequence 10100, the bound is obtained from Eq. (2.2.1)

$$P_2 < T_{10100} \Big|_{N=1,\ D=\exp(-E_s/N_0)} = e^{-6E_s/N_0}$$

Likewise, a union bound on first event error probability due to all paths competing with the all zeros path (all paths through the graph in Fig. 2.2.1) is

$$P_E < T(N,D) \Big|_{N-1,\ D=\exp(-E_s/N_0)} \qquad (2.2.3)$$

In order to get a bound on bit error probability, we note that the exponent of N in a path transfer function is the number

of information 1's (errors) on that path. A union bound on bit error probability would be obtained if the path transfer function were weighted by the number of bit errors on the path. One simple way of doing this is to take the derivative of T(N,D) with respect to N. This brings down the exponents of N -- the number of bit errors on a path -- into the coefficients. The bound on bit error probability is therefore

$$P_B < \left.\frac{dT(N,D)}{dN}\right|_{N=1,\ D=\exp(-E_s/N_0)} \qquad (2.2.4)$$

For the Gaussian channel these bounds can be tightened somewhat (Ref. 5):

$$P_E < \operatorname{erfc}\left(\sqrt{d_f E_s/N_0}\right)\ D^{-d_f} T(N,D)\Big|_{N=1} \qquad (2.2.5)$$

$$P_B < \operatorname{erfc}\left(\sqrt{d_f E_s/N_0}\right)\ D^{-d_f}\ \left.\frac{dT(N,D)}{dN}\right|_{N=1} \qquad (2.2.6)$$

with $D = \exp(-E_s/N_0)$ in both cases.

The difficulty of this approach is that the number of states grows exponentially with K and consequently the tedium involved in direct computation is effectively insurmountable for K>4.

On the other hand, the calculation of the transfer function is equivalent to a matrix inversion. Taking into account the

particular properties of a convolutional code transfer matrix, the transfer function can be evaluated numerically using an iterative technique. This technique is explained in detail in Appendix A. A computer program has been written to evaluate the transfer function bound as a function of K, code rate, and E_b/N_0. For rate 1/2 codes the performance bounds are presented in Section 2.2.4, for other rates, they are contained in Section 2.3.

Decoder performance predicted by the bounds at around 10^{-5} bit error rate is quite close to simulation results, allowing for finite receiver quantization in the simulations.

2.2.3 Viterbi Decoder Simulation Program. A program has been written to simulate the operation of a Viterbi decoder on a quantized Gaussian channel. The program is quite flexible, in that all of the parameters of interest in Viterbi decoding can be varied by changing program inputs. For rate 1/2 codes, the constraint length can be varied from 3 to 9. Simulated received data for PSK modulation on an additive white Gaussian noise channel can be generated for any value of E_b/N_0 with output quantization of 2, 4 and 8 levels. For each run the program provides:

a) Bit error rate for a variety of decoder path lengths, with output selection based on the most likely state.

b) Event error rate.

c) Average length in bits of an error event, from the first bit error to the last comprising the event.

In addition to the statistics in a), b) and c), which are

based on maximum likelihood state output selection, the following measured error rates are provided:

d) Bit error rate with majority output selection. Here the output on a majority of the decoder paths is chosen as the decoder output.

e) Bit error rate resulting from selecting an output from some state path whose metric value is better than some threshold value.

All of the gathered statistics except those in a) are for length 32 decoder state paths. The output decision technique simulated in d) and especially e), although they are slightly sub-optimal, are much simpler to implement in parallel processing Viterbi decoders than maximum likelihood selection.

Lastly, with an eye toward the 180° phase ambiguity problem with PSK modulation, the simulation program measures

f) Bit error rate when differential data encoding-decoding is used with codes transparent to 180° phase flips. This technique, along with the simulation results, is treated in Section 2.2.5.

2.2.4 Simulation and Numerical Performance Data

2.2.4.1 General Performance Results. The principal results of the simulations and code transfer function bounds are shown in Figs. 2.2.4.1, 2.2.4.2, and 2.2.4.3. All of these figures show bit error rate vs. E_b/N_0 for Viterbi decoders using the optimum

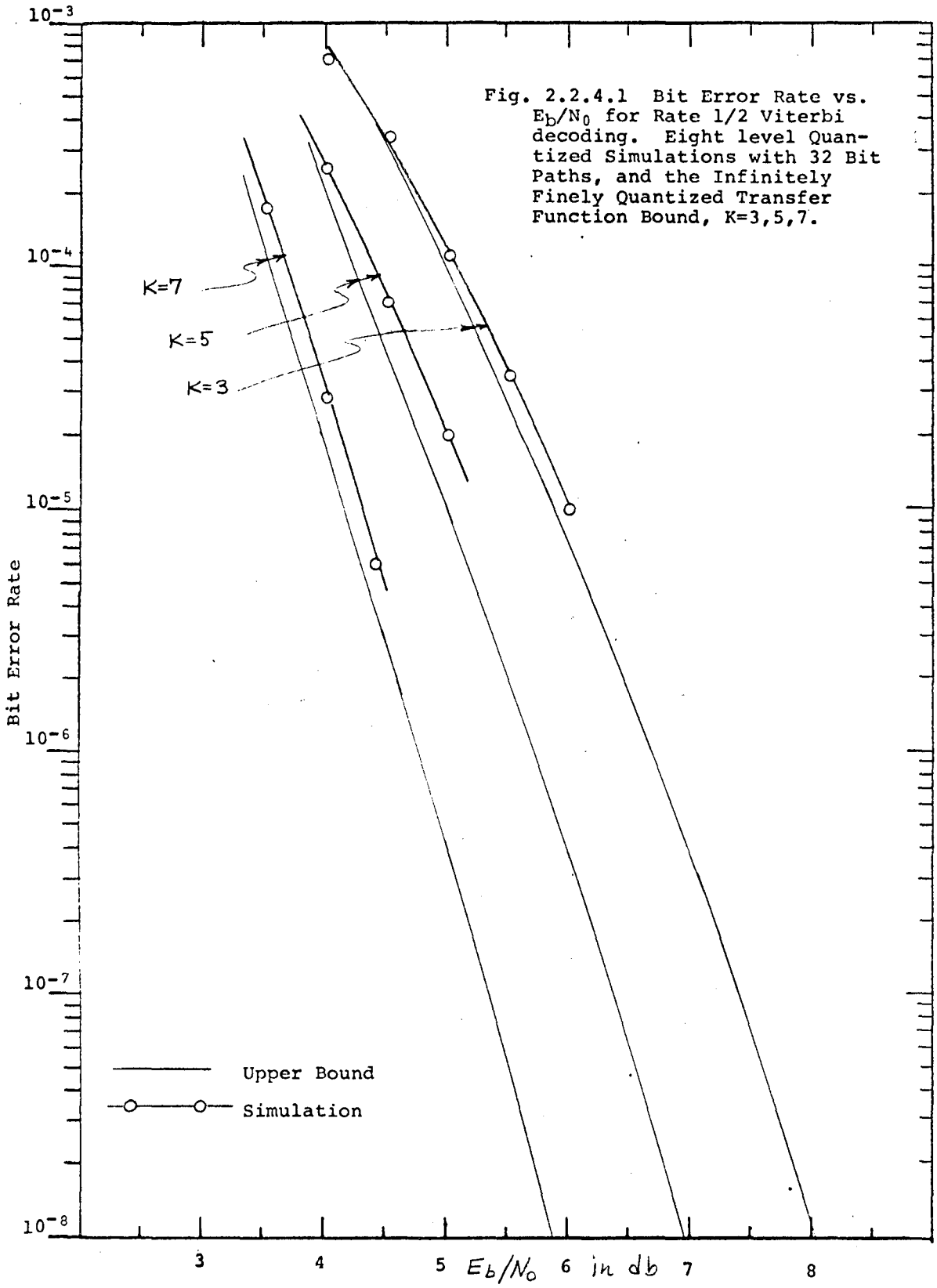


Fig. 2.2.4.1 Bit Error Rate vs. E_b/N_0 for Rate 1/2 Viterbi decoding. Eight level Quantized Simulations with 32 Bit Paths, and the Infinitely Finely Quantized Transfer Function Bound, K=3,5,7.

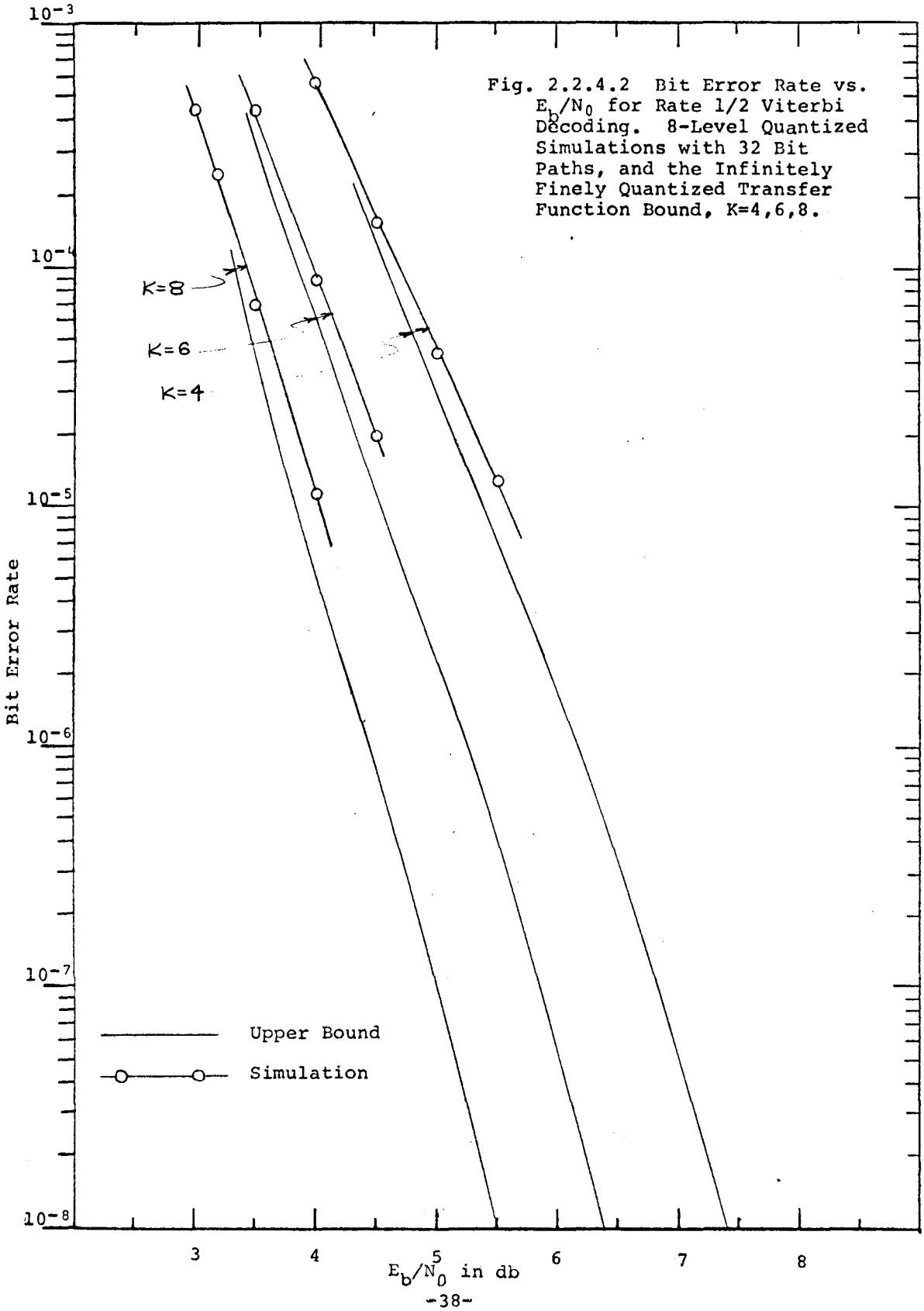


Fig. 2.2.4.2 Bit Error Rate vs. E_b/N_0 for Rate 1/2 Viterbi Decoding. 8-Level Quantized Simulations with 32 Bit Paths, and the Infinitely Finely Quantized Transfer Function Bound, K=4,6,8.

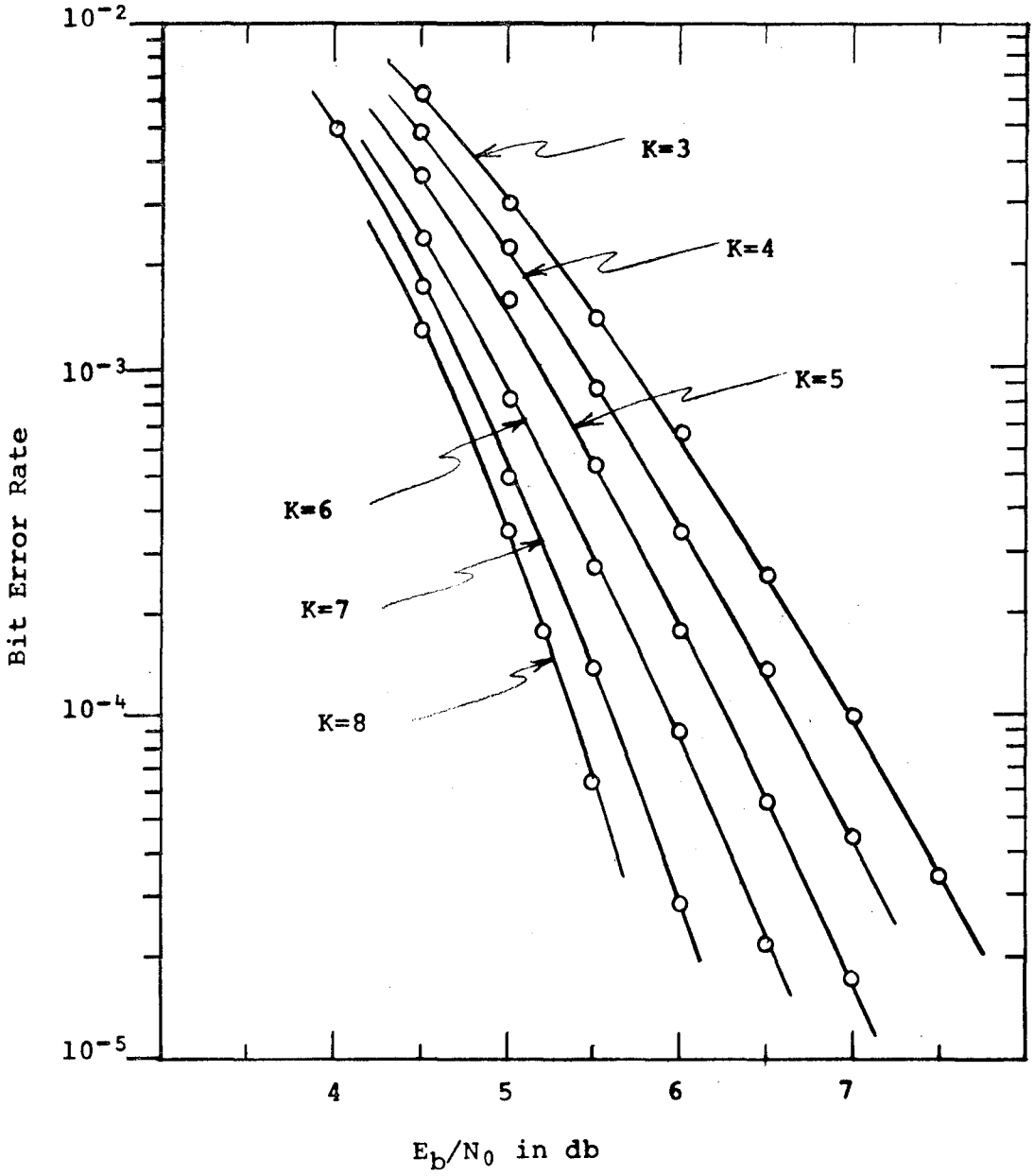


Fig. 2.2.4.3 Bit Error Rate vs. E_b/N_0 for Rate 1/2 Viterbi Decoding. Hard Quantized Received Data with 32-Bit Paths K=3 through 8.

rate 1/2 convolutional codes of Table 2.2.1. In all cases, the decoder state path length was 32 bits. In all simulation runs, at least 25 error events contributed to the compiled statistics.

The simulation results in Figs. 2.2.4.1 and 2.2.4.2 are for soft (8-level) receiver quantization. Equally spaced demodulation thresholds are used at $\pm 1.5\sigma$, $\pm\sigma$, ± 0.5 , and 0 where $\sigma^2=N_0/2$ is the noise variance. This choice of 8-level quantizer thresholds is within a broad range of near optimum values, as will be shown presently. The transfer function bound is for infinitely finely quantized received data. Allowing for the 0.20 to 0.25 db loss usually associated with 8-level receiver quantization compared with infinite quantization, the transfer function bound curves are in excellent agreement with simulation results in the 10^{-4} to 10^{-5} bit error rate range.

Since the accuracy of the transfer function bound increases with E_b/N_0, decoder performance can be ascertained accurately in the 10^{-5} to 10^{-8} region even in the absence of simulations.

Ideally, the symbol metrics associated with each of the 8 quantization levels would be proportional to the log-likelihood of receiving the given level, given the hypothesis of a "0" or a "1" transmitted. In the interest of keeping the number of bits required to represent metrics to a minimum, it was shown (Ref. 2) that equally spaced symbol metrics, for instance, the numbers 0-7, could be used with negligible performance degradation. We have taken the compression of metric representation one

step further. As is shown in section 2.4, an additional bit in the state metric can be saved if levels symmetrically located about the zero threshold have symbol metrics which are the negatives of one another. Thus, for the simulations presented in Figs. 2.2.4.1 and 2.2.4.2, the eight symbol metrics used were 4, 3, 2, 1, -1, -2, -3, -4. These symbol metrics clearly do not change in equal increments; however, simulations have shown that system performance does not suffer significantly.

Fig. 2.2.4.3 gives the simulation results for Viterbi decoding with hard receiver quantization. The same optimum rate 1/2, K=3 through 8 codes were used here as in the 8-level quantized simulations.

Several points are obvious from the performance curves

a) 2-level quantization is everywhere close to 2 db inferior to 8-level quantization. This seems to reinforce the folk theorem that hard quantization always leads to a 2 db loss in system efficiency.

b) Each increment in K provides an improvement in efficiency of something less than .5 db at a bit error rate of 10^{-5}.

c) Performance improvement vs. K increases with decreasing bit error rate.

2.2.4.2 Receiver Quantization. In order to observe the effects of varying receiver quantization more closely, simulation

performance data is presented in Fig. 2.2.4.4 for the K=5, rate 1/2 code, with 2, 4, and 8-level receiver quantization. The 8-level thresholds and metrics are identical to those of Fig. 2.2.4.1. In fact, the 2 and 8 quantization level curves are taken from Figs. 2.2.4.3 and 2.2.4.1 respectively. The 4-level thresholds were set at 0 and $\pm \sigma$. The metrics were chosen to be 2, 1, -1, -2, for the same reasons which suggested the 8-level metrics.

2.2.4.3 Path Memory. The Viterbi decoder is a maximum likelihood decoder only when its decision path memories are infinitely long. That is, decoding delay is infinite. For practical purposes, it is desirable to use path memories as short as possible. There is a path memory for each state in a Viterbi decoder. Providing storage and managing decision paths is a significant part of any Viterbi decoder. It is therefore worthwhile to study the performance degradation vs. path length for Viterbi decoding.

Fig. 2.2.4.5 shows bit error rate performance vs. E_b/N_0 for three path lengths (8, 16 and 32) using the rate 1/2, K=5 code, for both 2 and 8-level received data quantization. The length 32 path curve is identical to the K=5 curve in Fig. 2.2.4.1. Performance with length 32 paths is essentially identical to that of an infinite path decoder. Even for a path length of only 16, there is only a small degradation in performance. Other simulations have shown that a path length of 4 to 5 constraint lengths

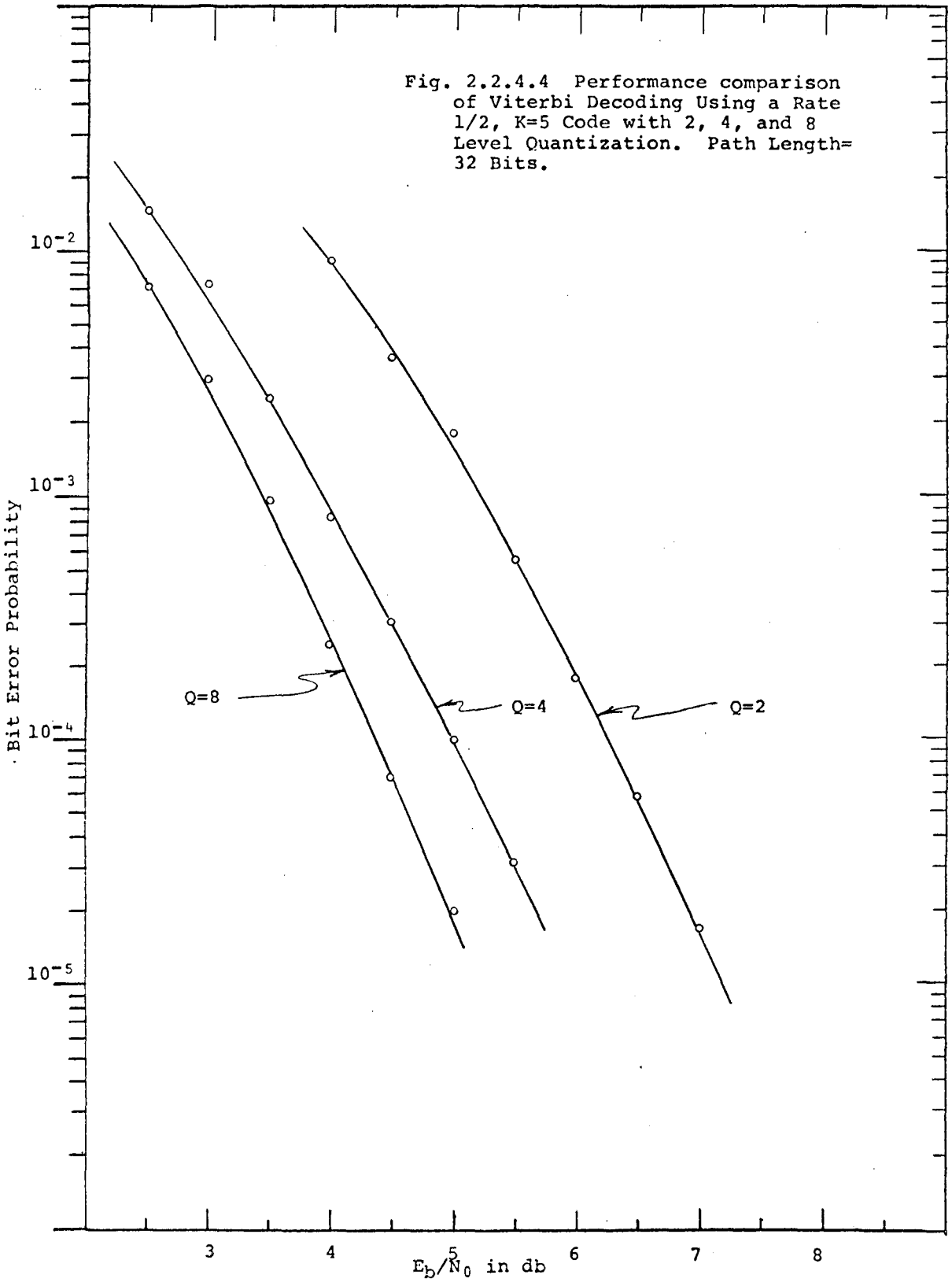


Fig. 2.2.4.4 Performance comparison of Viterbi Decoding Using a Rate 1/2, K=5 Code with 2, 4, and 8 Level Quantization. Path Length= 32 Bits.

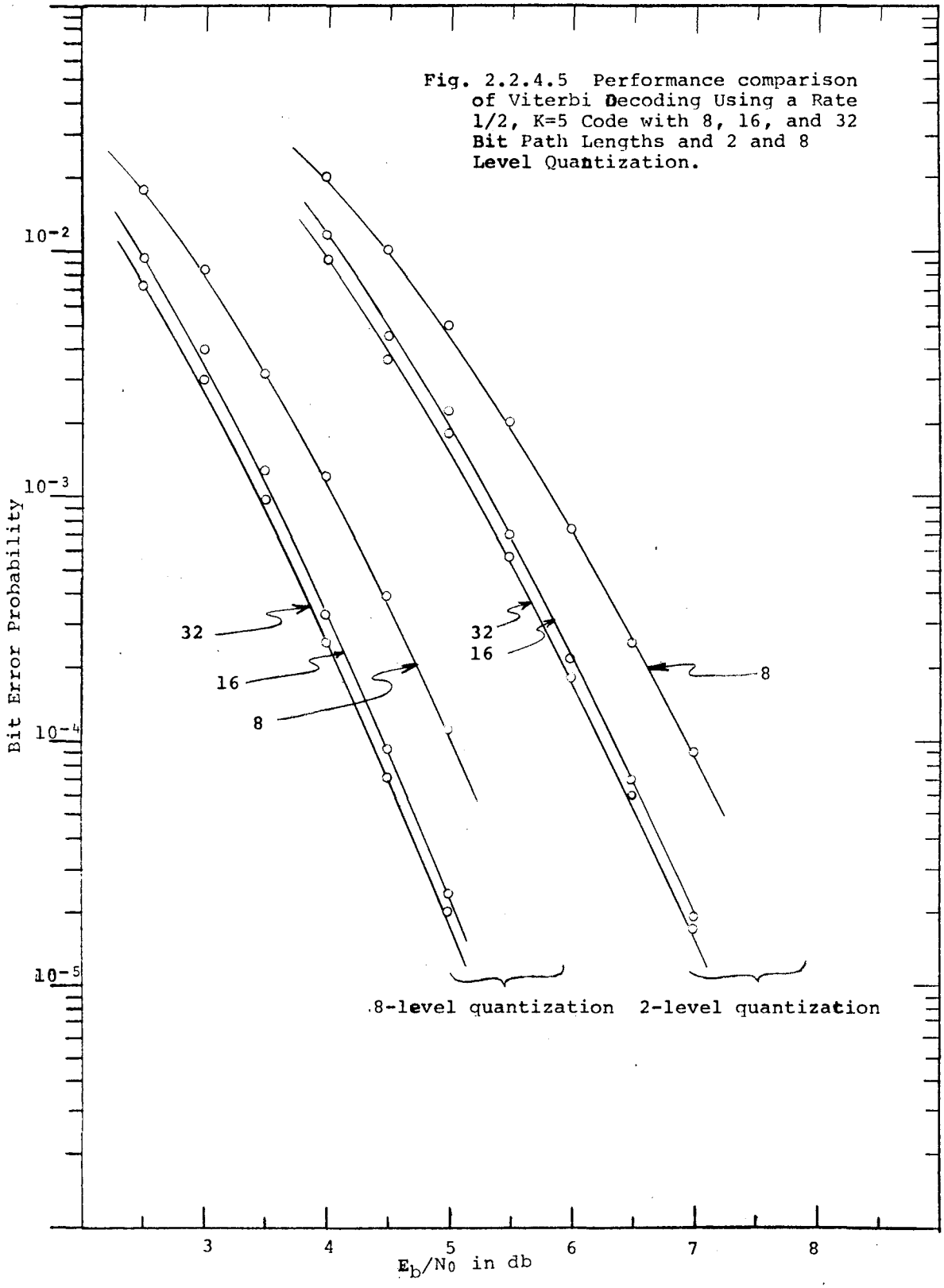


Fig. 2.2.4.5 Performance comparison of Viterbi Decoding Using a Rate 1/2, K=5 Code with 8, 16, and 32 Bit Path Lengths and 2 and 8 Level Quantization.

is sufficient for other constraint lengths as well.

2.2.4.4 Decoder Output Selection. In a Viterbi decoder with finite path memories, it is possible that not all state paths are merged at the point at which a decoded bit must be output. Physically this means that the oldest bits in each of the state path memories may not always agree. The decoder must output a bit however and there must be a means for selecting which of the 2^{K-1} oldest path bits to output.

The optimum method for selecting output bits is to choose the bit corresponding to the path with the best metric. This selection rule is very complex to mechanize in a high speed decoder, where the pairwise state comparisons are done in parallel. This fact has lead to a study of simpler output selection schemes, the aim being to find one which does not degrade performance appreciably. One very simple scheme is to choose a path at random from which to output decoded bits (or always output bits from the same path). This scheme,however, has been found to significantly degrade performance. In fact, if a path memory of n bits is required for a given performance goal with maximum likelihood output selection, then simulation has shown that a memory of up to 2n bits is required for the same performance with arbitrary output selection.

Another method is to output a "0" if a majority of the 2^{K-1} paths have a "0" as their oldest bit; otherwise, output a "1". This scheme is somewhat simpler to implement than maximum likelihood selection.

An efficient yet simple to implement scheme, which we have devised, is to select the output from some state path whose mettic is better than a certain threshold. This scheme is described in Section 2.4. It is called "less than four" selection. Fig. 2.2.4.6 compares the performance of a) maximum likelihood selection, b) majority selection, and c) "less than four" selection. The comparison is made for a K=7, rate 1/2 code with 8-level quantization, and path length 32. It is interesting to note that the performance of the K=5 decoder was the same for all three output selection schemes, with a 32 bit path memory. As the path memory gets long relative to K, there is a larger probability that all state paths will be merged by the time a bit must be output. Thus the output selection mechanism has less of an effect on performance.

2.2.5 Code Synchronization and Channel Reliability.

2.2.5.1 Node Synchronization and Phase Ambiguity Resolution. Because of the inherent continuity involved in convolutional coding, code synchronization at the receiver is usually much simpler than in the case of block codes. For convolutional decoding techniques involving a fixed number of computations per bit decoded, such as Viterbi decoding, the decoder initially makes an arbitrary guess of the encoder state to start decoding. If the guess is incorrect, the decoder will output several bits or, at most, tens of bits of unreliable data before assuming steady state reliable operation. Thus, the block synchronization problem does not really exist. There remains the problem of node

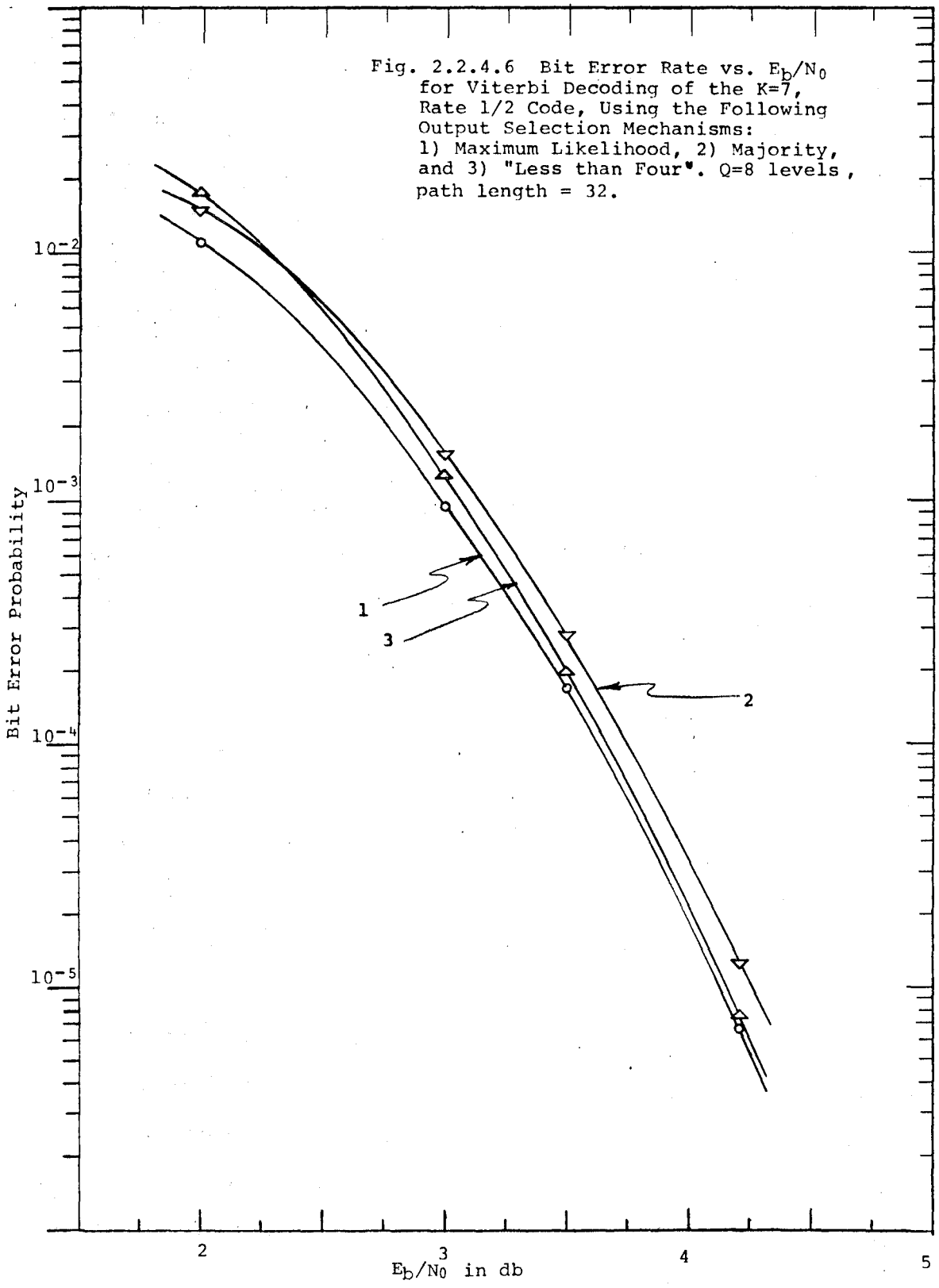


Fig. 2.2.4.6 Bit Error Rate vs. E_b/N_0 for Viterbi Decoding of the K=7, Rate 1/2 Code, Using the Following Output Selection Mechanisms: 1) Maximum Likelihood, 2) Majority, and 3) "Less than Four". Q=8 levels, path length = 32.

synchronization and, depending upon the modulation-demodulation technique used, the problem of 2 or 4 phase ambiguity resolution. For a rate 1/n code, there are n code symbols on each branch in the code tree. Node synchronization is obtained when the decoder has knowledge of which sets of n symbols in the received symbol stream belong to the same branch. In a purely serial received stream, this is a 1 in n ambiguity.

In addition, modems using biphase or quadriphase PSK with suppressed carriers derive a phase reference for coherent demodulation from a squaring or fourth power phase lock loop or its equivalent. This introduces ambiguities in that the squaring loop is stable in the in-phase and 180° out of phase positions, and the 4th power loop is, in addition, stable at ±90° from the in-phase position.

We have directed our efforts toward using the error detection capability of convolutional decoders, or the ability of the decoder to detect unsatisfactory system of operation, to detect and correct for incorrect node synchronization and the occasional phase flips in the phase tracking loop. It is now apparent that simple and effective techniques for maintaining node and phase synchronization completely within the decoder itself are feasible.

For the purpose of ease of illustration, the rate 1/2, hard-quantized receiver output case will be considered here. Techniques generalize easily to soft quantization and with somewhat more complexity to other rates. A Viterbi decoder operating on hard quantized received data will use Hamming distance for state metrics.

Relatively large metric values indicate a poorer match to received data than lower metric values. The smallest state metric at any time corresponds to the path with the best match to received data, and the metric itself is equal to the number of discrepencies between the received data and that path. Clearly, when the decoder is in correct node synchronization and the demodulator loop is locked properly, the path with the smallest Hamming distance will usually correspond to the correct path. The rate of increase of this path metric will depend on the channel error rate. For instance, if the crossover probability is p=.02 then, on the average, there will be an increase of 1 in the correct path metric for every 50 channel symbols. On the other hand, we intuitively expect that if node synchronization is lost, or if the phase lock loop locks onto an incorrect phase position,the match between the received data at the nearest codeword should be much poorer than 1 mismatch in 50 symbols. If, in fact, the best path metric increases more rapidly when off node or phase synchronization, this can be used to detect these maladies.

One simple technique would use an up-down counter to detect unreliable system operation. The counter counts <u>up</u> by k units (k>1) each time the best path metric increases by 1, while it counts <u>down</u> by 1 for each bit time. The parameter k is chosen so that the average drift of the counter is downward when in proper node and phase synchronization and upward when out of either phase or node synchronization or both. The first condition requires that kp<1 where p is the channel crossover

probability; while, the second condition requires $kp' > 1$ where p' is the as yet undetermined rate of increase of the best path metric with improper node or phase synchronization. The count would not be allowed to fall below zero. When the count exceeds a threshold value N, the assumed node synchronization or phase synchronization position is changed in the decoder according to a preset strategy.

Potentially effective methods of changing phase and node synchronization depend upon whether the system uses BPSK or QPSK. For BPSK, both the problem of node synchronization and 180° phase ambiguity exist. The 180° phase ambiguity can be circumvented by using differential encoding of the information prior to convolutional encoding and differential decoding after the channel decoder. A transparent convolutional code is required, i.e., the all 1's data sequence maps into the all 1's code sequence. This technique is discussed in section 2.2.5.2.

Another method of homing in on both node and phase synchronization is shown in Fig. 2.2.5.1.

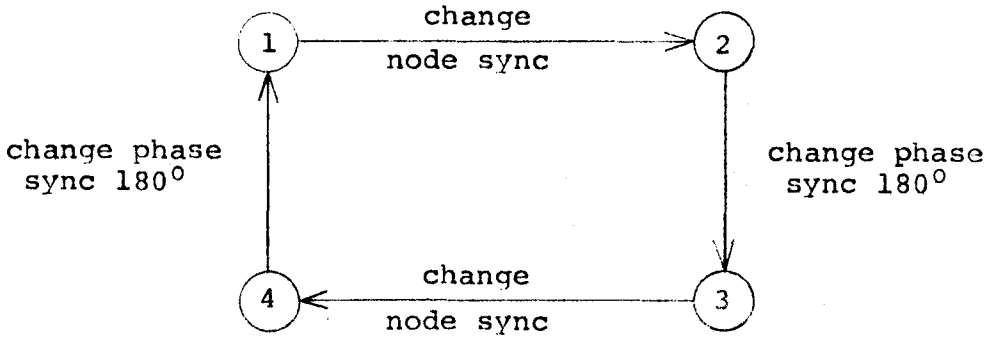


Figure 2.2.5.1 Biphase node-phase synchronization strategy.

State 1 represents an arbitrary initial position for phase and node synchronization. If the system reliability counter counts past N, a node synchronization change is attempted, bringing the system to state 2. If the count again passes the threshold, received symbols are complemented prior to decoding (equivalent to a 180° phase shift) bringing the system to state 3. Subsequent counter overflows cause the system to go to state 4 and then back to 1. There are only 4 possible states because there are 2 stable phase positions and, for rate 1/2, 2 node synchronization positions.

For a quadriphase modem and rate 1/2, there is no node synchronization problem since both parity bits on a branch are transmitted on one baud. However, now there are 4 stable phase positions; thus, there are 4 stable system states to contend with just as in the biphase case. These states, as well as the phase synchronization strategy are shown in Figure 2.2.5.2.

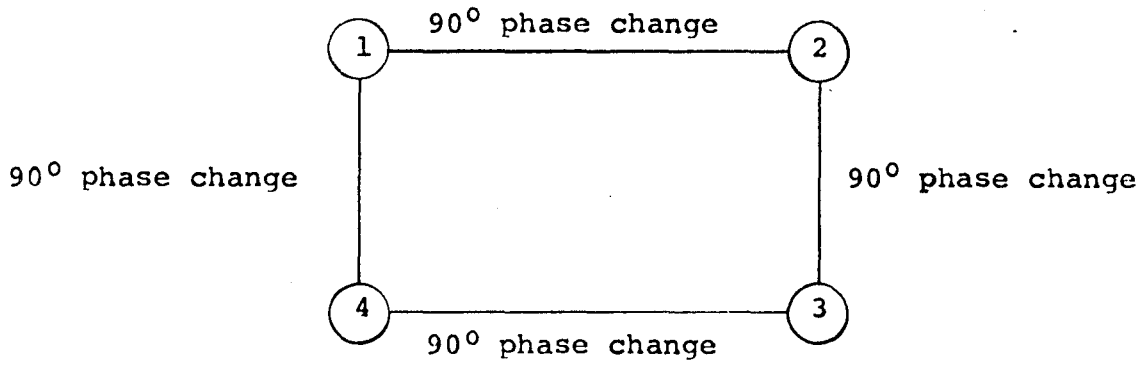


Figure 2.2.5.2 Quadriphase phase synchronization strategy.

The problem still remains as to whether the best path metric will increase rapidly enough, when phase or node synchronization is incorrect, to reliably detect these events while keeping the false alarm rate sufficiently low. Understanding of the operation of the decoder when synchronization is lost is aided by observing the effect of synchronization loss on a hypothetical code syndrome calculated at the receiver.

These effects can best be seen by working with the polynomial representations of the information, parity, and code generator sequences. The polynomial coefficients are the terms in the sequence of interest. For instance, the information sequence 101101... is represented by the polynomial

$$i(D) = 1 + D^2 + D^3 + D^5 + \ldots$$

The parity stream generated by passing $i(D)$ through a convolutional encoder with generator $g(D)$ is simply

$$p(D) = i(D)g(D)$$

A rate 1/2 code has two generators $g_1(D)$ and $g_2(D)$ which generate two streams $p_1(D)$ and $p_2(D)$. A representation of the encoder, channel and syndrome calculator is shown in Figure 2.2.5.3. The $n_i(D)$ are channel noise polynomials, a coefficient of 1 represents a channel error. All additions are mod-2.

Notice that in the absence of noise, i.e., $n_1(D) = n_2(D) = 0$,

$$\begin{aligned} s(D) &= i(D)g_1(D)g_2(D) + i(D)g_2(D)g_1(D) \\ &= 0 \end{aligned}$$

independent of $i(D)$; thus, the syndrome $s(D)$ depends only upon the noise polynomials and hence is in fact a true code syndrome.

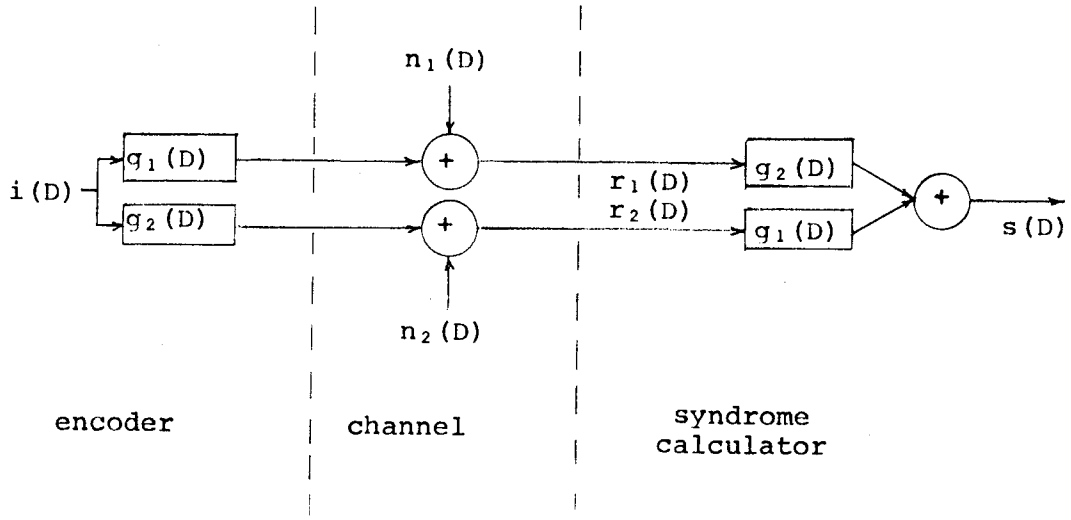


Figure 2.2.5.3 Representation of encoder, channel and syndrome calculator.

A hypothetical decoder operating on a segment of the syndrome polynomial should attempt to find the minimum weight channel error polynomials which could have caused the particular syndrome pattern. Decoding would then reduce to changing those bits corresponding to ones in the minimum weight error pattern. We shall now look at the functional form of the syndrome in the presence of various combinations of improper phase and node synchronization.

Considering the biphase modulation case first, suppose phase synchronization is correct and node synchronization is off. This is equivalent to the received lines being crossed in Fig. 2.2.5.3. Thus:

$$r_1(D) = i(D)g_2(D)$$
$$r_2(D) = i(D)g_1(D)$$
$$s(D) = i(D)\left[g_1^2(D) + g_2^2(D)\right],$$

in the absence of noise. Unlike the in-synchronization case, the syndrome now depends on the information sequence. In fact, the syndrome is the information sequence convolved with a new generator $g_1^2(D) + g_2^2(D)$. If the coefficients of $i(D)$ are EL (that is independant and equally likely to be 0 or 1) then the coefficients of $s(D)$ also have this property (Ref. 4).

Now suppose node synchronization is correct and the phase flipped 180°. Here

$$r_1(D) = \overline{i(D)g_1(D)}$$
$$r_2(D) = \overline{i(D)g_2(D)}$$

$$s(D) = \overline{i(D)g_1(D)}g_2(D) + i(D)g_2(D)g_1(D)$$

in the absence of noise. If the code were transparent to 180° phase flips, both $g_1(D)$ and $g_2(D)$ would have to be odd, that is, have an odd number of non-zero terms (coder taps). Since we do not want the transparent feature in this analysis, we will assume $g_1(D)$ is odd and $g_2(D)$ is even (if they were both even the code would be catastrophic). Now a sequence convolved with an even generator is the same as the result of convolving the sequence with the same generator; whereas, a sequence convolved with an odd generator is the <u>complement</u> of the result of convolving the complemented sequence with that generator. Thus,

$$s(D) = i(D)g_1(D)g_2(D) + \overline{i(D)g_2(D)g_1(D)}$$
$$= 1 + D + D^2 + D^3 + \ldots$$

Hence without noise, a 180° phase flip causes an all 1's syndrome.

When both improper node and phase synchronization occur, it is easily shown that without noise

$$s(D) = \overline{i(D)}\left[g_1^2(D) + g_2^2(D)\right]$$

Again this syndrome is EL if i(D) is EL.

In the quadriphase situation, the tracking loop can be ±90° or 180° out of proper phase alignment. When a 90° shift occurs, the in-phase and quadrature channel outputs are switched with one becoming complemented. For instance with a 90° shift,

$$r_1(D) = i(D)g_2(D)$$

$$r_2(D) = \overline{i(D)g_1(D)}$$

Again assuming $g_1(D)$ is odd and $g_2(D)$ is even

$$s(D) = i(D)\ [g_1^2(D) + g_2^2(D)]$$

This is the same syndrome as obtained with improper node synchronization. A -90° phase shift yields the syndrome

$$s(D) = \overline{i(D)}\ [g_1^2(D) + g_2^2(D)]$$

This is the same syndrome as that derived for the biphase improper node synchronization, and 180° phase shift case. Finally the 180° phase shift case is identical to that for the corresponding biphase situation.

Evidently, the quadriphase and biphase situations are quite similar. Quadriphase modulation entails no more ambiguities than biphase because the former provides node synchronization for free. Both techniques yield an all 1's no noise syndrome when locked 180° out of phase. Quadriphase modulation yields an EL syndrome when ±90° out of phase, while biphase does the same when out of node synchronization for both stable phase situations. The fact that the syndrome is EL depends, of course, on i(D) being EL. It can be readily seen from the off synchronization syndrome equations that certain information sequences will yield an all zero's

syndrome even when node or phase synchronization is incorrect. The all zeros and all ones information sequences are particularly bothersome in this regard.

An EL syndrome will be obtained if node synchronization is out or the phase has flipped by ±90°; in addition, the syndrome will be EL if one or both of the $n_i(D)$'s is EL. Thus an EL received stream will cause an EL syndrome. Bounds on the minimum distance of codewords (Ref. 7) shows that for the best rate 1/2 code, asymptotically the minimum distance increases as .11n where n is the code lenth in symbols. Thus the nearest codeword to an EL received sequence will tend to differ in about 1 symbol in every 9. This means when the decoder is out of node or phase synchronization, the best metric will on the average increase by 1 for each 9 received symbols. Actually, the real rate of increase may be somewhat greater than this because the Viterbi decoder will use a short constraint length code. The .11n asymptotic figure is approached in the limit of longer codes. The closest codeword to a random sequence will tend to be further away when the constraint length is short.

As mentioned previously, the factor k and the threshold N must be chosen such that the counter used to indicate data quality drifts upward when synchronization is incorrect and drifts downward when synchronization is correct. The actual values chosen for these parameters will determine:

(a) the expected time to first passage over the threshold, and hence change of system state, when node or phase

synchronization is incorrect. We will call the expected time to re-synchronization, E_{rs}.

(b) the expected first passage time when node and phase synchronization is correct. This is the expected time to false alarm, E_{fa}.

Obviously we want to make E_{rs} as short as possible and E_{fa} as large as possible. Analytical techniques have been successfully used to approximate E_{rs} and E_{fa} as a function of k, T and p. The analysis is based upon recognizing that the input to the up-down counter is a random walk with a reflecting boundary at zero and an absorbing boundary at N. First passage times are computed by solving the appropriate Fokker-Planck equation. This work is reported in detail in Appendix B.

In addition to the approximate analysis, simulations have been performed to tie down E_{rs} and E_{fa} more precisely. The count-up rate, k, was chosen to be 8 because that value is nearly optimum, and it is a power of 2 and, therefore, easy to implement. The rate 1/2, K=5 code was used with hard receiver quantization. Fig. 2.2.5.4 shows E_{fa} vs. the threshold T. This limited data supports the analytical results which state that E_{fa} rises exponentially with T. Naturally, for a given T, E_{fa} is larger for smaller channel crossover probabilities p.

Fig. 2.2.5.5 shows the other parameter E_{rs} as a function of T. E_{rs} also rises with T, but more slowly. In fact it is nearly linear with T, as predicted by theory (Appendix B). E_{rs} is not a function of p because it is the expected time to re-sync when the

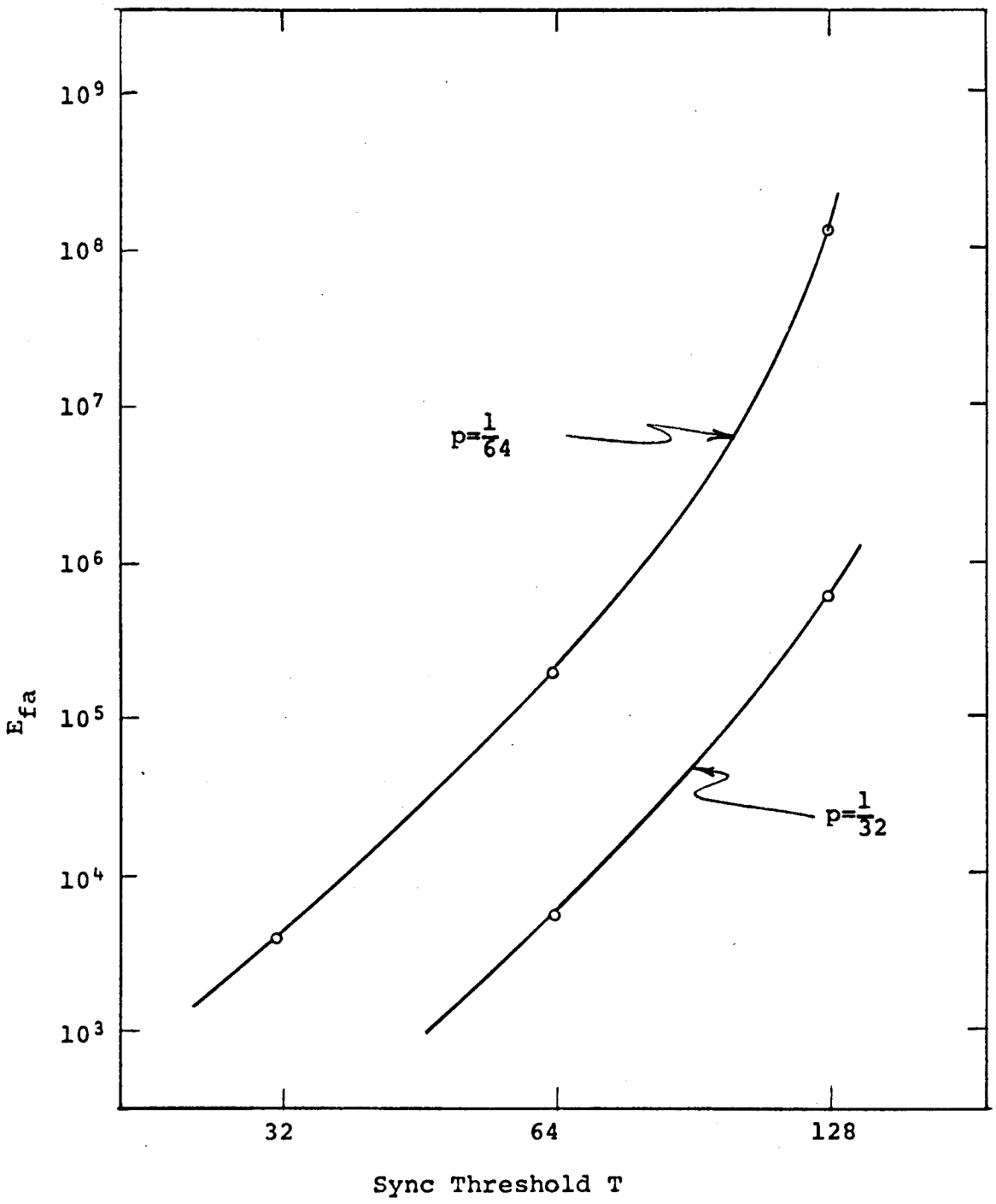


Fig. 2.2.5.4 Average number of bits decoded between false loss of sync events vs. sync counter threshold -- p is the channel crossover probability.

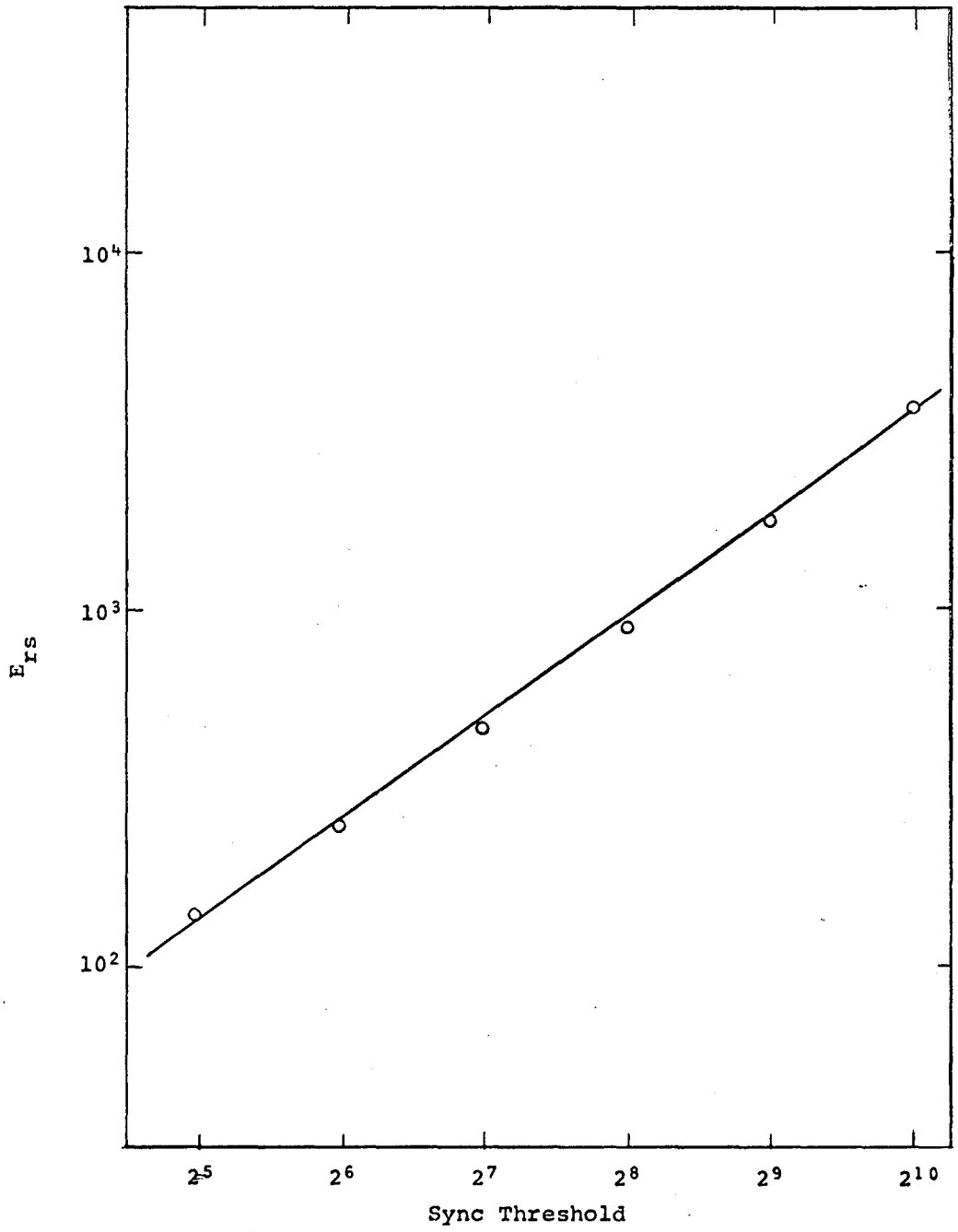


Fig. 2.2.5.5 Average number of bits to recover after loss of node synchronization vs. sync-counter threshold (hard decision decoder).

decoder is out of sync. When out of sync, the received data has the same random statistics regardless of the noise.

These results show that a value of T can be selected to make E_{fa} truely negligible, while maintaining E_{rs}, the resync time, as low as several hundred to a thousand bits. The increase in system bit error rate due to false loss of sync is

$$2E_{rs}/E_{fa}$$

This is because, on the average, for E_{fa} bits decoded sync is lost once. It takes, on the average, $2E_{rs}$ to return to the proper sync state.

2.2.5.2 Transparent Codes. As mentioned in the previous section, another way to resolve 180° phase ambiguities is to use a code which is transparent to 180° phase flips, precode the data differentially and use differential decoding. A transparent code has the property that the bit-by-bit complement of a codeword is also a codeword. Such a code must have an odd number of taps on each of its encoder mod-2 adders. This insures that if a given data sequence generates a certain codeword, its complement will generate the complementary code word.

If the received data is complemented due to a 180° phase reversal, it will still look like a codeword to the decoder, and will likely be decoded into the complement of the correct data sequence. Now decoding to the complement of the sequence input to the encoder is no problem if the data was precoded differentially. This means that information is contained in the occurrence or non-occurrence of transitions in the encoded output sequence rather

than the absolute sequence itself. These transitions occur in the same places even if the decoded sequence is complemented.

The major fault with this scheme is that when an isolated bit error occurs in the decoder output, it causes two differentially decoded errors, since two transitions are changed. At first glance, this would seem to indicate a doubling of the output bit error rate. In fact, this doubling does not occur because errors typically occur in short bursts. Two adjacent bit errors, for instance, cause only two differentially decoded bit errors. This indicates the possibility of only a small increase in bit error rate with differential encoding-decoding.

Fig. 2.2.5.6 shows bit error rate performance curves for the K=7, rate 1/2 code with and without differential encloding-decoding. The degradation is error probability is least at low E_b/N_0. Here decoder error bursts are relatively long (on the average one to two constraint lengths), so differential encoding-decoding loses very little. At higher E_b/N_0, bit error rate degradation is slightly larger--but nowhere near a factor of two. The worst bit error rate degradation factor is about 1.2 over the range shown. Using differential encoding-decoding, the E_b/N_0 required increases by less than 0.1 db. K=7 was selected for this example because the optimum code for this constraint length is transparent (see Table 2.2.1).

The use of differential encoding-decoding reduces the synchronization problem to 2-state node synchronization with BPSK, and 2-state 90° phase ambiguity resolution with QPSK.

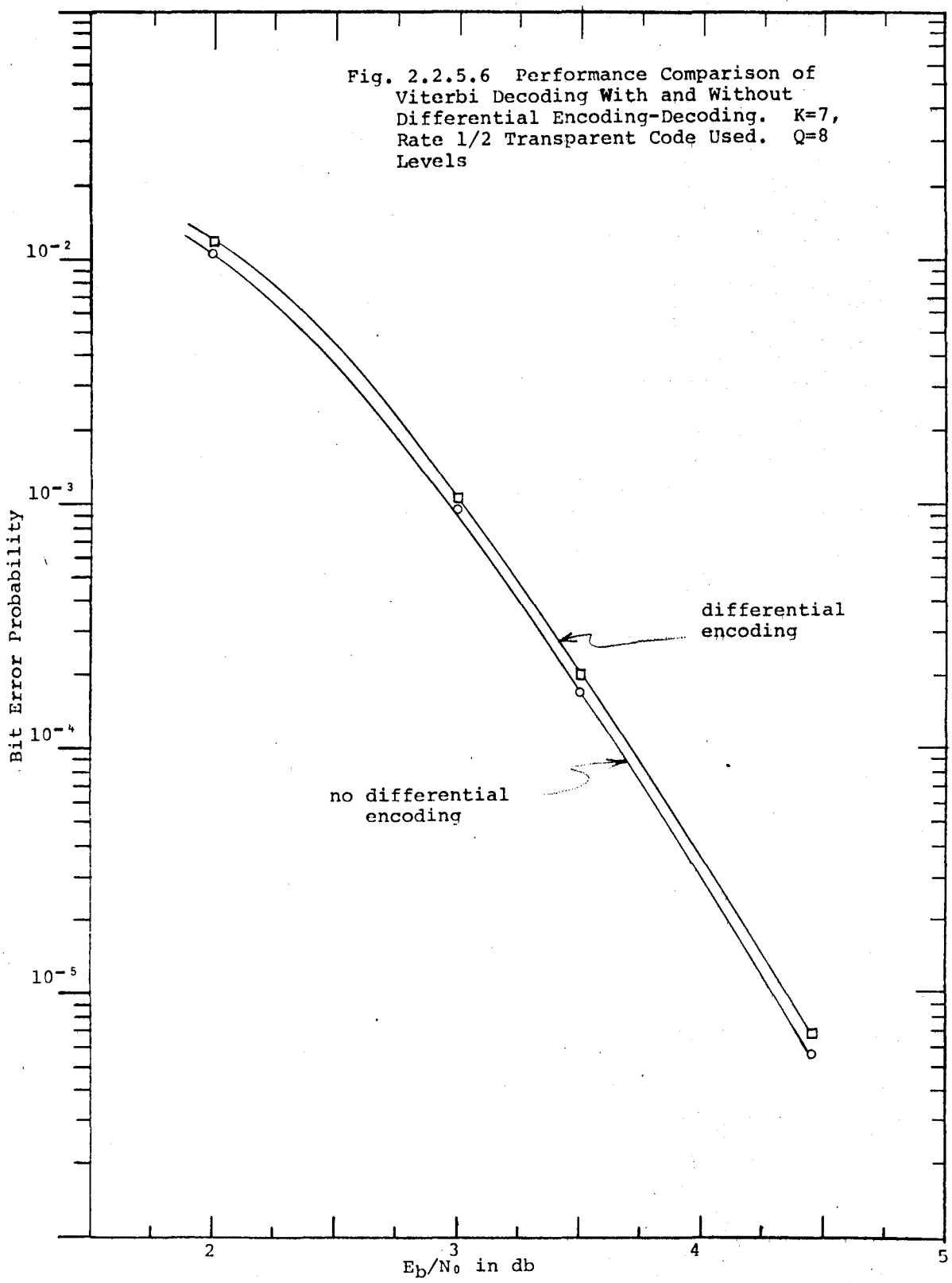


Fig. 2.2.5.6 Performance Comparison of Viterbi Decoding With and Without Differential Encoding-Decoding. K=7, Rate 1/2 Transparent Code Used. Q=8 Levels

2.2.5.3 Channel Reliability Information. The up-down counter used for node and/or phase synchronization also provides a means for very sensitive measurement of channel performance. In the absence of decoder errors, the counter counts up whenever a received symbol with a channel error is processed. Thus, the number of times the counter counts up per unit time is directly proportional to the channel error rate p. This will be true as long as the decoder output error rate is low, which corresponds to good system performance. If p becomes large due to a system failure or loss of code sync, the count will tend to remain above zero.

One method of monitoring system realiability is to sum the count over a number of bits which is large compared with 1/p, where p is the lowest channel error rate to be monitored. The integrator output will be stable and proportional to p when p is in the range corresponding to decoder error rates lower than about 10^{-2}. As p rises beyond this point, the integrator output will rise monotonically, but not proportionally. When p becomes greater than about.11, the integrator output will saturate. This is because the number of decoder corrections (metric increases) for a rate 1/2 code never exceeds .11 on the average as was discussed in the previous section.

Integration of the sync counter value over a fixed time window therefore provides a sensitive measure of p when the decoder is putting out useful data. It also is a good indicator of system failure.

2.2.6 Sensitivity to AGC Inaccuracy. Coded systems which make use of receiver outputs quantized to more than two levels require an analog-to-digital converter at the modem matched filter output, with thresholds that depend on the noise variance. For instance, all of the 8-level quantized Viterbi decoder simulations reported on thus far have used level thresholds at 0, $\pm 0.5\sigma$, $\pm\sigma$, and $\pm 1.5\sigma$.

Since the level settings are effectively controlled by the automatic gain control (AGC) circuitry in the modem, it is of interest to investigate the sensitivity of decoder performance to an inaccurate or drifting AGC signal. Fig. 2.2.6 shows the decoder performance variation as a function of A-D converter level threshold spacing (in all cases the thresholds are uniformly spaced). These simulations used the K=5 rate 1/2 code, with E_b/N_0 = 3.5 db. It is evident that Viterbi decoding performance is quite insensitive to wide variations in AGC gain. In fact, performance is essentially constant over a range of spacings from 0.5σ to 0.7σ. This allows for a variation in AGC gain of better than 20% with no significant performance degradation.

2.3 Other Code Rates.

The preceding sections have concentrated on Viterbi decoding of rate 1/2 convolutional codes. Most of the results on performance fluctuation due to decoder parameter variation carry over qualitatively, if not quantatively, to other code rates.

Code rates less than 1/2 will buy improved performance at the expense of increased bandwidth expansion and more difficult

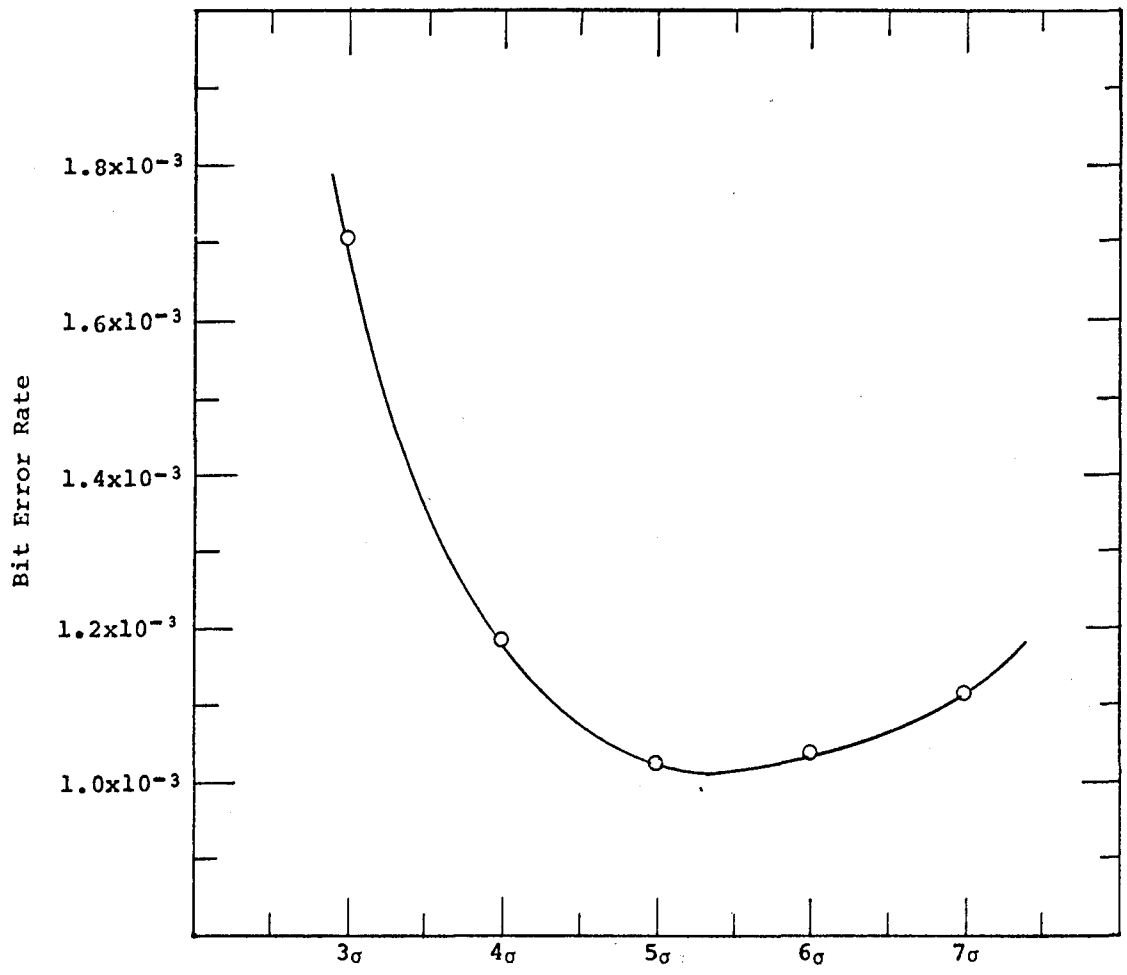


Fig. 2.2.6 Viterbi decoder Bit Error Rate Performance as a Function of Quantizer Threshold Level Spacing -- K=5, Rate 1/2, E_b/N_0=3.5 db, 8-level Quantization with Equally Spaced Thresholds.

symbol tracking due to decreased symbol energy to noise ratios. Rates above 1/2 conserve bandwidth but are less efficient in energy.

2.3.1 Description of Code Search Program. Optimum short constraint length, rate 1/3 codes have been found previously (Ref. 2,3). Our efforts in searching for good codes were confined to rate 2/3, K=6 and 8, and rate 3/4, K=6 codes. Since the number of possible codes is quite large (there are 2^{24} rate 2/3, K=8, and rate 3/4, K=6 codes), a fast method was needed to evaluate and select, or reject codes. The method chosen uses the convolutional code transfer function described in section 2.2.2 and Appendix A. Before going into the technique in detail, it will be instructive to discuss a result which limits the number of codes which must be considered.

The optimum rate 3/4, K=6 encoder is shown in Fig. 2.3.1.1. The encoder consists of a K stage shift register, as in the rate 1/2 case. For a general rate k/n code, however, k binary digits are shifted into the coder simultaneously. The coder stages are connected to n mod-2 adders. Note that the trellis formed by a rate k/n code has $2^{(K-k)}$ states with 2^k branches leaving and entering each node. This is because any one of 2^k groups of k binary digits can enter the coder at once. Decoding involves making a 2^k-wise decision for each of 2^{K-k} states per k bits decoded.

It can be shown that no generality is lost if the codes are restricted in the following way. Suppose we label the mod-2 adders

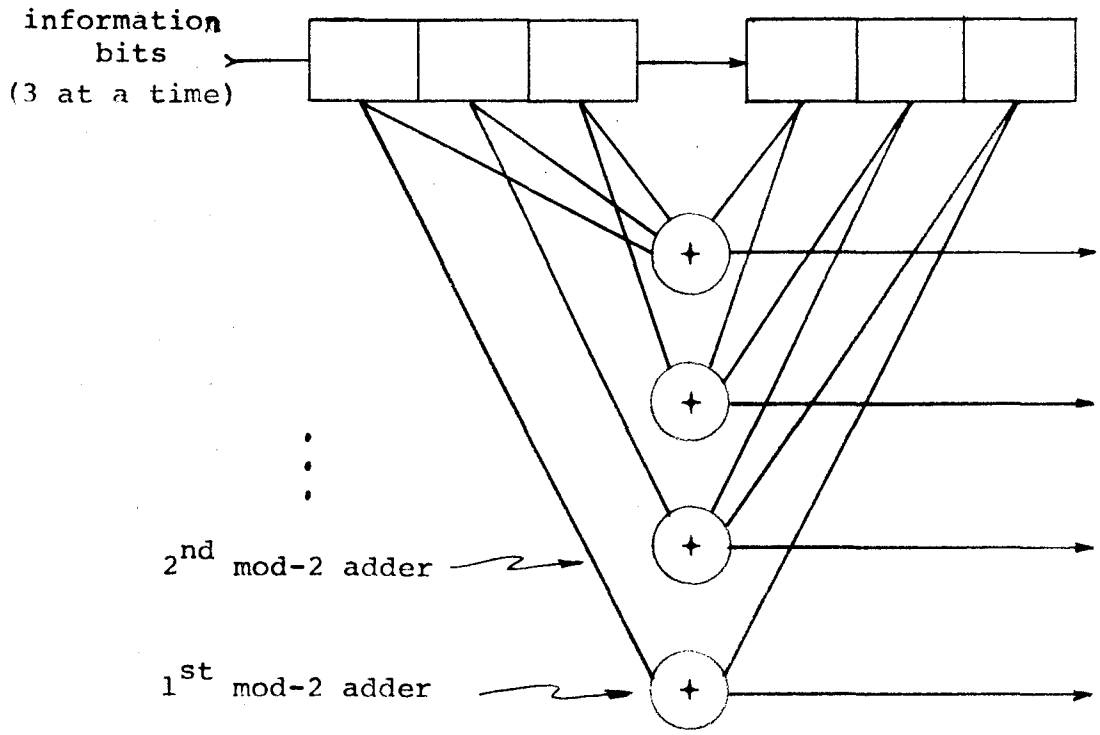


Fig. 2.3.1.1 Optimum Rate 3/4,
K=6 Code Encoder

from 1 through k, and concern ourselves with connections to the first k encoder stages. Connect adder 1 to stage 1, adder 2 to stage 2, ..., and adder k to stage k. Make no other connections between the first k stages and the first k adders. This reduces the number of codes to be searched by a factor of 2^{k^2}. The encoder in Fig. 2.3.1.1 is of the type described.

Recall that the code transfer function (for N=1) is of the form

$$T(D) = a_1 D^{d_f} + a_2 D^{d_f + 1} + \ldots + a_{i+1} D^{d_f + i} + \ldots \quad (2.3.1)$$

where d_f is the code minimum free distance. If we set $D=10^{-6}$, and if the rate of growth of the a_i satisfies certain conditions (see Appendix A), then $T(10^{-6})$ will be very close to $a_1 10^{-6d_f}$. The code search program has as an input a target value of d_f. It evaluates the first few terms of the transfer function, with $D=10^{-6}$, for each code, and tests to see if their sum exceeds $10^{-6(d_f-1)}$. If it does, this means that the code must have a minimum free distance smaller than d_f (or $a_1 > 10^6$ which is impossibly large). Thus, the code is rejected. On the other hand, if the iteratively evaluated transfer function remains below $10^{-6(d_f-1)}$ taken to a sufficiently large number of terms, the code's free distance is at least d_f, and the code is printed out.

This technique often results in the generation of many candidate codes (or none of the target d_f is too large). The codes are then tested using the numerical transfer function union bound program to approximate the bit error rate performance. Final code selections are made on this basis.

2.3.2 Good Rate 1/3, 2/3, and 3/4 Codes. Optimum rate 1/3, K=3 through 8, rate 2/3, K=6 and 8, and rate 3/4, K=6 codes are tabulated in Tables 2.3.2.1, 2.3.2.2, and 2.3.2.3 respectively. Included also is the free distance of each code d_f, the number of bit errors in paths at the minimum distance n_e, and the value of an upper bound on minimum free distance d_f^*, analogous to that obtained for rate 1/2 codes.

The rate 1/3 codes were reported previously (Ref. 3) and are simply repeated here. The rate 2/3 and 3/4 codes are the result of the code search program.

2.3.3 Simulation and Numerical Performance Data. Figure 2.3.3.1 shows bit error rate vs. E_b/N_0 performance obtained from simulations of Viterbi decoding with optimum rate 1/3, K=4,6, and 8 codes, and 2 and 8 level quantization. The K=4 and 6 results were reported elsewhere (Ref. 2) previously. The K=8 code used in Ref. 2 was suboptimal; thus, the curve shown here for the optimum code is somewhat better than the previously reported result.

Figures 2.3.3.2, 2.3.3.3, and 2.3.3.4 show numerical bound and simulation performance results for the rate 2/3 K=6, K=8 and rate 3/4 K=6 codes respectively. Simulation curves are for 2 and 8 level quantization, while the numerical bound curves are, as usual, for infinitely fine receiver quantization.

K	Code Generators	d_f	n_e	d_f^*
3	111 111 101	8	3	8
4	1111 1101 1011	10	6	10
5	11111 11011 10101	12	12	12
6	111101 101011 100111	13	1	13
7	1111001 1100101 1011011	14	1	15
8	11110111 11011001 10010101	16	1	17

Table 2.3.2.1 Optimum Rate 1/3 Codes

K	Code Generators	d_f	n_e	d_f^*
6	101111 011001 110010	5	5	6
8	10110110 01111001 11110111	7	86	8

Table 2.3.2.2 Optimum Rate 2/3 Codes.

K	Code Generators	d_f	n_e	d_f^*
6	100001 010011 001110 111101	4	40	4

Table 2.3.2.3 Optimum Rate 3/4 Code.

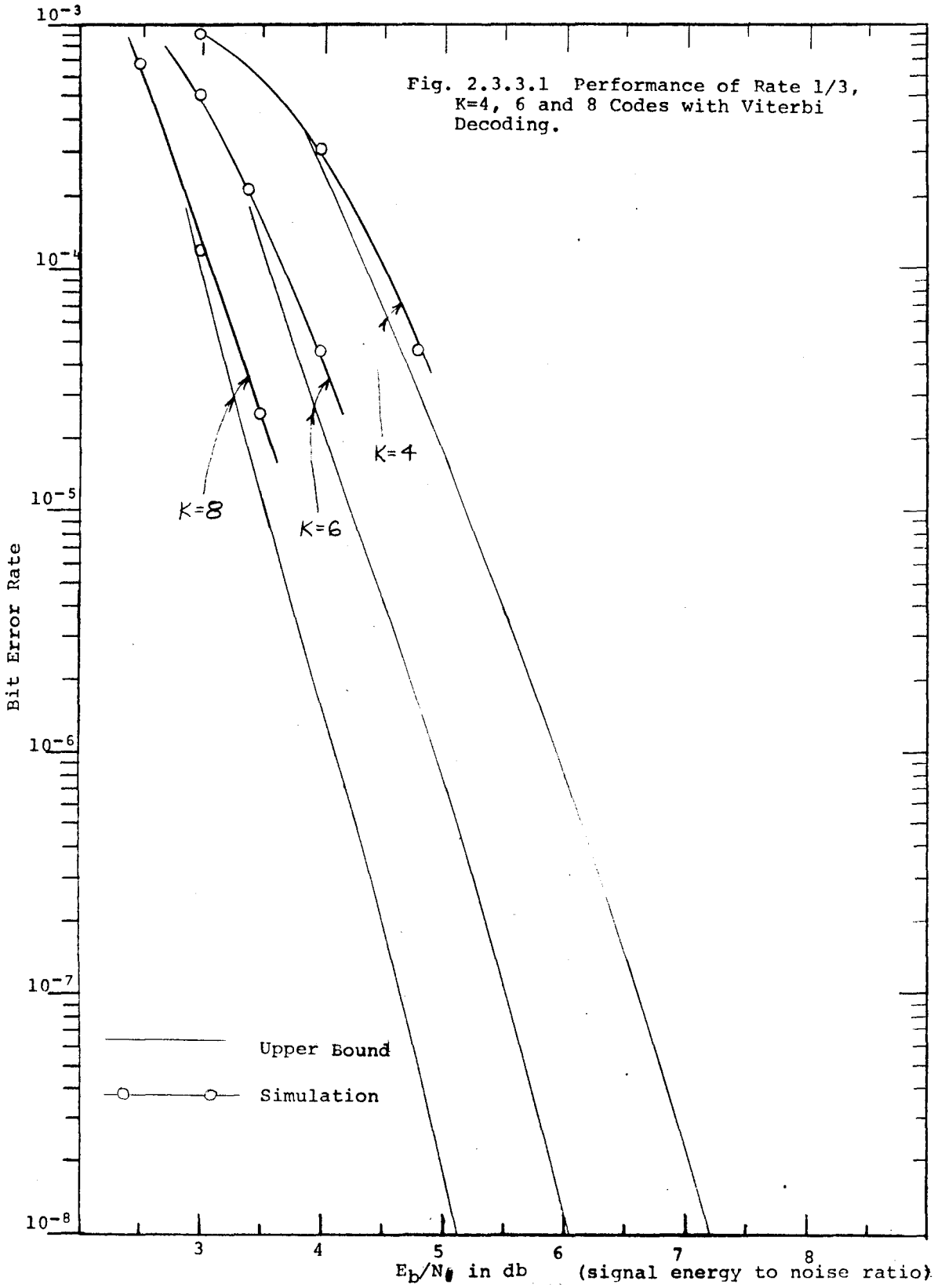


Fig. 2.3.3.1 Performance of Rate 1/3, K=4, 6 and 8 Codes with Viterbi Decoding.

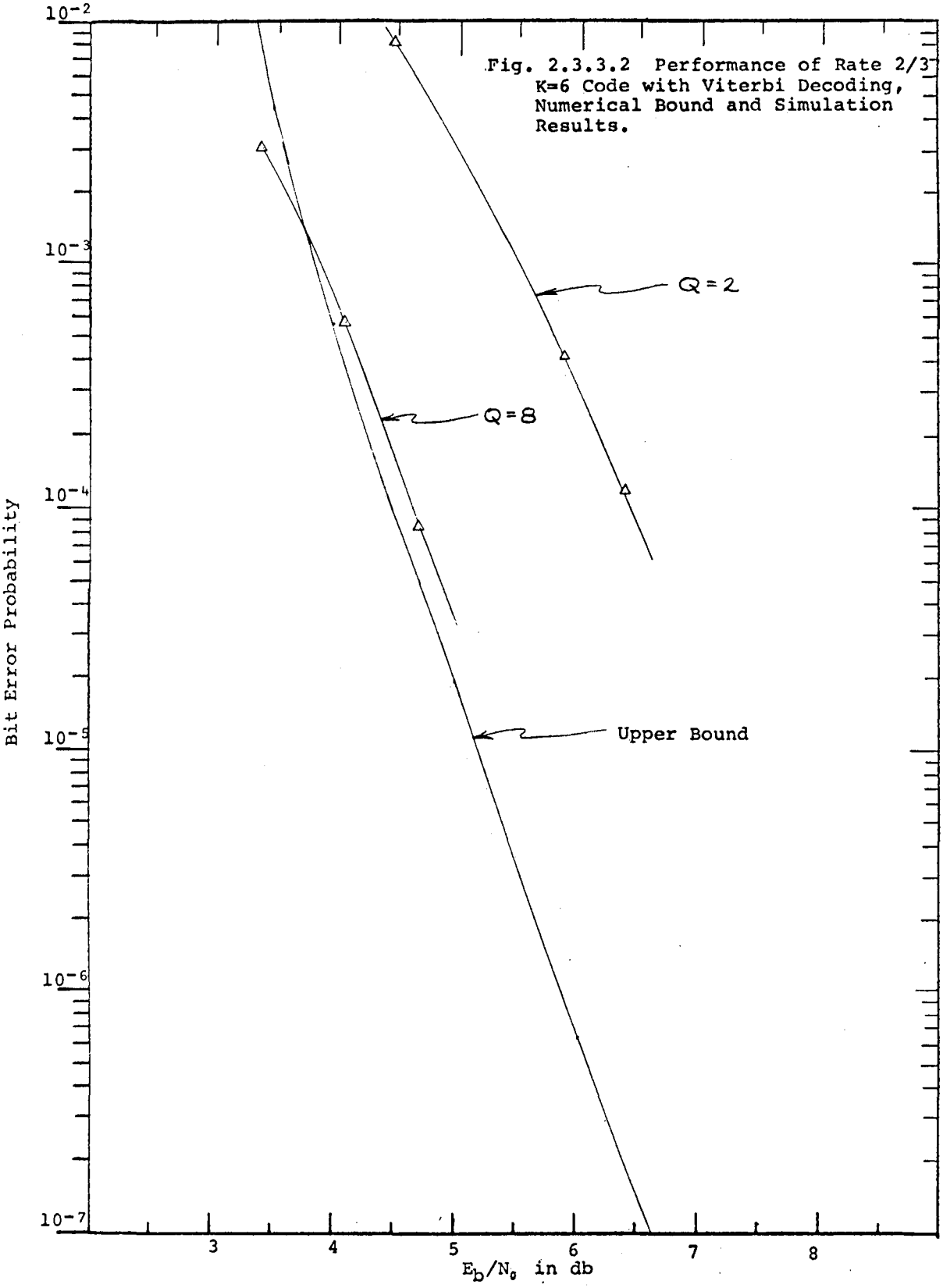


Fig. 2.3.3.2 Performance of Rate 2/3 K=6 Code with Viterbi Decoding, Numerical Bound and Simulation Results.

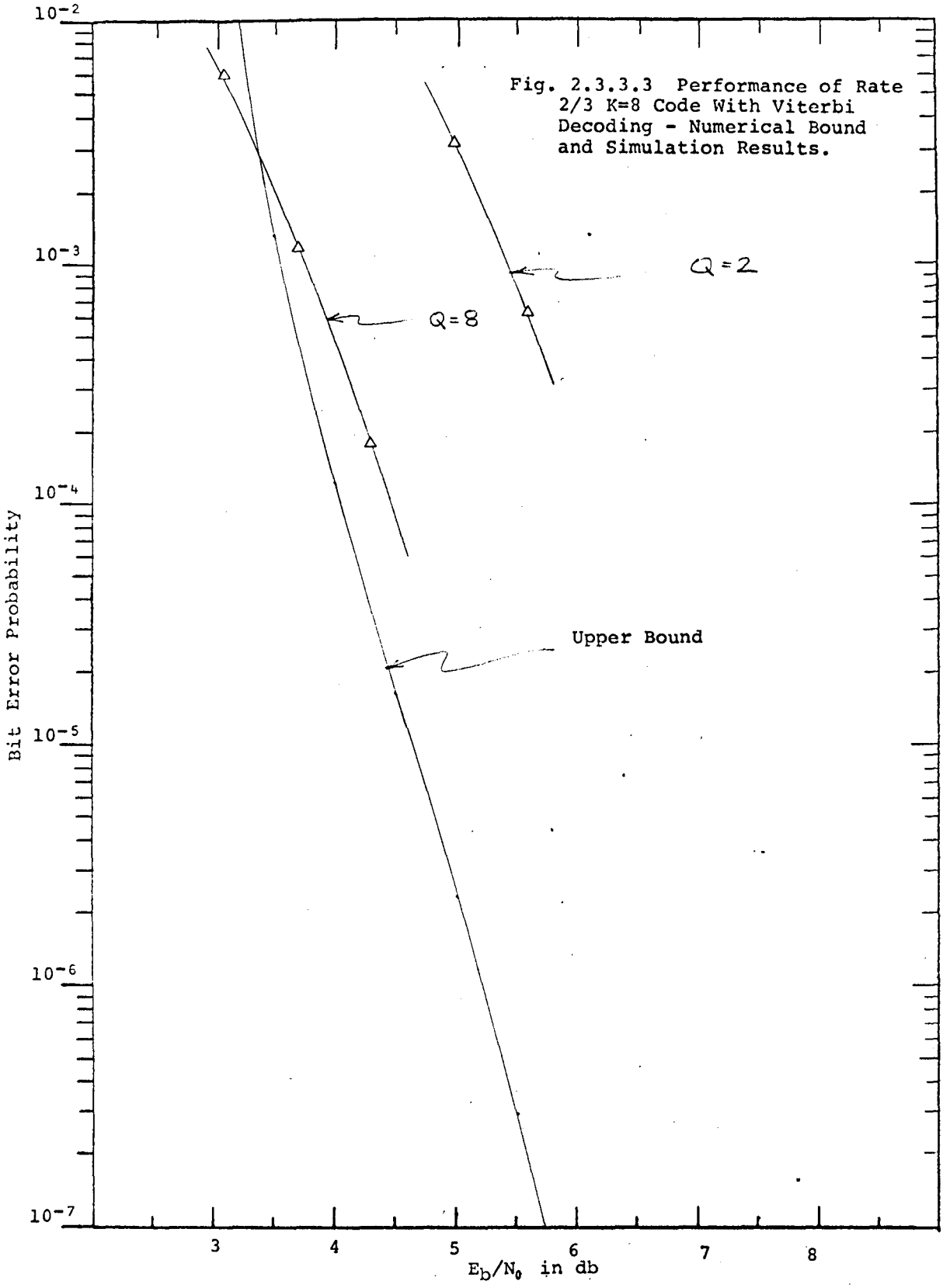


Fig. 2.3.3.3 Performance of Rate 2/3 K=8 Code With Viterbi Decoding - Numerical Bound and Simulation Results.

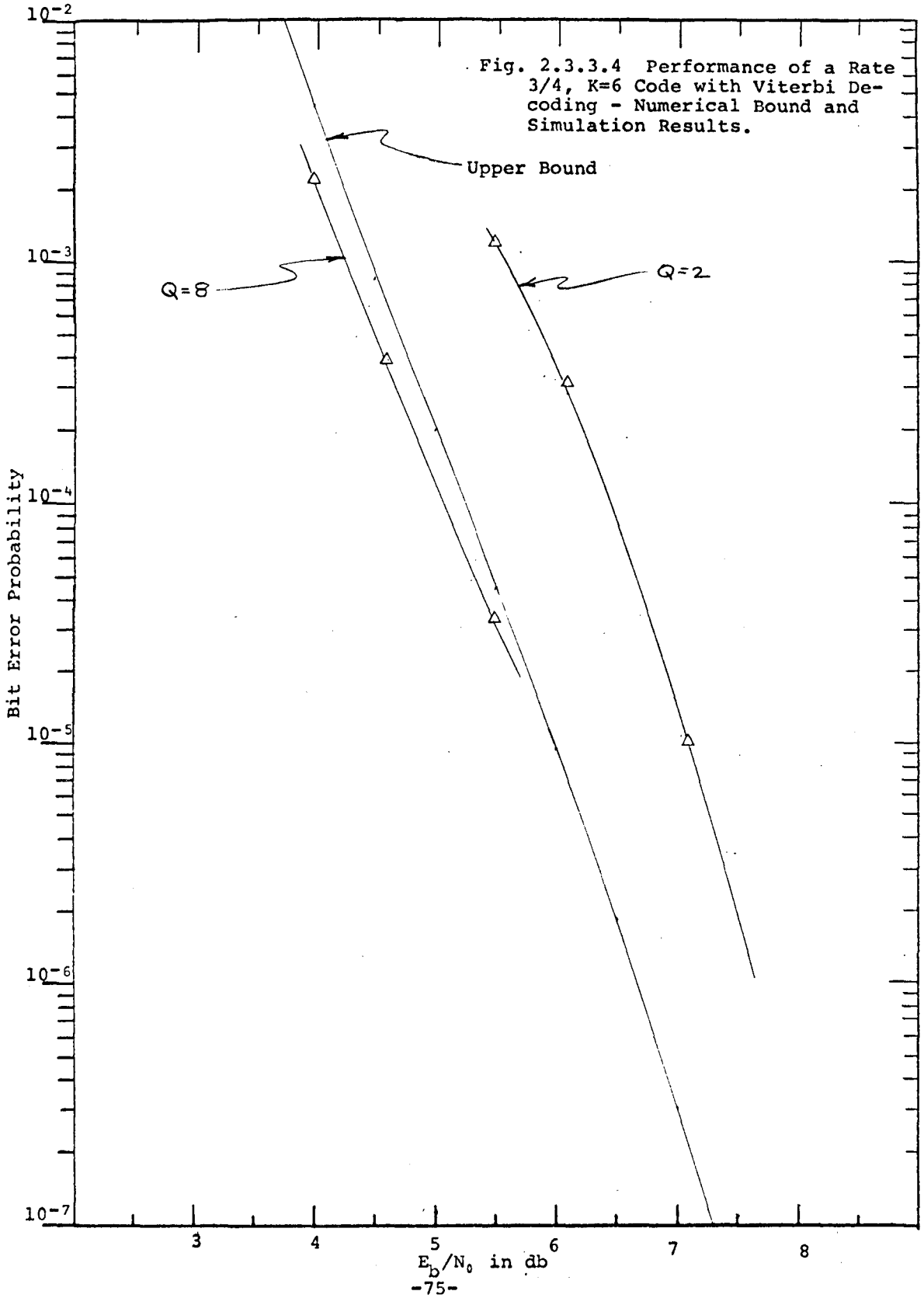


Fig. 2.3.3.4 Performance of a Rate 3/4, K=6 Code with Viterbi Decoding - Numerical Bound and Simulation Results.

2.3.4 Comparison with Rate 1/2 Code. Comparing the performance data obtained through simulations of Viterbi decoders with rate 1/2 (Figs. 2.2.4.1, 2.2.4.2, and 2.2.4.3), and rate 1/3 codes, it is apparent that the latter offers a 0.3 to 0.5 db improvement over the former for fixed K, in the range reported. This is close to the improvement in efficiency of a channel with capacity 1/3 compared with one of capacity 1/2, and is therefore expected.

Performance efficiency of the higher rate codes also is predictable when compared with the rate 1/2 codes over the range with the simulation data spans. The fairest comparisons are probably those between decoders with like number of states. Thus, the K=6 rate 2/3 data should be compared with the K=5, rate 1/2 data.

Figure 2.3.4 shows the union bounds on performances for the rate 2/3, K=6, and rate 1/2, K=5 codes. Both encoders have 16 states. $d_f = 7$ for the rate 1/2 code and 5 for the rate 2/3 codes. At very high E_b/N_0, the rate 1/2 must be superior. This is because asymptotically, at high E_b/N_0, the error probability goes as

$$P_e \sim \exp\ (d_f E_s/N_0) = \exp\ (d_f R E_b/N_0)$$

This gives the rate 1/2 code an advantage of about 0.2 db in the limit.

In Fig. 2.3.4, the difference between the two curves is about 0.1 db in the error probability range of 10^{-6} to 10^{-9}. This small

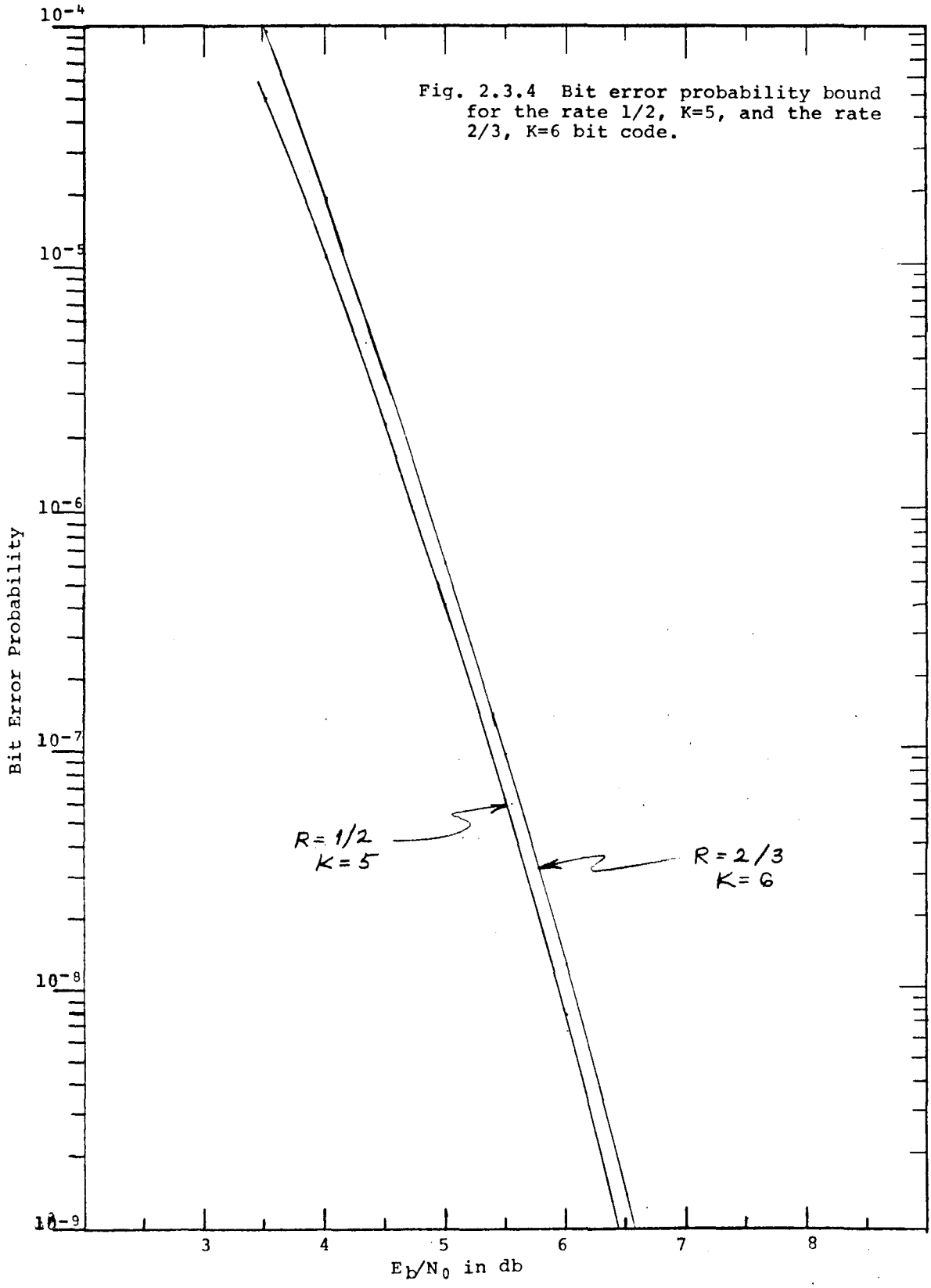


Fig. 2.3.4 Bit error probability bound for the rate 1/2, K=5, and the rate 2/3, K=6 bit code.

difference is due to the fact that the rate 2/3 code used is a particularly good code (it has a minimum distance coefficient of only 5 in the union bound for bit error probability). Some additional discussion on decoder complexity vs. rate is contained in section 2.4.

2.4 Viterbi Decoder Implementation*

2.4.1 Review of Decoder Algorithm. The present discussion is specialized to a decoder for a constraint length K=4 code. The design principles are the same for other values of K. Six features will be discussed in the following sections relating to:

1) the assignment of metric to the received data (metric compression)
2) the efficient storage of state metrics (overflow-protection)
3) the design of the decision-memory and the selection of the output bit (maximum likelihood selection)
4) the time sharing of state metric calculation
5) choice of logic family (TTL or MECL)
6) relation of cost to constraint length, code rate, speed, etc.

To set the stage for this discussion, we first review the fundamental operations of the Viterbi decoder. The code under discussion is that given by the K=4 convolutional encoder of Fig. 2.4.1.1. The states of the code at any time correspond to the contents of the first K-1=3 stages of the shift register. The coder is shown in state 011. It previously was in state 110; an input 0 caused a coder transition to state 011 and a coder output of the 2 check digits c_1c_2 = 01.

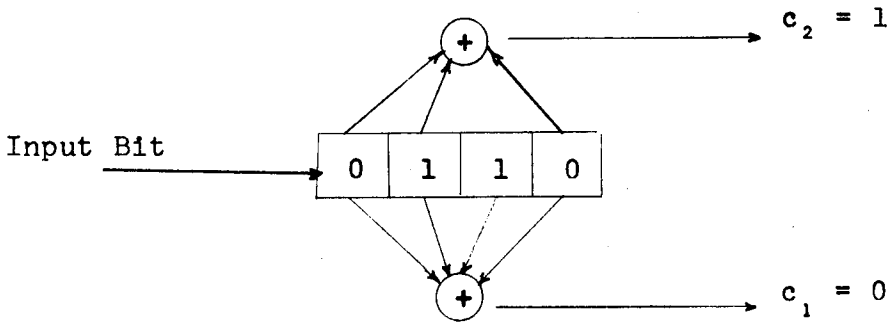
---

*Many of the ideas in this section concerning metric compression, overflow protection and output selection were first formulated in Ref. 10, and are the subject of patent applications either in preparation or pending.

A 4 state transition diagram for states 110, 111, 011, and 111 is given in Fig. 2.4.1.2. For any value of K, sets of 4 states group together with x0 and x1 on the left and 0x and 1x on the right, where x is any k-2 bit sequence. For good codes, both the first and last encoder shift register stages are connected to all the mod-2 adders. For these codes, the check digits c_1c_2 for the group of 4 are always complementary as depicted in Fig. 2.4.1.3.

A block diagram of a Viterbi decoder is shown in Fig. 2.4.1.4. The channel and modem cause the conversion of the transmitted digits c_1c_2 to received digits r_1r_2. From r_1r_2, 4 "metrics" are calculated by the input section, R_{00}, R_{01}, R_{10}, R_{11}, corresponding to the 4 possible values of c_1c_2. On the basis of these, the decoder must decide, for all values of x, whether state 0x was entered from state x0 or state x1 and similarly whether state 1x was entered from state x0 or state x1. These decisions are made by the ACS (add-compare-select) circuits. These binary decisions are denoted by the variables D_{ix}. For i=0 or 1, $D_{ix} = 0$ indicates a decision in favor of the transition from state x0 to state ix, whereas $D_{ix} = 1$ indicates a decision in favor of the transition x1 to ix.

The decisions are made on the basis of metrics associated with each state. Let M_{xj} denote the metric associated with state xj, j=0, 1. Then the decoder makes decisions -- see Figure 2.4.1.3 -- as follows (the convention that small metrics are "good" is used):



Note: ⊕ denotes modulo-2 addition

Fig. 2.4.1.1 K=4 Convolutional Encoder, Rate = ½

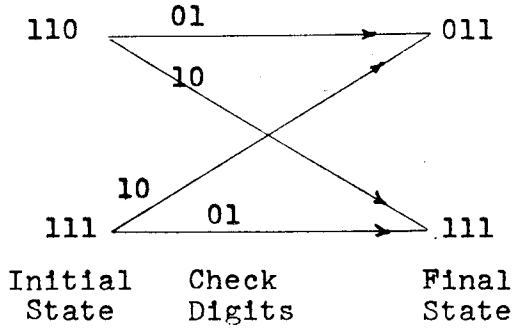


Fig. 2.4.1.2 4-State Transition Diagram for a One Input Step for Coder of Fig. 2.4.1

(0 input causes transition to upper state;
1 input causes transition to lower state)

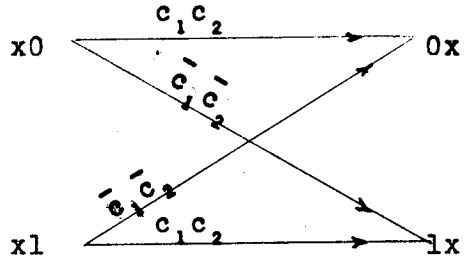


Fig. 2.4.1.3 General 4-State 1 Step Transition Diagram
($\bar{c}_i = 1 - c_i$)

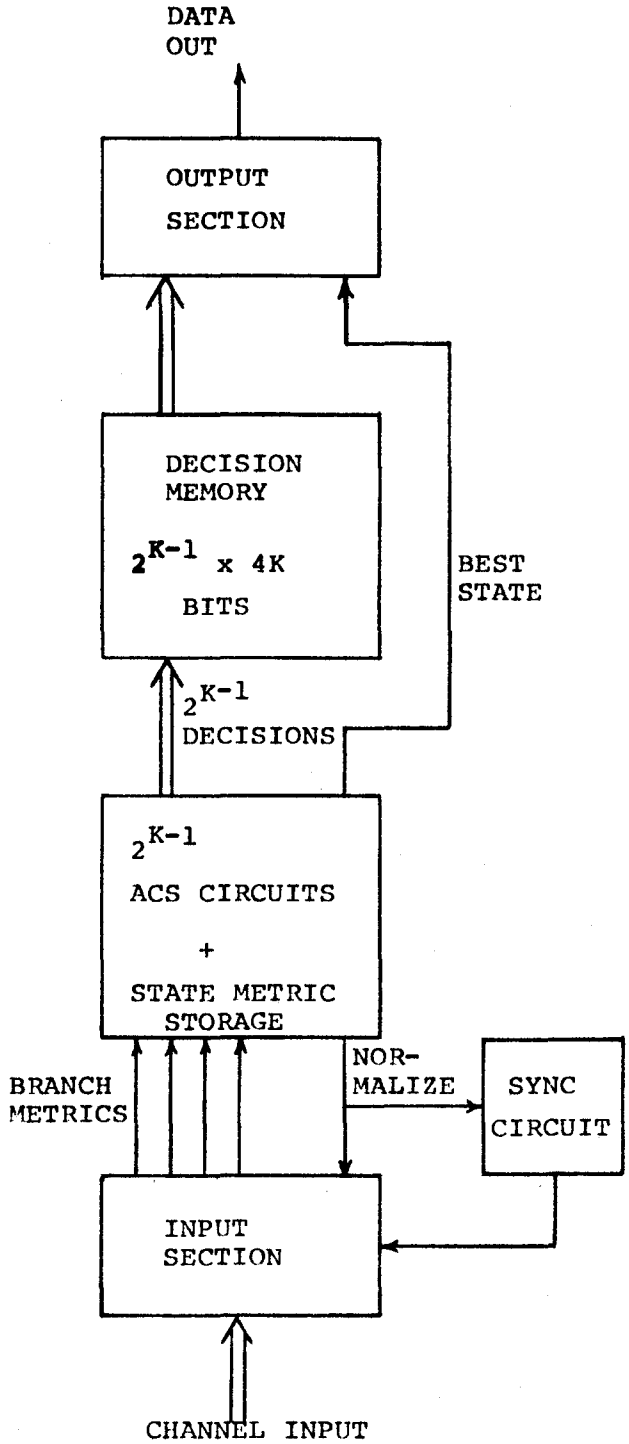


Fig. 2.4.1.4 VITERBI DECODER BLOCK DIAGRAM

$$
\begin{aligned}
M_{x0} + R_{c_1c_2} &\leq M_{x1} + R_{\bar{c}_1\bar{c}_2} && D_{0x} = 0 \\
M_{x0} + R_{c_1c_2} &> M_{x1} + R_{\bar{c}_1\bar{c}_2} && D_{0x} = 1 \\
M_{x0} + R_{\bar{c}_1\bar{c}_2} &\leq M_{x1} + R_{c_1c_2} && D_{1x} = 0 \\
M_{x0} + R_{\bar{c}_1\bar{c}_2} &> M_{x1} + R_{c_1c_2} && D_{1x} = 1
\end{aligned}
\qquad (2.4.1)
$$

Following the decisions, the set of D's (for K=4, D_{000}, D_{001}, D_{010}, D_{011}, D_{100}, D_{101}, D_{110}, D_{111}) are passed to the decision-memory section and are used to set the new values of the state metrics (exclusive of overflow-protection; see Section 2.4.3). For example, if $D_{0x} = 1$, the new value of the state metric M_{0x} is $M_{x1} + R_{\bar{c}_1\bar{c}_2}$.

For K=4, x takes on the values 00, 01, 10, and 11. Thus, 4 sets of decisions corresponding to (2.4.1.2) must be performed by the decoder. To implement the add and compare operations at high speed and reasonable cost, it is essential to minimize the number of bits used to represent the M's and R's. We discuss the minimization of the R representation, or metric compression, in Section 2.4.2 and the minimization of the M representation, overflow-protection, in Section 2.4.3. The method of overflow-protection has two dividends; it permits an inexpensive implementation of the maximum-likelihood output selection as discussed in Section 2.4.6 and of the code synchronization as discussed in Section 2.4.7. Tradeoffs in speed, complexity, logic family, constraint length, etc, are discussed in Section 2.4.8.

2.4.2 Metric Compression. The smallest achievable error probability would result if the received symbols, r_1 and r_2, were equal to the unquantized (or infinitely finely quantized) outputs of the appropriate matched filters, say u_1 and u_2. To permit digital implementation of the adder function, however, u_1 and u_2 are quantized to Q levels (For example, Q=2 denotes hard decisions where only the sign of the u_1's is kept). The loss in performance incurred by using Q=8 rather than Q=∞ is less than 0.2 db for a reasonable choice of quantizer thresholds.

For the decoder to be maximum likelihood, the branch metric, R_{ij}, should equal the log likelihood of the digits r_1 and r_2 received for that branch given that i and j are transmitted:

$$R_{ij} = \log P(r_1 \mid i \text{ trans}) + \log P(r_2 \mid j \text{ transmitted})$$

For a digital implementation, however, it is also necessary to quantize the $\{R_{ij}\}$ to a small number of levels. This quantization causes an additional performance loss.

We are interested in representing the branch metrics R_{00}, R_{01}, R_{10}, and R_{11} in a minimum number of bits. As noted earlier, the use of the quantized numbers r_1 and r_2 themselves to form the metrics provides system performance within 0.25 db (with Q=8) of the value achieved with Q=∞. Of course, as Q→∞, use of r_1 and r_2 becomes identical to use of u_1 and u_2 and hence is maximum

likelihood, while for Q=2, use of binary-valued r_1 and r_2 is the same as the use of Hamming distance and again is maximum likelihood.

Consider a Q level quantizer with the Q quantizer levels labelled by the integers 0, 1, 2, ..., Q-1 from most negative to most positive. (Any other labelling scheme can easily be mapped into this labelling scheme.)

If r_1 and r_2 denote the received symbols and take on the above values, then a choice of branch metrics shown to be acceptable by computer simulation is:

$$R'_{00} = r_1 + r_2$$
$$R'_{01} = r_1 + (Q-1-r_2)$$
$$R'_{10} = (Q-1-r_1) + r_2$$
$$R'_{11} = (Q-1-r_1) + (Q-1-r_2)$$

The LINKABIT technique of metric compression permits a reduction by 1 bit in the number of bits required to represent these numbers without degrading performance. Subtracting the same constant from each of the R_{ij} does not affect decoder decisions (see Eq. 2.4.1). We choose to subtract the smallest of the R_{ij} from each of the R_{ij}, thus yielding (at least) one value equal to zero and all the others non-negative. If R_{01} is smallest, for example, then

$$R_{\hat{0}\hat{0}} = R_{\hat{0}0} - R_{\hat{0}1} = 2r_2 - Q+1$$

$$R_{\hat{0}\hat{1}} = R_{\hat{0}1} - R_{\hat{0}1} = 0$$

$$R_{\hat{1}\hat{0}} = R_{\hat{1}0} - R_{\hat{0}1} = 2r_2 - 2r_1$$

$$R_{\hat{1}\hat{1}} = R_{\hat{1}1} - R_{\hat{0}1} = -2r_1 + Q-1$$

Observe in Eqs. 2.4.2 that if Q is odd then each of the R_{ij} is even and the least significant bit may be discarded. This is true in general and is the basis for the LINKABIT metric compression technique. If Q is even, however, this compression is not possible. Compression has a great impact on the amount of hardware required since, without compression, an extra bit is required throughout the arithmetic section and metric storage. This produces the peculiar result that 20% more hardware is required to implement Q=8 than Q=9. Fortunately, there is a simple solution to the problem of providing compression with an even value of Q, namely, to regard Q=8 as though it were Q=7 by mapping the two central zones of Q=8 into one zone. Although this results in slightly nonoptimum metrics, the degradation is quite small, especially at low error rates.

The above metric compression scheme is most easily implemented by using a read-only memory if the decoder is for data rates less than 10 Mbps. For rate 1/2, there are two received symbols, each quantized to three bits if Q=8 for a total of six bits. The ROM look up table must therefore have 64 entries. Since four branch metrics must be computed, each being a three bit number,

each word of the look up table must contain twelve bits. The table can be implemented by using two 512 bit ROM's.

A logic diagram of a TTL input section using ROM's is shown in Fig. 2.4.2.1. The circuit accepts either serial BPSK with the received digits, r_1 and r_2, interleaved or QPSK with digits r_1 and r_2 in parallel. The circuit then provides node sync by the selective insertion of a one symbol time delay in BPSK or by the selective interchange of the symbols in QPSK. If an interchange occurs, then r_1 is complemented. The sync circuit (explained in section 2.4.7), controls the bit insertion or interchange operations.

The symbols then address the 1024 bit ROM. The ROM is organized into 64 words of 12 bits each. The ROM outputs are the four branch metrics. The remaining circuitry subtracts four from the branch metrics when normalization (explained in section 2.4.3) is required.

At data rates higher than 10Mbps a different approach is used. The quantizer levels are labeled from -3 to +3 instead of from 0 to 6 as in the previous case. We then consider the symbols as sign-magnitude numbers, i.e., $r_1 = s_1\ t_1$, where s_1 is the sign and t_1 is the magnitude. The compressed metrics are then given by the following table:

s_1	s_2	R_{00}	R_{01}	R_{10}	R_{11}
0	0	0	t_2	t_1	t_1+t_2
0	1	t_2	0	t_1+t_2	t_1
1	0	t_1	t_1+t_2	0	t_2
1	1	t_1+t_2	t_1	t_2	0

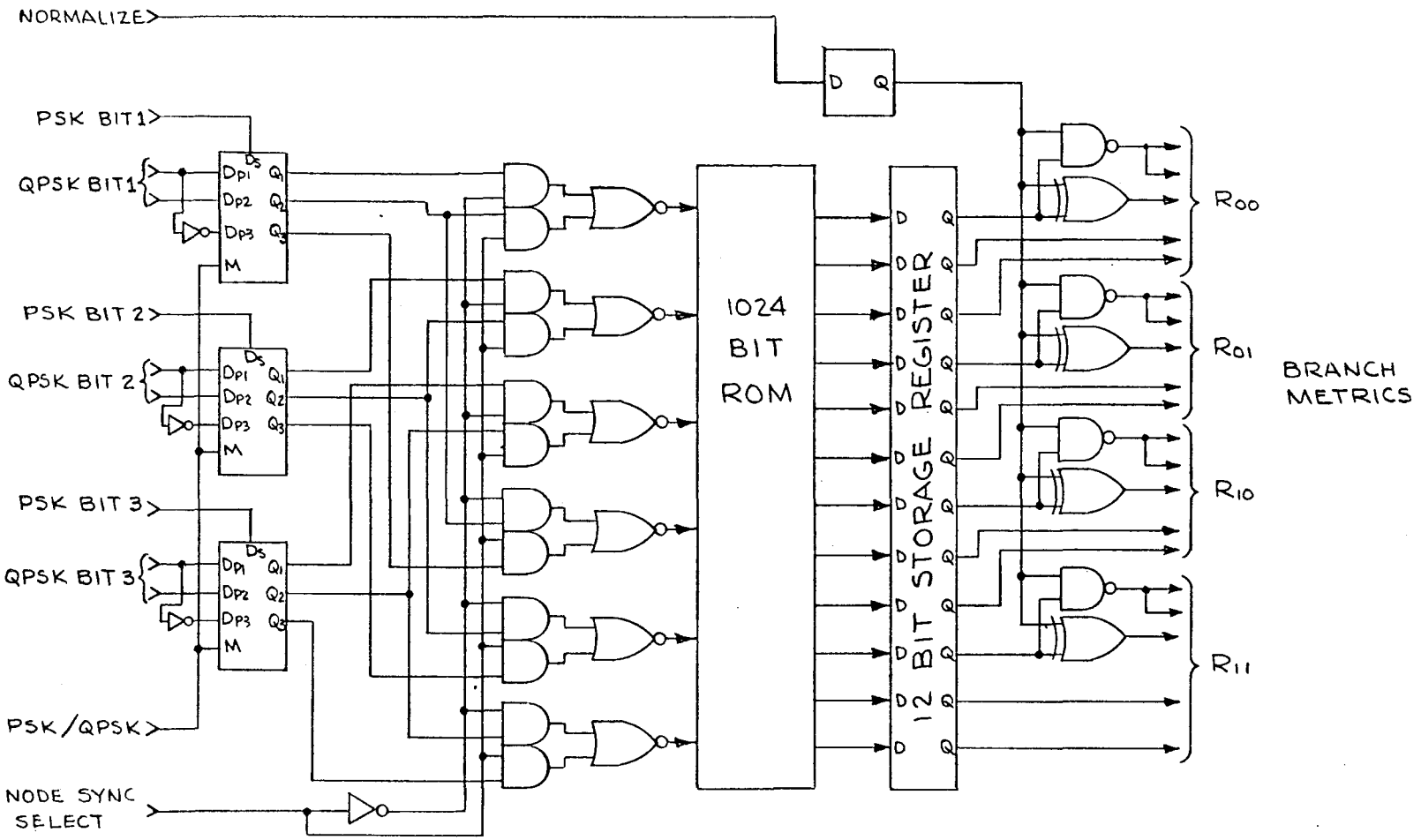


Figure 2.4.2.1

Logic Diagram, TTL Input Section

The assignments of this table are easily implemented as shown in Fig. 2.4.2.2. This circuit is preceded by node sync circuitry as in the previous example. The magnitudes are added and the branch metrics are obtained by using a multiplexer that is controlled by s_1 and s_2 to route 0, t_1, t_2 and $t_1 + t_2$ to the appropriate branch metrics. Normalization can be obtained by subtracting four as shown in the diagram.

2.4.3 Overflow Protection. In Section 2.4.2, we demonstrate how to compress the metrics to permit R_{00}, R_{01}, R_{10}, and R_{11} to be represented in the minimum number of bits. We now concentrate on mimimizing the maximum size of the 2^{K-1} state metrics, M_0, M_1, ..., M_{2K-1}. The technique is applicable to all K although the example is for K=4.

Consider the coder state diagram given in Fig. 2.4.3.1. It is possible to calculate the minimum Hamming distance in going from the 000-state to each of the other states. These numbers are shown in the square boxes adjacent to each state. Note that the maximum Hamming distance is 4. Because of the group property of the code, this means that any state is at most Hamming distance 4 from any other state.

This observation is critical to minimizing the number of bits required to represent the M's. For Q=8, Hamming distance 1 can imply an actual metric difference no greater than 3. Thus, it is possible for one state, say the 000 state, to have state metric $M_{000}=0$ while another state, for example the state 001, has state metric $M_{001}=12$. It is not possible for any state metric to be

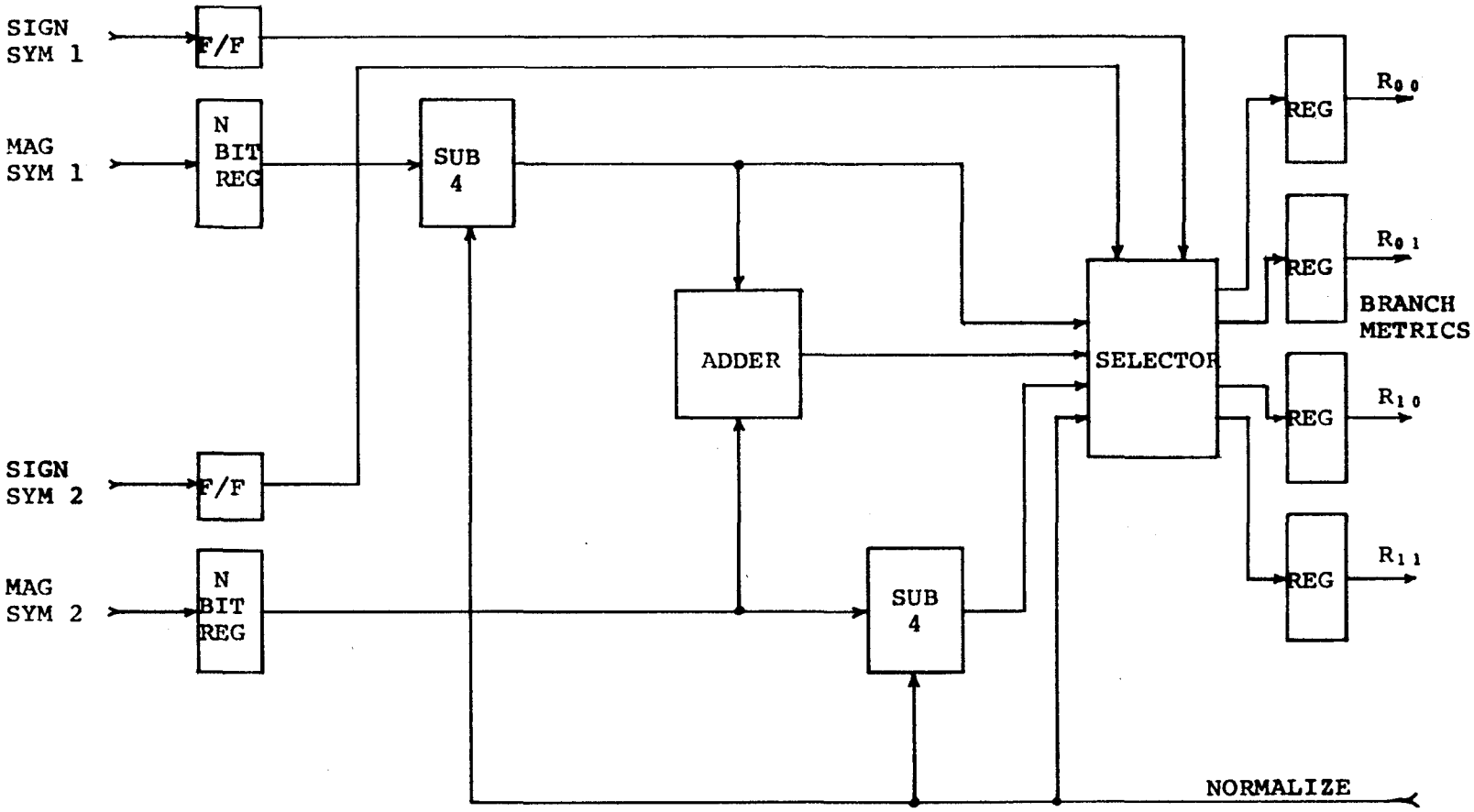


Fig. 2.4.2.2 VITERBI DECODER

ECL INPUT SECTION

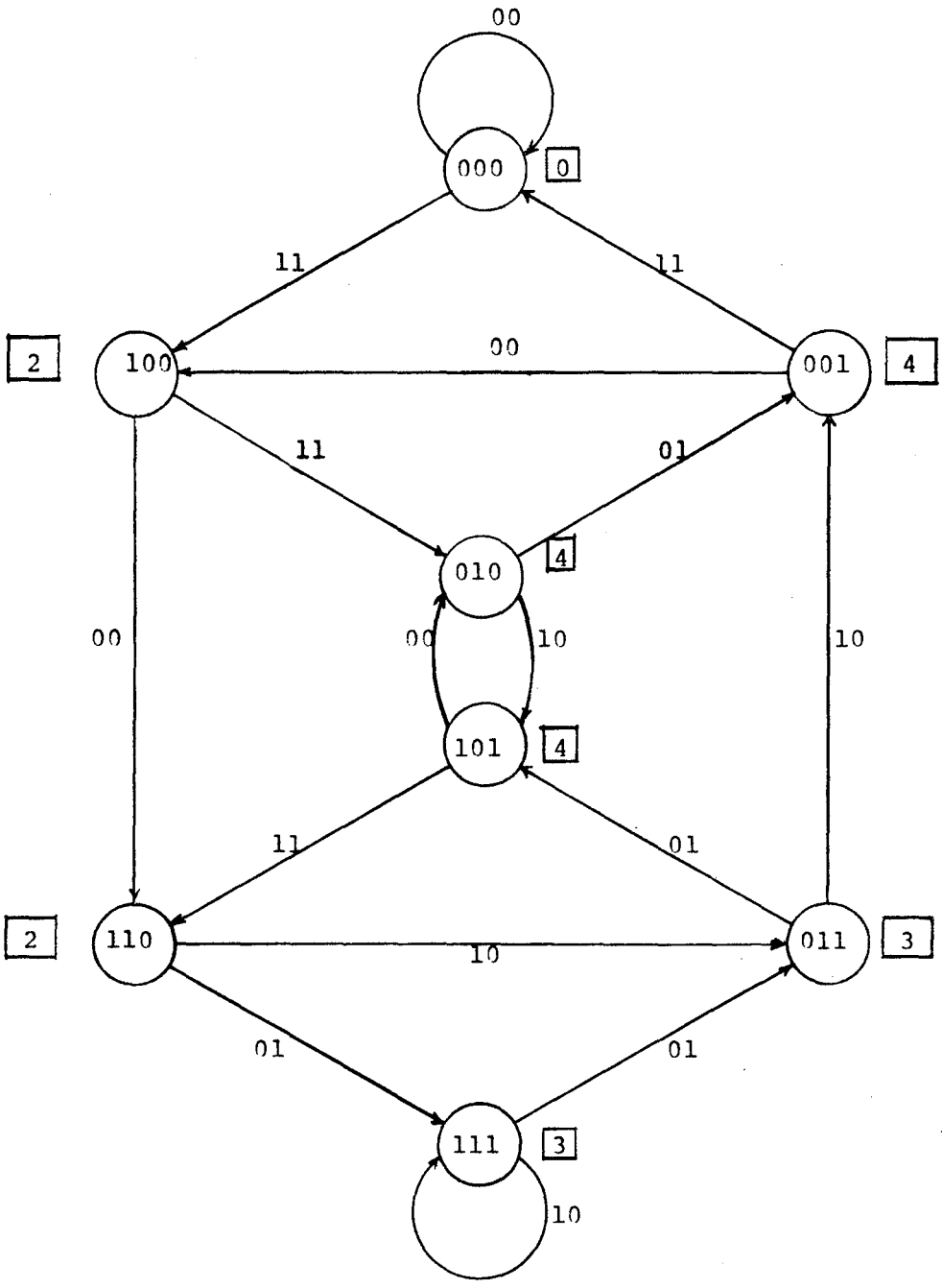


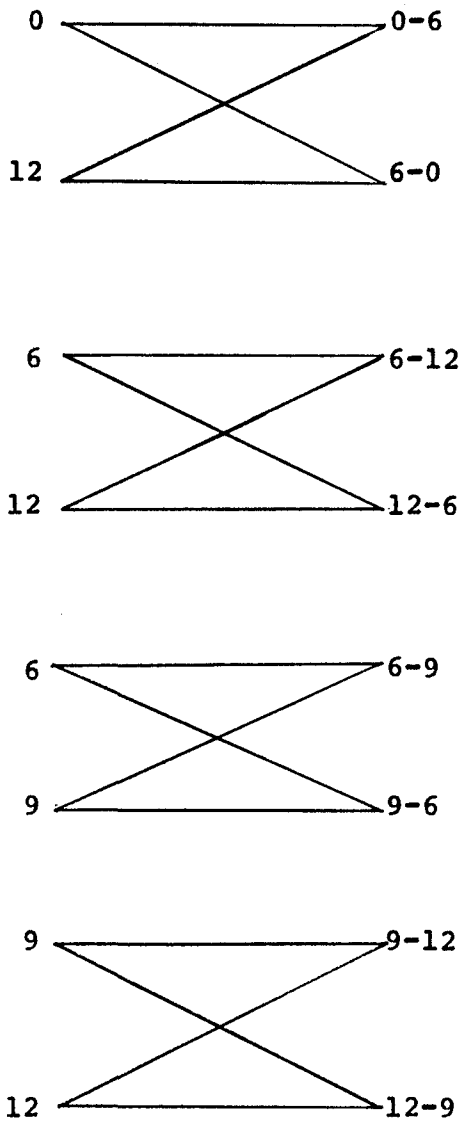
Fig. 2.4.3.1 - State Diagram with c_1c_2 Labelling Branches

Minimum Hamming Distance of Each State from the 000 State is Shown in Square Box Adjacent to State

larger than 12 when $M_{000}=0$ (or any other M=0), since the state in question can be reached from state 000 with at most Hamming distance 4 and hence compressed metric = 12.

We conclude that the difference between the smallest and largest state metrics never exceeds 12. It is therefore possible to represent the state metrics with 4 bits if a means of preventing overflows is also provided. To do so, we first observe that in a single transition, the smaller state metric in a pair of metrics can increase by at most 3. Furthermore, if the spread is equal to 12, the largest metric cannot increase at all. These claims are verified by examining the state-pair metric transition diagram in Fig. 2.4.3.2a and the possible sets of compressed metrics in Fig. 2.4.3.2b. The state-metrics are shown on the left in Fig. 2.4.3.2a for a case in which the spread between smallest and largest metric is the maximum value, 12. The range of the state metrics at the outputs are shown on the right (recall that the smaller (better) metric of the 2 metrics accessible to a state is chosen as the new state metric; see Equation 2.4.1.

We conclude that if the smallest state metric is in the range 0 to 4, the largest metric is less than or equal to 15 and hence does not overflow a 4 bit register. Furthermore, since the maximum increase in the smallest metric is 3, subtracting 4 from all state metrics whenever all of them exceed 3 prevents the set of state metrics from moving out of the range 0-15. This is the overflow-protection strategy that we have adopted. Note that we could equally well subtract 3 whenever the metrics are all greater



(a) State Metrics at Extreme Spread

R_{ij}	$R_{\bar{i}j}$	$R_{i\bar{j}}$	$R_{\bar{i}\bar{j}}$
0	3	3	6
0	2	3	5
0	1	3	4
0	2	2	4
0	1	2	3
0	1	1	2
0	0	2	2
0	0	1	1
0	0	0	0

(b) Possible Compressed Metric Sets

Fig. 2.4.3.2 State Metric Transition Ranges

than or equal to 3 but the test for M_{ij} greater or equal to 4 and the subtraction of 4 is more easily accomplished in hardware than similar operations using 3.

With the combination of metric compression and overflow protection, the additions $M_x + R_{ij}$ involve the addition of a 4 bit and a 3 bit number.

2.4.4 Storage of State Metrics. As explained in previous sections, 2^{K-1} state metric values must be stored for each of the possible encoder states. At each bit time, 2^K additions must be performed followed by 2^{K-1} comparisons and select operations. If the decoder is operating at a relatively high bit rate, then all of the operations must be performed in parallel; however, if the data rate is low to moderate, then the arithmetic operations may be performed partly or completely in serial. If the operations must be performed completely in parallel, there is no organization problem in the storage of state metrics since they must all be simultaneously accessible. If, however, the operations may be performed partly or completely in serial, then more efficient organization of the state metric storage becomes a possibility.

As explained in previous sections, each arithmetic operation consists of the addition of the branch metrics R_{ij} and $R_{\bar{i}\bar{j}}$ to the state metrics M_{0x} and M_{1x} resulting in metrics M_{x0} and M_{x1}. Suppose, for example, that completely serial operation is possible. The state metrics can be stored in a random-access memory such that state metric M_{0x} is stored in location 0x. The problem with this organization is that the result of the computation produces

state metric M_{x1}. Now state metric M_{x1} may not yet have been accessed in the present cycle of computations, so the result of the present computation may not yet be stored in this location. Therefore, state metric storage must be duplicated to temporarily store the results of the computations.

A more efficient technique makes use of the following property: if the parity of state 0x is even, then the parity of 1x is odd and vice versa. The state metrics may be stored in two sets of random-access memories, one of which contains the metrics of all states having even parity and the other containing metrics of all states having odd parity. For example, if x has even parity, then 0x has even parity and 1x has odd parity. In performing a computation, the metrics of states 0x and 1x are read out of the memories, the computation is performed and the resulting metrics x0 and x1 are written into the memories in the locations from which the original metrics were taken. Obviously, the metric of state 0x will be found in a different location in the memory at each computation. However, the progression of locations through which the metric 0x passes can be very simply computed.

The preceding example of metric storage using random-access memories is useful for up to two computations performed in parallel. If, however, more computations must be performed in parallel, random-access memories may no longer be used since the organization of such a random-access memory would become unduly complicated.

If more than two computations must be performed in parallel, then all state metrics must be simultaneously accessible and the arithmetic sections are switched between state metrics by the use of multiplexers. Also, the storage must be duplicated to temporarily store the results of the computations. The requirement for multiplexers and duplicated storage tends to offset the savings achieved by sharing a small number of arithmetic units.

2.4.5 Arithmetic Logical Section. The principle functions of the arithmetic-logical section are:

1. to implement the decision logic of equations (2.4.1), generating a set of decisions D_j,
2. to generate updated state metrics and to transfer the new metrics into the state metric registers following a clock,
3. to detect an increase in the minimum state metric above the value 3 and to subtract 4 from all state metrics following such a detection in accordance with the overflow management strategy,
4. to set the values of the output decision gating variable x_j.

An x_j is set equal to one only if its corresponding state metric is less than or equal to 3. (This function will be explained in the next section.)

These functions are implemented in the ACS (add-compare-select) functional block depicted in Fig. 2.4.5. Two ACS's are required for each pair of states if all computations are to be

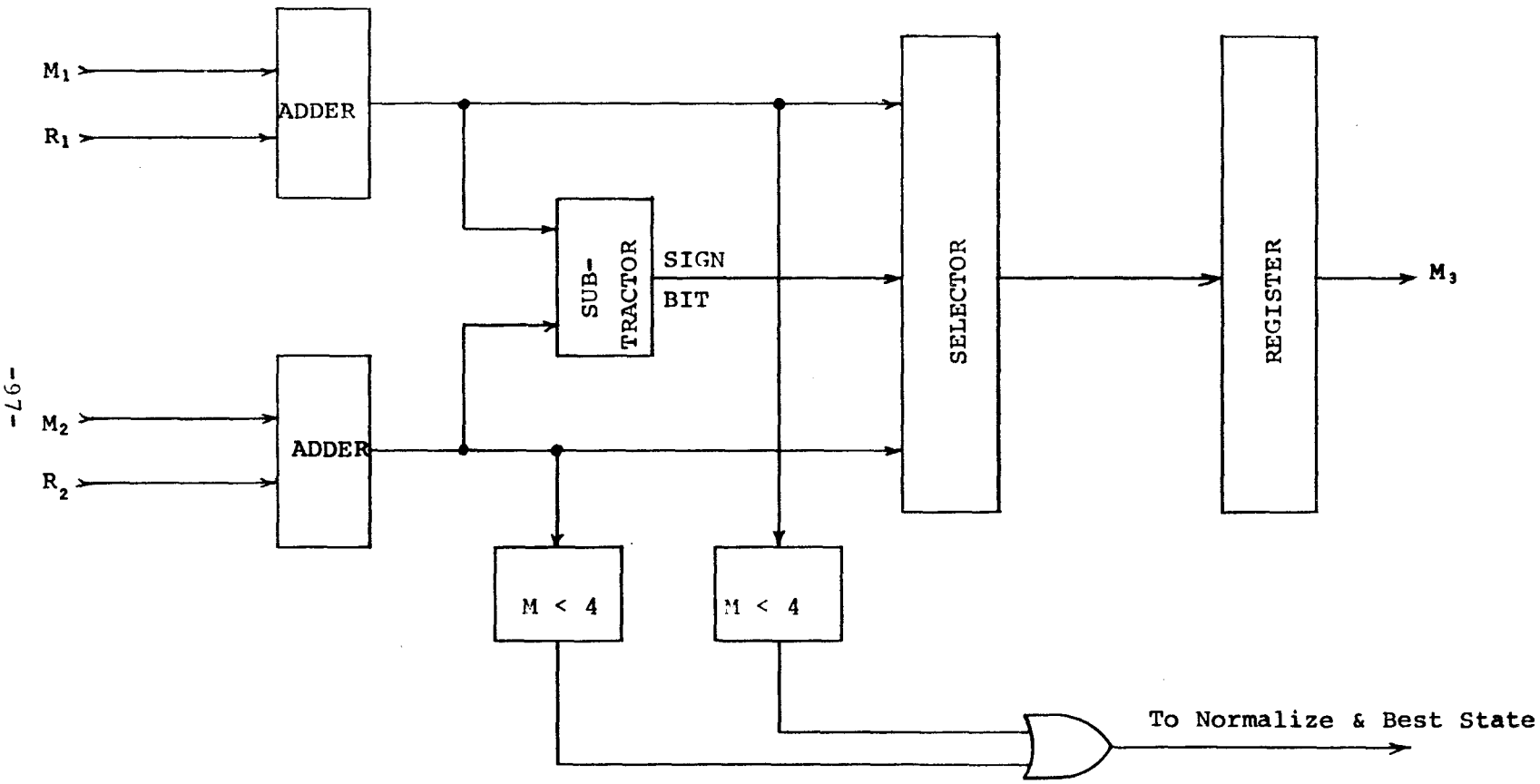


Fig. 2.4.5 VITERBI DECODER

ACS & METRIC REGISTER

carried out in parallel. Since the ACS block is the critical block both in terms of number of components and maximum speed capability, complete designs for this block have been worked out in both ECL logic and TTL logic using MSI. The ECL design can be constructed in three different versions. One version, using MECL III throughout, will obtain the maximum possible speed. The other two designs use MECL II½ and MECL II for lower speed and greater economy. Several different TTL designs were worked out but the one presented here was the best; both in terms of number of parts, cost, and speed so this will be the only one presented. Both designs are specialized to K=4 in that a 4 bit state metric and 3 bit branch metric are assumed.

Normalization can be obtained in two ways: 1) by determining that all state metrics exceed 4 and subtracting 4 from all of them during the same computation, or 2) determining that all state metrics exceed 4 and subtracting 4 from the branch metrics for the next computation. The first approach is suitable for small constraint lengths at high speeds where all arithmetic operations are performed in parallel. The second technique must be used at lower speeds and higher constraint lengths where partly or completely serial arithmetic operations are performed. It is clear that the first technique cannot be used in this case since most of the state metrics are computed and restored before it is known whether or not a subtraction of 4 is necessary. At high speeds where fully parallel operation is used and at lower constraint lengths, there is no particular advantage of one

technique over the other except that the first technique will be easier to understand and debug.

2.4.5.1 ECL Arithmetic-Logic Unit. A block diagram of the ECL-ACS unit is shown in Fig. 2.4.5.1, a, b, and c. Sheets a and b show the adders that add the state metrics to the branch metrics. Sheet c shows the comparator and selector. The adder used in this ACS unit is a form of adder known as the carry-save adder in which ripple carries are used between the stages. The designs of the carry circuits are such that there is only one logic delay per carry. Since only three carries must be generated, the total carry propagation delay is then only three logic delays. For an adder this size, there will be no speed advantage to using look-ahead-carry over this ripple carry technique, and furthermore, the ripple carry technique results in a more economical design.

The comparator is implemented by subtracting the two sums from each other. In this application, we are not interested in the actual difference but only in the carry out of the most significant stage of the subtractor. Hence, we need only implement the carry portion of the subtractor. Another interesting feature of this design is that the carries will be rippling through the adders simultaneously with the carries rippling through the comparator, so that the total add and subtract time is not equal to the sum of the add plus the subtract time but rather one logic delay more than the add time.

Another function that must be performed is determining whether the result is less than 4. If none of the results of the

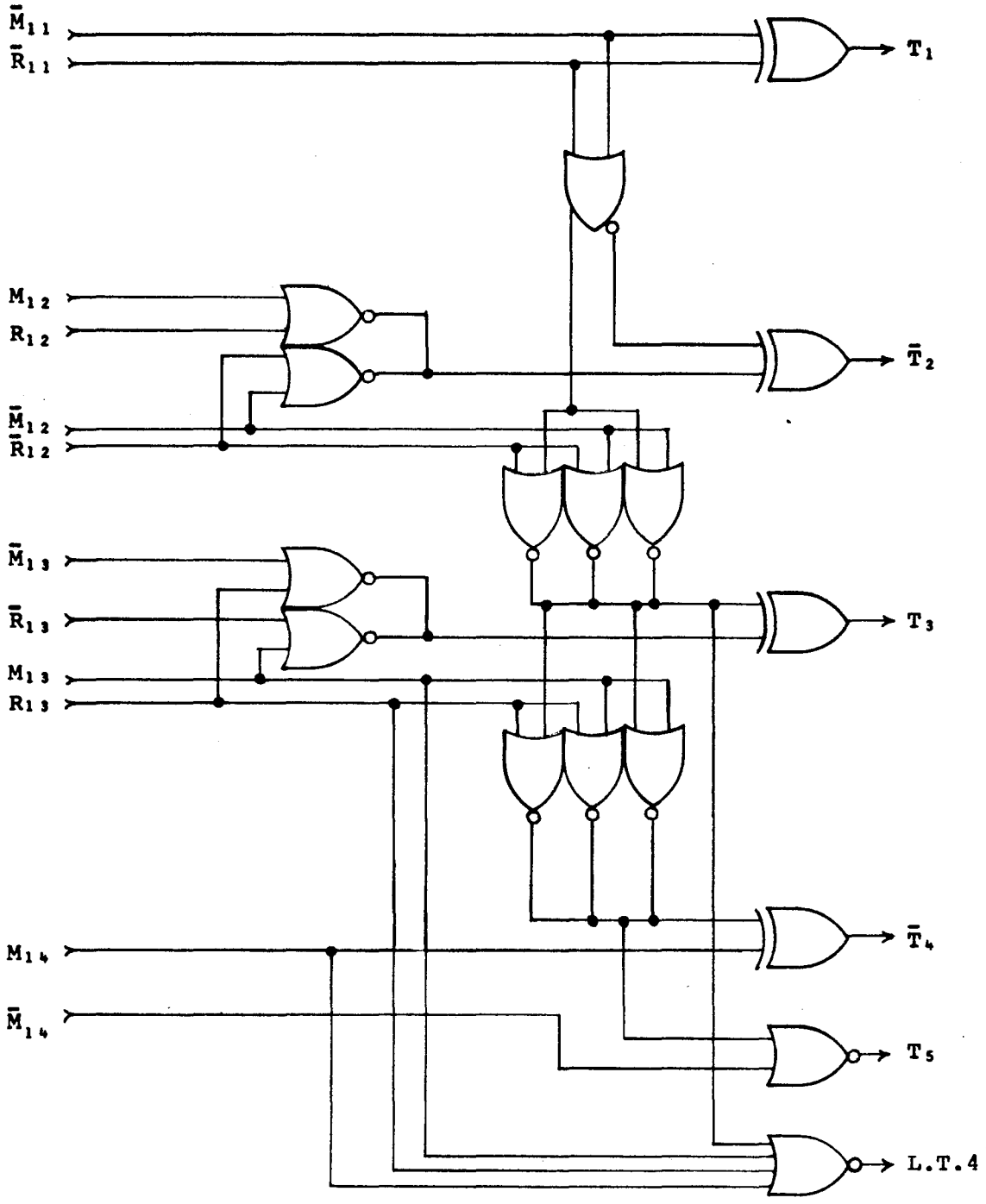


Fig. 2.4.5.1 a VITERBI DECODER
ECL ACS SECTION

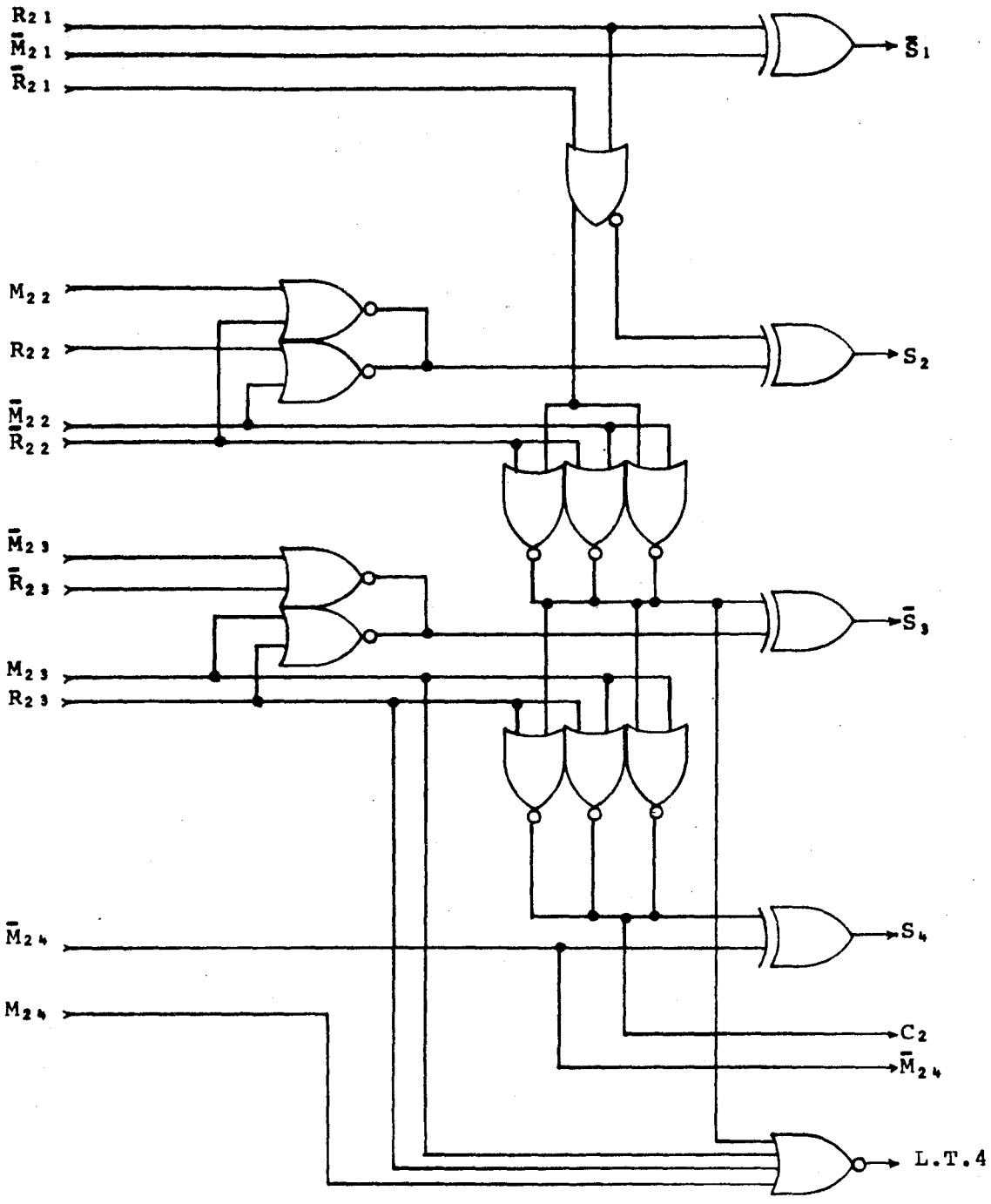


Fig. 2.4.5.1 b VITERBI DECODER
ECL ACS SECTION

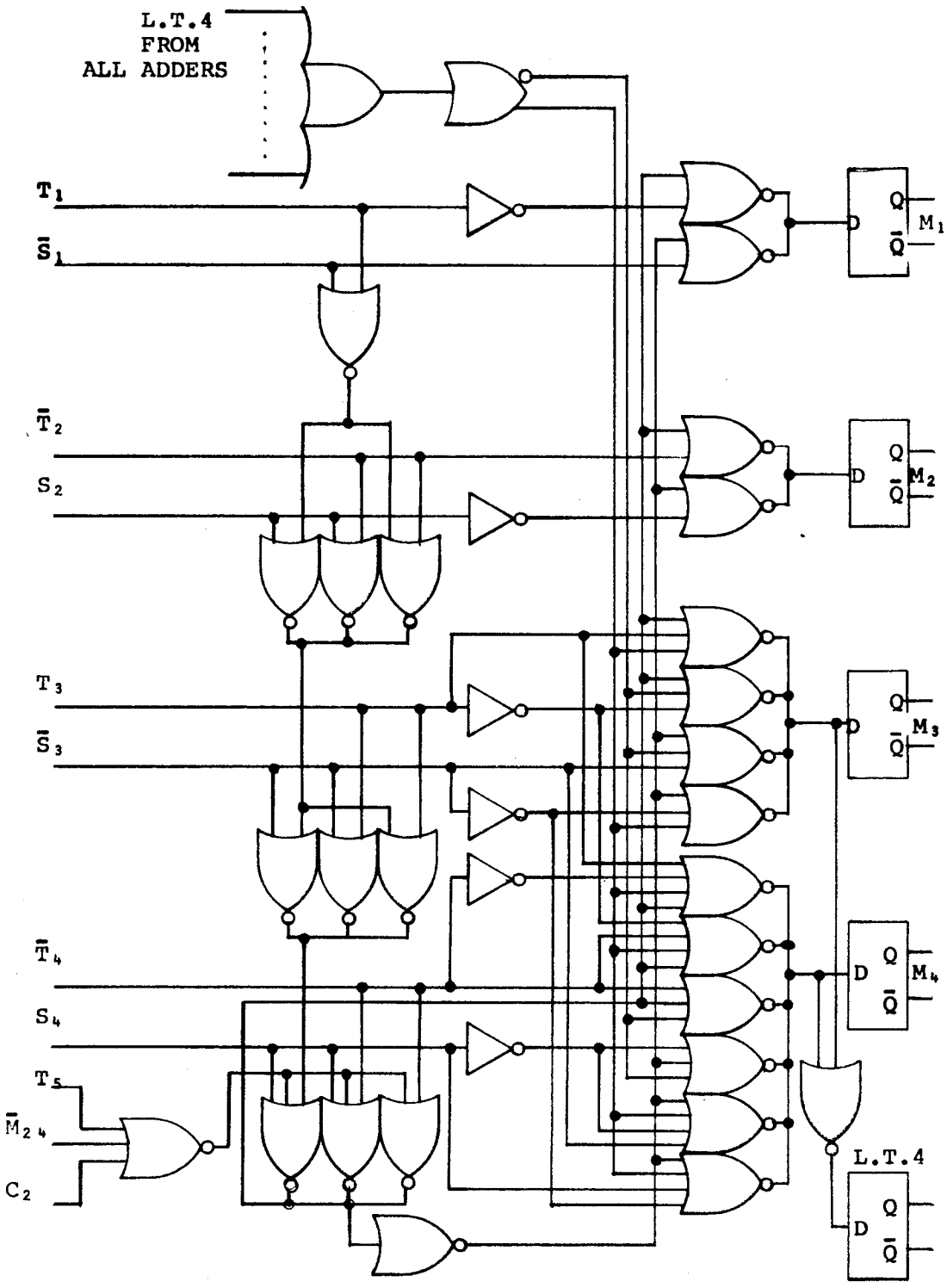


Fig. 2.4.5.1c VITERBI DECODER
ECL ACS SECTION

computations of the ACS unit produces a result less than 4, then 4 must be subtracted from all of the results in order to obtain the required normalization. The result will be less than 4 if either of the two sums is less than 4 since the smaller will be selected. Each of the sums will be less than 4 if the carry out of the second stage, the third and fourth bits of the state metric, and the third bit of the branch metric are all zero. Thus, the less than 4 signal may be obtained in only three logic delays. The less than 4 signals from all of the ACS units are then ORed together. If the result of the ORing operation is zero, then all resulting state metrics will be greater than 4. While this is taking place, 4 is subtracted from each of the sums. The output selector will then select first the smaller of two resulting sums and then select either the smaller or the smaller minus four, depending upon whether normalization is required. The results are then stored in the state metric storage flip flops, thus completing the computation.

Chip counts and typical computation rates have been obtained for this design implemented with three different versions of ECL logic, MECL III, MECL II½, and MECL II. The MECL III design will require 25 chips per ACS including the state metric storage. The typical maximum computation rate is 90 MHz. The MECL II½ design requires 35 chips and will operate at 40 MHz. The MECL II design requires 25 chips and will operate at 25 MHz. The MECL III version will be considerably more expensive than either of the MECL II versions since much more sophisticated techniques are required to

package the MECL III devices. These advanced techniques include the use of strip transmission lines, multilayer circuit boards, and complicated cooling because of the high power dissipation of the MECL III devices. The MECL II and MECL II½ designs may be packaged using approximately the same techniques as required for TTL. The MECL II devices are, however, slightly more expensive on a per chip basis and the level of integration available is somewhat less than that available with TTL, as will be seen in the following example of a TTL design.

2.4.5.2 TTL ACS Unit. A logic diagram of the TTL arithmetic section is shown in Fig. 2.4.5.2. The inputs to the arithmetic section are two state metrics and two branch metrics. The state metrics are each represented as 4 bit binary numbers; the branch metrics are represented as 3 bit binary number plus a sign bit. (Subtracting four from the branch metrics for normalization can result in negative values.) Both the branch metrics and the state metrics are represented in complemented form. The two additions are performed by two SN74181 4-bit arithmetic units operating in the add mode. The SN74181 is a 4-bit full adder with full carry look ahead. The sum is produced, typically, in 24 nanoseconds. If the sum is larger than 15 a carry-out of the 4th stage of addition is produced. A carry-out of the 4th position is also produced when the branch metric is negative. Therefore the 5th bit of the sum is generated by NANDing the carry-out with the sign of the branch metric.

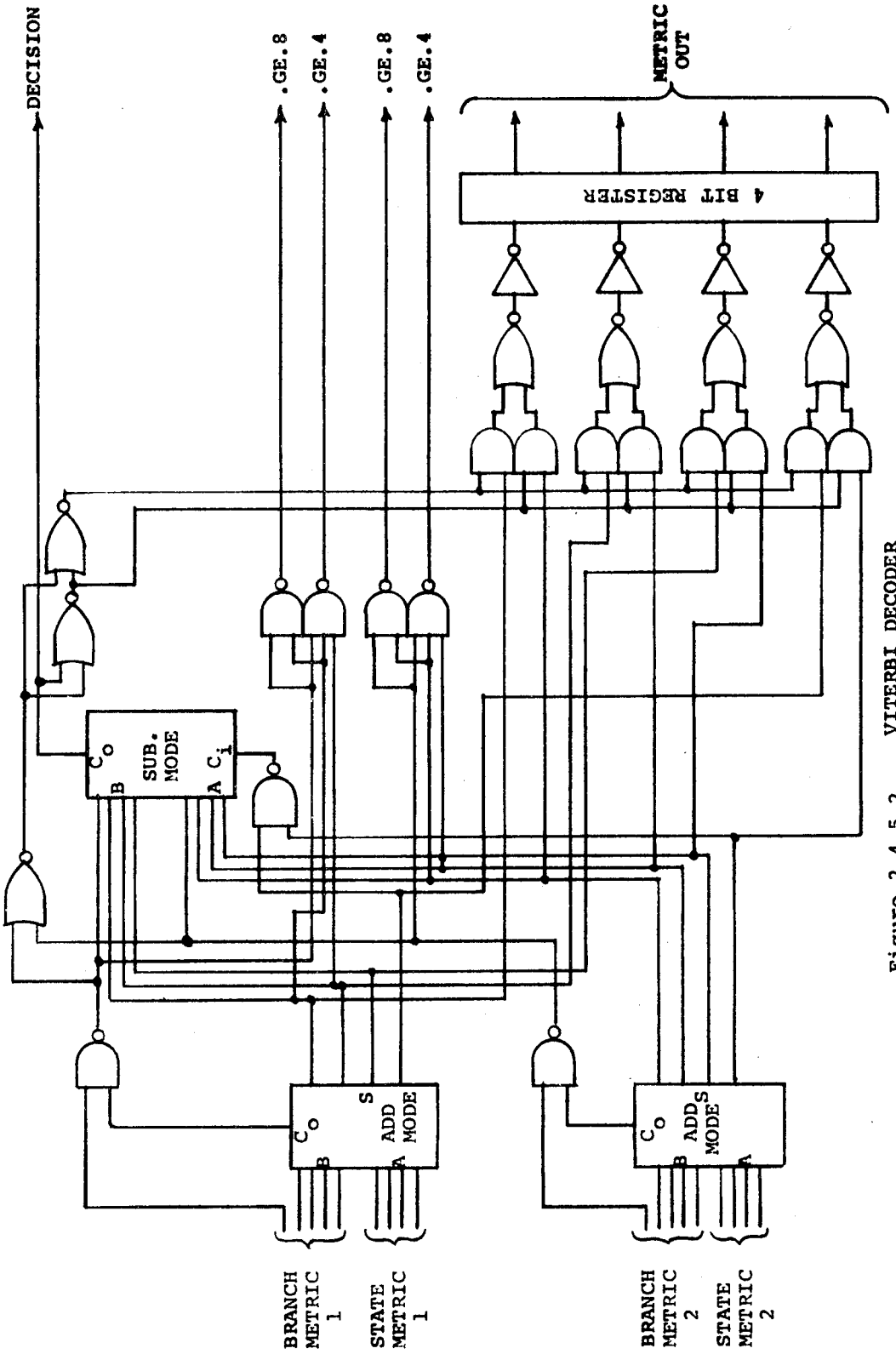


Figure 2.4.5.2 VITERBI DECODER TTL ACS

The smaller of the two resulting sums is determined by subtracting one from the other using an SN74181 in the subtract mode. A carry-out of the subtract will result if sum B is larger than sum A. Note that we must compare two 5 bit numbers and the SN74181 is only a 4 bit subtractor. The 5 bit comparison is performed very simply by subtracting the 4 most significant bits of the two sums in the SN74181. Since we are interested only in the carries in the subtraction process, we need only provide the carry-out of the subtraction of the two least significant bits as input to the carry-in of the 4 bit subtractor. Thus a 5 bit comparison can be accomplished with a single adder chip plus a single two input NAND gate. The resulting decision is stored in the decision memory section.

The smaller of the two sums is selected by the four and/or/invert gates provided by the two SN74H51 chips. The an/or/invert gates are connected as single pole-double throw switches. The result is stored in the state metoric register.

It is also necessary to determine whether the output is greater than or equal to 4, or greater than or equal to 8. The greater than or equal to 4 signal is used to control normalization and to determine the decoder output bit. The greater than or equal to 8 signal is used for the later purpose when all metrics are greater than or equal to 4. Greater than or equal to 4 is determined by examining the three most significant bits of the two sums. If the 3 most significant bits are all equal to 1, then the number is less than 4 (remember that the sums are expressed in

complemented form). Greater than or equal to 8 is determined by examining the two most significant bits. If both are equal to 1, then the result is less than 8. These functions are performed by NAND gates which invert the results, converting less than 4, for example, into greater than or equal to 4. If this logic were performed on the metric output, the time required would be added to the total ACS time. Instead, the greater than 4 and greater than 8 detection is performed on the two sums prior to comparison and selection. The desired result is obtained since, if the smaller of the two sums is less than 4, then at least one of the greater than or equal to 4 outputs is 0.

Each arithmetic (ACS) unit can be implemented with 10 integrated circuits including the metric storage. It should be noted that 3 of these IC's are large (24 pin) and relatively expensive compared to the other IC's. Including the propagation delay of the metric storage units, an arithmetic operation can be performed in under 100 nanoseconds.

2.4.6 Decision Memory and Output Selection. As each check digit pair is input to the decoder, decisions D_{ix} are made by the arithmetic section as explained in previous sections and transmitted to the memory-output section. For each state, 16 bits are stored as shown in Fig. 2.4.6. The first bit stored for state ix is the most recent decision D_{ix}. The remaining bits reflect the results of earlier decisions.

Consider decision D_{0x}. If $D_{0x} = 0$, the decoder has decided that for the most recently received check digits, state x0 is more

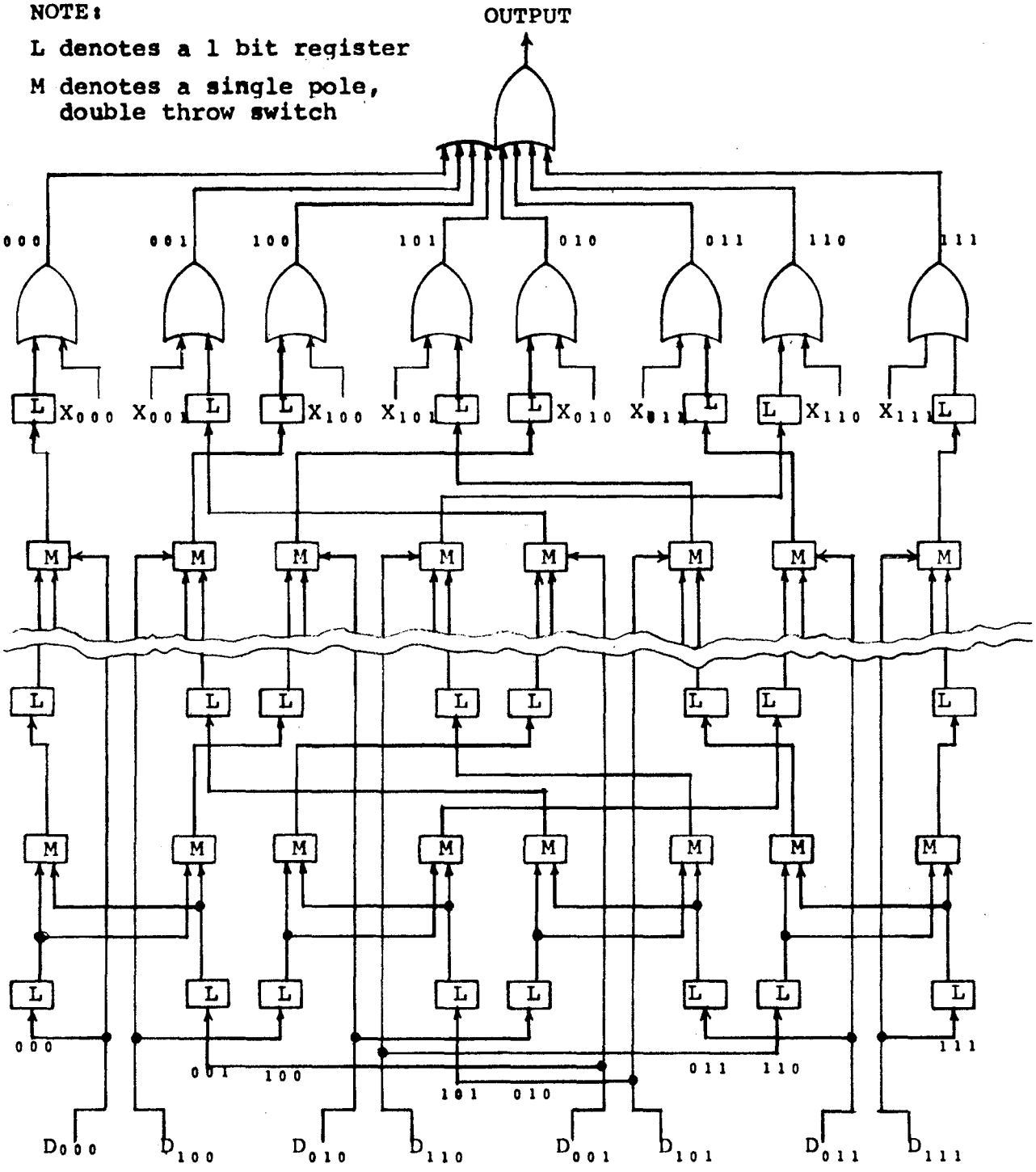


Fig. 2.4.6 DECISION MEMORY AND OUTPUT SECTION

likely than state x1 to have been the predecessor of state 0x. Conversely, if $D_{0x} = 1$, state x1 is a more likely predecessor to 0x than is x0. When $D_{0x} = i$, the 16 bits associated with state xi are shifted right 1 stage and transferred into the stages associated with state 0x. The first stage for state 0x is set equal to i, as shown in the figure. Note that the boxes denoted by L are flip-flops which each store one bit while the boxes labeled M are switches which transfer one of the two inputs to the output depending on the value of the appropriate D.

The bit shifted out of the 16th stage of the memory for each state is either discarded or, for one state, selected as the output from the decoder. The optimum decoder would select as output the bit from the state with the smallest value of M. Simulation results have shown that a memory of 12 bits per state would suffice if the smallest value of M were used. Such a selection is difficult to implement in hardware, since an examination of all eight state metric values is necessary in order to select the smallest.

An alternate approach is to always select the output from, say, the 000 state. This approach, ignoring the state metrics entirely, requires a memory of 24 bits per state to keep the degradation small.

The approach adopted by Linkabit is based on the overflow-management strategy and is almost optimum. The Linkabit decoder memory is conservatively extended to 16 bits to account for the slight nonoptimality. Recall that the overflow-management

strategy forces the smallest state metric to have a value between 0 and 3. Because nearest states have a Hamming distance of 2, the compressed metric separation generally exceeds 4, and in general only 1 state has a metric between 0 and 3. Moreover, if 2 (or more) states had metrics less than or equal to 3, they would tend to have similar past histories and hence the same 16th bit in memory.

The selection mechanism is based on examining each state metric and setting the variables $x_j = 0$ if $M_j \geq 4$ and equal to 1 otherwise, j = 000, 001, ..., 111. As shown in Fig. 2.4.6, the variable x_j is used to gate the output from the 16th stage of memory for state j via an OR gate to the decoder output. If more than one x_j is non-zero, the corresponding bits are ORed together to form the decoder output. This method is nearly optimum and very inexpensive to implement. The slight non-optimality is more than offset by the extension of the decision memories to 16 rather than 12 bits. Examples of the design of the memory output section are given below for both TTL and ECL logic families.

2.4.6.1 ECL Memory Output Section. The following design is based upon the use of the MECL II MC1040 Quad Latch. This circuit contains 4 latches with individual output gates. The circuit can be used to provide both the storage and the switching capability required for the decision memory. One and one third stages of storage per chip are obtainable using this integrated circuit. Each stage is stored in three latches and three different chips

as shown in Figure 2.4.6.1. The three latches are arranged as one master and two slave flip-flops. The output gating on the slave flip-flops is used to perform the switching operations. The total number of bits stored for K=4 is 8 x 16 = 128. At 4/3 bits per chip, this requires 96 chips. The output gating requires an additional three chips for a total of 99. This MECL II quad latch circuit can be used at speeds up to 40 MHz.

For higher speeds, MECL III must be used. Motorola is planning the introduction in the near future of a MECL III circuit equivalent to the MECL II MC1040. When this circuit is available, the memory output section for speeds up to 90 Megabits can be implemented in the same way as in the MECL II case. Until this chip becomes available, the decision memory can be constructed using one MC1670 flip-flop preceded by two 2-input NOR gates to accomplish the switching for each stage. One and one half chips are required per stage of storage. For K=4, this requires 192 chips.

2.4.6.2 TTL Memory Output Section. There is only one TTL MSI circuit that can be easily used to form the memory output section, the SN74L98, which consists of four storage registers, each of which is preceded by a two way switch. Using this circuit, 1/4 IC would be required per stage of storage. In the case of K=4, this would require 32 IC's. This is a low power, low speed IC with a maximum clock rate of only 3MHz. For higher speeds, a more brute force approach is necessary. For speeds up to 10 Megabits, the storage register can be obtained by using the

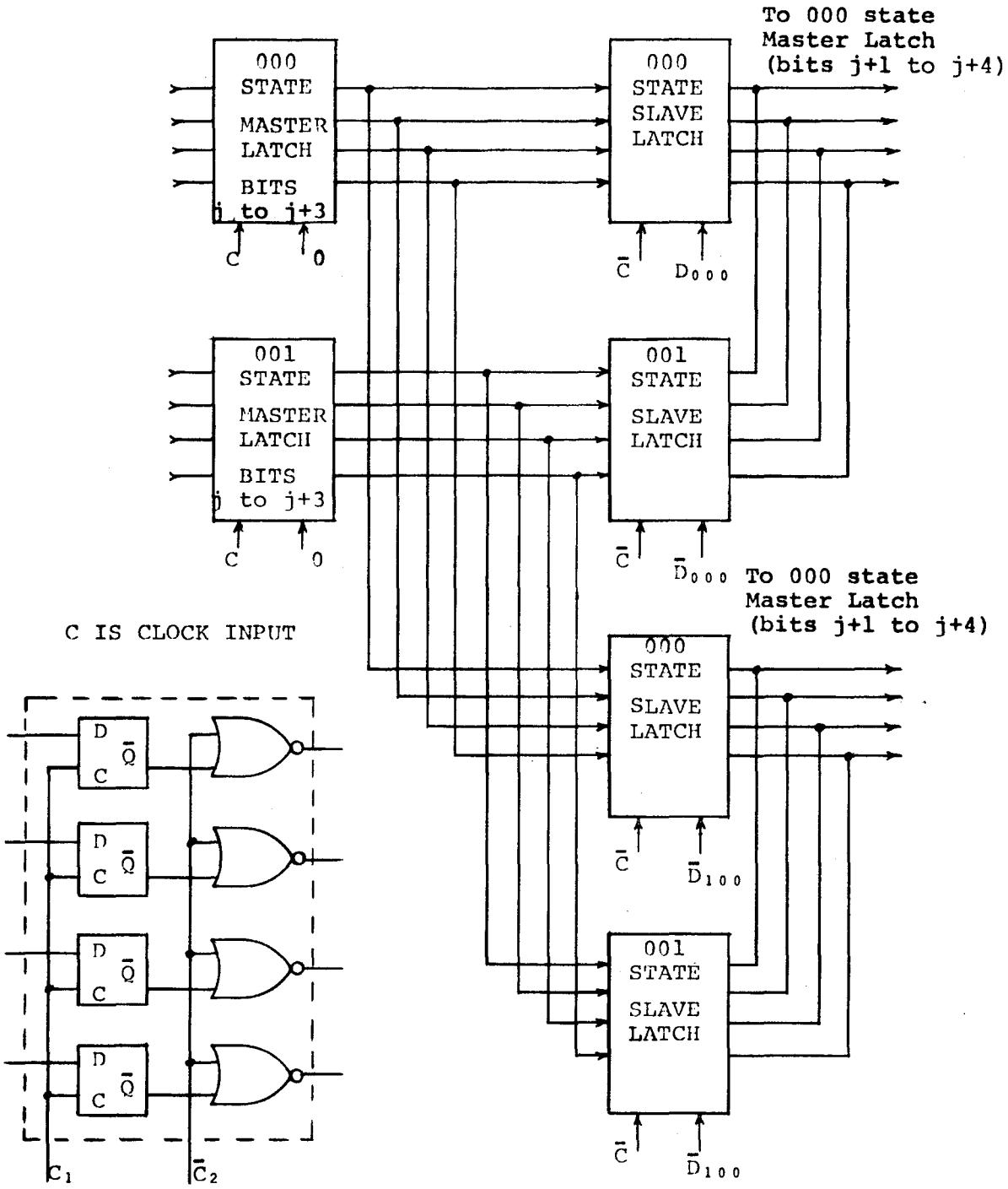


Fig. 2.4.6.1 - Four stages of 000 and 001 state memories using quad latches. Output gating on latches provides mechanism for selective memory transfer and shift.

SN7495 which contains 4 storage registers, together with a quad 2-input multiplexer chip. Thus, 1/2 IC would be required per bit of storage for a total of 64 chips in the case of K=4.

At lower data rates, it is possible to make use of a relative speed factor to decrease the number of components required in the decision memory. For example, at data rates below 500 Kilobits, the memory could be constructed using 16 SN7491 devices which are 8 bit shift registers. They would be connected as 8 16 bit shift registers. Since a relative speed factor is available, the register transfers can be accomplished by serially shifting the contents of the 8 registers through 8 2-input multiplexers. Thus, the memory could be implemented using only 18 IC's. At lower data rates, two quad 16 bit MOS static shift registers could be used to form the memory together with 2 chips for the switching, yielding a total of only 4 IC's for the whole memory section.

2.4.7 Synchronization Section. The purpose of the sync section is to obtain node synchronization. This is the only synchronization required by a transparent code Viterbi decoder. If the code used were nontransparent, then the sync section could also resolve the 180° phase ambiguity of the PSK of QPSK demod.

The sync circuit operates by comparing the rate of increase of the best state metric with its expected value. If the rate is too high then it is assumed that a bad node sync state exists and the node sync state is changed. This is accomplished, in the case of PSK, by either inserting or deleting a one symbol time

delay in the input section. If the input is from a QPSK modem then the sync state is changed by interchanging the received symbols r_1 and r_2 and inverting r_1.

A logic diagram of the sync circuit is shown in Fig. 2.4.7. As can be seen from the diagram, the LINKABIT synchronization technique is extremely simple. This, of course, demonstrates one of the inherent advantages of convolutional codes over block codes. The circuit operates as follows: Every time a metric normalization occurs, an up/down counter is counted down by one count. The rate at which this occurs is proportional to the rate of increase of the best state metric since normalization does not occur unless the metric value of all states exceeds 3. The actual rate is slightly greater than the hard decision error rate divided by four when the node sync state is correct. When the node sync state is incorrect, the normalization rate is much higher, approximately the bit rate divided by eight.

The up/down counter is counted up at the bit rate divided by a constant. The optimum choice for this constant is approximately 16. The up/down counter is not permitted to overflow, i.e., when the counter reaches all ones, the up count input is gated off. If the counter underflows, then the node sync state is changed. When the node sync state is correct, the average drift of the counter will be up and the counter will spend most of its time in the all ones state with occasional short downward excursions. When the node sync state is incorrect, the average drift of the counter is down and the counter will soon underflow, thus changing the node sync state. Obviously, there is a tradeoff

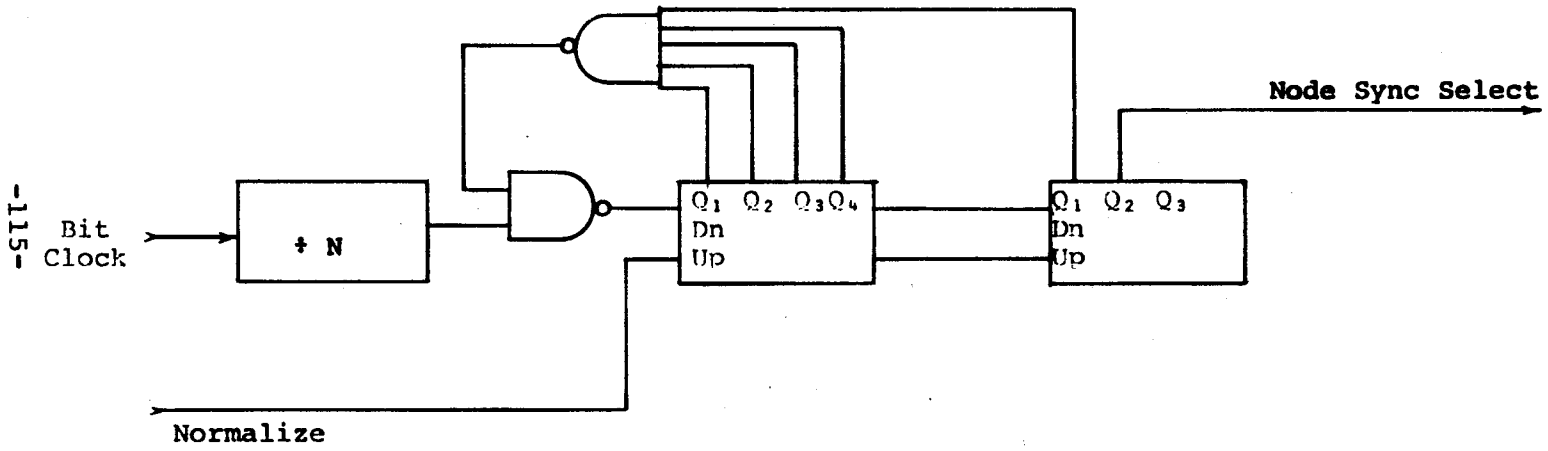


Figure 2.4.7

LOGIC DIAGRAM-SYNC CIRCUIT

involved in the size of the up/down counter. The larger the counter, the longer is the time required for synchronization, whereas, the shorter the counter, the larger the probability of false loss of node sync. The optimum size of the counter has not been determined exactly but is known to be between four and six stages.

The node sync state is actually changed by allowing the borrow-out of the most significant counter stage to toggle a flip-flop. The state of the flip-flop is the node sync state. Thus, when the borrow-out of the up/down counter occurs on counter underflow, the node sync state is changed. The node sync state is sent to the input section where it controls the addition or deletion of the one symbol time delay in PSK. In QPSK, the node sync state controls the interchanging of received symbols r_1 and r_2.

The up/down counter is implemented in TTL by using two SN74191 four stage up/down counter chips. If the counter is to be 6 stages long, then the first six stages of the two cascaded dividers form the required counter. The seventh stage becomes the node sync state flip-flop. Overflow is inhibited by ANDing together the first six stages of the counter. When the first six stages of the counter are all ones, further up counts are inhibited. A MECL circuit can be implemented in a similar manner.

2.4.8 Trade-Off Section.

2.4.8.1 Cost-Complexity Trade-Offs. This section presents the results of the cost-complexity tradeoff study for the Viterbi decoder, based on the K=4, rate 1/2, Q=7 designs presented in previous sections. Parts counts have been tabulated for four different designs; a 10 megabit TTL decoder, a 25 megabit MECL II decoder, a 40 megabit MECL II½, and a 90 megabit MECL III decoder. Weighting factors have been assigned to the different logic families to reflect differences in parts cost, design cost, and packaging cost. The results are presented in the table below. The weighting factors used were; 1 for TTL , 1.5 for MECL II, 2.5 for MECL II½, and 7 for MECL III. It should be pointed out that these weighting factors are based on a number of subjective judgements on the part of the author, and should not be considered to be exact or invariant. Any number of things could cause these relationships to change in time, for example, a continuation of the present TTL price war, new MSI circuit announcements, etc.

Logic Family	Max Data Rate (Mbps)	No. of IC's	Relative Cost
TTL MSI	10	185	1
MECL II	25	365	3
MECL II½	40	450	6
MECL III	90	470	18

The best way to build a 40 megabit Viterbi decoder will now be considered. With the designs that have been worked out in the previous sections there are four possibilities. The decoder could be built by using MECL III logic, or by using MECL II½ logic, or by building two 25 megabit MECL II decoders operating in parallel, or by building four 10 megabit TTL decoders operating in parallel.

The last two examples require that an overhead factor of approximately 10% be used to account for the cost of the extra encoders and the cost of tying the decoders together. The MECL III decoder would have a relative cost of 18, the MECL II½ a relative cost of 6, the MECL II decoder a relative cost of 6.6, and the TTL decoder a relative cost of 4.4. Thus, it appears that the TTL design would be the most inexpensive way to obtain a 40 megabit Viterbi decoder. Clearly, the MECL III design is not desirable from a cost standpoint. The TTL design appears to be superior over the whole range from 10 to 100 Megabits, however, a change in the weighting factors used could alter this conclusion. An additional advantage to the paralleling approach to obtain high speeds is that it is very easy to include provisions for fault isolation and maintenance. For example, suppose that a 40 megabit decoder were desired. This could be provided by building four TTL 10 megabit decoders. If a fifth decoder were provided, then fault isolation could be obtained by switching the spare decoder in parallel with each of the other decoders, and comparing the output. When a discrepancy is found, the fault isolation

circuitry would then automatically determine which of the five decoders is defective and automatically switch the spare decoder into its place. The maintenance of the faulty decoder would then take place while the remaining four decoders continue to operate on line.

In this section we have determined the relationship between data rate and cost. In the following sections we will determine the relationship between cost and other decoding parameters such as code rate, constraint length, and quantization level.

2.4.8.2 Cost Vs. Constraint Length. In the previous section we have shown that the relationship between cost and data rate is approximately linear. The relationship between cost and constraint length is, however, exponential. As the constraint length increases by one, the number of states required doubles. This will require doubling the total number of arithmetic units and more than doubling the decision memory. The decision memory will double in one dimension and increase linearly in the other dimension. Also the number of bits required to represent the state metrics will increase as the constraint length increases, since the distance of the code increases, thereby increasing the spread between the best and worst state metrics. Overall the decoder complexity relative to the K=4 design goes approximately as

$$\frac{K \cdot 2^{(K-1)}}{4 \cdot 2^3}$$

The cost relative to the K=4 decoder design will be then $K \cdot 2^{K-6}$. For example, a K=6 decoder will be approximately 6 times as complex and therefore 6 times as expensive as a K=4 decoder.

2.4.8.3 Cost vs. Code Rate. Changing the denominator of the code rate has its primary effect on the input section of a Viterbi decoder. In general there are 2^d branch metrics to be computed where d is the denominator of the rate. Thus the input section grows exponentially with the denominator of the rate. The size of the largest branch metric grows linearly with d. This causes an additional increase in complexity as a function of $\ln_2 d$. This occurs both in the input section and the arithmetic section since larger branch metrics require larger state metrics. The complexity multiplying factor relative to rate 1/2 is approximately

$$\frac{46 + 2^d \ln_2 d}{50}$$

The numerator of the rate affects decoder complexity in a rather complicated way. The number of states is an exponentially decreasing function of the numerator of the rate, thus decreasing the state metric storage requirement. However, the complexity of the arithmetic operations increases exponentially with the numerator of the rate. Thus, for rate 2/4, we do twice as complex an arithmetic operation on half as many quantities as we do for rate 1/2. The net result is no change in arithmetic hardware for a fully parallel decoder. The decision memory decreases exponentially with increasing numerator. The overall relative

complexity multiplying factor is approximately $.6 + .8 \times 2^{-n}$ where n is the numerator of the rate. This factor is relative to the rate 1/2, Q=8, K=4, TTL decoder.

2.4.8.4 Cost vs. Quantization. The number of quantizer levels is linearly related to the size of the branch and state metrics. Thus, the complexity is a function of $\ln_2 Q$. The complexity relative to the Q=8 design is given by

$$\frac{3 + \ln_2 Q}{6}$$

REFERENCES

SECTION 2

1. R. McEliece, and H. C. Rumsey, "Capabilities of Convolutional Codes," Jet Propulsion Laboratory, SPS 37-50, Vol. III, 1968.

2. J. A. Heller, "Short Constraint Length Convolutional Codes," Jet Propulsion Laboratory, SPS 37-54, Vol. III, 1968.

3. J. P. Odenwalder, Optimal Decoding of Convolutional Codes, Ph.D. Thesis, System Science Department, UCLA, Los Angeles, 1970.

4. J. M. Wozencraft and I. M. Jacobs, Principles of Communication Engineering, Wiley, New York, 1965.

5. A. J. Viterbi, "The State-Diagram Approach to Optimal Decoding and Performance Analysis for Memoryless Channels," Jet Propulsion Laboratory SPS 37-58, Vol. III, 1969.

6. A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," LINKABIT CORPORATION, January 1970.

7. W. W. Petersion, Error Correcting Codes, M.I.T. Press, Massachusetts - Wiley, New York, 1961.

8. A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Transactions on Information Theory, Vol. IT-13, Number 2, April 1967.

9. J. L. Massey and M. K. Sain, "Inverses of Linear Sequential Circuits," IEEE Transactions on Computers, Vol. C-17, April 1968.

10. LINKABIT Corporation, "Unsolicited Proposal - A Very High Speed Viterbi Decoder for Convolutional Codes," Submitted to the U.S. Army Satellite Communications Agency, Fort Monmouth, N.J., September 1969.

3.0 SEQUENTIAL DECODING

3.1 Background. Sequential decoding is a procedure for systematically searching through a code tree, using received information as a guide, with the objective of eventually tracing out the path representing the actually transmitted information sequence.

Most sequential decoder implementations to data have used some modification of the Fano algorithm. Briefly, the operation of the Fano algorithm is as follows. Starting at the first node in the code tree, a path is traced through the tree by moving ahead one node at a time. At each node encountered, the decoder evaluates a branch metric for each branch stemming from that node. The branch metric is a function of the transition probabilities between the received symbols and the transmitted symbols along the hypothesized branch.

The decoder will initially choose the branch with the largest metric value (corresponding to the closest fit to the received symbols). The metric is then added to a path metric, which is the running sum of branch metrics along the path presently being followed. Along with the path metric, the decoder keeps track of the running threshold T. As long as the path metric keeps increasing, the decoder assumes it is on the right track and keeps moving forward, raising T to lie within a fixed constant, Δ, below the path metric. If, on the other hand, the path metric decreases at a particular node, such that it becomes less than T, the decoder

assumes it may have made a mistake and backs up. It will then systematically search nodes at which the path metric is greater than T until it finds a path that starts increasing again, or until it exhausts all nodes lying above T. At this point it is forced to lower T, and search again. Eventually it will find a path that appears to have an increasing path metric.

Eventually, the decoder will penetrate sufficiently deep into the tree, that with high probability the first few branches followed are correct, and will not be returned to by the decoder in a backward search. At this point, the information bits corresponding to these branches can be considered decoded and the decoder may erase received data pertaining to these branches.

A major problem with sequential decoding is the variability in the number of computations required per information digit decoded. The number of computations is a measure of the time required to decode, for a fixed decoding speed in computations per second. A computation is defined, for the time being, as either looking forward or backward one branch and evaluating and testing the metric involved. The cumulative distribution of computations performed per digit decoded, c, has been upper and lower bounded for the discrete memoryless channel by a Pareto distribution (Ref. 1, 2), that is

$$\Pr[c>L] \sim k\, L^{-\alpha}, \qquad L>>1, \tag{3.1.1}$$

where k is a constant and α is determined by the relationship

$$R = \frac{E_o(\alpha)}{\alpha} \qquad (3.1.2)$$

where R is the code rate.

Here $E_o(\alpha)$ is a convex function of α which is determined by the channel transition probabilities, which are in turn a function of E_b/N_0. This function has the properties that $E_o(0) = 0$, and $E_o(1) = R_{comp}$. Therefore, we can see from Eq. (3.1.2) that if $R=R_{comp}$, $\alpha=1$. R_{comp} is the so called computational cutoff rate of sequential decoding.

Because $\alpha>1$ for $R<R_{comp}$, the average number of computations per node decoded is finite, but for rates greater than R_o, this average is unbounded. Actually, for finite constraint lengths, the computation distribution drops off faster than Pareto for very large L. Thus, the average computation remains finite but large for $R>R_o$.

Because of the variability of the amount of computation required, there is a non-zero probability that incoming received data will fill up the decoder memory faster than old outgoing data can be processed. If the decoder tries to search a node for which received data has passed out of buffer memory, an overflow is said to occur. When an overflow occurs, the decoder must have some mechanism for moving forward to new data, reacquiring code synchronization and starting to decode again. There are presently two techniques for doing this. One involves segmenting

the data into blocks. After each block, a fixed constraint length long sequence is inserted. Should the decoder buffer overflow while decoding a given block, it can simply give up decoding that block and jump to the beginning of the next block to resume decoding. Code sync is immediately attained through knowledge of the fixed data sequence preceeding a block. This technique has the disadvantage that it requires an initial search to acquire block sync, and there is a loss in efficiency due to the insertion of known sync bits into the data stream.

Another overflow recovery technique does away with data blocking (Ref. 3). When an overflow occurs, the decoder jumps ahead to new data, and guesses the coder state at that point based upon received data. This technique is described in detail in a subsequent section.

The probability of overflow for sequential decoding can be related to the distribution of computations per bit only in an approximate manner. Suppose the decoder has a speed factor of μ, that is, it is able to perform μ computations per branch worth of data received. Suppose also, a decoder buffer capable of storing B branches worth of received data is used. With an initially empty buffer, the decoder may perform μB computations in progressing one bit deeper before an overflow occurs. Thus, from Eq. (3.1.1), the initial overflow probability is

$$P_o = k(\mu B)^{-\alpha} \qquad (3.1.3)$$

Since overflows can occur through the concatenation of several shorter searches, one intuitively expects that the actual overflow probability would be larger than (3.1.3). However, as long as μ is somewhat larger than the average number of computations per bit, simulations have shown (3.1.3) to be remarkably accurate. Of course, when an overflow does occur, many bits will be lost, whatever the restarting method. Thus, the rate of bits lost due to overflow will be

$$P_{oB} = LP_o \tag{3.1.4}$$

When μ is close to the average computations per bit, as is the case in a high data rate sequential decoder, simulation is the only reliable means of determining overflow frequency.

The error probability with sequential decoding can be made as small as desired by increasing code constraint length. Long constraint lengths are practical for sequential decoding because decoder complexity is only a weak function of code length, unlike Viterbi decoding.

It has been shown (Ref. 4) that for systematic codes, the undetectable error probability can be upper bounded by

$$P(e) < k' \, 2^{-K(1-R)R_{comp}/R} \tag{3.1.5}$$

for $R<R_{comp}$, where K is the code constraint length. The actual achievement of this rate of decrease in P(e) with K is dependent on the choice of branch metrics for the decoder. This will be

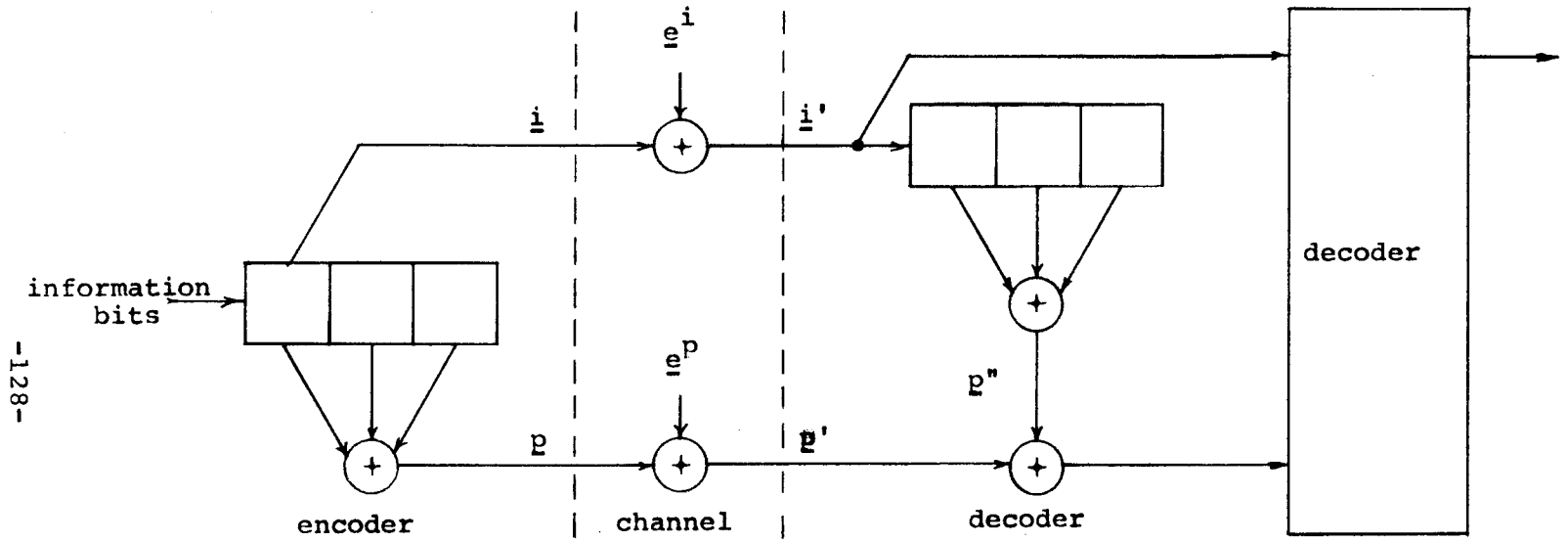


Fig. 3.2.1 Communication System
With Syndrome Decoder

discussed in section 3.2.

3.2 Hard Decision Decoder. This section will deal in part with the interesting Fano algorithm modifications that offer potential advantages for a high speed hard decision sequential decoder. Simulations will be reported for decoders using these modifications. Non-real time simulations have provided information on distributions of computations and undetected error rate. Real time simulations, using simulated decoder speed factors and buffer sizes, determine the overflow behavior of the decoder.

Also considered are code synchronization, code selection, and data quality information.

3.2.1 Syndrome Sequential Decoder. We will restrict our attention to systematic rate 1/2 convolutional codes with hard receiver quantization, for the sake of example. The technique generalizes easily to non-systematic codes, other rates, and even soft decisions.

For a rate 1/2 systematic code, a received information bit and a received parity bit are input to the decoder at each bit time. To form the code syndrome, the received information bits are passed through a replica of the encoder and the generated parity bits are exclusive-ORed with the received parity bits. Fig. 3.2.1 shows a representation of the encoder, channel, and syndrome calculator for a K=3 code. The noisy channel is modelled by the mod-2 addition of occasional errors (ones) to the encoded information and parity streams.

Clearly, in the absence of noise, the syndrome bits input to the decoder are all zero regardless of the information sequence. This is because the parity bits generated in the syndrome calculator, $\underline{p}''$, and the received parity bits, $\underline{p}'$, are both equal to the actual parity bits, $\underline{p}$. Thus, since the code and the channel action are linear, the syndrome is a function only of the noise sequences. We can assume, therefore, without loss of generality, that the all zeros code sequence is transmitted.

This being the case, note that a single error in the information stream manifests itself in Fig. 3.2.1 as three consecutive 1's in the syndrome. In general, an information error causes the code generator to be exclusive-ORed into the syndrome. Each parity error, on the other hand, causes a single 1 to be exclusive-ORed into the syndrome.

It can be shown that putting the received data into the form of a syndrome is information lossless. A decoder operating on $\underline{s}$ can perform as well as one operating on $\underline{i}'$ and $\underline{p}'$. The function of a decoder operating on a syndrome sequence is to determine the most likely information and parity error sequences that could have resulted in that particular syndrome sequence. For a binary symmetric channel, this corresponds to determining the minimum weight error sequence consistent with the syndrome. The decoder forces the syndrome sequence to zero, by exclusive-ORing a "1" where it believes a parity error occurred, and the code generator where it believes an information error occurred.

A syndrome sequential decoder keeps track of a metric as it "zeros" the syndrome. Each time it hypothesizes the occurrence of an error, the metric decreases. When it hypothesizes no error, the metric increases. If the decoder finds it has to correct too many errors in forcing the syndrome to zero, it will back up and change hypothesized information error decisions. Note that each information error decision affects the syndrome over a full constraint length.

Functionally, the syndrome decoder can be viewed as a box whose input is a syndrome sequence, and whose eventual output is an information error location sequence. This sequence is then used to correct errors in the received information sequence to form the decoder output.

3.2.2 Algorithm Modifications

3.2.2.1 Guess and Restart Overflow Strategy. The guess and restart technique was developed and successfully implemented in a sequential decoder previously (Ref. 3). When a buffer overflow occurs in a sequential decoder due to a long search, the decoder must jump forward in the syndrome stream and resume decoding. If the data is not blocked, the decoder does not have definite knowledge of the coder state when it starts decoding again. What is required is knowledge of one constraint length of information bits at the point decoding is resumed.

The best "guess" that can be made is that no information errors have occurred in the vicinity of the restart point. This "guess"

can be implemented by assuming a zero syndrome when restarting. If information errors actually occurred, the decoder will likely overflow again, requiring another restart. If the guess was correct--an if the succeeding data is not too noisy--the decoder will work its way through the buffer and resume normal operation.

The decoder skips over a segment of the syndrome in restarting after an overflow. The information error decisions are set equal to zero over this segment. Thus when an overflow occurs, the decoder output corresponding to the unprocessed data will be the raw, uncorrected received information bits. These bits have errors occurring at the channel error rate.

Simulations using guess and restart are presented in section 3.2.4.

3.2.2.2 Quick Threshold Loosening. A sequential decoder operating below R_{comp} (E_b/N_0 greater than 4.6 db in the case of a rate 1/2 code with hard quantization) spends much of its time plowing forward on the correct path, or in short searches, correcting single or double errors. Long searches contribute less and less to the average computation per bit as E_b/N_0 goes up. Reduction of the time consumed just "plowing forward" or "keeping up with the data" is treated in section 5.0 of this report. Here we are concerned with reducing the number of short searches required. The technique of quick threshold loosening achieves this end for a hard quantized, rate 1/2 decoder.

A single error in the received data stream is usually sufficient to cause the path metric to fall below threshold and initiate a backwards search. In general, the decoder must search backward

at this point, rather than lowering the threshold and resuming the forward search. This is because, in backing up, the decoder may find another path that it can follow without lowering the threshold. Had the decoder lowered the threshold when the metric first violated it, the possibility exists of getting on a path for which the algorithm will not allow further threshold tightening. Thus, the threshold cannot be lowered unless all accessable paths above the threshold eventually lead below it.

For a rate 1/2 hard quantized decoder there are three possible branch metrics. The first corresponds to two matches between branch code symbols and received data. The second is for one match and one mismatch, and the third for two mismatches. Furthermore, good codes always have the property that branches stemming from the same node have complementary code symbols on them. Suppose the symbol match metric is equal to 1, and the mismatch metric is -a (the optimum range for a is from about 9 to 11). This means that either one branch leaving a node has metric +2 and the other -2a, or they both have metric 1-a. Fig. 3.2.2.1 shows the way in which a typical short search is initiated due to a single error.

When node A is reached for the first time, the threshold is tightened to the value T_1. The decoder then proceeds on to node B, C and D. Looking forward from D the decoder sees a tie vote (branch metrics of 1-a on both branches). Both path metrics at nodes F and E are below T_1. This would normally initiate a back-search thru nodes C, B and A, requiring tests of metrics at nodes

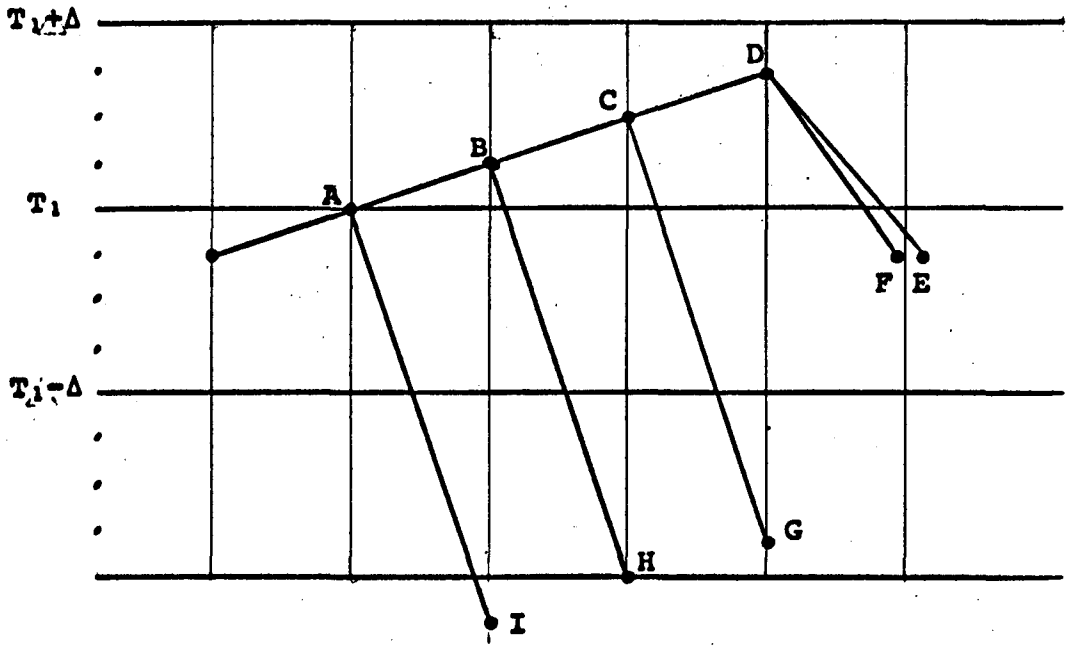


Fig. 3.2.2.1 Typical Sequential Decoder Search Due to a Single Error.

G, H and I. Then at node A the threshold would be lowered to $T_1-\Delta$ and the decoder would step forward again thru B, C, D and then E or F. In this situation the decoder could have avoided the backsearch by lowering the threshold to $T_1-\Delta$ when threshold T_1 was first violated in looking forward from node D. This can be done because there is no other path remaining above T_1 which can be followed. This "instant" threshold lowering can be done if $\Delta \leq a-1$. This restriction insures that a threshold violation on the path being followed determines that no other path remains above threshold. Quick threshold loosening is allowed only when the threshold has been previously tightened to its present value (it is tightened to T_1 at node A in Fig. 3.2.2.1). If the threshold has been previously loosened all bets are off, since now a backsearch after threshold violation is necessary to insure non-existence of another path that remains above the present threshold.

The quick threshold lowering scheme is easy to hardware implement.* It only requires a one bit flag indicating whether or not the threshold had been previously tightened, and logic to prevent entering backsearch mode when a "quick threshold loosening" is possible.

A simple modification of the "quick threshold loosening" scheme allows decoding past isolated pairs of error without initiating a backsearch to lower the threshold.

Fig. 3.2.2.2 shows a section of the code metric tree near a pair of symbol errors. The correct path segment consists of

---

* G.D. Forney informed us this form of threshold loosening was implemented in Codex's 5 Mbps sequential decoder (Ref. 3).

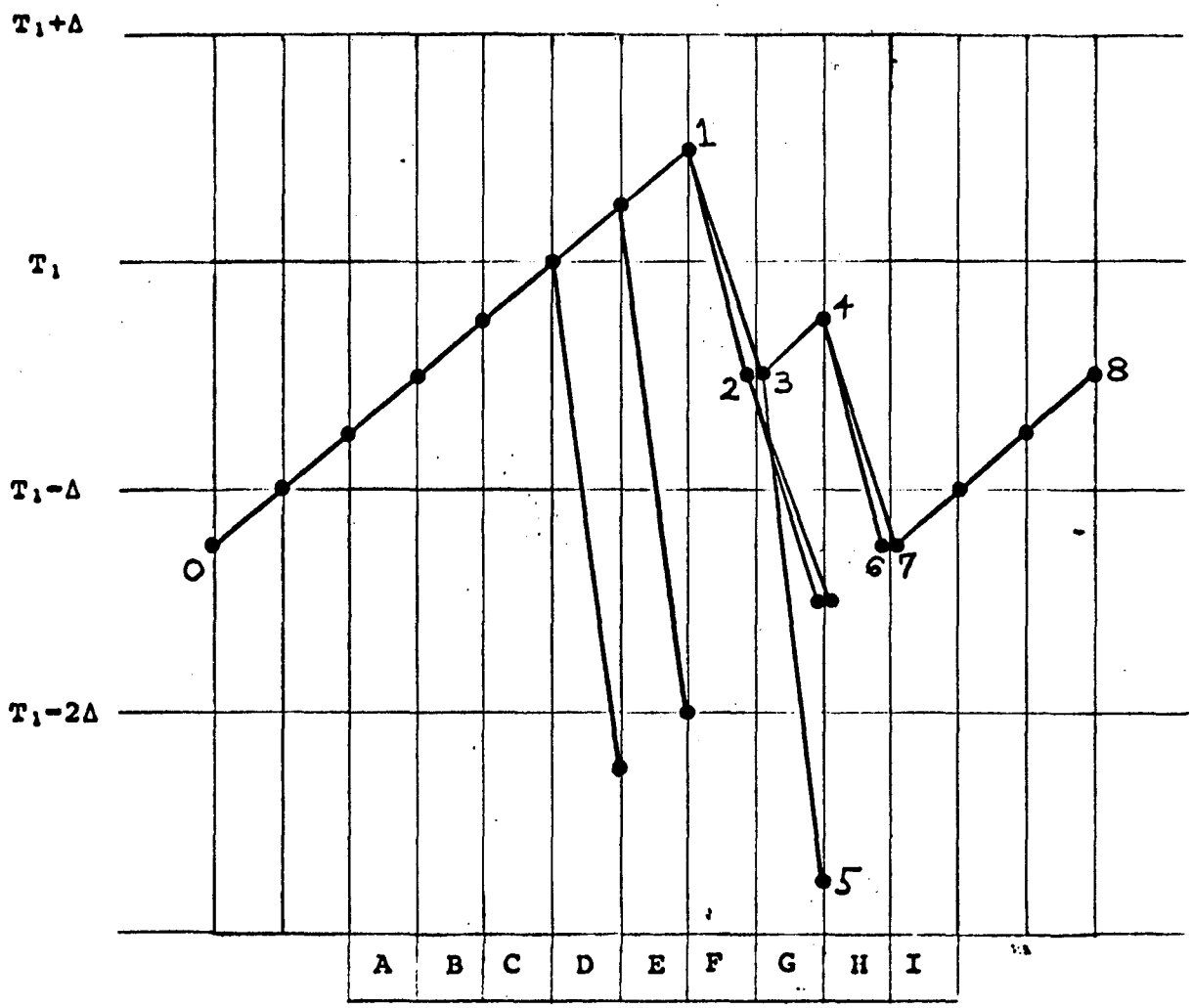


Fig. 3.2.2.2 Tree Section in the Vicinity of a Pair of Channel Errors.

branches from nodes 0 to 8. Symbol errors have occurred at branch levels F and H. If the initial two bits of the code generator are 11, then not only do 2 branches stemming from a common node have complementary symbols, but the four branches stemming from these nodes must contain all four combinations of the two code symbols. This insures that a tie vote at a node (such as node 1 in Fig. 3.2.2.2) must be followed by another tie vote at one node (node 2), and by a complete match and a complete mismatch stemming from the other node (node 3). Therefore, when an error occurs, the metrics on all paths except one, must fall through at least two threshold levels.

Suppose that the threshold has been tightened twice in a row, as is the case in going from node 0 to node 1. The decoder may go to node 3 and lower the threshold to $T_1 - \Delta$, since there are no other unsearched paths in the T_1 to $T_1+\Delta$ interval. At node 4 the decoder faces another tie vote that takes the metric below the present threshold, $T_1-\Delta$. Now because of the properties of codes with generators beginning with 11, there cannot be any unsearched path segments in the range $T_1-\Delta$ to T_1. Thus the threshold may be lowered again, and the decoder may continue on to nodes 7 and 8.

This modification of "quick threshold loosening" is also simple to implement. An up-down counter which counts from 0 to 2 is needed. Starting at "0", each time the threshold is tightened the count is incremented by 1, saturating at 2. When the threshold is violated in looking forward, it may be immediately

loosened if the count is positive. If so, the count is decrement by 1. When the count is zero, a backsearch must be initiated. As in ordinary "quick threshold loosening", we must have $\Delta = a-1$. When stepping forward to a node in the case of a tie vote, care must be taken to not step first to the node having another tie vote following it (node 2 in Fig. 3.2.2.2). This requires the ability to look at one syndrome bit into the future.

These "quick threshold loosening" schemes have been simulated to determine their effectiveness in reducing the number of short searches. Fortunately the value $\Delta = a-1$ is the near optimum choice of threshold spacing. These simulations, along with almost all of those reported here, use code reported in Ref. 3. Factors influencing the choice of code are discussed in section 3.2.3. Fig. 3.2.2.3 shows the distribution of computations per bit decoder with

1) The unmodified Fano algorithm
2) "Quick threshold loosening"
3) Modified "quick threshold loosening".

These simulations were performed with $p = .02$. Included in the figure is the average number of computations per bit, $\bar{c}$, for each run. Both quick threshold loosening schemes clearly eliminate searches with computations below about 20 to 40.

The modified "quick threshold loosening" is somewhat more effective than the loosening scheme described first. This is because it is capable of eliminating searches due to more error patterns. Fig. 3.2.2.4 shows a comparison of the computation

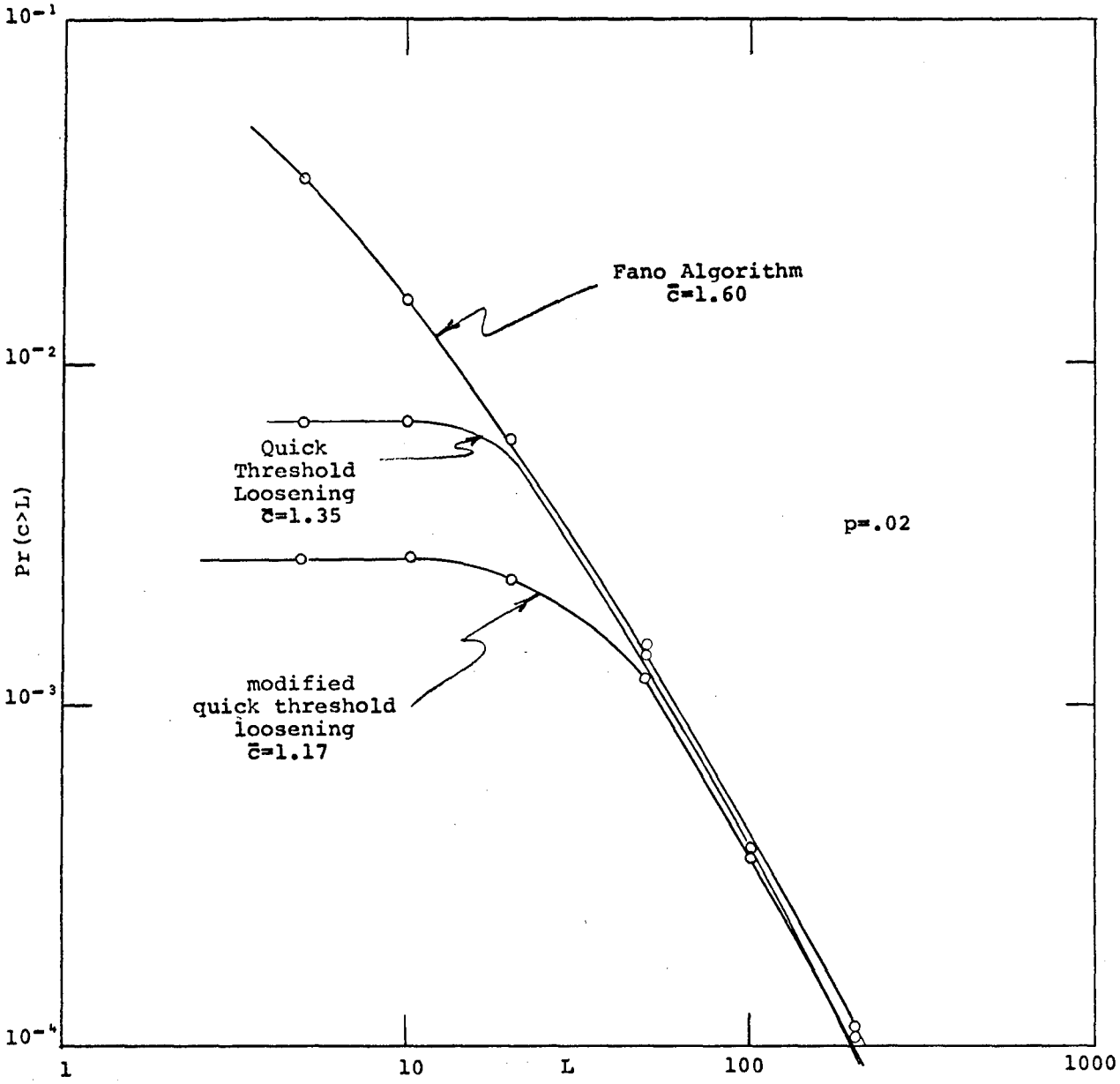


Fig. 3.2.2.3 Distribution of Computations per Bit Decoded for Standard Fano Algorithm and "Quick Threshold Loosening" Sequential Decoders. Threshold Spacing = 8, a=9, Channel Error Rate = .02.

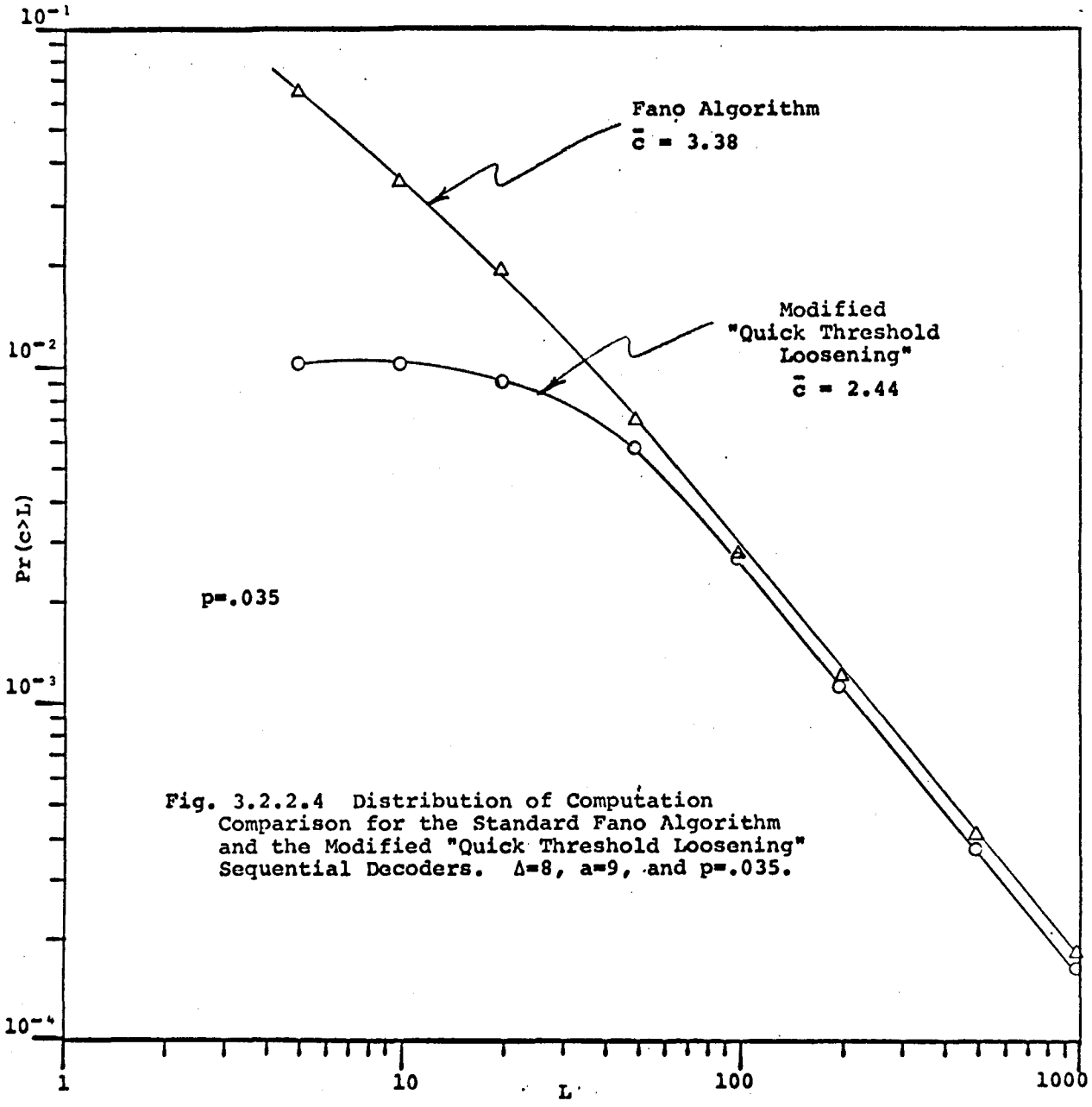


Fig. 3.2.2.4 Distribution of Computation Comparison for the Standard Fano Algorithm and the Modified "Quick Threshold Loosening" Sequential Decoders. Δ=8, a=9, and p=.035.

distributions for the standard Fano algorithm and modified quick threshold loosening decoders for p = .035.

Neither scheme substantially affects the tails of the computation distribution. This is because the long searches typically involve lowering the threshold by more than one or two levels, at which point in the search quick threshold loosening is disabled.

Because of the simplicity and effectiveness of the scheme, quick threshold loosening was used in many of the simulations, including all of the real time simulations that will be described.

3.2.2.3 Look Ahead Sequential Decoding. Look Ahead Sequential Decoding is another technique which attempts to improve the distribution of decoding computations. The standard Fano syndrome sequential decoder examines and acts upon the syndrome one bit at a time. If a path being followed is destined to fall below the current threshold, the decoder must follow the path until it actually does fall below the threshold before backing up and changing direction.

With look ahead decoding, the decoder examines N syndrome bits at a time. This is equivalent to looking forward N branches into the tree from the present branch. Using a table look-up procedure, the decoder determines if there is any path N branches ahead that satisfies the current threshold. If such a path exists, the decoder is allowed to step forward on a next branch if that can be done without violating the threshold. If no path exists N branches ahead that satisfies the threshold, then it is useless to allow the decoder to proceed forward. In that case, the look

ahead decoder initiates a backsearch, regardless of whether or not the next branch path metric satisfies T.

This scheme clearly does not do anything the standard Fano algorithm does not eventually do; however, it has the potential for doing considerable less. It reduces the depth to which a path must be searched before it is rejected.

Fig. 3.2.2.5 compares the distribution of computations for sequential decoding with and without look ahead, with a look ahead parameter of N = 6 branches. Use of the look ahead decoder with N = 6 results in an improvement in the computational distribution of about a factor of two.

From this figure, it is apparent that while the distribution is lower, the Pareto exponent, or the slope of the curves, is the same. It can be shown analytically that no sequential decoding algorithm which "looks ahead" a finite number of branches can change the asymptotic slope of the computation distribution curve. The most that can be hoped for is a lowering of the constant in front of the distribution (k in Eq. 3.1.1).

It is interesting to note that the effect of a scheme like looking ahead, which reduces the frequency of long searches, will be to reduce initial buffer overflow probability; while a scheme like "quick threshold loosening" will allow a full buffer to empty more quickly - allowing for a smaller decoder speed factor.

Practically speaking, the look-ahead mechanism is not as simple to implement as "quick threshold loosening", especially in very high speed decoders. The syndrome table look-up is a complex

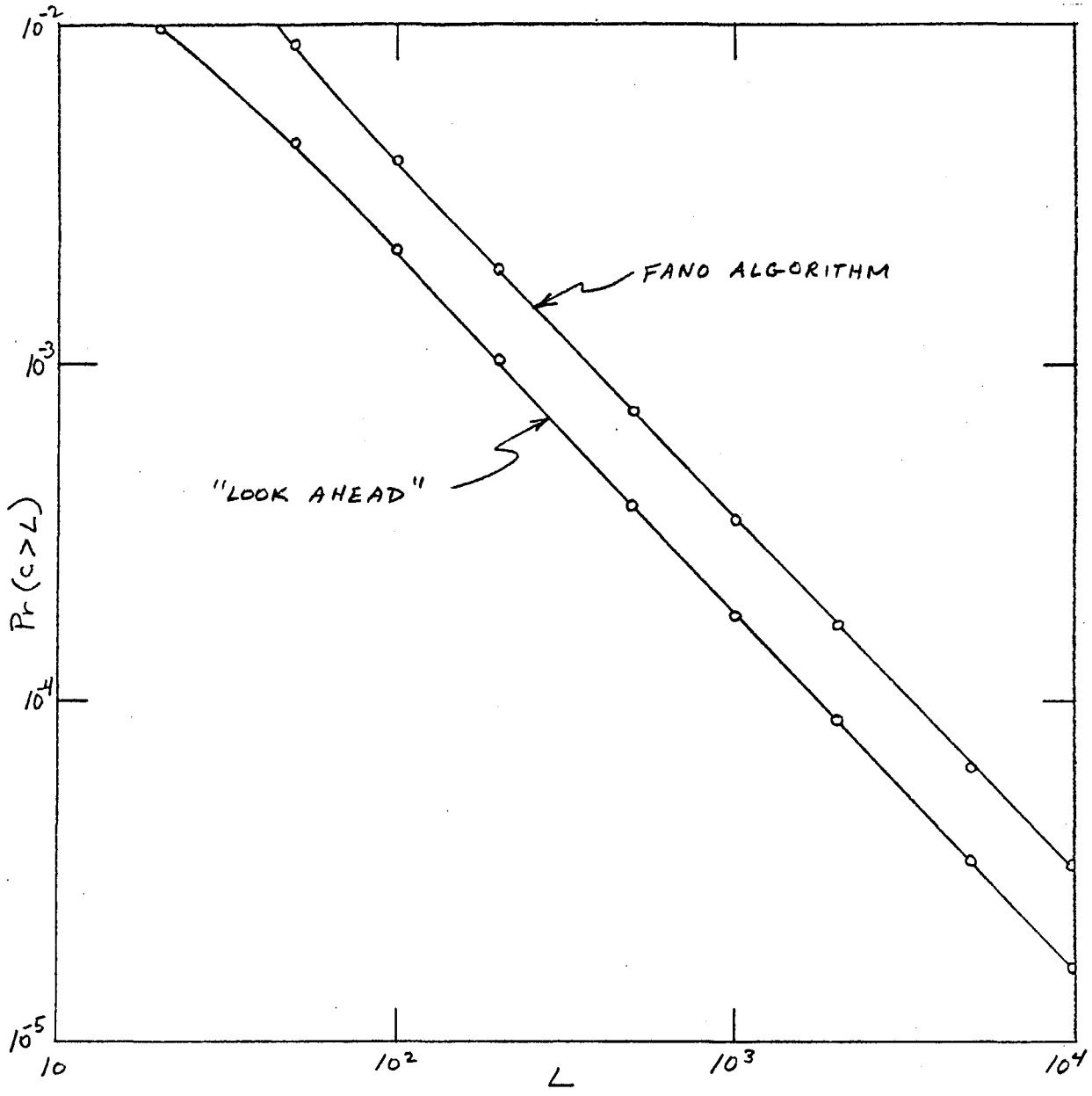


Fig. 3.2.2.5 Distribution of Computations per Bit Decoded For Standard and "Look-Ahead" Fano Algorithm Sequential Decoders. Threshold Spacing = 10, a = 9, Channel Error Rate = .039.

logical function which grows more complex exponentially with N. For N larger than about 6 it would have to be implemented using a read-only memory. Implementation would tend to slow down the decoder computation rate, which partially counteracts the improved computation distribution. Also, although the improvement in the distribution looks substantial, it only represents an improvement of about 0.1 db over the standard algorithm at points in the distribution which affect overflow probability with the real-time decoder parameters that were studied. For these reasons, look-ahead decoding was not used in the real-time decoder simulation reported in section 3.2.4.

3.2.2.4 Sequential Decoding with Sideways Looks. When a sequential decoder is stepping back to a node in the code tree, and the branch it is stepping through is a "best branch", its next step will be to look forward along the next best branch. In practice looking at the next best branch can be accomplished very conveniently before stepping back. If this node satisfies the threshold, and if the previous node also does, then a "step sideways" can be made directly to the next best node. Thus, what was two computations becomes one computation. Sideways looks will be counted as one rather than two computations in the simulation which follows.

3.2.3 Decoder Undetected Error and Computational Performance. The sequential decoder simulations described thus far, and the ones presented in this section are for decoders operating in a non-real

time mode. That is, received data is generated as the decoder needs it. The performance data gathered is, therefore, that for a decoder in which buffer overflows do not occur. Real time decoder behavior, with a simulated buffer, is reported in the next section.

3.2.3.1 Code Selection. Choosing codes is not as critical for sequential as it is for Viterbi decoding. Decoder complexity is not a strong function of code constraint length; so, the undetected error performance of a code can be improved by increasing K rather than trying to optimize a code for a given value of K. Still there are several reasons for having as good a code as possible.

1) The guess and restart overflow technique performance degrades with increasing constraint length. This is because a constraint length worth of data must be correctly "guessed" to restart decoding.
2) The constant, k, in the computational distribution is somewhat sensitive to the code. Good code distance properties will result in value of k=1 or less.
3) The encoder replicas in the decoder do grow linearly with K, resulting in some additional cost and complexity.

It has been found through simulation that of the known codes, the various truncations of the rate 1/2 systematic codes due to Lin and Lyne (Ref. 7), and Bussgang and Forney (Ref. 3,5) proved best in undetected error rate, and distribution of computations.

Forney's extensions of Bussgang's code seem to have the slight advantage. Since this code has already been used in two high speed sequential decoder implementations we have concentrated our efforts on it. A search for more optimal codes is probably not worthwile because

1) The number of codes of constraint length 40 or thereabout is huge.
2) The truncations of Forney's code have minimum free distances close to the upper bound on d_f.

Forney's code, to constraint length 45 has the generator 715473701317465 in octal digits. The first two binary digits in the generator are 11, satisfying the requirement of the modified "quick threshold loosening" scheme. All simulations in this report use this code generator or its shorter constraint length truncations.

3.2.3.2 Decoder Parameters. Two decoder parameters which must be selected are the threshold spacing Δ, and the symbol mismatch metric, -a.

Simulations have shown (Ref. 3) that decoder computational performance is not extremely sensitive to Δ. A broad minimum in $\bar{c}$, the per bit average computation, exists centered about $\Delta = 10$. This value of Δ is convenient, as it turns out, in that it meets the requirements of the "quick threshold loosening" scheme.

Whereas computational performance is not strongly affected by Δ, it is very sensitive to the symbol metric ratio 1/-a. Here 1

is the metric assigned to a match in code and received symbols, and -a is the metric assigned to a mismatch. When operating at $R=R_{comp}$ (corresponding to p = .0447 for rate 1/2), the value of a that maximizes the computational distribution Pareto exponent is a = 9.1. Any different value of a will decrease the Pareto exponent. However, as p decreases, R_{comp} increases, and the optimal value of a increases. At p = .035, the best choice for a is near 10. From an implementational point of view, by far the simplest values of a to use are odd integers. Therefore, the choice seems to be between 9 and 11.

It has been shown (Ref. 4) that for a systematic code, when a is chosen to optimize the distribution of computation, the error probability does not go down as rapidly with K as shown in the bound in Eq. 3.1.5. This means that significantly larger values of K are needed than would be necessary for a larger a. Our simulations, which are in substantial agreement with those in Ref. 3, indicate, for instance, that for p = .0447 (R_{comp} = 1/2), about the same bit error rate is attained with K = 45 and a = 9 as with K = 37 and a = 11. That error rate was just above 10^{-5}. Other simulations indicate that the behavior of the error rate with K is consistant with the bound of Eq. 3.1.5 when a = 11. Also, since the optimum a is closer to 11 than 9 when p is greater than .035 (which is really the range of interest for high data rate, low error rate operation), we have elected to concentrate on simulation with a metric ratio of 1/-11.

3.2.3.3 The Distribution of Decoding Computations. Before proceeding to present the computational statistics gathered through simulation, we redefine a computation to be consistant with what happens in a hardware computational cycle. For this purpose it is more accurate to define a computation as having occurred when the decoder steps rather than looks forwards, backward or sideways. This eliminates counting as a computation: for instance, a look forward by the decoder that does not result in a step forward.

Using this new definitation Fig. 3.2.3.1 present the distribution of computations per bit decoded for a range of channel error rates, p. For each value of p, 8×10^6 bits were decoded. The straight lines are best fits to a section of the tail of the distributions where significant data exists. The negative of the slopes of these lines are the measured exponents of the assumed Pareto distribution. As expected, the fit to the Pareto distribution is excellent. Table 3.2.3.1 shows the measured Pareto exponent α_m vs. p. Also included is the theoretical Pareto exponent, α_T, which assumes the use of the optimum metric ratio, 1/-a. Clearly, there is some degradation in the exponent for the higher values of p. (corresponding to operation near R_{comp}); however, for $p \geq .035$, the measured and theoretical values are close. This indicates that the optimum range for a is broad when p is low.

3.2.3.4 Measured Undetected Error Rates. Table 3.2.3.2 shows the measured number of bit errors vs. p for the constraint length K = 33, 37, and 41 codes. In each case 8×10^6 bits were

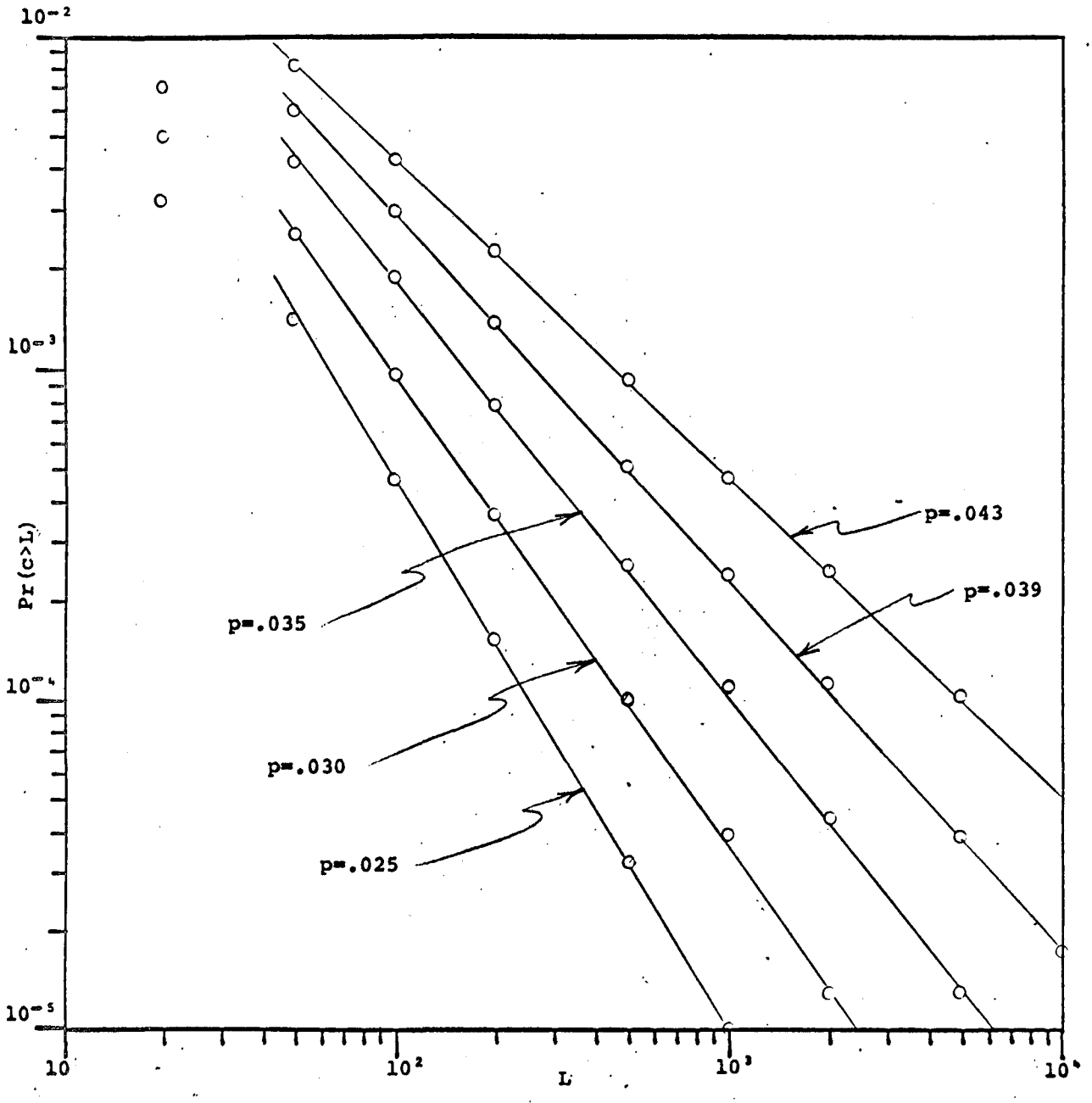


Fig. 3.2.3.1 Sequential Decoding Distribution of Computations for Several Channel Error Rates, p. Runs are all of 8×10^6 Bits. K=41, Δ=10, Metric Ratio is 1/-11.

E_b/N_0	p	α_T	α_m	$\bar{c}$
4.7 db	.043	1.05	0.97	7.08
4.9 db	.039	1.16	1.12	3.09
5.1 db	.035	1.28	1.29	2.03
5.4 db	.030	1.46	1.44	1.50
5.8 db	.025	1.67	1.66	1.25

Table 3.2.3.1 Measured and Theoretical Computational Distribution Parameters. K=37, Δ=10, Metric Ratio is 1/-11.

p	K		
	33	37	41
.043	390	205	79
.039	128	120	0
.035	37	0	0
.030	25	0	0
.025	0	0	0

Table 3.2.3.2 Undetected Bit Errors vs. p and K, For Decoder Runs of 8×10^6 Bits Each.

decoded. Only a small number of error events, if any, have occurred at the smaller values of p, making this data rather insignificant. At p = .043, however, the error probability is fairly close to $2^{-K/2}$, as the upper bound predicts.

3.2.4 Real Time Sequential Decoder Simulation. In the real time sequential decoder simulation program, the decoder buffer waiting line is simulated by a counter. The counter is incremented every μ decoder computations to simulate the periodic arrival of received data. The counter is decremented when a new node level is reached by the decoder. Thus, the count represents the number of branches of received data in the buffer waiting to be decoded. When the count reaches a number equal to the simulated buffer size, an overflow is declared. At this time the guess and restart routine attempts to jump forward about one constraint length, and resume decoding.

In a high data rate decoder, where the speed factor μ is only slightly larger than $\bar{c}$ (the average number of computations per bit), the overflow probability will not obey the heuristically derived formula of Eq. 3.1.3. Also, when an overflow does occur, it will take many resync trials to successfully start decoding again. This is true because even when the correct syndrome guess is made, the decoder speed advantage is so slight and the buffer is so full, that the decoder will often overflow again.

Fig. 3.2.4.1 shows measured decoder composite output error probability vs. constraint length for a decoder with a buffer size

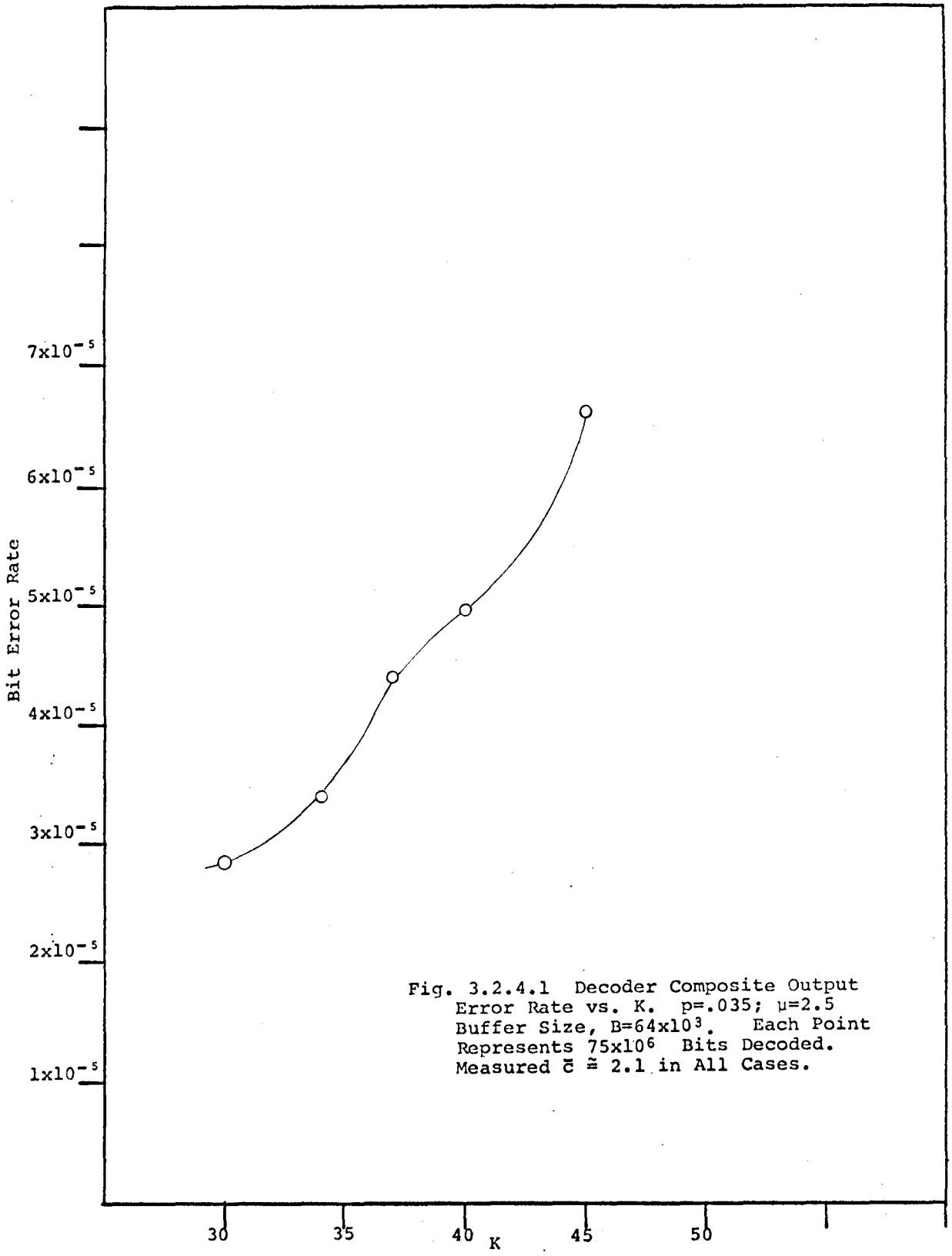


Fig. 3.2.4.1 Decoder Composite Output Error Rate vs. K. p=.035; μ=2.5 Buffer Size, B=64x10³. Each Point Represents 75x10⁶ Bits Decoded. Measured $\bar{c} \cong 2.1$ in All Cases.

of 64 x 10^3 bits, a speed factor, μ, of 2.5 and a channel error rate p = .035. Here composite output error rate refers to errors due to overflows, as well as undetected errors. In all cases the average number computations per bit was about 2.1. This figure is somewhat surprising in that error rate goes up with constraint length. This peculiar behavior can be explained as follows. With the longer constraint lengths, errors due to overflows completely dominate the composite error rate. At a constraint length of 45, the average number of related overflow events following an initial overflow is about 16. Each overflow causes about 300 raw undecoded information bits to be output by the decoder. Since the error rate on these undecoded bits is p = .035, each overflow results in about 10 output errors. Since overflows occur in bursts of an average of 17 apiece, an overflow burst typically results in 170 output errors.

Shortening the constraint length will cause undetected errors to take the place of some of the overflows. The average number of bit errors in an undetected error event is far less than 170. Thus, decreasing the constraint length has the net effect of improving the composite error rate. Of course, after a point undetected errors will dominate, and decreasing K further will increase the error rate. This point, however, has not been reached in going down to constraint length 30, as shown in the figure.

The reason that about 300 bits are output in uncorrected form in a overflow, is as follows. When an overflow occurs,

error position decisions up to 256 branches back from the point of deepest decoder penetration are considered unreliable, and are erased. The decoder then jumps over about a constraint length worth of untouched data, for a net of about 300 bits. The reason that 256 decisions are erased is that, in a long search, the decoder may back up and change decisions as many as 200 nodes back or more. The most recent nodes must therefore not be considered finally decoded until the decoder has progressed at least 200 nodes deeper into the tree.

Fig. 3.2.4.2 shows composite bit error rate, in the neighborhood of 10^{-5}, vs. p for two decoder speed factors, $\mu = 2.5$ and $\mu = 1.5$. Clearly the curves are extremely steep. Operation at a speed factor of 1.5 with a bit error rate of 10^{-5} requires a decrease in p of only about .006 compared with operating at the same bit error rate with a speed factor of 2.5.

This comparison can be made even more dramatic if it is put in terms of data rate and E_b/N_0. Suppose we have a sequential decoder which is capable of 100 mega-computations per second. At a data rate of 40 Mbps, the speed factor is 2.5. A composite bit error rate of 10^{-5} is achievable with a p of about .033 or an $E_b/N_0 = 5.2$db.

Likewise the decoder can operate at a data rate of 66 Mbps with a p of about .027, or $E_b/N_0 = 5.7$ db. The decoder data rate can be increased by over 50%, with the same output error rate, at the expense of only about 0.5 db!

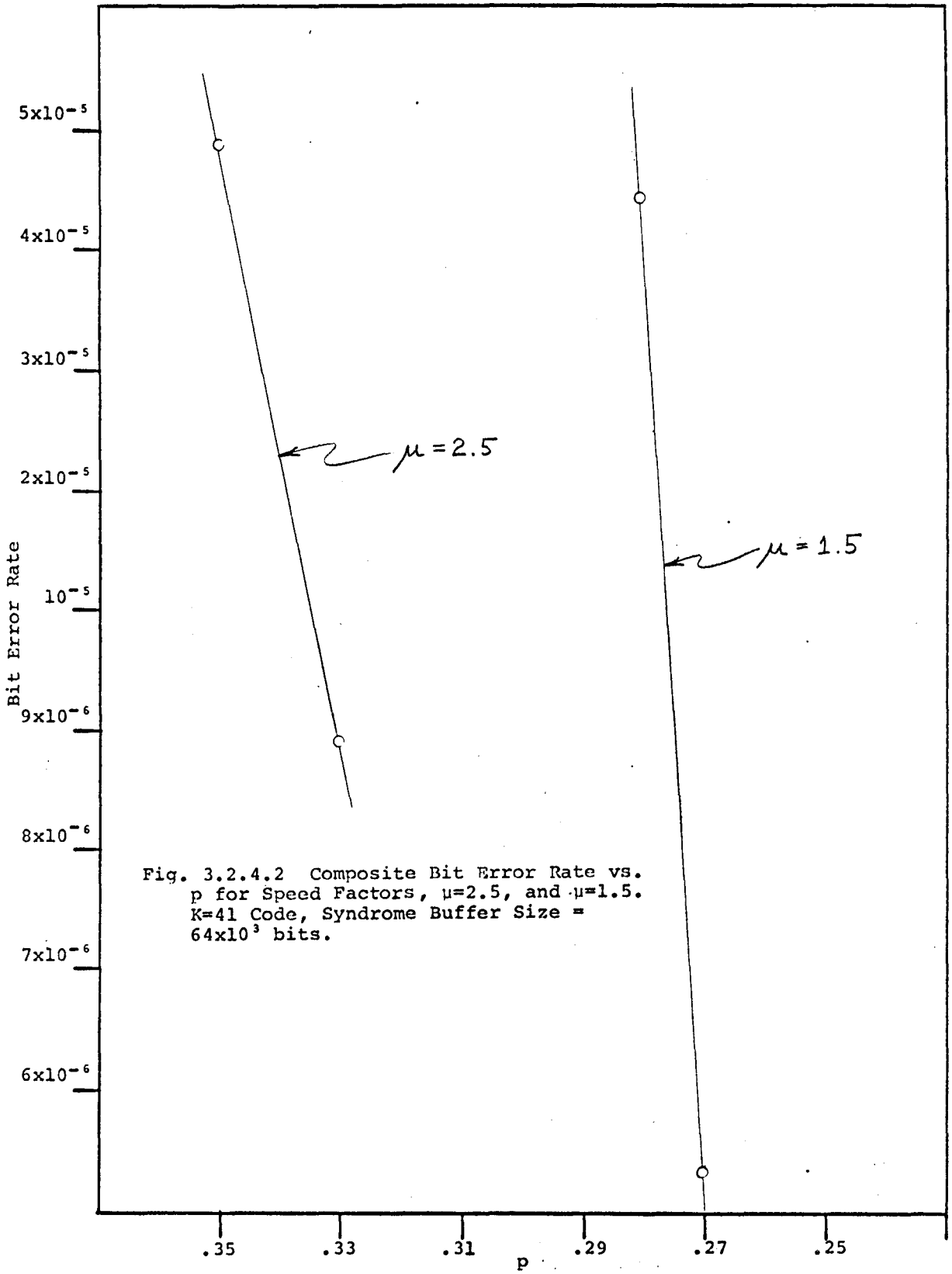


Fig. 3.2.4.2 Composite Bit Error Rate vs. p for Speed Factors, μ=2.5, and μ=1.5. K=41 Code, Syndrome Buffer Size = 64×10^3 bits.

Clearly there is a continuity of data rates and E_b/N_0 that correspond to a composite bit error rate of 10^{-5}. Fig. 3.2.4.3 shows a curve of such data rates vs. E_b/N_0. The curve is an interpolation and extrapolation of the two points obtained from Fig. 3.2.4.2.

3.2.5 Erasures vs. Undetected Errors. In some applications erasures are not nearly as bad as undetected errors. If this is the case, output error can easily be decreased by orders of magnitude by

1) Increasing K until overflow errors dominate
2) When an overflow occurs declare the 300 or so bits affected by the overflow as erased.

This results in a bit erasure rate of about 1/p times the original composite error rate. However, undetected error rate can be as small as desired, depending on K.

3.2.6 Systematic vs. Nonsystematic Codes. All of the simulations reported were run using systematic codes. Systematic codes have the advantage over nonsystematic codes that in the event of decoder failure, the raw information bits are available directly as back-up. This advantage has been obviated, to a degree, by the invention of the "quick-look" nonsystematic codes (Ref. 6). Use of these codes allows for simple information bit generation without a decoder. The resulting information stream does have an error rate of about 2p, however. There are two main advantages in using nonsystematic codes

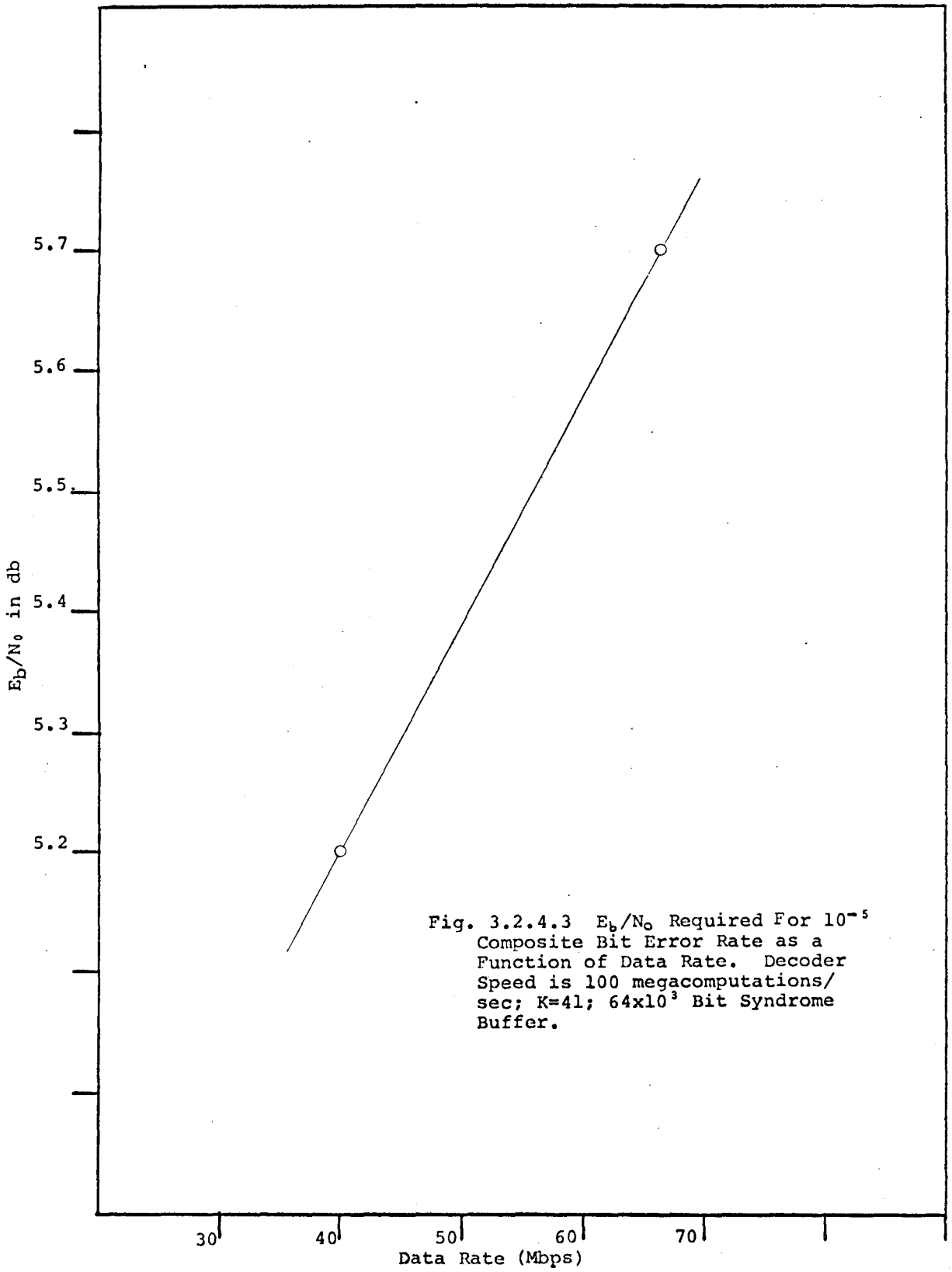


Fig. 3.2.4.3 E_b/N_o Required For 10^{-5} Composite Bit Error Rate as a Function of Data Rate. Decoder Speed is 100 megacomputations/sec; K=41; $64x10^3$ Bit Syndrome Buffer.

1) A constraint length of only about half the systematic code constraint length is required for the same detected error rate performance, and

2) The optimum metric ratio, on the basis of the distribution of computations, also results in the achievement of the optimum undetected error probability exponent. This is unlike the case for systematic codes, where a larger value of "a" is required for optimal error performance.

The metric ratio advantage in 2) is completely lost at $p > .035$. In this most interesting range, for high speed decoders, the optimum metric ratio range is broad enough to allow joint optimization of the computation distribution and error exponent with systematic codes.

With the guess and restart overflow strategy, some form of "guess" of undecoded information bits must be output when the decoder overflows. Using a non-systematic code, these "guesses" will be less reliable than with a systematic code.

For these reasons we have elected to concentrate on systematic codes.

3.2.7 Code Synchronization and Channel Reliability Prediction. Code synchronization and channel reliability can be handled in much the same manner as discussed in conjunction with Viterbi decoding. Recall in that case that an up-down counter was used, which counted up k on the occurrence of an error and down by 1 at each bit time, never going below zero. Code sync state is changed when the count exceeds a value T. In the sequential decoder it is even simpler. The counter now counts up by one when a threshold is loosened, and down by one when a threshold is tightened. The skewing of the count is done automatically by the skewed metric ratio. False alarm, and resync rates are directly obtainable from the E_{fa} and E_{rs} curves in Figs. 2.2.5.4 and 2.2.5.5.

3.3 Soft Decision Sequential Decoding

3.3.1 Syndrome Decoder. A syndrome sequential decoder can be used to advantage with a soft decision decoder as well as a hard decision decoder. The syndrome is formed using only the hard quantized information in the received data. Passed on to the decoder, along with the syndrome bits, is 2 data quality bit per branch for 4 level quantization, or 4 quality bits per branch for 8-level quantization.

The decoder uses the 3 or 5 bit per branch information to generate information error decisions just as in the hard decision case. Now, however, efficiency is improved by the availability

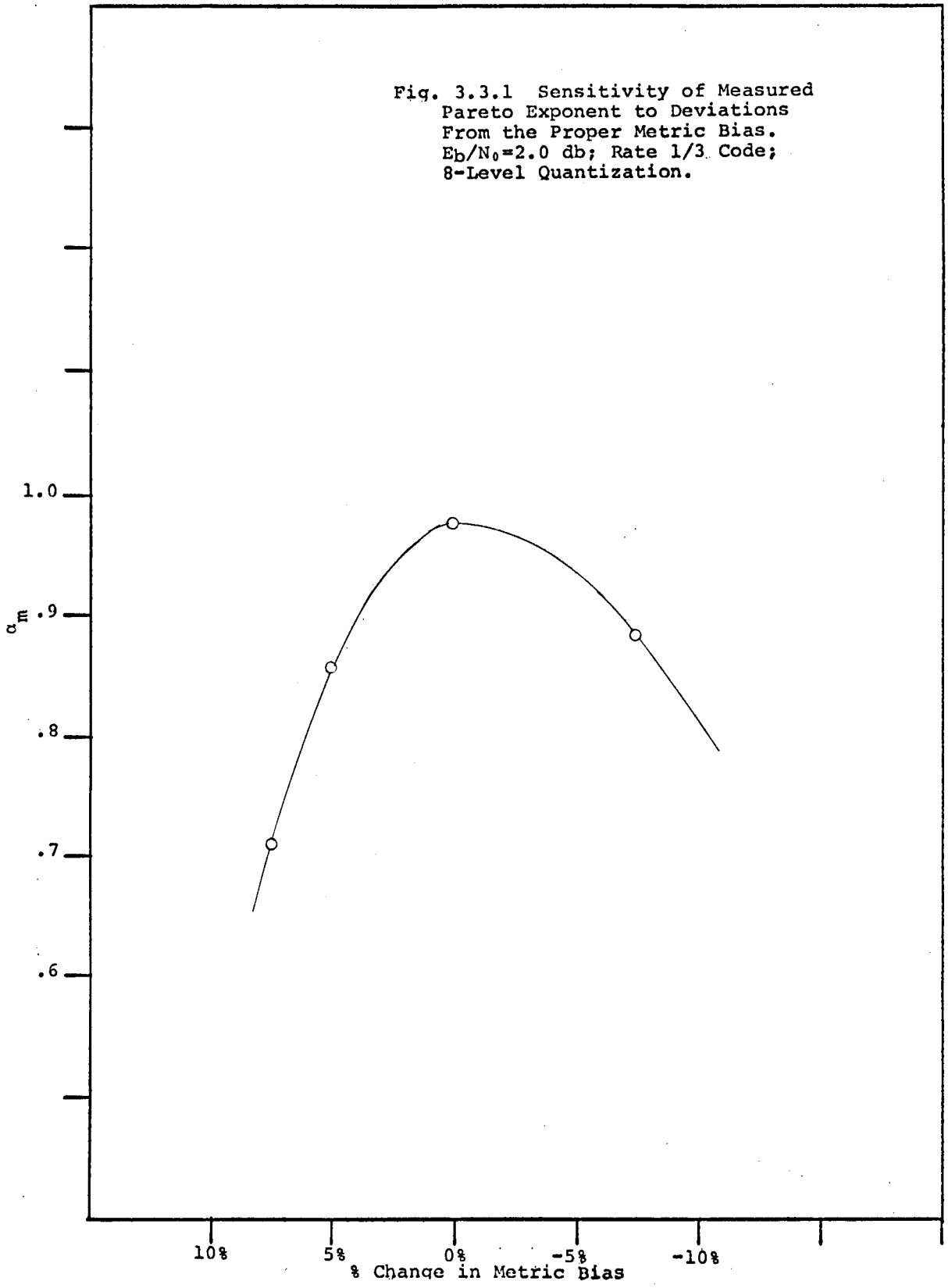


Fig. 3.3.1 Sensitivity of Measured Pareto Exponent to Deviations From the Proper Metric Bias. E_b/N_0=2.0 db; Rate 1/3 Code; 8-Level Quantization.

of the quality data. The error decisions are used to correct the stored, raw, hard quantized received information bits.

3.3.2 Fano Algorithm Modifications. The guess and restart overflow strategy is applicable to a soft decision decoder. Here the "guesses" will be less frequently correct, because of the lower required E_b/N_0 with a soft decision decoder.

Quick threshold loosening does not carry over to soft decisions. The wider range of branch metrics makes it hard, if not impossible, to take advantage of code and metric structure to allow quick threshold loosening.

3.3.3 Sensitivity to Incorrect AGC. Unlike Viterbi decoding, soft decision sequential decoding is extremely sensitive to improper threshold level setting, due to inaccurate or drifting AGC. This is illustrated in Fig. 3.3.1. In this figure, the effect on the Pareto exponent of variations in the decoder metric bias as a percentage of the maximum branch metric are shown. Clearly, even these small changes in metric bias result in large changes in computation distribution.

3.3.4 Comparisons of Soft and Hard Decision Sequential Decoders. An 8-level quantized sequential decoder, with the same size received data buffer as a hard decision decoder, can buffer only 1/5 as many received branches. Also, the increased complexity of the decoder logic will mean a computation speed reduction of about a factor of 2 under that of a hard decision decoder (see section 3.4).

This along with the AGC problem, and the nonexistence of a "quick threshold loosening" scheme will reduce the 2 db performance advantage, inherent in fine receiver quantization, over hard decisions. In fact at very high speeds, where speed factors are low, there may be no performance advantage to a soft decision decoder. For these reasons - especially the AGC sensitivity, we cannot recommend soft decision sequential decoding for very high speed communication.

A technique is described in section 5.0, however, which uses a Viterbi predecoder before data enters a soft decision sequential decoder. This technique effectively improves the sequential decoder speed factor, and lowers the amount of memory required, and may provide a means of attaining the efficiency of a soft decision sequential decoder at high speeds.

3.4 Sequential Decoder Implementation. This section considers implementation techniques for syndrome sequential decoders operating at data rates from 1 to 40 Mbps. Trade-offs consid-

BLOCK DIAGRAM

SYNDROME SEQUENTIAL DECODER

FOR RATE ½, SYSTEMATIC CODE

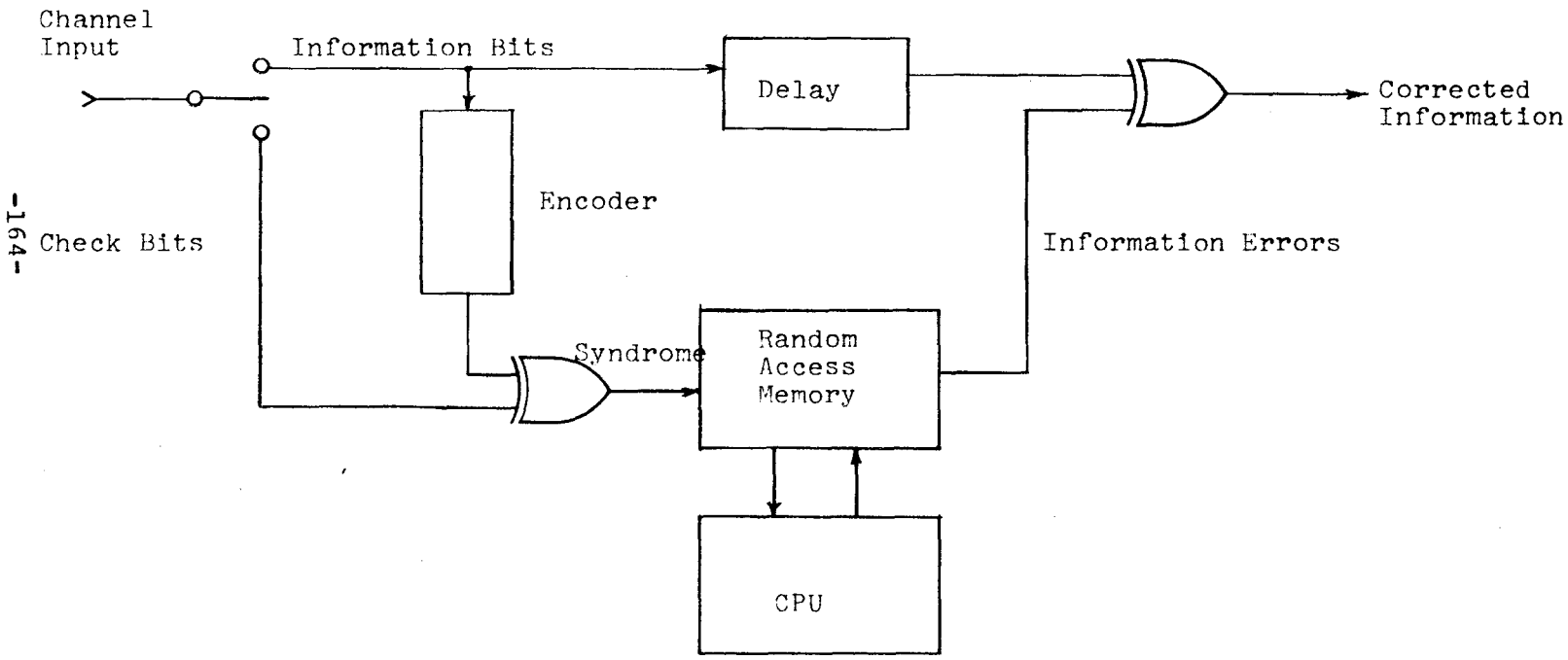


Figure 3.4

ered are choice of logic family and hard vs. soft decisions.

A block diagram of a syndrome sequential decoder is shown in Fig. 3.4. The input from the channel is separated into two bit streams, one containing received information bits and the other containing the received check bits. The received information bits are passed through an encoder which is identical to the one used at the transmitter, and are exclusive-ORed with the received check bits, thus generating the syndrome. The syndrome is then stored in a random-access memory. In the case of soft decisions, the quality bits are stored in the memory with the syndrome. Meanwhile, the information bits are stored in a delay line which is equal in length to the total delay through the random-access memory and CPU. The processer reads the syndrome bits (and quality bits, if any) from the random-access memory and, using a modification of the Fano algorithm, determines a likely information error sequence. The decoded information error sequence is then read back into the random-access memory. The information error sequence remains in the random-access memory until the corresponding received information bits are shifted out of the delay line. The information bit sequence and the information error sequence are then exclusive-ORed, correcting any errors present in the received information sequence. The resulting corrected data is the decoder output.

All implementations considered would use semiconductor memory devices to form the main memory, TTL logic circuits for memory buffer, control, and syndrome generation and either TTL

or MECL for input and output interface circuits as required by the data rate. The choice of logic family for the CPU section is more critically dependent on the maximum data rate desired.

The implementation study has found that the maximum computation rates for MECL III, MECL II, and TTL are, respectively, 100, 25, and 13 megacomputations per second. The corresponding data rates for speed factor 2.5 are, respectively, 40, 10, and 5 Mbps. The relative cost factors are the same as for the Viterbi decoder, i.e., 18, 3, 1 for MECL III, MECL II and TTL, respectively.

The trade off study for Viterbi decoding showed that the most cost effective way to obtain 40 Mbps decoding was to parallel four 10 Mbps TTL decoders. Such is not the case for sequential decoding.

A 40 Megabit sequential decoder could be obtained by building one decoder with a MECL III CPU, or four decoders in parallel with MECL II CPU's or eight decoders in parallel with TTL CPU's. In the case of the MECL III decoder, CPU accounts for approximately 60% of the total cost and the memory and I/O circuitry account for 40%. The memory and I/O circuitry for a MECL II CPU decoder would run about 85% of the cost for the MECL III decoder. The memory and I/O for a TTL CPU decoder will cost about 75% of that required for the MECL III decoder. The results for 40 Mbps are shown in the following table:

CPU TYPE	COST OF MEMORY	COST OF CPU	RELATIVE COST OF 1 DECODER	NUMBER OF DECODERS	RELATIVE TOTAL COST
MECL III	.4	.6	1.0	1	1.0
MECL II	.35	.1	0.45	4	1.8
TTL	.3	.03	0.33	8	2.64

Thus, the MECL III design turns out to be the most inexpensive for this data rate. The MECL III decoder for hard decisions is presented in detail in the following section.

3.4.1 40 Mbps Sequential Decoder. A detailed block diagram of the 40 Mbps sequentail decoder is shown in Fig. 3.4.1. The input from the channel is accepted in either of two forms; as in serial bit stream at a maximum rate of 80 Megasymbols per second, or as two parallel lines at a maximum rate of 40 Megasymbols/ second each, corresponding to a maximum data rate of 40 Megabits per second.

The parallel inputs are provided for decoding QPSK modulated data. Received information and parity bits are routed separately from the modem to the decoder. 90° demodulator phase ambiguities are resolved by the code synchronization circuitry.

The 80 Mbps serial input is for use with BPSK modems. In

this case, the interleaved received information and parity bits are decommutated in the decoder. Node synchronization is provided again by the code sync circuitry. For both BPSK and QPSK, 180° phase ambiguities will be handled by using codes transparent to 180° phase flips and differentially encoding and decoding the data.

The input circuits exclusive of the serial input decommutation are implemented using MECL II logic. The input lines are then buffered down to eight parallel lines operating at a maximum rate of 10 Megabits per second per line. The eight resulting lines are then converted to TTL logic levels and delivered to the encoder replica and buffer.

The syndrome is then formed from the received information and parity bits. The syndrome is collected into 72 bit words and delivered to the random-access memory. The random-access memory is a semiconductor memory consisting of 1,024 words of 72 bits each. The memory has a read/write cycle time of 400 nanoseconds. The syndrome words are written into sequential locations in the random-access memory. Prior to writing the syndrome word, the present contents of the addressed word are read out of the memory. This word contains the decoded information error sequence. The word is sent on to the output buffer where it is exclusive-ORed with the delayed information bit sequence.

The random-access memory has two ports. The input-output section is connected to the first port and has priority. The CPU buffer is connected to the second port. The CPU buffer reads a

72 bit word out of the random-access memory and delivers it to the CPU four bits at a time whenever the CPU requests new data. After reading out the new syndrome word, a new information error word is written into the memory at the same address. When the CPU buffer memory address catches up to the input-output address, the CPU buffer must wait until a new word is written into the random-access memory from the input-output section. When the CPU buffer falls so far behind the input-output section that the input-output section attempts to write over an undecoded word, a buffer overflow is declared. Whenever an overflow occurs, a signal is sent to the external equipment indicating the next word of data is likely to contain errors. The overflow signal is also sent to the Fano algorithm logic so that it may restart at a point further ahead in the memory.

When the CPU section reaches the front of the back-up buffer, a new four bit syndrome word is requested from the memory section via the CPU buffer. This four bit word is exclusive-ORed into the encoder. The encoder is capable of shifting in either direction. If the algorithm logic determines that the present node contains an information bit error, then the code impulse function is exclusive-ORed into the encoder. Likewise, when the decoder is backing up, the impulse functions that were exclusive-ORed into the encoder while proceeding forward must be removed. The bits shifted out of the right hand side of the encoder while moving forward correspond to the check bit error sequence. These bits are shifted into the back-up buffer along with the informa-

HIGH SPEED SEQUENTIAL DECODER BLOCK DIAGRAM

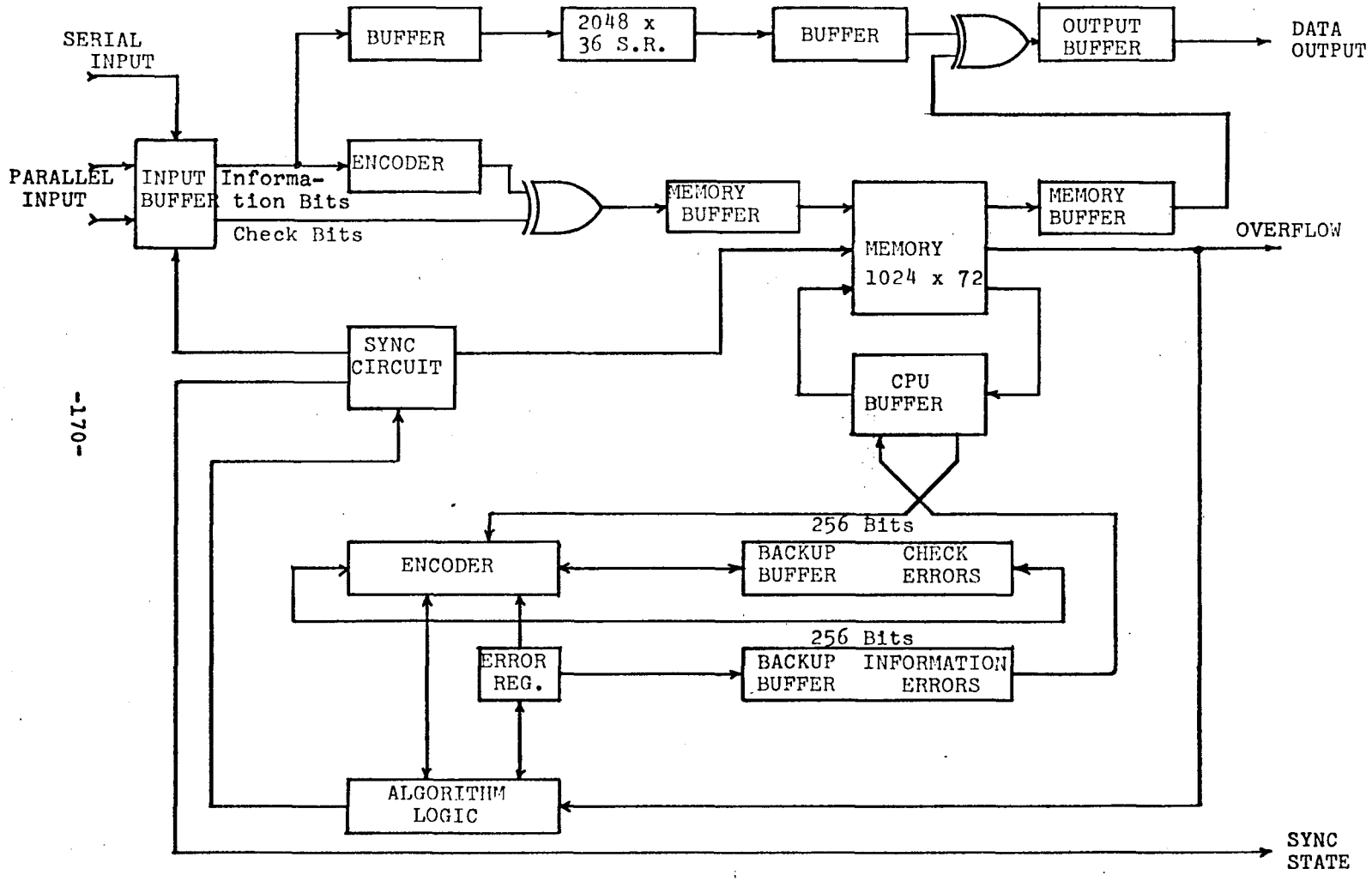


Figure 3.4.1

tion error sequence. When the decoder is backing up, the original syndrome is reconstituted by shifting the check error sequence back into the right hand side of the encoder, and by exclusive-ORing in the code impulse function wherever an information error previously was hypothesized. The syndrome bits are shifted out of the left hand side of the encoder when backing up and are shifted into the back-up buffer.

Functionally, the back-up buffer is a right-left shift register that is two bits wide and 256 bits long. The back-up buffer is actually implemented by using a very fact ECL random-access memory that is addressed by up-down counter, thus giving the effect of a right-left shift register. The function of the algorithm logic is to direct the progress of the CPU through the decoding tree. The algorithm logic determines whether the decoder may proceed forward or backward and determines the changes in the decoder metric.

3.4.2 Code Synchronization. The function of the synchronization circuit is to obtain correct code sync. This is accomplished by comparing the rate of metric threshold loosenings with the rate of metric threshold tightenings. If the code sync state is correct, then the rate of metric tightenings will exceed that of metric loosenings. However, if the code sync state is incorrect, then metric loosenings will exceed the rate of metric tightenings. The sync circuit contains a counter which is counted up every time a metric threshold loosening occurs and is counted down

every time a metric threshold tightening occurs. The counter has a reflecting boundary at zero. If the counter overflows while counting up, a bad sync state is declared and the code sync state is changed. Code sync resolves the 90° phase ambiguity in the QPSK input case and accomplished node sync in the BPSK case. The counter is large enough that the probability of falsely declaring a bad sync state is negligible compared with decoder error rate. The code sync state is changed if the input is from a PSK modem by inserting or deleting a one symbol time delay. If the input is from a QPSK modem the sync state is changed by interchanging the two symbols and inverting one of them. Also, when the sync state is changed the CPU buffer address is set equal to the input-output address and the CPU is restarted at that point.

3.4.3 Input Buffer. A detailed logic diagram of the input buffer is shown in Fig. 3.4.3. The inputs from the channel are each received on twisted pairs which are terminated in the characteristic impedance of the line, and then fed to a set of line receivers. The output of the clock line receiver is then fed to the clock conditioning circuit which provides all of the necessary clocks to the decoder. The data inputs are then fed to three flip-flops which format the various inputs into two parallel lines, one line containing the information bit stream and the other line containing the check bit stream.

If the serial PSK input is used, the three flip-flops are connected as a three bit shift register. The information bit

SEQUENTIAL DECODER INPUT BUFFER

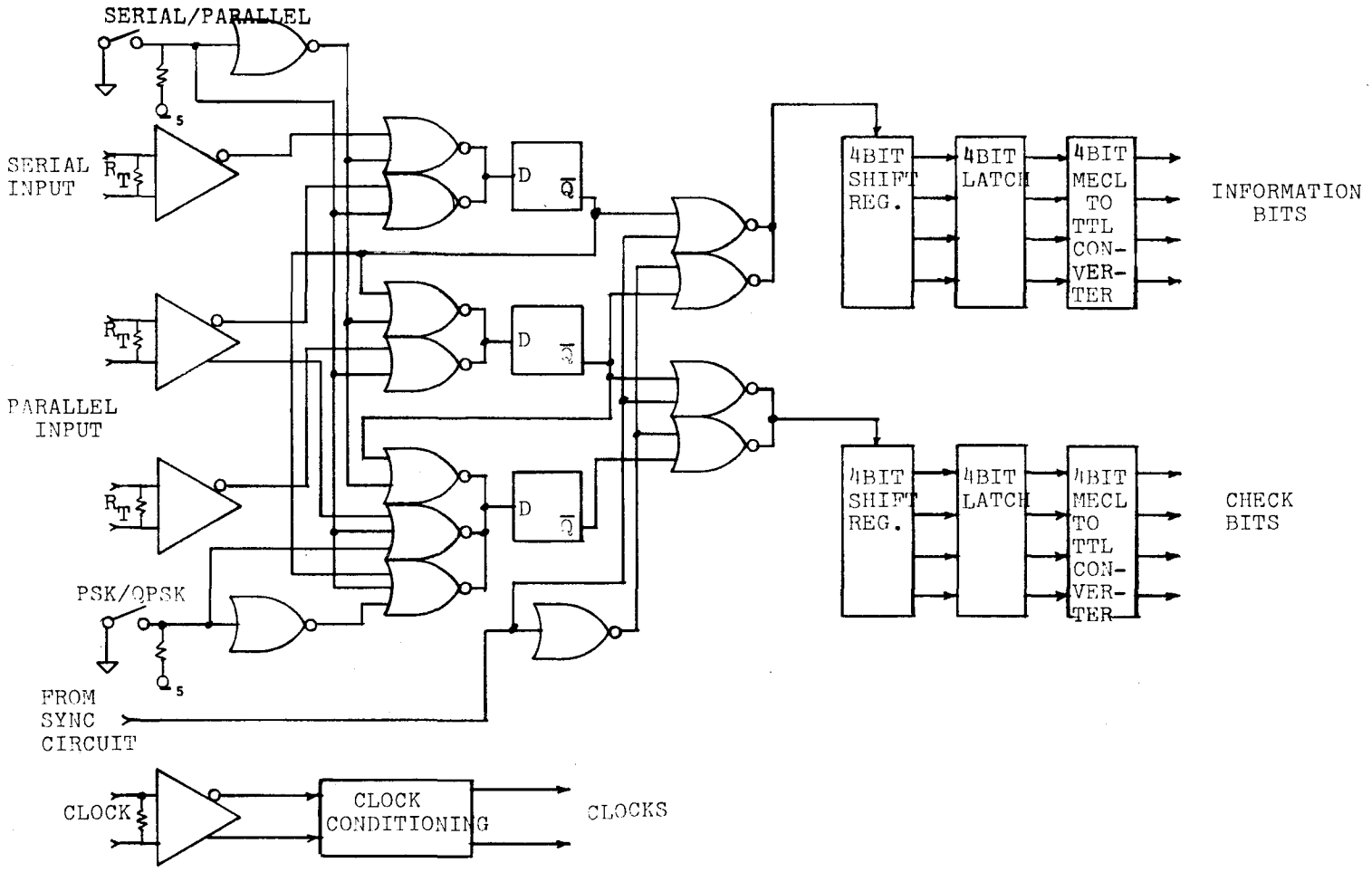


Figure 3.4.3

output is taken from the first register and the check bit is taken from the second register in one sync state. In the other sync state, the information bit is taken in the second flip-flop and the check bit is taken from the third flip-flop, thus resulting in a selective insertion of a one symbol time delay as required to obtain node sync.

If the input is on two parallel lines from a BPSK modem, the first bit is shifted into the first flip-flop and the second bit is shifted into the second flip-flop at each bit time. Also, the output of the first flip-flop is shifted into the third flip-flop. The information bit and check bit may now be taken from the three bit register as in the previous case thereby obtaining node sync.

If the input is from a QPSK modem, then the first line is shifted into the first flip-flop. The complement of the first line is shifted into the third flip-flop. The second input line is shifted into the second flip-flop. Code sync is then obtained by using the same set of switches as in the previous case.

The information and check bits are each shifted into a four bit shift register. Every four bit times, the contents of the two shift registers are loaded into two four bit latches. The two four bit latches each drive a quad MECL II to TTL converter. The resulting TTL signals are then delivered to the rest of the decoder.

3.4.4 Received Information Bit Storage. Received Information Bits must be stored in a 72,000 bit long delay line while the

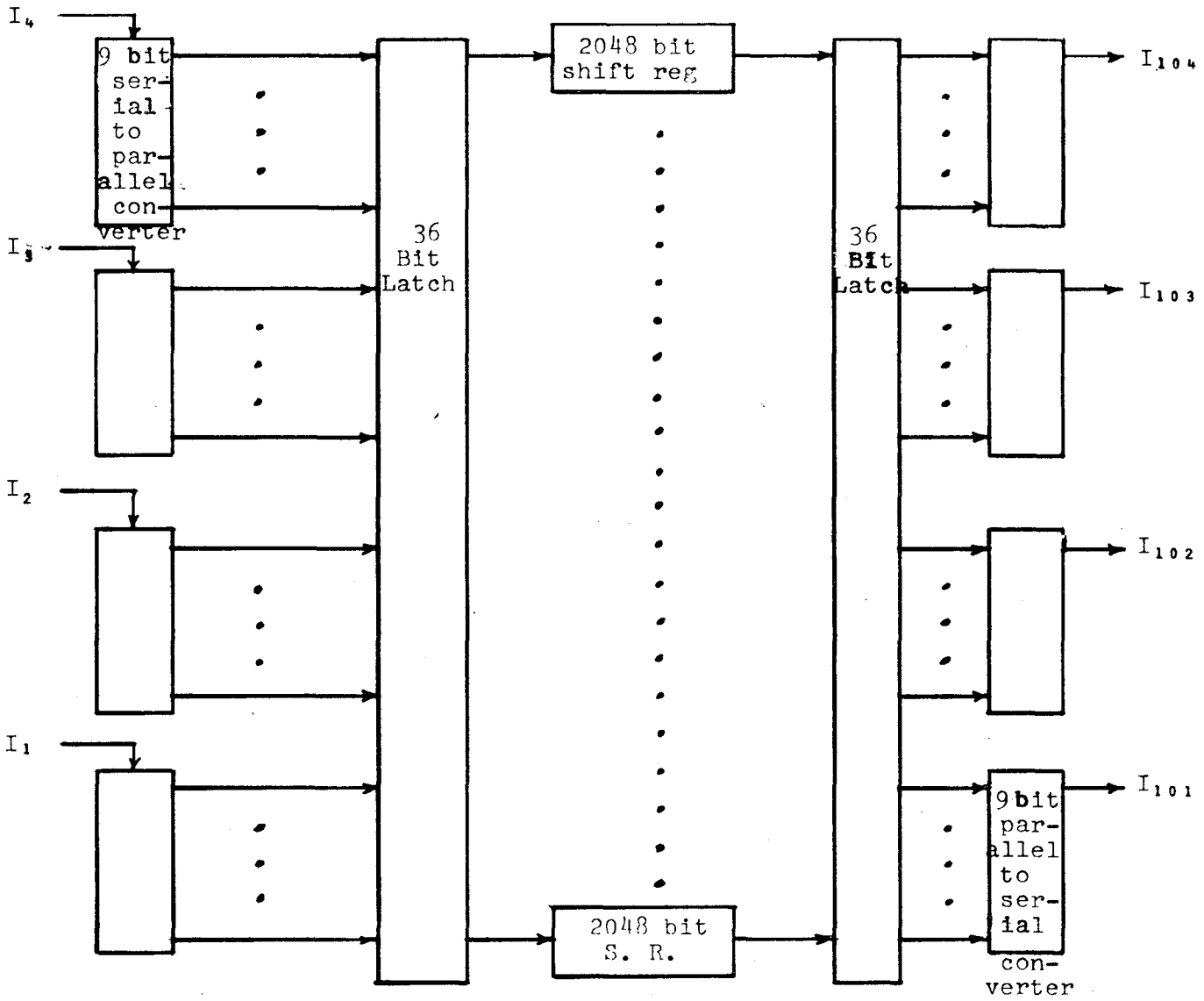

I_4
9 bit serial to parallel converter
I_3
I_2
I_1
36 Bit Latch
2048 bit shift reg
2048 bit S. R.
36 Bit Latch
I_{104}
I_{103}
I_{102}
I_{101}
9 bit parallel to serial converter

Figure 3.4.4

decoding is taking place. A block diagram of the information bit storage is shown in Fig. 3.4.4. This storage is accomplished by using dynamic MOS shift registers, since this form of storage is presently the most inexpensive available. Since the MOS shift registers used are not capable of operating at 40 Megabits, a number of registers must be operated in parallel to obtain this effective speed.

The information bits from the input buffer are collected into 36 bit words, using a serial to parallel converter followed by a 36 bit latch. The output of the 36 bit latch drives 36 2,048 bit long shift registers. The output of the shift registers are clocked into a 36 bit latch. The latch drives a parallel to serial converter which converts the information bits back to four parallel lines. Since the MOS registers are dynamic shift registers, there is a minimum clock rate of 10 KHZ which corresponds to a data rate of 300 kilobits. If it is desired to provide for operation of data rates lower than 300 kilobits, the following technique could be used: Below 300 Kbps, the dynamic shift registers are replaced by static registers of reduced length. The total storage is reduced by a factor of 128. The speed factor - buffer size product remains higher at 300 Kbps than at 40 Mbps.

3.4.5 Syndrome Generator. A logic block diagram of the syndrome generator is shown in Fig. 3.4.5. The syndrome generator receives the information bits and check bits from the input section in two words of four bits each. The syndrome generator

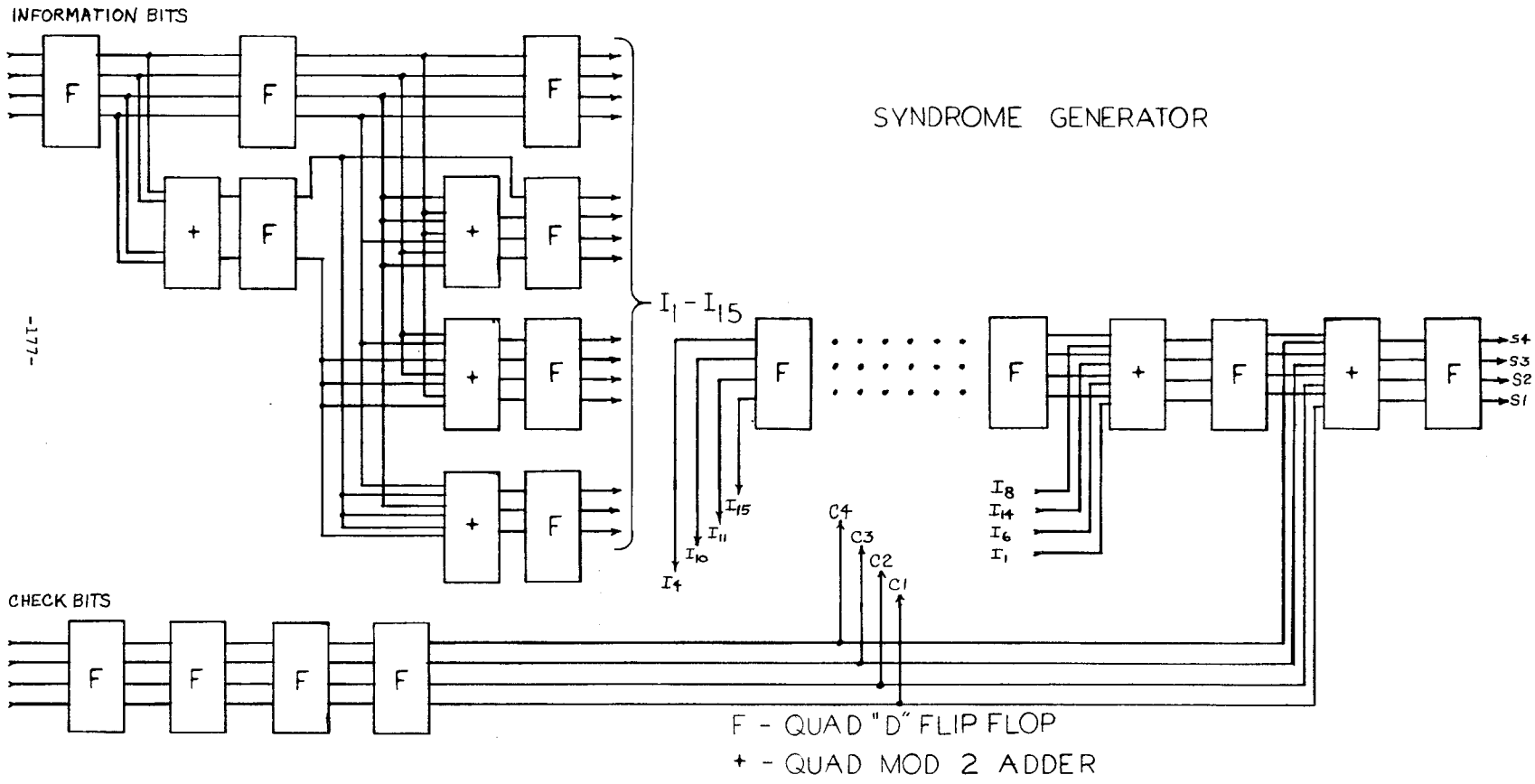

INFORMATION BITS
SYNDROME GENERATOR
F
+
$I_1 - I_{15}$
I_4
I_{10}
I_{11}
I_{15}
I_8
I_{14}
I_6
I_1
C4
C3
C2
C1
S4
S3
S2
S1
CHECK BITS
F - QUAD "D" FLIP FLOP
+ - QUAD MOD 2 ADDER

FIGURE 3.4.5

computes the syndrome, four bits at a time, thereby permitting it to operate at the data rate divided by four. The syndrome generator is implemented using SN7495 shift registers and SN7486 quad modulo two adders. Since this logic is fast enough to permit only one level of mod two addition between the flip-flops, all 16 possible mod two combinations of the four parallel information bits must be computed. This is accomplished in two levels of flip-flops. In the first level, information bits one and two and information bits three and four are each modulo two added and stored in two flip-flops. In the next level, the nine remaining combinations are computed, thus resulting in the signals I-1 thru I-15. The check bit word is delayed three clock times, so that it appears at the appropriate time in respect to the information bits. The remainder of the syndrome generator consists of four, 12 bit long shift registers with a modulo two adder between each stage. One input to each mod two adder comes from the previous stage in the register. The other input comes from one of the I-1 thru I-15 signals. The output of the next to the last stage of the syndrome generator is a set of computed check bits. The syndrome is formed in the last stage of the syndrome generator by exclusive-ORing the four received check bits with the four generated check bits. The resulting syndrome is delivered to the memory.

3.4.6 Decoder Memory. The function of the decoder memory is to store the syndrome while it is awaiting processing by the CPU

section of the decoder. A block diagram of the memory is shown in Fig. 3.4.6. Since the number of computations required to decode one bit is a random variable, and the rate of computations is fixed, a large amount of storage is required in order to make the probability of buffer overflow sufficiently small. Simulations have shown that a memory size of 72,000 bits will provide the required performance with 40 Megabit data and a computation rate of about 10^8 Fano algorithm computations/second. The previous two quantities also set the throughput rate of the memory, since the CPU must be able to access about 10^8 bits per second in order to achieve its maximum computation rate. An additional 40 Megabits per second must be accessed in order to store and retrieve the syndrome input and the information error sequence output; thus, the total throughput requirement of the memory is about 140 Megabits per second.

At the present time, economical semiconductor random-access memories are available with read/write cycle times in the neighborhood of 300 to 400 nanoseconds. If a 400 nanosecond memory is used, then a very wide word is necessary in order to obtain the required throughput rate. Random-access memory cards are presently available from several manufacturers containing 1,024 words of 18 bits each in the required speed range. Combining three of these cards to form a 54 bit word does not quite meet the required throughput rate, consequently four cards will be used, resulting in a 72 bit wide word, in order to achieve the desired througnput rate.

SEQUENTIAL DECODER MEMORY

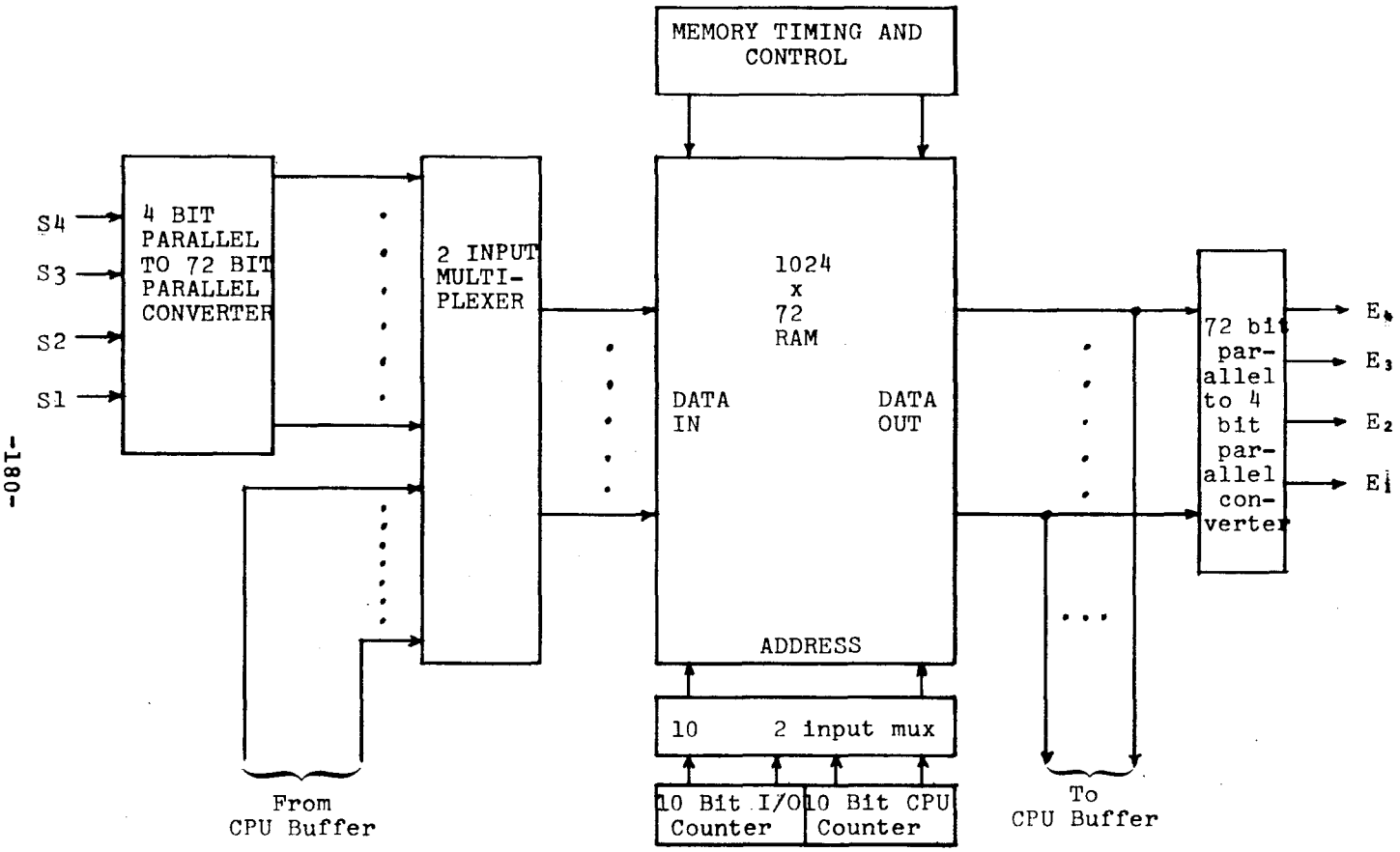
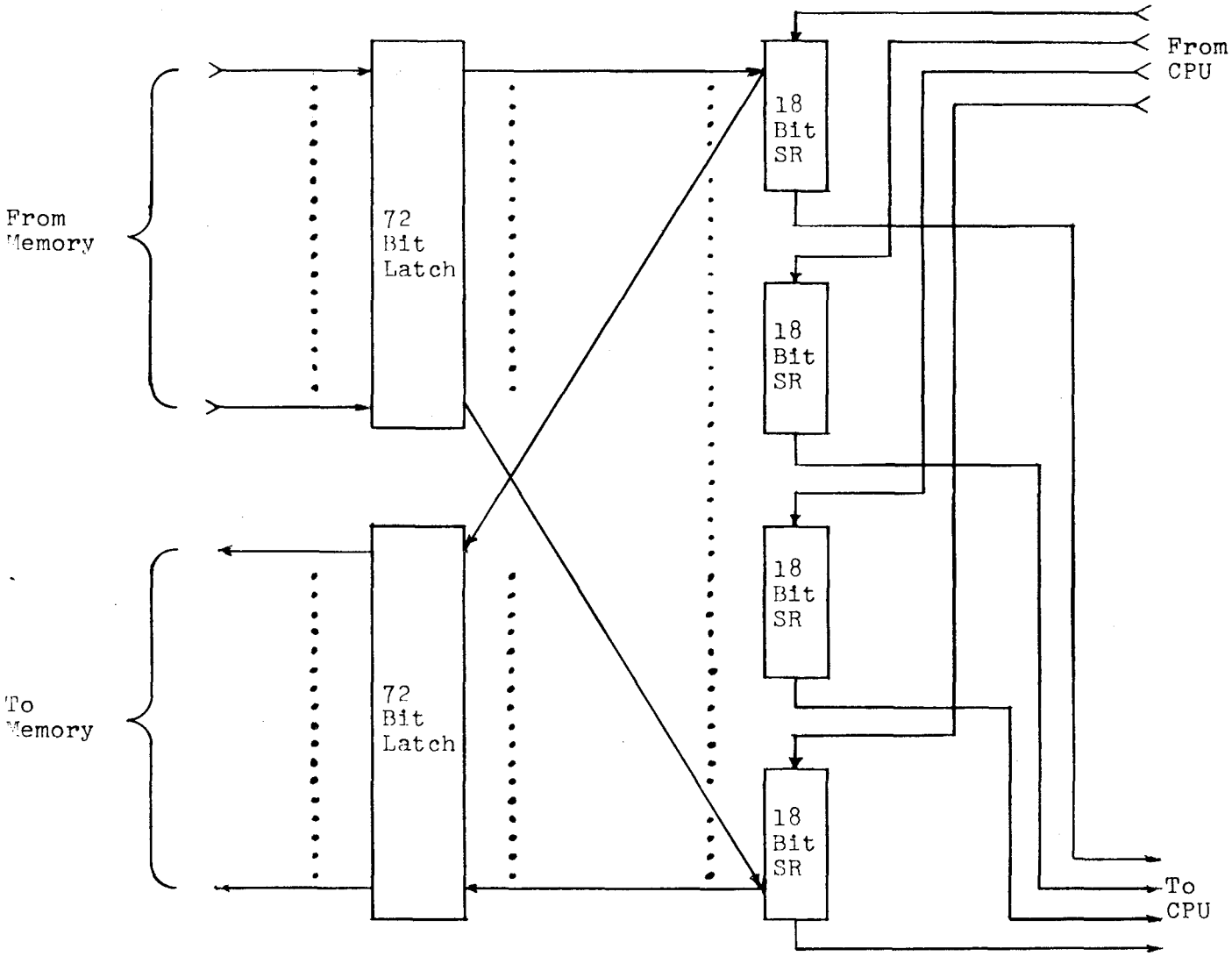


Figure 3.4.6

The memory must have two ports; one port to serve the input-output and the other port to serve the CPU. The input-output port must have priority so that data will not be lost.

The syndrome is collected into 72 bit words for access into the memory. When 72 bits have been collected, the memory timing and control circuit is signalled and the next available cycle is given to the input-output. A 72 bit two input multiplexer selects the syndrome word for data input into the random-access memory. During the cycle, the word stored in the present input-output address is read out and loaded into a 72 bit latch. After the read operation, the 72 bit syndrome word is written into this address and the cycle is completed. The word read into the 72 bit latch is then converted into a four bit parallel line and is sent to the output circuit. When the CPU requires a memory access, the 72 two input multiplexers and the 10 two bit address multiplexers connect the CPU address counter and the CPU data lines to the memory and the cycle is initiated. The memory addresses are stored in two 10 bit counters, one for the input-output and one for the CPU. The memory timing and control circuit controls the read/write process. This circuit also generates a signal whenever a buffer overflow occurs.

3.4.7 CPU Buffer. A block diagram of the CPU buffer is shown in Fig. 3.4.7. The function of the CPU buffer is as follows: When a 72 bit syndrome word is read from the memory, it is stored in the 72 bit latch. When the four 18 bit shift registers are



CPU BUFFER

Figure 3.4.7

empty, the contents of the 72 bit latch are loaded in parallel into the four 18 bit shift registers. The contents of these registers are then shifted out to the CPU four bits at a time. With each shift, a four bit word is shifted into the four 18 bit shift registers from the CPU. Thus, after 18 shifts, the old word will have been shifted into the CPU and a new 18 bit word will have been shifted back into the shift register. At this time, the present contents of the four 18 bit shift registers are loaded in parallel into the second 72 bit latch and the contents of the first 72 bit latch, which now contains a new word, is then loaded into the four 18 bit registers and the process continues. If a new 72 bit word is not yet ready, then the CPU waits until a memory access can be obtained. The latches and the shift registers are implemented using SN7495 devices.

3.4.8 CPU. A block diagram of the CPU is shown in Fig. 3.4.8.1. The four bit words from the CPU buffer are translated from TTL logic levels to MECL logic levels and stored in a four bit latch. The contents of the four bit latch are loaded in parallel into the encoder every four computations when the decoder is proceeding forward to new nodes. A logic diagram of three stages of the encoder is shown in Fig. 3.4.8.2. The encoder is capable of shifting in either direction. The code impulse function can be exclusive-ORed with the present contents of the encoder on a shift in either direction. The encoder may also be synchronously reset. Each stage of the encoder consists of one

flip-flop and one quad two input NOR gate. The output of the right hand side of the encoder is the hypothesized check error sequence. This sequence is stored in the back-up buffer. When the decoder is backing up, the bits that are shifted out of the left hand side of the encoder are stored in the back-up buffer. If an information bit error is hypothesized at a node, then the code impulse function is exclusive-ORed into the encoder, and a one is shifted into the four bit information error register. The information error sequence is also shifted into the back-up buffer when proceeding forward. When backing up, the information error sequence is returned to the four bit register and the original syndrome is reconstituted in the encoder by shifting in the check error sequence and by exclusive-ORing the contents of the encoder with the code impulse function at every node at which an information error was hypothesized. The information error sequence that is shifted out of the back-up buffer when proceeding forward is shifted into a four bit right/left shift register. Every four shifts forward to new nodes, the contents of this register are loaded into the four bit latch. The contents of the latch are then translated from MECL logic levels to TTL logic levels, and returned to the CPU buffer. The function of the back-up buffer is that of a 256 bit long, two bit wide, right/left shift register. Since this size register would be prohibitively expensive if implemented using MECL III logic, the function is actually implemented by the use of a very fast ECL random-access memory. A random-access memory can be made to look

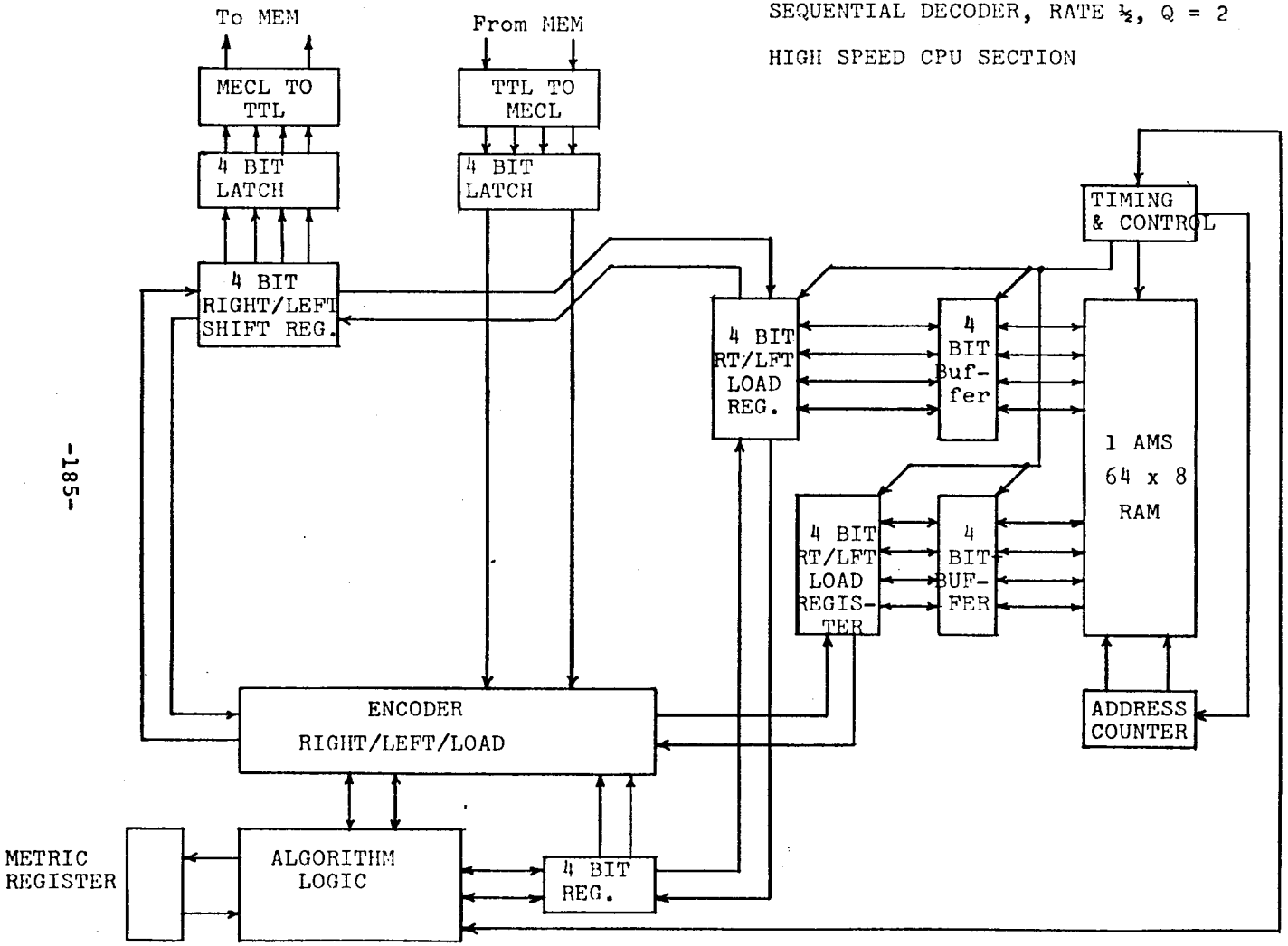
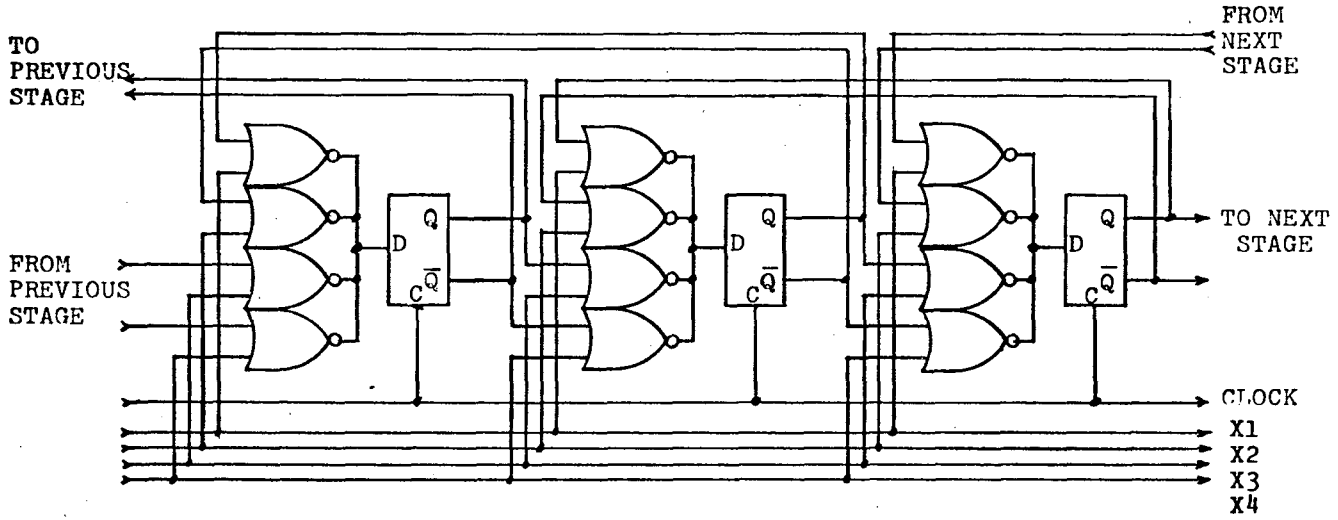

SEQUENTIAL DECODER, RATE ½, Q = 2
HIGH SPEED CPU SECTION
To MEM
From MEM
MECL TO TTL
TTL TO MECL
4 BIT LATCH
4 BIT LATCH
TIMING & CONTROL
4 BIT RIGHT/LEFT SHIFT REG.
4 BIT RT/LFT LOAD REG.
4 BIT Buf-fer
1 AMS 64 x 8 RAM
4 BIT RT/LFT LOAD REGIS-TER
4 BIT BUF-FER
ENCODER RIGHT/LEFT/LOAD
ADDRESS COUNTER
METRIC REGISTER
ALGORITHM LOGIC
4 BIT REG.

Figure 3.4.8.1

CPU RIGHT/LEFT ENCODER



TRUTH TABLE	X1	X2	X3	X4
FOR	1	1	1	0
FOR. EOR	1	1	0	1
BACK	1	0	1	1
BACK. EOR	0	1	1	1
RESET	1	1	1	1

Figure 3.4.8.2

like a right/left shift register if the random-access memory is addressed by an up-down counter, which counts up when a right shift is desired and counts down when a left shift is desired. The RAM is a 64 word, eight bits per word, ten nanosecond access time memory that is manufactured by Advanced Memory Systems. The RAM is interfaced to the CPU through the two, four bit, right/left, parallel load registers and the two, four bit, buffer registers. A six bit up-down counter provides the address.

The algorithm logic controls the progress of the decoder through the decoding tree. Each computation the algorithm logic decides, 1) whether to move forward, backward, or sideways, 2) whether the node being computed has an information bit error, 3) whether to raise or lower the threshold, and 4) the amount by which the decoding metric should change. Two infrequently used functions are:

1) back-up buffer overflow, and

2) main buffer overflow

When back-up overflow occurs, the decoder has come to the end of the back-up buffer and can go no further in a back search. In this situation, the algorithm logic lowers the threshold and goes back to forward searching. In the case of a main buffer overflow, the decoder must attempt to restart at some point ahead of its present location. The decoder jumps to the front of the back-up buffer, requests a new word from the main buffer, resets the encoder, resets the metric register, and attempts to begin decoding in the forward direction. If the decoder is unable to restart,

the main buffer will soon overflow again and another restart attempt will be made.

3.4.9 Physical Description. The CPU is implemented using MECL III logic. Approximately 250 MECL III IC's are required. The CPU can be packaged on one or two multilayer circuit boards. Five or six layers will be required. Two layers are reserved for ground and power planes. The remaining layers are required to provide the necessary interconnections. All circuit board runs longer than 2 inches are terminated strip transmission lines of 50 ohm impedance.

The MECL III stud mounted flat package is used. The packages are mounted with the stud passing through the circuit board. Cooling can be provided by soldering a U shaped fin to the stud on the side of the board opposite the component. Cooling air will then be blown across the fins. Using this technique, the temperature rise in the equipment will be less than 20 degrees. With 50°C ambient temperature, the maximum component temperature will then be 70°C.

The remainder of the decoder is implemented using MECL II, TTL and MOS logic devices. Approximately 325 of these devices are required. They can be mounted on five or six circuit boards, using the welded stitch wire technique. The main buffer can be packaged on four additional circuit boards, and the back-up buffer can be packaged on one circuit board.

The complete decoder, together with power supplies, can be

packaged in a standard 10½" high rack mountable chassis. All data input and output connectors can be mounted on the front panel together with all necessary controls.

3.4.10 Modifications Required by Soft Decisions. As explained in previous sections, an improvement in E_b/N_0 of 1.5 db is available by increasing Q from 2 to 4. But the cost is a substantial increase in the hardware requirement.

A soft decision syndrome sequential decoder would form the syndrome as in the hard decision case by regarding the most significant bit of each symbol as a hard decision. This is most easily done if the quantizer levels are labelled by a sign-magnitude representation. The sign bit then corresponds to a hard decision.

The syndrome is stored in the main memory along with the magnitude (or quality) bits for each symbol. This requirement triples the size of the main memory for Q = 4 since a syndrome bit, and one quality bit for each symbol must be stored for each node. The hard decision bit for the information bits is stored in a delay line as in the case of the hard decision case. This requirement causes the hardware in the memory and I/O section of the decoder to increase by a factor of 2.5.

The CPU uses the syndrome and quality bits to determine a likely information error sequence. This sequence is then stored in the main memory for later use in correcting the delayed information bit stream. All bit paths in the CPU must now be three

times wider than for a hard decision decoder, thus, tripling this portion of the CPU hardware. The algorithm logic becomes considerably more complicated since the metrics must be quantized to a much finer level than in the hard decision decoder. This results not only in an increased number of gates, but in an increased number of logic levels, thus slowing down the computation rate. The computation rate will be reduced from about 100 Megacomputations per second in the hard decision case to about 70 Megacomputations for Q = 4. The CPU hardware will increase by a factor of three. Thus the overall increase in hardware (and in cost) will be a factor of 2.8 relative to a hard decision decoder.

REFERENCES

SECTION 3

1. Savage, J. E., "The Computation Problem with Sequential Decoding," Ph.D. Thesis, Department of Electrical Engineering, MIT, February 1965.

2. Jacobs, I.M., and E. R. Berlekamp, "A Lower Bound to the Distribution of Computations for Seguential Decoding," IEEE Transactions on Information Theory, Vol. IT-13, April 1967.

3. Codex Corporation, Final Report on High-Speed Sequential Decoder Study, Contract DAAB07-68-C-0093, U.S. Army Satellite Communication Agency, Fort Monmouth, New Jersey, 1968.

4. Bucher, E. A., and J. A. Heller, "Error Probability Bounds for Systematic Convolutional Codes," IEEE Transactions on Information Theory, IT-16, Number 2, March 1970.

5. Bussgang, J. J., "Some Properties of Binary Convolutional Code Generators," IEEE Transactions on Information Theory IT-11, 1965.

6. Massey, J. L., "Quick-Look Convolutional Codes," National Aeronautics and Space Administration Coding Conference, Jet Propulsion Laboratory, Pasadena, 1970.

7. Lin, S., and H. Lyne, "Some Results on Binary Convolutional Code Generators," IEEE Transactions on Information Theory, IT-13, pp. 134-139, 1967.

4.0 CODING FOR DATA OF VARYING SPEED AND ERROR RATE REQUIREMENTS

Consideration has been given to the problem of transmitting a data stream on which, by time-division multiplexing, data from sources with different data rates and error requirements have been combined. Three approaches have been considered and analytical results obtained for each. These are designed to protect the high-reliability low-rate data by one of the following methods:

a) concatenated coding (originally outlined in section 4 of the proposal for this contract.)

b) lengthened symbol times for the low-rate data

c) use of lower rate codes for the low-rate data

To obtain specific results, it was assumed that the ratio of high-rate to low-rate is no greater than 10 and that the required high-rate bit error probability is 10^{-3}, which probably represents a worst case.

Also, the basic code for the high-rate data was taken to be the best rate 1/2 constraint length 4 convolutional code. While this is used primarily because it has been thoroughly analyzed and simulated, it is also a reasonable candidate for data rates above 10 Mbps with Viterbi decoding.

4.1 Concatenated Coding. The goal is to find a very simple outer code to be used on the low data-rate source only, which will decrease the error probability to the desired level. The overall coding system is shown in Fig. 4.1. If the outer code has rate 1/2, the overall E_b/N_0 is increased only by 0.4 db.

In order to render the inner code errors nearly independent, interleaving must be introduced. Five constraint lengths of interleaving seem more than sufficient. Thus, an interleaving memory of 20 bits is all that is required for constraint length 4.

The outer code then operates essentially on a binary-symmetric channel with crossover probability $p=10^{-3}$. A two-error correcting BCH code is unsatisfactory because it requires R=7/15 and only achieves

$$P_B \approx \frac{7}{15} \binom{15}{3} p^3 \approx 2.1 \times 10^{-7}$$

A more satisfactory block code, the Golay (24, 12) three-error correcting code has R=1/2 and

$$P_B \approx \frac{1}{2} \binom{8}{24} \binom{24}{4} p^4 \approx 1771p^4 = 1.7 \times 10^{-9}$$

However, this requires a moderately complex decoder when operating at high data rates. Block code synchronization may add to the complexity if not already provided by a separate framing reference.

A more suitable approach is to use a convolutional outer code. The two-error correcting, rate 1/2 code (Fig. 4.2a) yields for the BSC

$$P_B \approx 50p^3 \approx 5 \times 10^{-8}$$

while the best three-error-correcting rate 1/2 code yields (Fig. 4.2b)

$$P_B \approx 560p^4 \approx 5.6 \times 10^{-10}$$

In each case, the results are better than the corresponding block codes, the decoder is simpler, and code synchronization is not required for the convolutional code. Node synchronization, which is required, can be obtained using the methods outlined in section 2.2.5.

While Viterbi decoding is required to obtain the above results, a feedback decoder, the L-1011 presently marketed by LINKABIT Corporation, obtains nearly equivalent results for the BSC, and affords a much simpler mechanization for both the two- and three-error correcting decoders.

4.2 Lengthened Symbol Times for Low-Rate Data. This is undoubtedly the simplest approach. If a rate 1/2 outer code were used for the low-rate data, its effective E_b/N_0 would be increased by 3 db (resulting in an overall increase of 0.4 db). Rather than using a code, the low rate symbols might simply be repeated or lengthened. To achieve $P_B=10^{-3}$ with a K=4 code for the high-rate data, the required E_b/N_0 = 3.75 db as established by analysis and simulation. Thus, using this approach for the low-rate data, we would have E_b/N_0 = 6.75 db. For high E_b/N_0, the bit error probability (without quantization) for this code is asymptotically

$$P_B \sim 2 \operatorname{erfc} \sqrt{6E_b/N_0} \approx 1.1 \times 10^{-7}$$

This figure may be slightly optimistic if 8-level quantization is

used.

4.3 Lower Rate Codes. Instead of doubling the symbol length, we may keep this fixed but double the number of symbols, thus changing from an R=1/2 code for the high-rate data to an R=1/4 code for the low-rate data. Preliminary analysis indicates that the rate 1/4 constraint length 4 code with minimum bit error probability at high E_b/N_0 has the generator matrix

$$G = \begin{bmatrix} 1111 \\ 1011 \\ 1101 \\ 1101 \end{bmatrix}$$

and that for this code at $E_b/N_0 = 6.75$ db

$$P_B \sim 4 \operatorname{erfc} \sqrt{6.5 E_b/N_0} \approx 6.5 \times 10^{-8}$$

If such a lower-rate code is used for the low-rate data, separate decoders must be used for the high-and low-rate data. The two decoders may time-share some subsystems, such as the arithmetic unit, but in some respects, they must be distinct. In view of the very modest improvement, it is questionable whether the complexity is warranted relative to the approach of doubling the symbol time, which requires virtually no additional decoding complexity.

On the other hand, it also appears that concatenation with a simple convolutional outer code gains about two orders of magnitude in performance over the other two approaches.

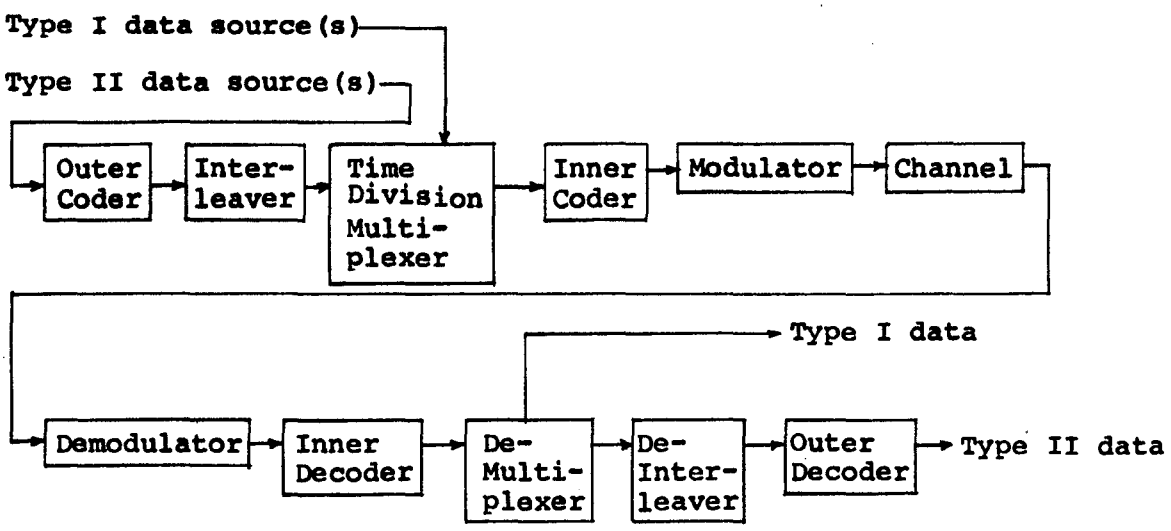
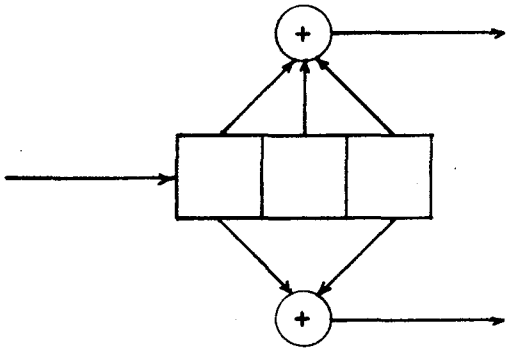
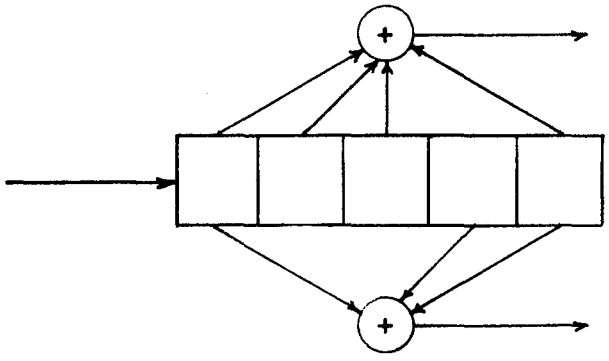


Figure 4.1. Coding for data of varying error-rate requirements.



(a) two-error correcting



(b) three-error correcting

Figure 4.2. Outer Convolutional Code

REFERENCES

SECTION 4

1. A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," LINKABIT Corporation Seminar on Convolutional Codes, January 26, 1970.

5.0 PREDECODING FOR A SEQUENTIAL DECODER: A HYBRID IMPLEMENTATION FOR VERY LOW ERROR PROBABILITY.*

5.1 Introduction. To achieve very low error rates with a sequential decoder at very high data speeds, it is necessary to operate well below R_{comp}. For at high speeds, even a speed factor of 2.5 may be prohibitively expensive, especially in a soft quantized sequential decoder. Furthermore, for very low sequential overflow probability, large quantities of storage are required, particularly with 8-level quantization which requires six bits of storage per branch.

On the other hand, if we operate well below R_{comp}, most of the data can be correctly decoded by a short constraint length Viterbi decoder. As was pointed out in the original description of the algorithm (Ref. 1), a Viterbi decoder can decode a long constraint length (K) convolutional code treating it as if the constraint length were much shorter ($k<<K$), by operating only on the first k symbols of the convolutional code generators. Of course, when an error occurs, remerging to the correct path is extremely unlikely and generally all subsequent bits will be decoded incorrectly. The point is, however, that most of the data can be correctly decoded in this way and only the more difficult (noisy) segments of data are incorrectly decoded, and if errors can be detected, then segments can be passed on to a more powerful sequential decoder.

---

*This technique was proposed by G. D. Forney, Jr., who was a consultant on this study.

The approach that is therefore suggested is to use a short constraint length Viterbi decoder (k=5) to "predecode" a long constraint length convolutional code, detecting the incorrectly decoded segments, and passing these on to a sequential decoder, which is more powerful since it utilizes the full constraint length (K>30) of the code. The mechanization block diagram is shown in Figure 5.1.

We assume that data is encoded, in frames of 1000 bits, into a constraint length K=40, rate 1/2 nonsystematic convolutional code, each frame being followed by a tail of 39 known branches, to be used for resynchronization. The received demodulated data (soft or hard quantized) is passed first to a Viterbi predecoder operating as a decoder for a k=5 code corresponding to the first 5 symbols of the generator sequences. All decoded data is passed to a long digital delay line capable of storing D=256 decoded frames (256 K bits).

All undecoded received data from a given frame is also passed to a one-frame buffer (6000 bits for 8-level soft quantized data*; 1000 syndrome bits for hard quantized data). The Viterbi decoder output is also monitored in an effort to detect all frames in which an error occurred. This can be performed in a number of ways. The probability that an incorrect path remerges with the correct path at any given node is of the order of 2^{-K}. Hence, with K=40,

---

*This can be reduced to 5000 bits if we use a soft decision syndrome decoder, as discussed in Section 3.3.

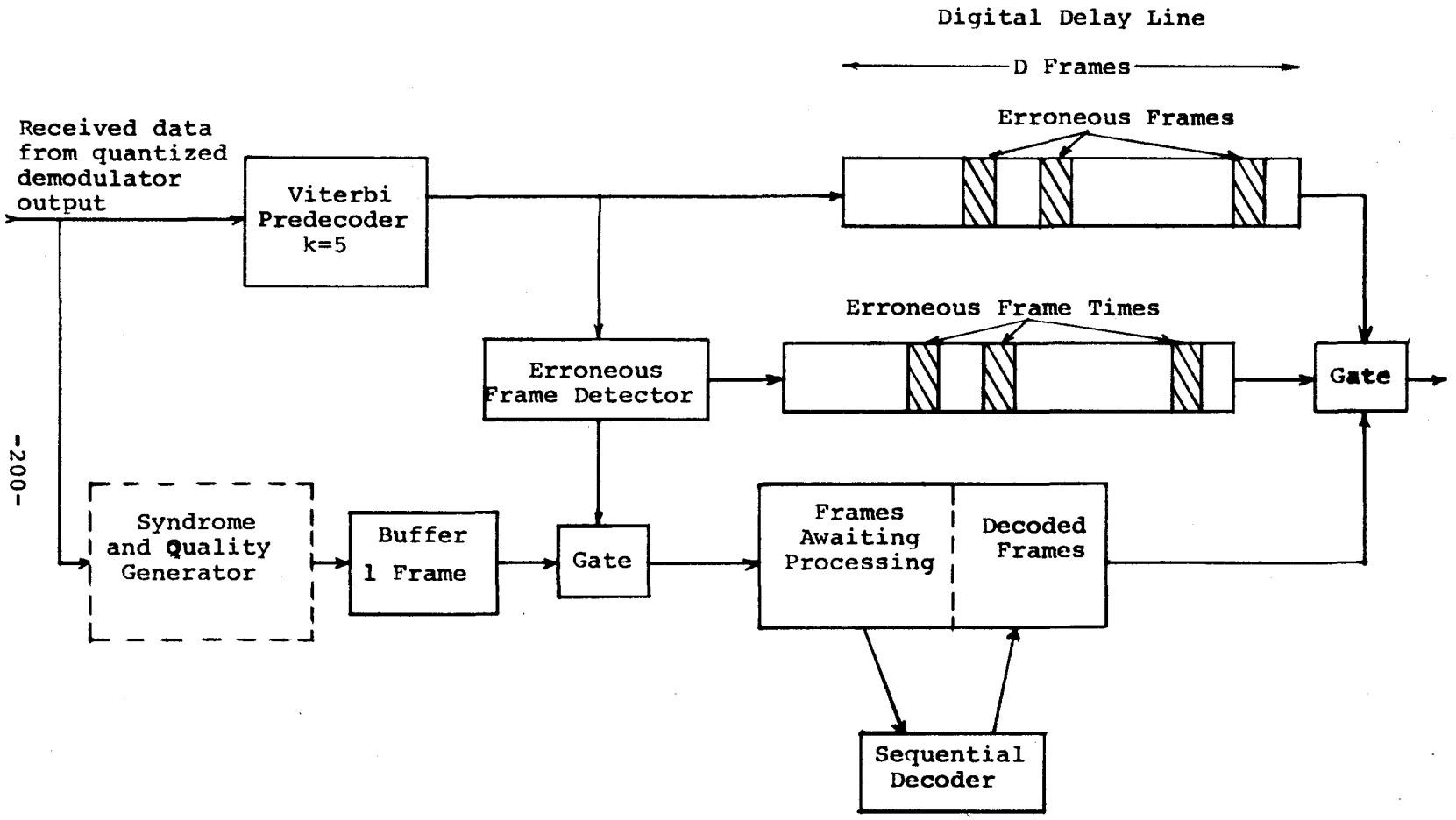


Fig. 5.1 Hybrid Implementation of Sequential Decoder and Viterbi Predecoder.

it will remerge at any one of 10^3 nodes with probability on the order of 10^{-9}. By merely observing K-k-1=34, decoded tail branches and comparing them with their known values, the probability that an error will not be detected is much less than 10^{-8}.

When an erroneous frame is detected, it is tagged for future reference and gated to a large B frame buffer to await processing by the sequential decoder. This buffer is divided into two segments. In the front portion is stored data from frames awaiting sequential decoding. In the rear portion are the already sequentially decoded frames awaiting insertion into the decoded data stream as it exits from the delay line. In all, there is storage available for B undecoded and B decoded frames.

There remain the problems of determining a) the percentage of deletions, which establishes the sequential decoder load and buffer size requirements, b) the computational complexity of sequential decoding of the erroneously predecoded frames, and c) the overflow probabilities due to finite delay D and finite buffer size B.

We shall consider these three problems in the state order.

5.2 Deletion Probabilities of Predecoder. Deletions correspond to first event errors in a short constraint length convolutional code decoded by a Viterbi decoder. With soft quantization (Q=8), a rate 1/2 k=5 code was simulated at an E_b/N_0=4.5 db. Out of 1500 frames of 852 bits each, 32 were erased resulting in an estimated erasure probability $p_\phi \approx .02$. Extrapolating to 1000 bit

frames, we estimate conservatively a deletion probability no greater than p_ϕ=.025. On the other hand, with E_b/N_0=4 db, the deletion probability exceeds 8 percent.

With hard quantization, simulations were run at a crossover probability p=.025 which corresponds to E_b/N_0=5.7 db. The result of decoding approximately 5000 frames of 500 bits each was a deletion probability $p_\phi \approx .10$. A particularly short trellis memory (7 branches) was employed in order to make a resulting implementation particularly simple and inexpensive. In the following examples, we shall consider operating a hard quantized system at $E_b/N_0 = 6.2$ where p=.02 and the Pareto exponent is 2.0. However, since we use longer frames (1000 bits), we shall use an estimated deletion rate $p_\phi \approx .10$ even at this higher E_b/N_0 value.

5.3 Computational Complexity of Sequential Decoding for Erroneously Predecoded Deleted Frames. Extensive simulations were performed on hard quantized data only. First 14,000 frames with crossover probability p=.025 were sequentially decoded using the Fano algorithm with quick threshold loosening. The resulting distribution of computations was very closely approximated by the Pareto distribution with exponent ρ=1.67, which follows precisely from the theoretical result

$$\frac{E_o(\rho)}{R} = \rho$$

since R=1/2 and $E_o(1.67)\Big|_{p=.025} = 0.83$

Then with p.025, the 466 erroneously predecoded frames (approximately 10% of the total--see previous section), were sequentially decoded. The distribution of computations was in all cases above the previous one, and at the high end it approached between 8 and 10 times the ordinary distribution. In retrospect, this is exactly as expected. Predecoding will correctly decode all the easier cases but generally fail on most frames which require longer computation searches in sequential decoding. Now suppose that it fails on all the long computation searches (say, above 1000 computations/bit) and succeeds whenever such long searches are not present in a frame. Then the incorrectly decoded frames will contain all of these long searches, and since sample size is reduced by a factor of 10, the probability distribution of long computations is raised by a factor of 10.

In any case, we shall be upper bounding the decoding complexity for the incorrectly decoded frames if we use the ordinary Pareto distribution divided by p_ϕ, the frame deletion (or incorrect predecoding) probability.

Also of importance is the fact that for ordinary sequential decoding, the average number of computations per bit was 1.245, while for the 10% of the frames which were incorrectly predecoded, this rose to only 1.89 computation per bit. Thus, even though the tail of the distribution rose by a factor of 10, the average

computation effort rose by only a factor of 1.5.

5.4 Overflow Probability of Hybrid Implementation. Overflow with resulting deletion or errors can occur in either of two ways:

a) A search is so long that an incorrectly predecoded (deleted) frame reaches the end of delay line D prior to the completion of its processing by the sequential decoder;

b) While a long search is proceeding on a given frame, the sequential decoder buffer fills up with B frames awaiting processing and cannot accept any new deleted frames.

We assume that normally the buffer is nearly empty so that successive overflows are essentially independent. This is justified for sufficiently large Pareto exponents ($\rho \geq 2$), which we shall insure. Then for an initially empty buffer, the probability of overflow for any given frame is $P_0 = p_\phi$ [Pr (overflow on a deleted frame)]

$$P_0 = p_\phi \sum_{K=B}^{D} \Pr\left(\begin{array}{l}\text{more than B}\\ \text{deleted frames}\\ \text{out of K frames}\end{array}\,\middle|\,\begin{array}{l}\text{sequential decoding}\\ \text{lasts K frame times}\end{array}\right)$$

$$\cdot \Pr\left(\begin{array}{l}\text{sequential decoding of}\\ \text{frame lasts K frame times}\end{array}\right)$$

$$+ p_\phi \Pr\left(\begin{array}{l}\text{sequential decoding of given}\\ \text{frame lasts D frame times}\end{array}\right)$$

where p_ϕ is the probability of deleting (incorrectly predecoding) a frame.* It is shown in Appendix C that

$$P_0 \lesssim \frac{\exp\,(Bp_\phi/2)}{Bp_\phi} \frac{10^{3(1-\rho)}}{(\mu B/p_\phi)^\rho} + \frac{10^{3(1-\rho)}}{(\mu D)^\rho} \qquad (5.1)$$

provided $p_\phi < B/2D$.

where p_ϕ = deletion probability

B = buffer size in frames

D = delay line size in frames

10^3 = frame length in bits

ρ = Pareto exponent

μ = speed factor

Thus, it appears that the buffer size is effectively increased by a factor of $1/p_\phi$.

5.5 System Analysis of Possible Hybrid Implementation. We now analyze both soft quantized and hard quantized hybrid systems, and compare each with ordinary sequential decoders. Our goal will be to achieve an error probability on the order of 10^{-8}. In order to establish a basis for comparison, we assume that in each case we have available a digital delay line of length 256 K bits and a sequential decoder buffer capable of storing 64 K bits. Also we assume that the data speed is so high that $\mu=2.5$ computa-

---

*Note, a correctly predecoded frame naturally cannot overflow since it is not processed by the sequential decoder. hence, the overall overflow probability is p_ϕ times the overflow probability for deleted frames. In particular, if we require $P_0 \simeq 10^{-8}$ with a $p_\phi \simeq 0.1$, the overflow probability for deleted frames must only be no greater than 10^{-7}.

tions per bit is the highest speed factor feasible.

Thus in each case D=256. For a soft quantized (Q=8) system B=64/6 = 10.7 (since each branch requires 6 bits of storage), while for a hard quantized (Q=2) system, B=64 (since only one syndrome bit needs to be stored per branch, and either parity bit is inserted in the delay line for the deleted frames).

We consider first the requirements of ordinary (non-hybrid) sequential decoding. For soft quantization, we assume a K=25 rate 1/2 nonsystematic code blocked in 1000 bit frames separated by 24 branch blocks of known symbols. Then the overflow probability (error probability if deletions are treated as errors) is

$$P_0 \lesssim \frac{10^3}{(10^3 B\mu)^{\rho}}$$

with B=10.7 and μ=2.5, it is clear that we must have ρ=2.5 for $P_0 \lesssim 10^{-8}$. This corresponds to $E_b/N_0 \simeq 4.7$ db. An additional 0.1 db loss results from the resynchronizing sequence of 24 bits for every 1000 data bits. The undetected error probability is approximately $P_E \lesssim 10^3\, 2^{-kR_{comp}/R}$ Since R_{comp} corresponds to $E_b/N_0 \simeq 2.6$ db and K=25, we find $P_E \lesssim 10^{-9}$. Thus, the error rate due to both deletions and undetected errors is less than 10^{-8}.

For the hard quantized ordinary sequential decoder, we take the automatic resynchronization implementation described in Section 3.2.2.1 which does not require framing. We take either a K=25 nonsystematic code or a K=45 systematic code. Here B=64

since we store only the syndrome, using the delay line with D=64 to store the information bit (for a systematic code) or a parity bit (for a nonsystematic code). Then as is shown in Section 3.2, the bit error probability due to overflows is roughly

$$P_B \leq \frac{200}{(10^3 B\mu)^{\rho}}$$

If μ=2.5 and B.64, it is clear that with ρ=2, we have $P_B \leq 8\times10^{-9}$. This corresponds to E_b/N_0 = 6.2 db. The undetected error probability is of the same order of magnitude.

We now turn to hybrid implementations, beginning with soft quantization. Having fixed the buffer and delay line sizes, the only parameters to be varied are speed factor μ and E_b/N_0, which establishes the Pareto exponent, ρ. Clearly to minimize cost of implementation, we should try to minimize μ. On the basis of average number of computations, it would appear that μ could be reduced almost in proportion to p_ϕ, the deletion rate, since only a fraction p_ϕ of the frames must be processed by the sequential decoder and the average number of computations only rises slightly for these frames.

However, it appears from (5.1) that the minimum value of μ is limited by the magnitude of p_ϕ, B, D. We indicated in Section 5.2 that with Q=8 and E_b/N_0 = 4.5 db, the deletion rate $p_\phi \approx .025$ and that lower E_b/N_0 results in greatly increased deletion rates. Also sequential decoding with Q=8 and E_b/N_0=4.5 db results in a Pareto exponent ρ=2.2 (Ref. 2). Then with D=256, B=10.7, ρ=2.2,

it is possible to make $\mu=0.4$ and achieve $P_0 \simeq 10^{-8}$. With K=40, the undetected error probability is well below this. Thus, we have reduced the speed factor by 6. If we made $E_b/N_0 = 4.7$ db with a corresponding $\rho=2.5$ (as for the ordinary sequential decoder above, we could make $\mu=0.1$ with a corresponding saving of a factor of 24 in speed).

For hard quantized hybrid decoding, we have D=256, B=64. If we take $\rho=2$ corresponding to crossover p=.02 and $E_b/N_0=6.2$ db, we have estimated in Section 5.2 that $p_\phi \simeq .10$. Then it appears from equation (5.1) that we can make μ no less than 1.25 with a resulting $P_0 \simeq 10^{-8}$. In every case, K=40 requires an additional 0.2 db for resynchronization. These results are summarized in Table 5.1.

5.6 Conclusions. With soft quantization (Q=8), we have found that the principal advantage of hybrid decoding is the saving of a factor of 6 in speed factor and 0.2 db in E_b/N_0 or a factor of 24 in speed with no gain in E_b/N_0. This is due primarily to the fact that for undeleted frames only 1 bit/branch storage is required rather than 6 bits/branch. For a soft quantized sequential decoder operating on data transmitted at speeds of 20 Mbps, such a saving is crucial for feasibility and cost, since soft quantized metric computations need be made only at 8 MHz (for $\mu=.4$) or 2 MHz (for $\mu=.1$) speeds rather than 50 MHz (for $\mu=2.5$), as would be required with ordinary sequential decoding.

TABLE 5.1

SUMMARY OF HYBRID AND SEQUENTIAL DECODER IMPLEMENTATION FOR $P_E = 10^{-8}$

D = 256 K bit delay, B = 64 K bit buffer

	Soft Quantization, Q=8			Hard Quantization			
	Speed Factor μ	Pareto Exponent ρ	E_b/N_0 (db)	Speed Factor μ	Pareto Exponent ρ	p	E_b/N_0 (db)
Sequential Decoder K=25 (Nonsys-tematic)	2.5	2.5	4.7 + 0.1	2.5	2	.02	6.2*
Hybrid (Viterbi Predecoder k=5)	0.4	either .2.2	4.5 + 0.2	1.25	2	.02	6.2 + 0.2
K=40 (Non-systematic)	0.1	2.5	4.7 + 0.2				

*No framing required

On the other hand, with hard quantization (Q=2), only the moderate saving of a factor of 2 in computation speed is achieved, primarily because the storage saving is not nearly as great. The only advantage is that afforded by quadrupling delay line size. Further increase of D will improve matters, of course. For example, we might increase D to 512 frames (512 K bits) and thereby achieve a reduction of speed factor to μ=0.6 (an overall reduction of a factor of 4 in speed). Further significant reduction of μ for hard quantized data is not feasible, since the deletion rate is p_ϕ = 0.1 but the average computation rate for sequential decoding of deleted frames is 1.5 times the average for all frames; thus, the speed can certainly not be reduced by more than a factor of 6 without degrading performance. (For soft quantization, the much lower deletion rate p_ϕ = 0.025 made a much greater speed reduction possible.)

Hybrid decoding increases complexity in three ways. It requires:

a) A predecoder operating at the data speed, and frame error detection equipment.

b) A long delay line is required (the cost of such serial storage represents a small increment).

c) Blocking of data and reinsertion synchronization.

For a soft quantized sequential decoder, blocking of data is probably required in any case, and the price of (a) and (b) above is small indeed for a speed factor reduction of an order

of magnitude which it gains. In fact, at data speeds above 20 Mbps, this may be the only way to achieve 10^{-8} error rates with the E_b/N_0 advantage of soft quantized data.

On the other hand, with a hard quantized sequential decoder blocking of data is not required. The moderate speed factor advantage may not be sufficient to justify the costs of the hybrid system.

REFERENCES

SECTION 5

1. A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," IEEE Transactions on Information Theory, Vol. IT-13, Number 2, April, 1967.

2. I. M. Jacobs, "Sequential Decoding for Efficient Communication from Deep Space," IEEE Transactions on Communication Technology, COM-15, No. 4, August, 1967.

APPENDIX A

Computational Technique for Evaluation of Convolutional Code Performance

The calculation of a convolutional code transfer function essentially involves the inversion of the code transfer matrix. For the K=3, rate 1/2 code shown in Fig. 2.2.1 we have the following linear relations among nodes, or states of the diagram:

$$\begin{bmatrix} x_{10} \\ x_{11} \\ x_{01} \end{bmatrix} = \begin{bmatrix} 0 & 0 & N \\ ND & ND & 0 \\ D & D & 0 \end{bmatrix} \cdot \begin{bmatrix} x_{10} \\ x_{11} \\ x_{01} \end{bmatrix} + \begin{bmatrix} ND^2 \\ 0 \\ 0 \end{bmatrix}$$

and $T(N,D) = D^2 \, x_{01} \; (N,D)$ (A.1)

Thus denoting the state column vector, $\underset{\sim}{x}$, the transfer matrix A, and the column vector $(1,00\ldots0) = \underset{\sim}{1}$, we find that in general * we must solve

$$[I - A] \; \underset{\sim}{x} = ND^{\delta_i}(\underset{\sim}{1})$$

*In this case $\delta_i=2$; in general the best binary code for any K and R will have all 1's in the first branch, corresponding to $\delta_i=1/R$.

where δ_i is the weight of the initial branch. The magnitude of the eigenvalues of the matrix A are all less than unity, for otherwise the code can be shown to be catastrophic. Consequently, the inverse of I-A exists and

$$\underset{\sim}{x} = ND^{\delta_i} [I-A]^{-1} \underset{\sim}{1}$$

$$= ND^{\delta_i} [I+A + A^2 + \ldots + A^n + \ldots] \underset{\sim}{1} \qquad (A.2)$$

Finally it follows from eq. A.3 that the transfer function is the scalar bilinear form

$$T(N,D) = ND^{(\delta_i+\delta_f)} \underset{\sim}{1}' [I+A + A^2 + \ldots + A^n + \ldots] \underset{\sim}{1} \qquad (A.3)$$

where $1' = (0,0,\ldots 1)$ is a row vector and δ_f is the weight of the final branch.

In general, the normalized truncation error is bounded by

$$|\epsilon_n| / ND^{(\delta_i+\delta_f)} < \sum_{k=n}^{\infty} ||A||^k = \frac{||A||^n}{1-||A||}$$

The norm of a matrix is just the magnitude of its largest eigenvalue, which must be less than unity as noted above. Obviously, for a given n this is a decreasing function of D.

The first-event error probability of eq. 2.2.5 is obtained directly from A.3 by setting N=1 and D at the channel parameter values desired. The bit error probability of eq. 2.2.6 is obtained by numerically differentiating eq. A.3 at N=1; that is, by approximating the derivative by

$$\left.\frac{dT(N,D)}{dn}\right|_{N=1} \simeq \frac{T(1+\varepsilon,D) - T(1,D)}{\varepsilon} \qquad (A.4)$$

Since T(N,D) is a polynomial in N with positive coefficients, the second derivative is always positive. Consequently, taking $\varepsilon>0$, yields an upper bound on the derivative while $\varepsilon<0$ yields a lower bound.

In general, multiplying matrices of dimension 2^K is a very lengthy numerical procedure. But here we note that actually all that is required is successive multiplication of matrices by a vector, since the nth (vector) term of eq. A.2 is obtained from the (n-1)st by

$$\underset{\sim}{A^n 1} = A \cdot \underset{\sim}{A^{n-1} 1}$$

Also, the matrix A has at most, 2 nonzero entries per row an in all 2^K-3 entries. Thus the total number of multiplications required in computing the first n terms of eq. A.2 and A.3 is less than $n2^K$. Thus even a K=10 code at high channel noise level, which may require n=100 for accuracy, can be evaluated in seconds.

We note finally that the distance properties of a given code can be evaluated using eq. A.3 independent of the channel characteristics. For setting N=1 and $D=10^{-d}$ in eq. A.3, where d is an integer, we obtain

$$T(D) = 10^{-d\delta_\mu} (k_\mu + k_{\mu+1}\, 10^{-d} + k_{\mu+2}\, 10^{-2d} + \ldots) \qquad (A.5)$$

where k_μ is the number of paths of minimum weight δ_μ, $k_{\mu+1}$ is

the number of paths of weight $\delta_{\mu+1}$, etc. Thus provided $k_\mu+\gamma<10^d$ for $\gamma = 1, 2, \ldots,$ the first few nonzero digits will determine the integer $k_{\mu+1}$, etc. The number of terms which can be correctly determined in this way depends on the word size of the computer. It may also be possible to determine higher order terms by subtracting the effect of already determined lower order terms and renormalizing.

The total number of bits in error in the union of all erroneous paths at a given distance from the correct path can similarly be obtained by first differentiating as in eq. A.4 and then applying the procedure of eq. A.5. The coefficient with subscript $\mu+\gamma$ now denotes the total number of bit errors in all paths at distance $\delta_\mu+\gamma$ from the correct path.

Effect of Memory Truncation

This technique can be extended to determine the effect of memory truncation on the first-event error probability. Suppose path memory is truncated n branches prior to the last received branch and that a maximum likelihood decision is made to determine the output bit at that point. In this case, an error occurs whenever the likelihood function of any state (or node in the state diagram) exceeds that of the all zeros state, assuming this was the correct path. The probability of this event is union bounded by

$$T_t(D) = D^{\delta_i} \underset{\sim}{u} A^{n-1} \underset{\sim}{1}$$

where $\underset{\sim}{u}$ = 1111...1 (row vector)

and $\underset{\sim}{1}$ = 100...0 (column vector)

and A is a function of D only.

Note that the first branch is accounted for by D^{δ_i}. We note that truncation eliminates the ordinary errors due to remerging paths beyond the nth. Thus a union bound on the overall first-event error probability is

$$P_E < \sum_{n-2} T(D) + T_t(D)$$

$$= D^{\delta_i} \left\{ D^{\delta_f} \underset{\sim}{1}' \left[I + A + A^2 + \ldots A^{n-2} \right] \underset{\sim}{1} + \underset{\sim}{u} A^{n-1} \underset{\sim}{1} \right\}$$

For the Gaussian channel, this may be refined by multiplying this expression by $\operatorname{erfc}(\sqrt{\delta_\mu RE_b/N_0}) D^{-\delta_\mu}$ as in eq. 2.2.5.

APPENDIX B

A variety of communication systems can be mechanized to include an output variable which is bimodal, with a large mean when system operation is unreliable and a small mean when it is reliable. An important source of possible unreliability is in the system synchronization. The required variable to determine unsynchronized operation is the estimate of the number of errors made, which in a coded system may be the modulo-2 sum of the received sequence and the nearest possible codeword. For synchronized operation, the expected relative frequency of a one, p_s, is just the channel probability of error, while for unsynchronized operation, it will be considerably higher, $p_u >> p_s$. We shall assume that successive symbols of this observed sequence are independent.*

Detection of unreliable operation then proceeds as follows. After each bit (event) time, a one is subtracted from a counter if the bit was zero. If the bit was a one, an integer k-1 ($k>1$) is added to the counter. Only non-negative summands are stored; if the total sum ever becomes negative, it is reset to zero. Thus, we have a reflecting boundary at the origin. Whenever the count reaches a threshold N, we detect unreliable operation. Thus there is an absorbing boundary at N. By making $kp_s<1$ and $kp_u>1$, it follows that the expected drift is $kp_u-1 > 0$ (to the right)

---

*This is clearly true for synchronized operation if channel errors are independent; for unsynchronized operation, dependencies will actually improve operation.

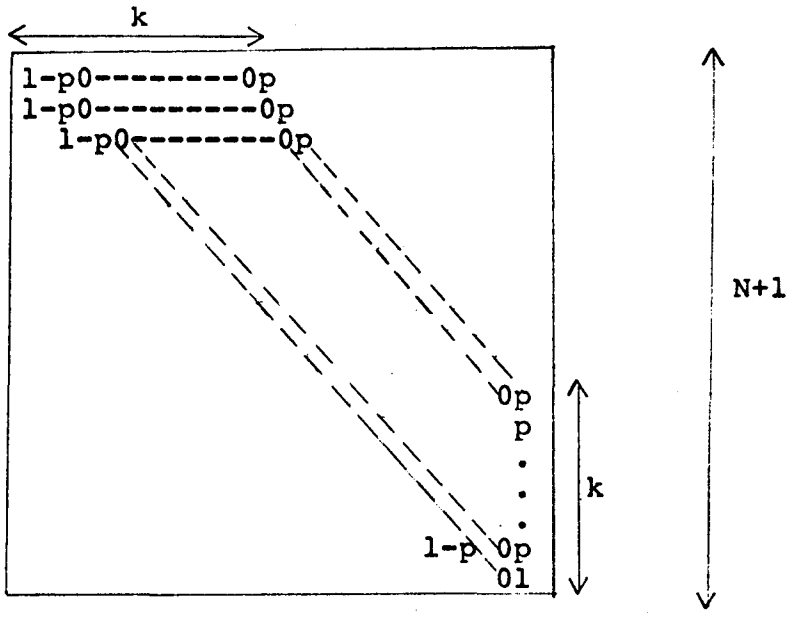
when the system is unreliable (unsynchronized), while it is $kp_s - 1 < 0$ when it is reliable (synchronized).

As is standard, we define a <u>false alarm</u> as the event that reliable operation causes a threshold crossing, and <u>detection</u> as the event that unreliable operation causes a threshold crossing. Of interest are the first passage time statistics in both cases, and in particular, the first and second moments; i.e., mean time to false alarm and detection and the corresponding variances.

Exact Analysis

For independent events, the Markov sequence random walk is completely characterized statistically by the transition matrix, where p equals p_s or p_u corresponding to either system mode (Ref. 1).

P =



We number the rows and columns from 0 to N, corresponding to the states (contents) of the counter. The P_{ij} term indicates the probability of a transition from state i to state j. We assume as a worst case* that the counter always starts in state 0.

---

*This will be the case in the synchronized mode if we take our time origin as the instant of initial synchronization. For the unsynchronized mode, the initial state will be the state of the counter when synchronization is lost, but the first passage time to detection will then be upper bounded by taking this to be 0.

Then the probability distribution of state occupancy $\underset{\sim}{\pi}^{(1)} = (\pi_0^{(1)}, \pi_1^{(1)} \ldots \pi_N^{(1)})$ after one transition (bit time) is

$$\underset{\sim}{\pi}^{(1)} = (1, 0, 0, \ldots 0)\ P \tag{B.1}$$

and after n transitions it is

$$\underset{\sim}{\pi}^{(n)} = \underset{\sim}{\pi}^{(n-1)} P = (1, 0, 0 \ldots 0)\ P^n \tag{B.2}$$

Now we are interested in the distribution of time to first arrival at state N. Clearly, the Nth component of $\underset{\sim}{\pi}^{(n)}$

$$\pi_N^{(n)} = \Pr(\nu_N \leq n) \tag{B.3}$$

where ν_N is the time of first arrival (passage) at the threshold N. Thus equations B.1 and B.2 yield the desired distribution. From this, we can obtain the mean and variance of first passage time by

$$E(\nu_N) = \sum_{n=1}^{\infty} n \left[\pi_N^{(n)} - \pi_N^{(n-1)}\right] = \sum_{n=0}^{\infty} \left[1-\pi_N^{(n)}\right] \tag{B.4}$$

$$\begin{aligned} Var(\nu_N) &= \sum_{n=1}^{\infty} n^2 \left[\pi_N^{(n)} - \pi_N^{(n-1)}\right] - E^2(\nu_N) \\ &= 2 \sum_{n=0}^{\infty} n\left[1-\pi_N^{(n)}\right] - \left\{\sum_{n=0}^{\infty} \left[1-\pi_N^{(n)}\right]\right\}^2 \end{aligned} \tag{B.5}$$

The numerical algorithm which generates (3) simply post-multiplies (1, 0...0) by P n times successively. Each time it selects the last term of the resulting vector, which is the distribution $\pi_N^{(n)}$. Also, it augments an accumulator to form the mean of equation B.4 and also forms the weighted function of equation

B.5 to obtain the variance.

While equations B.3, B.4, and B.5 yield all the results desired, the mean time and hence the mode of the distribution will tend to be very large for the false alarm and hence the total number of iterations required for a meaningful result may be several million, thus rendering this direct approach impractical.

Asymptotic Analysis

When the ratio of the threshold N to the maximum step size k-1 is very large, we may model this process as a continuous random walk. For this purpose, let us consider spatial parameters and continuous time, each step taking Δt seconds and being either $-\Delta x$ or $+(k-1)\Delta x$. Thus, the threshold becomes

$$a \triangleq N\Delta x \tag{B.6}$$

the mean drift per unit time

$$m \triangleq (kp-1)(\Delta x/\Delta t) \tag{B.7}$$

and the drift variance per unit time

$$\sigma^2 \triangleq k^2 p(1-p)(\Delta x)^2/\Delta t \tag{B.8}$$

The moment-generating function for the first passage time of this continuous random walk satisfies the equation (Ref. 2)

$$\frac{\sigma^2}{2}\frac{d^2\hat{f}(x,\lambda)}{dx^2} + m\frac{d\hat{f}(x,\lambda)}{dx} = \lambda f(x,\lambda) \tag{B.9}$$

where $\hat{f}(x,\lambda) = E\left[e^{-\lambda}f(x,t)\right] = \int_0^\infty e^{-\lambda t}f(x,t)dt$

and $f(x,t) = d/dt\left[Pr(\nu_t<t \mid \text{starting at } x)\right]$

From this, we can also obtain equations for the mean first passage time $E(\nu_t \mid \text{starting at } x) = \left.\frac{-\partial f(x,\lambda)}{\partial\lambda}\right|_{\lambda=0}$:

$$\frac{\sigma^2}{2}\frac{d^2t_1(x)}{dx^2} + m\frac{dt_1}{dx} = -1 \tag{B.10}$$

and more generally for the jth moment $t_j(x) = E(v_t^j \mid \text{starting at } x)$

$$\frac{\sigma^2}{2} \frac{d^2 t_j(x)}{dx^2} + m\, dt_j(x) = -j t_{j-1}(x) \qquad \text{(B.11)}$$

$$j = 2, 3, \ldots\ldots$$

Thus each moment can be obtained by iterating on the solution for the next lower equation. Of course, we could also obtain the negative jth moment by differentiating the moment generating function $f(x,\lambda)$ and setting $\lambda=0$.

The boundary conditions associated with equations B.10 and B.11 are obtained as follows. Since the threshold (absorbing boundary) is at x=a, and since the dependent variable x indicates the starting point,

$$t_j(a) = 0 \qquad j = 1, 2, \ldots\ldots \qquad \text{(B.12a)}$$

Also, whenever **x** becomes negative, it is automatically returned to zero, giving rise to the boundary condition,

$$\left.\frac{dt_j(x)}{dx}\right|_{x=a} = 0 \qquad j = 1, 2, \ldots. \qquad \text{(B.12b)}$$

For the moment generating function

$$\hat{f}(x,\lambda) = \sum_{j=0}^{\infty} \frac{t_j(x)}{j} (-\lambda)^j$$

since $t_j(0) = 1$, we obtain the boundary conditions

$$\hat{f}(a,\lambda) = 1$$

$$\left.\frac{d\hat{f}(x,\lambda)}{dx}\right|_{x=a} = 0$$

The solution of B.10 for the mean first passage time with the boundary conditions (B.12a, b) is

$$t_1(x) = \frac{e^{-2m\ a/\sigma^2} - e^{-2m\ x/\sigma^2}}{2m^2/\sigma^2} + \frac{a-x}{m}$$

Insertion of this solution into B.11 for j=2, yields the second moment $t_2(x)$. Of interest are,

$$E(\nu_t \mid \text{starting at } 0) = t_1(0) = \frac{1}{m}\left[\frac{e^{-2m\ a/\sigma^2} - 1}{2m/\sigma^2}\right] \qquad \text{(B.15)}$$

and

$$\text{Var}(\nu_t \mid \text{starting at } 0) = t_2(0) - t_1^2(0)$$

$$= \frac{\sigma^2}{m^3}\left[a(1+2e^{-2am/\sigma^2}) + (e^{-2am/\sigma^2} - 1)(e^{-2am/\sigma^2} + 5)\ \sigma^2/4m\right] \qquad \text{(B.16)}$$

It is also possible to solve for the entire moment-generating function by solving equation (9) with the boundary conditions (B.13a, b). The result evaluated at x=0 is

$$f(0,\lambda) = E\left[e^{-\lambda\nu} \mid \text{starting at } 0\right]$$

$$= \frac{\xi_2 - \xi_1}{\xi_2 e^{\xi_1 a} - \xi_1 e^{\xi_2 a}}$$

$$\xi_1, \xi_2 = (-m \pm \sqrt{m^2 + 2\sigma^2\lambda})/\sigma^2$$

$t_1(0)$ and $t_2(0)$ of equations B.15 and B.16 could also be obtained as the negatives of the first and second derivatives of B.17 at $\lambda=0$. The inversion of equation B.17 to obtain the density

function* is not a simple procedure and could be of questionable value.

*A Chernoff bound on the distribution is easily obtained and is of some value in the unsynchronized detection case.

Interpretation of Asymptotic Results

In order to properly interpret the asymptotic results of equations B.15 and B.16, we must note that the continuous approximation assumes that step sizes are much less than the threshold size; that is k/N<<1. Also for the unsynchronized case

$$kp_u - 1 > 0$$

while for the synchronized case

$$kp_s - 1 < 0$$

Referring to the definitions B.6, B.7, and B.8, and letting $\nu = \nu_t \Delta t$, we have in the unsynchronized case

$$E\left[\nu^u \mid \text{starting at } 0\right] = \frac{1}{kp_u-1}\left\{N + \frac{\exp\left[-2N(kp_u-1)/k^2p_u(1-p_u)\right] - 1}{2(kp_u-1)/k^2p_u(1-p_u)}\right\}$$

$$\approx \frac{N}{kp_u-1} \qquad \text{(B.18)}$$

which is essentially linear in N, the threshold value, and

$$\text{Var}\left[\nu^u \mid \text{starting at } 0\right] \approx \frac{k^2p_u(1-p_u)N}{(kp_u-1)^3}\left\{1+2\exp\left[-2N(kp_u-1)/k^2p_u(1-p_u)\right]\right\}$$

$$\approx \frac{Nk^2p_u(1-p_u)}{(kp_u-1)^3} \qquad \text{(B.19)}$$

and the normalized variance

$$\frac{\text{Var}(\nu^u)}{E^2(\nu^u)} \approx \frac{k^2p_u(1-p_u)}{N(kp_u-1)} = \frac{k(1-p_u)}{N\left[1-1/(kp_u)\right]} \qquad \text{(B.20)}$$

In the synchronized case, on the other hand,

$$E\left[\nu^s \mid \text{starting at } 0\right] = \frac{1}{1-kp_s}\left\{\frac{\exp\left[2N(1-kp_s)/k^2p_s(1-p_s)\right] - 1}{2(1-kp_s)/k^2p_s(1-p_s)} + N\right\} \tag{B.21}$$

which grows nearly exponentially with N, while

$$\mathrm{Var}\left[\nu^s \mid \text{starting at } 0\right] \approx \frac{\left[k^2p_s(1-p_s)\right]^2}{4(1-kp_s)^4} \exp\left[4N(1-kp_s)/k^2p_s(1-p_s)\right] \tag{B.22}$$

so that

$$\frac{\mathrm{Var}(\nu^s)}{E^2(\nu_s)} \approx 1 \tag{B.23}$$

From B.20, we see that for detection of unsynchronized operation, the normalized variance is very small for k/N, so that the mode is quite peaked. On the other hand, from (B.23), it follows, that for a false alarm when the system is synchronized, the mode is quite broad, reminiscent of either a Poisson or a Rayleigh distribution.

The basic assumption which justifies the continuous model is that $k/N \ll 1$. If this is not the case, the asymptotic formulas lose their validity and the numerical exact solution is required.

Application to Loss-of-Phase-Lock Indicators

As an application of the above techniques to other than decoder synchronization, consider an unmodulated carrier or subcarrier tracking loop. A convenient measure of loop performance is the sign of cos $\phi(t)$ where $\phi(t)$ is the instantaneous phase error, and cos $\phi(t)$ can be generated by multiplying the received signal by the quadrature VCO output and low pass filtering. Thus cos $\phi(t)>0$ implies $|\phi(t)| < \pi/2$, while cos $\phi(t)<0$ implies $\pi/2 < |\phi(t)| < \pi$.

Now suppose we sample this signal periodically, but with a sufficiently long period that successive samples are nearly independent. The counter contents are incremented by k-1 whenever the sample is negative and reduced by 1 whenever the sample is positive. When the phase-locked loop is properly tracking, the phase error probability density function (Ref. 3) is given exactly for a first-order loop, and approximately for higher order loops, by the expression

$$P_s(\phi) = \frac{e^{\alpha\cos\phi}}{2\pi I_o(\alpha)}, \quad -\pi \leq \phi \leq \pi$$

where $\alpha = S/NoB_L$

When the loop is out of lock, on the other hand

$$P_u(\phi) = \frac{1}{2\pi}, \quad -\pi \leq \phi \leq \pi$$

Thus, clearly, using the notation of the previous sections,

$$P_u = 2\int_{\pi/2}^{\pi} P_u(\phi)\, d\phi = \frac{1}{2}$$

while

$$p_s = 2\int_{\pi/2}^{\pi} P_s(\phi)\, d\phi$$

and is given for various values of α in Table 1, which is extracte from Figure 4.7 of Reference 3. Thus even at 0 db, $p_u > 2p_s$ so that the approach seems practically feasible.

α	p_s
6 db	<.001
3 db	.09
2 db	.13
1 db	.18
0 db	.22

Table B.1 p_s as a function of signal-to-noise ratio

REFERENCES

1. W. Feller, "An Introduction to Probability Theory and Its Application," Wiley, 1950.

2. D. A. Darling and A. J. F. Siegert, "The First Passage Problem and a Continuous Markov Process," Annals of Mathematical Statistics, Vol. 24, pp. 624-639, 1953.

3. A. J. Viterbi, "Principles of Coherent Communication", McGraw-Hill, 1966.

APPENDIX C

Overflow Probability in a Hybrid System

The overall overflow probability of the system for an initially empty buffer is

$$P_0 = p_\phi \cdot \Pr(\text{overflow in a deleted frame})$$

$$= p_\phi \left\{ \sum_{K=B}^{D} \Pr\left(\begin{matrix}\text{more than B deleted frames} \\ \text{in K trials}\end{matrix} \;\middle|\; \begin{matrix}\text{sequential decoding} \\ \text{lasts K frame times}\end{matrix} \right) P_s(K) + \Pr(K>D) \right\} \qquad (C.1)$$

where $P_s(K) = \Pr\left(\begin{matrix}\text{sequential decoding of deleted frame} \\ \text{lasts exactly K frame times}\end{matrix} \right)$

We have established in Section 5.3 that the computational distribution for deleted frames is upper bounded by the Pareto distribution divided by the deletion rate p_ϕ.

Thus for 1000 bit frame, we shall use

$$\Pr(K \geq J) \leq \frac{10^3}{p_\phi (10^3 J\mu)^\rho}$$

and consequently*

$$P_s(K) \leq \frac{10^{3(1-\rho)}}{p_\phi \mu^\rho K^\rho} \qquad (C.2)$$

Since frames are deleted independently for a memoryless channel

*This is the only strictly valid bound which we can obtain from a bound on the cumulative distribution. Also this is the only upper bound which we shall use whose tightness is questionable. All further bounds are very tight, and all approximations can be changed into tight upper bounds by including appropriate additional terms which approach zero for very large buffer sizes.

$$\Pr\left(\begin{matrix}\text{more than } B \\ \text{deleted frames}\end{matrix} \;\middle|\; \begin{matrix}\text{sequential decoding} \\ \text{lasts } K \text{ frame times}\end{matrix}\right)$$

$$= \sum_{j=B+1}^{K} \binom{K}{j} p_\phi^B (1-p_\phi)^{K-B} \leq \frac{e^{KH(B/K)} p_\phi^B (1-p_\phi)^{K-B}}{\sqrt{2\pi B}} \qquad (C.3)$$

The inequality is a Chernoff bound for the binomial distribution provided $B/K > 2p_\phi$ and

$$H(Y) = -Y\ell nY-(1-Y)\ \ell n(1-Y)$$

Thus inserting C.2 and C.3 in C.1, we obtain for the overall overflow (error) probability

$$P_\theta \leq \sum_{K=B+1}^{D} \left\{ \frac{e^{KH(B/K)}}{\sqrt{2\pi B}} (1-p_\phi)^{K-B} \frac{p_\phi^B \ 10^3(1-\rho)}{(K\mu)^\rho} \right\} + \frac{10^3(1-\rho)}{(D\mu)^\rho}$$

$$= \frac{10^3(1-\rho)}{\mu^\rho} \left\{ \frac{1}{D^\rho} + \sum_{K=B}^{D} \frac{\exp B\left[xH(1/x)+(x-1)\ell n(1-p_\phi)\right]}{\sqrt{2\pi}} \frac{p_\phi^B}{x^\rho B^{\rho+\frac{1}{2}}} \right\} \qquad (C.4)$$

where $x = K/B$ and $B/D >> 2p_\phi$

Since K ranges over the integers and B>10, x increases in increments of $1/B<0.1$. Thus, we may accurately approximate the sum by an integral and obtain

$$P_0 \leq \frac{10^3(1-\rho)}{\mu^\rho} \left[\frac{1}{D^\rho} + \frac{p_\phi^B}{B^{\rho+\frac{1}{2}}} \int_1^{D/B} (2\pi)^{-\frac{1}{2}} x^{B-\rho} \left(1-\frac{1}{x}\right)^{-(x-1)B} e^{-\gamma(x-1)}\, dx \right] \qquad (C.5)$$

where $\gamma = -B\ell n(1-p_\phi) > 0$ (C.6)

Using the tight inequality,

$$\left(1-\frac{1}{x}\right)^{-(x-1)} \leq e$$

we obtain,

$$P_0 \leq \frac{10^3(1-\rho)}{\mu^\rho}\left[\frac{1}{D^\rho} + \frac{p_\phi^B\, e^{B+\gamma}}{\sqrt{2\pi}\, B^{\rho+\frac{1}{2}}} \int_1^{D/B} x^{B-\rho}\, e^{-\gamma x}\, dx\right] \tag{C.7}$$

The integral is just the incomplete gamma function which is closely bounded by taking the complete integral

$$P_0 \leq \frac{10^3(1-\rho)}{\mu^\rho}\left[\frac{1}{D^\rho} + \frac{p_\phi^B\, e^{B+\gamma}\, \Gamma(B-\rho+1)}{\sqrt{2\pi}\, B^{\rho+\frac{1}{2}}\, \gamma^{B-\rho+1}}\right]$$

Now substituting (C.6) for γ and noting that

$$\frac{e^\gamma}{B^{B-\rho+1}} = \frac{(1-p_\phi)^{-B}}{B^{B-\rho+1}\left[-\ell n(1-p_\phi)\right]^{B-\rho+1}} \simeq (Bp_\phi)^{\rho-B-1}(1-p_\phi/2)^{-B}$$

we obtain

$$P_0 \leq \frac{10^3(1-\rho)}{\mu^\rho}\left[\frac{1}{D^\rho} + \frac{p_\phi^{\rho-1}\, e^B\, \Gamma(B-\rho+1)}{\sqrt{2\pi}\, B^{B+3/2}(1-p_\phi/2)^B}\right] \tag{C.8}$$

Finally using Stirling's formula

$$\Gamma(n+1) \simeq \sqrt{2n\pi}\left(\frac{n}{e}\right)^n$$

we obtain, using $(1-\rho/B)^{B-\rho} \approx e^{-\rho}$ since $B>>\rho$,

$$P_0 \lesssim \frac{10^3(1-\rho)}{\mu^\rho} \left[\frac{1}{D^\rho} + \frac{p_\phi^{\rho-1}}{B^{\rho+1}(1-p_\phi/2)^B}\right]$$

or since $(1-p_\phi/2)^{-B} \simeq e^{Bp_\phi/2}$

$$P_0 \lesssim 10^3(1-\rho)\left[\frac{1}{(\mu D)^\rho} + \left(\frac{e^{Bp_\phi/2}}{Bp_\phi}\right)\frac{1}{(\mu B/p_\phi)^\rho}\right], \quad \text{(C.9)}$$

provided $B/D > 2p_\phi$, the first term being due to delay line overflows and the second to buffer overflows. Thus the buffer size is effectively increased by the factor $1/p_\phi$.