N73-11141

NASA TECHNICAL REPORT

NASA TR R-396

NASA TR R-396

CASE FILE COPY

SIMULATION RESULTS FOR THE VITERBI DECODING ALGORITHM

by Bartus H. Batson, Robert W. Moorehead, and S. Zafar H. Taqvi

Manned Spacecraft Center
Houston, Texas 77058

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION • WASHINGTON, D. C. • NOVEMBER 1972

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TR R-396		

4. Title and Subtitle	5. Report Date
SIMULATION RESULTS FOR THE VITERBI DECODING ALGORITHM	November 1972
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Bartus H. Batson and Robert W. Moorehead, MSC; and S. Zafar H. Taqvi, Lockheed Electronics Company, Inc.	MSC S-291
	10. Work Unit No.
9. Performing Organization Name and Address	914-50-50-17-72
Manned Spacecraft Center Houston, Texas 77058	11. Contract or Grant No.
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Report
National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Concepts involved in determining the performance of coded digital communications systems are introduced. The basic concepts of convolutional encoding and decoding are summarized, and hardware implementations of sequential and maximum-likelihood (Viterbi) decoders are described briefly. Results of parametric studies of the Viterbi decoding algorithm are summarized. Bit error probability is chosen as the measure of performance and is calculated, by using digital computer simulations, for various encoder and decoder parameters. Results are presented for code rates of one-half and one-third, for constraint lengths of 4 to 8, for both hard-decision and soft-decision bit detectors, and for several important systematic and nonsystematic codes. The effect of decoder block length on bit error rate also is considered, so that a more complete estimate of the relationship between performance and decoder complexity can be made.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
*Convolutional Encoding *Sequential Decoding *Convolutional Decoding *Viterbi Decoding *Coded Digital Communications Systems *Digital Communications Systems	

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
None	None	58	$3.00

* For sale by the National Technical Information Service, Springfield, Virginia 22151

CONTENTS

TABLES

FIGURES

SIMULATION RESULTS FOR THE VITERBI DECODING ALGORITHM

By Bartus H. Batson, Robert W. Moorehead, and S. Zafar H. Taqvi*
Manned Spacecraft Center

SUMMARY

Performance predictions for convolutional decoders using the Viterbi decoding algorithm are presented in this report. Bit error probability is chosen as the measure of performance and, by using digital computer simulations, is calculated as a function of energy per bit per noise spectral density for various encoder and decoder parameters. Coding gains based on comparisons with uncoded, coherent phase-shift-keyed system performance are determined for code rates of one-half and one-third and for constraint lengths of 4 to 8. Both hard-decision and soft-decision decoders are considered, and bit error probability is calculated for both systematic and nonsystematic codes. The effect of decoder block length on decoding performance also is included to provide a more complete estimate of the relationship between performance and decoder complexity.

INTRODUCTION

One characteristic that has made all-digital communications links appear increasingly attractive for many applications in recent years is that error control encoding and decoding can be applied to achieve significant improvements in overall link performance. The introduction of coding into a digital link allows, for a fixed transmit (or receive) power level and for an allowable bit error probability, transfer of more information per unit time. Alternately, for a fixed information rate, the introduction of coding can provide a reduction in the transmit or receive power level required to maintain a specified error probability. The exact increase in information rate that can be achieved, or the amount of coding gain (allowable reduction in power level) that is realizable, depends on the particular class of encoding and decoding technique employed and on various encoder and decoder parameters that must be selected by the communications system design engineer.

---

*Lockheed Electronics Company, Inc.

Convolutional codes generally are conceded to be better than block codes for many channels (ref. 1), particularly with respect to ease of implementation, equipment complexity, power consumption, weight, and flexibility. Various algorithms, including several sequential decoding algorithms (refs. 2 and 3), are available for decoding convolutional codes. Sequential decoders incorporate searchback operations in the hypothesis and testing of various paths through the convolutional code tree. Thus, if a path is hypothesized and subsequently is determined to be in error, the decoder has the capability to discard that path, back up, and test other possible paths. The advantage of such a capability is that the probability of an undetected error appearing at the decoder output is extremely small. However, a memory unit is required to store past data for possible recall. In addition, a buffer is required to store incoming bits while the decoding operations (including searchbacks) are being performed. Because there is always some finite probability that the number of operations required to decode a particular bit can be quite large, it is possible for the input buffer to overflow. During such an overflow condition, the decoder output consists either of uncorrected channel bits or of erasures.

Parameters that affect the achievable error probability when using sequential decoding include information rate, code rate, code constraint length, input buffer size, and memory size (which determines the allowable number of searchbacks). In general, for fixed power levels, a smaller error probability can be obtained by decreasing information rate, by decreasing code rate (adding more redundancy to the transmitted sequence), or by increasing any of the other parameters noted previously. The code constraint length used for systems employing sequential decoding is typically rather large (greater than 20).

A primary advantage of sequential decoding is that a rather large performance gain is achievable. However, the variable decoding time per bit that results because of searchbacks is a distinct disadvantage for some applications. Other disadvantages include the requirements for a memory unit and an input buffer. Fortunately, two of these disadvantages (the variable decoder output rate and the requirement for an input buffer) become inconsequential when the data rate is low enough to allow a large speed advantage (computation rate per data rate) of the logic unit. Sequential decoding, therefore, is a very attractive technique for use in systems having moderate data rates (below perhaps a few megabits per second).

The Viterbi algorithm (ref. 4) for decoding convolutional codes recently has received considerable attention, largely because of certain inherent advantages over the various sequential decoding algorithms. This algorithm has been shown (ref. 5) to be maximum-likelihood and, therefore, optimum for the decoding of convolutional codes. The primary advantage of a Viterbi algorithm decoder is speed. Whereas the performance gain (over no coding) achievable by using sequential decoding is limited primarily by the information rate, the gain achievable by using Viterbi decoding is limited primarily by the constraint length of the code and is relatively independent of the information rate. This independence is possible because no searchbacks are required by the Viterbi algorithm, and only a very small logic speed advantage is required. Viterbi decoders have the additional advantage of operating at a fixed rate; thus, no input buffer is required.

The primary disadvantage of Viterbi decoders is that the decoder hardware complexity increases exponentially with increasing code constraint length. Because

performance gain also increases with constraint length, hardware constraints impose a limit on the performance gain achievable by using Viterbi decoders. In practice, these hardware constraints dictate that the constraint length be limited to approximately 8. Fortunately, the decoder hardware requirements do not increase substantially for lower rate codes or when soft decisions are provided by the receiver.

SYMBOLS

CD	coding gain
E_b	energy per information bit
E_c	energy per channel bit
K	constraint length of convolutional code
L	search length or block length of Viterbi decoder
N_0	single-sided noise spectral density
P_e	bit error probability
Q	number of receiver quantization levels
R	rate of convolutional code (R = 1/V)
$S_0(t), S_1(t)$	transmitted waveforms corresponding to binary 0 and 1
$S_0^*(t), S_1^*(t)$	received (noisy) waveforms corresponding to binary 0 and 1
V	number of encoded (channel) bits per information bit

FUNDAMENTALS OF DIGITAL COMMUNICATIONS SYSTEMS EMPLOYING CONVOLUTIONAL ENCODING AND DECODING

In the following sections, the concepts involved in determining the performance of the general coded digital communications system are introduced, and the basic concepts of convolutional encoding and decoding are discussed. This material is intended merely to provide background information that may aid some readers in developing a more complete understanding of the convolutional decoding problem.

The General Coded Digital Communications System

A simple coded digital communications system is illustrated in block diagram form in figure 1. The system consists of a source, which generates data in the form of binary digits (information bits); an encoder, which converts each information bit into V channel bits (V = 2 for this example) according to a certain scheme governed by the code; and a transmitter, which generates analog waveforms $S_0(t)$ and $S_1(t)$ corresponding to 0 and 1 for transmission through the channel. The signal is corrupted by noise (generally assumed to be additive, white, and Gaussian) in the channel, and the input to the receiver consists of a sequence of noisy waveforms $S_0^*(t)$ and $S_1^*(t)$. A decision device, which processes the noise waveforms and provides estimates of the corresponding transmitted channel bits, is incorporated in the receiver. Because the input to the decision device is noisy, there is always some finite probability that a bit decision is erroneous. It has been shown (ref. 6) that, for binary signaling over an additive, white, Gaussian noise channel and for optimum (correlation or matched filter) detection, the probability of bit error at the receiver output is

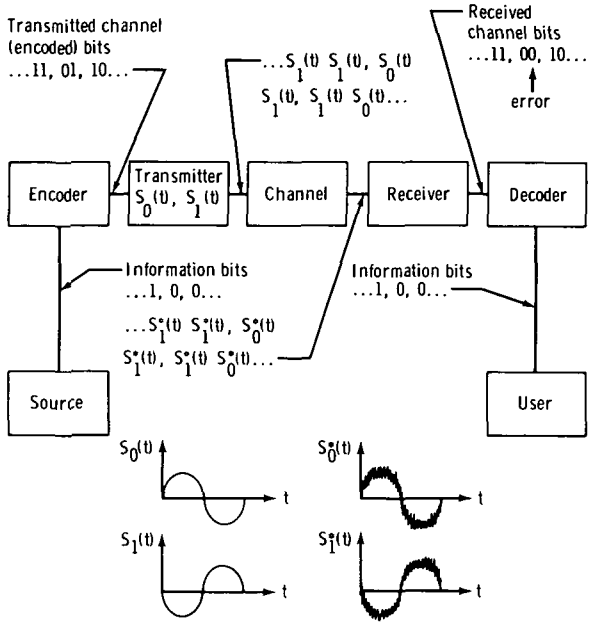


Figure 1.- Block diagram of coded digital communications system.

$$P_e = \frac{1}{2}\,\mathrm{erfc}\sqrt{\frac{E_c}{N_0}} \qquad (1)$$

where E_c is the signal energy of a channel bit at the input to the decision device, N_0 is the single-sided noise spectral density of the receiver, and erfc ( ) is the complementary error function defined by

$$\mathrm{erfc}\ X = \frac{2}{\sqrt{\pi}}\int_X^\infty e^{-\xi^2}\,d\xi \qquad (2)$$

If coding is not employed by the system of figure 1, however, the receiver bit detector makes decisions on information bits directly. With constant transmit and receive power levels for the coded and uncoded systems, more signal energy is available for a direct decision on an information bit than for a decision on a channel bit. The redundancy added to the transmitted signal when coding is incorporated into the system results in less energy per channel bit and, therefore, in a higher bit error probability at the receiver output. The task of the decoder, which operates on the reconstructed sequence

of channel bits at the receiver output, is to correct as many bit errors as possible. For the achievement of a net coding gain, the information bit error probability after decoding must be less than it was for the uncoded system. An uncoded information bit error probability curve can be constructed by substituting E_b (energy per information bit) for E_c in equation (1). This uncoded bit error probability curve is shown in figure 2.

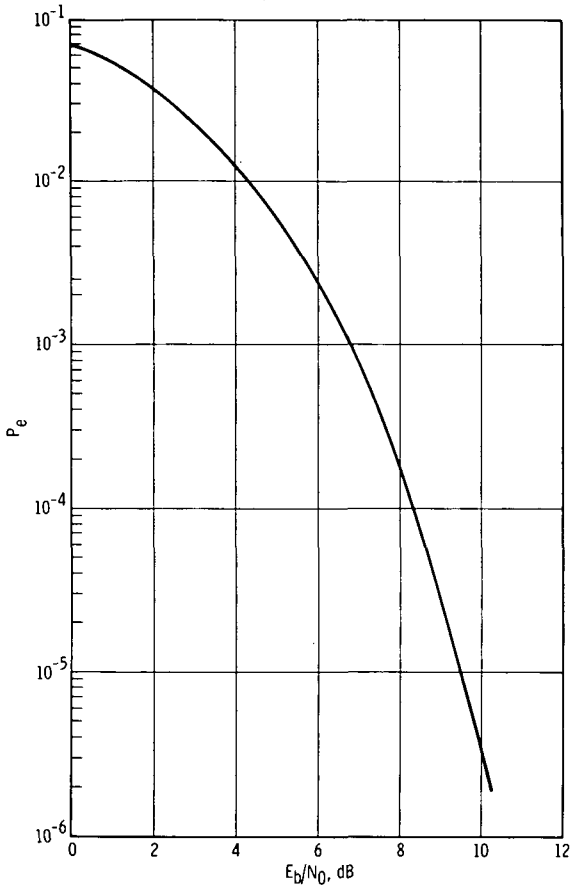


Figure 2. - Bit error probability as a function of E_b/N_0 for an uncoded digital communications system.

Several encoding techniques are available for incorporation into digital transmission systems. These techniques generally are categorized as either block or convolutional. For block codes (frequently referred to as algebraic codes), a certain structured block of channel bits is assigned to each possible group of information bits. Block codes are highly structured, in a mathematical sense, and the various decoding algorithms for block codes generally either exploit code properties that result from this mathematical structure or apply probabilistic information obtained from the received signal. For the reasons pointed out in the introduction to this paper, convolutional codes were chosen for the current investigation.

Convolutional Encoding Fundamentals

For convolutional encoding, each information bit (rather than each block of information bits) is encoded into V channel bits. A simple convolutional encoder, which consists of a shift register with K stages connected in some prescribed manner to V modulo 2 adders, is shown in figure 3. For each input bit shifted into the register, there are V encoded output bits, corresponding to one revolution of the commutator. Therefore, the length of the output (channel) sequence will be V times the length of the input message. The rate of the code is defined as

$$R = \frac{1}{V} \tag{3}$$

and is a measure of the number of message (information) bits per transmitted symbol. The length of the shift register K is referred to as the constraint length of the code and

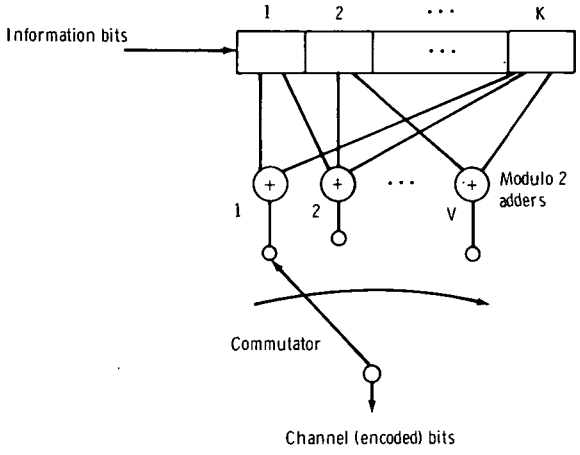
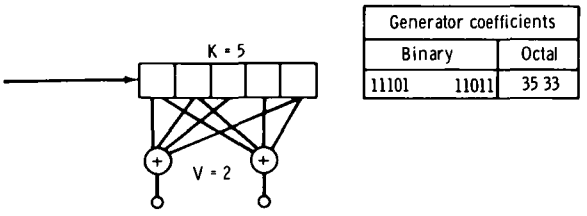


Figure 3. - Binary (K, V) convolutional encoder.

is, roughly, a measure of the duration in which the encoded output bits are affected by any particular input bit. That is, each group of V channel bits depends on the current information bit and on the K - 1 previous information bits.

A particular convolutional encoder may be described in terms of a set of generator coefficients that specify which stages of the shift register are connected to each modulo 2 adder. For example, two typical encoder configurations are shown in figure 4. The binary digits represent actual connections to each adder (e.g., binary 11101 11011 indicates that the first adder is connected to the first, second, third, and fifth stages of the five-stage register, and that the second adder is connected to the first, second, fourth, and fifth stages). The binary generator coefficients sometimes are converted to octal form for notational convenience. Thus, binary 11101 11011 becomes octal 35 33, and binary 110101 101111 becomes octal 65 57.



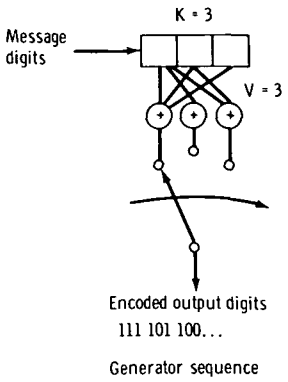
(a) First example.

(b) Second example.

Figure 4. - Typical convolutional encoder connections.

An alternate means of describing a particular encoder configuration is the generator sequence. For a (K, V) convolutional encoder, the generator sequence is equivalent to the output sequence that results from transmitting a K-bit input message consisting of a leading 1 followed by all zeros. One method by which the generator sequence can be determined is shown in figure 5. However, there is another way of determining the generator sequence without construction of a table.

Note that if a single 1 is located in some stage (e.g., stage x) of the shift register, the output digit from any given adder will be a 1 if that adder is connected to stage x, or a 0 otherwise. Thus, the first V digits of the generator sequence represent the modulo 2 adders that are connected to the first stage of the K-stage register, the second V digits represent the adders that are connected to the second stage of the register, and



Shift no.	Input digits	Register contents	Encoded output
Initial state	--	000	--
1	1	100	111
2	0	010	101
3	0	001	100
4	0	000	000
5	0	000	000
•	•	•	•
•	•	•	•
•	•	•	•

Figure 5. - Determination of the generator sequence for a (3, 3) convolutional code.

so forth. For the example shown in figure 5, all three adders are connected to the first stage; therefore, the first three digits of the generator sequence are 111. Because only the first and third adders are connected to the second stage of the register, the second three digits of the generator sequence are 101. Likewise, the last three digits of the generator sequence are 100. The length of the generator sequence (111 101 100) is KV = (3)(3) = 9 bits.

Because the generator coefficients specify which stages of the shift register are connected to each modulo 2 adder and the generator sequence specifies which modulo 2 adders are connected to each stage of the shift register, there should be (and indeed is) a direct relationship between the generator sequence and the generator coefficients. Given the generator coefficients, the generator sequence can be determined by inspection. Thus, for the example shown in figure 4(a), in which the generator coefficients are 11101 11011, the generator sequence is 11 11 10 01 11 and is obtained by pairing like digits (first with first, second with second, etc.) of the two generator coefficients. A reverse procedure, in which the generator sequence is first separated into groups of V digits, can be followed to obtain the generator coefficients from the generator sequence.

For those particular encoder configurations in which the first modulo 2 adder is connected to only the first stage of the shift register, the resulting codes are said to be systematic. For systematic codes, then, the first bit of each V-bit channel sequence is the same as the current information bit. All codes that are not systematic are said to be nonsystematic.

As will be shown later, the Viterbi decoding algorithm yields better performance with nonsystematic convolutional codes. From a set of convolutional codes, only certain "good" codes are used for implementation. These good codes lead to minimum decoding error probability. A computer search technique usually is applied to select a good convolutional code, based on preselected criteria (such as maximized minimum distance, maximized free distance, and noncatastrophic error propagation properties).

Convolutional Decoding Fundamentals

Each bit shifted into the K-bit register results in one of two possible V-bit output sequences. One possible output sequence corresponds to shifting in a 0, whereas the other corresponds to shifting in a 1. The specific V-bit sequence that results when a bit is shifted into the register, however, depends on the previous K - 1 bits that are retained in the register. Thus, a given input message bit affects the current V-bit output sequence and the next K - 1 V-bit output sequences as well.

The behavior of any convolutional encoder may be illustrated diagrammatically by a tree structure, as shown in figure 6 for the encoder of figure 5. The labels on the branches indicate encoder outputs. By convention, the code tree is arranged so that the upper branch from any node corresponds to shifting a 0 into the K-stage register and the lower branch to shifting in a 1. The encoded output sequence corresponding to a given input message sequence may be found by following the appropriate path through the code tree. For example, an input message sequence of 1011... results in an output sequence of 111 101 011 010 ....

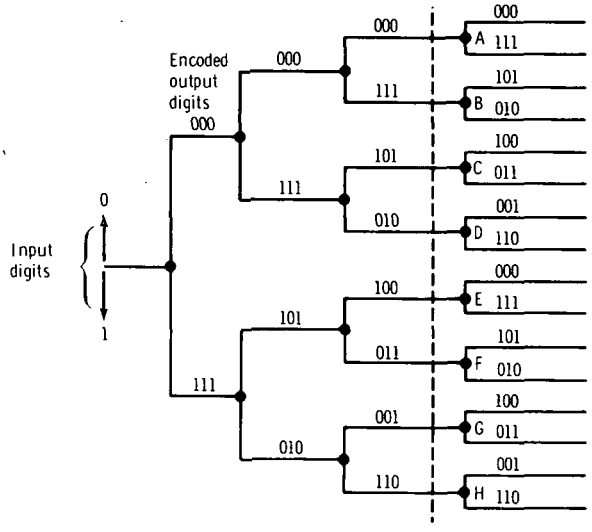


Figure 6. - Code tree for a (3, 3) convolutional encoder with generator sequence 111 101 100.

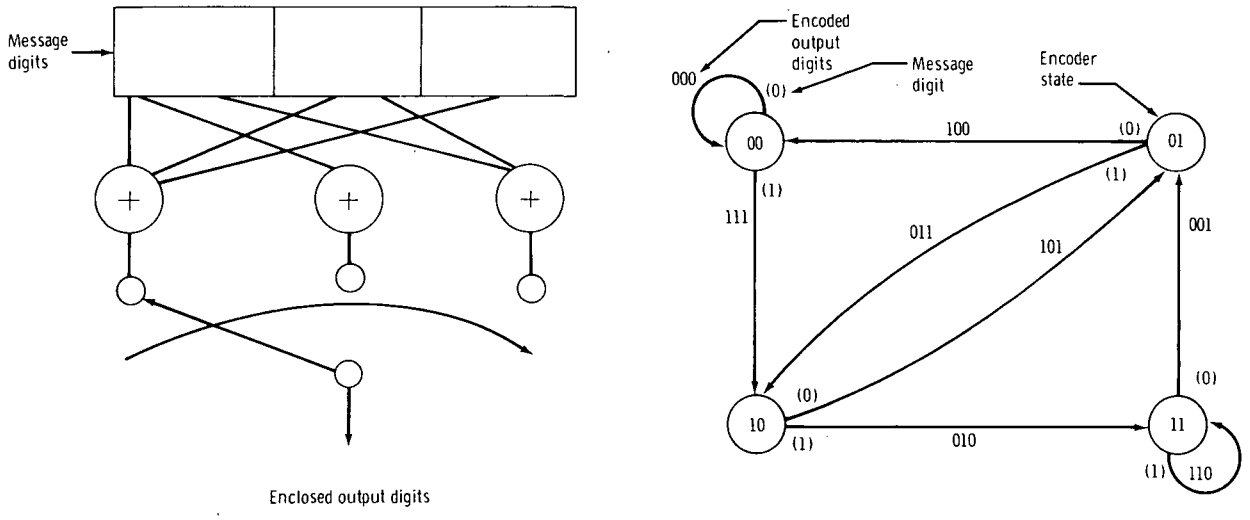
Decoding of convolutional codes invariably is based on the code tree structure. Sequential decoding algorithms assume a tentative transmitted message, encode this message with a replica of the encoder, and compare the resultant coded output sequence with the actual received sequence. If these two coded sequences agree to within some specified amount, decoding is assumed to have been accomplished. If agreement between the two sequences does not meet the desired criteria, another tentative message is assumed, and the process is repeated.

The optimum convolutional decoder chooses the path through the code tree that has maximum likelihood, given the received sequence. That is, the decoded message sequence will provide a coded output sequence that is closer (differs in fewer bit positions) to the actual received sequence than the coded output sequence corresponding to any other possible message sequence. Because the code tree apparently is infinite (i.e., the number of branches doubles each time a bit is shifted into the encoder), choosing a maximum-likelihood path through a code tree such as that shown in figure 6 would appear to be a hopeless problem. This is not the case, however, as will be pointed out in subsequent discussion.

Inspection of figure 6 reveals that, although the code tree does grow without bound as more and more message digits are shifted into the encoder, the growth is completely repetitive after a point. Note that as the Kth (third) message bit is shifted into the encoder, eight possible output branches (000 to 111) exist. As the (K + 1)th message bit is shifted in, there are 16 possible branches, but only eight of these are distinct. Because the upper eight branches are identical to the lower eight branches, it is not necessary to show all 16 branches on the diagram. The tree structure could be simplified greatly by connecting nodes A and E, B and F, C and G, and D and H. If this is done for the (K + 1)th input bit, it becomes evident that the same procedure could be repeated for the (K + 2)th input bit, again for the (K + 3)th input bit, and so forth. Thus, at no location in the tree is it necessary to draw more than 2^K total branches or 2^{K-1} total nodes.

Additional insight into the basic simplicity of the convolutional tree structure can be gained by examining the state diagram of the encoder. An encoder state is defined as the contents of the first (most recent) K - 1 stages of the K-stage shift register. There are 2^{K-1} possible states, corresponding to the 2^{K-1} possible combinations of K - 1 binary digits. Given an encoder state, only two possible encoder states can be entered as a message digit is shifted into the register. The first possible state corresponds to shifting in a 0, whereas the other possibility corresponds to shifting in a 1. Because the most ancient (Kth) bit in the encoder register is dumped as a message bit is shifted in and, therefore, cannot affect any subsequent state, that Kth bit is not considered when defining the register state.

The state diagram for the (3, 3) convolutional encoder that was considered in previous examples in this report is shown in figure 7. The circled numbers represent encoder states, the single digits (in parentheses) represent input message digits, and the triple digits represent the encoded output digits that occur as the encoder state is changed. For example, if the current encoder state is 10, two possible states can be reached as a message digit is shifted in. These possible states are 01 and 11 and correspond, respectively, to 0 and 1 inputs. Assuming a 0 input, the total shift register contents will be 010, resulting in the encoded output digits 101. Alternately, if a 1 is shifted in, the register contents will be 110, resulting in the encoded output digits 010. This procedure was followed to obtain the complete encoder state diagram shown in figure 7.



(a) Encoder. (b) State diagram.

Figure 7. - State diagram for a (3, 3) convolutional encoder with generator sequence 111 101 100.

The state diagram allows a convenient and straightforward determination of the output message corresponding to a particular sequence of input digits. For example, if the input sequence is 10110..., and the encoder is initially at state 00 in figure 7, the first digit (a 1) results in an output of 111 and a new encoder state of 10. The second digit (a 0) changes the state to 01 and provides an output of 101. The third digit (another 1) changes the state back to 10 and provides an output of 011, and so forth. This procedure could be repeated for an indefinite sequence of input message bits and is more convenient than tracing through the tree diagram.

The state diagram can be redrawn in a form that is even more convenient for discussing the operation of the Viterbi decoding algorithm. The redrawn state diagram is shown in figure 8, in which the 2^{K-1} states are represented as nodes on a trellis structure. The branches of the trellis represent possible moves from state to state. It should be emphasized again that only two possible states can be entered from a given state. Likewise, a given state can be entered from one of only two possible previous states.

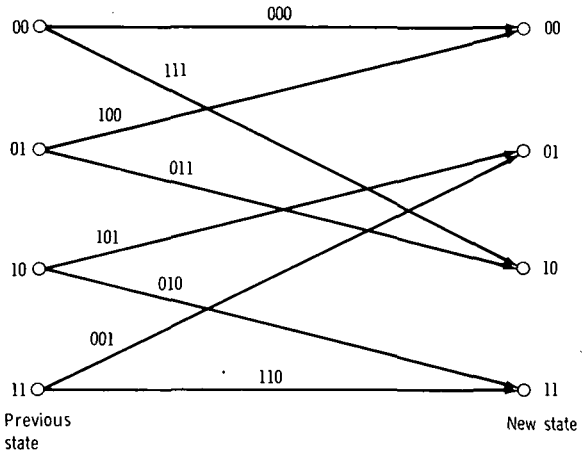


Figure 8. - Redrawn state diagram for a (3, 3) convolutional encoder with generator sequence 111 101 100.

The possible moves (from state to state) are independent of the particular generator sequence being used. For example, state 00 can be entered only from state 00 or state 01, regardless of the modulo 2 adder connections of the encoder. However, the encoded output digits that correspond to each move (and are labeled beside each possible move) are code-dependent. For instance, a different generator sequence results, in general, in encoded output digits other than 101 and 001 corresponding to the two paths terminating in state 01.

In the trellis structure of figure 8, the uppermost of the two paths leaving any one node corresponds to shifting a 0 into the register, while the lower path corresponds to shifting in a 1. This statement can be verified by comparing figure 8 with the original state diagram shown in figure 7. All four paths terminating in the upper two states (00 and 01) are paths that resulted when a 0 was shifted into the register, and all four paths terminating in the lower two states (10 and 11) are paths that resulted when a 1 was shifted in because the first digit of a new register state has to be the same as the message digit that was just shifted in. Thus, states 00 and 01 can be entered only if a 0 is shifted in.

The encoded output sequence corresponding to a given input message sequence can be determined entirely from the trellis structure of figure 8. However, for convenience, this structure is sometimes expanded to the form shown in figure 9. The expanded trellis in figure 9 actually is an alternate way of viewing the code tree shown earlier in figure 6. Although only the first set of nodes is labeled on the expanded trellis structure,

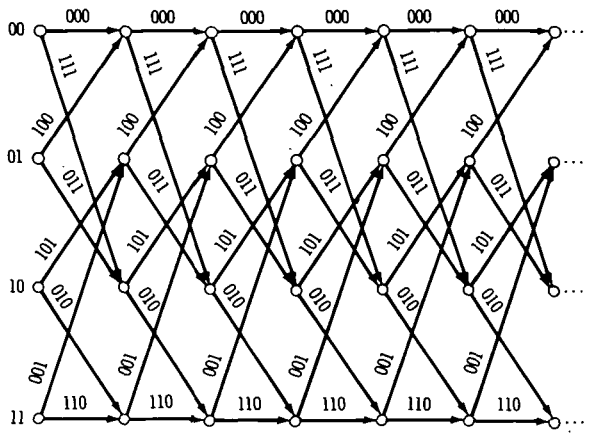


Figure 9. - Expanded version of the state diagram (trellis structure) for a (3, 3) convolutional encoder with generator sequence 111 101 100.

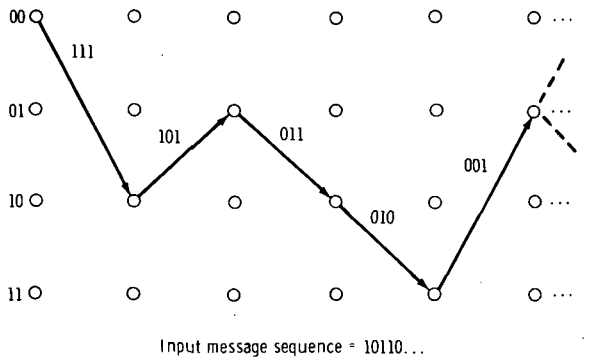


Figure 10. - A typical path through the expanded trellis structure for a (3, 3) convolutional encoder with generator sequence 111 101 100.

successive nodes at the same level correspond to the same state (i.e., the uppermost node always corresponds to state 00, etc.). The manner in which a path through the expanded trellis is determined for a typical input message sequence (10110...) is illustrated in figure 10. The resulting output sequence is 111 101 011 010 001....

The Viterbi decoding algorithm, which will be discussed in the next section of this report, is visualized best in terms of the expanded trellis structure. This algorithm basically attempts to find a path through the trellis that is as close as possible (differs in the fewest bit positions) to the received encoded sequence. The information sequence (input message sequence) corresponding to this path then is assumed to be the same as the original input message at the encoder. Given the received channel sequence, the algorithm is optimum in the sense that the most probable transmitted message is selected.

DESCRIPTION OF THE VITERBI DECODING ALGORITHM

From the state diagrams (including the trellis structures) shown previously in this report, it can be observed that once two paths are in the same state (by merging at a common node), both paths have identical extensions out of that state. Because this is the case, it can be observed that both paths subsequently will correlate equally well with the received sequence. Therefore, because the objective is to find the path that correlates best with the received sequence, it should be possible to eliminate one of the two paths that enter any state. By eliminating the smaller correlated path entering each state, only those paths that never can be candidates for the highest correlated path through the trellis structure are discarded. This is the general idea on which the Viterbi decoding algorithm is based.

The Viterbi algorithm was discussed first by A. J. Viterbi (ref. 4) and was shown to be asymptotically optimum. In a later paper (ref. 5), G. D. Forney observed that Viterbi's algorithm is optimum in the maximum-likelihood sense.

When implementing a Viterbi algorithm decoder, several hardware variations are possible. The particular implementation to be considered here maintains a running score, or branch metric, on each of the 2^{K-1} most likely paths through the trellis structure. The information sequences corresponding to these 2^{K-1} paths are called survivor sequences. At each step in the algorithm, the survivor sequences terminating in each of the 2^{K-1} nodes are determined. The scoring system is such that the most likely path through the trellis is the path having the lowest score.

The implementation of the Viterbi algorithm consists of the following steps.

1. All scores and survivor sequences initially are set to 0.

2. A received branch (V-bit segment) is correlated with each of the two possible branches out of each of the 2^{K-1} states, and delta scores (ΔS's) are generated. A ΔS is the number of bit positions in which the received branch differs from the branch with which it is being correlated. There are 2^K such correlations to be performed. For example, for the $K = 3$, $V = 3$ case that was considered previously in this report, it is necessary to correlate each received 3-bit branch with each of the $2^3 = 8$ possible 3-bit branches (000 to 111).

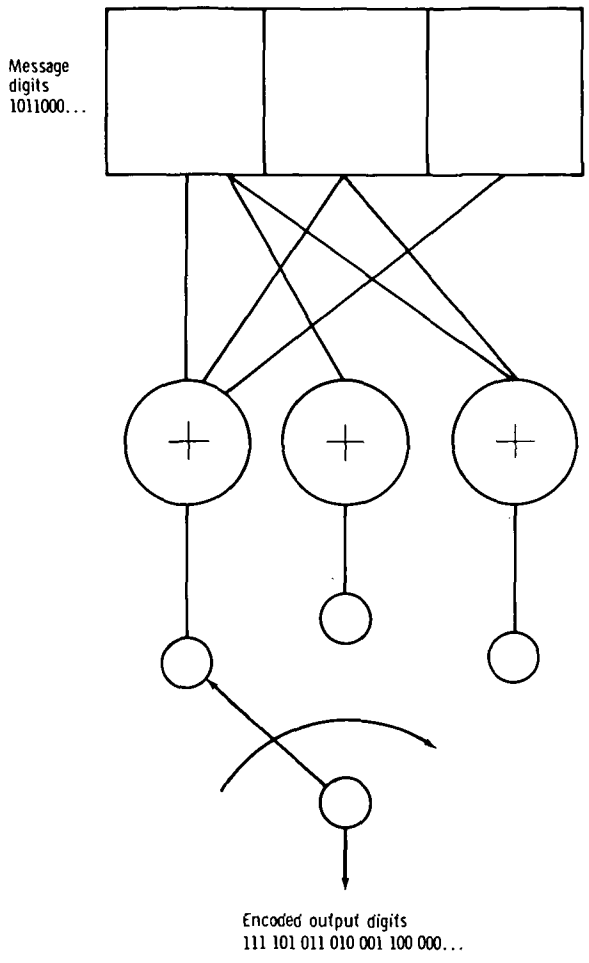
3. The ΔS's for the two paths leaving each state are added to the previous scores (initially 0) for that state (actually, for the path which previously terminated in that state).

4. Scores for the two paths terminating in each of the next 2^{K-1} states are compared. The path having the lowest score is retained, and the path having the highest score is dropped. (In case the two scores are equal, one of the paths is dropped arbitrarily.) Thus, 2^{K-1} running scores are retained.

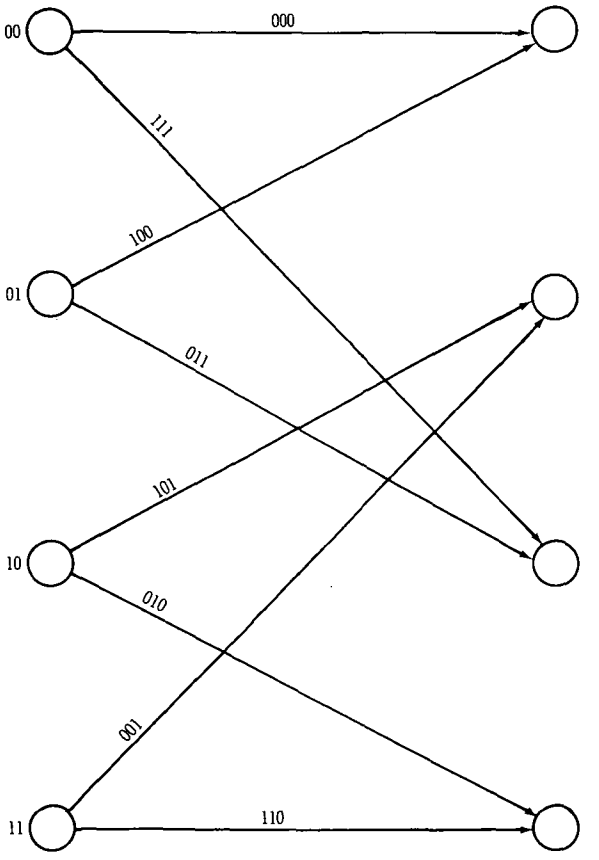
5. The survivor sequences for the paths terminating in each of the 2^{K-1} states then are stored, along with their running scores, and steps 2 to 5 are repeated. A set of 2^{K-1} registers is required to store the 2^{K-1} survivor sequences, and another set of 2^{K-1} registers is required to store the running scores for each of these survivor sequences. A straightforward scheme for storing the survivor sequences and scores is to provide a survivor sequence (SS) register and a score (SC) register for each state. That is, the SS_{00} and SC_{00} registers always will be used for storage of the survivor sequence (and its score) that terminates in state 00. Similarly, the SS_{01} and SC_{01} registers will be used for the sequence terminating in state 01 and so forth. This scheme requires that the capability exists for transfer of the entire contents of one register to another register. For example, if the survivor path that terminates in state 00 is the one from state 01, the new contents of register SS_{00} should be the contents previously stored in register SS_{01}, plus the newest bit (a 0 in this case) that resulted in state 00 being entered. Similarly, the new contents of register SC_{00} should be the contents previously stored in register SC_{01}, plus the ΔS for the branch between states 01 and 00.

Operation of the Viterbi algorithm can be visualized by means of an example. The operation for a noise-free received sequence (no bit errors), showing the encoder and its state diagram and illustrating the manner in which the survivor paths are traced through the trellis structure, is presented in figure 11. A step-by-step summary of the SS and SC register contents is contained in table I. Step 1 is merely an initialization step, during which all the SS and SC registers are set to 0.

At step 2, the first received branch (111) is correlated with the $2^3 = 8$ possible branches (000 to 111) shown in figure 11(b). The ΔS's that are generated are equal to the number of bit positions in which the received branch differs from the branch under consideration. Thus, the ΔS between the first received branch and the 000 branch is 3, the ΔS between the first received branch and the 111 branch is 0, and so forth. The two paths terminating in each of the four states then are compared; the path with the lowest score is retained, and the path with the highest score is dropped. For state 00, the 000 branch (which has a score of 3) is dropped in favor of the 100 branch (which has a score of 2). Likewise, the 101 branch

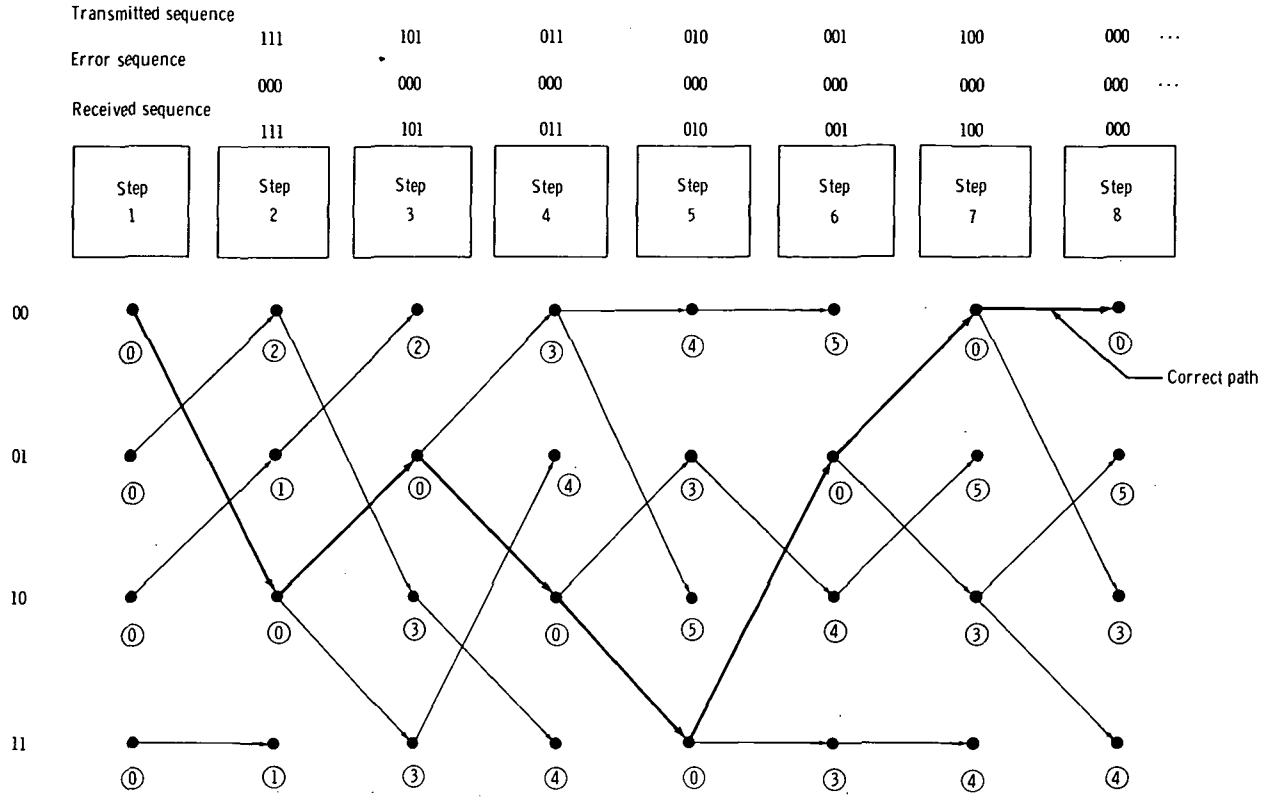


(a) Encoder.



(b) Encoder state diagram.

Figure 11. - An example of Viterbi algorithm operation (noise free).



(c) Tracing survivor paths through the trellis structure.

Figure 11. - Concluded.

is retained for state 01, the 111 branch for state 10, and the 110 branch for state 11. Scores for each of the surviving paths (only one branch in length at this stage) are shown encircled under the states in which the paths terminate. As shown in table I, these scores, plus the survivor sequences, are stored in the appropriate SC and SS registers. (Although the scores are indicated in decimal form for convenience, they actually would be stored in binary form.)

At step 3, the second received branch (101) is correlated with the eight possible branches to yield new ΔS's. The ΔS for each branch is added to the score for the state which that branch leaves, giving eight new scores. The two paths terminating in each state are compared, and the path with the lowest score is retained. For purposes of illustration, in case the two paths being compared have equal scores, the uppermost path is retained arbitrarily, and the lower one is rejected. Hence, of the two possible paths terminating in state 10, each having a score of 3, the path coming out of state 00 is selected. Similarly, the path coming out of state 10 is selected as the path terminating in state 11. Again, the storage of survivor sequences and scores at the end of step 3 is shown in table I. The most ancient bits in the SS registers are those on the left-hand sides. The contents of the SS_{00} register are obtained by shifting in the previous contents of the SS_{01} register, followed by a 0. Similarly, the SS_{01} register

TABLE I. - REGISTER CONTENTS (SS AND SC) FOR VITERBI ALGORITHM EXAMPLE (NOISE FREE)

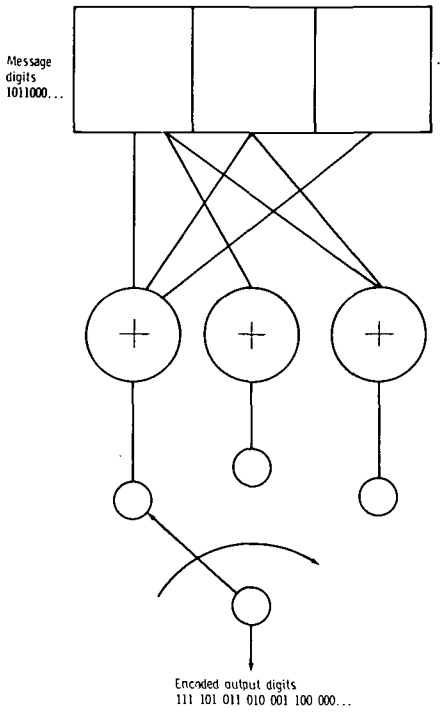
Step no.	Register contents							
	SS_{00}	SC_{00}	SS_{01}	SC_{01}	SS_{10}	SC_{10}	SS_{11}	SC_{11}
1	--	--	--	--	--	--	--	--
2	0	2	0	1	1	0	1	1
3	00	2	10	0	01	3	11	3
4	100	3	110	4	101	0	011	4
5	100 0	4	101 0	3	100 1	5	101 1	0
6	100 00	5	101 10	0	101 01	4	101 11	3
7	101 100	0	101 010	5	101 101	3	101 111	4
8	101 100 0	0	101 101 0	5	101 100 1	3	101 101 1	4
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

contents are obtained by shifting in the previous contents of the SS_{10} register, followed by a 0; the SS_{10} register contains the previous contents of the SS_{00} register, followed by a 1; and the SS_{11} register contains the previous contents of the SS_{10} register, followed by a 1.

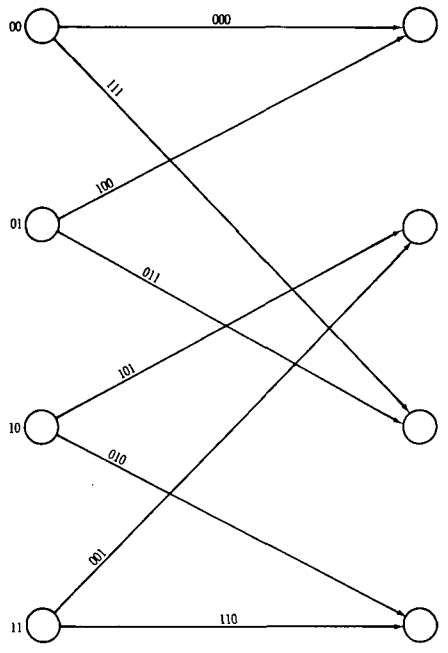
Operation of the algorithm for steps 4 to 8 of the current example is contained in figure 11 and in table I. The reader is encouraged to trace the various steps through in detail and to verify the tabulated results. The register interchange operations, in particular, should become much more obvious as this is done.

Another example of the Viterbi algorithm is provided in figure 12 and in table II. In this case, however, it is assumed that some of the received bits are in error. Again, the reader is encouraged to follow the procedure that was followed in the previous example and to verify the tabulated results of figure 12 and table II.

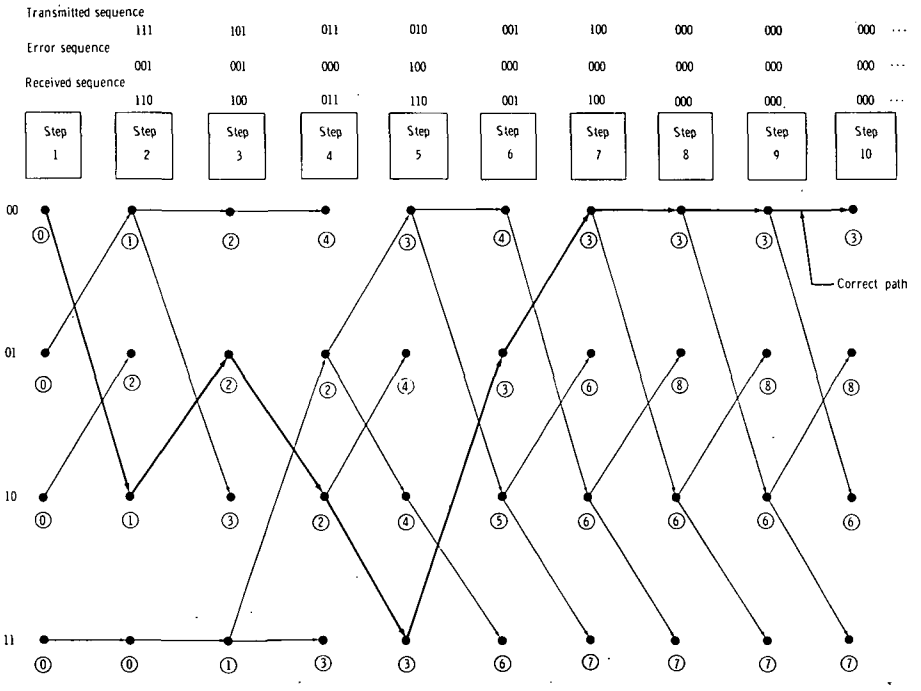
The basic operation of the Viterbi algorithm should be clear at this point. However, one very important question has not yet been answered: When is a bit decision made? In tables I and II, it is shown that as each successive branch is received, the lengths of the 2^{K-1} survivor sequences increase by 1 bit. It is necessary to make a decision on a bit after some finite length of time. Up to this point, all 2^{K-1} survivor sequences simply have been stored along with their associated scores. If the SS registers are made L bits long and if L is large enough, all 2^{K-1} SS registers eventually will agree on the initial bit in the survivor sequences. For the noise-free example (table I), all registers agreed on the first two bit positions after only five steps. For the example with errors summarized in table II, however, five steps were required for a unanimous decision on only the first bit. As will be indicated in the subsequent simulation results, if the SS register lengths are made equal to approximately five or six constraint lengths ($L \approx 5K$), all registers will, with high probability, agree on the initial few bits of the survivor sequences, even when the data are very noisy (i.e., there are many received errors).



(a) Encoder.



(b) Encoder state diagram.



(c) Tracing survivor paths through the trellis structure.

Figure 12. - An example of Viterbi algorithm operation (with errors).

TABLE II. - REGISTER CONTENTS (SS AND SC) FOR VITERBI ALGORITHM EXAMPLE (WITH ERRORS)

Step no.	Register contents							
	SS_{00}	SC_{00}	SS_{01}	SC_{01}	SS_{10}	SC_{10}	SS_{11}	SC_{11}
1	--	--	--	--	--	--	--	--
2	0	1	0	2	1	1	1	0
3	00	2	10	2	01	3	11	1
4	000	4	110	2	101	2	111	3
5	110 0	3	101 0	4	110 1	4	101 1	3
6	110 00	4	101 10	3	110 01	5	110 11	6
7	101 100	3	110 010	6	110 001	6	110 011	7
8	101 100 0	3	110 001 0	8	101 100 1	6	110 001 1	7
9	101 100 00	3	101 100 10	8	101 100 01	6	101 100 11	7
10	101 100 000	3	101 100 010	8	101 100 001	6	101 100 011	7
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

Thus, a bit decision can be made after an L-bit delay in which L is of the order of five or six constraint lengths. To allow for the unlikely event in which all SS registers do not agree on the oldest bit in storage, the decoding decision rule will be to choose the oldest bit of the survivor sequence having the lowest score. A decoded bit can be shifted out of one end of the L-bit register as a new bit is shifted into the decoder. Thus, except for an L-bit delay, decoding is performed in real time, on a bit-by-bit basis.

The oldest bits contained in the SS registers of tables I and II (after several steps) do indeed correspond to the original message input bits at the encoder. The second example (fig. 12 and table II) not only illustrates operation of the Viterbi algorithm, but demonstrates the error correction capability of a convolutional code that is decoded by that algorithm.

An important decoder design parameter is the required survivor sequence register length L, sometimes referred to as the decoder search length or block length. As discussed in the introduction, the performance gain over no coding that is achievable by using Viterbi algorithm decoding is dependent heavily on code constraint length K. Other parameters that affect the achievable performance gain, but do not impact the required decoder hardware as severely as constraint length, are code rate R and number of receiver quantization levels Q. The quantization process involves representing the integrated signal level (from the receiver demodulator) as one of a finite number (Q) of possible levels. In general, two quantization schemes, known as "hard" and "soft" decisions, are used. The term "hard decisions" (Q = 2) implies that a threshold (usually 0 volt) is set and that the input bit to the decoder is recognized as 0 or 1, depending on whether the integrated signal level is above or below the threshold. A disadvantage of hard-decision schemes is that all bit decisions are weighted equally, regardless of the relative proximity to threshold of the various integrated signal levels. This disadvantage is overcome by soft-decision schemes, which incorporate multiple thresholds. One common soft-decision scheme uses eight threshold levels (Q = 8). In general, $\log_2 Q$ bits are required to indicate to the decoder the relative magnitude of each channel bit.

In addition, Viterbi decoder operation is code-dependent. For example, it has been shown (ref. 7) that better performance is obtainable if nonsystematic codes are used rather than systematic codes. The effects of these various parameters and of various code types on the Viterbi decoder performance are reported in the following sections of this paper.

SIMULATION DATA AND PERFORMANCE PREDICTIONS

Parametric studies of the Viterbi decoding algorithm were performed by using digital computer (Univac 1108) simulations. Details of the FORTRAN IV computer program are outlined in appendix A, and a complete program is listed in appendix B. The program originally used an all-zero information bit sequence (without loss of generality) and selected the upper of the two paths leading to any node in the trellis diagram when the two scores are equal. By consistently making decisions favoring the upper paths (which correspond to 0 information bits), the results were somewhat biased on the optimistic side. The magnitude of this bias was determined later to be almost insignificant (0.15 decibel) when the program was modified (1) to make a random path selection when the scores for the two paths leading into a node are equal and (2) to accept a random sequence of data bits and to make random path selections.

Details on the various codes that were simulated are tabulated in appendix C. The codes used are the "good" nonsystematic codes reported in references 8 and 9 and the good systematic codes reported in reference 10. The data presented in appendix C include the rate, the constraint length, the generator sequence, the generator coefficients, and the shift register representation of each code used in the computer simulations.

Appendix D contains the "raw" data obtained from the Univac 1108 computer simulations. The data tabulated in appendix D for each code include the number of information bits used for each computer run, the value of E_b/N_0 (energy per information bit per noise spectral density) and the resulting channel error rate at the decoder input, the number of output (information bit) errors, and the output information bit error rate. All data shown on the error probability plots presented later in this section were derived from appendix D.

Effects of Noise on Predicted Performance

Some early simulation runs produced results that were found to be approximately 2 decibels on the pessimistic side (at 1×10^{-4} bit error probability), when compared to theoretical bounds predicted by Viterbi (ref. 11). These results were obtained by using a random generator (RAND 4) that is based on Lehmer's method. The program is listed in appendix B. As indicated in the program listing, the noise is generated in two phases. Phase I involves the generation of uniformly distributed random numbers. In phase II, these random numbers are translated into Gaussian noise in the dimensioned array QUANT. A subsequent study of various noise generation techniques disclosed that, of all the tests performed on the uniformity of the random numbers, the dimensionality V (where 1/V is the code rate) of the serial and run correlation was very important for this application. Because simulation runs using the Ran-10 program, which was tested in particular for the correlation criterion, agreed much more closely with the theoretical bounds, the Ran-10 generator was used for the simulation data presented in this report. The Ran-10 noise is generated by the power residue method (ref. 12) and has an initial value of 373620336005 octal and a multiplier value of 1045 octal.

Effects of Code Rate on Decoder Performance

The bit error probability curves shown in figure 13, which were derived directly from the simulation data presented in appendix D, indicate that for a constant constraint length, the performance improvement as a result of coding is greater for rate one-third

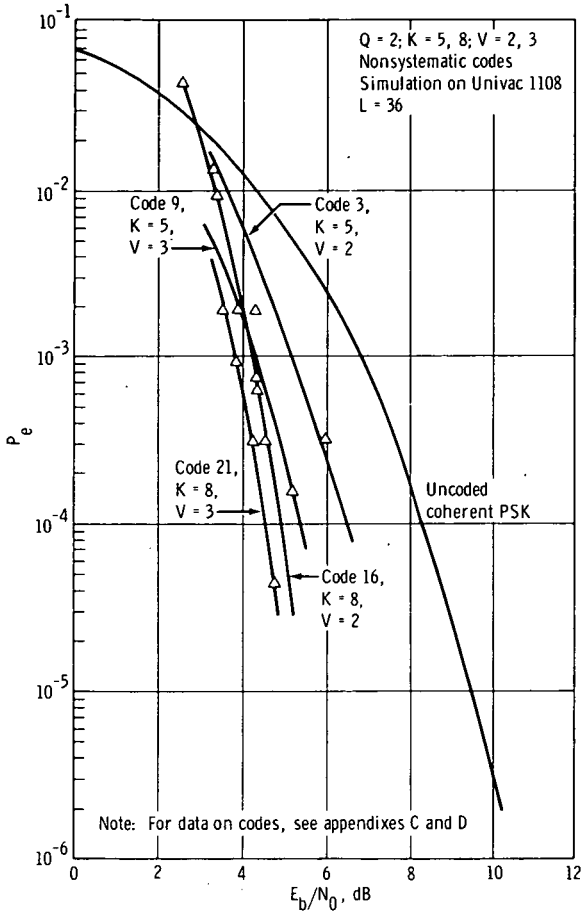


Figure 13. - Effects of code rate on Viterbi decoder performance.

than for rate one-half. For K = 5, this increase is approximately 1.1 decibels, whereas for K = 8, the increase is only approximately 0.4 decibel (for hard decision) at 1×10^{-4} bit error probability. Note that for a given rate and constraint length, the absolute performance gain (over no coding) decreases with increasing error probability and that as the error probability approaches 1×10^{-1}, the coding gain approaches 0 or even becomes negative.

Effects of Receiver Quantization on Decoder Performance

Figure 14 was obtained by replotting two of the hard-decision (Q = 2) curves of figure 13 and by adding a corresponding set of curves for soft decisions (Q = 8). The soft-decision curves were obtained for the same set of codes as the hard-decision curves. Figure 14 is indicative that an additional performance improvement of approximately 1.6 to 1.9 decibels at a bit error probability of 1×10^{-4} can be realized by using soft rather than hard decisions at the decoder input.

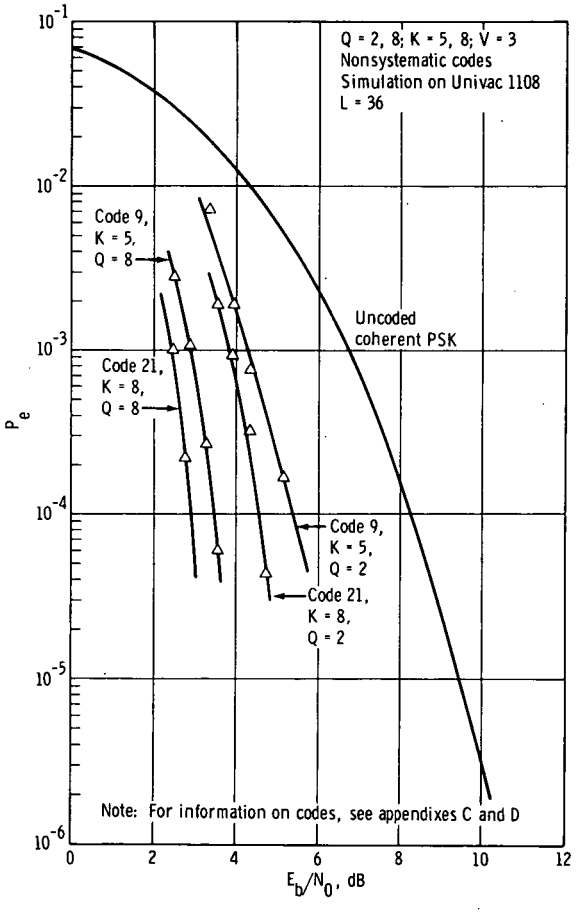
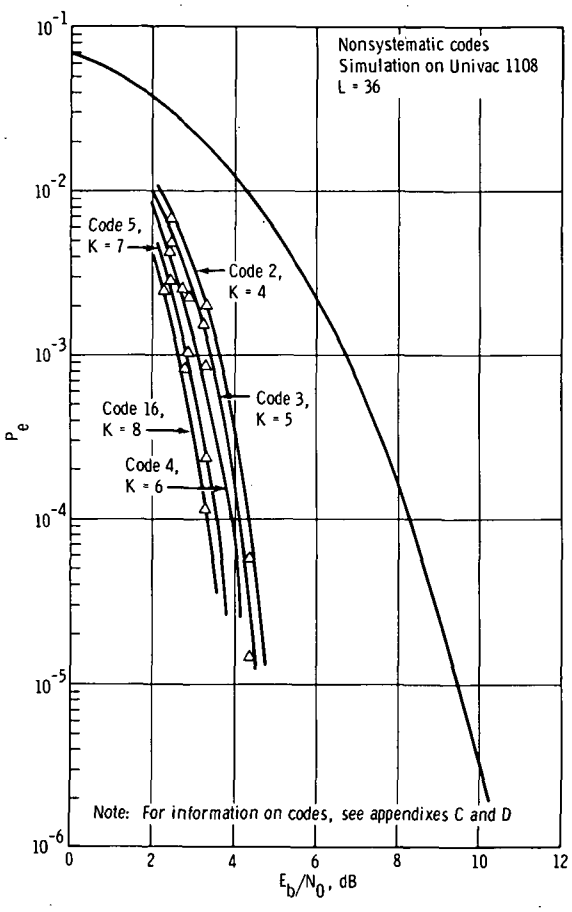


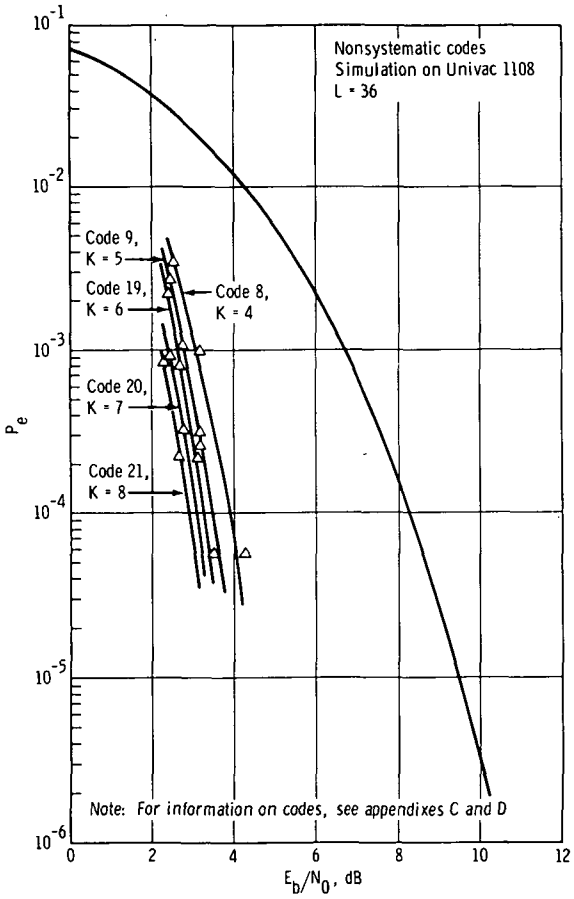
Figure 14. - Effects of receiver quantization levels (Q) on Viterbi decoder performance.

Effects of Code Constraint Length on Decoder Performance

Although it can be seen from figures 13 and 14 that the performance improvement obtainable by using Viterbi algorithm decoding is greater for $K = 8$ than for $K = 5$, it is of interest to explore this variation in somewhat greater detail. Performance curves for $Q = 8$, $V = 2$, and $K = 4$ to $K = 8$ are shown in figure 15(a), and the corresponding curves for $V = 3$ are shown in figure 15(b). These figures are indicative that the performance gain increases with each increase in K. As mentioned previously, however, the hardware complexity of the decoder is highly dependent on K; consequently, a practical upper limit on K is approximately 8. Figures 15(a) and 15(b) are indicative that the increase in performance gain between $K = 4$ and $K = 8$ is approximately 1.0 decibel for either $V = 2$ or $V = 3$ (for soft decision) at 1×10^{-4} bit error probability.

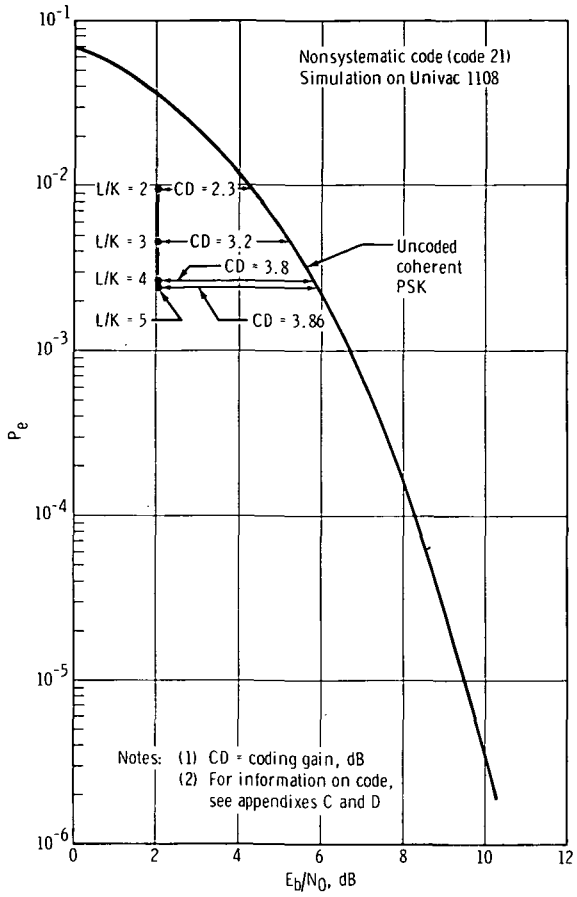


(a) $Q = 8$, $V = 2$.

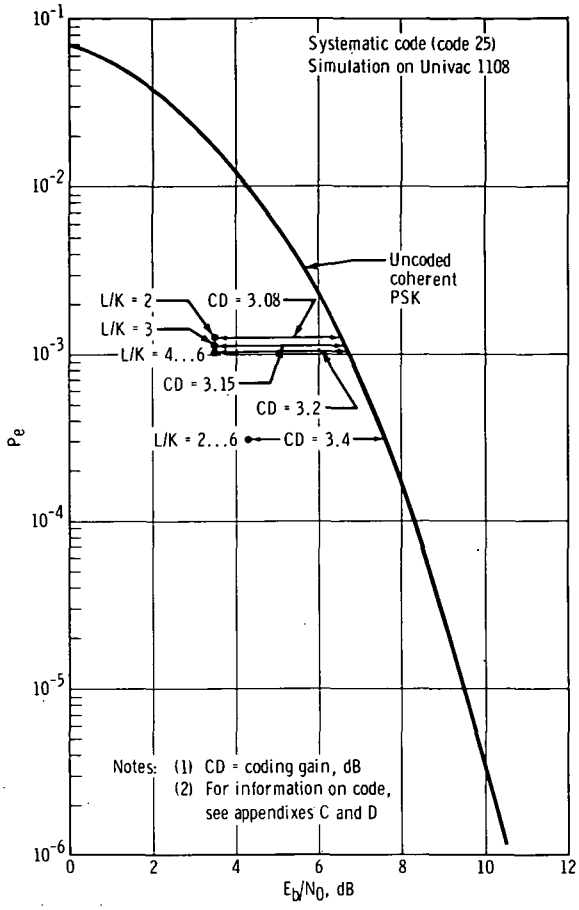


(b) $Q = 8$, $V = 3$.

Figure 15. - Effects of code constraint length on Viterbi decoder performance.



(c) K = 8, V = 3, Q = 8.



(d) K = 5, V = 3, Q = 8 (systematic code).

Figure 17. - Concluded.

CONCLUDING REMARKS

The coding gains possible (at a bit error probability of 1×10^{-4}) by using Viterbi algorithm decoding are summarized in table III. These coding gains were determined directly from figures 13 and 15 and represent gains achievable by using the nonsystematic codes.

TABLE III. - PERFORMANCE GAINS ACHIEVABLE BY USING VITERBI ALGORITHM DECODING (BEST CODES)

K	V	Q	Approximate gain achieved at $P_e = 1 \times 10^{-4}$ as compared to uncoded PSK,[a] dB	Reference
5	2	2	1.90	Figure 13
8	2	2	2.90	
5	3	2	3.40	
8	3	2	3.70	
4	2	8	3.90	Figure 15(a)
5	2	8	4.10	
6	2	8	4.35	
7	2	8	4.65	
8	2	8	4.95	
4	3	8	4.35	Figure 15(b)
5	3	8	4.80	
6	3	8	5.00	
7	3	8	5.20	
8	3	8	5.35	

[a]Phase-shift keying.

The simulation results and performance predictions presented in this report can be used to establish certain design goals for Viterbi algorithm decoders and for communications systems employing convolutional encoding and Viterbi decoding. It was determined that higher performance gains (over no coding) are obtainable by using nonsystematic codes, lower code rates, and longer code constraint lengths, and by incorporating a soft-decision capability at the receiver. Furthermore, it was found that a decoder search length of four times the constraint length is sufficient for nonsystematic codes, whereas a search length of two times the constraint length was sufficient for the systematic code considered.

The relatively large coding gains realizable, together with the hardware advantages discussed in this report, make the Viterbi decoding algorithm especially attractive for incorporation into many practical communications systems. Viterbi decoding appears to be particularly well suited for the relatively high data rate systems characteristic of manned space flight.

Manned Spacecraft Center
National Aeronautics and Space Administration
Houston, Texas, July 26, 1972
914-50-50-17-72

REFERENCES

1. Jacobs, Irwin Mark: Sequential Decoding for Efficient Communication From Deep Space. IEEE Transactions on Communication Technology, vol. COM-15, no. 4, Aug. 1967, pp. 492-501.

2. Fano, Robert M.: A Heuristic Discussion of Probabilistic Decoding. IEEE Transactions on Information Theory, vol. IT-9, no. 2, Apr. 1963, pp. 64-74.

3. Gallagher, Robert G.: Information Theory and Reliable Communication. John Wiley and Sons, Inc., 1968.

4. Viterbi, Andrew J.: Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. IEEE Transactions on Information Theory, vol. IT-13, no. 2, Apr. 1967, pp. 260-269.

5. Anon.: Coding System Design for Advanced Solar Missions, Final Report. NASA CR-73176, Dec. 1967.

6. Schwartz, Mischa; Bennett, William R.; and Stein, Seymour: Communication Systems and Techniques. McGraw-Hill, 1966.

7. Bucher, Edward A.; and Heller, Jerrold A.: Error Probability Bounds for Systematic Convolutional Codes. IEEE Transactions on Information Theory, vol. IT-16, no. 2, Mar. 1970, pp. 219-224.

8. Viterbi, A. J.; Odenwalder, J. P.; Rosenberg, W. J.; and Zeoli, G. F.: Concatenation of Convolutional and Block Codes, Final Report. NASA CR-114358, June 1971.

9. Anon.: Potential Applications of Digital Techniques to Apollo Unified S-Band Communications Systems, Final Report. NASA CR-108300, Nov. 1970.

10. Lin, S.; and Lyne, H.: Some Results on Binary Convolutional Code Generators. IEEE Transactions on Information Theory, vol. IT-13, no. 1, Jan. 1967, pp. 134-139.

11. Viterbi, A. J.: Convolutional Codes and Their Performance in Communication Systems. Linkabit Corporation Seminar, Jan. 26-29, 1970, Los Angeles, California.

12. Hull, T. E.; and Dobell, A. R.: Random Number Generators. SIAM Review, vol. 4, no. 3, July 1962, pp. 230-254.

APPENDIX A

PROGRAM DEFINITION FOR COMPUTER SIMULATION OF THE VITERBI DECODING ALGORITHM

In this appendix, the digital computer program developed to simulate a communications channel employing convolutional encoding and Viterbi algorithm decoding is described. The program is based on the requirements of the Univac 1108 computer and has been made sufficiently flexible to accept variations in the encoder and decoder parameters. With the existing program capability, the bit error probability at the decoder output can be determined for any input signal-to-noise ratio and for any set of encoder and decoder parameters.

PROGRAM INPUT REQUIREMENTS

Input Data

The program has provisions for the following inputs.

Number of passes. - The number of passes (PASS) must be supplied as data. Each PASS processes 324 information bits. This number is nine times the word length (36 bits) for the machine and is selected for output considerations.

Number of adders. - The number of encoder modulo 2 adders ($1 \leq V \leq 3$) must be supplied as data (RATE).

Constraint length. - The constraint length K must be supplied as data (CL).

Encoder hookup connections. - Encoder hookup connections, known as generator coefficients (HOOKUP(1) to HOOKUP (RATE)), must be supplied as data.

Quantization threshold levels. - The quantization threshold levels for the soft-decision logic at the decoder (QUANT(1) to QUANT(8)) must be supplied as data.

Crossover probability. - The binary symmetric channel crossover probability (P) must be supplied for hard decisions.

Data Card Formats

The data card formats are as follows.

Data card 1. - Data card 1 contains the following information.

PASS: PASS is contained in columns 1 to 5. For example: PASS = 20; column 4 = 2, column 5 = 0.

RATE: RATE is contained in columns 6 to 10. For example: RATE = 2; column 10 = 2.

CL: CL is contained in columns 11 to 15. For example: CL = 8; column 15 = 8.

Data card 2. - Data card 2 contains the following information.

HOOKUP(1): HOOKUP(1) is contained in columns 1 to 5 in octal. For example: HOOKUP(1) = $1110110|_2$ binary = $166|_8$ octal; column 3 = 1, column 4 = 6, column 5 = 6.

HOOKUP(2): HOOKUP(2) is contained in columns 6 to 10 in octal.

HOOKUP(3): HOOKUP(3) is contained in columns 11 to 16 in octal.

Data card 3. - Data card 3 contains the following information.

QUANT(1) to QUANT(8): QUANT(1) to QUANT(8) are contained in columns 1 to 5, 6 to 10, 11 to 15, 16 to 20, 21 to 25, 26 to 30, 31 to 35, and 36 to 40 in decimal. For example: QUANT(4) = 741; column 18 = 7, column 19 = 4, column 20 = 1.

P: P is contained in columns 41 to 45. For example: P = 1.0 percent = 0.01; column 43 = ., column 44 = 0, column 45 = 1.

In addition to data cards 1 to 3, the program requires external noise subroutines RAN and ZOR, which generate sets of uniformly distributed random numbers between 0 and 1.0.

PROGRAM OUTPUT DETAILS

Output Data

For the supplied data input discussed previously, the program provides the following outputs.

1. The encoder hookup connections (generator coefficients) used in generating the convolutional code (in octal)

2. The quantization threshold levels corresponding to a given information bit signal-to-noise ratio or bit error rate

3. A complete printout of the quantized channel bits (three channel bits correspond to one information bit) for each PASS (Each PASS processes 324 information bits.)

4. A complete printout of the decoder output (information bits) for the all-zero input and for each PASS (This output is in octal; that is, if, for example, the decoded message is 010001000..., the printout will be 210.)

5. The number of output decoding errors in each PASS and the accumulated error rate

After all required bits are processed, the following information is printed out.

1. The message length (number of information bits processed)

2. The encoded message length

3. The total number of decoding errors

4. The probability of decoding errors (information bits) in percent.

The input error rate that corresponds to the quantization threshold levels supplied also is printed out.

A Sample Output

For a K = 5, V = 2 convolutional code for which the necessary parameters are supplied to the program as data, the sample output will be as follows.

```
SOFT DECISION 3.0 PERCENT FOR (5,2)
HOOKUP = 00035 00023 00000
QUANT = 432 630 797 908 966 990 998 1000

ERROR RATE                  XX
ERRORS THIS PASS            XX
INPUT CHANNEL BITS          XX
DECODED INFORMATION BITS    XX

The total number of outputs of this type is equal to PASS.

SUMMARY OF RESULTS
   Noise quantization level corresponding to 3.0 percent (information)
   MESSAGE LENGTH                                  133488
   ENCODED MESSAGE LENGTH                          266976
   TOTAL NUMBER OF DECODING ERRORS                    672
   PROBABILITY OF DECODING ERRORS IN                .503
      PERCENT
```

FUNCTIONAL DETAILS

Definitions and Variables

The complete computer program, written in FORTRAN IV for the Univac 1108 machine, is listed in appendix B. Comment cards are inserted at appropriate places to make the program self-explanatory. However, the following functional definitions and notations are provided for the convenience of the reader.

PASS. - PASS is the integer value of multiples of 324 information bits to be processed.

RATE. - RATE is the number of modulo 2 adders of the convolutional encoder.

CL. - CL is the constraint length of the code.

HOOKUP. - HOOKUP represents connections (octal representation) of modulo 2 adders to the stages of the convolutional encoder shift register (fig. A-1).

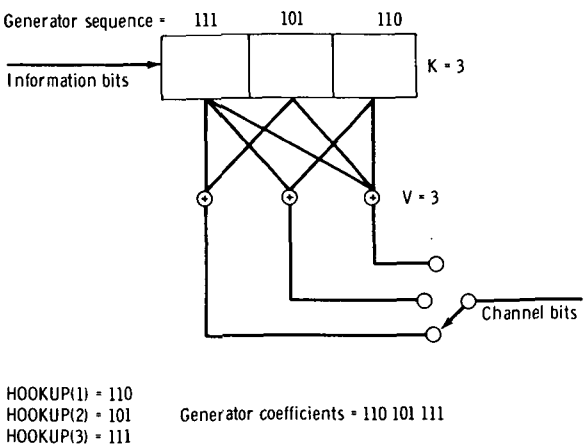


Figure A-1. - Convolutional encoder example.

QUANT. - QUANT represents the quantization threshold levels corresponding to a given bit error rate.

REG, SREG. - REG and SREG are equivalent variable names that represent a maximum of 1×2^8 dimensions or a maximum of 1×2^4 storage dimensions used at a time of processing a single information bit. The dimension 1×2^8 corresponds to a value of CL = 8 and can be changed accordingly. The two 1×2^4 dimensions are used alternatively for two consecutive information bits. REG is a 36-bit sequence of 0 and 1 storing the decoded information bits for a block length of 36 bits. For a given CL (= K), the dimension corresponds to twice the number of nodes in the trellis diagram.

SCORE. - SCORE has a maximum of 1×2^8 dimensions. As in the case of REG, a maximum of half its dimension is used alternately for two consecutive information bits in calculating the score in the transition from one state to another (moving to a node in the trellis diagram). A maximum of 1×2^4 dimensions (corresponding to the total number of nodes) is used to store the updated score for the corresponding paths for deciding on the best estimate of the input bit.

ENCODE. - ENCODE is the channel bit input to the decoder. This input simulates the noise added to the channel bits corresponding to all-zero information bits. ENCODE has a dimension of 6 by 324, half of which is used for each alternate PASS.

TABLE. - For a given hookup connection, TABLE represents the output of the encoder for a state transition. The TABLE values for the example are: TABLE(1) to TABLE(8) = 000, 011, 101, 110, 111, 100, 010, and 001, respectively. The first 2^{K-1} dimension of the TABLE refers to the outputs corresponding to a 0 information bit input, whereas from $2^{K-1} + 1$ to 2^{K} correspond to a 1 information bit input. To the previously stored scores, the difference of the received segment and the TABLE values leading to a node are added, and the lower of the two scores and the corresponding path are stored.

AMASK(1) to AMASK(36). - AMASK is a 36-bit integer value such that

AMASK(1) = 100000000000000000000000000000000000

.
.
.

AMASK(12) = 000000000001000000000000000000000000

.
.
.

The dimension refers to the position of 1 in the all-zero sequence.

OMASK(1) to OMASK(36). - OMASK is a 36-bit integer value such that

OMASK(1) = 011111111111111111111111111111111111

.
.
.

OMASK(12) = 111111111101111111111111111111111111

.
.
.

The dimension refers to the position of 0 in the all-one sequence.

BUFFER. - BUFFER is a 36-bit decoded message in octal representation.

OUTPUT. - OUTPUT is the decoded information bit corresponding to the path with the smallest score.

FIRST, SEC. - The FIRST and SEC pointers are set alternately to 0 and 128 (corresponding to the maximum 2^{K-1}). While processing odd-numbered information bits,

FIRST is set to 0 and SEC is set to 128; whereas for even information bits, the settings are reversed. This interchange of FIRST and SEC values helps in keeping the SREG and SCORE values.

TOP, BOT. - The TOP and BOT pointers are set alternately to 0 and 324. While processing odd-numbered blocks of 324 bits (1, 3, 5..., corresponding to running PASS values of 36, 34, 32...), TOP is set to 0 and BOT is set to 324; whereas for the even-numbered blocks, the settings are reversed.

TOPP, BOTP. - The TOPP and BOTP pointers are set alternately to 0 and 324 after every output printout corresponding to a PASS (324 information bits).

INDEX. - INDEX is the counter for the output. Its value is incremented by 1 for each information bit processed after the first 36 bits. The output is printed out when INDEX = 324, then INDEX is reset to 0. This means that there is no output until 324 bits (one PASS) plus 36 bits are processed, after which the first 324 bits are printed out. Subsequently, INDEX gives 324 bits output for every 324 bits processed.

ERROR, ACCUM. - The ERROR counter counts the number of errors in each PASS, and ACCUM is the accumulated updated number of errors.

A. - The A is the decoded information bit error rate in percent.

SR, LB. - The SR and LB pointers are used to set the information bits in the SREG and also to calculate the error corresponding to the path with minimum score.

H. - The H is the node pointer for the path with the smallest score.

Block Definitions

Although the program is not grouped into definite, discrete sections, it may be broken into the following blocks for the purpose of functional details.

Block A. - Block A defines the dimensional statement and reads and writes the data input.

Block B. - Block B defines the constants and calculates the OMASK and the AMASK.

Block C. - Block C calculates the 2^K $(= 2^{CL})$ TABLE values, which are the encoder output for all possible state transitions.

Block D. - Block D generates random numbers uniformly distributed between 0 and 1.0 by calling an external subroutine RAND. If soft decisions are used, block D generates values of ENCODE, the soft-decision channel bits corresponding to a required bit error rate.

Block E. - Block E processes 324 bits for each PASS. The processing involves comparing the 2^{K-1} pairs of TABLE values with the corresponding ENCODE values,

updating the REG and SCORE values for the survival paths, calculating the decoding error, and packing the decoded output in BUFFER. The functions of five subblocks in block E can be explained as follows.

Block E_1: Block E_1 compares the two paths leading to a node (e.g., TABLE(1) and TABLE(2) leading to node 00) with the ENCODE and updates the SCORE values.

Block E_2: Block E_2 updates the INDEX values for printout.

Block E_3: Block E_3 packs the decoded output into BUFFER and calculates and prints out the updated error rate, the number of errors, the ENCODE values, and the BUFFER for every 324 bits processed.

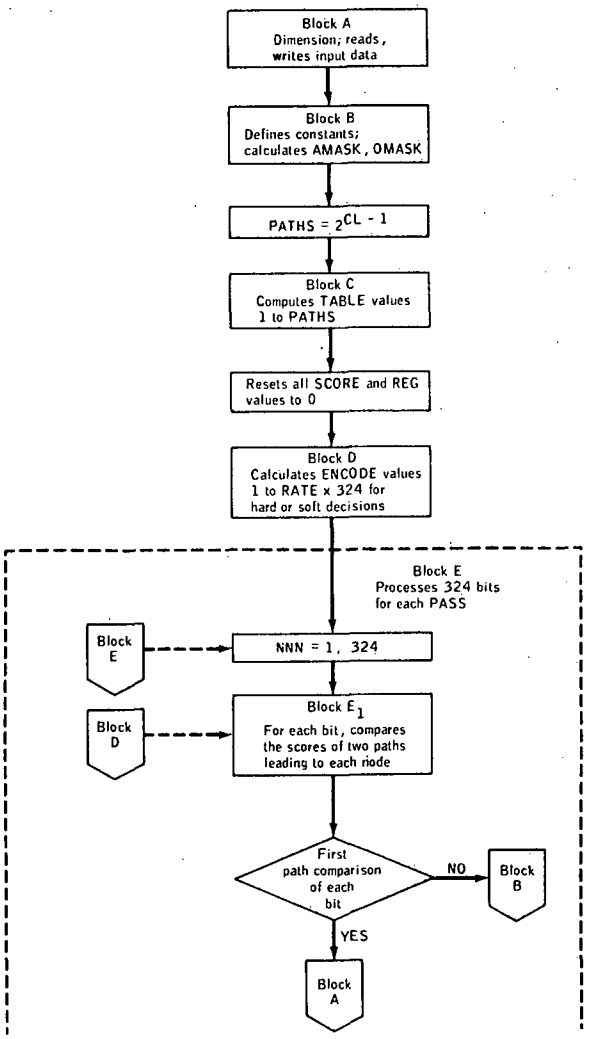
Block E_4: Block E_4 sets the SR and LB pointers.

Block E_5: Block E_5 updates the SREG and SCORE values for each bit processed.

Block E_6: Block E_6 resets FIRST and SEC values after each bit is processed and sets pointer H for the path with minimum score to be used in block E_2.

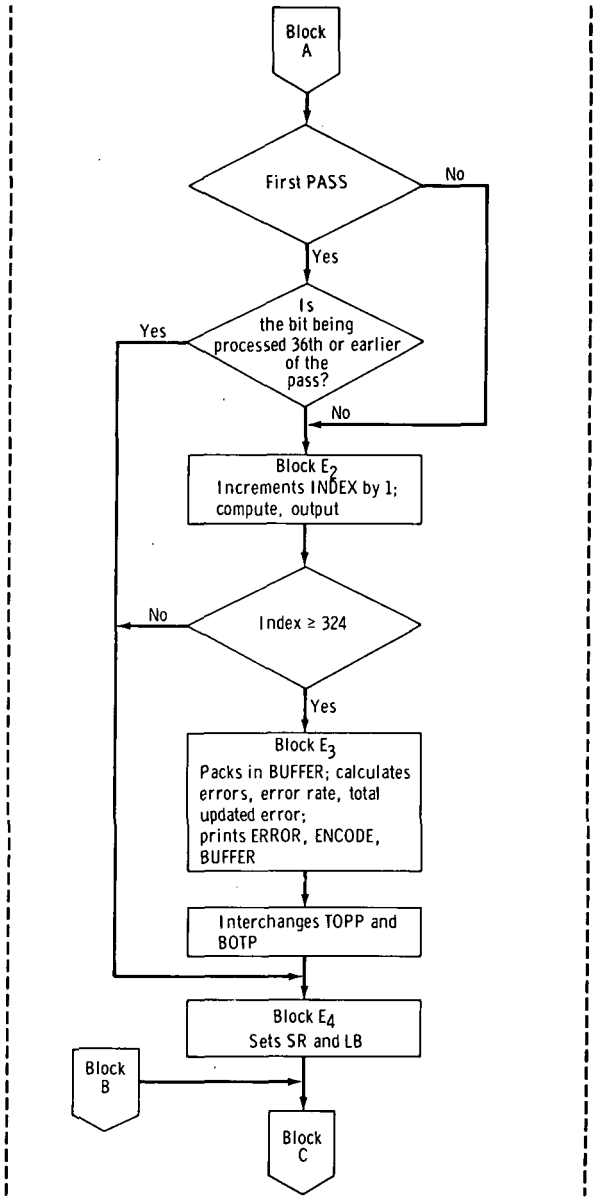
Block F. - Block F resets the TOP and BOT pointers after each PASS, checks that the bits corresponding to every PASS have been processed, computes the final error rate, and prints out the summary of the simulation results.

Figure A-2 is a flow chart for computer simulation of the Viterbi decoding algorithm.

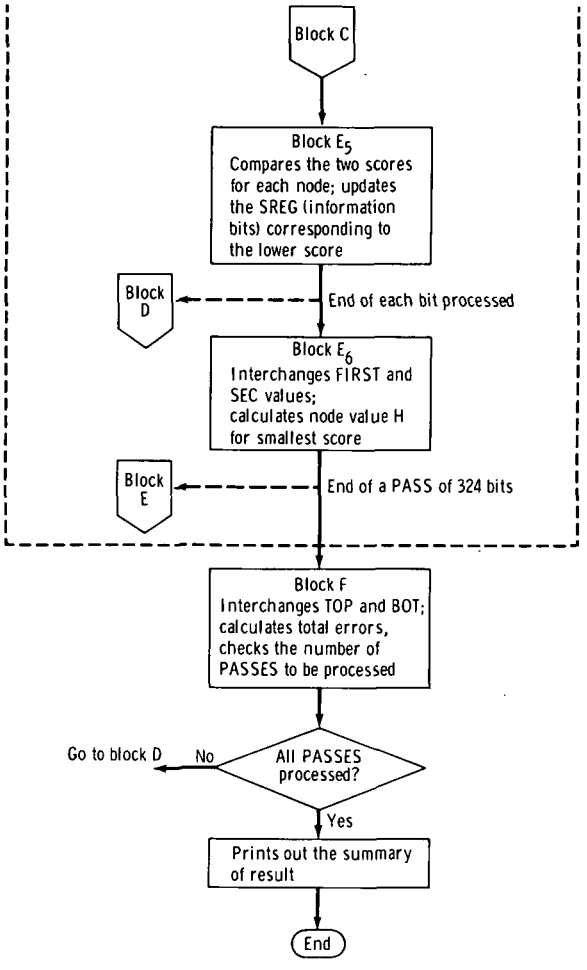


(a) Start.

Figure A-2. - Flow chart for computer simulation of the Viterbi decoding algorithm.



(b) Continuation.



(c) End.

Figure A-2. - Concluded.

APPENDIX B

LISTING OF VITERBI DECODER SIMULATION PROGRAM

```
      C      VITERBI DECODER ALGORITHM COMPUTER PROGRAM LISTING ON UNIVAC 1108
      C
      C      BEGINNING OF BLOCK A, DEFINES DIMENSIONAL STATEMENTS
      C      READS AND WRITES THE DATA INPUT
            IMPLICIT INTEGER(A-Z)
            REAL SPRED,FACC,FINIT,FPASS,A,AK,RANDA
            REAL PDCE,P
            REAL RNUM,RANDOM,ZOR
            ABNORMAL RANDOM,ZOR
            DIMENSION MSG(18),OUTERR(9)
            DIMENSION AMASK(36),OMASK(36),QUANT(8),REG(256,2)
           1, OUTPUT(648),SCORE(256),BUFFER(9),TABLE(256),HOOKUP(10),
           2 ENCODE(5843),SREG(256,2)
            EQUIVALENCE (REG,SREG)
            DATA OMASK(1)/04000000000000/,AMASK(1)/037777777777/
            DATA MSTART/010405/,SSTART/037362033600S/
      CTHIS PROGRAM IS MACHINE INDEPENDENT EXCEPT FOR WORD LENGTH CHANGES
      CWHICH DO AFFECT THE SIMULATED SHIFT REGISTER LENGTHS AND FORMATION
      CDATA INPUT
      C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            READ(5,630) INIT,KATE,CL
            READ(5,631)(HOOKUP(I),I=1,RATE)
            WRITE(6,640)(HOOKUP(I),I=1,RATE)
            READ(5,630)W1,W2
      1     READ(5,630,END=750)(QUANT(I),I=1,8),P
            WRITE(6,641)(QUANT(I),I=1,8)
            PASS = INIT
            W3=W1+W2
            PRINT 650,W3,W1,W2
      650   FORMAT(1HO,7X,15HPATHS OF LENGTH,13,25H BITS ARE BEING USED WITH,
           1 13,18H BITS IN WORD1 AND,13,15H BITS IN WORD2,)
      C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      CCONSTANT DEFINITION
      C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      C      BEGINNING OF BLOCK B, CALCULATES THE
      C      CONSTANTS, OMASK AND AMASK
            ACCUM=0
            FIRST=0
            SEC=128
            TOP=0
            BOT=324
            TOPP=0
            BOTP=324
            ERRTOT=0
            INDEX=0
            OMASK(36)=1
            DO 10 J=1,34
            K=36-J
      10    OMASK(K)=2*OMASK(K+1)
            DO 30 J=2,36
      30    AMASK(J)=OR((AMASK(J-1)-OMASK(J)),OMASK(J-1))
            GMASK=7
      C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      CCALCULATION OF CONTROL CONSTANTS
      C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
            PATHS=2**(CL-1)
      C      * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      C      END OF BLOCK B
      C      ************************************************************
      CBUILDING OF NODE PATH TABLE
      C      BEGINNING OF BLOCK C, CALCULATES THE
      C      TABLE VALUES FOR STATE TRANSITIONS
```

```
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
      KK=2*PATHS
      DO 90 J=1,KK
      M=0
      JJ = ABS(J-1)
      DO 80 N=1,RATE
      K=AND(JJ,HOOKUP(N))
      KOUNT=0
      DO 50 L=1,36
      KADD=AND(K,OMASK(L))
      IF (KADD) 50,50,40
40    KOUNT=XOR(KOUNT ,1)
50    CONTINUE
      K=KOUNT
      LL=36
      M=M*8
      DO 70 MM=1,3
      IF (K) 70,70,60
60    M = OR(M,OMASK(LL))
70    LL=LL-1
80    CONTINUE
90    TABLE(J)=M
C
C     END OF BLOCK C
C
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C RANDOM BIT GENERATION AND QUANTIZATION
C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
C     BEGINNING OF BLOCK D, THIS BLOCK IS TO
C     BE CHANGED WHEN CHANGING THE
C     SIMULATION FROM HARD DECISION TO SOFT
C     DECISION OR VICE-VERSA, BLOCK GENERATES THE INPUT
C     SEQUENCE TO THE DECODER WHICH IS THE ENCODED MESSAGE
C     CORRESPONDING TO RANDOM INPUT PLUS NOISE
      DO 100 J=1,PATHS
      REG(J,1)=0
      REG(J,2)=0
      REG(J+128,1)=0
      REG(J+128,2)=0
      SCORE(J)=0
100   SCORE(J+128)=0
      SR=W1+1
      SW=1
      SX=2
      LW=1
      IS=SSTART
      MI=MSTART
      SHIFT = 0
      RANDX = RANDOM(309998125)
      IS3 = 262139
      IM3 = 282729
110   CONTINUE
      IN=1
      IM=1
      IJ=0
      TP=0
      IF(TOP.NE.0)TP=9
      DO 131 I=1,324
      RNUM = ZOR(0)
      IF(RNUM-.5)135,136,136
135   MSG(IM+TP)=AND(MSG(IM+TP),AMASK(IN))
      RMSG = 0
      GO TO 137
136   MSG(IM+TP) = OR(MSG(IM+TP),OMASK(IN))
      RMSG = 1
137   IN = IN + 1
      IF(IN.LT.37)GO TO 138
      IN = 1
      IM = IM + 1
138   SHIFT = SHIFT/2 + RMSG*2**(CL-1)
      DO 130 J=1,RATE
      K=AND(SHIFT,HOOKUP(J))
      KOUNT = 0
```

```
137•            DO 850 L=1,36
138•            KADD = AND(K,OMASK(L))
139•            IF(KADD)850,850,840
140•      840   KOUNT = XOR(KOUNT,1)
141•      850   CONTINUE
142•            RMSG = KOUNT * 7
143•            IJ = IJ + 1
144•            CALL RAN(IS,MI,AK)
145•            K = AK * 1000.
146•            DO 120 L=1,8
147•            M = K-QUANT(L)
148•            IF(M)130,120,120
149•      120   CONTINUE
150•            IF(L.GT.8)L=8
151•      130   ENCODE(RATE*TOP+IJ) = XOR(ABS(L-1),RMSG)
152•      131   CONTINUE
153•      C
154•      C     END OF BLOCK D
155•      C
156•      C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
157•      CINPUT TO VITERBI ALGORITHM
158•      C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
159•      C     BEGINNING OF BLOCK E, THIS BLOCK PROCESSES
160•      C     324 INFORMATION BITS CORRESPONDING TO EACH PASS
161•            SPRED=0
162•            ERROR=0
163•            DO 440 NNN=1,324
164•            MMM=RATE*(NNN-1)
165•      C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
166•      CVITERBI ALGORITHM
167•      CNNN= NODE COUNTER
168•      CJ = PATH COUNTER (1-PATHS)
169•      CK = GENERAL VARIABLE
170•      CL = ORIGINAL STATE COUNTER
171•      CM = DOUBLE ORIGINAL COUNTER
172•      CN = FINAL STATE COUNTER
173•      CJJ=HOLDER OF POSSIBLE ENCODER
174•      CKK = WORKING VALUE
175•      CLL = SCORE HOLDER - FIRST
176•      CC MM = SCORE HOLDER - SECOND
177•      C     * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
178•            DO 400 J=1,PATHS
179•            L=J-1
180•            M=2*L
181•            M=M+1
182•            N=M
183•            NN=N-PATHS
184•            IF (NN) 150,140,140
185•      140   N=N-PATHS
186•      150   JJ=TABLE(M)
187•      C     BEGINNING OF SUBBLOCK E1. COMPARES THE TWO
188•      C     PATHS LEADING TO A NODE WITH THE INPUT
189•      C     FROM BLOCK D AND UPDATES THE SCORE
190•      CFIRST COMPARISON
191•            LL=0
192•            NDX = RATE*TOP+MMM+RATE+1
193•            DO 180 NN=1,RATE
194•            KK=AND(JJ,GMASK)
195•            KK = KK-ENCODE(NDX-NN)
196•            IF (KK) 160,170,170
197•      160   KK=-KK
198•      170   LL=LL+KK
199•      180   JJ=JJ/8
200•            JJ=TABLE(M+1)
201•      CSECOND COMPARISON
202•            MM=0
203•            DO 210 NN=1,RATE
204•            KK=AND(JJ,GMASK)
205•            KK = KK-ENCODE(NDX-NN)
206•            IF (KK) 190,200,200
207•      190   KK=-KK
208•      200   MM=MM+KK
209•      210   JJ=JJ/8
```

```
      LL=LL+SCORE(N+FIRST)
      MM=MM+SCORE(N+1+FIRST)
CSTORAGE OF OUTPUT BIT
      IF (L) 220,220,340
220   NN=INIT-PASS
      IF (NN) 230,230,240
230   NN=NNN-#3
C     BEGINNING OF SUBBLOCK E2, UPDATES THE
C     INDEX VALUE FOR THE PRINTOUT. STARTS
C     COUNTING ONLY AFTER #3 BITS ARE PROCESSED
      IF (NN) 300,300,240
240   K=AND(REG(H,LW),OMASK(LB))
      INDEX=INDEX+1
      OUTPUT(INDEX+TOPP)=K
      NN=INDEX-324
      IF (NN) 300,250,250
250   INDEX=0
C
C     END OF SUBBLOCK E2
C
C FORMATTING FOR PRINTOUT
C
C     o o o • • • • • • • • • • • • • • • • • • • • • • • •
CIJ = FIRST OUTPUT TO PROCESS
CIK = LAST OUTPUT BIT TO PROCESS
CIN = INCREMENT POSITION INDEX
C     o o o o • • • • • • • • • • • • • • • • • • • • • •
C     BEGINNING OF SUBBLOCK E3, PACKS THE DECODED
C     DATA INTO BUFFER, CALCULATES AND PRINTS
C     OUT UPDATED ERROR RATE, NO. OF ERRORS
C     ooooooo••••••••••••••••••••••••••••••••••••••••••••••••••••••
C     ENCODE AND BUFFER
      IJ=1
      IK=36
      TP = 0
      IF(BOT.NE.0)TP=9
      DO 290 IM=1,9
      IN=1
      DO 280 I=IJ,IK
      IF(OUTPUT(I+TOPP))260,270,260
260   BUFFER(IM)=OR(BUFFER(IM),OMASK(IN))
      IF(FLD((IN-1),1,MSG(IM+TP)).NE.1)ERROR=ERROR+1
      GO TO 280
270   BUFFER(IM)=AND(BUFFER(IM),AMASK(IN))
      IF(FLD((IN-1),1,MSG(IM+TP)).NE.0)ERROR=ERROR+1
280   IN=IN+1
      IJ=IJ+36
      OUTERR(IM) = XOR(BUFFER(IM),MSG(IM+TP))
290   IK=IK+36
CPRINTOUTP
600   ACCUM=ACCUM+ERROR
      FACC=ACCUM
      FINIT=INIT
      FPASS=PASS
      A=FACC/((FINIT-FPASS) * 324.0)
561   FORMAT(20H TRANSMITTED MESSAGE)
551   FORMAT(16H DECODED MESSAGE)
571   FORMAT(25H ERROR IN DECODED MESSAGE)
C
C     END OF SUBBLOCK E3
C
CSHIFT OF OUTPUT REGISTER
      IF(TOPP)291,291,292
291   TOPP=324
      BOTP=0
      GO TO 293
292   TOPP=0
```

```
277•            BOTP=324
278•      293   CONTINUE
279•      C     BEGINNING OF SUBBLOCK E4, SETS THE
280•      C     SR AND LB POINTERS,LB FOLLOWS SR
281•      300   SR=SR-1
282•            IF (SR) 310,310,320
283•      310   IF(SW.NE.1)GO TO 311              * USE ONLY ONE WORD
284•            SR=W1
285•            SX=2
286•            SW=1
287•            GO TO 320
288•      311   SR=W2
289•            SX=1
290•            SW=2
291•      320   LB=SR-1
292•            IF (LB) 330,330,340
293•      330   IF(LW.NE.1)GO TO 331              * USE ONLY ONE WORD
294•            LW=1
295•            LB=W1
296•            GO TO 340
297•      331   LW=2
298•            LB=W2
299•      C     END OF SUBBLOCK E4
300•      C     SCORE COMPARISON
301•      C     BEGINNING OF SUBBLOCK E5, FOR EVERY
302•      C     BIT PROCESSED, UPDATES 2**K VALUES
303•      C     OF SREG AND SCORE
304•      340   NN=LL-MM
305•            IF(NN)350,351,360
306•      351   CALL RAN3(IS3,IM3,AK)
307•            IF(AK .LT. .5)GO TO 360
308•      CFIRST CHOICE IS SMALLER
309•      350   SREG(J+SEC,SW)=AND(SREG(N+FIRST,SW),AMASK(SR))
310•            SREG(J+SEC,SX)=SREG(N+FIRST,SX)
311•            SCORE(J+SEC)=LL
312•            GO TO 370
313•      CLAST CHOICE IS SMALLER
314•      360   SREG(J+SEC,SW)=AND(SREG(N+1+FIRST,SW),AMASK(SR))
315•            SREG(J+SEC,SX)=SREG(N+1+FIRST,SX)
316•            SCORE(J+SEC)=MM
317•      370   NN=M-N
318•            IF (NN) 380,390,380
319•      380   SREG(J+SEC,SW)=OR(SREG(J+SEC,SW),OMASK(SR))
320•      390   CONTINUE
321•      C     END OF SUBBLOCK E5
322•      C     END OF LOOP OF STATES
323•      400   CONTINUE
324•      C     INTERCHANGE OF REGISTERS
325•      C     BEGINNING OF SUBBLOCK E6, RESETS THE
326•      C     'FIRST' AND 'SEC' VALUES AFTER EVERY BIT IS
327•      C     PROCESSED, SETS THE POINTER M
328•      C     CORRESPONDING TO THE MINIMUM SCORE
329•      C     PATH
330•            IF (FIRST) 420,410,420
331•      410   FIRST=128
332•            SEC=0
333•            GO TO 430
334•      420   FIRST=0
335•            SEC=128
336•      430   CONTINUE
337•      CFIND MINIMUM
338•            F=999999
339•            DO 460 G=1,PATHS
340•            NN=F-SCORE(G+FIRST)
341•            IF (NN) 460,460,450
342•      450   F=SCORE(G+FIRST)
343•            M=G+FIRST
344•      460   CONTINUE
345•      C     END OF SUBBLOCK E6 AND BLOCK E
346•      C     ************************************************************
347•      C     NUMERICAL SPREAD OF SCORES
348•      C     ************************************************************
349•      C     END OF LOOP OF 324 BITS PROCESSED
```

```
350•      440   CONTINUE
351•      C     BEGINNING OF BLOCK F, RESETS TOP
352•      C     AND BOT POINTERS FOR EVERY PASS, PRINTS OUT
353•      C     THE FINAL SUMMARY
354•      C     END OF BLOCK F
355•      C
356•            IF (TOP) 470,470,480
357•      470   CONTINUE
358•            TOP=324
359•            BOT=0
360•            GO TO 490
361•      480   CONTINUE
362•            TOP=0
363•            BOT=324
364•      490   CONTINUE
365•            DO 500 I=1,PATHS
366•      500   SCORE(I+FIRST)=SCORE(I+FIRST)-F
367•      C     • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • •
368•      CRETURN FOR ANOTHER PASS
369•      C     • • • • • • • • • • • • • • • • • • • • • • • • • • • • • • •
370•            ERRTOT=ERRTOT+ERROR
371•            PASS=PASS-1
372•            IF (PASS) 510,510,110
373•      510   WRITE(6,700)
374•      700   FORMAT(1H1,///,16X,18HSUMMARY OF RESULTS)
375•            WRITE(6,701) P
376•      701   FORMAT(1HO,20X,42HNOISE QUANTIZATION LEVEL CORRESPONDING TO ,F4.1,
377•           19H PERCENT.)
378•            NMESDG =(INIT-1)*324
379•            NCHNDG = RATE*NMESDG
380•            WRITE(6,702)NMESDG
381•      702   FORMAT(1HO,20X,16HMESSAGE LENGTH =,I39)
382•            WRITE(6,703)NCHNDG
383•      703   FORMAT(1HO,20X,24HENCODED MESSAGE LENGTH =,I31)
384•            WRITE(6,706)ERRTOT
385•      706   FORMAT(1HO,20X,33HTOTAL NUMBER OF DECODING ERRORS =,I22)
386•            PDCE = FLOAT(100*ERRTOT)/FLOAT(NMESDG)
387•            WRITE(6,707)PDCE
388•      707   FORMAT(1HO,20X,43HPROBABILITY OF DECODING ERROR, IN PERCENT =,
389•           1F12.3)
390•            GO TO 1
391•      750   CONTINUE
392•            PRINT 642,IS
393•      642   FORMAT(1HO,5X,34HINITIALIZATION VALUE FOR RANID =,O12)
394•            STOP
395•      C
396•      C
397•      520   FORMAT (15H  ERROR RATE = ,E15.8)
398•      530   FORMAT (21H  ERRORS THIS PASS = ,I5)
399•      540   FORMAT (2X,5(4X,2O11))
400•      550   FORMAT (2X,5(4X,O20))
401•      610   FORMAT(8H PASS = ,I4,5X,6H CL = ,I3,5X,8H RATE = ,I3)
402•      611   FORMAT(25H TOTAL DECODING ERRORS = ,I6)
403•      630   FORMAT(8I5,F5.3)
404•      631   FORMAT(8O5)
405•      640   FORMAT(1HO,7X,9HHOOKUP = ,8(O5,2X))
406•      641   FORMAT(1HO,7X,8HQUANT = ,8(I5))
407•            END


  1•            SUBROUTINE RAN(IS,IM,A)
  2•            IS=IM*IS
  3•            II=IS/34359738367
  4•            IS=IS-(II*34359738367)
  5•            A=ABS(IS)
  6•            A=A/34359738367
  7•            RETURN
  8•            END
```

```
C       BEGINNING OF ALTERNATE VERSION OF BLOCK D.
C       THIS VERSION, WHICH IS TO BE USED FOR
C       HARD-DECISION SIMULATIONS, SHOULD
C       REPLACE STATEMENTS 115 TO 154.

110     CONTINUE
        NN=RATE*324
        DO 130 J=1, NN
        CALL RAN(IS, MI, AK)
        IF(AK. LT. P)GO TO 133
        ENCODE(RATE*TOP+J)=0
        GO TO 130
133     ENCODE(RATE*TOP+J)=1
130     CONTINUE

C       END OF ALTERNATE BLOCK D
```

APPENDIX C

DETAILS OF CONVOLUTIONAL CODES USED IN VITERBI DECODER SIMULATIONS

This appendix contains configuration details of "good" convolutional codes that have been suggested by several investigators (refs. 8 to 10). The parameters (K, V, generator sequence, and generator coefficients) of each of the various codes are summarized in tables C-I to C-III, and the same information is presented in pictorial form (shift register representation) in figures C-1 to C-3, respectively. Although simulations were not performed using all the codes described in this appendix, details on the unused codes are included for reference and potential use by some readers.

REFERENCE

C-1. Heller, J. A.: Sequential Decoding, Short Constraint Length Convolutional Codes. Jet Propulsion Laboratory Space Program Summary 37-54, vol. III, Dec. 1968, pp. 171-177.

TABLE C-I. - NONSYSTEMATIC CODES SUGGESTED IN REFERENCE 8

Code number	K	V	Generator sequence	Generator coefficients
1	3	2	11, 10, 11	$7, 5\|_8$ = 111, 101
2	4	2	11, 11, 10, 11	$17, 15\|_8$ = 1111, 1101
3	5	2	11, 10, 10, 01, 11	$35, 23\|_8$ = 11101, 10011
4	6	2	11, 10, 11, 10, 01, 11	$75, 53\|_8$ = 111101, 101011
5	7	2	11, 10, 11, 11, 00, 01, 11	$171, 133\|_8$ = 1111001, 1011011
6	8	2	11, 10, 11, 10, 10, 01, 01, 11	$371, 247\|_8$ = 11111001, 10100111
7	3	3	111, 110, 111	$7, 7, 5\|_8$ = 111, 111, 101
8	4	3	111, 110, 101, 111	$17, 15, 13\|_8$ = 1111, 1101, 1011
9	5	3	111, 110, 101, 110, 111	$37, 33, 25\|_8$ = 11111, 11011, 10101
10	6	3	111, 100, 110, 101, 011, 111	$75, 53, 47\|_8$ = 111101, 101011, 10011
11	7	3	111, 110, 101, 101, 010, 001, 111	$171, 145, 133\|_8$ = 1111001, 1100101, 1011011
12	8	3	111, 110, 100, 111, 010, 101, 100, 111	$367, 331, 225\|_8$ = 11110111, 11011001, 10010101

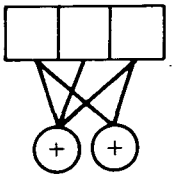
TABLE C-II. - NONSYSTEMATIC CODES SUGGESTED IN REFERENCE 9

Code number	K	V	Generator sequence	Generator coefficients
13	5	2	11, 00, 01, 11, 11	$23, 27\|_8$ = 10011, 10111
14	6	2	11, 01, 10, 01, 01, 11	$51, 67\|_8$ = 101001, 110111
15	7	2	11, 10, 11, 11, 00, 01, 11	$171, 133\|_8$ = 1111001, 1011011
16	8	2	11, 10, 01, 10, 11, 01, 01, 11	$331, 257\|_8$ = 11011001, 10101111
17	4	3	111, 101, 011, 111	$15, 13, 17\|_8$ = 1101, 1011, 1111
18	5	3	111, 011, 101, 011, 111	$25, 33, 37\|_8$ = 10101, 11011, 11111
19	6	3	111, 011, 110, 011, 001, 111	$51, 75, 67\|_8$ = 101001, 111101, 110111
20	7	3	111, 100, 011, 011, 101, 011, 111	$145, 133, 137\|_8$ = 1100101, 1011011, 1011111
[a]21	8	3	111, 001, 011, 100, 101, 011, 110, 111	$233, 247, 355\|_8$ = 10011011, 10100111, 11101101

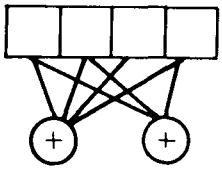
[a]Originally proposed by J. A. Heller (ref. C-1).

TABLE C-III. - SYSTEMATIC CODES SUGGESTED IN REFERENCE 10

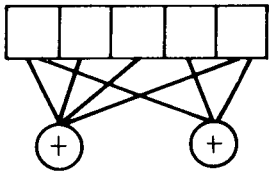
Code number	K	V	Generator sequence	Generator coefficients
22	5	2	11, 01, 00, 01, 00	$20, 32\|_8$ = 10000, 11010
23	7	2	11, 01, 00, 01, 00, 01, 00	$100, 152\|_8$ = 1000000, 1101010
24	8	2	11, 01, 00, 01, 00, 01, 00, 01	$200, 325\|_8$ = 10000000, 11010101
25	5	3	111, 001, 010, 010, 001	$20, 26, 31\|_8$ = 10000, 10110, 11001
26	7	3	111, 001, 010, 010, 001, 011, 011	$100, 133, 147\|_8$ = 1000000, 1011011, 1100111
27	8	3	111, 001, 010, 010, 001, 011, 011, 010	$200, 267, 316\|_8$ = 10000000, 10110111, 11001110



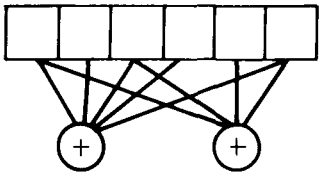
(a) Code 1 (V = 2, K = 3).



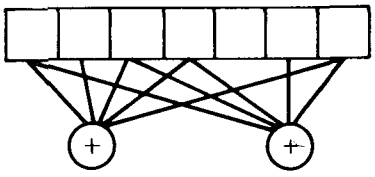
(b) Code 2 (V = 2, K = 4).



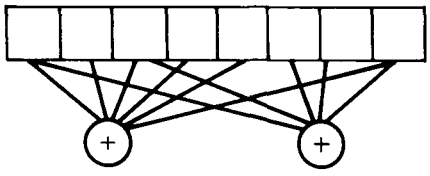
(c) Code 3 (V = 2, K = 5).



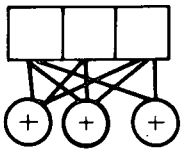
(d) Code 4 (V = 2, K = 6).



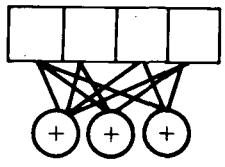
(e) Code 5 (V = 2, K = 7).



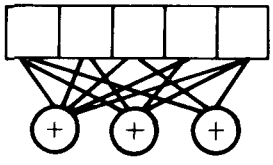
(f) Code 6 (V = 2, K = 8).



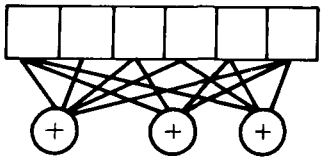
(g) Code 7 (V = 3, K = 3).



(h) Code 8 (V = 3, K = 4).

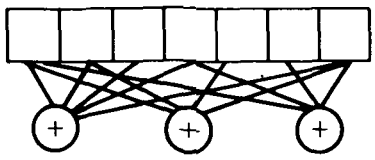


(i) Code 9 (V = 3, K = 5).

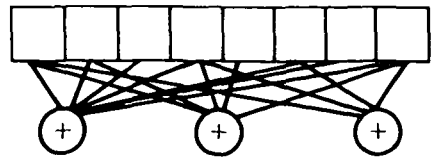


(j) Code 10 (V = 3, K = 6).

Figure C-1. - Shift register representation of the encoders using codes suggested in reference 8.

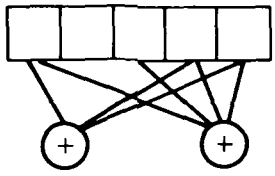


(k) Code 11 (V = 3, K = 7).

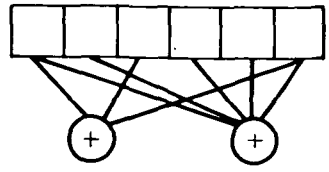


(l) Code 12 (V = 3, K = 8).

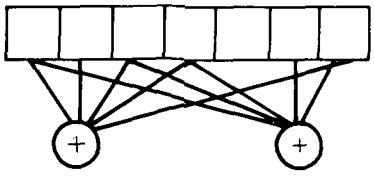
Figure C-1. - Concluded.



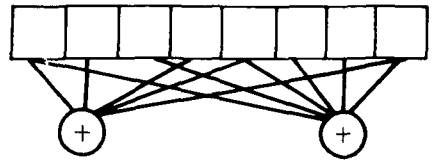
(a) Code 13 (V = 2, K = 5).



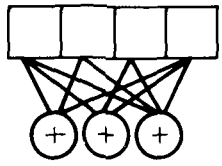
(b) Code 14 (V = 2, K = 6).



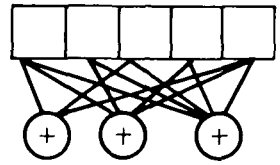
(c) Code 15 (V = 2, K = 7).



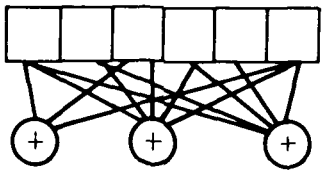
(d) Code 16 (V = 2, K = 8).



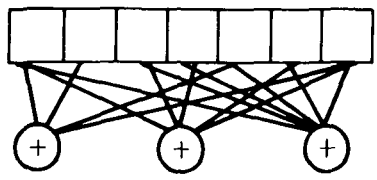
(e) Code 17 (V = 3, K = 4).



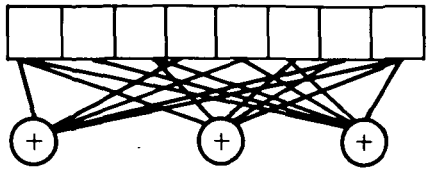
(f) Code 18 (V = 3, K = 5).



(g) Code 19 (V = 3, K = 6).

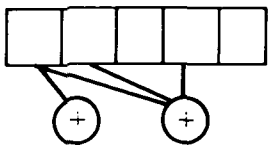


(h) Code 20 (V = 3, K = 7).

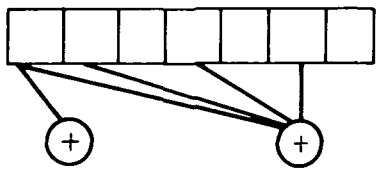


(i) Code 21 (V = 3, K = 8).

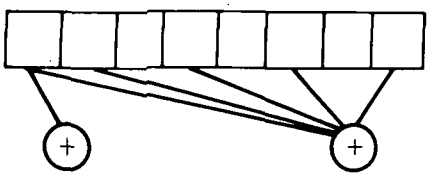
Figure C-2. - Shift register representation of the encoders using codes suggested in reference 9.



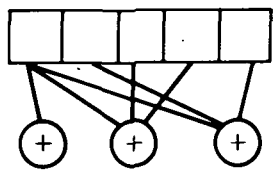
(a) Code 22 (V = 2, K = 5).



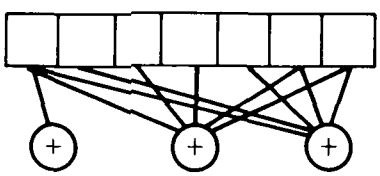
(b) Code 23 (V = 2, K = 7).



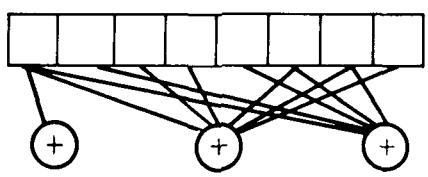
(c) Code 24 (V = 2, K = 8).



(d) Code 25 (V = 3, K = 5).



(e) Code 26 (V = 3, K = 7).



(f) Code 27 (V = 3, K = 8).

Figure C-3. - Shift register representation of the encoders using codes suggested in reference 10.

APPENDIX D

SIMULATION DATA FOR VITERBI DECODER SIMULATIONS

The input and output data for the Univac 1108 computer simulations of the Viterbi decoding algorithm are summarized in this appendix. For each code investigated, tables D-I to D-III contain the following information.

Code	Identification (the particular code)
Q	Receive quantization levels (Q = 2 is 2-level or hard-decision; Q = 8 is 8-level or 3-bit soft decision.)
K	Constraint length of the code
V	Number of modulo 2 adders in the encoder (Code rate = 1/V.)
Information bits	Number of information bits processed (The number of channel bits is V times this number.)
Output errors	Number of information bits in error after decoding
Input error probability	Channel bit error probability at decoder input or information bit error probability before coding
Output error probability	Information bit error probability at decoder output $\left(\frac{\text{number of output errors}}{\text{number of information bits}}\right)$
E_b/N_0	Ratio of energy per information bit to single-sided noise spectral density at decoder input

TABLE D-I. - VITERBI HARD-DECISION (Q = 2) SIMULATION DATA

(NONSYSTEMATIC CODES)[a]

Code number	K	V	Information bits	Output errors	Input error probability (b)	Output error probability	E_b/N_0, dB
3	5	2	66744	22	3.038×10^{-2}	3.3×10^{-4}	5.99
				92	4.523×10^{-2}	1.38×10^{-3}	4.58
				128	4.981×10^{-2}	1.92×10^{-3}	4.32
				915	6.975×10^{-2}	1.371×10^{-2}	3.38
9	5	3	66744	0	5.00×10^{-2}	--	6.09
				11	6.993×10^{-2}	1.65×10^{-4}	5.14
				52	9.00×10^{-2}	7.8×10^{-4}	4.31
				127	10.008×10^{-2}	1.9×10^{-3}	3.92
16	8	2	66744	22	4.523×10^{-2}	3.3×10^{-4}	4.58
				46	4.881×10^{-2}	6.9×10^{-3}	4.32
				628	6.975×10^{-2}	9.41×10^{-3}	3.38
				2839	8.99×10^{-2}	4.254×10^{-2}	2.55
21	8	3	66744	3	8.013×10^{-3}	4.5×10^{-5}	4.72
				22	9.0×10^{-2}	3.3×10^{-4}	4.31
				63	10.008×10^{-2}	9.4×10^{-4}	3.92
				129	10.996×10^{-2}	1.93×10^{-3}	3.52

[a]Refer to figure 11 and table I.

[b]Channel bit error probability (after coding).

TABLE D-II. - VITERBI SOFT-DECISION (Q = 8) SIMULATION DATA (NONSYSTEMATIC CODES)[a]

Code number	K	V	Information bits	Output errors	Input error probability	Output error probability	E_b/N_0, dB
2	4	2	66744	4	[b] 1.0×10^{-2}	5.99×10^{-5}	4.325
				128	[b] 2.0	1.91×10^{-3}	3.24
				446	[b] 3.0	6.68×10^{-3}	2.48
3	5	2	66744	1	[b] 1.0	1.49×10^{-5}	4.325
				100	[b] 2.0	1.5×10^{-3}	3.24
				306	[b] 2.5	4.58×10^{-3}	2.84
				319	[b] 3.0	4.78×10^{-3}	2.48
4	6	2	66744	56	[b] 2.0	8.39×10^{-4}	3.24
				154	[b] 2.5	2.307×10^{-3}	2.84
				305	[b] 3.0	4.57×10^{-3}	2.48
5	7	2	66744	16	[b] 2.0	2.397×10^{-4}	3.24
				67	[b] 2.5	1.004×10^{-3}	2.84
				190	[b] 3.0	2.847×10^{-3}	2.48
16	8	2	66744	8	[b] 2.0	1.2×10^{-4}	3.24
				56	[b] 2.5	8.39×10^{-4}	2.84
				167	[b] 3.0	2.5×10^{-3}	2.48
8	4	3	66744	4	[b] 1.0	5.99×10^{-5}	4.325
				67	[b] 2.0	1.004×10^{-3}	3.24
				224	[b] 3.0	3.356×10^{-3}	2.48
9	5	3	66744	4	[c] 11	5.99×10^{-5}	3.54
				18	[b] 2.0	2.7×10^{-4}	3.24
				71	[b] 2.5	1.06×10^{-3}	2.84
				189	[b] 3.0	2.83×10^{-3}	2.48
19	6	3	66744	22	[b] 2.0	3.296×10^{-4}	3.24
				55	[b] 2.5	8.24×10^{-4}	2.84
				148	[b] 3.0	2.217×10^{-3}	2.48
20	7	3	66744	15	[b] 2.0	2.247×10^{-4}	3.24
				23	[b] 2.5	3.446×10^{-4}	2.84
				65	[b] 3.0	9.74×10^{-4}	2.48
21	8	3	66744	15	[b] 2.0	2.2×10^{-4}	2.72
				64	[b] 3.0	9.6×10^{-4}	2.48

[a] Refer to figures 12 to 14 and tables I and II.

[b] Information bit error probability (before coding).

[c] Channel bit error probability (after coding).

TABLE D-III. - VITERBI SOFT-DECISION (Q = 8) SIMULATION DATA

(SYSTEMATIC CODES)[a]

Code number	K	V	Information bits	Output errors	Input error probability (b)	Output error probability	E_b/N_0, dB
22	5	2	66744	2	3.0×10^{-2}	3×10^{-5}	5.49
				13	4.8	1.9×10^{-4}	4.86
				43	5.0	6.4×10^{-4}	4.33
				177	7.0	2.65×10^{-3}	3.38
				437	9.0	7.09×10^{-3}	2.55
25	5	3	66744	6	7.0	9×10^{-5}	5.14
				21	9.0	3.1×10^{-4}	4.31
				72	11.0	1.08×10^{-3}	3.54

[a]Refer to figures 13 and 14.

[b]Channel bit error probability.

NASA-Langley, 1972 — 7 S-291

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D.C. 20546

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE $300

SPECIAL FOURTH-CLASS RATE
BOOK

POSTAGE AND FEES PAID
NATIONAL AERONAUTICS AND
SPACE ADMINISTRATION
451

U.S.MAIL

POSTMASTER: If Undeliverable (Section 158
Postal Manual) Do Not Return

"The aeronautical and space activities of the United States shall be conducted so as to contribute . . . to the expansion of human knowledge of phenomena in the atmosphere and space. The Administration shall provide for the widest practicable and appropriate dissemination of information concerning its activities and the results thereof."

—NATIONAL AERONAUTICS AND SPACE ACT OF 1958

NASA SCIENTIFIC AND TECHNICAL PUBLICATIONS

TECHNICAL REPORTS: Scientific and technical information considered important, complete, and a lasting contribution to existing knowledge.

TECHNICAL NOTES: Information less broad in scope but nevertheless of importance as a contribution to existing knowledge.

TECHNICAL MEMORANDUMS: Information receiving limited distribution because of preliminary data, security classification, or other reasons. Also includes conference proceedings with either limited or unlimited distribution.

CONTRACTOR REPORTS: Scientific and technical information generated under a NASA contract or grant and considered an important contribution to existing knowledge.

TECHNICAL TRANSLATIONS: Information published in a foreign language considered to merit NASA distribution in English.

SPECIAL PUBLICATIONS: Information derived from or of value to NASA activities. Publications include final reports of major projects, monographs, data compilations, handbooks, sourcebooks, and special bibliographies.

TECHNOLOGY UTILIZATION PUBLICATIONS: Information on technology used by NASA that may be of particular interest in commercial and other non-aerospace applications. Publications include Tech Briefs, Technology Utilization Reports and Technology Surveys.

Details on the availability of these publications may be obtained from:

SCIENTIFIC AND TECHNICAL INFORMATION OFFICE

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

Washington, D.C. 20546