②

TECHNIQUES FOR IMPROVING THE RELIABILITY OF DATA TRANSMISSION OVER THE HF RADIO CHANNEL

FINAL REPORT

by

①

Alberto Leon-Garcia
Department of Electrical Engineering
University of Toronto
Toronto, Ontario

31 March, 1984

prepared for

Department of Communications, Canada
under Contract OST83-00101

P
91
C655
L455
1984

DD 4589466
DL 4589533

1.0 INTRODUCTION

This report presents the results of several investigations concerning the application of Viterbi decoding to transmission over the Syncompex binary FSK modem. The report is organized as follows. Section 2 discusses the implementation of a real-time Viterbi decoder for use with the Syncompex modem. The organization of the software that implements the Viterbi decoder is discussed in detail and documentation is included in Appendices. Speed and memory requirements are also discussed. Section 3 considers several performance issues of Viterbi decoding. Experimental and simulation results are presented for the performance of the decoder for various settings of decoder parameters. Simulation results are also presented for the performance of Viterbi decoding when combined with internal interleaving. Finally Section 4 considers the extension of the present transmission protocol to a system that utilizes a multi-FSK signal format. The present transmission protocol is analyzed to establish a basis for the evaluation of the other protocols. The transmission protocol discussed by Lin is analyzed in detail in the context of Viterbi decoding. Methods are developed for evaluating the throughput performance. This protocol is then compared to selective repeat ARQ and to selective repeat ARQ with code diversity. A new adaptive protocol is proposed that achieves the performance of the Lin protocol while requiring a simpler implementation.

2.0 REAL-TIME VITERBI DECODER

The first objective of the project was the implementation of a real-time Viterbi decoder for use with code-diversity transmission on the Syncompex modem. This allowed us to verify the attainable decoding speeds and enabled us to undertake more comprehensive experimental studies on the performance of Viterbi decoding as various system parameters are varied. This chapter describes the hardware and software developed to accomplish these goals.

The chapter is organized as follows. Section 2.1 gives an overview of the system. Section 2.2 describes the approach used in the software implementation of the Viterbi decoder. Section 2.3 describes the organization of the system software including brief descriptions of the main modules. Timing and memory requirements are also discussed. The chapter concentrates on the main ideas only but details are included in the appendices.

2.1 SYSTEM OVERVIEW

Figure 1 summarizes the signal processing required to obtain a sequence of discrete-channel outputs suitable for Viterbi decoding. This processing is done by what we will refer to as the ADC board. The modem provides three analog output signals. The outputs of Chips #1 and #2 are the noncoherently-demodulated ("eye") baseband signals corresponding to the binary sequences produced by the convolutional encoding at the transmitter. These signals have a baud rate of 75 symbols per second. Chip #3 provides a clock signal recovered from the eye signals using selection diversity. This clock signal is offset with respect to the eye-signal baud period so a delay is introduced to obtain a clock signal suitable for controlling the integrate-and-dump operations. The integrator outputs are sampled at the appropriate instants and the digitized samples are then passed to the Viterbi decoder.

The Viterbi decoder is implemented in software using a Motorola 6809 microprocessor. The decoder software is written so that various system parameters, inclu the code, the channel-output quantization, and the decision depth, can be varied. The decoder outputs the estimates of the information sequence in a form suitable for analysis by a data error analyzer. The 6809 microprocessor is also programmed to process these outputs in order to compile error statistics. These statistics are displayed on a computer terminal as the real-time decoding takes place.

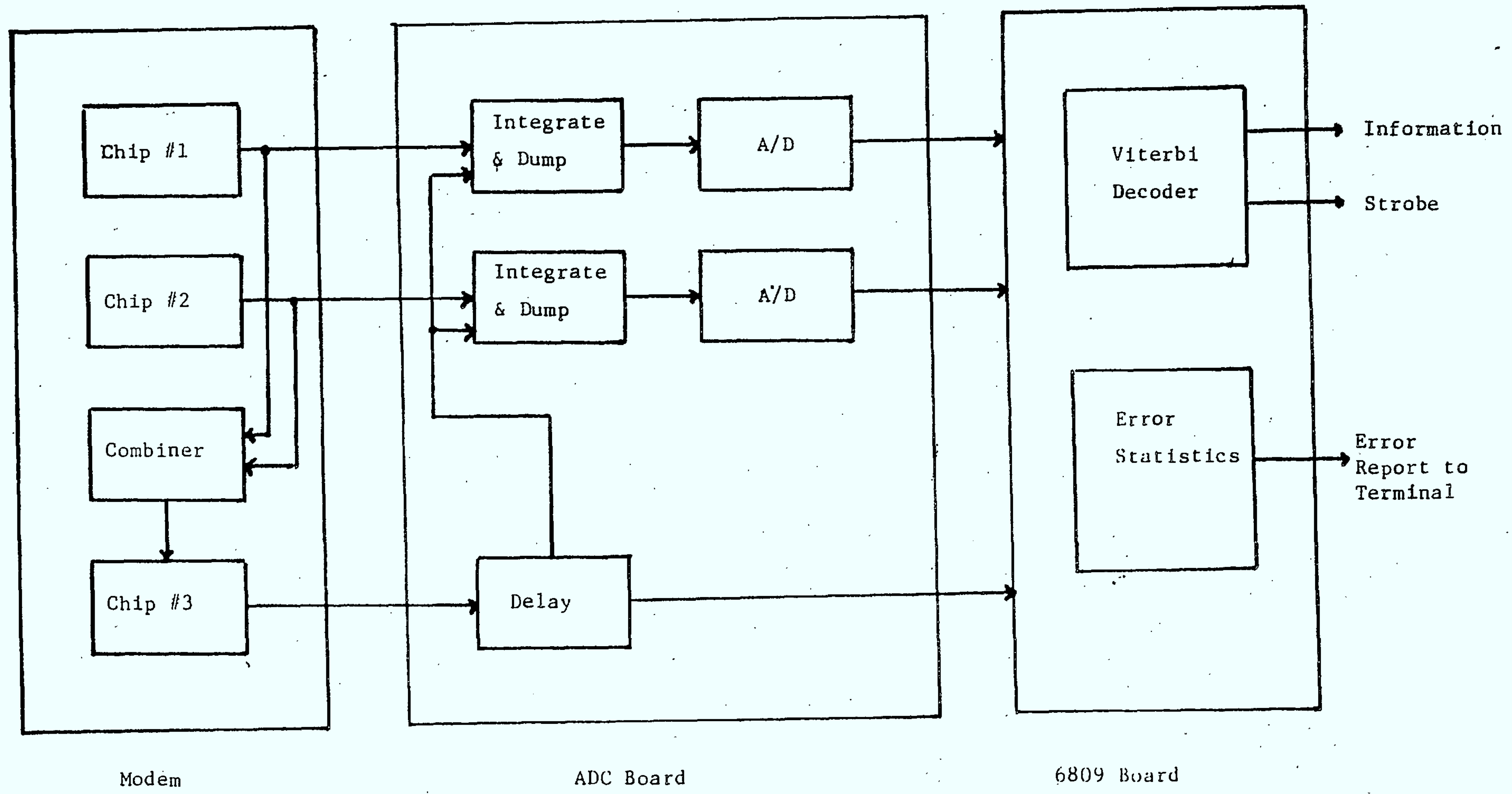

Chip #1
Chip #2
Combiner
Chip #3
Integrate & Dump
Integrate & Dump
A/D
A/D
Delay
Viterbi Decoder
Error Statistics
Information
Strobe
Error Report to Terminal
Modem
ADC Board
6809 Board

FIGURE 1

2.2 DECODER IMPLEMENTATION APPROACH

In this section we will first discuss the Viterbi decoding algorithm in general and then we will outline how it was implemented in software.

The task of the Viterbi decoder is to estimate the information sequence that was fed into the encoder at the transmitter. This is equivalent to finding the sequence of encoder states that produces the encoded sequence that is closest to the observed channel output sequence according to some metric. The decoder stores two entries for each possible encoder state. The first entry contains the minimum-distance sequence of states leading to the given state up to the given time instant. We will refer to this entry as the "path history" of the given state. The second entry contains the distance of the corresponding encoded sequences to the observed channel output sequence. We will refer to this entry as the "state metric."

Each time a channel output pair is passed to the decoder, the entries for all possible states, henceforth referred to as the State Information Tables, are updated using the following sequence of steps (see Figure 2):

1. Branch Metric Computation

The distances ("branch metrics") of the new channel-output pair with respect to the four possible encoder outputs, namely 00,01,10,11, need to be found. This can be done by computation or by table-lookup depending on the complexity of the metric used. The four branch metrics obtained in this step will be used in the subsequent steps.

2. Add, Compare, Select (ACS)

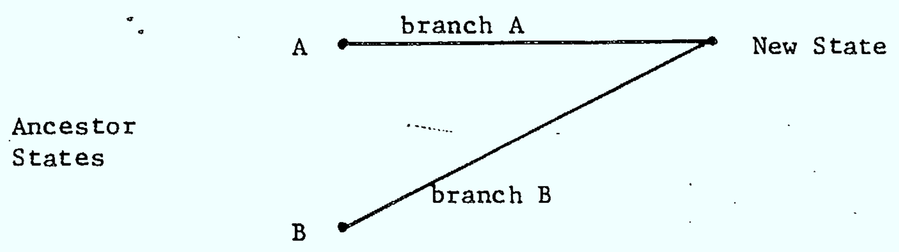
Each encoder state has only two possible ancestor states that can immediately precede it. The best new sequence leading to a given state is thus found by comparing these two ancestor states. The state metric of each ancestor state is added to the branch metric corresponding to the corresponding state transition; the two resulting metrics are compared; and the ancestor state with the smaller metric is selected. The best new sequ for the given state is found by concatenating the selected ancestor state to the path history of the selected ancestor state. The new state metric of the given state is given by the smaller of the two metrics used in the compare step. The ACS procedure is carried out for each of the possible encoder states.

Upon completion of the above steps, the State Information Tables will be completely updated and the decoder will be ready for another channel output pair.

The above two steps form the heart of the Viterbi decoding algorithm. Two additional steps are required because of

Branch	Branch Metric
00	metric
01	metric
10	metric
11	metric

Branch Metric Table



metric(A) + branch metric(A) $\gtrless$ metric(B) + branch metric(B)

FIGURE 2

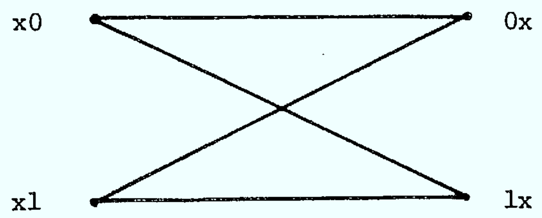


FIGURE 3

practical considerations:

3. Truncation of Path History

The length of the path histories need to be kept bounded in order to keep the decoding delay bounded and to keep the memory requirements reasonable. Periodically it becomes necessary to make a firm decision on the estimate for the oldest segment of past history currently stored in order to free up memory. The optimum procedure involves finding the state with the smallest state metric at that time instant and the required segment of past history is then set to the corresponding segment in the "winning" state's path history entry. The memory space allotted to storing this segment of path history then becomes available for storing new segments of path history. The amount of time "wasted" in searching for the best metric can be kept proportionally small by making each segment consist of several information bits. The above view also suggests the partitioning and handling of the path history as a ring buffer with each unit of path history equal to one segment. By selecting these segments to be close to one byte in length, the need to perform bit manipulation is completely circumvented. This is a very important consideration in software implementations.

4. Scaling of State Metrics

As decoding proceeds the range of values occupied by the state metrics will steadily increase. In order to keep these values within the range that can be assumed within the finite precision of the machine, it becomes necessary to periodically reset the metric values so that they fall within the desired range. The rate at which the state metric values increase is proportional to the number of bits used in the branch metric calculations. If the number of bits is not too large, the state metrics will need rescaling very infrequently.

From the above discussion it is clear that the principal factor in determining the attainable decoding speed is the at of time required to carry out the ACS operation. We now examine this operation more closely.

Let the state of the encoder be given by the binary representation of the contents of its shift register with the newest bit equal to the most significant bit. If the constraint length is v, the state is given by a v-1 bit binary number. To make the discussion concrete, consider the case v=7. The number of states is then 2**6=64. Consider a destination state in the range 0 to 31, that is, a state with binary representation 0x, where x is 5 bit number. (See figure 3.) The source states for this destination state are x0 and x1. Furthermore note that destination state 1x has the same set of source states. Since the ACS operation for both of these destinations involve the same state metrics and state histories it is efficient to process the pair of destination states together.

Note that the source states in figure 3 have consecutive indices. Thus if the state metrics and path histories are arranged according to the numerical value of the state, the information required by the ACS operation of various states is obtained by working down the State Information Table.

Before discussing how the State Information Table was organized we need one further property. Consider a destination state with representation 0y0, where y is a 4 bit number. Note that the state metric and path information of 0y0 and 0y1 will be required together when the next channel output is processed. It would thus be convenient to handle the ACS operations of destination states 0y0 and 0y1 at about the same time so that the results of the operations can be stored together. This suggests handling the ACS operations in groups of four as shown in figure 4. This is how the ACS operation was implemented in our project and it accounts for how the State Information Table was formed.

The State Information Table contains two entries for each state. The first entry is a two-byte word specifying the state metric. The second entry is a one-byte word of recent path history corresponding to one segment of path history. Long term history is stored in a separate ring buffer which is updated when a segment of recent history has accumulated. The state information for pairs of source states is bundled together as shown in figure 4. Two State Information Tables were used. These tables alternate between being sources and destinations of the information generated during the ACS operations. By using in-place storage techniques it is possible to function with only one State Information Table. We found however that the implementation of this approach implied memory requirements elsewhere larger than that of the Table itself.

In omplementation of the Viterbi decoder, the ACS operation is driven by a State Organization Table that provides the addresses of locations where information is to be found or stored. Before explaining the State Organization Table we need to consider how the branch metric calculations are carried out. Let ab and cd be the two branch labels associated with the ACS operation of some given destination state as shown in figure 5. Each branch label can take on 4 possible values so there are a total of 16 possible pairs of branch labels. Let rs be the pair of channel output symbols that are to be processed. The ACS operation for the given destination state will then require the branch metrics d(ab,rs) and d(cd,rs), where d(.,.) denotes the metric function. The branch metric calculations are handled as follows. As soon as the channel outputs rs are read in, all 16 possible pairs of branch metric values are computed and stored in some fixed locations where they can be readily accessed by the ACS operations.

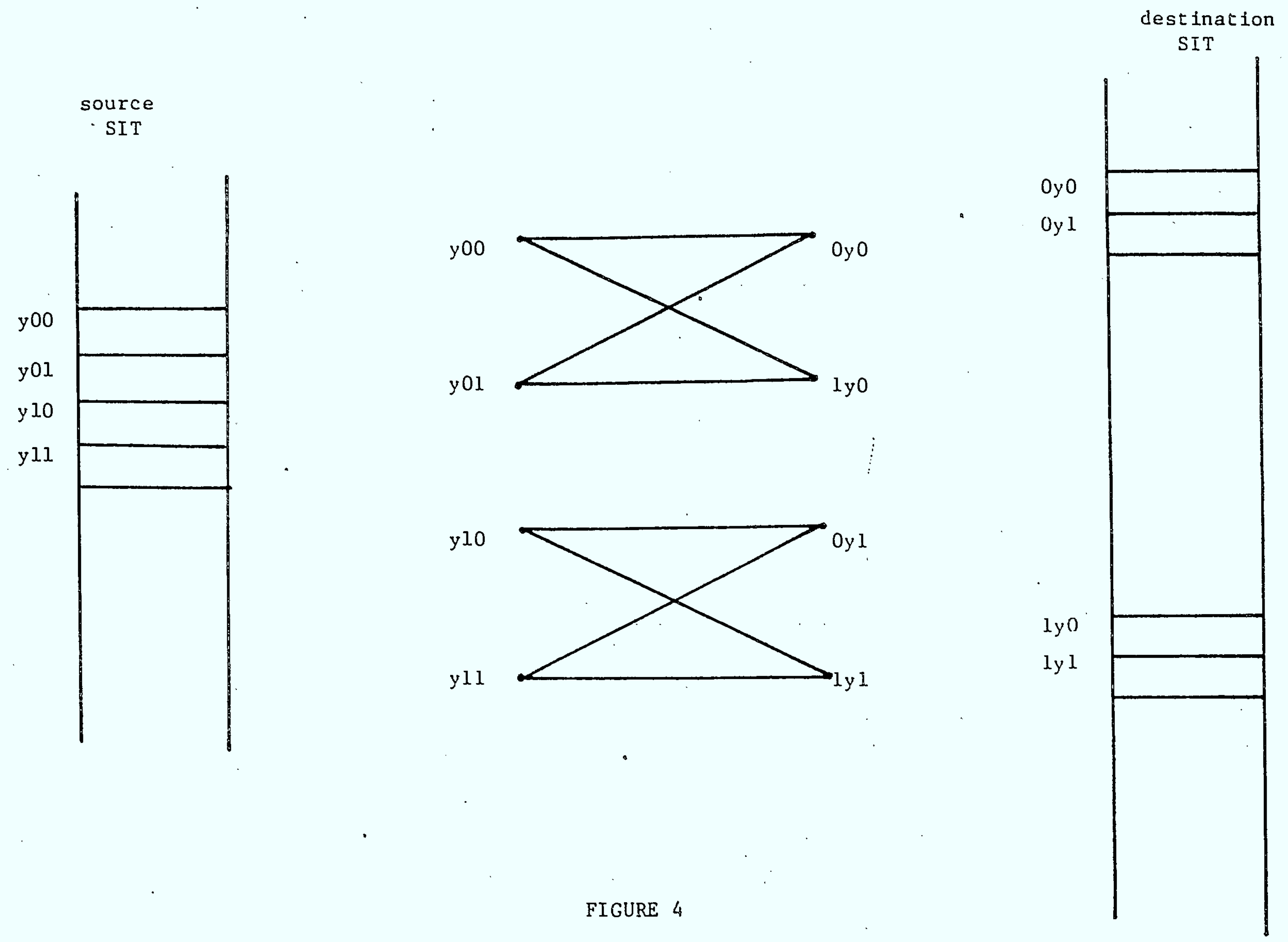
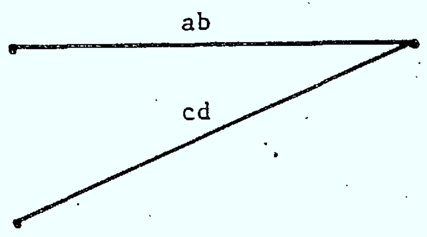
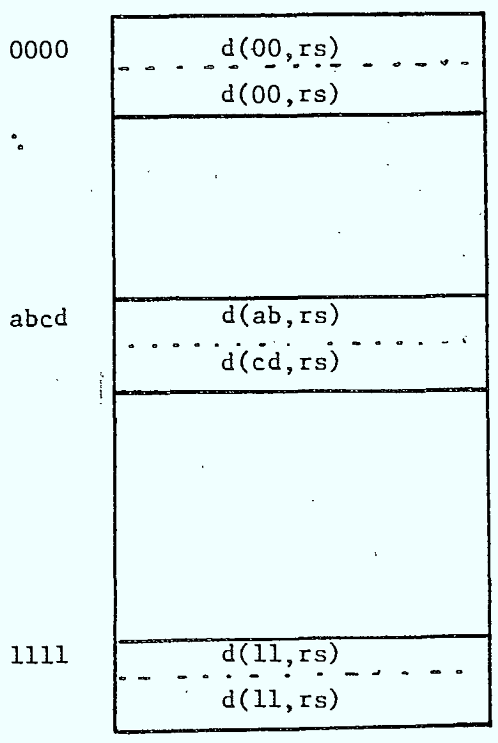

source
SIT
y00
y01
y10
y11
y00
y01
y10
y11
0y0
1y0
0y1
1y1
destination
SIT
0y0
0y1
1y0
1y1

FIGURE 4





Branch Metric Table

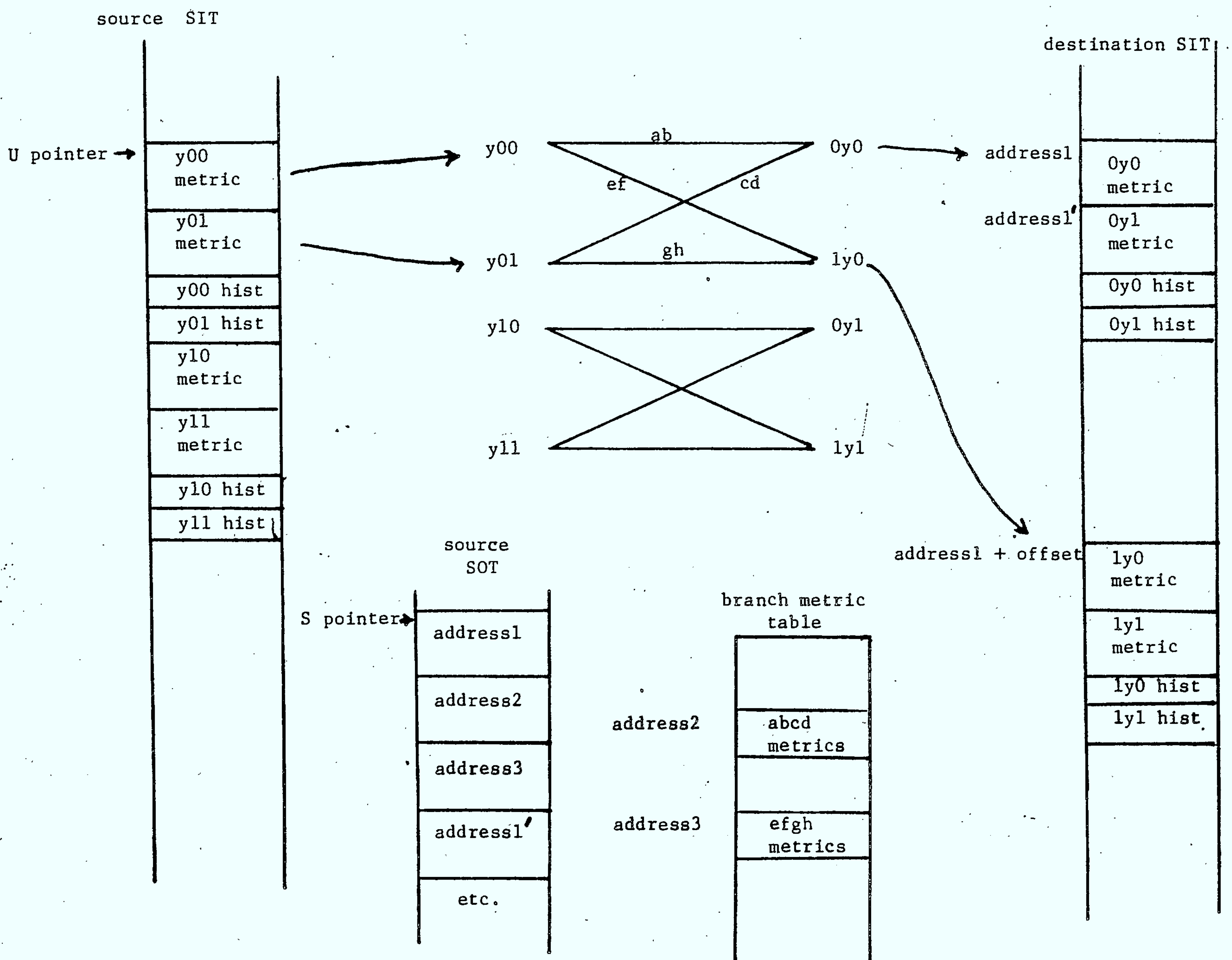
FIGURE 5

As soon as the branch metric calculations have been completed and stored, the ACS operation for the states is initiated. The ACS operation for each state will require two sets of entries from a source State Information Table (SIT) and will generate two sets of entries to be stored in the destination SIT. The function of the State Organization Table (SOT) is to provide the addresses of memory locations where the appropriate branch metric labels are to be found and where the destination state information are to be stored. The State Organization Table consists of pairs of groups of 3 two-byte words. The first group provides the addresses required by the ACS operation of states of the form 0y0 and 1y0. The second group provides the address information for states of the form 0y1 and 1y1. Two State Organization Tables are required since the addresses involved will depend on which SIT is acting as the source and which as the destination. The SOT structure and the flow of information is shown in figure 6.

As indicated above the decoding of channel output symbols proceeds until one segment of recent history has accumulated. It is then necessary to output the oldest segment of history in order to free up enough memory to store the next segment of path history. We now explain how this is done. It can be seen from figure 3 that the destination state and the source state have all but one bit in common in their binary representations. In a conventional approach to history updating the identity of the previous "winning" ancestor state is stuffed into the path history of a given destination state. This is shown in figure 7a for a constraint length 4 code. When the time comes to output segments of history, a search is carried out for the state with the smallest metric, say state abc in the example, and then it is necessary to "trace back" to identify the segment that is to be output. The trace back is done by using the contents of the path history registers: the contents of abc point back to location bcd, which in turn points back to location cde, and so o

Now consider the following different procedure. At time t+1 we proceed as before and stuff the identity of the "winning" ancestor state into a recent path history register. However for the subsequent time instants up to t+v-1, we stuff the contents of the recent path history register of the the "winning" ancestor state. An example is shown in figure 7b. Note that figures 7a and 7b give the same information, namely the best sequence leading to abc has the three most recent bits def. In tracing back, however, the second approach can jump immediately from abc to location def v-1 time units earlier. The trace back operation can thus be carried out much more quickly.

Once the trace back operation has been completed and a segment of v-1 bits has been selected for output, the contents of the recent path history entries in the State Information Table are transferred to the newly freed memory locations in the ring buffer containing the long term path history as shown in figure 8. The decoder is then ready to undertake another cycle of processing another segment of v-1 channel output pairs.



source SIT
U pointer
y00 metric
y01 metric
y00 hist
y01 hist
y10 metric
y11 metric
y10 hist
y11 hist
y00
y01
y10
y11
ab
ef
cd
gh
0y0
1y0
0y1
1y1
destination SIT
address1
address1'
0y0 metric
0y1 metric
0y0 hist
0y1 hist
address1 + offset
1y0 metric
1y1 metric
1y0 hist
1y1 hist
source SOT
S pointer
address1
address2
address3
address1'
etc.
branch metric table
address2
abcd metrics
address3
efgh metrics

FIGURE 6

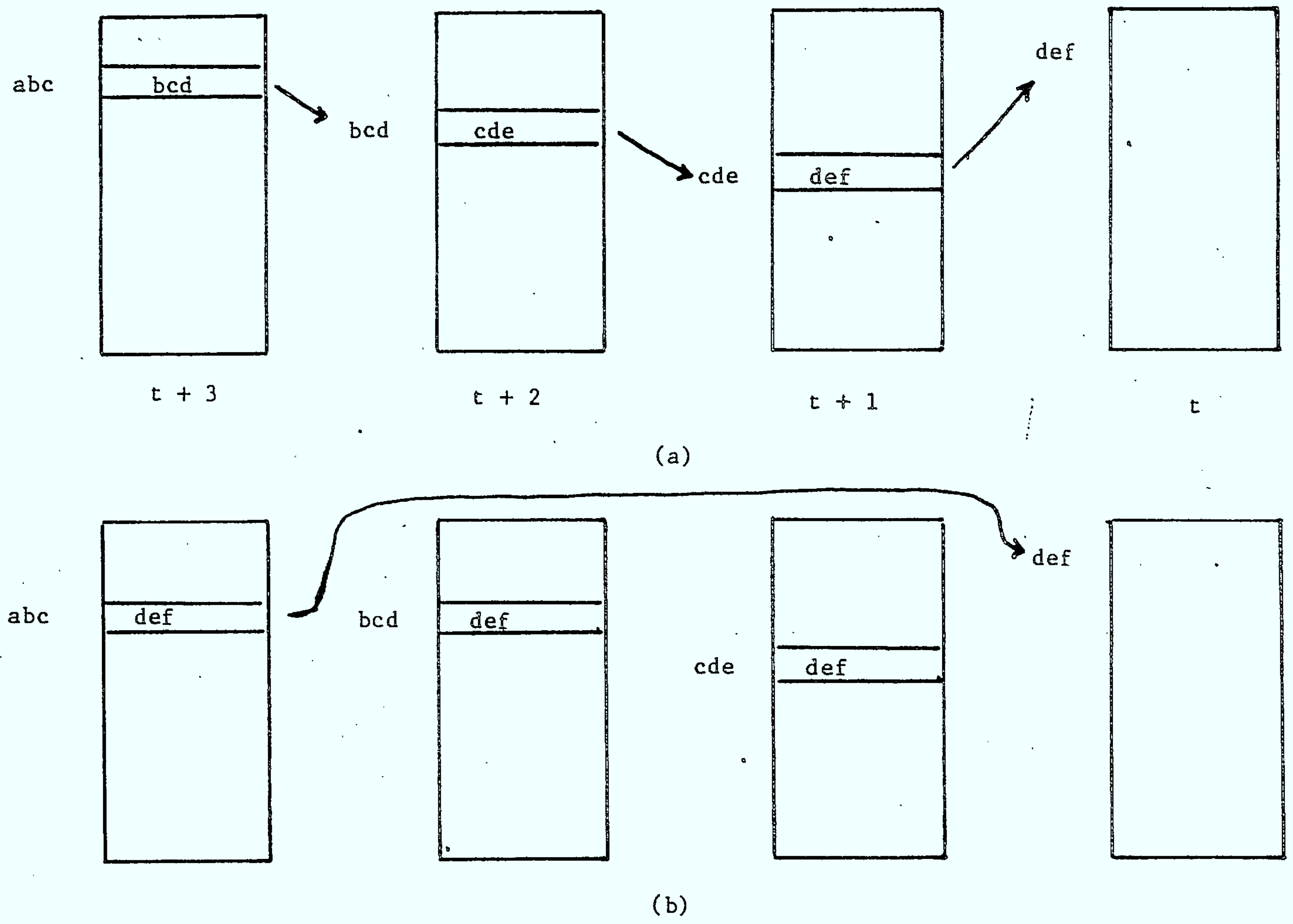

abc
bcd
bcd
cde
cde
def
def
t + 3
t + 2
t + 1
t
(a)
abc
def
bcd
def
cde
def
def
(b)

FIGURE 7

2.3 SOFTWARE ORGANIZATION

The system software consists of two parts. One part implements the real-time Viterbi decoder. The second part carries out the error checking functions as well as the compilation of error statistics.

Figure 8 shows the hierarchy of software modules responsible for the real-time Viterbi decoding. The main program begins by calling the subroutine INIT to initialize program variables and storage areas, and it then enters a loop that defines one decoding cycle. One decoding cycle consists of the following subroutine calls:

```
Call ACS1
Call SCALE
Call ACS2
Call ACS1
Call ACS2
Call ACS1
Call ACS2
Call SEARCH
Call HIST
```

The above decoding cycle assumes a path history segment of 6 bits corresponding to a constraint length 7 code. The modules ACS1 and ACS2 implement the ACS operations. They differ only in which State Information Table and State Organization Table act as source and destination for the ACS operations. The module SCALE makes sure that the values of the state metrics remain within the range that can be handled. The module SEARCH identifies the state with the smallest path metric and the module HIST implements the traceback operation and the transfer of recent path history to long term path history. The module HIST returns the segment of 6 information bits that are selected for output after the traceback operation. In order to produce a nearly synchronous output of information bits, these bits are output one at a time during the subsequent 6 ACS Subroutine calls.

The ACS subroutine begins by calling the module SHIFT. This module begins by calling PUTBIT and STROBUP in order to output an information bit and set the strobe up in order to indicate valid data to the data error analyzer. Note that PUTBIT calls ERRCHK which invokes the error checking and error compilation software modules. SHIFT then calls GETBIT which reads the channel outputs from the ADC board and then maps these vs through the table #MAP. The choice of entries in this table allows us to implements nonlinear quantization in addition to the usual linear quantization. SHIFT then calculates the 16 possible branch metric entries and stores them in the direct page area starting with the address label T0000. Finally SHIFT calls STROBDN to lower the strobe while the data is still valid. Upon return from SHIFT the ACS module proceeds to carry out the ACS operation for all the states in groups of four as indicated in the previous section.

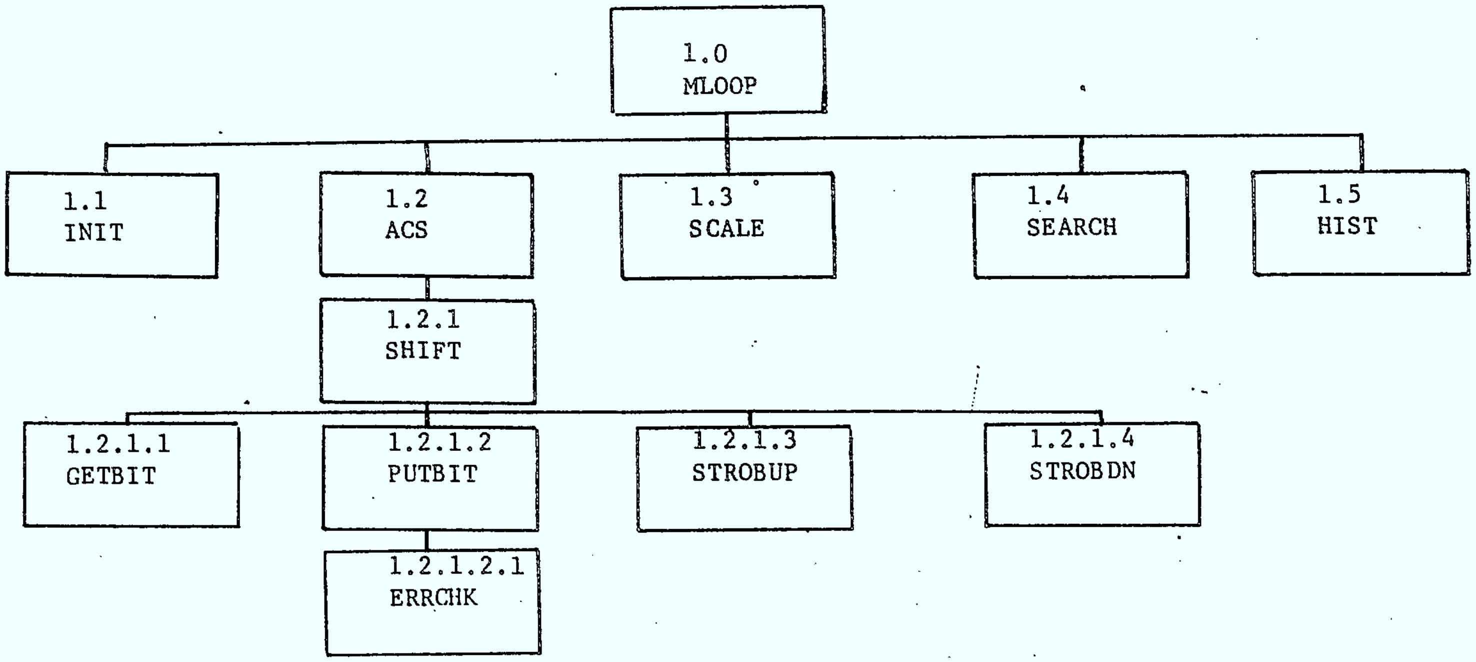

1.0
MLOOP
1.1
INIT
1.2
ACS
1.3
SCALE
1.4
SEARCH
1.5
HIST
1.2.1
SHIFT
1.2.1.1
GETBIT
1.2.1.2
PUTBIT
1.2.1.3
STROBUP
1.2.1.4
STROBDN
1.2.1.2.1
ERRCHK

FIGURE 8

The number of instruction cycles required to carry out the above modules is given by:

ACS	371 + 289 x 2**(v-3)
SEARCH	31 + 55 x 2**(v-2)
HIST	138 + 32 x 2**(v-2)
SCALE	23 + 46 x 2**(v-2)

and the number required by one decoding cycle is

(v-1) x ACS + SEARCH + HIST + SCALE.

The number of instruction cycles required by a decoding cycle for a constraint length 7 code is 34,418 which is equal to 35.9 ms. for the .96 MHz clock used in the current implementation. At the information rate of 75 bps, the 6 bits of a decoding cycle are produced in 80 ms. Thus for a constraint length 7 code the microprocessor is idle more than 50% of the time. (This was observed experimentally by observing the strobe signal on an oscilloscope.) If the constraint length is increased to 8, the number of cycles increases to 76,037 and the time to 79.2 ms. The 7 information bits are produced in 93.3 ms, so the microprocessor is now busy decoding about 85% of the time.

An upper bound on the information rate that can be handled with this software can be obtained by assuming that modifications are made so that SEARCH, HIST, and SCALE become negligible. For a constraint length 7 code, the maximum information rate that can be handled is then 192 bps. For a constraint length 8 code the maximum is 100 bps.

A real-time dedicated microprocessor implementation of the decoding algorithm would have the following memory requirements. The principal components of ROM memory are the program, the two State Organization Tables, and the quantization map. The respective memory requirements are 600 bytes, 2 x 3 x 2**(v-1) bytes, and 2**(v-1) bytes. For a constraint length 8 code, this adds up to approximately 1.5 kbytes. The principal components of RAM memory are the two State Information Tables, and the path history ring buffer. The respective memory requirements are 2 x 3 x 2** (v-1) and 8 x 2**(v-1), where a history depth of 8 segments has been assumed. This adds up to 1.8 kby

Figure 9 shows the module hierarchy for the error checking and error compilation software. When synchronized to the information sequence, ERRCHK takes the information bit that has just been output and compares it to that predicted by the module NEXTPN which is designed to emulate the PN sequence generator that was used at the transmitter. The bit count and the bit error counts are then tallied. Statistics are compiled in blocks of length #BLOCKLEN. At the end of each block, a single character report is output to the terminal using module PUTC. At the end of each block, the number of bit errors is compared to #ERRLIM. If the number of errors is greater than this threshold, the resynchronization procedures are begun in the next block by

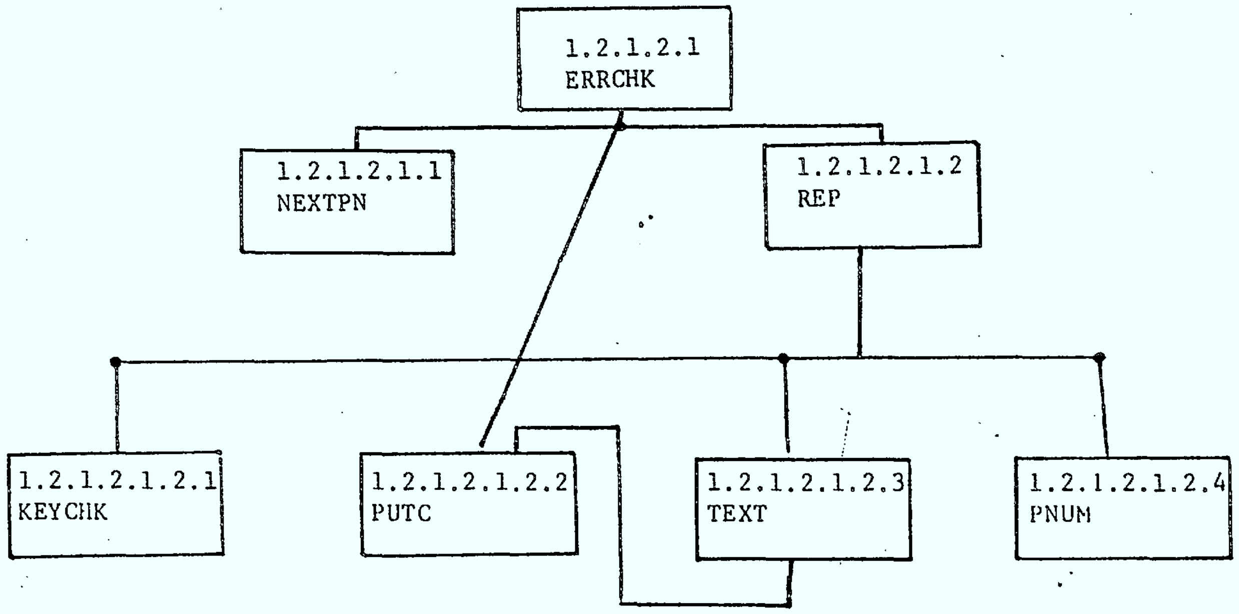

1.2.1.2.1
ERRCHK
1.2.1.2.1.1
NEXTPN
1.2.1.2.1.2
REP
1.2.1.2.1.2.1
KEYCHK
1.2.1.2.1.2.2
PUTC
1.2.1.2.1.2.3
TEXT
1.2.1.2.1.2.4
PNUM

FIGURE 9

resetting the contents of the PN sequence generator in NEXTPN to that of the last v-1 information bit estimates.

The module REP implements the printing of the block error reports and counts on the terminal with text explanations and decimal output.

Appendix A contains the module descriptions and Appendix B contains the assembly language code along with detailed comments.

3.0 VITERBI DECODER PERFORMANCE ISSUES

In this section we present results of two investigations of Viterbi decoder performance. The first investigation deals with the dependence of decoder performance on the values of several algorithm parameter values. Experimental and simulation results are presented for decoders with different number of quantization levels, different history depth values, and with linear and nonlinear quantization. The second investigation considers the performance of Viterbi decoder performance when combined with internal interleaving. Simulation results are presented for systems of different interleaving depths and for channels with different burst error characteristics. We begin the section by describing the simulation model used in our investigations.

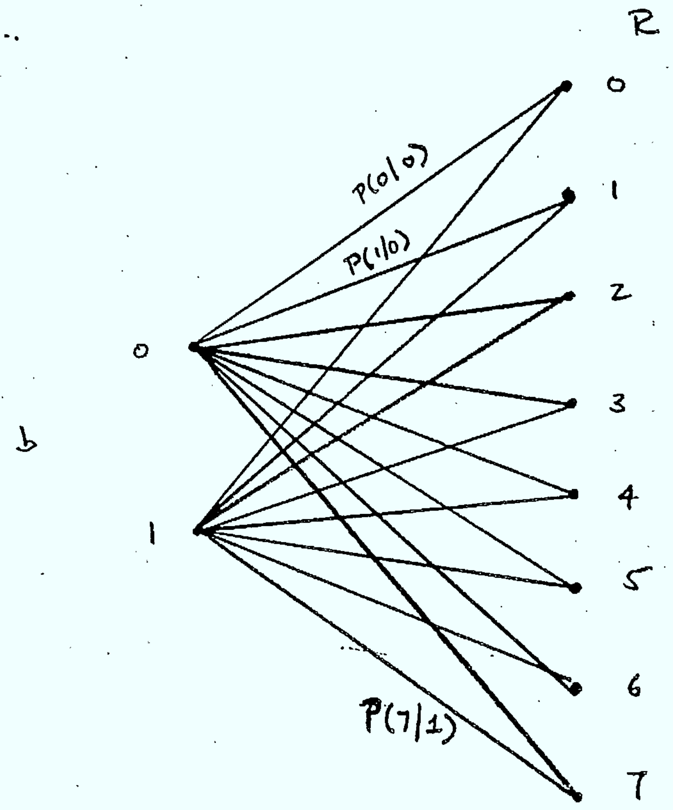
3.1 SIMULATION MODEL

Simulation programs were written to provide a means for quickly and easily testing various Viterbi decoder configurations as well as for simulating various channel transmission conditions. A Viterbi decoder program was written in BASIC to run on the IBM PC. The program completely parallels the implementation of the M6809 real-time decoder in order to allow the simulation of changes in the real-time system. The details of the program therefore do not need to be repeated here.

A second program was written to simulate bursty channel conditions on dual parallel FSK channels. The model simulates independent fading on the two channels as was well as simultaneous (flat) fading on the channels. At any given time instant each channel is in one of three states: Good, Bad, or Flat. At any given time instant the channel pair can be in one of five states:

0	Good-Good
1	Good-Bad
2	Bad-Good
3	Bad-Bad
4	Flat-Flat

While a channel is in the good state, it randomly generates octal output symbols R for each binary input b according to the transition probability shown in Figure 1. The octal output is intended to represent the output of a 3-bit quantizer. Similarly when a channel is in state bad or flat, it generates outputs according to predesignated transition probabilities.



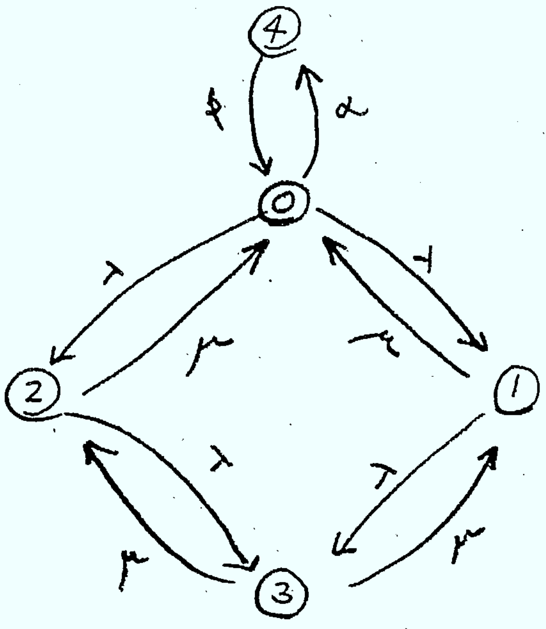


Figure 1

The time evolution of the channel state pair follows a continous-time Markov chain with transition-rate diagram shown in Figure 1. Here λ is the rate at which an individual channel goes from the good state to the bad state, and μ is the transition rate in the opposite direction; α is the rate at which the channel state pair goes jointly from the good-good state to the flat state. The simulation program generates random, exponentially distributed holding times X (in bits) for each state, and then rounds them up to the next integer greater than or equal to X. The next state is selected according to the state transition probabilities that correspond to the transition-rate diagram.

All the simulations discussed in this report simulated independent fading only. The bad state was always represented by the 3-bit quantized Gaussian channel shown in Figure 2a. This (single) channel has a raw bit error rate of .159 and a 128-bit block error rate of essentially 1. The good channel was chosen to be either that shown in Figure 2b or that in 2c. The channel in 2b, hereafter called the good channel, has a raw bit error rate of .023 and a block error rate of 94.7%. The channel in 2c, hereafter called the very good channel, has corresponding rates of .00135 and 15.9% respectively. It should be emphasized that the error rates for these channels are relatively high because they correspond to single channels; when the two channels are combined the performance improves considerably.

The channel parameters used in a given simulation will be specified by prefixing each state with its mean holding time. Thus 250G/50B denotes a channel in which the subchannels fade independently with the good state having a mean holding time of 250 bits, and the bad state a mean holding time of 50 bits. The mean holding time of a state pair is given by the reciprocal of the sum of the rates out of the state pair. Thus the mean holding time of the bad-bad state is 25 bits and that of the good-good state 125 bits.

3.2 EFFECT OF DECODER PARAMETERS ON DECODER PERFORMANCE

The software of the real-time decoder was written so that the number of quantization levels and the history depth could be changed easily. Instructions for carrying out these changes are included in the documentation. An arbitrary nonlinear quantization scheme can be produced by changing a 64-entry table through which the A/D samples are mapped. The convolutional code can also be changed, but this requires changing the longer state organization tables. Here we will report on the results of experiments that vary the number of quantization bits, the history depth, and the quantization mapping.

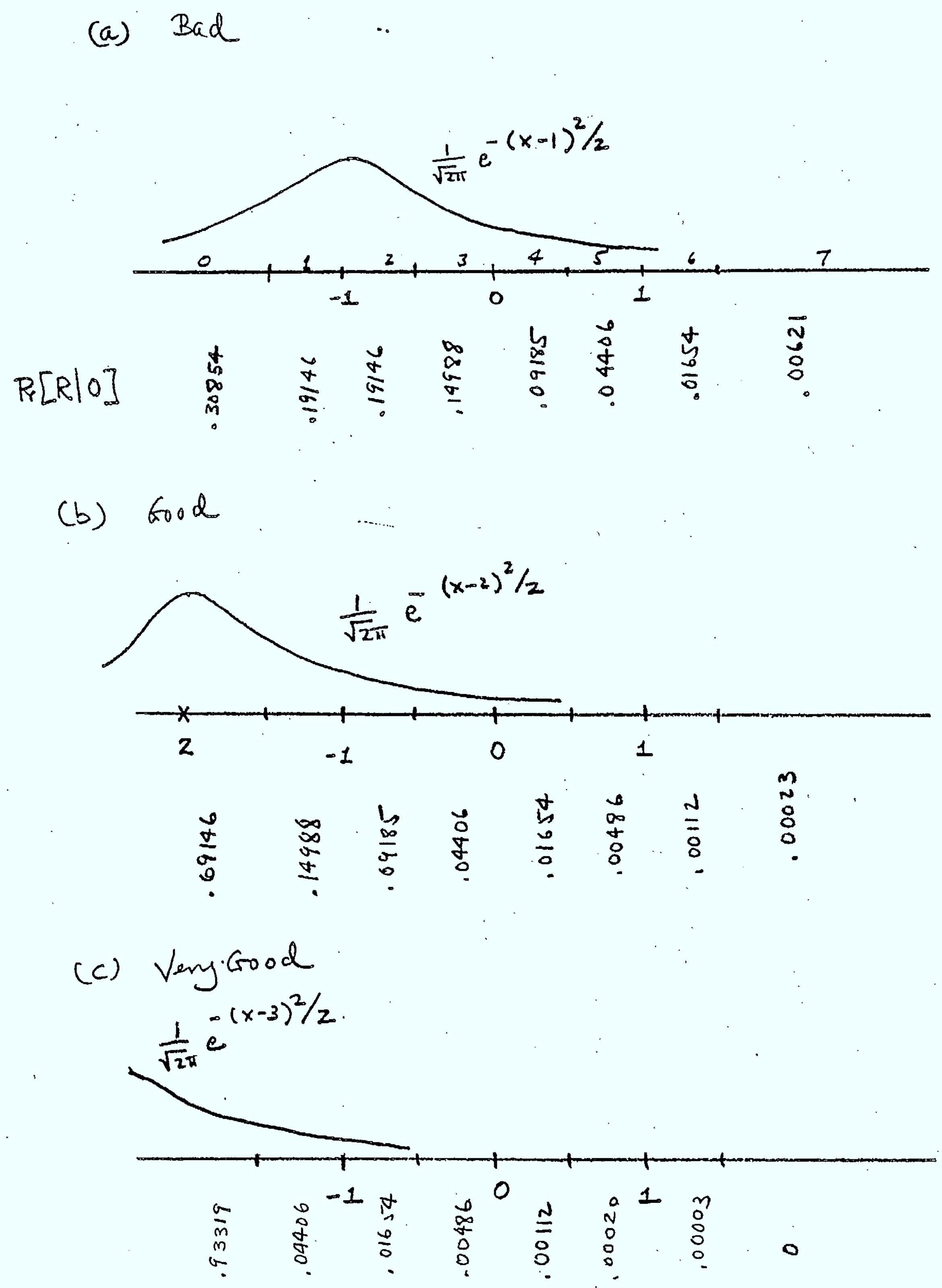

(a) Bad
$\frac{1}{\sqrt{2\pi}} e^{-(x-1)^2/2}$
0 1 2 3 4 5 6 7
-1 0 1
Pr[R|0]
.30854
.19146
.19146
.14988
.09185
.04406
.01654
.00621
(b) Good
$\frac{1}{\sqrt{2\pi}} e^{-(x-2)^2/2}$
2 -1 0 1
.69146
.14988
.09185
.04406
.01654
.00486
.00112
.00023
(c) Very Good
$\frac{1}{\sqrt{2\pi}} e^{-(x-3)^2/2}$
-1 0 1
.93319
.04406
.01654
.00486
.00112
.00020
.00003
0

Figure 2

The software of the real-time decoder includes modules that implement error counting and reporting functions. The block size for block error counts is programmable and was set to be 64 bits. The output of the decoder is arranged in blocks of this size and the number of errors counted. If no errors are found, a dot is printed on the screen and simultaneously stored on floppy disk of an IBM PC running CROSSTALK. If errors are found, the number of errors is compared to a threshold, ERRLIM, which is also programmable. If the threshold is exceeded, a resynchronization operation is initiated in the next block and an 'S' printed. Otherwise the number of errors is printed. The duration of the experiments is also programmable, and the error count is displayed at the end of each experiment. Figure 3 is a sample printout of one of the experiments.

The present system has the following nominal settings: 6-bit quantization, linear mapping, history depth 8, and 133/171 rate 1/2 convolutional code. A series of experiments were conducted where the nominal system was compared to systems in which one of the settings was changed to:

-- 1-bit quantization
-- 3-bit quantization
-- Mu-law mapping
-- Inverse Mu-law mapping
-- history depth 4
-- history depth 6

Each experiment involved changing the software and then demodulating and decoding the (approximately) same segment of tape recorded audio signal corresponding to 1728 blocks of information. To control for fluctuations in the results due to factors other than the change in parameter, the nominal system was interspersed among the other experiments.

Each experiment was run twice. Table 1 shows the results of each experiment. The experiments are listed in the order in which they were carried out because significant variations were observed in the performance of the control (nominal system) experiment. The results are also displayed in Figue 4 where it can be seen that the control experiment varied in error rate from 0.75% to 2.78%. The error rates of all the other experiment except one fell in this range. The one exception was for the system with 1-bit quantization which was clearly inferior. The printout for one of the 1-bit experiments is shown in Figure 5. This printout can be compared to that of Figure 1 which corresponded to the control experiment that had the best performance. Thus the only conclusion that can be made from the experimental results is that 1-bit quantization is significantly inferior to soft decision systems.

The simulation programs were used to investigate the effect

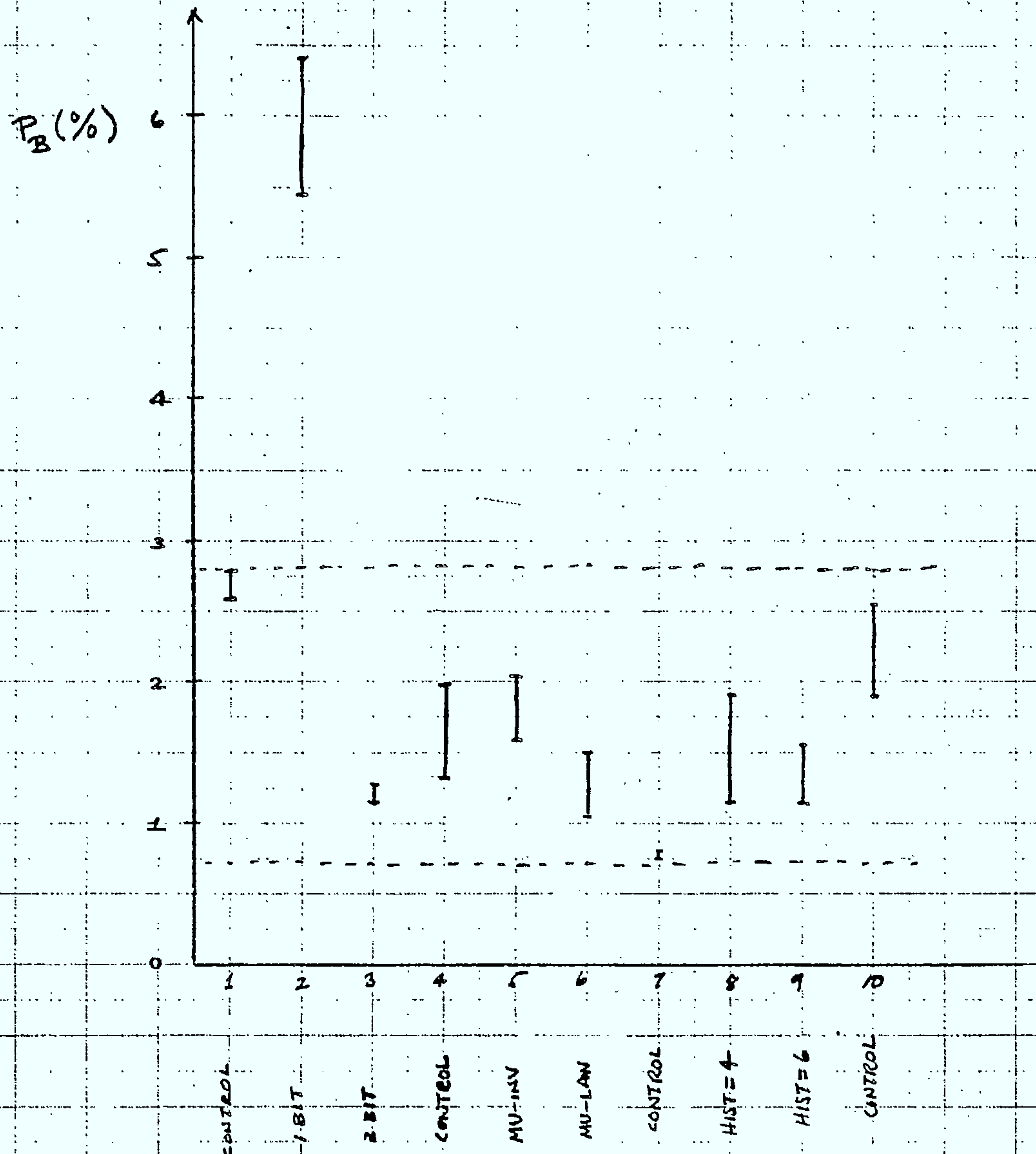
```
*EXPT. 6.1: NOMINAL, MAPPING = LINEAR.
*J
SSS1S
................................................................
................................................................
................................................................
................................................................
.........4......................................................
................................................................
..8.............................................................
................................................................
................................................................
................................................................
....................8...........................................
................................................................
................................................................
................................................................
................................................................
................................................................
....................................1...........................
................................................................
................................S.2.............................
...................................42...........................
...............4...........................................3....
......................................................3.........
..5.............................................................
................................................................
................................................................
................................................................
......5.........................................................

TOTAL BLOCKS READ 1728
NUMBER OF ERROR FREE BLOCKS 1715
NUMBER THAT HAD BAD BITS 12
NUMBER THAT CAUSED RE SYNC 1
```

Figure 3

1.	CONTROL	48/1728 = 2.78%
		45/1728 = 2.60%
2.	1 Bit Qtn.	111/1728 = 6.42%
		94/1728 = 5.44%
3.	3 Bit Qtn.	20/1728 = 1.16%
		22/1728 = 1.27%
4.	CONTROL	23/1728 = 1.33%
		34/1728 = 1.97%
5.	MU-INV	35/1728 = 2.02%
		29/1728 = 1.68%
6.	MU LAW	26/1728 = 1.50%
		18/1728 = 1.04%
7.	CONTROL	13/1728 = 0.75%
		14/1728 = 0.81%
8.	HIST = 4	33/1728 = 1.91%
		20/1728 = 1.16%
9.	HIST = 6	20/1728 = 1.16%
		27/1728 = 1.56%
10.	CONTROL	33/1728 = 1.91%
		44/1728 = 2.55%

Table 1.



P_B(%)
6
5
4
3
2
1
0
1
2
3
4
5
6
7
8
9
10
CONTROL
1 BIT
2 BIT
CONTROL
MU-INV
MU-LAN
CONTROL
HIST = 4
HIST = 6
CONTROL

Figure 4

```
*EXPT. 1.2: 1 BIT.
*J
S
...............1...2S......................2S...................
...................59.......S............................6......
................................................................
...9.......71......3..................................2....3....
4........8......................................................
...........5...............................3.......3.....S.8....
...........................3...............................5....
...........................................2....................
..............................................S.................
...........3......9........6..............3.......7.............
..37......SS...........3..3.......SS.......................7....
..........................3.......SS..............SS............
......1...3................2......S.......3...............3.....
..........3.......................................S.............
..................S..........S..................................
..........73....................................................
...........................9....................................
.........1SS.....21..............4.......95.....................
.........SS..............SS..............9......................
.6.......SS...............8.....................................
......4..........6........S.......................6.............
.8.......7...................................SSS........SS...3..
.....2...........................................3..............
................................4...............................
.................S..............................................
........................SS...................................4..
................................................................

TOTAL BLOCKS READ 1728
NUMBER OF ERROR FREE BLOCKS 1634
NUMBER THAT HAD BAD BITS 60
NUMBER THAT CAUSED RE SYNC 34
```

Figure 5

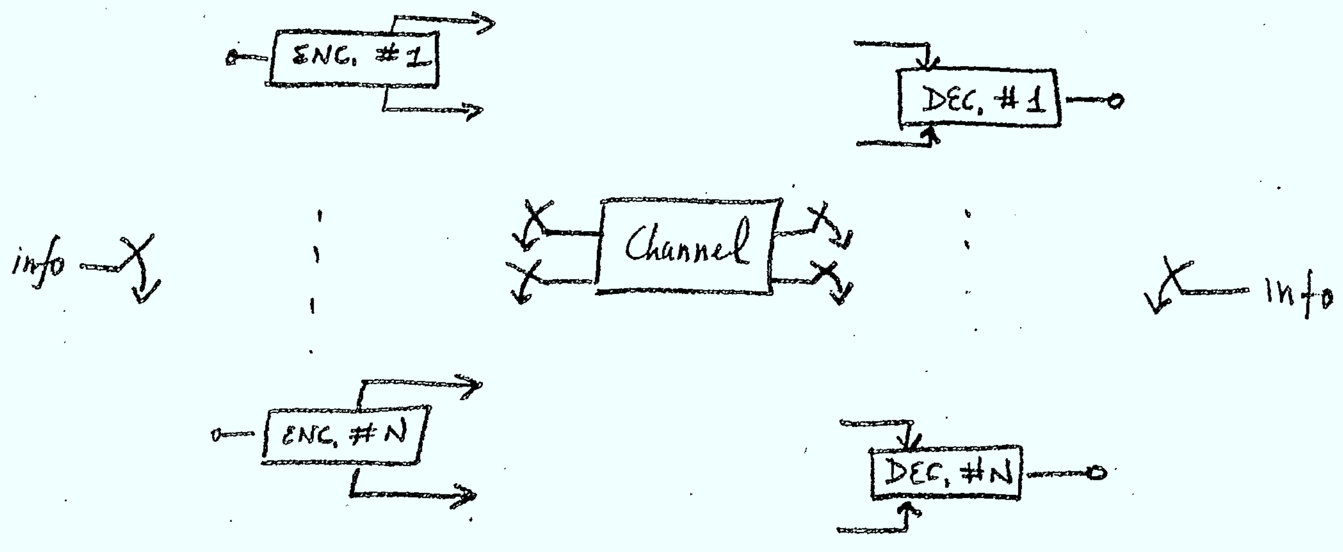
of decoder parameter settings on the error rate performance. It was expected that the simulation results would give a better indication of the relative importance of each setting since it is easier to control for variations in each trial. Each simulation was run for 40,000 bits which had previously been established to be sufficient to produce a representative relative frequency count for each of the state pairs on the 250G/50B channel introduced in section 3.1. The 40,000 bits correspond to approximately 330 128-bit blocks. The running time of each simulation on the IBM PC running compiler BASIC is approximately 4 and 1/2 hours. The performance of the decoder for the 171/133 code is shown below:

setting:	P_B	P_b
1-bit qtn, hist 8	9.25%	8.00×10^{-3}
3-bit qtn, hist 2	7.76%	2.90×10^{-3}
3-bit qtn, hist 4	4.20%	3.73×10^{-3}
3-bit qtn, hist 8	4.52%	2.88×10^{-3}

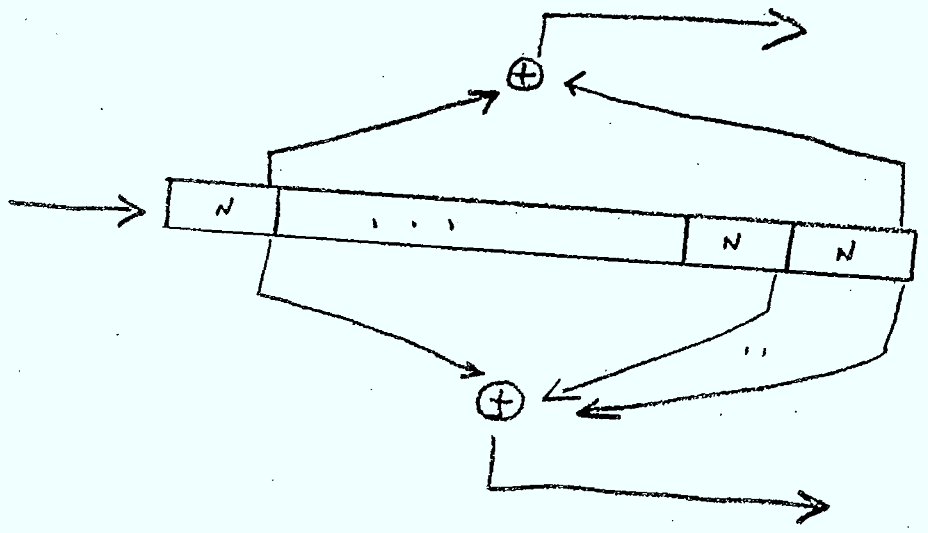
Single bit quantization again has the worst performance in both bit- and block-error rate. Decreasing the history depth from 4 to 2 results in an increase in block error rate, but decreasing from 8 to 4 does not result in a significant change (the two simulations differ by 1 block error only). Decreasing the history depth does not necessarily increase the bit error rate. The printouts of the simulations revealed that the history depth 2 system had numerous short bursts of errors whereas the longer history systems had fewer but much longer bursts. From a block error rate point of view a history depth 2 system is not unacceptable. As well, it appears that the history depth can be decreased from 8 without incurring a loss in block error rate performance.

3.3 VITERBI DECODING WITH INTERLEAVING.

In this section we present results on the performance of Viterbi decoding with internal interleaving. The basic idea of internal interleaving is to split the transmission of data into a number of parallel streams that are encoded and decoded separately as shown in Figure 6a. In practice it is not necesary to replicate the encoders and decoders, but instead the logic needed to carry out these operations is time-shared among the N streams. Consequently interleaving is achieved without introducing a frame structure and at only a linear increase in the memory requirements. Indeed the combined encoder-interleaver is equivalent to the longer constraint length code shown in figure 6b. It can be shown that this



(a)



(b)

Figure 6

code has the same minimum distance as the shorter code, so that its performance in a random error environment will be the same. In a burst error environment, however, the longer code will have the better performance.

The same computer simulation programs discussed in the previous section were used to investigate Viterbi decoding with internal interleaving. It is clear that only one of the encoder-decoder pairs needs to be simulated so no change is required in the Viterbi decoder program. The only change required in the channel simulation program is that only every L-th output of the channel is passed to the decoder if the degree of interleaving is L.

In the simulation the channel was considered to be in a burst error mode if the channel was in the bad state. The burst error statistics were specified by fixing the mean holding time in the bad state, and the percent of the time spent in the nonburst mode was specified by selecting the mean holding time to be some multiple of that of the bad state. Each simulation consisted of approximately 330 128-bit blocks. In the first set of simulations the mean burst error length was 50 bits and the proportion of time spent in the burst mode was 1/6. By introducing degree 5 interleaving, the effective channel seen by the decoder is approximately a 50G/10B channel. The block error rate is reduced by a factor of 3.5 and the bit error rate by a factor of 6.5. (See Table 2.) The third experiment in this set simulated a 50G/10B channel without interleaving. It can be seen that the results are quite close to those of the channel with the same effective parameters after degree 5 interleaving.

In the second set of experiments, the mean burst length was increased to 100 bits and the proportion of the time spent in the burst mode was reduced to 1/3. The system without interleaving performs quite poorly and the introduction of interleaving introduces only some improvement. The degree 10 interleaving system has a mean holding time of 5 bits in the bad-bad state, the same as the degree 5 system in the first set of experiments. The mean holding time in the good-good state however is only 10 bits so the decoder does not have enough good samples, on the average, to get back on the correct path.

For a given fixed ratio of time spent in the good state to time spent in the bad state, the Viterbi decoder will span the following two extremes. When the bursts are very long the decoder will encounter 4 modes of sustained duration corresponding to the 4 possible state pairs. The performance will be the weighted average of the performance during each of these modes. For the cases considered above the performance will be dominated by that of the bad-bad state. The bit error

Channel	N	Effective channel	Block Error Rate	Bit Error Rate
250G/50B	1	250G/50B	4.5%	2.9×10^{-3}
250G/50B	5	~ 50G/10B	1.2%	4.3×10^{-4}
50G/50B	1	50G/10B	0.97%	2.8×10^{-4}
200G/100B	1	200G/100B	11.7%	1.4×10^{-2}
200G/100B	5	40G/20B	8.1%	5.2×10^{-3}
200G/100B	10	20G/10B	5.4%	2.6×10^{-3}

Table 2.

rate for this code has been found to be .14 by simulation. The proportion of time spent in the bab-bad state for the first set of experiments is 1/36, so the average bit error rate should be about 3.9x10-3 when the burst length is very long. This is close to the figure obtained for the 250B/50B system without interleaving. For the second set of experiments the predicted bit rate is 1.55x10-2 which again is quite close to that of the system without interleaving.

As the degree of interleaving is increased until it exceeds the mean burst length, the decoder will reach a plateau with performance of that of a system in which the channel randomly selects one of the two modes (good or bad) with probability equal to the proportion of time spent in the given modes. No attempt was made to establish what the limiting performance for this extreme would be.

4.0 TRANSMISSION PROTOCOLS FOR MULTITONE MODEM

An objective of the project was to extend the present transmission protocol to increase the information rate that can be reliably achieved using a multi-FSK signal format and combining code diversity with ARQ techniques. In this section we present an analysis of the present transmission protocol; then we present an analysis of a protocol proposed by Shu Lin and an adaptive protocol that combines the Lin protocol with a code diversity ARQ system.

4.1 PRESENT TRANSMISSION PROTOCOL

The present transmission protocol defines a sequence of packet exchanges that effect the transfer of a message from a sender terminal to an acceptor terminal. The channel is shared by both terminals so transmissions are constrained to be half-duplex with packets travelling in opposite directions alternately accessing the channel. The terminals set up the transfer of a message by exchanging calling and response packets. The message transfer is carried out through the exchange of message and acknowledgment packets. The message transfer is ended with the sender terminal transmitting termination packets.

The present protocol handles the transmission of one message at a time. A message may consist of up to 1280 bytes. Prior to transmission the message is segmented into subblocks of 16 bytes. The acceptor terminal is given the number of such subblocks during the set up phase. Each subblock has a sequence number byte and a CRC byte attached to it to form a subpacket, which forms the basic unit of retransmission. Subpackets are transferred to the acceptor terminal in fixed-length frames that can accomodate 8 subpackets. Each frame consists of a header followed by 8 subpackets, with padding used to fill up unoccupied subpacket slots if necessary. Thus by design each subpacket is bundled separately and thus its retransmission can be handled separately from that of other subpackets.

Fixed-length frames, called message packets, and fixed-length acknowledgment packets alternate in using the channel. The message packets consists of an 11.5 byte header followed by the 8 subpackets. Of the 1244 bits in the message packet at most 1024 can be used for information. Acknowledgment packets of 148 bits follow each message packet transmission. The propagation and software delays add an equivalent of 10 bits at 100 bps transmission rate. Barring errors, after the exchange of one message packet at most 1024 information bits will have been transferred in the time 1402 bits could have been transmitted on the channel. Thus the present protocol has a maximum throughput efficiency of 73%.

We now present an analysis of the present transmission protocol. The purpose of carrying out such an analysis is to lay the framework within which any extended protocol should be evaluated. In this analysis we will neglect the effect of errors in the acknowledgment packets. Let Kbe the message length in subpackets, $1 \leq K \leq 80$, and suppose that a frame can accomodate up to L subpackets. Let e be the probability that a subpacket is received in error (using hard decisions) and assume that subpacket errors occur independently. The sequence of message packet transmissions can be divided into two phases: in mode A the number of outstanding subpackets at the transmitter is greater than or equal to L so frames carry a full set of subpackets in each transmission; in mode B the number outstanding is less than L so each frame transmitted is partly empty. Clearly mode B transmissions introduce inefficiencies in the use of bandwidth and the purpose of the analyses is to quantify these inefficiencies.

Suppose that $K = QL + R$, where $0 \leq R < L$; then at least Q mode A transmissions will be required. Let MF(K) be the total number of mode A transmissions rquired for a message of length K. In Appendix C we show that the mean of MF(K) is:

$$E(MF(K)) = Q + \sum_{i=Q}^{\infty} \left(\sum_{j=0}^{K-L} Pr(s_i = j) \right)$$

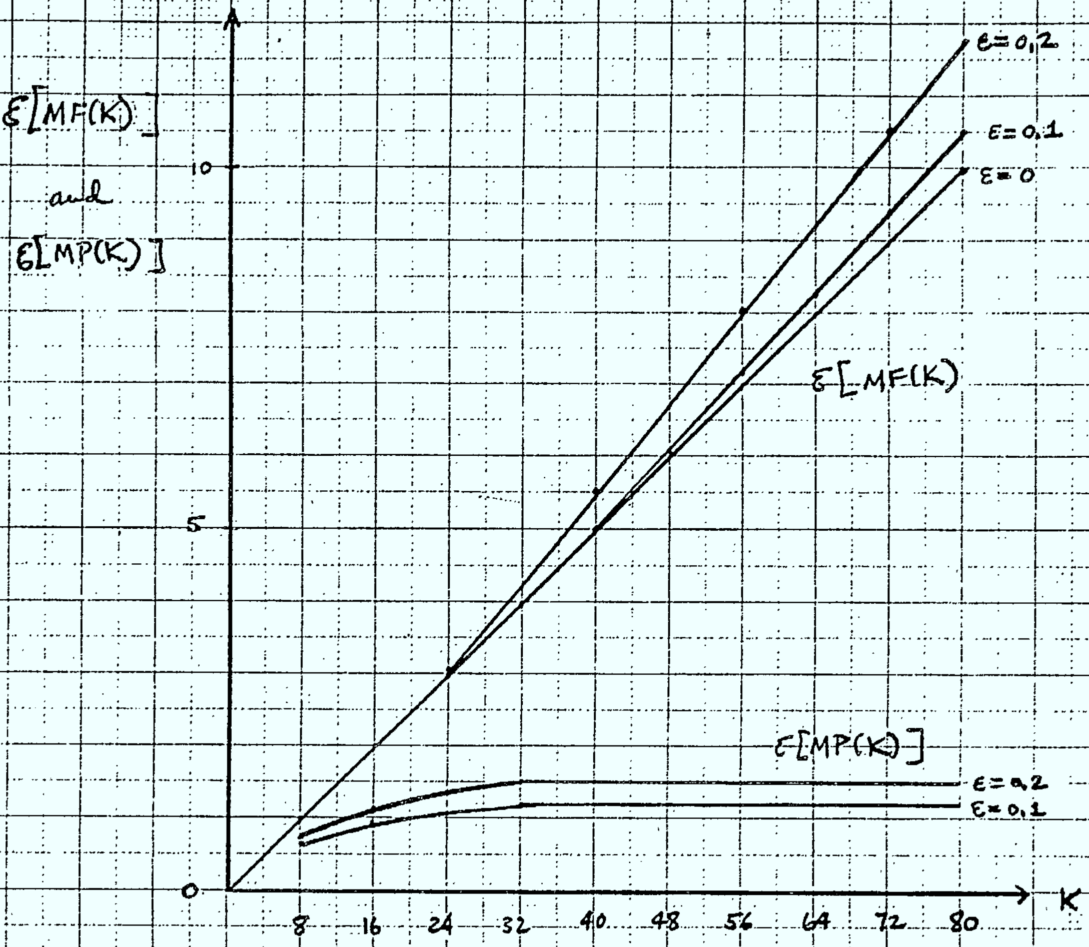
where s_i is a Binomial random variable with parameters iL and e. The over the Binomial terms can be approximated by the error function (i.e. integral over a Gaussian) and only a few terms in the infinite series turn out to be significant. Figure 1 displays the mean of MF(K) versus K.

Mode B transmissions will commence when the number of subpackets remaining for transmission becomes less than L. Suppose that this number is r, $0 \leq r < L$. Partly empty frame transmissions will continue until all r subpackets have successfully been transferred to the acceptor terminal. Let MP(r) be the number of frame transmissions required to accomplish this. In Appendix C we also show that the mean of MP(r) is given by:

$$E(MP(r)) = 1 + \sum_{j=1}^{r} \binom{r}{j} \frac{e^j}{1-e^j} (-1)^{j+1}$$

Figure 2 shows E(MP(r) as a function of r. As expected the number of transmission increases with r. Since the range of r depends on L, Figure 2 can be used to quantify the loss in efficiency due to too large a value of L.

The number of outstanding subpackets r during the first mode B transmision is a random variable that depends on K.



ε[MF(K)]
and
ε[MP(K)]
10
5
0
ε=0,2
ε=0,1
ε=0
ε[MF(K)]
ε[MP(K)]
ε=0,2
ε=0,1
K
8
16
24
32
40
48
56
64
72
80

Figure 1.

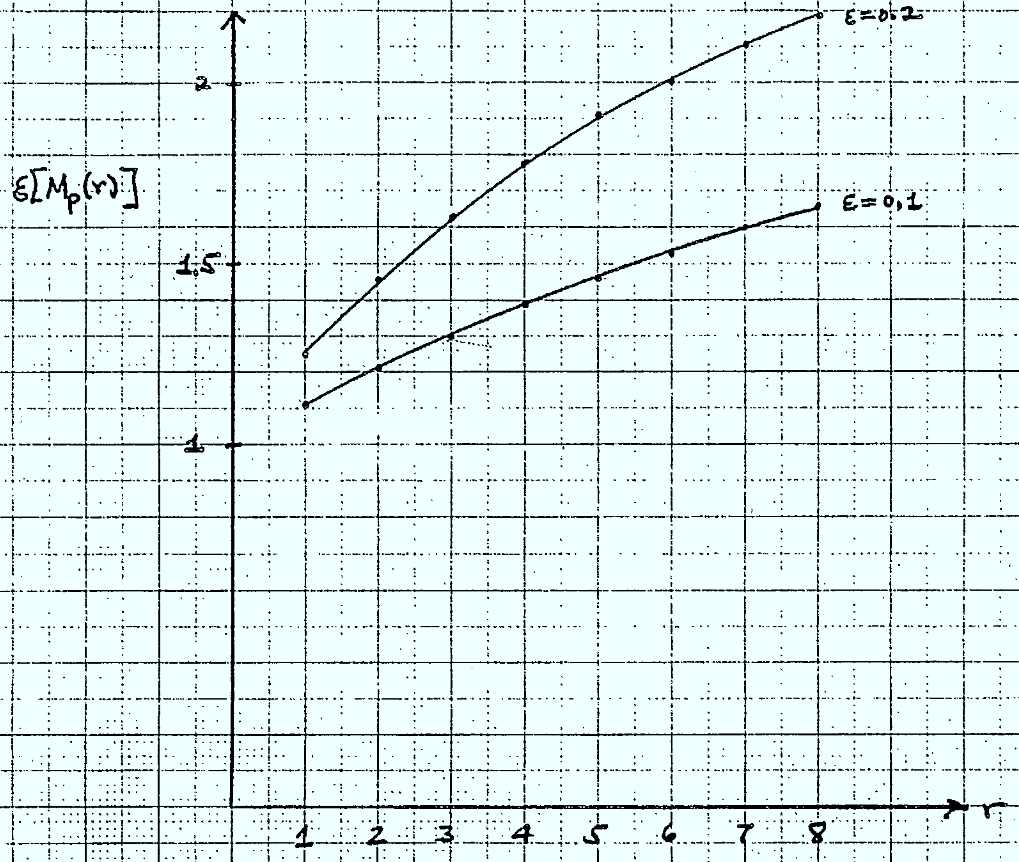


Figure 2

In the Appendix we derive the distribution of r conditioned on K, and we then compute the mean of MP(r) averaged over r. For large values of K, r becomes uniformly distributed in the interval $0 \leq r \leq L-1$. Figure 1 shows this mean as a function of K.

The two curves in figure 1 can be added to obtain the mean number of total frame transmissions required to transfer a message of length K, that is E(M(K)). The throughput efficiency is then given by:

$$\text{EFF} = \frac{K}{E(M(K))(L+H)}$$

$$= \left(\frac{K/L}{E(M(K))}\right)\left(\frac{L}{L+H}\right)$$

$$= \text{EFFREL} \times \text{MAX EFF}$$

where H is the overhead incurred in each message packet/acknowledgment cycle, and where EFFREL is defined as the relative efficiency. Figure 3 shows the relative efficiency as a function of K. It can be seen that as K increases the relative efficiency approaches that of selective repeat ARQ, namely 1 - e. In effect for large K, the inefficiencies due to mode B transmissions are negligible. Thus if we are considering very long messages, we can directly analyze the transmissions of subpackets and ignore the frame structure of the system.

The relative efficiencies shown in Figure 3 correspond to three values of subpacket error probabilities, e=0, e=0.1, and e=0.2. It can be seen that for this range of values the relative efficiencies do not differ greatly. In experiments conducted last year subblock error rates of 10-2, 10-1, and 2x10-1 were observed for code diversity, frequency diversity, and single channel transmissions. One concludes from Figure 3, that the combination of code diversity with selective repeat ARQ, henceforth denoted by ARQCD, for this range of error probabilities is not worthwhile since code diversity incurs 50% overhead, and thus a higher throughput is achievable if this overhead were replaced by additional subpacket transmissions. On the other hand, as the error rate increases diversity transmission can be expected to keep the througput from deteriorating to zero much longer than single channel FSK because of its ability to correct errors. Clearly what is needed is a transmission protocol that dynamically varies between these two extremes as the channel error conditions vary. We will introduce such a protocol in the last section.

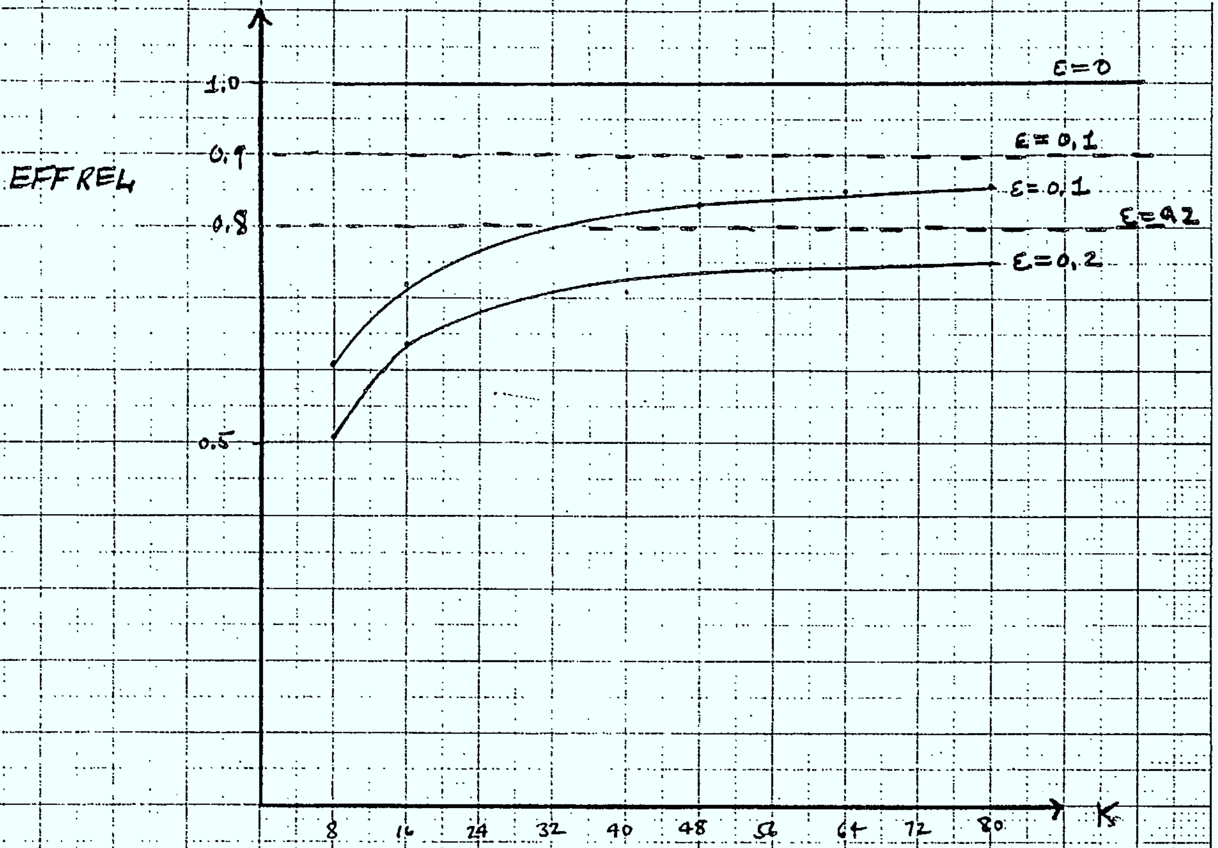


Figure 3

4.2 A DIVERSITY-ON-DEMAND TRANSMISSION PROTOCOL

Suppose that M FSK signals are available for information transmission and that each signal is organized as in the existing transmission protocol but with the headers combined into one single header as shown in Figure 4. The first part of the header up to the software sync byte would remain the same. The second part would be capable of carrying more information than presently required so it could be shortened or modified as necessary. Similar considerations would apply to the acknowledgment packets.

Once the message packets header and acknowledgment packet lengths are selecеted (and here we are assuming that they will remain fixed), the maximum achievable throughput will be fixed and the particular form of ARQ will affect the performance only through the relative efficiency. We will assume that the transmitted messages are very long so that mode B transmissions can be neglected. The relative efficiency will be given by E(N), where N is the number of transmissions required by a given subpacket.

The basic mechanism of ARQ schemes, retransmision, is essentially a diversity technique, namely time diversity without combining. Viewed this way it is clear that ARQ schemes do not make use of all of the information available at the receiver and that better performance should be possible by using some form of combining. We will consider the use of code diversity as the means of utilizing the extra information. The scheme considered has been discussed by Wang and Lin (Trans. on Communications, May 1983). We will henceforth refer to the scheme as ARQDD.

For throughput analyses purposes we can consider the protocol as if a single subpacket is being retransmitted at a time. The protocol employs a cyclic code for error detection and a rate 1/2 convolutional code for code diversity. Each information subpacket (including the sequence number) is encoded using the cyclic code. The resulting block is convolutionally encoded with the encoder intialized to the zero state and enough zeros appended to the block so as to drive the encoder back to the zero state. Each subpacket is thus bundled separately so that its retransmission can be handled independently of that of other subpackets.

The upper and lower branches of the encoder output are buffered separately and only the upper branch is transmitted at first. At the receiver the received sequence is divided by the appropriate polynomial that allows the recovery of the cyclically encoded block when errors have not been introduced in transmission. The outcome of this division and the subsequent check of the recovered block are used to detect errors. If none are found the subpacket is accepted. If the block is found to have some errors, the receiver sends a negative acknowledg-

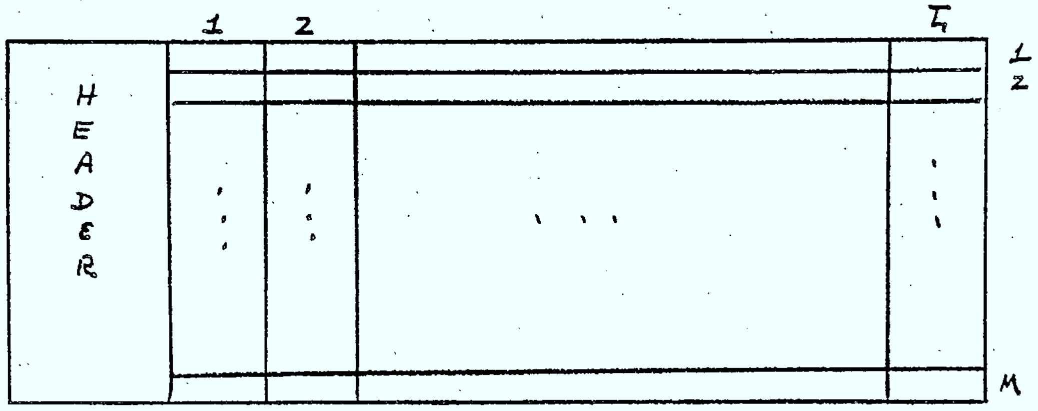


figure 4

ment requesting a transmission of the other branch. The second branch is processed in the same way: it is divided, checked for errors and accepted if none are found. If at this point errors are found, both received branches are used to carry out Viterbi decoding. The sequence output by the decoder is then checked for errors using the cyclic code. At no extra cost in bandwidth we have obtained an extra opportunity to correctly decode the information. Note that all decoding prior to Viterbi decoding uses hard decisions, but that Viterbi decoding itself can use soft decisions.

If the second transmission and subsequent Viterbi decoding fail, a request for a retransmission of the first branch is made. The branch is processed as the second branch was processed in the previous step. In Lin's version of the protocol, the older version of a branch is discarded. However if soft decision decoding is used, the old and new versions of a branch could be combined prior to Viterbi decoding. The alternate transmission of branches continues until the subpacket is successfully transferred or until some upper limit is reached.

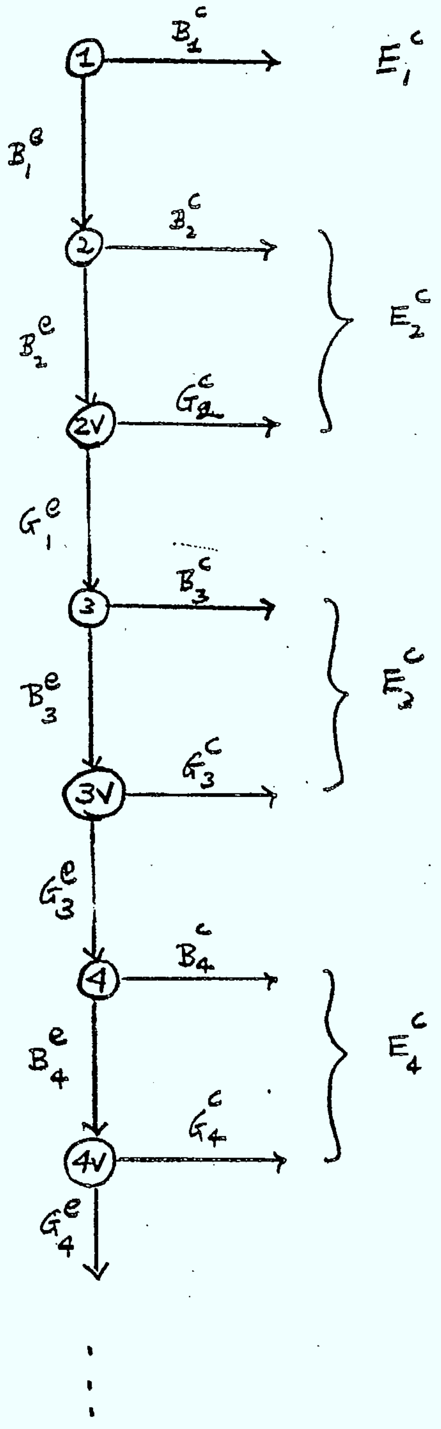
The possible sequence of events for ARQDD is shown in Figure 5 where Bc(i) is the event that the ith transmission is received errorfree and Be(i) is the event that it is found in error. We will assume for simplicity that the probability of failing to detect errors is negligible relative to the probabilities of the other events. Gc(i) is the probability that Viterbi decoding successfully decodes a pair of erroneous branches after the ith transmission. Assume that block errors occur at independently and with the same probability:

$$P_c = \Pr(\ Bc(i)\ )$$

and

$$1 - P_c = \Pr(\ Be(i)\ )$$

Let V_o^c be the probability that the first Viterbi decoding is successful and let V^c be the probability that subsequent Viterbi decodings are successful. Note that the first probability is conditioned on the two branches having had errors detected on them, whereas the second probability in addition has the condition that one of the branches participated in a previous unsuccessful Viterbi decoding. Thus the second probability will be less than the first. The corresponding sequence of event probabilities is shown in Figure 6. To calculate the mean number of transmissions we need only consider the event Ec(i) and the event Ee(i) that correspond to the ith transmission being successful and unsuccessful respectively. Since Ee(i) occurs if the ith transmissions has errors and the subsequent Viterbi decoding fails, we have that



1
B_1^c
E_1^c
B_1^e
2
B_2^c
E_2^c
B_2^e
2V
G_2^c
G_1^e
3
B_3^c
E_3^c
B_3^e
3V
G_3^c
G_3^e
4
B_4^c
E_4^c
B_4^e
4V
G_4^c
G_4^e

Figure 5

$$\begin{tikzcd}
1 \arrow[r, "P_C"] \arrow[d, "P_B"'] & P_C \\
2 \arrow[r, "P_C"] \arrow[d, "P_B"'] & P_B P_C \\
2V \arrow[r, "V_C^0"] \arrow[d, "V_B^0"'] & P_B^2 V_C^0 \\
3 \arrow[r, "P_C"] \arrow[d, "P_B"'] & P_B^2 V_B^0 P_C \\
3V \arrow[r, "V_C"] \arrow[d, "V_B"'] & P_B^3 V_B^0 V_C \\
4 \arrow[r, "P_C"] \arrow[d, "P_B"'] & P_B^3 V_B^0 V_B P_C \\
4V \arrow[r, "V_C"] \arrow[d, "V_B"'] & P_B^4 V_B^0 V_B V_C \\
\vdots &
\end{tikzcd}$$

Figure 6

$$\Pr(\,Ee(i)\,) = \Pr(\,Be(i)\,)\ \Pr(\,Ge(i)\,)$$

$$= (\,1 - P_c)\ (1 - V^c\,)$$

The simpler sequence of event probabilities is shown in Figure 7. The probabilities of N are given by:

$$\Pr(N=i) = \begin{cases} P_c & i=1 \\ P_B(\,P_c + V_o^c\,) & i=2 \\ P_B^{i-1} V_B^o\ V_B^{i-3}\ (P_c + P_B\ V^c\,) & i \geq 3 \end{cases}$$

where $P_B = 1 - P_c$ and $V_B = 1 - V_c$. The mean number of transmissions is then found to be:

$$E(N) = P_c + 2P_B(P_c + P_B V_c^o) + \frac{1 + 2(P_c + P_B V_c)}{P_c + P_B V_c}$$

and the relative throughput efficiency is then $1/E(N)$.

The evaluation of the throughput efficiency requires the transmission error probability P_B, and the two probabilities of successful Viterbi decoding, V_o^c and V_c. Note that P_B is the error probability that results using hard decisions. Later in this section we will estimate these parameters for a bursty channel by computer simulation. We will find that V_c requires much computation to estimate so it would be useful to have bounds on E(N) that depend only on the other two parameters.

An upper bound is obtained for E(N) by analyzing the inferior system posed by Lin where Viterbi decoding always uses a new pair of branches and thus can take place only during every other transmission. The sequence of event probabilities is shown in Figure 8 and the resulting mean is an upper bound to E(N):

$$E(N) < \frac{1 + P_B}{1 - P_B^2\ V_B^o}$$

We can also obtain a lower bound as follows. Since V_c is always less than V_o^c, we can replace V_c by V_o^c in the exact formula for E(N) and obtain an optimistic estimate. Combining the upper and lower bounds we have:

$$\begin{tikzcd}
\textcircled{1} \arrow[r] \arrow[d, "P_B"'] & P_C \\
\textcircled{2} \arrow[r] \arrow[d, "P_B V_B^0"'] & P_B(P_c + P_B V_c^0) \\
\textcircled{3} \arrow[r] \arrow[d, "P_B V_B"'] & P_B^2 V_B^0 (P_c + P_B V_c) \\
\textcircled{4} \arrow[r] \arrow[d, "P_B V_B"'] & P_B^3 V_B V_B^0 (P_c + P_B V_c) \\
\textcircled{5} \arrow[r] \arrow[d] & P_B^4 V_B^2 V_B^0 (P_c + P_B V_c) \\
\vdots &
\end{tikzcd}$$

Figure 7

$$\begin{tikzcd}
\boxed{1} \arrow[r, "P_c"] \arrow[d, "P_B"'] & P_c & \\
\boxed{2} \arrow[r, "P_c"] \arrow[d, "P_B"'] & P_B P_c & \\
\boxed{2V} \arrow[r, "V_c^0"] \arrow[d, "V_B^0"'] & P_B^2 V_c^0 & P_B(P_c + P_B V_c^0) \\
\boxed{3} \arrow[r, "P_c"] \arrow[d, "P_B"'] & P_B^2 V_B^0 P_c & \\
\boxed{4} \arrow[r, "P_c"] \arrow[d, "P_B"'] & P_B^3 V_B^0 P_c & \\
\boxed{4V} \arrow[r, "V_c^0"] \arrow[d, "V_B^0"'] & P_B^4 V_B^0 V_c^0 & P_B^3 V_B^0 (P_c + P_B V_c^0) \\
\vdots & &
\end{tikzcd}$$

Figure 8

$$1 + \frac{P_B}{1 - P_B V_B^o} < E(N) < \frac{1 + P_B}{1 - P_B^2 V_B^o}$$

When the channel is very good we have P_B much less than 1 and the bounds yield

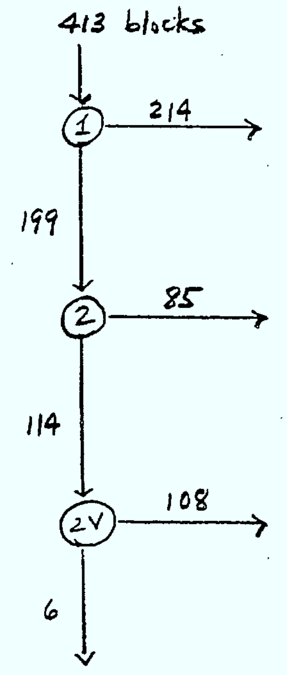
$$E(N) \approx 1 + P_B$$

which intuitively agrees with the fact that Viterbi decoding would seldom be required. When the channel is very noisy the P_B approaches 1 and

$$\frac{2 + V_B^o}{1 - V_B^o} < E(N) < \frac{2}{1 - V_B^o}$$

In order to obtain the parameters required to estimate the performance of ARQDD, we modified the simulation programs to estimate the required parameters. An information block of 120 bits (all zeros in the simlation) was "encoded" using the 171/133 code and the resulting encoded branches (two blocks of 126 zeros) were produced. The first branch was passed through the bursty channel. The resulting sequence was inspected for hard errors and if none was found the block was counted as correct on the first transmission. Otherwise a new branch was "requested." In producing the new branch the initial state of the channel was reselected at random in order to simulate the time diversity nature of the retransmissions. The ARQ protocol was followed in the prescribed way and a tally of the various events was kept. Table 1 shows the results of three simulation experiments and Figure 9 shows the corresponding results in graphical form.

In the first two experiments the channels alternate between periods of good error conditions and periods of bad error conditions. The first experiment corresponds to the 250VG/50B channel introduced earlier in the report. In this experiment the channel is relatively "very good" in that a significant (51%) of the blocks are recieved error free after the first transmission. The bounds for ARQDD are very tight in this case and the throughput for ARQDD is 67%, a 16% improvement over selective repeat ARQ which would have a throughput of 51%. In the second experiment the channel is 250G/50B and only 4% of the packets get through on the first transmission. Thus the throughput has collapsed to nearly zero for selective repeat ARQ. The bounds for ARQDD are again very tight and



$P_c = \frac{214}{413} = 51\%$

$V_c^o = \frac{108}{114} = 95\%$

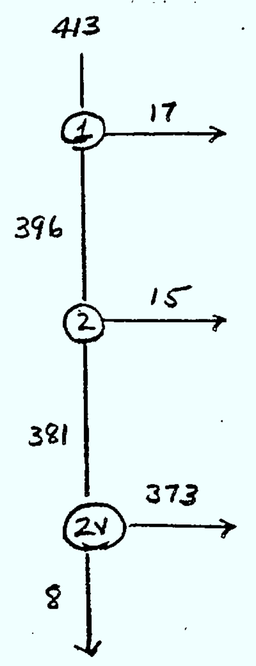
ARQCD → $\frac{1}{2}V_c^o = 48\%$

$1{,}492 < \mathcal{E}[N] < 1{,}497$

ARQDD → $\eta = 67\%$

250VG / 50 B

# 1



$P_c = \frac{17}{413} = 4\%$

$V_c^o = \frac{373}{381} = 98\%$

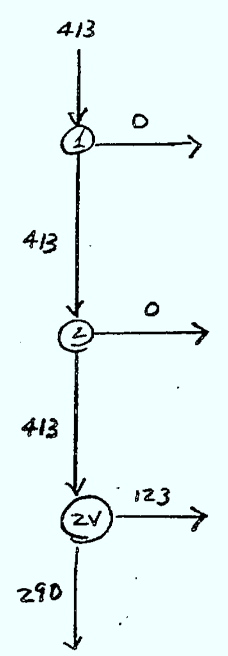
$\frac{1}{2} V_c^o = 49\%$

$1{,}979 < \mathcal{E}[N] < 1{,}997$

$\eta = 50\%$

250 G / 50 B

# 2



$P_c = 0\%$

$V_c^o = \frac{123}{413} = 30\%$

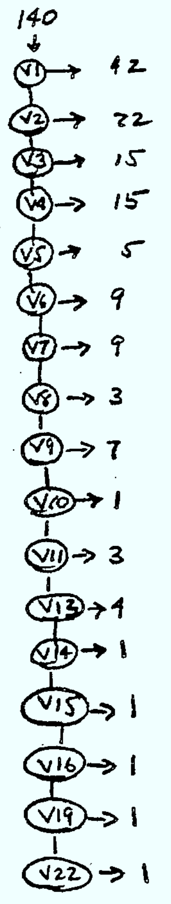
$\frac{1}{2} V_c^o = 15\%$

$4{,}33 < \mathcal{E}[N] < 6{,}67$

$15\% < \eta < 23\%$

250B / 50B

# 3



$\mathcal{E}[N \mid N>1] = 4.70$

$V_c = \frac{1}{4.70} = .2126$

$\mathcal{E}[N] = 5{,}29$

$\eta = 19\%$

# 3a

Table 1

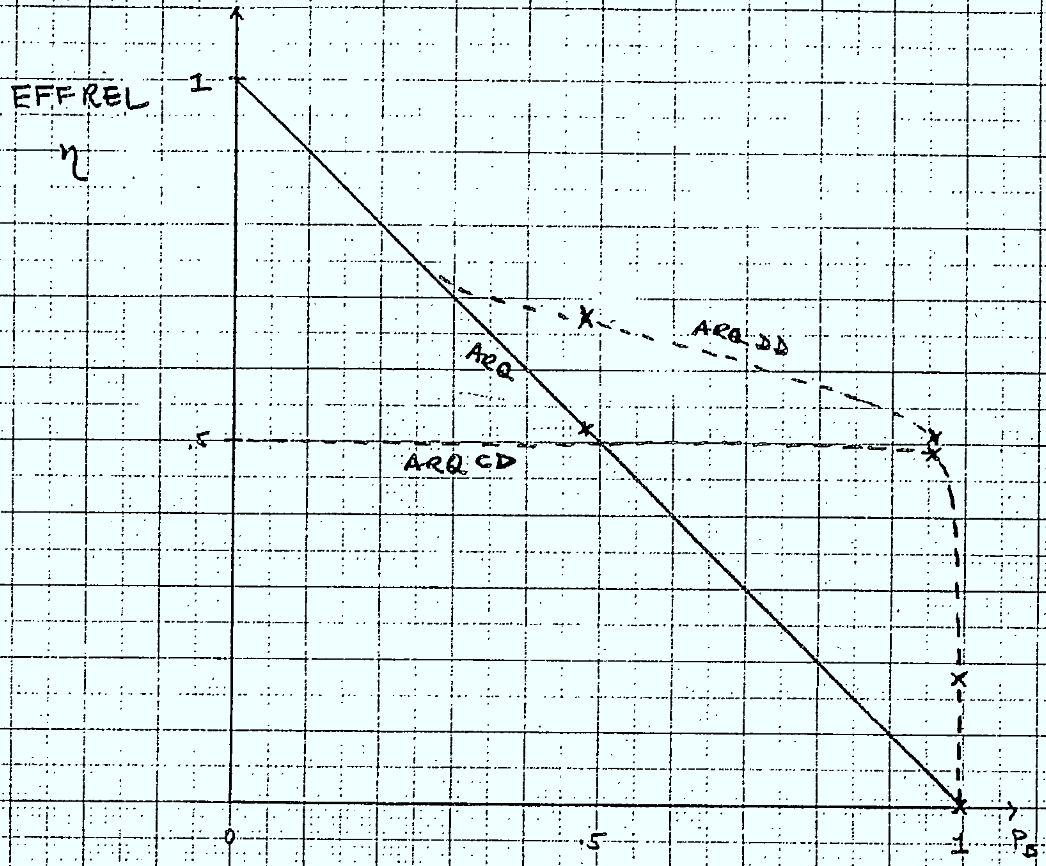


figure 9

throughput is 51%, a tremendous improvement over selective repeat ARQ. The difference of course is due to the error correcting capability of the scheme. For this range of channel conditions the block error rate is nearly 1 with error detection only, but the convolutional code is cpable of correcting most of the error patterns, so that most blocks are recieved correctly after the two transmissions required to carry out Viterbi decoding. Essentially the system has switched to code diversity operation with the diversity branch being provided by time diversity. This is equivalent in throughput to ARQCD which provides the diversity branch in simultaneous frequency diversity. Note however that ARQCD entails a smaller delay in delivering a given block to the receiver.

The third experiment has the channel continously in the bad state. The throughput efficiency of selective repeat ARQ is zero, but the efficiency of ARQDD is 19%. Note that in this case, the bounds were not tight and it was necessary to run a second simulation to estimate V_c. Because of the large number of errors in the transmissions, many Viterbi decodings are required before a block is successfully decoded at the receiver. The simulation to estimate this parameter took about 11 hours on the IBM PC. This experiment demonstrates that ARQDD continues transmitting information through the very noisy channel way after ordinary ARQ schemes have collapsed. The system thus appears to be extremely well-suited to HF radio transmission where the ability to adapt to changing channel conditions is essential.

4.3 AN ADAPTIVE TRANSMISSION PROTOCOL

The adaptivity of ARQDD to the changing channel conditions is accomplished through the automatic retransmission of erroneous subpackets. For the multi-FSK system under consideration a large number of subpackets are transmitted in each frame. Under very noisy conditions this will require the retransmission of a large proportion of each frame. This will require considerable complexity in terms of the buffer management and sequence numbering operations. Thus it is preferable if the protocol operates so that the number of retransmissions is kept low, while operating at a throughput efficiency close to that of ARQDD. As indicated in the previous section, when the throughput of ARQDD is near 50%, it is equivalent in throughput to ARQCD which has a throughput of V_c^o and which is shown in Table 1 as well as in Figure 9. It can be seen that by switching to ARQCD when the throughput of ARQDD falls to near 50%, significant simplifications in the implementation will be obtained at little loss in throughput. Since ARQCD has a throughput that never exceeds 50%, the protocol should switch to ARQDD and perhaps even ordinary ARQ when the channel conditions are favorable.

The obvious decision rule for switching between the two protocols is to compare the number of subpackets that require retransmission to a threshold. The switch from ARQDD to ARQCD is effected when the number of packets that require retransmission exceeds the threshold. The switch in the reverse direction should be based on the number of subpackets that arrive incorrectly prior to Viterbi decoding since we are trying to establish that the system can operate satisfactorily using essentially hard decisions only. The required count can be made by retaining the same subpacket format and predecoding procedures at the receiver. Each subpacket would first be divided and error checked prior to Viterbi decoding. If either of the branches received in two different frequencies is correct, then Viterbi decoding can be skipped, and as well the count required to implement the decision rule for switching can be carried out. The retention of this part of the ARQDD protocol also gives the receiver the option to handle many more subpackets per frame than it would be able to decode in ARQCD where every branch pair would undergo Viterbi decoding.

4.4 OPEN ISSUES

The discussion in this section has concentrated only on the throughput performance of the various schemes for several relatively simple models. It has been established that code diversity significantly extends the range of channel conditions within which information can be reliably transmited. As well the combination of code diversity with ARQ makes it possible to operate at high throughput efficiencies when channel conditions are favorable. We expect that these conclusions will hold for real channels as well because the schemes make no specific assumptions about the channel statistics and because the schemes are inherently adaptive to a coarse statistic, block error rate.

The schemes need to be investigated further with attention paid to implementation details. The memory and processing requirements need to be estimated. The algorithm can then be optimized for maximum performance within the constraints placed on the processing and memory requirements.

APPENDIX A

```
MODULE NAME:   MAIN LOOP (VITERBI)
MNEM:          MLOOP
HIER:          1.0
DESC:          This is the top level module of the hierarchy of
the Real-Time Viterbi Decoder.  Upon being entered, it
initializes variables and storage areas.  It then enters the main
loop of the program that repetitively implements the decoding
cycle.

CALLED MODULES:
     NAME:  INITIALIZE
     MNEM:  INIT          HIER:  1.1

     NAME:  ADD, COMPARE, SELECT
     MNEM:  ACS1, ACS2    HIER:  1.2

     NAME:  SCALE METRICS
     MNEM:  SCALE         HIER:  1.3

     NAME:  SEARCH FOR BEST STATE
     MNEM:  SEARCH        HIER:  1.4

     NAME:  HISTORY UPDATE
     MNEM:  HIST          HIER:  1.5


MODULE NAME:   INITIALIZE
MNEM:          INIT
HIER:          1.1
DESC:          Initialize input ports.  Initialize output port.
Set long term history to zeros.  Setup PN generators.  Zero block
variables.

COMMON DATA:
     BF1  (WRITE)
     ALLOK, NOBLOCK, ERRS, RESYNC,  (WRITE)
     PN1, GPN  (WRITE)
     OUT  (WRITE)
     OUTPORT, PCR, ACR, IFR, OUTDDR, DDRA, DDRB  (WRITE)
     HO  (WRITE)

CALLED FROM:
     NAME:  MAIN LOOP (VITERBI)
     MNEM:  MLOOP         HIER:  1.0


MODULE NAME:   ADD, COMPARE, SELECT
MNEM:          ACS1, ACS2
HIER:          1.2
DESC:          These two routines each update one of the state
information tables.  ACS1 uses State Organization Table #1 (SOT1)
to update the information in buffer #1 (BF1), the results being
put in buffer #2.  ACS2 uses SOT2 to update BF2 and the results
are put in BF2.  The states are updated in groups of four states
until all states are done.  At the start of the ACS operation
```

data output, data input, and metric distances are calculated.

```
LOCAL DATA:
     SPSAVE  (READ/WRITE)

COMMON DATA:
     FOR ACS1:  BF2, SOT1  (READ)
                BF2  (READ/WRITE)
     FOR ACS2:  BF2, SOT2 (READ)
                BF1  (READ/WRITE)
     T0000,T0001,...T1111  (READ)

CALLED FROM:
     NAME:  MAIN LOOP (VITERBI)
     MNEM:  MLOOP        HIER:  1.0

CALLED MODULES:
     NAME:  SHIFT OUT DATA, GET DATA
     MNEM:  SHIFT        HIER:  1.2.1


MODULE NAME:  SCALE METRICS
MNEM:         SCALE
HIER:         1.3
```

DESC: Subtracts previous best metric from all state metrics. Also, if any metric gets above 32000 then it is reduced to approximately 16000. The truncation should not be required if calls are made frequently to this routine.

CALLING PARAMETERS:
     Register U contains the address of the buffer to be scaled (BF1 OR BF2).
     Variable: BESTM should be equal to or less than the best metric.

```
LOCAL DATA:
     SCOUNT  (READ/WRITE)

COMMON DATA:
     BF1, BF2  (READ/WRITE)
     BESTM  (READ)

CALLED FROM:
     NAME:  MAIN LOOP (VITERBI)
     MNEM:  MLOOP        HIER:  1.0


MODULE NAME:  SEARCH FOR BEST STATE
MNEM:         SEARCH
HIER:         1.4
```

DESC: Loop through all states to find the lowest metric. Save the metric and path pointer for later use.

CALLING PARAMETERS:
     Register U should point to buffer to be searched.

LOCAL DATA:
    COUNT (READ/WRITE)

COMMON DATA:
    BESTM, BESTV (READ/WRITE)
    BF1, BF2 (READ)

CALLED FROM:
    NAME: MAIN LOOP (VITERBI)
    MNEM: MLOOP HIER: 1.0

MODULE NAME: HISTORY UPDATE
MNEM: HIST
HIER: 1.5
DESC: Trace back operation using best path pointer to find most likely output. Move recent history (path pointers) to long term history. Create new path pointers.

CALLING PARAMETERS:
    Register X contains a pointer to the first path pointer in whichever buffer is selected.
    BESTV should contain the best path pointer found by SEARCH.

LOCAL DATA:
    CURCOL (READ/WRITE)

COMMON DATA:
    BESTV (READ)

CALLED FROM:
    NAME: MAIN LOOP (VITERBI)
    MNEM: MLOOP HIER: 1.0

MODULE NAME: SHIFT OUT DATA, GET DATA
MNEM: SHIFT
HIER: 1.2.1
DESC: Output data. Raise strobe to indicate data valid. Get eye values. Rotate eye values into position. Calculate branch distances. Lower strobe.

LOCAL DATA:
    TEMPA, TEMPB (READ/WRITE)
    COMA,COMB (READ/WRITE)

CALLED FROM:
    NAME: ADD,COMPARE, SELECT
    MNEM: ACS1, ACS2 HIER: 1.2

CALLED MODULES:
    NAME: GET DATA FROM ADC BOARD
    MNEM: GETBIT HIER: 1.2.1.1

```
     NAME:  OUTOUT BIT
     MNEM:  PUTBIT        HIER:  1.2.1.2

     NAME:  RAISE STROBE
     MNEM:  STROBUP       HIER:  1.2.1.3

     NAME:  LOWER STROBE
     MNEM:  STROBDN       HIER:  1.2.1.4
```

MODULE NAME: GET DATA FROM ADC BOARD
MNEM: GETBIT
HIER: 1.2.1.1
DESC: Wait for data ready from ADC board. Read value from ADC board. After isolating lower seven bits run value through the quantization map.

LOCAL DATA:
MAP (READ)
Register A & B each hold one of the mapped, integrated eye values.

COMMON DATA:
IFR (READ/WRITE)
ADAT, BDAT (READ)

CALLED FROM:

```
     NAME:  SHIFT OUT DATA, GET DATA
     MNEM:  SHIFT         HIER:  1.2.1
```

MODULE NAME: OUTPUT BIT
MNEM: PUTBIT
HIER: 1.2.1.2
DESC: Shift out a bit from the variable OUT. This value is placed on the output port.

CALLING PARAMETERS:
OUT should contain the most likely output bits found in the routine HIST.

COMMON DATA:
OUT (READ/WRITE)
OUTPORT (READ/WRITE)
DOUT (WRITE)

CALLED FROM:

```
     NAME:  SHIFT OUT DATA, GET DATA
     MNEM:  SHIFT         HIER:  1.2.1
```

CALLED MODULE:

```
     NAME:  DO BLOCK ERROR CHECKING
     MNEM:  ERRCHK        HIER:  1.2.1.2.1
```

```
MODULE NAME:  RAISE STROBE
MNEM:         STROBUP
HIER:         1.2.1.3
DESC:         Raise strobe to indicate output data valid.

COMMON DATA:
     OUTPUT  (READ/WRITE)
     DOUT  (WRITE)

CALLED FROM:
     NAME:  SHIFT DATA OUT, GET DATA
     MNEM:  SHIFT         HIER:  1.2.1


MODULE NAME:  LOWER STROBE
MNEM:         STROBDN
HIER:         1.2.1.4
DESC:         Lower strobe on data output port.

COMMON DATA:
     OUTPUT  (READ/WRITE)
     DOUT  (WRITE)

CALLED FROM:
     NAME:  SHIFT DATA OUT, GET DATA
     MNEM:  SHIFT         HIER:  1.2.1


MODULE NAME:  DO BLOCK ERROR CHECKING
MNEM:         ERRCHK
HIER:         1.2.1.2.1
DESC:         Keeps a running record of the most recent output
bits so that the internal shift register can be reloaded if
synchronization is lost.  The bit to be output is compared to the
internal PN generator and any differences are counted.  When the
block is done, one of three characters is printed:

          "."  -  no errors in block
          "*"  -  less than ERRLIM errors
          "s"  -  ERRLIM or more errors  (Reload shift register)

The appropriate counter is also incremented.

CALLING PARAMETERS:
     BITOUT should contain bit just output.

LOCAL DATA:
     BCOUNT  (READ/WRITE)
     STOR1, STOR2  (READ/WRITE)

COMMON DATA:
     BITOUT  (READ)
     ERRS, RESYNC, ALLOK, NOBLOCKS  (READ/WRITE)
```

```
CALLED FROM:
     NAME:   OUTPUT BIT
     MNEM:   PUTBIT          HIER:   1.2.1.2

CALLED MODULES:
     NAME:   NEXT PSEUDO-NOISE BIT
     MNEM:   NEXTPN          HIER:   1.2.1.2.1.1

     NAME:   REPORT BLOCK ERRORS
     MNEM:   REP             HIER:   1.2.1.2.1.2

     NAME:   CHECK KEYBOARD
     MNEM:   KEYCHK          HIER:   1.2.1.2.1.2.1

     NAME:   PRINT CHARACTER
     MNEM:   PUTC            HIER:   1.2.1.2.1.2.2
```

```
MODULE NAME:   NEXT PSEUDO-NOISE BIT
MNEM:          NEXTPN
HIER:          1.2.1.2.1.1
```

DESC: Generate next pseudo-noise value from internal shift register and feed it into the shift register to carry on. The sequence generated is 511.

```
LOCAL DATA:
     CHECK  (READ/WRITE)
     PN1, PN2  (READ/WRITE)
     Register A returns the PN bit.

CALLED FROM:
     NAME:   DO BLOCK ERROR CHECKING
     MNEM:   ERRCHK          HIER:   1.2.1.2.1
```

```
MODULE NAME:   REPORT BLOCK ERRORS
MNEM:          REP
HIER:          1.2.1.2.1.2
```

DESC: Report on the block errors encountered. Report includes: total blocks read; number of error free blocks; number that had bad bits; and number that caused resynchronization.

CALLING PARAMETERS:
NOBLOCK, ALLOK, ERRS, RESYNC should contain appropriate block error values.

```
LOCAL DATA:
     TB, OK, NB, RS, MEND  (READ)

COMMON DATA:
     NOBLOCK, ALLOK, ERRS, RESYNC  (READ)

CALLED FROM:
     NAME:   DO BLOCK ERROR CHECKING
     MNEM:   ERRCHK          HIER:   1.2.1.2.1
```

```
CALLED MODULES:
     NAME:  OUTPUT STRING
     MNEM:  TEXT          HIER:  1.2.1.2.1.2.3

     NAME:  PRINT DECIMAL NUMBER
     MNEM:  PNUM          HIER:  1.2.1.2.1.2.4


MODULE NAME:  CHECK KEYBOARD
MNEM:         KEYCHK
HIER:         1.2.1.2.1.2.1
DESC:         Get  a  character  from  the  keyboard  if  one  is
waiting, else return zero (null).

LOCAL DATA:
     Returns the read character in register A.

COMMON DATA:
     STATUS, DATA  (READ)

CALLED FROM:
     NAME:  DO BLOCK ERROR CHECKING
     MNEM:  ERRCHK        HIER:  1.2.1.2.1


MODULE NAME:  PRINT CHARACTER
MNEM:         PUTC
HIER:         1.2.1.2.1.2.2
DESC:         Print a character on the terminal.

CALLING PARAMETERS:
     Register A contains ASCII value to be printed.

COMMON DATA:
     STATUS  (READ)
     DATA  (WRITE)

CALLED FROM:
     NAME:  DO BLOCK ERROR CHECKING
     MNEM:  ERRCHK        HIER:  1.2.1.2.1

     NAME:  OUTPUT STRING
     MNEM:  TEXT          HIER:  1.2.1.2.1.2.3

     NAME:  PRINT DECIMAL NUMBER
     MNEM:  PNUM          HIER:  1.2.1.2.1.2.4


MODULE NAME:  OUTPUT STRING
MNEM:         TEXT
HIER:         1.2.1.2.1.2.3
DESC:         Print a string on the terminal  until a zero (null)
is encountered.
```

CALLING PARAMETERS:
    Register X holds starting address of string.

CALLED FROM:
    NAME: REPORT BLOCK ERRORS
    MNEM: REP HIER: 1.2.1.2.1.2

CALLED MODULE:
    NAME: PRINT CHARACTER
    MNEM: PUTC HIER: 1.2.1.2.1.2.2

MODULE NAME: PRINT DECIMAL NUMBER
MNEM: PNUM
HIER: 1.2.1.2.1.2.4
DESC: The D register is printed on the terminal as a decimal number with zero blanking. Unsigned.

CALLING PARAMETERS:
    Register D contains value to be printed.

LOCAL DATA:
    K10000, K1000, K100, K10, K1 (READ)
    ZBLANK, DCOUNT, TEMP (READ/WRITE)

CALLED FROM:
    NAME: REPORT BLOCK ERRORS
    MNEM: REP HIER: 1.2.1.2.1.2

CALLED MODULE:
    NAME: PRINT CHARACTER
    MNEM: PUTC HIER: 1.2.1.2.1.2.2

APPENDIX B

```
*****************************************************************
*
*       TEST PROGRAM TO VERIFY THE OPERATION OF THE
*       NEW VITERBI DECODER ROUTINES.
*       HAS REAL I/O ROUTINES TO INTERFACE TO THE ADC BOARD
*
*       BLOCK ERROR REPORTING INSERTED.
*
*                       MARCH 8, 1984.
*
****************************************************************

BLOCKLEN        EQU     64
ERRLIM          EQU     10
BLIMIT          EQU     1728

BF1     EQU     MOB1
BF2     EQU     MOB2
        SETD    1
        ORG     $200     ; START PROGRAM AT $200

START   LDS     #$100    ; LOAD STACK POINTER TO USE REGION $000->$100
        LDA     #$01     ; MOVE DIRECT PAGE TO $0100 (AWAY FROM STACK)
        TFR     A,DP
        LBSR    INIT     ; INITIALIZE VARIABLES AND STORAGE AREAS

MLOOP   BSR     ACS1     ; ADD COMPARE SELECT CYCLE
        LDU     #BF2
        LBSR    SCALE
        BSR     ACS2     ; ADD COMPARE SELECT CYCLE
        BSR     ACS1     ; ADD COMPARE SELECT CYCLE
        BSR     ACS2     ; ADD COMPARE SELECT CYCLE
        BSR     ACS1     ; ADD COMPARE SELECT CYCLE
        BSR     ACS2     ; ADD COMPARE SELECT CYCLE
        LDU     #BF1
        BSR     SEARCH   ; LOCATE BEST METRIC AND POINTER
        LDU     #BF1+4
        LBSR    HIST     ; TRACE BACK AND UPDATE POINTERS
        BRA     MLOOP
```

```
*****************************************************************
*
*       INITIALIZE THE DATA AREAS AND I/O PORTS.
*       THIS ROUTINE INSURES THAT THE PROGRAM CAN BE RESTARTED
*       FROM ANY POINT AND THE SYSTEM WILL FUNCTION PROPERLY.
*       NOTE: ASSUMES THAT INTERRUPTS ARE ALREADY DISABLED.
*
****************************************************************

INIT    CLR     OUTPORT ; CLEAR OUTPUT PORT IMAGE
        LDA     #$00    ; LOAD PERIPHERAL CONTROL REGISTER CODE
        STA     PCR     ; INITIALIZE THE I/O PORTS
        CLR     ACR
        LDA     #$FF    ; CLEAR ALL INTERRUPT FLAGS
        STA     IFR
        STA     OUTDDR  ; SET DECODED OUTPUT PORT TO ALL OUTPUTS

        LDA     #0      ; A ZERO ON EACH BIT INDICATES ALL INPUTS
        STA     DDRA    ; SET A/D INPUT PORTS TO ALL INPUTS (8 BITS)
        STA     DDRB

        LDX     #HO     ; INITIALIZE HISTORY TO ZEROS
        LDD     #512    ; SET COUNTER TO NUMBER OF BYTES IN
*                       ; LONG TERM HISTORY (ASSUMES #STATES=64,
*                       ; LONG TERM HISTORY DEPTH=8)

LOOP1   CLR     ,X+

        SUBD    #1      ; COUNT=COUNT-1
        BNE     LOOP1   ; REPEAT TILL COUNT=0

        LDX     #BF1    ; INITIALIZE FIRST BUFFER OF STATE INFO
        LDD     #1      ; SET A=0 AND B=1
LOOP2   CLR     ,X+     ; CLEAR BOTH METRICS (4 BYTES)
        CLR     ,X+
        CLR     ,X+
        CLR     ,X+
        STD     ,X++    ; SET PATH POINTERS (SEQUENTIAL)
        ADDD    #$202   ; ADD 2 TO A AND B
        CMPA    #64     ; CHECK IF A= LAST STATE
        BNE     LOOP2   ; REPEAT TILL DONE

        CLR     OUT     ; CLEAR CURRENT OUTPUT SHIFT REGISTER
        LDA     #$FF    ; PUT A SOMETHING IN THE PN GENERATOR
        STA     PN1     ; TO MAKE SURE THAT IT STARTS OK
        STA     SFLAG   ; SET START FLAG,(RESET ON 1ST OK BLOCK)

        LDD     #0      ; CLEAR BLOCK COUNT STUFF QUICK
        STD     ALLOK   ; NUMBER OF CORRECT BLOCKS
        STD     NOBLOCK ; TOTAL NUMBER OF BLOCKS
        STD     ERRS    ; NUMBER OF BLOCKS WITH ERRORS<ERRLIM
        STD     RESYNC  ; NUMBER OF BLOCKS WITH ERRORS>ERRLIM
        RTS
```

```
*********************************************************************
*
*       THIS IS THE STANDARD DECODING ROUTINE.  DOES ALL REQUIRED
*       STATE INFORMATION UPDATING FOR A SINGLE BIT.
*
*       ACS1    USES BUFFER 1 (BF1) AS SOURCE AND BF2 AS DESTINATION
*       ACS2    USES BUFFER 2 (BF2) AS SOURCE AND BF1 AS DESTINATION
*
*       ON ENTRY: NO PARAMETERS REQUIRED
*       ON EXIT:  CONTENTS OF DESTINATION BUFFER MODIFIED
*                 NO SPECIAL INFO IS RETURNED
*
*********************************************************************

DONE    LDS     SPSAVE  ; RESTORE STACK POINTER
        RTS             ; AND RETURN TO MAIN LINE

ACS1    LBSR    SHIFT   ; OUTPUT A BIT AND WAIT FOR NEW INPUT
        STS     SPSAVE  ; PRESERVE STACK POINTER
        LDS     #SOT1-12 ; SETUP POINTER TO INSTRUCTIONS
        LDU     #BF1-12 ; SETUP STATE INFO POINTER (SOURCE)
        BRA     EVEACS  ; JUMP INTO THE ACS ROUTINE

ACS2    LBSR    SHIFT   ; OUTPUT A BIT AND WAIT FOR NEW INPUT
        STS     SPSAVE  ; PRESERVE STACK POINTER
        LDS     #SOT2-12 ; SETUP POINTER TO INSTRUCTIONS
        LDU     #BF2-12 ; SETUP STATE INFO POINTER (SOURCE)

EVEACS  LEAU    12,U    ; UPDATE STATE INFORMATION
        LEAS    12,S    ; AND STATE ORGANIZATION POINTERS
        LDY     ,S      ; LOAD POINTER TO DESTINATION
        BEQ     DONE    ; IF DESTINATION POINTER=0 THEN DONE
        LDX     ,U      ; GET METRIC A
        LDD     [2,S]   ; GET BRANCH DISTANCES TO NEW STATE A
        LEAX    A,X     ; ADD METRIC A TO BRANCH DISTANCE
        STX     ,Y      ; TENATIVE WINNER: SAVE AT DESTINATION

        LDX     2,U     ; GET METRIC B
        ABX             ; ADD BRANCH DISTANCE TO METRIC B
        CMPX    ,Y      ; COMPARE TO TENATIVE WINNER
        LDD     4,U     ; GET PATH POINTERS FOR FUTURE USE
        BCC     ATOAE   ; FROM COMPARE ADJUST NEW STATE INFO ACCORDINGLY

*                       ; METRIC B BETTER: ALL STATE INFO MUST BE UPDATED
        STX     ,Y      ; SAVE METRIC
        STB     4,Y     ; SAVE PATH POINTER
        BRA     NXTEACS ; GOTO NEXT ACS OPERATION

*                       ; METRIC A BETTER: JUST UPDATE PATH SINCE METRIC DONE
ATOAE   STA     4,Y     ; SAVE PATH POINTER

NXTEACS LEAY    96,Y    ; CALC NEW DESTINATION POINTER ((#STATES/2)*3)
        LDX     ,U      ; GET METRIC A
        LDD     [4,S]   ; GET BRANCH DISTANCES TO NEW STATE B
        LEAX    A,X     ; ADD METRIC A TO BRANCH DISTANCE
        STX     ,Y      ; TENATIVE WINNER: SAVE AT DESTINATION
```

```
          LDX      2,U       ; GET METRIC B
          ABX                ; ADD BRANCH DISTANCE TO METRIC B
          CMPX     ,Y        ; COMPARE TO TENATIVE WINNER
          LDD      4,U       ; GET PATH POINTERS FOR FUTURE USE
          BCC      ATOBE     ; FROM COMPARE ADJUST NEW STATE INFO ACCORDINGLY

*                            ; METRIC B BETTER: ALL STATE INFO MUST BE UPDATED
          STX      ,Y        ; SAVE METRIC
          STB      4,Y       ; SAVE PATH POINTER
          BRA      ODDACS    ; GOTO NEXT ACS OPERATION

*                            ; METRIC A BETTER: JUST PATH SINCE METRIC DONE
ATOBE     STA      4,Y       ; SAVE PATH POINTER


ODDACS    LDY      6,S       ; LOAD POINTER TO DESTINATION
          LDX      6,U       ; GET METRIC A
          LDD      [8,S]     ; GET BRANCH DISTANCES TO NEW STATE A
          LEAX     A,X       ; ADD BRANCH DISTANCE TO METRIC A
          STX      ,Y        ; TENATIVE WINNER: SAVE AT DESTINATION

          LDX      8,U       ; GET METRIC B
          ABX                ; ADD BRANCH DISTANCE TO METRIC B
          CMPX     ,Y        ; COMPARE TO TENATIVE WINNER
          LDD      10,U      ; GET PATH POINTERS FOR FUTURE USE
          BCC      ATOAO     ; FROM COMPARE ADJUST STATE INFO ACCORDINGLY

*                            ; METRIC B BETTER: ALL STATE INFO MUST BE UPDATED
          STX      ,Y        ; SAVE METRIC
          STB      3,Y       ; SAVE PATH POINTER
          BRA      NXTOACS   ; GOTO NEXT ACS OPERATION

*                            ; METRIC A BETTER: JUST UPDATE PATH SINCE METRIC DONE
ATOAO     STA      3,Y       ; SAVE PATH POINTER


NXTOACS   LEAY     96,Y      ; CALC NEW DESTINATION POINTER ((#STATES/2)*3)
          LDX      6,U       ; GET METRIC A
          LDD      [10,S]    ; GET BRANCH DISTANCE TO NEW STATE B
          LEAX     A,X       ; ADD METRIC A TO BREACH DISTANCE
          STX      ,Y        ; TENATIVE WINNER: SAVE AT DESTINATION

          LDX      8,U       ; GET METRIC B
          ABX                ; ADD BRANCH DISTANCE TO METRIC B
          CMPX     ,Y        ; COMPARE TO TENATIVE WINNER
          LDD      10,U      ; GET PATH POINTERS FOR FUTURE
          BCC      ATOBO     ; FROM COMPARE ADJUST STATE INFO ACCORDINGLY

*                            ; METRIC B BETTER: ALL STATE INFO MUST BE UPDATED
          STX      ,Y        ; SAVE METRIC
          STB      3,Y       ; SAVE PATH POINTER
          LBRA     EVEACS    ; GOTO NEXT ACS OPERATION

*                            ; METRIC A BETTER: JUST UPDATE PATH SINCE PATH DONE
ATOBO     STA      3,Y       ; SAVE PATH POINTER
          LBRA     EVEACS
```

```
*********************************************************************
*
*       INSURE THAT METRICS NEVER OVERFLOW. MUST BE CALLED BEFORE
*       BEST METRICS REACH 16000.
*
*       SUBTRACT BESTM FROM ALL METRICS.  HALVE ANY METRIC GREATER
*       THAN 32K TO PREVENT OVERFLOW.
*
*       ON ENTRY: REG. U POINTS TO INFO BUFFER TO BE SCALED
*       ON EXIT:  THE INDICATED BUFFER HAS BEEN MODIFIED
*                 NO SPECIAL INFO IS RETURNED
*
*********************************************************************

SCALE   LDA     #32     ; SET COUNT= NUMBER OF STATE PAIRS (#STATES/2)
        STA     SCOUNT

SCLOOP  PULU    D        ; GET FIRST METRIC OF PAIR
        SUBD    BESTM   ; SCALE METRIC
        BPL     NOTRC1  ; CHECK IF CLOSE TO OVERFLOW
        LSRA            ; CHOP IF CLOSE TO OVERFLOW
NOTRC1  STD     -2,U    ; SAVE UPDATED METRIC BACK

        PULU    D,Y     ; GET METRIC (Y = PATH POINTERS   UNUSED)
        SUBD    BESTM   ; SCALE METRIC
        BPL     NOTRC2  ; CHECK IF CLOSE TO OVERFLOW
        LSRA            ; CHOP IF CLOSE TO OVERFLOW
NOTRC2  STD     -4,U    ; SAVE UPDATED METRIC BACK

        DEC     SCOUNT  ; COUNT=COUNT-1
        BNE     SCLOOP  ; REPEAT
        RTS             ; RETURN TO MAIN LINE
```

```
*********************************************************************
*
*       LOCATE STATE WITH LOWEST METRIC.  THE METRIC AND PATH
*       POINTER FOR THIS STATE ARE SAVED IN BESTM AND BESTV.
*
*       ON ENTRY: REG. U POINTS TO THE INFO BUFFER TO BE SEARCHED
*       ON EXIT : LOCATIONS BESTM, BESTV CONTAIN BEST METRIC AND
*                 PATH POINTER RESPECTIVELY.
*                 NOTE: CONTENTS OF ALL REGISTERS LOST
*
*********************************************************************

SEARCH  LDX     #$0FFFF ; SET BEST METRIC = WORST POSSIBLE
        STX     BESTM
        LDA     #32     ; SET COUNT= NUMBER OF STATE PAIRS (#STATES/2)
        STA     COUNT
        PULU    X,Y     ; GET FIRST TWO METRICS FROM STATE INFO BUFFER
        BRA     SMID    ; GOTO COMPARE

SLOOP   PULU    D,X,Y   ; GET TWO METRICS (D = PREVIOUS PATH POINTERS, UNUSED)
SMID    CMPX    -2,U    ; COMPARE JUST FETCHED METRICS TO EACH OTHER
        BCC     YBEST   ; SELECT WHICHEVER IS BETTER

XBEST   CMPX    BESTM   ; COMPARE FIRST METRIC TO BEST METRIC
        BCC     LCHK    ; IF NO NEW BEST METRIC THEN SKIP TO END
*                       ; ELSE NEW BEST METRIC: UPDATE BEST METRIC
        LDA     ,U      ; GET PATH POINTER CORRESPONDING TO NEW BEST METRIC
        STA     BESTV   ; SAVE IT AWAY
        STX     BESTM   ; UPDATE BEST METRIC
        BRA     LCHK    ; SKIP TO END


YBEST   CMPY    BESTM   ; COMPARE SECOND METRIC TO BEST METRIC
        BCC     LCHK    ; IF NO NEW BEST METRIC THEN SKIP TO END
*                       ; ELSE NEW BEST METRIC: UPDATE BEST METRIC
        LDA     1,U     ; GET PATH POINTER CORRESPONDING TO NEW BEST METRIC
        STA     BESTV   ; SAVE IT AWAY
        STY     BESTM   ; UPDATE BEST METRIC

LCHK    DEC     COUNT   ; COUNT=COUNT-1
        BNE     SLOOP   ; AND REPEAT
        RTS
```

```
*****************************************************************
*
*       THE HISTORY UPDATE SECTION
*       THE PATH POINTER FOUND BY SEARCH IS USED TO LOOKUP THE
*       BEST OUTPUT FROM THE LONG TERM HISTORY.  THIS SETION ALSO
*       UPDATES THE LONG TERM HISTORY BY MOVING THE POINTERS STORED
*       IN THE STATE INFORMATION INTO THE LONG TERM HISTORY AND
*       CREATING NEW POINTERS TO REFERENCE THE HISTORY.
*
*       ON ENTRY: BESTV IS THE INITIAL POINTER TO LONG TERM HISTORY
*       ON EXIT:  OUT HOLDS THE BITS (ONE CONSTRAINT LENGTH) THAT
*                 ARE THE MOST LIKELY OUTPUT
*
*****************************************************************

HIST    LDY     CURCOL  ; GET CURRENT STARTING COLUMN OF HISTORY (CURCOL)
        LDA     BESTV   ; GET INITIAL PATH POINTER (CORRISPONDS TO BESTM)
        LDX     ,Y      ; GET ADDRESS OF FIRST COLUMN OF HISTORY
        LDA     A,X     ; LOOKUP NEXT PATH POINTER IN THAT COLUMN
        LDX     -2,Y    ; GET ADDRESS OF PREVIOUS COLUMN OF HISTORY
        LDA     A,X     ; LOOKUP NEXT POINTER IN THAT COLUMN
        LDX     -4,Y
        LDA     A,X     ; THIS PROCESS CONTINUES FOR HOWEVER MANY
        LDX     -6,Y    ; COLUMNS OF HISTORY ARE BEING USED (8)
        LDA     A,X
        LDX     -8,Y    ; FOR A DEPTH OF 8 LEVELS THE LAST COLUMN IS
        LDA     A,X     ; REFERENCED BY -14,Y  IF ONLY 6 LEVELS WERE
        LDX     -10,Y   ; BEING USED THE LAST REFERENCE WOULD BE -10,Y
        LDA     A,X
        LDX     -12,Y
        LDA     A,X
        LDX     -14,Y
        LDA     A,X     ; FINAL POINTER IS MOST LIKELY OUTPUT
        STA     OUT     ; FOR TESTING SAVE IN THE OUTPUT LOCATION

        LEAY    2,Y     ; MOVE CURRENT COLUMN POINTER TO NEXT COLUMN
        CMPY    #COLLIM ; IF END OF LIST REACHED THEN
        BNE     NXCOL

        LDY     #COLBEG ; START AT BEGINNING AGAIN
NXCOL   STY     CURCOL  ; SAVE BACK AS CURRENT COLUMN


*                       ; MOVE PATH POINTERS INTO LONG TERM HISTORY
        LDY     ,Y      ; GET POINTER TO START OF NEW CURRENT COLUMN
        LDD     #1      ; SET D=1

HLOOP   LDX     ,U      ; GET PAIR OF PATH POINTERS
        STD     ,U      ; PUT IN NEW PAIR OF PATH POINTERS
        STX     ,Y++    ; SAVE OLD PAIR IN HISTORY
        LEAU    6,U     ; ADVANCE BUFFER POINTER
        ADDD    #$202   ; INCREMENT NEW PAIR OF PATH POINTERS
        CMPA    #64     ; CHECK FOR DONE
        BNE     HLOOP   ; REPEAT
        RTS             ; RETURN TO MAIN LINE
```

```
*****************************************************************
*
*       THE SHIFT ROUTINE COORDINATES ALL THE PARALLEL I/O
*       ROUTINES.
*       THE BRANCH DISTANCES ARE ALSO CALCULATED AS SOON AS
*       THE INTEGRATED EYE VALUES ARE READ IN.
*
*****************************************************************

*       ; SELECT APPROPRIATE MASK BY COMMENTING (*)
*       ; ALL UN-NEEDED MASKS

*MASK   EQU     %1111111  ; MASK FOR 7 BIT QUANTIZATION
MASK    EQU     %0111111  ; MASK FOR 6 BIT QUANTIZATION
*MASK   EQU     %0011111  ; MASK FOR 5 BIT QUANTIZATION
*MASK   EQU     %0001111  ; MASK FOR 4 BIT QUANTIZATION
*MASK   EQU     %0000111  ; MASK FOR 3 BIT QUANTIZATION
*MASK   EQU     %0000011  ; MASK FOR 2 BIT QUANTIZATION
*MASK   EQU     %0000001  ; MASK FOR HARD DECODING

*                         ; REMEMBER TO ADJUST THE ASTRISKS BELOW
*                         ; TO AGREE WITH ABOVE SELECTED MASK


SHIFT   LBSR    PUTBIT   ; OUTPUT DATA
        LBSR    STROBUP  ; ACTIVE STROBE TO INDICATE OUTPUT DATA VALID
        BSR     GETBIT   ; GET DATA AND ROTATE INTO POSITION

        LSRA             ; 7 BIT SOFT DECODING IF LAST * IS HERE
*       LSRA             ; 6 BIT SOFT DECODING IF LAST * IS HERE
*       LSRA             ; 5 BIT SOFT DECODING IF LAST * IS HERE
*       LSRA             ; 4 BIT SOFT DECODING IF LAST * IS HERE
*       LSRA             ; 3 BIT SOFT DECODING IF LAST * IS HERE
*       LSRA             ; 2 BIT SOFT DECODING IF LAST * IS HERE

*                          NO SOFT DECODING USED (HARD DECISION)
*                          IF NO ASTRICKS APPEAR OPPOSITE THE LSRA

        LSRB
*       LSRB
*       LSRB
*       LSRB
*       LSRB
*       LSRB

*                          THE ASTRISK(S) FOR THE ABOVE 6 LINES SHOULD
*                          FOLLOW THE SAME PATTERN AS FOR THE LSRA

        STA     TEMPA    ; SAVE EYE0 VALUE FOR LATER CALCULATIONS
        STB     TEMPB    ; SAVE EYE1 VALUE FOR LATER CALCULATIONS
        EORA    #MASK    ; CALCULATE NEGATIVE OF EYE1 VALUE
        STA     COMA     ; SAVE IT FOR LATER CALCULATION
        ADDA    TEMPB    ; CALCULATE THE BRANCH DISTANCE FOR THE LABEL 10
        STA     T1000
        STA     T1001
        STA     T1010
        STA     T1011
        STA     T0010+1
        STA     T0110+1
        STA     T1010+1
        STA     T1110+1

        EORB    #MASK    ; CALCULATE NEGATIVE OF EYEO VALUE
        STB     COMB     ; SAVE IT FOR CALCULATIONS
        ADDB    TEMPA    ; CALCULATE THE BRANCH DISTANCE FOR THE LABEL 01
        STB     T0100
        STB     T0101
        STB     T0110
        STB     T0111
        STB     T0001+1
        STB     T0101+1
        STB     T1001+1
        STB     T1101+1

        LDA     TEMPA
        ADDA    TEMPB    ; CALCULATE THE BRANCH DISTANCE FOR THE LABEL 00
        STA     T0000
        STA     T0001
        STA     T0010
        STA     T0011
        STA     T0000+1
        STA     T0100+1
        STA     T1000+1
        STA     T1100+1

        LDA     COMA
        ADDA    COMB     ; CALCULATE THE BRANCH DISTANCE FOR THE LABEL 11
        STA     T1100
        STA     T1101
        STA     T1110
        STA     T1111
        STA     T0011+1
        STA     T0111+1
        STA     T1011+1
        STA     T1111+1

        LBSR    STROBDN ; LOWER STROBE WHILE DATA STILL VALID
*       LBSR    KEYCHK
*       BEQ     TEMPO    ; OPTIONAL: IF KEY PUSHED STOP
*       RTN
TEMPO   RTS
```

```
*****************************************************************
*
*       WAIT FOR ADC BOARD TO INDICATE THAT STABLE EYE DATA
*       IS AVAILABLE ON THE PARALLEL PORTS.  THIS ROUTINE USES
*       THE INTERNAL (PIA) EDGE TRIGGERED FLAG TO DETECT
*       IF DATA IS WAITING.  LESS RELIABLE LEVEL TRIGGERING
*       COULD BE USED AS BIT 7 OF ADAT IS CONNECTED TO THE ADC
*       BOARD READY SIGNAL.
*
*       NOTE: THERE ARE TWO GETBIT ROUTINES, ONE FOR REAL DATA
*               AND ONE FOR TESTING SOFTWARE, GETBIT (TEST).
*               ONE OF THE ROUTINES SHOULD BE COMENTED OUT AT
*               ALL TIMES.
*
*       ON ENTRY: NO PARAMETERS
*       ON EXIT:  A,B HOLD MAPPED VALUES OF THE EYE SIGNALS
*                 X LOST
*
*****************************************************************

GETBIT  LDA     IFR     ; CHECK IF BOTH PORTS HAVE DATA COMING IN
        CMPA    #$12
        BNE     GETBIT  ; IF NOT THEN WAIT UNTIL DATA COMING IN
        STA     IFR     ; CLEAR FLAGS
        LDB     ADAT    ; READ ONE OF THE INTEGRATED EYE VALUES
        LDA     BDAT    ; READ THE OTHER INTEGRATED EYE VALUE
        COMA            ; COMPLEMENT TO MAKE COMPATIBLE WITH SOFTWARE
        COMB
        ANDA    #$7F    ; GET RID OF BIT 7 SINCE IT IS UNDEFINED
        ANDB    #$7F    ; GET RID OF BIT 7 SINCE IT IS UNDEFINED
        LDX     #MAP    ; GET POINTER TO START OF MAPPING AREA
        LDA     A,X     ; RUN BOTH EYE VALUES THROUGH THE MAPPING
        LDB     B,X     ; REGION OF MEMORY
        RTS             ; RETURN
```

```
******************************************************************
*
*       EXTRACTS AN OUTPUT BIT FROM THE SHIFT REGISTER (OUT)
*       WHICH, SETUP BY HIST, CONTAINS THE MOST LIKELY OUTPUT
*       SEQUENCE.  THIS DATA IS SENT TO THE OUTPUT PORT.
*
******************************************************************

PUTBIT  LDA     #$18     ; CONVERT TO ASCII SO THAT THE VALUE MAY
        LSR     OUT      ; EASILY BE PRINTED ON THE TERMINAL IF NEEDED
        ROLA

*       LBSR    PUTC     ; OPTIONAL: OUTPUT BIT MAY BE PRINTED

        ANDA    #$01     ; CONVERT BACK TO BINARY
        LDB     OUTPORT  ; GET PREVIOUS VALUE
        STA     BITOUT   ; STORE CURRENT AND SET STATUS BITS
        BNE     SETING   ; CHECK IF OUTPUT IS 1 OR 0
        ANDB    #$FE     ; ZERO IS OUTPUT: CLEAR LOW BIT
        BRA     PUTDONE
SETING  ORB     #1       ; ONE IS OUTPUT: SET LOW BIT
PUTDONE STB     OUTPORT  ; SAVE PORT VALUE
        STB     DOUT     ; ACTUALLY OUTPUT VALUE
        LBSR    ERRCHK   ; DO ERROR CHECKING
        RTS


STROBUP LDA     OUTPORT  ; RAISE STROBE, INDICATING DATA OUTPUT VALID
        ORA     #$2
        STA     DOUT
        STA     OUTPORT
        RTS


STROBDN LDA     OUTPORT  ; LOWER STROBE, COMPLETING THE CLOCK PULSE
        ANDA    #$FD
        STA     DOUT
        STA     OUTPORT
        RTS
```

```
*****************************************************************
*
*       THE IS THE BLOCK ERROR CHECKING SECTION, WITH ITS OWN
*       PN SEQUENCE GENERATOR.
*
*       ON ENTRY: USES VARIABLE: BITOUT TO SAMPLE OUTPUT
*       ON EXIT:  OUTPUT IS ON TERMINAL
*                 NO SPECIAL INFO RETURNED
*
*****************************************************************

ERRCHK  LDA     BITOUT  ; ROTATE DECODED BIT INTO STORAGE IN CASE
        RORA            ; PN SHIFT REGISTER NEEDS RELOADING LATER
        ROL     STOR1
        ROL     STOR2

        LDD     BCOUNT  ; ADD ONE TO BLOCK BIT COUNT
        ADDD    #1
        STD     BCOUNT

NO      LBSR    NEXTPN  ; PREDICT WHAT THIS BIT SHOULD BE
        EORA    BITOUT  ; COMPARE ACTUAL AND PREDICTED BITS
        BEQ     E1      ; JUMP TO NORMAL SECTION IF THEY ARE THE SAME
        LDD     ERRORS  ; ADD ONE TO THE ERROR COUNT OTHERWISE
        ADDD    #1
        STD     ERRORS

E1      LDD     BCOUNT  ; CHECK FOR END OF BLOCK
        CMPD    #BLOCKLEN ; USING BLOCKLEN
        LBNE    FINI    ; JUMP TO END OF ROUTINE IF NOT END OF BLOCK

        LDD     ERRORS  ; DO END OF BLOCK REPORT
        BEQ     E2      ; GOTO NO ERRORS SECTION IF NO ERRORS
        CMPD    #ERRLIM ; CHECK IF RESYNCING NEEDED
        BCC     E7      ; GOTO RESYNCING SECTION
        LDD     ERRS    ; ADD ONE TO BAD BLOCK COUNT
        ADDD    #1
        STD     ERRS
        LDD     ERRORS  ; GET THE NUMBER OF BIT ERRORS
        TFR     B,A     ; (IGNORE MSB A, =0 SINCE BLOCKLEN=16)
        ADDA    #'0     ; PRINT OUT # OF ERRORS
        CMPA    #'9     ; USE ALPHABIT IF DIGITS 0-9 INSUFFICIENT
        BLS     E3      ; 0-9 IS SUFFICIENT
        ADDA    #7      ; ADD OFFSET INTO A-Z SYMBOLS
        BRA     E3      ; PRINT STUFF

E7      LDA     STOR2   ; GET RID OF DON'T CARE BITS
        ANDA    #1      ;   SAVING ONLY THE LSB
        STA     STOR2
        LDD     STOR1   ; USE STORAGE TO RESYNC
        BEQ     E8      ; DON'T FILL PN511 REGISTER WITH 0-STATE!
        STD     PN1
E8      LDD     RESYNC  ; ADD ONE TO RE SYNC COUNT
        ADDD    #1
        STD     RESYNC
        LDA     #'S     ; LOAD CHARACTER INDICATING RESYNC
        BRA     E3      ; PRINT STUFF
```

```
E2      LDD     ALLOK    ; THE BLOCK IS PERFECT: ADD ONE TO PERFECT COUNT
        ADDD    #1
        STD     ALLOK

        LDA     SFLAG    ; GET RE-SYNC FLAG
        BEQ     E5       ; JUST FINISHED INITIAL RE-SYNC?
        CLR     SFLAG    ;   YES: INITIALIZE STATISTICS
        LDD     #0
        STD     RESYNC
        STD     NOBLOCKS
        STD     ERRS
        LDX     #MEND    ; OUTPUT CR-LF TO ISOLATE 1ST RE-SYNC
        LBSR    TEXT

E5      LDA     #'.      ; PRINT A DOT TO INDICATE PERFECT BLOCK
E3      LBSR    PUTC
        CLR     ERRORS   ; CLEAR BLOCK COUNTERS
        CLR     ERRORS+1 ; ERROR COUNT
        CLR     BCOUNT
        CLR     BCOUNT+1 ; BLOCK BIT COUNTER
        LDD     NOBLOCKS ; ADD ONE TO NUMBER OF BLOCKS PROCESSED
        ADDD    #1
        STD     NOBLOCKS
        CMPD    #BLIMIT
        BEQ     E6       ; IF DONE ALL BLOCKS, STOP AND REPORT

        ANDB    #%00111111 ; HAVE PROCESSED MOD 64 BLOCKS?
        BNE     E4       ; NOPE
        LDX     #MEND    ; YEP, OUTPUT A CRLF
        LBSR    TEXT     ;

E4      LBSR    KEYCHK   ; CHECK FOR KEYBOARD COMMAND
        BEQ     FINI     ; JUMP TO RETURN IF NO KEY PUSHED
E6      LBSR    REP      ; DO REPORT
        RTN              ; STOP AND RETURN TO MONITOR
FINI    RTS              ; RETURN
```

```
****************************************************************
*
*       THIS IS A PN SEQUENCE GENERATOR USED BY THE BLOCK ERROR
*       CHECKING ROUTINES.  SETUP FOR PN SEQUENCE 511
*
*       ON ENTRY: NO ENRTY PARAMETERS
*       ON EXIT:  REG. A HOLDS A SINGLE BIT EXPECTED OUTPUT
*
****************************************************************
NEXTPN  CLRA                ; CLEAR BIT PARITY COUNT
        LDB     PN1         ; LOAD B WITH SHIFT REGISTER
        BITB    #$10        ; CHECK TAP AT BIT 5
        BEQ     N1
        INCA                ; ADD ONE IF BIT SET
N1      LDB     PN2         ; LOAD B WITH REST OF SHIFT REGISTER
        BITB    #$1         ; CHECK TAP AT BIT 9 OF SHIFT REGISTER
        BEQ     N2
        INCA                ; ADD ONE IF BIT SET
N2      ANDA    #1          ; ISOLATE LOWEST BIT OF TOTAL
        STA     CHECK       ; SAVE RESULT FOR LATER
        RORA                ; ROTATE RESULT INTO THE CARRY
        ROL     PN1         ; ROTATE CARRY INTO PN SHIFT REGISTER
        ROL     PN2

N3      LDA     CHECK       ; RELOAD RESULT INTO A REGISTER

        RTS
```

```
******************************************************************
*
*       THE REPORT FUNCTION PRINTS THE BLOCK ERROR VARIABLES ON
*       THE SCREEN WITH TEXT EXPLAINATIONS AND DECIMAL OUTPUT
*
*       ON ENTRY: NO PARAMETERS EXCEPT ERROR COUNTS IN MEMORY
*       ON EXIT:   A,B,X LOST
*
*******************************************************************

REP     LDX     #TB       ; PRINT NUMBER OF BLOCKS MESSAGE
        LBSR    TEXT
        LDD     NOBLOCK ; PRINT NUMBER OF BLOCKS READ IN DECIMAL
        LBSR    PNUM
        LDX     #OK       ; PRINT NUMBER OF CORRECT BLOCKS MESSAGE
        LBSR    TEXT
        LDD     ALLOK     ; PRINT NUMBER OF BLOCKS THAT WERE CORRECT
        LBSR    PNUM
        LDX     #NB       ; PRINT BAD BLOCKS MESSAGE (ERRORS>ERRLIM)
        LBSR    TEXT
        LDD     ERRS      ; PRINT NUMBER OF BLOCKS WITH A FEW ERRORS
        LBSR    PNUM
        LDX     #RS       ; PRINT NUMBER OF RESYNCS MESSAGE
        LBSR    TEXT
        LDD     RESYNC    ; PRINT NUMBER OF BLOCKS CAUSING RESYNCING
        LBSR    PNUM
        LDX     #MEND     ; ADVANCE TO NEXT DISPLAY LINE
        LBSR    TEXT
        RTS


*                         ; TEXT USED IN THE REPORT FUNCTION

TB      FCB     13,10,13,10
        FCC     "TOTAL BLOCKS READ "
        FCB     0
OK      FCB     13,10
        FCC     "NUMBER OF ERROR FREE BLOCKS "
        FCB     0
NB      FCB     13,10
        FCC     "NUMBER THAT HAD BAD BITS "
        FCB     0
RS      FCB     13,10
        FCC     "NUMBER THAT CAUSED RE SYNC "
        FCB     0
MEND    FCB     13,10,0
```

```
**********************************************************************
*
*       THIS SECTION CONTAINS THE I/O ROUTINES USED TO COMMUNICATE
*       WITH THE THE TERMINAL.
*
*       ROUTINES INCLUDE:
*
*       KEYCHK  - FETCH CHAR. FROM KEYBOARD OR NULL IF NONE
*                 RETURN VALUE IN REG A
*
*       GETC    - WAIT FOR KEYBOARD CHAR.  RETURN VLAUE IN REG A
*                 (CURRENTLY UNUSED BUT IS AVAILABLE)
*
*       PUTC    - PRINT CHAR IS REG A ON TERMINAL.  REG B LOST
*
*       TEXT    - PRINT NULL TERMINATED STRING POINTED TO BY
*                 REG X.  VALUE OF REG A,B LOST
*
*       PNUM    - PRINT THE VALUE OF REG D AS AN UNSIGNED INTEGER.
*                 ZERO BLANKING IS DONE.  VALUE OF REG A,B,X LOST.
*
**********************************************************************


*
*       TERMINAL DRIVER ROUTINES
*
STATUS  EQU     $CFDD       ; TERMINAL STATUS REGISTER
DATA    EQU     $CFDC       ; TERMINAL DATA REGISTER

KEYCHK  LDA     STATUS    ; CHECK STATUS FOR KEY PUSH
        ANDA    #$08
        BEQ     CDONE     ; IF NO KEY PUSHED THEN JUST RETURN
        LDA     DATA      ; ELSE GET THE KEY'S VALUE AND RETURN
CDONE   RTS


GETC    BSR     KEYCHK    ; KEEP CHECKING THE KEYBOARD TILL
        BEQ     GETC      ; A KEY IS PUSHED.
        RTS


PUTC    LDB     STATUS    ; CHECK STATUS TO SEE IF TERMINAL READY
        ANDB    #$10
        BEQ     PUTC      ; WAIT UNTIL SPACE AVAILABLE TO OUTPUT CHAR
        STA     DATA      ; OUTPUT CHARACTER AND RETURN
        RTS


TEXT    LDA     ,X+       ; GET CHARACTER AT POINTER
        BEQ     TDONE     ; CHECK TO SEE IF ITS THE LAST CHAR. (NULL)
        BSR     PUTC      ; PRINT THE CHARACTER AND LOOP IF NOT NULL
        BRA     TEXT
TDONE   RTS               ; NULL ENCOUNTERED SO RETURN
```

```
PNUM     LDX      #K10000 ; GET START OF POWERS LIST
         CLR      ZBLANK  ; SET FOR ZERO BLANKING
NLOOP    CLR      DCOUNT  ; CLEAR DIGIT COUNT
ILOOP    SUBD     ,X      ; SUBTRACT THE CURRENT POWER
         INC      DCOUNT  ; DCOUNT SHOWS HOW MANY TIMES CURRENT POWER FITS
         BCC      ILOOP   ; REPEAT IF THE POWER FIT
         DEC      DCOUNT  ; ACTUALLY FIT ONE LESS THAN DCOUNT
         ADDD     ,X      ; ADD POWER CAUSE WENT TOO FAR
         STD      TEMP    ; SAVE IT AWAY WHILE THE DIGIT IS PRINTED
         LDA      DCOUNT  ; PREPARE DIGIT
         BNE      PDIGIT  ; IF DIGIT IS NOT ZERO: PRINT REGARDLESS
         TST      ZBLANK  ; ELSE TEST IF ZEROS ARE BEING BLANKED
         BEQ      SKIPDGT ; BLANK IT

PDIGIT   ADDA     #'0     ; CONVERT DIGIT TO ASCII
         BSR      PUTC    ; PRINT IT ON THE TERMINAL
         INC      ZBLANK  ; NOW THAT A DIGIT HAS BEEN PRINTED DISABLE BLANKING
SKIPDGT  LEAX     2,X     ; MOVE TO NEXT POWER ON THE LIST
         LDD      TEMP    ; GET REMAINDER OF TRHE NUMBER
         CMPX     #K1     ; REPEAT IF NOT AT END OF POWER LIST
         BNE      NLOOP
         LDA      TEMP+1  ; THE REMAINDER IS NOW THE LAST DIGIT
         ADDA     #'0     ; CONVERT TO ASCII
         BSR      PUTC    ; PRINT IT AND RETURN
         RTS

*        CONSTANTS USED IN THE ROUTINE PNUM

K10000   FDB      10000
K1000    FDB      1000
K100     FDB      100
K10      FDB      10
K1       FDB      1


*********************************************************************
*
*        I/O ROUTINES THAT USE THE I/O PORT TO GET INPUT EYE
*        VALUES AND OUTPUT DECODED DATA TO DATA ERROR ANALYZER.
*
*********************************************************************


BDAT     EQU      $CFE0   ; PORT B  DATA REGISTER
ADAT     EQU      $CFE1   ; PORT A  DATA REGISTER
DDRB     EQU      $CFE2   ; PORT B  DATA DIRECTION REGISTER
DDRA     EQU      $CFE3   ; PORT A  DATA DIRECTION REGISTER
ACR      EQU      $CFEB   ; BOTH PORTS  AUXILLARY CONTROL REGISTER
PCR      EQU      $CFEC   ; BOTH PORTS  PERIPHERAL CONTROL REGISTER
IFR      EQU      $CFED   ; BOTH PORTS  INTERRUPT FLAG REGISTER
IER      EQU      $CFEE   ; BOTH PORTS  INTERRUPT CONTROL REGISTER

DOUT     EQU      $CFF1   ; OUTPUT REGISTER FOR DECODED DATA AND CLOCK
OUTDDR   EQU      $CFF3   ; DATA DIRECTION REGISTER FOR ABOVE
```

```
**********************************************************************
*
*       THE STATE ORGANIZATION TABLE IS CURRENTLY SET UP TO DECODE
*       THE OUTPUT FROM A DECODER THAT IMPLEMENTS 171/133 CODE IN
*       THE FOLLOWING WAY.
*
*       SEE FORTRAN PROGRAM 'NEWSOT' TO CALCULATE THE STATE
*       ORGANIZATION TABLE FOR OTHER CODES.
*
*                   _______________________
*                  !  171 ENCODER           !
*                  !_______________________!-------->
*                    /       \   \   \   \
*                   /         \   \   \   \
*                ___/___________\___\___\___\__
*  INPUT========>!   !   !   !   !   !   !   !
*                !___!___!___!___!___!___!___!
*                  \   \       !   !       /
*                   \   \      !   !      /
*                    \___\_____!___!_____/_
*                  !  133 ENCODER           !-------->
*                  !_______________________!
*
*
*
**********************************************************************

        ORG     $0C00

SOT1    FDB     M0B2,T0011,T1100
        FDB     M1B2,T1001,T0110
        FDB     M2B2,T1100,T0011
        FDB     M3B2,T0110,T1001
        FDB     M4B2,T1100,T0011
        FDB     M5B2,T0110,T1001
        FDB     M6B2,T0011,T1100
        FDB     M7B2,T1001,T0110
        FDB     M8B2,T0011,T1100
        FDB     M9B2,T1001,T0110
        FDB     M10B2,T1100,T0011
        FDB     M11B2,T0110,T1001
        FDB     M12B2,T1100,T0011
        FDB     M13B2,T0110,T1001
        FDB     M14B2,T0011,T1100
        FDB     M15B2,T1001,T0110
        FDB     M16B2,T0110,T1001
        FDB     M17B2,T1100,T0011
        FDB     M18B2,T1001,T0110
        FDB     M19B2,T0011,T1100
        FDB     M20B2,T1001,T0110
        FDB     M21B2,T0011,T1100
        FDB     M22B2,T0110,T1001
        FDB     M23B2,T1100,T0011
        FDB     M24B2,T0110,T1001
        FDB     M25B2,T1100,T0011
        FDB     M26B2,T1001,T0110
        FDB     M27B2,T0011,T1100
        FDB     M28B2,T1001,T0110
        FDB     M29B2,T0011,T1100
        FDB     M30B2,T0110,T1001
        FDB     M31B2,T1100,T0011
        FDB     0,0,0
```

```
SOT2    FDB     M0B1,T0011,T1100
        FDB     M1B1,T1001,T0110
        FDB     M2B1,T1100,T0011
        FDB     M3B1,T0110,T1001
        FDB     M4B1,T1100,T0011
        FDB     M5B1,T0110,T1001
        FDB     M6B1,T0011,T1100
        FDB     M7B1,T1001,T0110
        FDB     M8B1,T0011,T1100
        FDB     M9B1,T1001,T0110
        FDB     M10B1,T1100,T0011
        FDB     M11B1,T0110,T1001
        FDB     M12B1,T1100,T0011
        FDB     M13B1,T0110,T1001
        FDB     M14B1,T0011,T1100
        FDB     M15B1,T1001,T0110
        FDB     M16B1,T0110,T1001
        FDB     M17B1,T1100,T0011
        FDB     M18B1,T1001,T0110
        FDB     M19B1,T0011,T1100
        FDB     M20B1,T1001,T0110
        FDB     M21B1,T0011,T1100
        FDB     M22B1,T0110,T1001
        FDB     M23B1,T1100,T0011
        FDB     M24B1,T0110,T1001
        FDB     M25B1,T1100,T0011
        FDB     M26B1,T1001,T0110
        FDB     M27B1,T0011,T1100
        FDB     M28B1,T1001,T0110
        FDB     M29B1,T0011,T1100
        FDB     M30B1,T0110,T1001
        FDB     M31B1,T1100,T0011
        FDB     0,0,0
```

```
************************************************************************
*
*       THIS IS THE STATE INFORMATION TABLE.  IT CONTAINS A TWO
*       BYTE METRIC AND A ONE BYTE POINTER FOR EACH OF THE 64 STATES.
*
************************************************************************
        ORG     $0E00

M0B1    FDB     0
M1B1    FDB     0
P0B1    FCB     0
P1B1    FCB     1
M2B1    FDB     0
M3B1    FDB     0
P2B1    FCB     2
P3B1    FCB     3
M4B1    FDB     0
M5B1    FDB     0
P4B1    FCB     4
P5B1    FCB     5
M6B1    FDB     0
M7B1    FDB     0
P6B1    FCB     6
P7B1    FCB     7
M8B1    FDB     0
M9B1    FDB     0
P8B1    FCB     8
P9B1    FCB     9
M10B1   FDB     0
M11B1   FDB     0
P10B1   FCB     10
P11B1   FCB     11
M12B1   FDB     0
M13B1   FDB     0
P12B1   FCB     12
P13B1   FCB     13
M14B1   FDB     0
M15B1   FDB     0
P14B1   FCB     14
P15B1   FCB     15
M16B1   FDB     0
M17B1   FDB     0
P16B1   FCB     16
P17B1   FCB     17
M18B1   FDB     0
M19B1   FDB     0
P18B1   FCB     18
P19B1   FCB     19
M20B1   FDB     0
M21B1   FDB     0
P20B1   FCB     20
P21B1   FCB     21
M22B1   FDB     0
M23B1   FDB     0
P22B1   FCB     22
P23B1   FCB     23
```

```
M24B1    FDB     0
M25B1    FDB     0
P24B1    FCB     24
P25B1    FCB     25
M26B1    FDB     0
M27B1    FDB     0
P26B1    FCB     26
P27B1    FCB     27
M28B1    FDB     0
M29B1    FDB     0
P28B1    FCB     28
P29B1    FCB     29
M30B1    FDB     0
M31B1    FDB     0
P30B1    FCB     30
P31B1    FCB     31
M32B1    FDB     0
M33B1    FDB     0
P32B1    FCB     32
P33B1    FCB     33
M34B1    FDB     0
M35B1    FDB     0
P34B1    FCB     34
P35B1    FCB     35
M36B1    FDB     0
M37B1    FDB     0
P36B1    FCB     36
P37B1    FCB     37
M38B1    FDB     0
M39B1    FDB     0
P38B1    FCB     38
P39B1    FCB     39
M40B1    FDB     0
M41B1    FDB     0
P40B1    FCB     40
P41B1    FCB     41
M42B1    FDB     0
M43B1    FDB     0
P42B1    FCB     42
P43B1    FCB     43
M44B1    FDB     0
M45B1    FDB     0
P44B1    FCB     44
P45B1    FCB     45
M46B1    FDB     0
M47B1    FDB     0
P46B1    FCB     46
P47B1    FCB     47
M48B1    FDB     0
M49B1    FDB     0
P48B1    FCB     48
P49B1    FCB     49
M50B1    FDB     0
M51B1    FDB     0
```

```
P50B1    FCB    50
P51B1    FCB    51
M52B1    FDB    0
M53B1    FDB    0
P52B1    FCB    52
P53B1    FCB    53
M54B1    FDB    0
M55B1    FDB    0
P54B1    FCB    54
P55B1    FCB    55
M56B1    FDB    0
M57B1    FDB    0
P56B1    FCB    56
P57B1    FCB    57
M58B1    FDB    0
M59B1    FDB    0
P58B1    FCB    58
P59B1    FCB    59
M60B1    FDB    0
M61B1    FDB    0
P60B1    FCB    60
P61B1    FCB    61
M62B1    FDB    0
M63B1    FDB    0
P62B1    FCB    62
P63B1    FCB    63


M0B2     FDB    0
M1B2     FDB    0
P0B2     FCB    0
P1B2     FCB    0
M2B2     FDB    0
M3B2     FDB    0
P2B2     FCB    0
P3B2     FCB    0
M4B2     FDB    0
M5B2     FDB    0
P4B2     FCB    0
P5B2     FCB    0
M6B2     FDB    0
M7B2     FDB    0
P6B2     FCB    0
P7B2     FCB    0
M8B2     FDB    0
M9B2     FDB    0
P8B2     FCB    0
P9B2     FCB    0
M10B2    FDB    0
M11B2    FDB    0
P10B2    FCB    0
P11B2    FCB    0
M12B2    FDB    0
M13B2    FDB    0
P12B2    FCB    0
P13B2    FCB    0
```

M14B2	FDB	0
M15B2	FDB	0
P14B2	FCB	0
P15B2	FCB	0
M16B2	FDB	0
M17B2	FDB	0
P16B2	FCB	0
P17B2	FCB	0
M18B2	FDB	0
M19B2	FDB	0
P18B2	FCB	0
P19B2	FCB	0
M20B2	FDB	0
M21B2	FDB	0
P20B2	FCB	0
P21B2	FCB	0
M22B2	FDB	0
M23B2	FDB	0
P22B2	FCB	0
P23B2	FCB	0
M24B2	FDB	0
M25B2	FDB	0
P24B2	FCB	0
P25B2	FCB	0
M26B2	FDB	0
M27B2	FDB	0
P26B2	FCB	0
P27B2	FCB	0
M28B2	FDB	0
M29B2	FDB	0
P28B2	FCB	0
P29B2	FCB	0
M30B2	FDB	0
M31B2	FDB	0
P30B2	FCB	0
P31B2	FCB	0
M32B2	FDB	0
M33B2	FDB	0
P32B2	FCB	0
P33B2	FCB	0
M34B2	FDB	0
M35B2	FDB	0
P34B2	FCB	0
P35B2	FCB	0
M36B2	FDB	0
M37B2	FDB	0
P36B2	FCB	0
P37B2	FCB	0
M38B2	FDB	0
M39B2	FDB	0
P38B2	FCB	0
P39B2	FCB	0
M40B2	FDB	0
M41B2	FDB	0
P40B2	FCB	0
P41B2	FCB	0

```
M42B2    FDB      0
M43B2    FDB      0
P42B2    FCB      0
P43B2    FCB      0
M44B2    FDB      0
M45B2    FDB      0
P44B2    FCB      0
P45B2    FCB      0
M46B2    FDB      0
M47B2    FDB      0
P46B2    FCB      0
P47B2    FCB      0
M48B2    FDB      0
M49B2    FDB      0
P48B2    FCB      0
P49B2    FCB      0
M50B2    FDB      0
M51B2    FDB      0
P50B2    FCB      0
P51B2    FCB      0
M52B2    FDB      0
M53B2    FDB      0
P52B2    FCB      0
P53B2    FCB      0
M54B2    FDB      0
M55B2    FDB      0
P54B2    FCB      0
P55B2    FCB      0
M56B2    FDB      0
M57B2    FDB      0
P56B2    FCB      0
P57B2    FCB      0
M58B2    FDB      0
M59B2    FDB      0
P58B2    FCB      0
P59B2    FCB      0
M60B2    FDB      0
M61B2    FDB      0
P60B2    FCB      0
P61B2    FCB      0
M62B2    FDB      0
M63B2    FDB      0
P62B2    FCB      0
P63B2    FCB      0
```

```
***********************************************************************
*
*       128 BYTE AREA USED TO MAP THE EYE VALUES RECIEVED FROM
*       THE ADC BOARD.
*
*       CURRENT PROFILE IS A ONE TO ONE MAPPING ( X=>X )
*
**********************************************************************

MAP     FCB     0,1,2,3,4,5,6,7
        FCB     8,9,10,11,12,13,14,15
        FCB     16,17,18,19,20,21,22,23
        FCB     24,25,26,27,28,29,30,31
        FCB     32,33,34,35,36,37,38,39
        FCB     40,41,42,43,44,45,46,47
        FCB     48,49,50,51,52,53,54,55
        FCB     56,57,58,59,60,61,62,63
        FCB     64,65,66,67,68,69,70,71
        FCB     72,73,74,75,76,77,78,79
        FCB     80,81,82,83,84,85,86,87
        FCB     88,89,90,91,92,93,94,95
        FCB     96,97,98,99,100,101,102,103
        FCB     104,105,106,107,108,109,110,111
        FCB     112,113,114,115,116,117,118,119
        FCB     120,121,122,123,124,125,126,127

* ALTERNATE MAP, MU-LAW: MU = 100.
*
*MAP FCB  0,0,0,1,1,1,1,2
     FCB  2,2,2,3,3,3,3,4
     FCB  4,4,5,5,5,5,6,6
     FCB  6,7,7,8,8,8,9,9
     FCB  10,10,10,11,11,12,12,13
     FCB  13,14,15,15,16,17,17,18
     FCB  19,20,21,22,23,24,25,27
     FCB  28,30,32,35,38,42,47,56
     FCB  71,80,85,89,92,95,97,99
     FCB  100,102,103,104,105,106,107,108
     FCB  109,110,110,111,112,112,113,114
     FCB  114,115,115,116,116,117,117,117
     FCB  118,118,119,119,119,120,120,121
     FCB  121,121,122,122,122,122,123,123
     FCB  123,124,124,124,124,125,125,125
     FCB  125,126,126,126,126,127,127,127
*
*
```

```
* ALTERNATE MAP, INVERSE MU-LAW: MU = 100.
*
*MAP FCB  0,4,9,13,16,20,23,26
     FCB  28,31,33,35,37,39,41,43
     FCB  44,45,47,48,49,50,51,52
     FCB  53,54,54,55,56,56,57,57
     FCB  58,58,59,59,59,60,60,60
     FCB  61,61,61,61,62,62,62,62
     FCB  62,62,62,63,63,63,63,63
     FCB  63,63,63,63,63,63,63,63
     FCB  64,64,64,64,64,64,64,64
     FCB  64,64,64,64,64,65,65,65
     FCB  65,65,65,65,66,66,66,66
     FCB  67,67,67,68,68,68,69,69
     FCB  70,70,71,71,72,73,73,74
     FCB  75,76,77,78,79,80,82,83
     FCB  84,86,88,90,92,94,96,99
     FCB  101,104,107,111,114,118,123,127
*
*
```

```
*********************************************************************
*
*       THIS IS THE LIST OF ADDRESSES USED TO REFERNCE THE VARIOUS
*       COLUMNS OF HISTORY DURING HISTORY UPDATING.  CURCOL REFERS
*       TO THIS TABLE TO FIND THE LOCATION OF THE CURRENTLY ACTIVE
*       COLUMN OF HISTORY.
*
*********************************************************************

        FDB     H1
        FDB     H2
        FDB     H3
        FDB     H4
        FDB     H5
        FDB     H6
        FDB     H7
COLBEG  FDB     H0
        FDB     H1
        FDB     H2
        FDB     H3
        FDB     H4
        FDB     H5
        FDB     H6
        FDB     H7
COLLIM  FDB     H0
        FDB     H1
        FDB     H2
        FDB     H3
        FDB     H4
        FDB     H5
        FDB     H6
        FDB     H7


*********************************************************************
*
*       THIS IS THE AREA RESERVED FOR THE STORAGE OF LONG TERM
*       HISTORY.  THERE ARE CURRENTLY 8 COLUMNS WHICH EACH HOLD
*       6 BITS (CONSTRAINT LENGTH) OF HISTORY.
*
*********************************************************************

        ORG     $1000

H0      RMB     64
H1      RMB     64
H2      RMB     64
H3      RMB     64
H4      RMB     64
H5      RMB     64
H6      RMB     64
H7      RMB     64
```

```
****************************************************************
*
*       THIS IS THE DIRECT PAGE AREA.  THIS IS WHERE ALL VARIABLES
*       ARE STORED SINCE ACCESS IS SLIGHTLY FASTER.
*
****************************************************************

        ORG     $100

SPSAVE  FDB     0          ; LOCAL: ACS  TEMP. STORAGE FOR SYSTEM SP
TEMP    FDB     0          ; LOCAL: PNUM  TEMP. NUMBER STORAGE
COUNT   FCB     0          ; LOCAL: SEARCH  LOOP COUNTER
SCOUNT  FCB     0          ; LOCAL: SCALE  LOOP COUNTER

BESTM   FDB     0          ; COMMON: SEARCH & SCALE  BEST METRIC VALUE
BESTV   FCB     0          ; COMMON: SEARCH & HIST  POINTER TO BEST PATH
TEMPA   FCB     0          ; LOCAL: SHIFT  STORAGE FOR EYE0
TEMPB   FCB     0          ; LOCAL: SHIFT  STORAGE FOR EYE1
COMA    FCB     0          ; LOCAL: SHIFT  STORAGE FOR NEG. EYE0
COMB    FCB     0          ; LOCAL: SHIFT  STORAGE FOR NEG. EYE1

T0000   FDB     0          ; COMMON TO SHIFT AND ACS
T0001   FDB     0          ; THESE 16 LOCATIONS HOLD THE PRE-CALCULATED
T0010   FDB     0          ; BRANCH DISTANCES
T0011   FDB     0
T0100   FDB     0
T0101   FDB     0
T0110   FDB     0
T0111   FDB     0
T1000   FDB     0
T1001   FDB     0
T1010   FDB     0
T1011   FDB     0
T1100   FDB     0
T1101   FDB     0
T1110   FDB     0
T1111   FDB     0

CURCOL  FDB     COLBEG    ; LOCAL: HIST  POINTER TO POSITION IN RING BUFFER

ERRORS  FDB     0          ; LOCAL: ERRCHK  BLOCK ERROR COUNTER
BCOUNT  FDB     0          ; LOCAL: ERRCHK  BLOCK BIT COUNTER
STOR1   FCB     0          ; LOCAL: ERRCHK  2 BYTES MOST RECENT BITS
STOR2   FCB     0          ;                (SHIFT REGISTER)
PN1     FCB     0          ; LOCAL: NEXTPN  1/2 OF PN SHIFT REGISTER
PN2     FCB     0          ; LOCAL: NEXTPN  OTHER 1/2 OF PN SHIFT REGISTER
CHECK   FCB     0          ; LOCAL: NEXTPN  PN OUTPUT BIT TEMP. STORAGE
BITOUT  FCB     0          ; COMMON: PUTBIT & ERRCHK  RECENT DECODED OUTPUT BIT

SFLAG   FCB     0          ;  LOCAL:  ERRCHK  FLAGS 1ST RESYNC END

RESYNC  FDB     0          ; LOCAL: ERRCHK  RESYNC COUNT
NOBLOCK FDB     0          ; LOCAL: ERRCHK  TOTAL BLOCK COUNT
ERRS    FDB     0          ; LOCAL: ERRCHK  BAD BLOCK COUNT
ALLOK   FDB     0          ; LOCAL: ERRCHK  NUMBER OF GOOD BLOCKS
DCOUNT  FCB     0          ; LOCAL: PNUM  DIGIT COUNT
ZBLANK  FCB     0          ; LOCAL: PNUM  ZERO BLANKING FLAG
OUT     FCB     0          ; COMMON: HIST & PUTBIT  6 BIT OUTPUT SR FOUND BY HIST
OUTPORT FCB     0          ; COMMON: PUTOUT, STROBUP & STROBDN  IMAGE OF OUT PORTI

        END START
```

APPENDIX C

Let K be the message length and L the number of subpackets that can be accomodated in a frame. In mode A each frame transmission reduces the backlog by Q to L subpackets. In mode B each frame transmission reduces the backlog by no more than the number of subpackets outstanding just prior to the frame transmission.

Mode A Analysis:

Let MF(K) be the number of mode A transmission required for a message of length K. If K=QL+R, then at least Q mode A transmissions will be required. Let s_i be the number of successful subpackets transmitted after i frame transmissions. Then s_i is the sum of i independent Binomial random variables and is thus itself Binomial with parameters iL and e, since the number of successful transmissions per frame is Binomial with parameters L and e. Mode A transmissions will cease after s_i first exceeds K-L since the backlog will then be less than L. Thus:

$$\Pr(\,MF(K) > i) = \Pr(\,s_i \leq K - L\,)$$

$$= \sum_{j=0}^{K-L} \Pr(\,s_i = j\,)$$

$$= \sum_{j=0}^{K-L} \binom{L}{j} e^{L-j}\,(1-e)^j$$

The expected value of MF(K) is given by

$$E(M(K)) = \sum_{k=1}^{\infty} \Pr(\,MF(K) \geq k)$$

$$= Q + \sum_{i=Q+1}^{\infty} \Pr(\,MF(K) \geq i)$$

$$= Q + \sum_{i=Q}^{\infty} \left\{ \sum_{j=0}^{K-L} \Pr(\,s_i = j) \right\}$$

The expression inside the brackets can be evaluated exactly for Q=1,2. For larger values of i the Gaussian approximation for a Binomial random variable can be used:

$$\sum_{j=0}^{K-L} \Pr(s_i = j) \approx \int_{-\infty}^{(K-L-Lie)/(Le(1-e)i)^{1/2}} \frac{1}{\sqrt{2\pi}} e^{-x^2/2} \, dx$$

The terms in the infinite series vanish very quickly so only a few terms are required to evaluate the series.

Mode B Analysis

Let r be the number of outstanding packets just prior to the first mode B transmission, $0 \leq r \leq L-1$. Each of these subpackets will have one chance per frame transmission as long as they have not been received correctly at the receiver. Thus the number of transmissions required by each of these subpackets is geometrically distributed with probability of success 1-e. Mode B transmissions will continue as long as there are outstanding packets. Thus the total number of mode B transmissions is given by:

$$MP(r) = \max(N_1, N_2, \ldots, N_r)$$

where N_i are geometrically distributed random variables with parameter 1-e. Since the subpacket errors are assumed to be independent:

$$\begin{aligned} P(MP(r) \leq j) &= \Pr(N_1 \leq j, \; N_2 \leq j, \; . \; . \; ., \; N_r \leq j) \\ &= \prod_{k=1}^{r} \Pr(N_k \leq j) \\ &= (1 - e^j)^r \end{aligned}$$

The mean number of mode B transmissions is then given by

$$\begin{aligned} E(MP(r)) &= \sum_{k=1}^{\infty} \Pr(MP(r) \;\; k) \\ &= 1 + \sum_{k=1}^{r} \binom{r}{k} (-1)^{k+1} \frac{e^{k-1}}{1-e^{k-1}} \end{aligned}$$

The probability distribution that the residual is r as a function of K is found as follows. For $i \geq Q$, and $0 \leq r < L$,

$$\Pr(r \,/\, MF(K) = i) = \Pr(r \,/\, K-L < s_i \leq K)$$

$$= \frac{\Pr(s_i = K-r)}{\Pr(K-L < s_i \leq K)}$$

By unconditioning over MF(K) we obtain:

$$\Pr(r) = \sum_{i=Q}^{\infty} \frac{\Pr(s_i = K-r)}{\Pr(K-L < s_i \leq K)} \qquad \Pr(MF(K)=i)$$

This expression is wieghted average of overlapped length-L segments of the probability mass function of s_i for values exceeding K-L. As K increases the overlapped segments approach a uniform distribution and Pr(r) will approach 1/L.

LEON-GARCIA, ALBERTO

Techniques for improving the reliability of data transmission over...
