DIGITAL MAGNETIC RECORDING AND VITERBI'S RECEIVER

by

Mr. Pui Chor Wong

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

Winnipeg, Manitoba

DIGITAL MAGNETIC RECORDING AND VITERBI'S RECEIVER

BY

PUI CHOR WONG

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

MASTER OF SCIENCE

To Mother, Sister and Priscilla.

ABSTRACT

High density digital magnetic recording is primarily a bandwidth limited channel resulting in intersymbol interference. Viterbi's algorithm, which is a maximum likelihood receiver, is studied specially for channels of this type. The channel and various coding schemes are modelled by a trellis representation. The overall system including Viterbi's algorithm is simulated using a combination of hardware and software. This is done for NRZI and MFM coding and three noises, Gaussian, Laplacian and quartic. A nonlinear intersymbol interference model is also introduced in the thesis though no simulation runs were performed.

ACKNOWLEDGEMENTS

I am grateful to lot of people without whom this thesis would not have been completed. They are Dr. Shwedyk, Dr. Gulak and Mr. E. Brusse. I would also like to thank NSERC for financial support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter I

INTRODUCTION

1.1 PRELIMINARY

With the invention of the digital computer, magnetic recording became indispensible for data storage. Core memories in the earliest computers were built using magnetic materials. Although magnetic core memories have been replaced recently by solid state memories, magnetic memories still have an important role in bulk storage. A great deal of research has been and is being done on magnetic recording systems to improve data access speed, increase data reliability and increase the recording bit density. This thesis is concerned with studying methods to achieve higher density magnetic recording. For this purpose, the magnetic recording system is modelled as a communication channel. Communication theory principles are then applied to study the channel.

1.2 DIGITAL MAGNETIC RECORDING SYSTEM (DMRS)

Five basic elements can be identified in a DMRS as shown in Figure 1.1(a). These are:

1. magnetic read/write head(s)
2. magnetic tape/disc
3. tape/disc transport
4. write amplifier
5. readback amplifier

The message in binary form is encoded into a write current which is further amplified by the write amplifier in order to provide proper drive for the write head. The magnetic flux set up by the write current results in a magnetization flux pattern being stored on the medium. The magnetization pattern is composed of two different pole directions where the magnetic material is saturated with equal field strength of opposite intensity to represent the binary symbols 0 and 1.

Based on the magnetization pattern stored on the medium, a voltage signal during the readback process is induced in the readback head. The relation between the voltage and the flux is:

$$e(t) = \left[\frac{dm(x)}{dx}\right] \times \left[\frac{dx}{dt}\right] \qquad (1.1)$$

where $e(t)$ is the induced voltage, $m(x)$ is magnetization pattern with respect to space or distance, and dx/dt is the speed of the tape/disc transport. If the magnetization pattern is ideal, the received signal will be a sequence of impulses. Practically, the received signal is a sequence of identically shaped pulses whose polarity is determined by the flux transition. A single flux transition produces an isolated pulse which is also sometimes known as the characteristic pulse. Various functional expressions have been developed for the isolated pulse [1,2,3]. Gulak [4] has fitted a quartic function to the pulse and this is the functional expression used in this thesis. The pulse shape is shown in Figure 1.1(b). Over the data rates found in a typical disc recording system the pulse is invariant in both shape and duration.

The entire recording process from the flux transitions produced by the input binary data to the read electronics output can be modelled as the pulse amplitude modulation (PAM) system shown in Figure 1.2(b). Unlike the typical PAM system encountered in communication system, the pulse sequence received has the special property of alternating in polarity. Thus each pulse in the sequence is correlated with the previous pulses. To properly utilize the channel, certain encoding methods are required [5,6,7]. Encoding methods considered in this thesis include Non-Return-to-Zero Inverse (NRZI), Modified Frequency Modulation (MFM), Convolutional Code of rate 1/2 (1/2 CC) together with NRZI or MFM code. These codes are discussed in Chapter 2. Other codes such as Frequency Modulation (FM), Non-Return-to-Zero (NRZ), and Miller's Modified Frequency Modulation (M FM) are not considered here.

Random noise sources of a DMRS include thermal noise, electronic noise, media noise, noise due to transport vibration and others. Though typically this noise, particularly the thermal and electronic, can be well modelled as white and gaussian (WGN), there is evidence that the probability density function can be otherwise. Gulak [4] has suggested that the probability density function is better approximated by a quartic expression. In this thesis, various noise models are considered.

The recording technique described above is usually called saturation recording. For nonsaturation recording the data is modulated with a analog signal which usually occupies more space on magnetic medium. Thus saturation recording is widely used in high bit density DMRS.

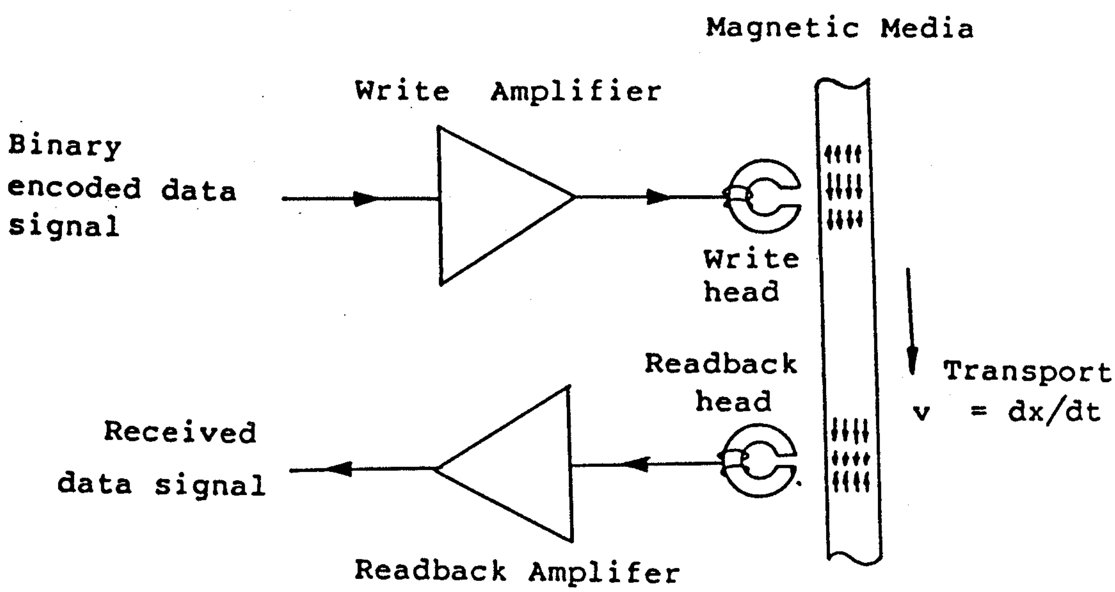


Figure 1.1 (a) Basic Digital Magnetic Recording System

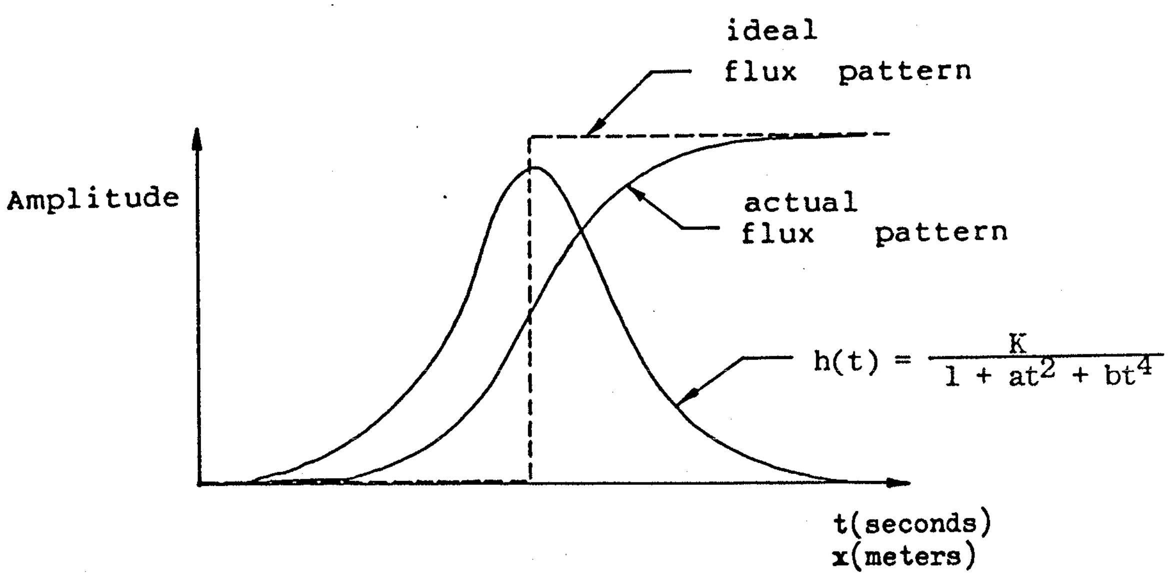


Figure 1.1 (b) Readback pulse, h(t). Axis scaling arbitrary.

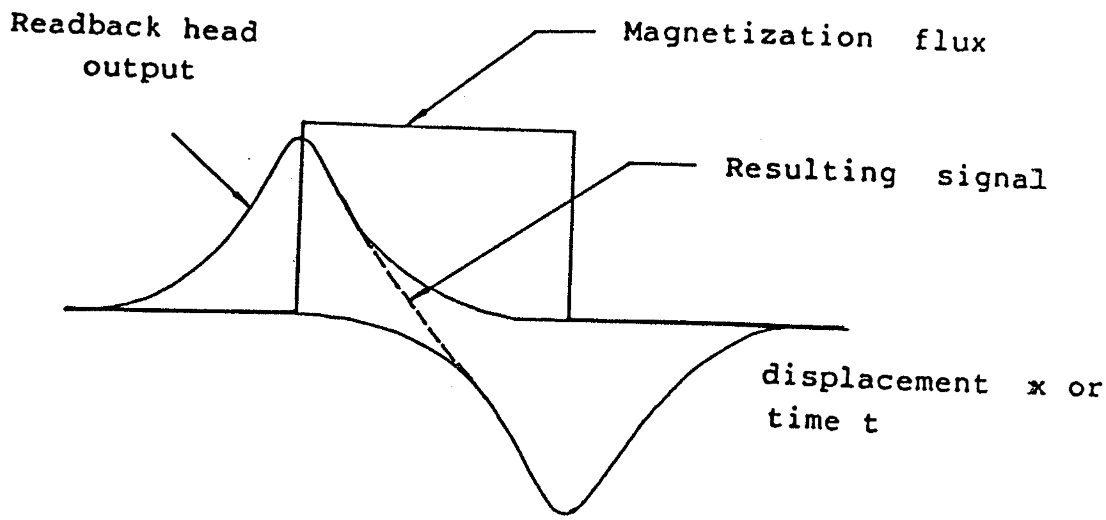


Figure 1.2 (a) Resulting signal from two adjacent saturation flux reversals

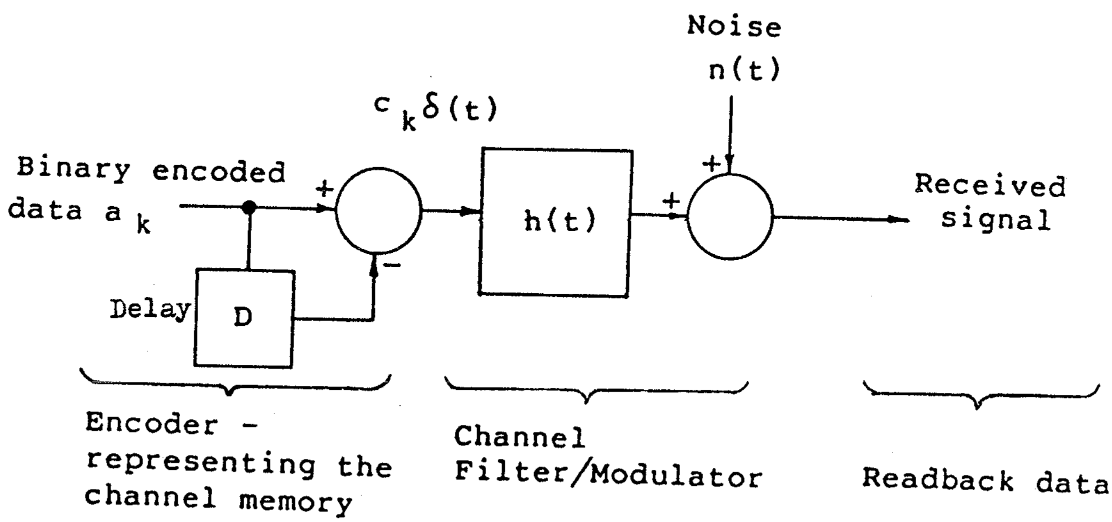


Figure 1.2 (b) An equivalent PAM system

1.3 HIGH DENSITY MAGNETIC RECORDING

As mentioned previously, saturation recording provides higher bit density storage compared with the nonsaturation recording technique. The reason is because in nonsaturation magnetic recording, the data is modulated by an analog signal which occupies a certain area on the medium for at least a period or half a period of a cycle. In saturation recording, a transition in polarity occupies less area (in the ideal case) as it is based on properties of the magnetic materials.

A major consideration in a high bit density DMRS is intersymbol interference (ISI). This produces distortions such as peak shifts and amplitude variations in the readback pulse. Within a certain range of separation, ISI can be analysed by using linear superposition. Two adjacent saturation flux reversals are shown in Figure 1.2(a) to illustrate the effect of ISI.

ISI is the dominant factor affecting high bit density recording. Chu [8] has studied ISI distortion by a computer simulation. A great deal of work has been done in reducing pulse width through techniques such as pulse slimming filters [9,10]. Decision feedback equalization (DFE) has been studied as a method of reducing the effect of ISI [4]. These techniques have the common objective of reliably detecting the pulse sequence in the presence of ISI and random noise. However, none are optimum with respect to ISI and random noise.

The optimal decoder/receiver for a channel with ISI and random noise is Viterbi's algorithm [11]. Although the algorithm was first applied to decoding convolutional codes, it was found later by Forney to be applicable for a channel with ISI [12]. Forney also showed that this algorithm is in fact a dynamic programming technique. The algorithm is a maximum likelihood sequence detector (MLSD) and is optimum when the noise is white and gaussian. H. Kobayasi [13] modelled the DMRS as a partial response system and applied Viterbi's algorithm. His work however is only applicable to the channel with no ISI. This thesis is concerned with the application of Viterbi's algorithm in the DMRS to combat ISI and random noise. Various encoding schemes and random noise sources are simulated and the algorithm performance is then studied.

1.4 OUTLINE OF THE THESIS

After the introduction in Chapter 1, Chapter 2 presents a detailed description of digital magnetic recording from a communication theory point of view. Various encoding schemes are discussed, trellises for these codes are developed and Viterbi's algorithm for decoding these codes is then developed. Chapter 3 describes the simulation of both the recording channel and Viterbi's receiver. Both hardware and software aspects of implementation are discussed. Results and discussions are found in Chapter 4. Evaluation of decoder performance with noise other than gaussian is also presented in this chapter. The time required for the simulation runs are also discussed here. Conclusions are presented in Chapter 5 along with recommendations for future research.

Chapter II

MAGNETIC CHANNEL

2.1 INTRODUCTION

A basic DMRS was described in Chapter 1. It consisted of 5 elements; the read/write head, medium, transport, write and read amplifier. An equivalence was established between the DMRS and a PAM communication model. In this chapter this equivalence is further developed. In particular attention is focussed on encoding for the channel.

As mentioned, in a DMRS the output pulse sequence alternates in polarity due to the inherent memory in the readback process. This property is modelled by a differential encoder. Also to properly utilize the channel various encoding schemes such as NRZI, FM, MFM, M^2FM and others have been proposed for the magnetic channel as mentioned in previous chapter. The next section of this chapter describes the trellis representations of two popular encoding schemes, NRZI and MFM. The trellis is a graphical represention which facilitates greatly the development of a Viterbi receiver.

Next intersymbol interference is considered. A brief review of the derivation of the maximum likelihood sequence estimator is presented and Viterbi's receiver for ISI channels is derived. Again it turns out that the natural representation to explain the receiver is a trellis.

Viterbi's receiver was originally developed for decoding convolutional codes. Thus a specific convolutional code is considered with the aim of providing error correction. Trellises for the cascade of the 1/2 CC with NRZI or MFM codes are developed. These trellises are used to derive the distance properties of the code. From these an overall trellis for the DMRS channel is obtained. The Viterbi receiver has a common structure for the different encoding schemes mentioned above, or various cascade combinations thereof. Only minor changes in the calculation of the branch metric are needed.

2.2 ENCODING

2.2.1 Inherent Encoding in the Channel

NRZ (Non-Return-to-Zero) coding is the simplest coding that can be used for any binary communication system. The digits "0" and "1" are simply represented by the two saturation levels of the magnetic medium which is why sometimes NRZ is called NRZL (Level). Since the readback process responds only to a flux transition the received waveform in the absence of noise is either a positive pulse, a negative pulse or no pulse. This coding is shown in Figure 2.1. The output ternary symbols (+1,0,-1) depend not only on the present input but also on the previous input. The channel thus has an inherent memory of length one. The output codeword from the readback head is given by

$$c_k = a_k - a_{k-1} \tag{2.1}$$

where c_k is the ternary output code symbol; a_k and a_{k-1} are input symbols and are binary. This process is called differential encoding.

There are several possible representations for the encoding process: sequential state machine representation, state diagram representation, tree representation, and trellis representation as shown in Figure 2.2(a) to 2.2(d). Trellis representation provides a graphical description for each codeword of the code. Distances between codewords can be visualised so that the code performance can be interpretated and evaluated.

From equation 2.1, if an error occurs in c_k, there would be an error in determining a_k. To see this, rearrange the equation as

$$\hat{a}_k = r_k + \hat{a}_{k-1} \tag{2.2}$$

where $r_k = c_k + n_k$; $\hat{a}_k$, $\hat{a}_{k-1}$ are estimates of a_k, a_{k-1}, and $a_{k-1} = c_{k-1} + a_{k-2}$. Therefore if c_{k-1} is in error

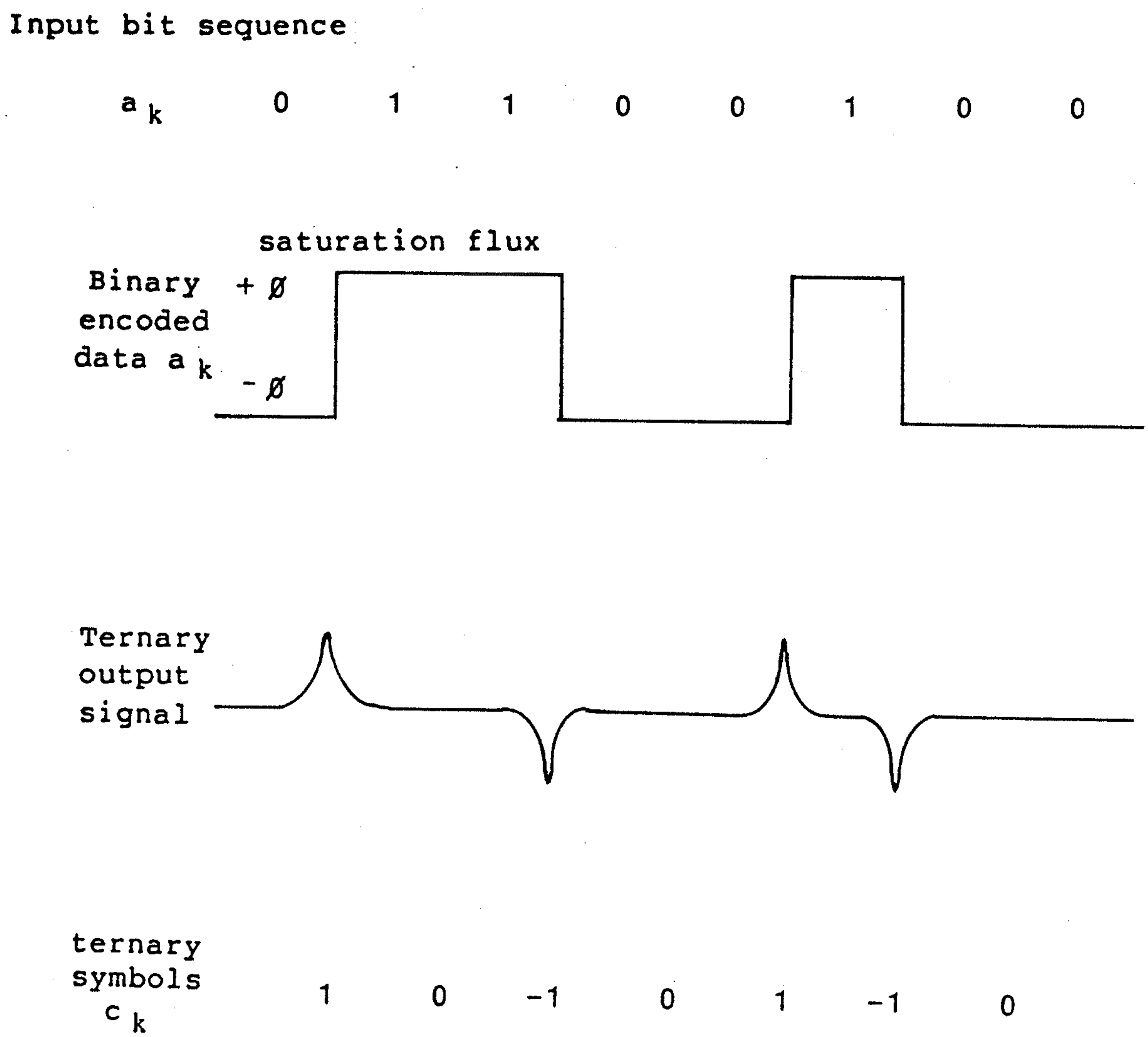
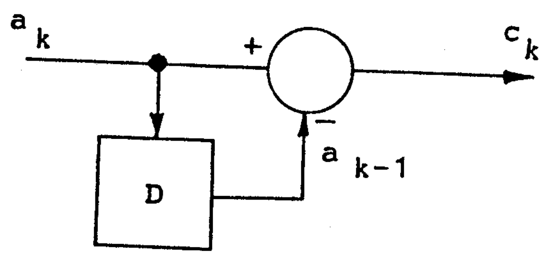
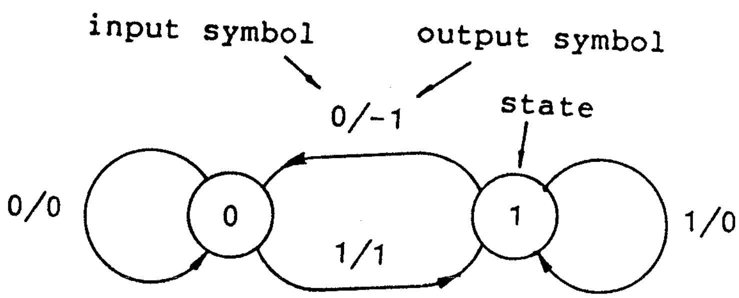


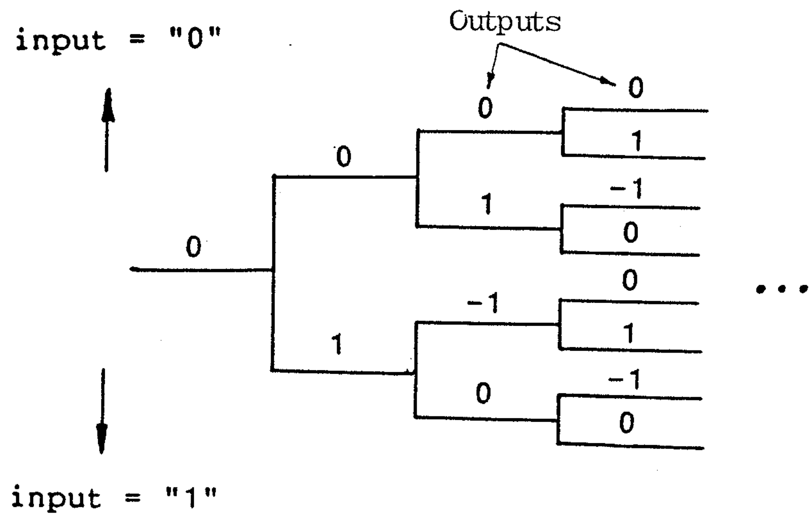
Figure 2.1 NRZ Coding



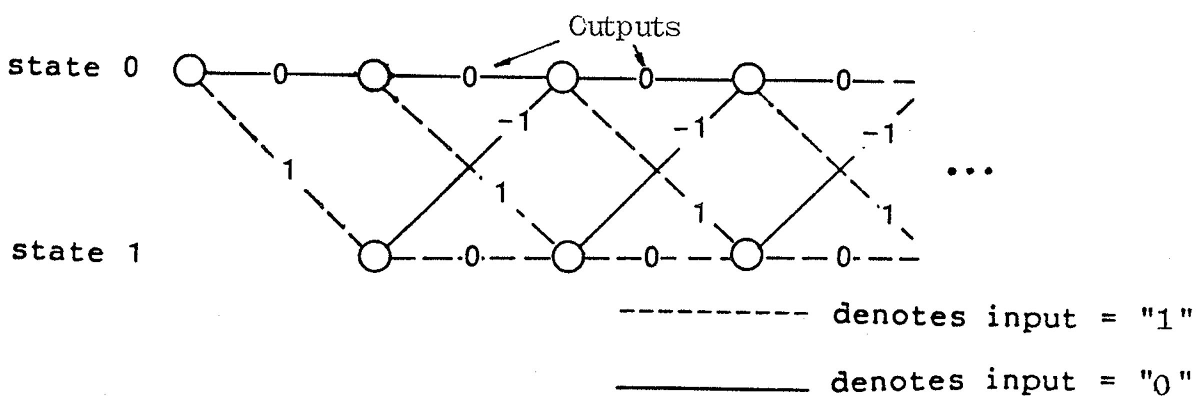
(a) state machine representation



(b) state diagram representation



(c) tree representation



(d) trellis representation

Figure 2.2 State machine (a), state diagram (b), tree (c) and trellis (d) representations of the NRZ code

Note: The above notation representing input symbols, output symbols, and states will be followed throughout the thesis.

then so is a_k, implying that errors propagate. NRZI is an encoding method which prevents this error propagation.

2.2.2 NRZI Code

Unlike NRZ code, NRZI represents the digit "1" by a transition from a saturation magnetic level to the opposite saturation magnetic level and the digit "0" by no transition. The code is shown in Figure 2.3. The encoder (Figure 2.4(a)) takes an NRZ code and outputs an NRZI code based on the equation

$$b_k = b_{k-1} \oplus a_k \tag{2.3}$$

where a_k is the present input digit, b_{k-1} is the previous output bit and b_k is the present output bit, a_k, b_k, b_{k-1} are all binary, and symbol $\oplus$ represents modulo-2 arithmetic.

Now the b_k's are the input sequence to the channel. Though the encoder has a memory of one and as discussed above the channel has an inherent memory of one the overall system memory is still one. That is the present ternary output symbol of the channel still depends only on the present and previous input according to the following equation

$$\begin{aligned} c_k &= b_k - b_{k-1} \\ &= b_{k-1} \oplus a_k - b_{k-1} \\ &= \pm a_k \end{aligned} \tag{2.4}$$

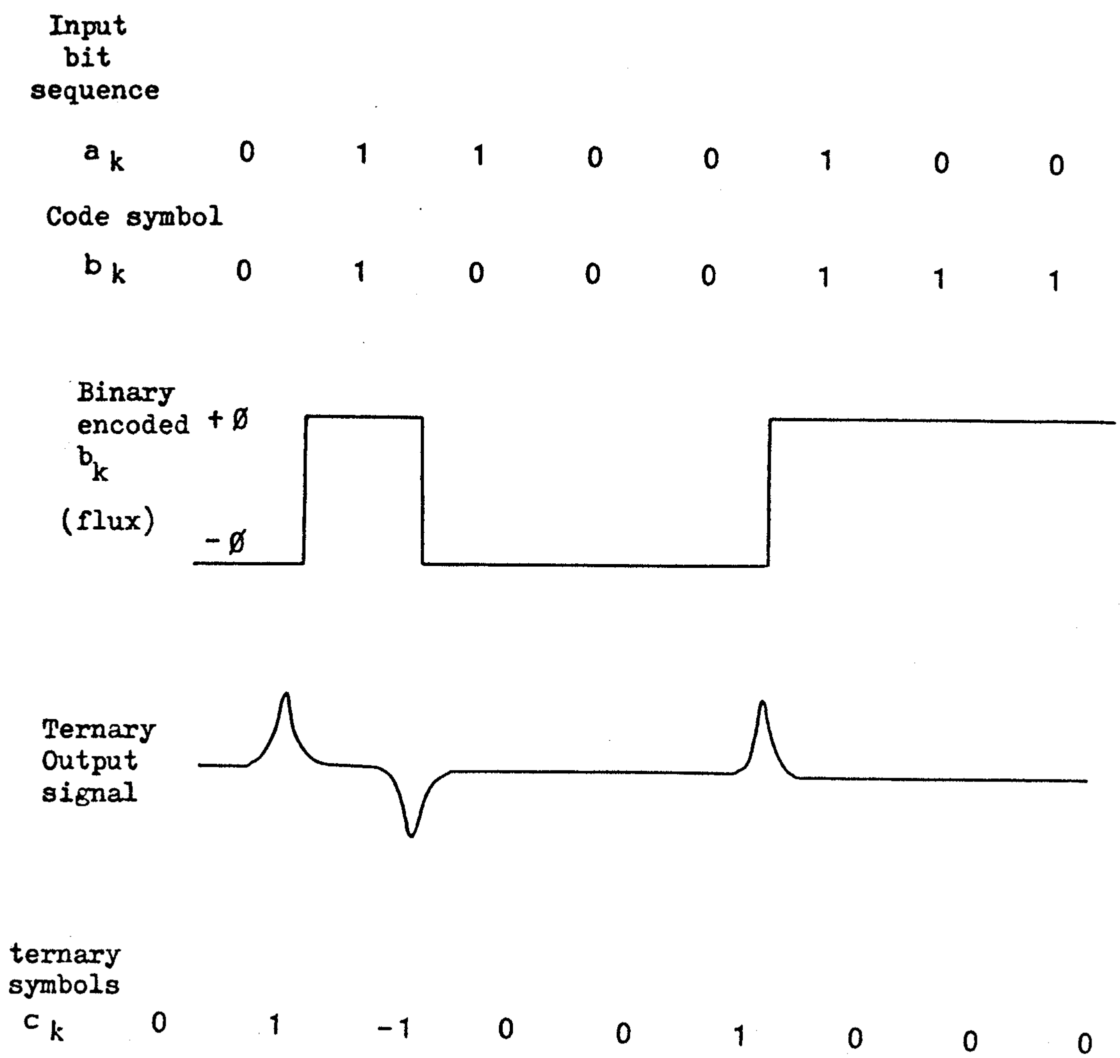


Figure 2.3 NRZI Coding

Error propagation is prevented since the output symbol is not effected by the previous input. The previous input determines only the sign for the output.

Again there are different representations for NRZI as shown in Figure 2.4(a) to 2.4(d). Cascading the inherent channel encoding process with the NRZI encoder results in the trellis of Figure 2.4(d). The trellis has two states corresponding to the memory length of one. Compared to the NRZ code trellis of Figure 2.2(d), one observes that the structure is the same. This is expected since in both instances there are two states due to the memory length of one. The only difference is in the output symbols. The output symbol for NRZI code can be expressed as

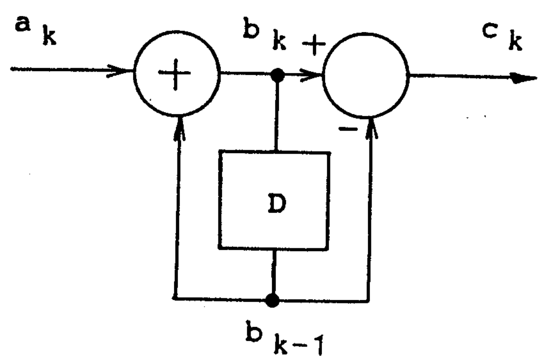
$$c_k = q_k - q_{k-1} \tag{2.5}$$

while the next state is given by,

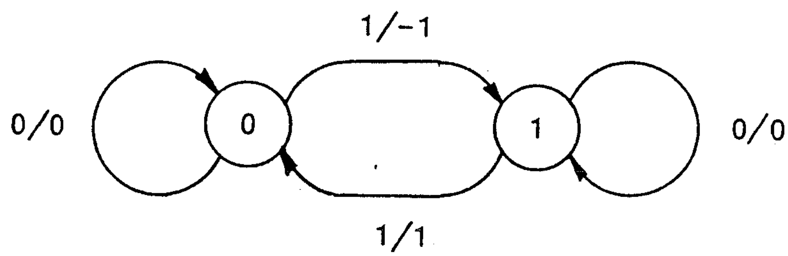
$$q_k = a_k + q_{k-1} \tag{2.6}$$

where q_{k-1} is the present state, a_k is the present input, c_k is the present output and q_k is the next state.

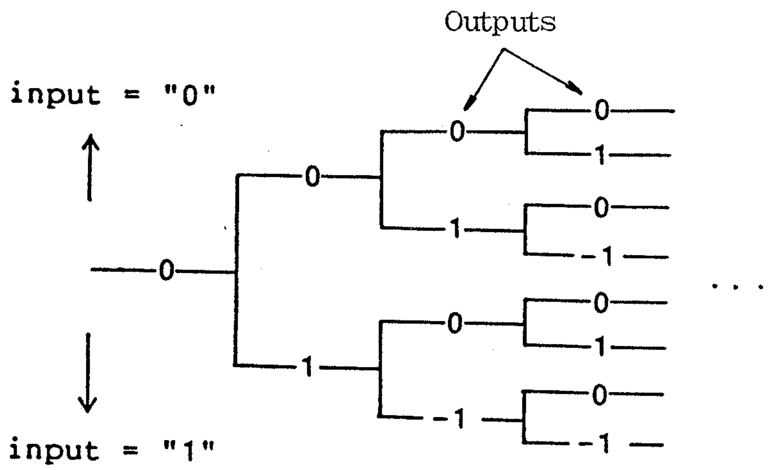
Both NRZ and NRZI are linear codes which provide a null sequence for a zero input codeword. Since a long string of zeros leads to no transitions, timing information is not available for these two codes. To imbed timing information in the code, FM coding was developed. It provides timing synchronization for the receiver but is inferior in



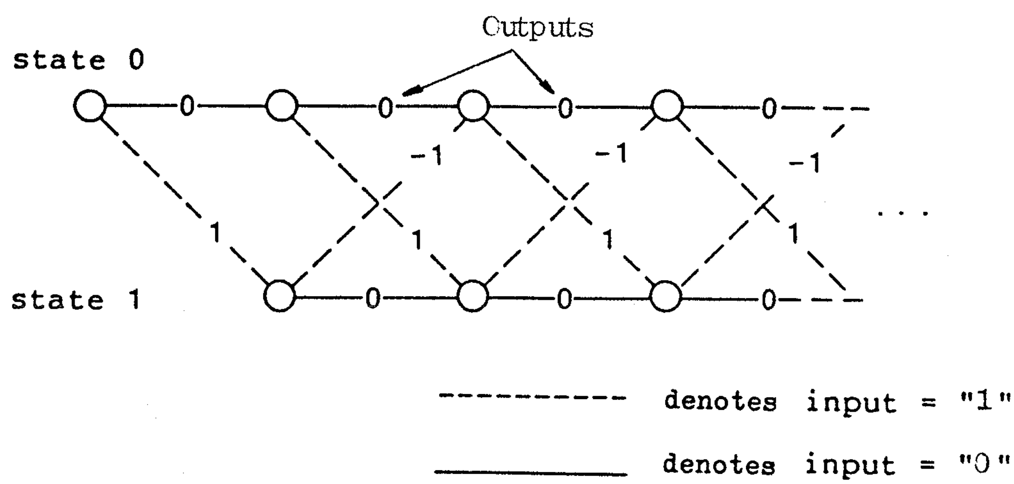
(a) state machine representation



(b) state diagram representation



(c) tree representation



(d) trellis representation

Figure 2.4 State machine (a), state diagram (b), tree (c) and trellis (d) representations of the NRZI code terms of density. MFM provides both density and timing advantages over other codes and is the most popular code in use.

2.2.3 MFM Code

A brief description of the code is as follows:

1. If the present input is a "1", the present output is the complement of the previous output.
2. If the present input is a "0", the present output will be equal to previous output if the next input is a "1". If the next input is again a "0" then the present output will be equal to previous output for one half of a bit interval and will be equal to the complement of the previous output for the rest of that bit interval.

Figure 2.5 illustrates the encoding process. The encoder has the form as shown in Figure 2.6(a). It accepts one input symbol and outputs two symbols. There are two memory elements and it has therefore a memory of length 2.

Since the memory length is two, the number of states equals to 4. The four states correspond to states {0, 1, 2, 3} or {00, 01, 10, 11} according to the following convention. The least significant digit is the rightmost digit and also corresponds to the most recent memory for the input. The state diagram representation is shown in Figure 2.6(b).

The tree for the encoder is shown in Figure 2.6(c) and the trellis representation is shown in Figure 2.6(d). The structure of the trellis for MFM is not regular since the code is obviously a nonlinear code due to the NOR function.

The output codeword and the new states are given by the following equations

$$q1_k = a_k \tag{2.7}$$

$$q2_k = F(q1_{k-1}, a_k) \oplus q2_{k-1} \oplus a_k \tag{2.8}$$

$$b1_k = F(q1_{k-1}, a_k) \oplus q2_{k-1} \tag{2.9}$$

$$b2_k = b1_k \oplus a_k \tag{2.10}$$

$$c_{2k-1} = b1_k - b2_{k-1} \tag{2.11}$$

$$c_{2k} = b2_k - b1_k \tag{2.12}$$

where $b1_k$, and $b2_k$ are the outputs of MFM code, F(.) is NOR function of enclosed variables, a_k is the present input, $q1_k$ and $q2_k$ are next states, c_{2k-1} and c_{2k} are ternary outputs.

The difference between MFM and NRZI is that for a null input sequence, the MFM output contains digits "0" and "1". However the shortest time interval between digit "1"s is the same for both codes even though the MFM encoder outputs 2 symbols for every input symbol. Thus the ISI terms are the same for either code.

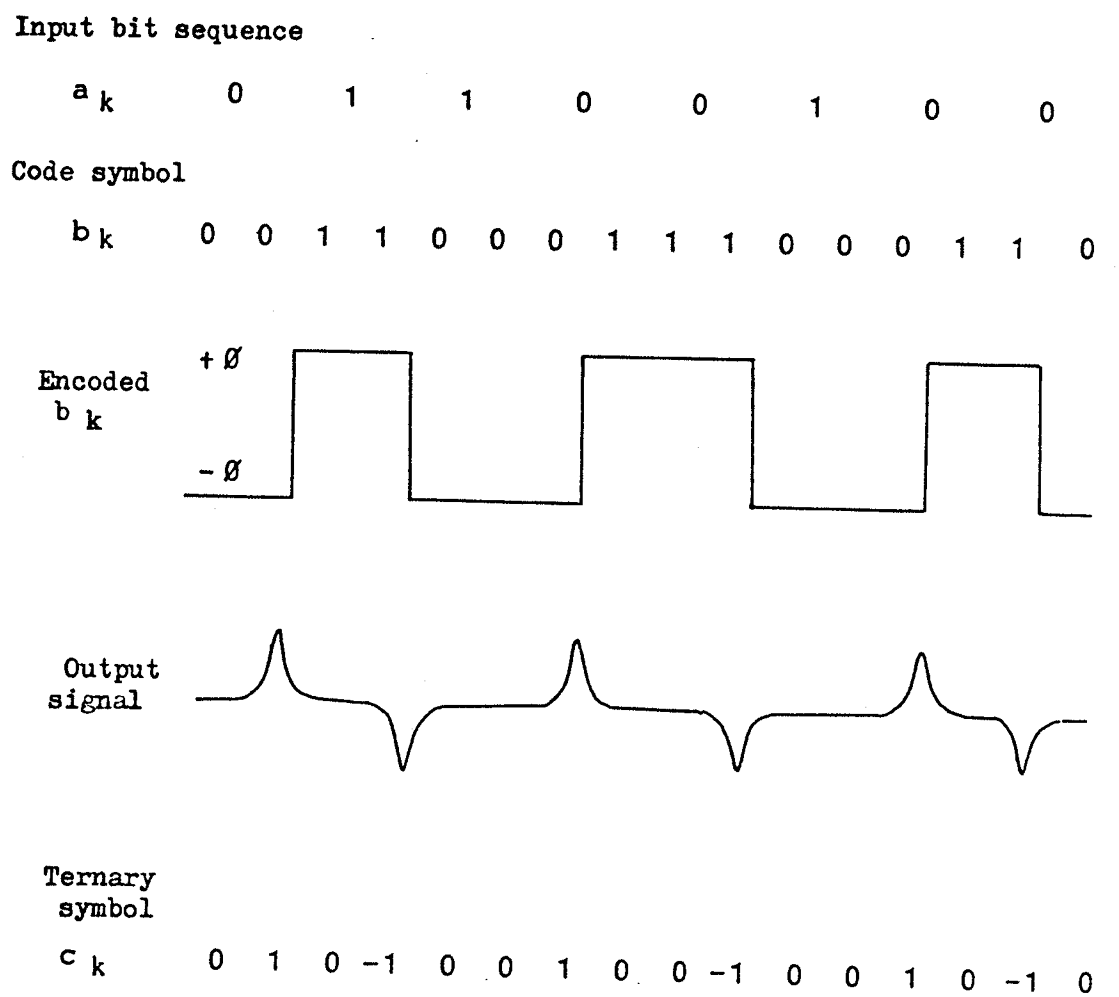
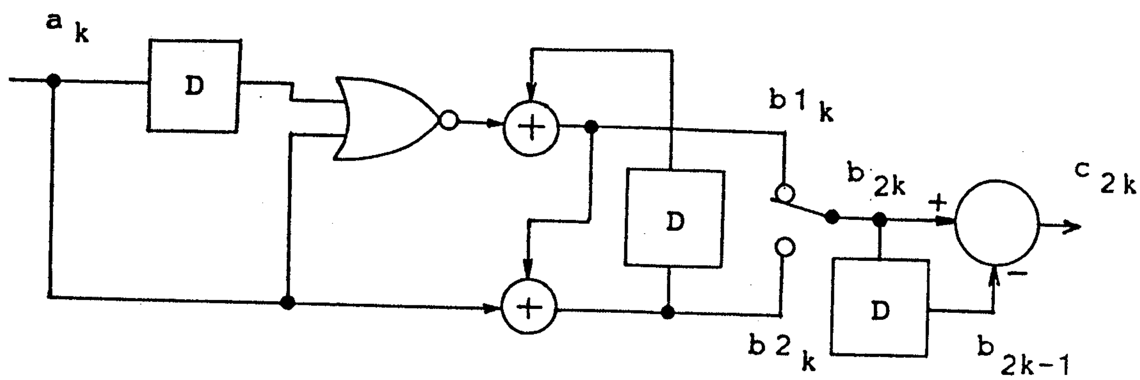
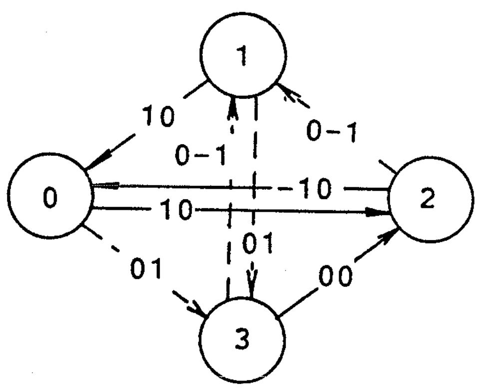


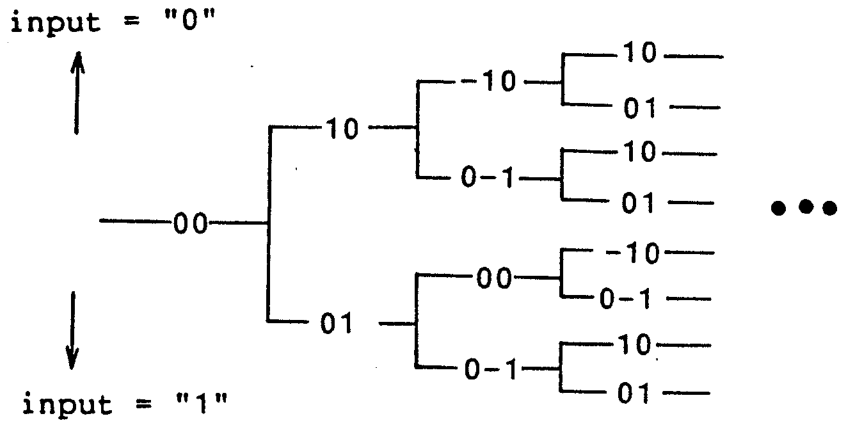
Figure 2.5 MFM Coding



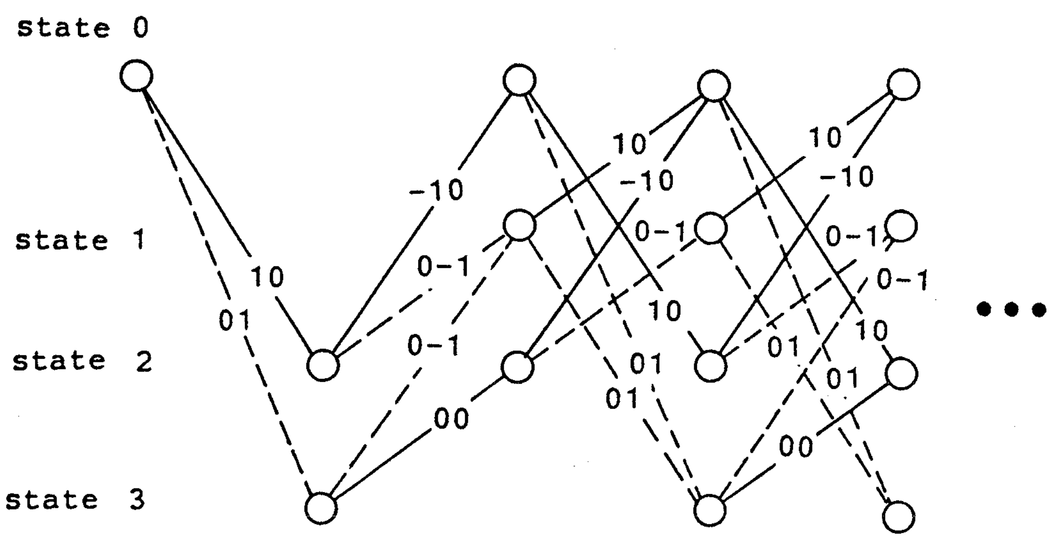
(a) State Machine Representation



(b) State Diagram Representation



(c) Tree Representation



(d) Trellis Representation

Figure 2.6: State machine (a), state diagram (b), tree (c) and trellis (d) representations of the MFM code

Note: State 0 corresponds to the two memory elements in the encoder being zero, state 1 corresponds to memory D1 being 1 and D2 being 0, state 2 corresponds to D1 being 0 and D2 being 1, state 3 corresponds to D1 and D2 being 1.

2.3 INTERSYMBOL INTERFERENCE AND VITERBI RECEIVER

2.3.1 Viterbi's algorithm

A DMRS is basically a band-limited channel which results in ISI that leads to degradation of receiver performance. This can be minimized by Viterbi's algorithm which provides an optimum sequence estimate. For completeness Viterbi's algorithm is presented here. It follows very closely the derivation presented in [14].

The ternary PAM system is shown in Figure 2.7. This corresponds to the DMRS channel. As mentioned the binary to ternary encoding is due to the inherent differential encoder in the magnetic channel. The ternary digits c_i are modulated by the channel filter h(t) such that the transmitted signal is

$$s(t) = \sum_{i=-N}^{N-1} c_i \, h(t-iT) \tag{2.13}$$

where c_i is a ternary symbol output from the differential encoder.

The ternary symbols are independent regardless of the encoding part. For message length of 2N, the total number

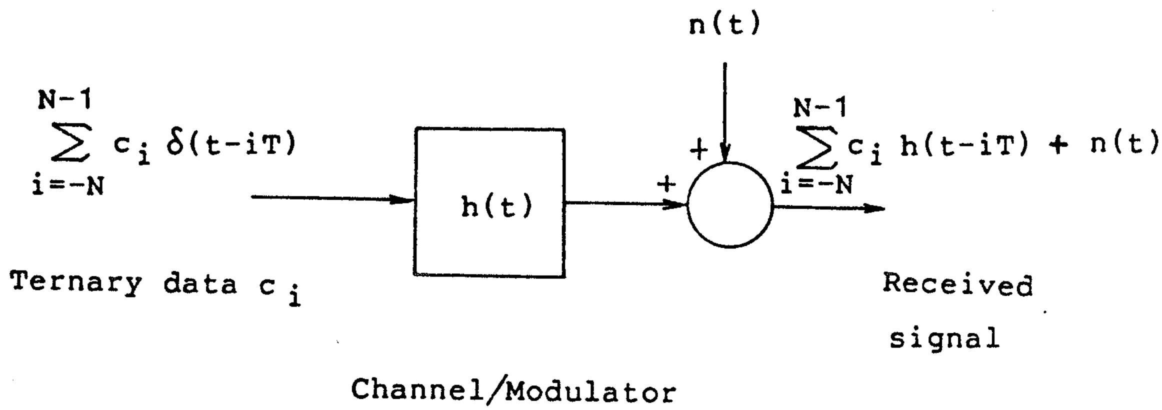


Figure 2.7 Ternary PAM system

of distinct sequences or signals are 3^{2N}. To find the maximum likelihood sequence the received signal is compared to all possible sequences. The comparison is based on a set of samples y_k called sufficient statistics obtained from the output of the matched filter. The maximum likelihood decision is made by computing the likelihood function when sequence, $\bar{c}_m$, is compared to sequence $\bar{c}_{m'}$, i.e., choose $\bar{c}_m$ if

$$\ln \left(\frac{p(\bar{y}|\bar{c}_m)}{p(\bar{y}|\bar{c}_{m'})} \right) \geq 0 \tag{2.14}$$

for all message $m' \neq m$.

For white gaussian noise, the likelihood function can be expressed as

$$\begin{aligned} \Lambda_{mm'} &= \ln \left(\frac{p(\bar{y}|\bar{c}_m)}{p(\bar{y}|\bar{c}_{m'})} \right) \\ &= \frac{2}{N_o} \int_{-\infty}^{+\infty} \left[c_m(t) - c_{m'}(t) \right] y(t)\, dt \qquad (2.15) \\ &\quad - \frac{1}{N_o} \int_{-\infty}^{+\infty} \left[c_m^2(t) - c_{m'}^2(t) \right] dt \end{aligned}$$

which reduces to

$$\Lambda_m = \frac{2}{N_o}\int_{-\infty}^{+\infty} c_m(t)y(t) - \frac{1}{N_o}\int_{-\infty}^{+\infty} c_m^2(t)\,dt \tag{2.16}$$

$$= \frac{2}{N_o}\sum_{k=-N}^{N-1} c_{mk}y_k - \frac{1}{N_o}\sum_{j=-N}^{N-1} c_{mk}c_{mj}h_{k-j}$$

where $h_{k-j} = \int_{-\infty}^{+\infty} h(t-kT)\,h(t-jT)\,dt$

$$= h_i \quad ; \quad i = k - j$$

Since h_i is symmetric, Λ_m becomes

$$\frac{1}{N_o}\sum_{k=-N}^{N-1} \left(2c_{mk}y_k - c_{mk}^2 h_o - 2c_{mk}\left(\sum_{i=1}^{L-1} c_{mk-i}h_i\right)\right)$$

$$= \frac{1}{N_o}\sum_{k=-N}^{N-1} \lambda_{mk} \tag{2.17}$$

where λ_{mk} is called the kth branch metric of message m

and

$$\lambda_{mk} = 2c_{mk}y_k - c_{mk}^2 h_o - 2c_{mk}\sum_{i=1}^{L-1} c_{mk-i}h_i$$

Now consider

$$\lambda_k = 2c_k y_k - c_k^2 h_0 - 2c_k \sum_{i=1}^{L-1} c_{k-i} h_i \tag{2.18}$$

where subscript m has been dropped.

This expression for the branch metric depends on the present received sample y_k, coefficients h_i $(i=0,\ldots,L-1)$ which are known a priori, the present input c_k and L-1 previous inputs c_{k-i} $(i=1,\ldots,L-1)$. Considering these L-1 previous inputs to define a state then as a new input sample c_{k+1} is received the state changes from $\{c_k, c_{k-1}, \ldots, c_{k-L+1}\}$ to $\{c_{k+1}, c_k, \ldots, c_{k-L}\}$. All the possible paths necessary for the computation of the c_k can be visualized by a trellis.

For a ternary PAM system with ISI and no encoder, the trellis is shown in Figure 2.8(a) and for a DMRS, the trellis is shown in Figure 2.8(b).

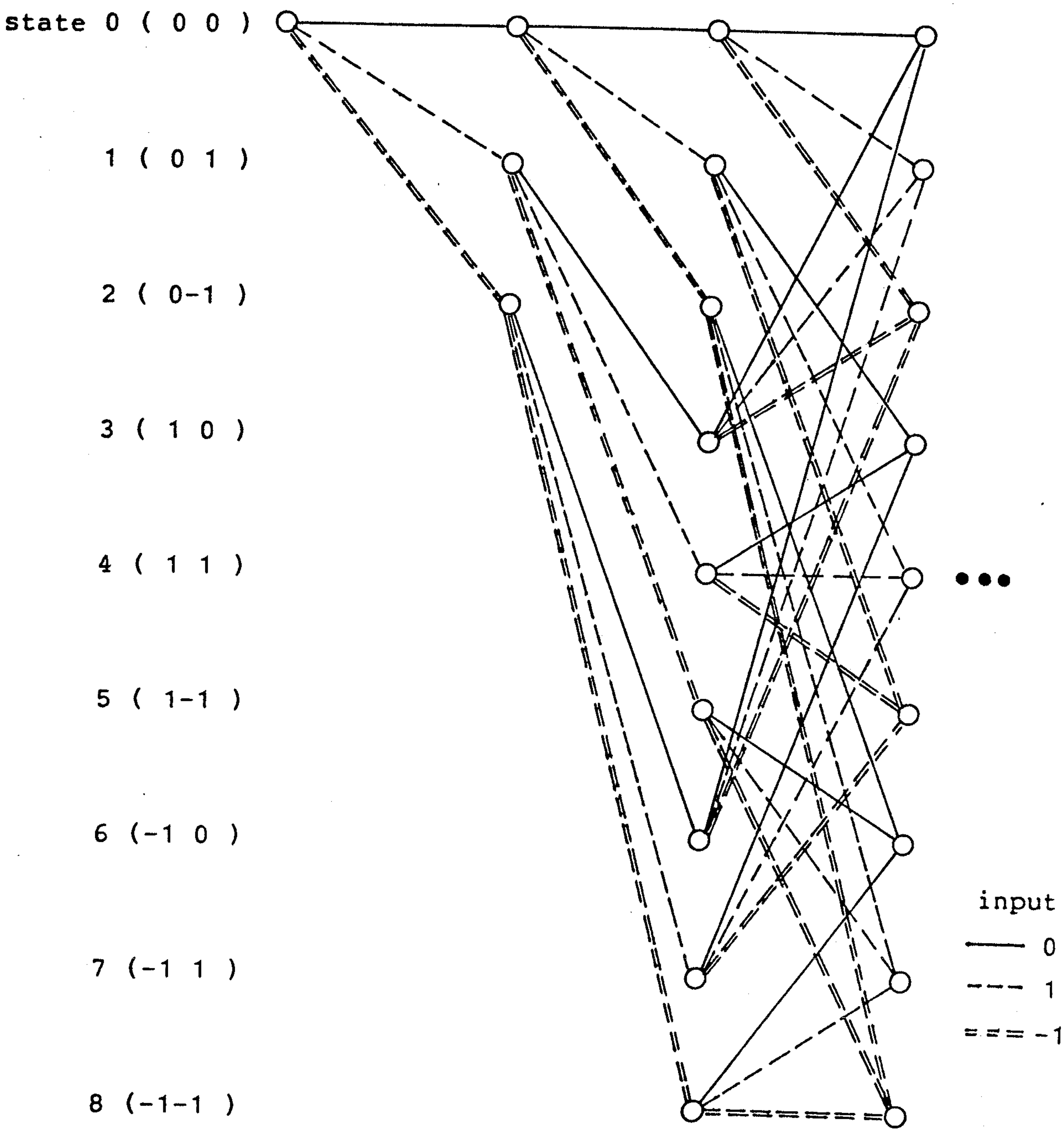


Figure 2.8 (a) Trellis of ternary PAM system with ISI for L=3

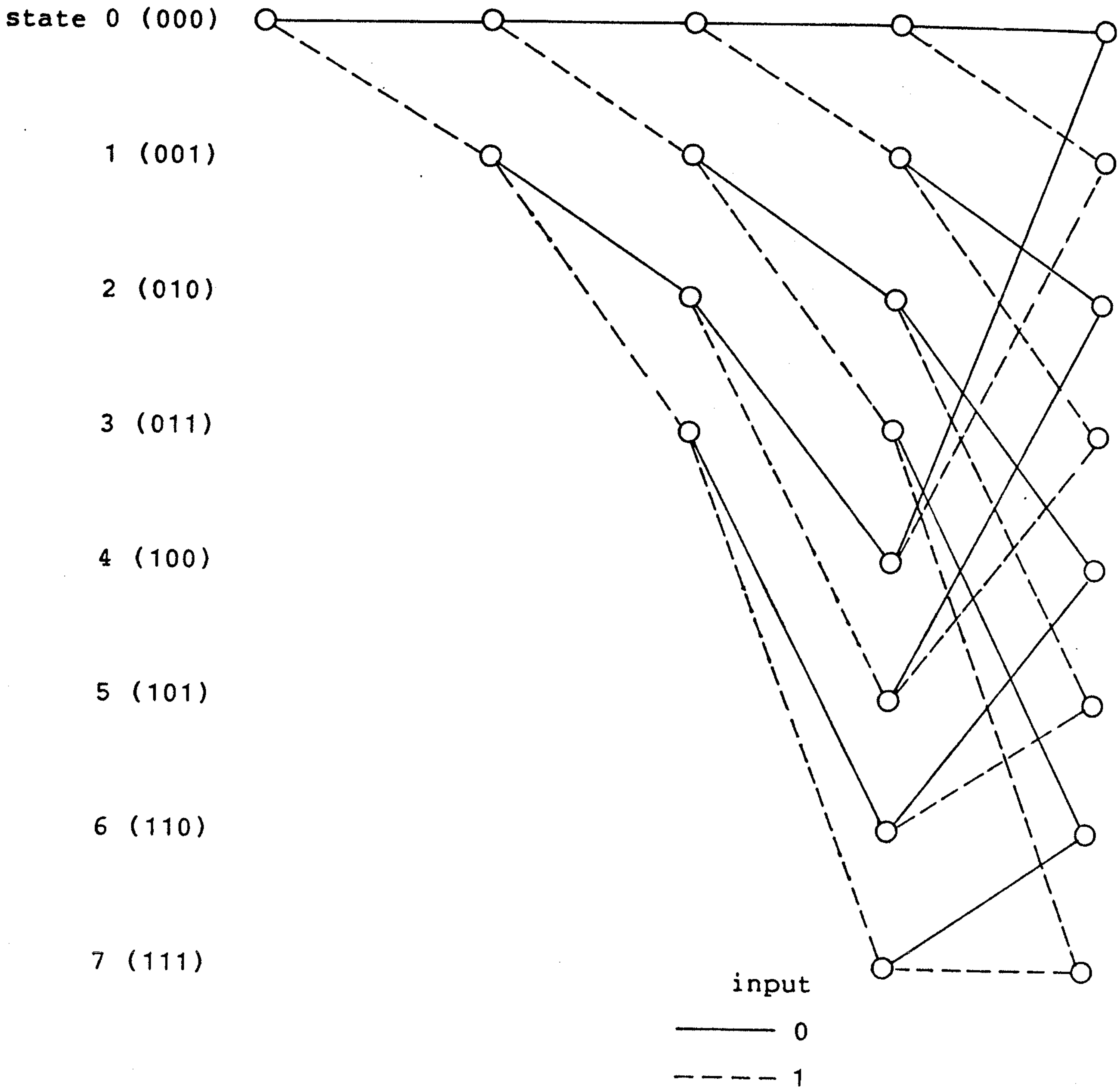


Figure 2.8 (b) Trellis of DMRS with ISI for NRZ and L=3

2.3.2 Branch metrics for the DMRS with ISI

According to equation 2.18 for the branch metric, the ternary symbols can be replaced by the encoding function. The branch metric depends on the source symbols and in this case are binary. Substituting equation 2.1 into equation 2.18, the equation becomes

$$\lambda_k = 2(a_k - a_{k-1})y_k - (a_k - a_{k-1})^2 h_o$$

$$-2(a_k - a_{k-1}) \sum_{i=1}^{L-1} (a_{k-i} - a_{k-i-1}) h_i \tag{2.19}$$

For the NRZI code, the branch metric can be obtained by substituting equation 2.5 into equation 2.18.

$$\lambda_k = 2(q_k - q_{k-1})y_k - (q_k - q_{k-1})^2 h_o$$

$$-2(q_k - q_{k-1}) \sum_{i=1}^{L-1} (q_{k-i} - q_{k-i-1}) h_i \tag{2.20}$$

The trellis for NRZI coding and ISI is shown in Figure 2.9(a). The least significant digit corresponds to the most recent input symbol remembered. The most significant digit is the memory element in the channel which changes according to equation 2.6.

Similary the branch metric for MFM code is obtained by substituting equations 2.11 and 2.12 into equation 2.18. It becomes

$$\lambda_k = \lambda_a + \lambda_b \tag{2.21}$$

where

$$\lambda_a = 2(b1_k - b2_{k-1})y_k - (b1_k - b2_{k-1})^2 h_o$$
$$-2(b1_k - b2_{k-1})\left[\sum_{i=even}(b1_{k-\lceil\frac{i}{2}\rceil} - b2_{k-\lceil\frac{i+1}{2}\rceil})h_i\right.$$
$$\left.+\sum_{i=odd}(b2_{k-\lceil\frac{i}{2}\rceil} - b1_{k-\lceil\frac{i+1}{2}\rceil})h_i\right]$$

$$\lambda_b = 2(b2_k - b1_k)y_k - (b2_k - b1_k)^2 h_0$$
$$-2(b2_k - b1_k)\left[\sum_{i=odd}(b2_{k-\lceil\frac{i}{2}\rceil} - b1_{k-\lceil\frac{i+1}{2}\rceil})h_i\right.$$
$$\left.+\sum_{i=even}(b1_{k-\lceil\frac{i}{2}\rceil} - b2_{k-\lceil\frac{i+1}{2}\rceil})h_i\right]$$

and $\lceil x \rceil$ denotes the least integer not less than x.

The trellis is shown in Figure 2.9(b). The least significant digit also represents the most recent input re-

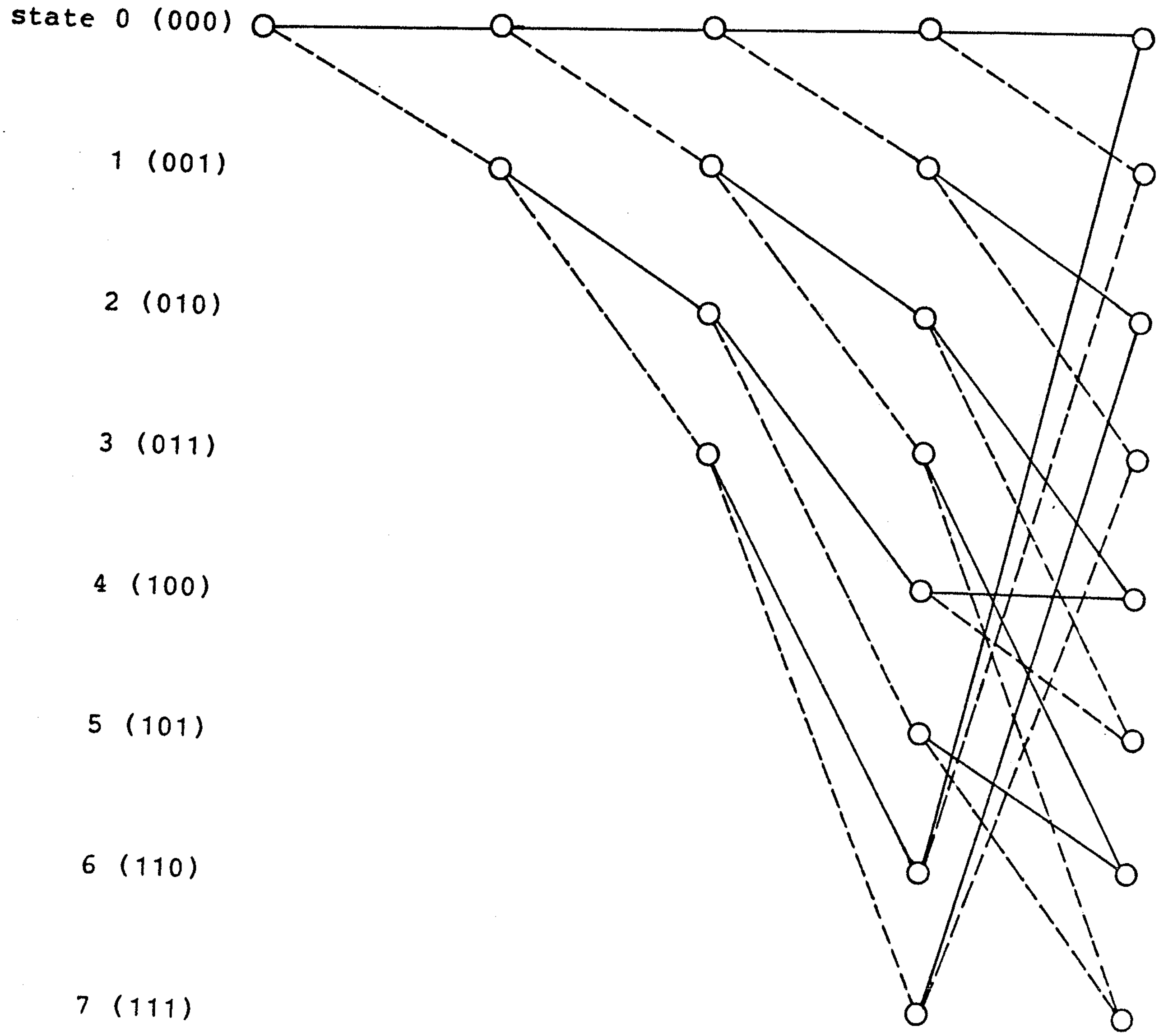


Figure 2.9 (a) Trellis of DMRS with NRZI code and L=3

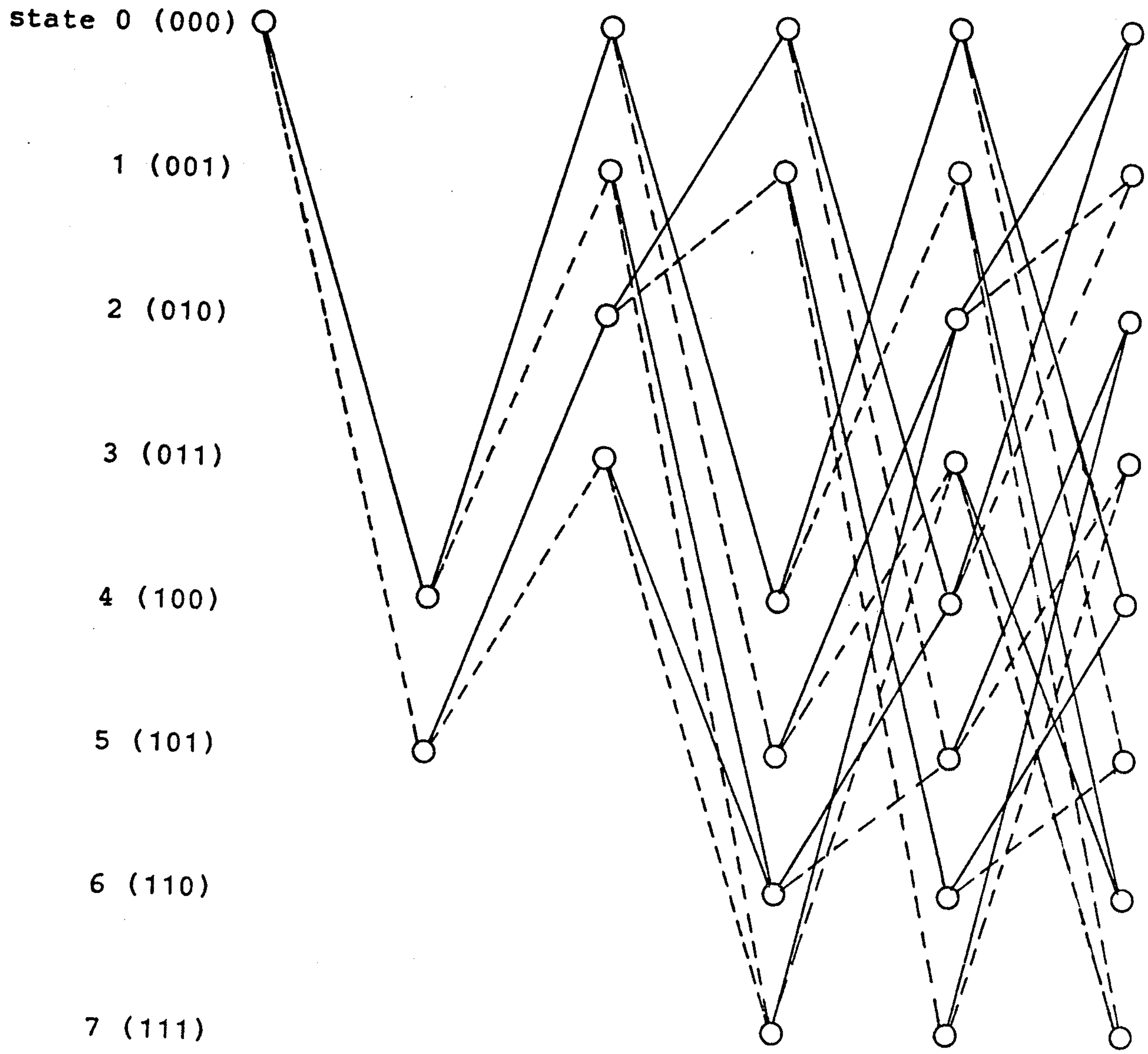


Figure 2.9 (b) Trellis of DMRS with MFM code and L=2 or 3

membered and the most significant digit corresponds to the memory element of the channel. The next most significant digit corresponds to the encoder memory element.

2.4 CONVOLUTIONAL CODE

Since Viterbi's algorithm was originally devised for convolutional code, it would be natural to apply convolutional coding as a means of error correction for any communication system using Viterbi's algorithm. The complexity of the algorithm depends on the number of states as seen from the system trellis. For a DMRS, the number of states increases with ISI, inherent magnetic state, encoder memories and when a convolutional encoder is in cascade with the system, the number of states increases with the constraint length of the code.

To investigate the benefits to be gained from error coding, a simple 1/2 CC was considered. The study was meant to establish how the cascade of the convolutional encoder with the previous encoder affected the trellis; what the final algorithm complexity is and if it is possible to analyze the overall error performance.

The encoder for a 1/2 CC is shown in Figure 2.10(a) and its trellis representation is shown in Figure 2.10(b). When this encoder is applied to DMRS, the overall trellis is as shown in Figure 2.11 and Figure 2.12.

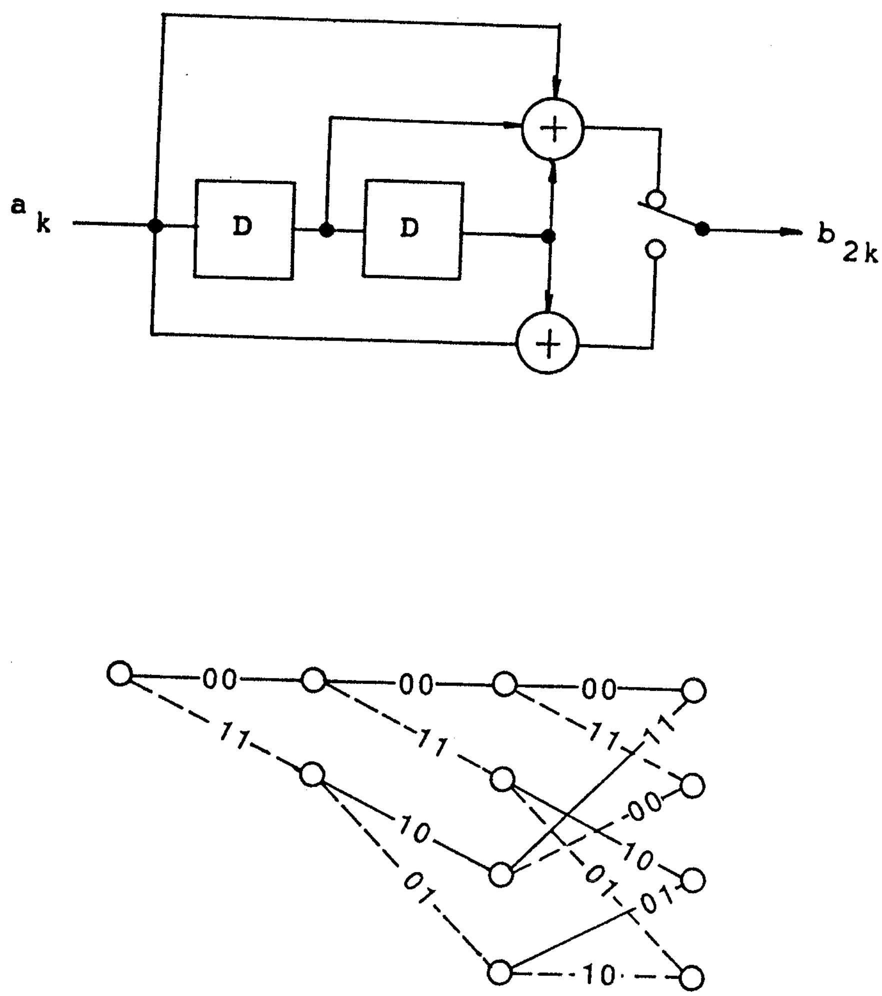


Figure 2.10 1/2 CC and its trellis

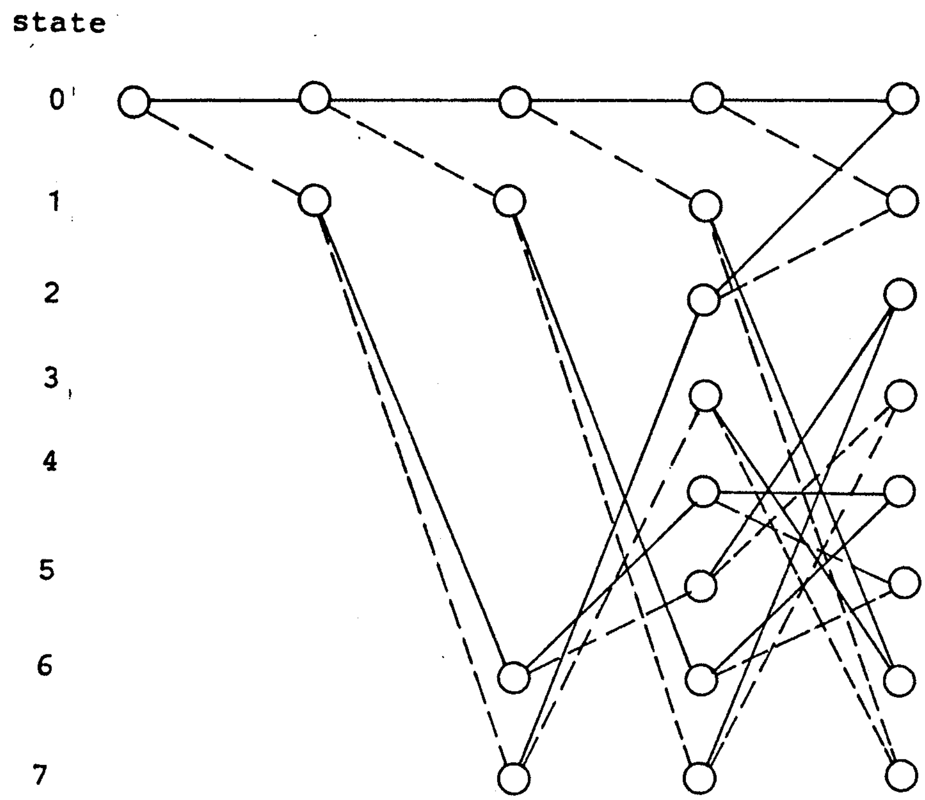


Figure 2.11 Trellis of 1/2 CC with NRZI and L=1

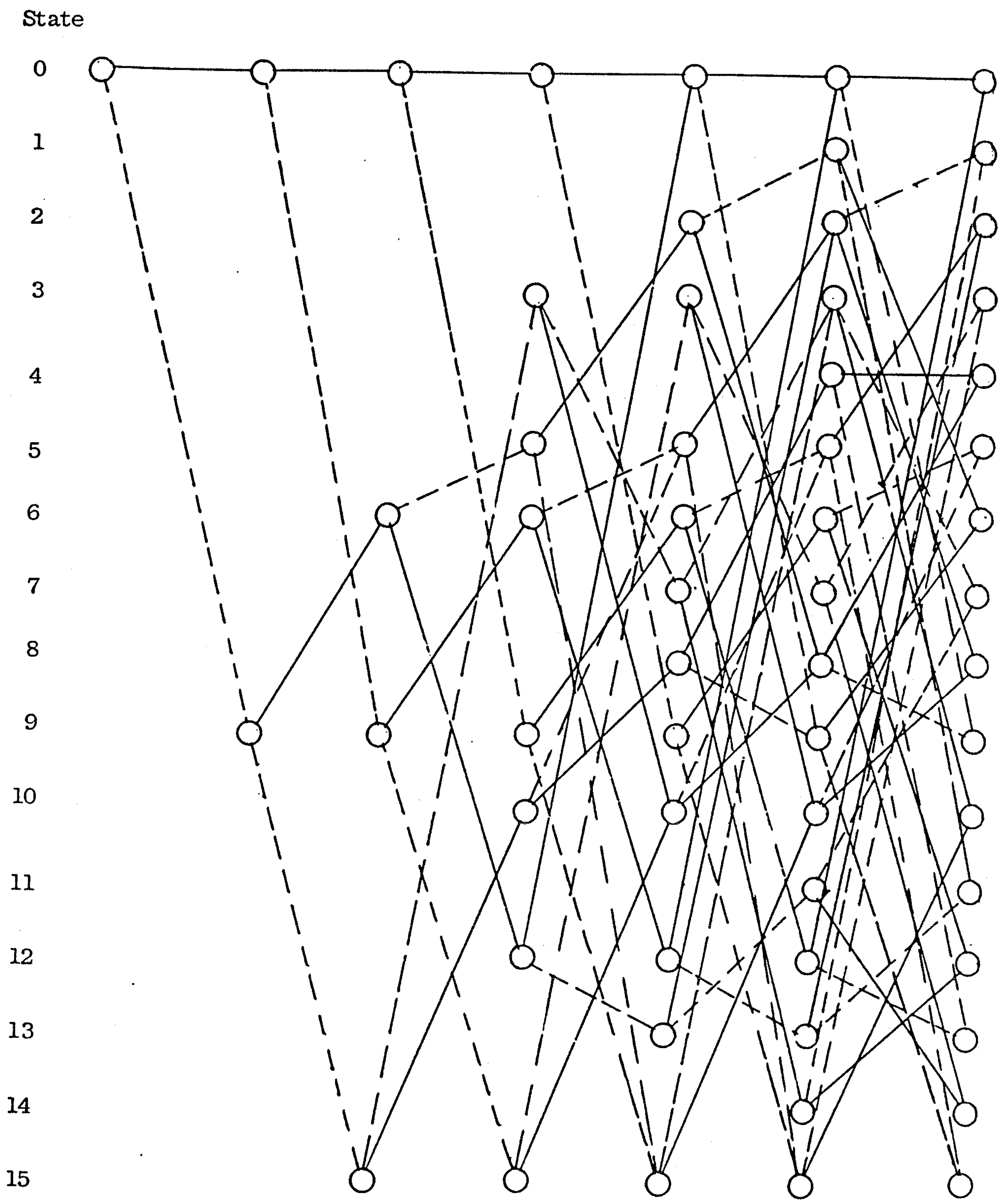


Figure 2.12 1/2 CC with MFM and L=1

The complexity of trellis is seen to increase exponentially with ISI and number of memory elements of the encoder. This affects the difficulty of simulation due to the large increase in computation time.

2.5 VARIOUS DISTANCES OF CODES

The distance measure that is used for block codes is usually the hamming distance which is simply the number of bits that two codewords differ in. The minimum distance which is the smallest hamming distance among the hamming distances of every pair of the codewords determines how well the code performs.

The distance measure used for convolutional code is normally the free distance. For linear convolutional code, the free distance is the hamming distance of the shortest path deviated from the zero codeword. This distance provides an idea of the bit error performance of the code. The shortest path can be obtained pictorially from the trellis. Thus by looking at the trellis, the code performance can be established.

In a DMRS, the combination of all encoders is in general not linear. However using the trellis developed for such a system, it should be possible to define a distance measure which would indicate system performance.

Similar to the concept of the hamming distance for binary symbols, the hamming distance for a ternary PAM system can be reasonably defined as

$$d = \sum |a_i - b_i| \qquad , \text{ for } i=-N,\ldots,N-1 \qquad (2.22)$$

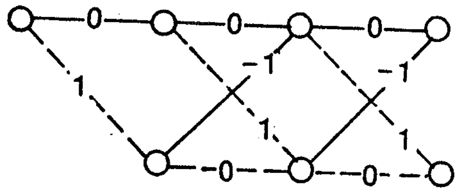
where a and b are elements of two codewords $\bar{a}$ and $\bar{b}$.

The mapping from binary to ternary in the DMRS makes the system nonlinear. In general to evaluate the distance of a nonlinear code is very difficult. For such a code, it is necessary to do a pairwise distance computation between each pair of codewords to determine the minimum distance of the code

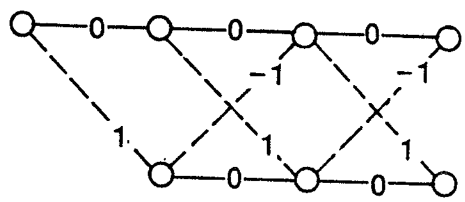
The trellises of NRZ, NRZI and MFM codes are shown again in Figure 2.13. One distance measure for the above codes would be as following:

(1) Collect the complete set of paths that left the zero path and merged back to zero path; there may be infinite number of such paths, in this case try to examine a path that never merged back to the zero path.

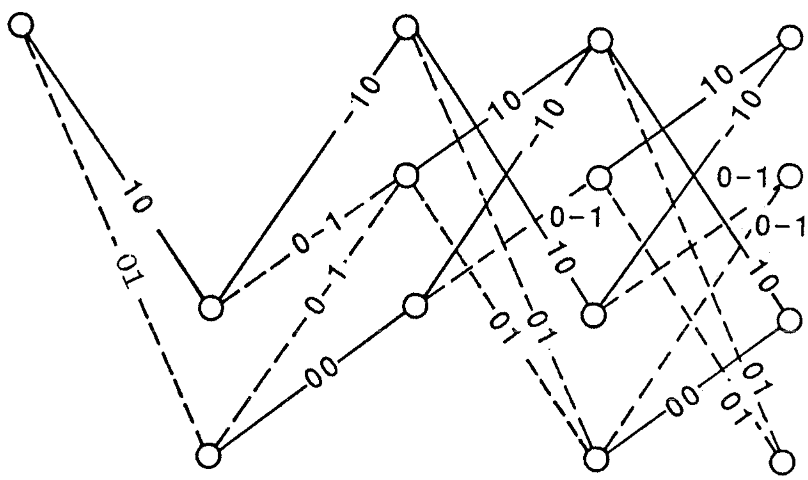
(2) Find all pairwise distances according to equation 2.22 and select the minimum distance; if the minimum distance is from the pair which has one path which never merged back to zero path, the other path is likely to be the same, that is, it is never merged back to the zero path but the equation 2.22 for the pair must converge.



$d_{free} = 2$



$d_{free} = 2$



$d_{free} = 2$

Figure 2.13 Trellis of NRZ, NRZI and MFM for dfree estimation

Another possible measure that eliminates the possibility of having paths that never merge back to the zero path is as follows:

(1) For each state, find the two shortest paths that left the same state and merged back to the same state.

(2) Find also the distance according to equation 2.19 for all such pairs associated with each state.

(3) Also find all such distances among all the states, that is, find the two shortest path that left one state and merged to another state; then compute the distance between these paths. This may be called inter-state distance.

(4) Now compare all the above distance and choose the minimum distance which will provide measure for the performance of the code.

Comparing the three codes in this chapter, MFM is found to provide the best distance among the others. The comparison is shown by Table 2.1.

TABLE 2.1

Distances of various codes

	000	001	010	011	100	101	110	111
000	0							
001	*1,1,2	0						
010	2,1,4	3,2,4	0					
011	1,2,2	2,4,2	1,1,2	0				
100	2,1,5	3,2,5	4,3,3	3,3,5	0			
101	3,2,5	2,3,5	4,3,5	4,2,5	1,1,2	0		
110	2,2,4	3,3,6	2,3,2	3,4,4	2,1,3	3,3,3	0	
111	1,3,6	2,2,4	3,4,4	2,5,2	1,2,3	2,3,3	1,1,2	0

* d_a, d_b, d_c denote distance of NRZ, NRZI and MFM code (3 bits)

Chapter III

DMRS SIMULATION

3.1 INTRODUCTION

The previous chapter developed trellis representations for the magnetic recording channel with various encoding schemes and intersymbol interference. Based on this it would be desirable to predict system performance, particularly for the bit error probability. Ideally this prediction should be valid for the various encoders or combinations thereof, various noise sources and varying amount of intersymbol interference. Though analytical results are available for a few special cases, example - Gaussian noise with no intersymbol interference, there is no general analytical approach to predict the system performance. Even for the special cases the analytical results are in the form of bounds. The other alternative is to implement an actual system - this is not very flexible, and is costly and time consuming.

For these reasons a system simulation was developed. It was first attempted to do the simulation entirely in software since this would provide maximum flexibility. The intention was to provide a simulation system not only for the research in this thesis but also for future research in magnetic recording. Quite a few subroutines were developed in the Pascal language on an Amdahl v7 mainframe. These included routines for source generation, Gaussian, Laplacian and Quartic noise generation, various encoder routines, a matched filter routine and routine for algorithm and system evaluation. Various parameters such as constraint length, code rate, memory length, tap assignment for the convolutional code, number of runs and number of errors accumulated before system halts were allowed.

The simulation however was very time consuming and disappointingly slow. The error probabilities of interest are in the range of 10^{-5} or less. To obtain a reasonable statistical estimate of the error probability for a given system configuration it was decided that at least ten error events should occur. Thus an error probability of 10^{-6} would require approximately 10^{6} input bits. This led to estimated time of at least 2 or 3 months (depending on the coding) to complete the simulation. Thus it was decided to go to a simulation involving both hardware and software. The most time consuming aspects of the simulation would be done by hardware. Though this perhaps reduces the flexibility somewhat it was a necessary step. The hardware/software tradeoffs are discussed the next section.

3.2 TIME - HARDWARE/SOFTWARE TRADEOFFS

Figure 3.1 is a block diagram of the simulation. It is similar to the channel block diagram used in Chapter 2 with each block corresponding to a routine if the simulation is totally in software. The most time consuming block is the passing of the input noise through the matched filter, a process that involves convolution. This is identified in Figure 3.2 by modifying the system to provide two paths - one from noise source and other from signal source, to provide the signal input for Viterbi's algorithm. The convolution requires 100 or more samples to accurately represent the output correlated noise during one input bit interval. The resultant noise sample is a weighted sum of these 100 samples and requires 100 multiplications and 99 additions. As a typical example, a floating point multiplication subroutine requires 85 instructions [15] and assuming the average number of cycles per instruction is 5; it will then require 425 machine cycles. For a machine running at 1 MHz would require over 42.5 milliseconds. This accounts for only one bit sample. A million trials will need 42500 seconds which is over 11 hours of CPU time. This does not include the time for the other routines.

The other time consuming routine is the Viterbi's algorithm itself. Computation of the branch metric requires 1 multiplication and 1 addition operation only but the number of branch metric amounts to 2 times the number of states.

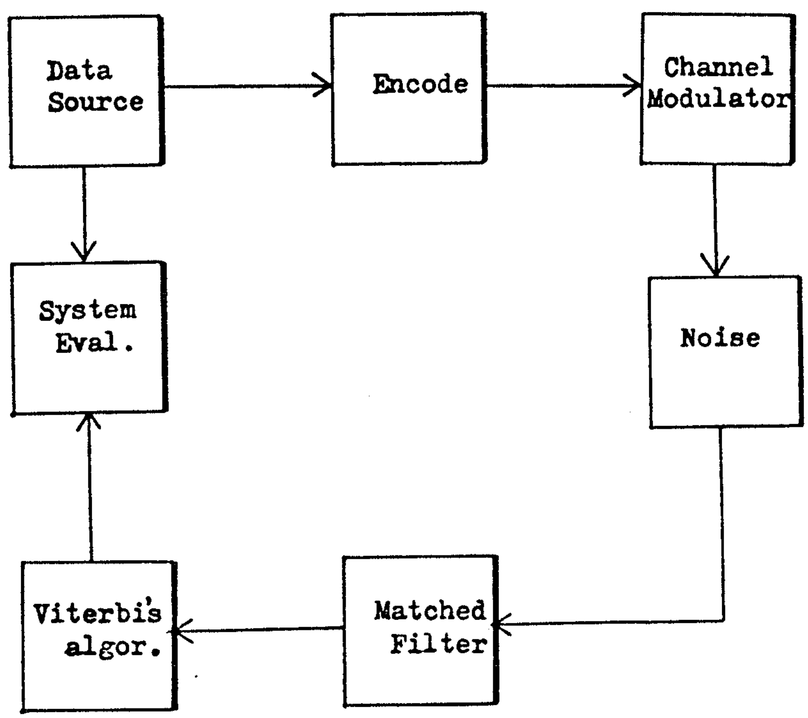


Figure 3.1 Simulation Block Diagram

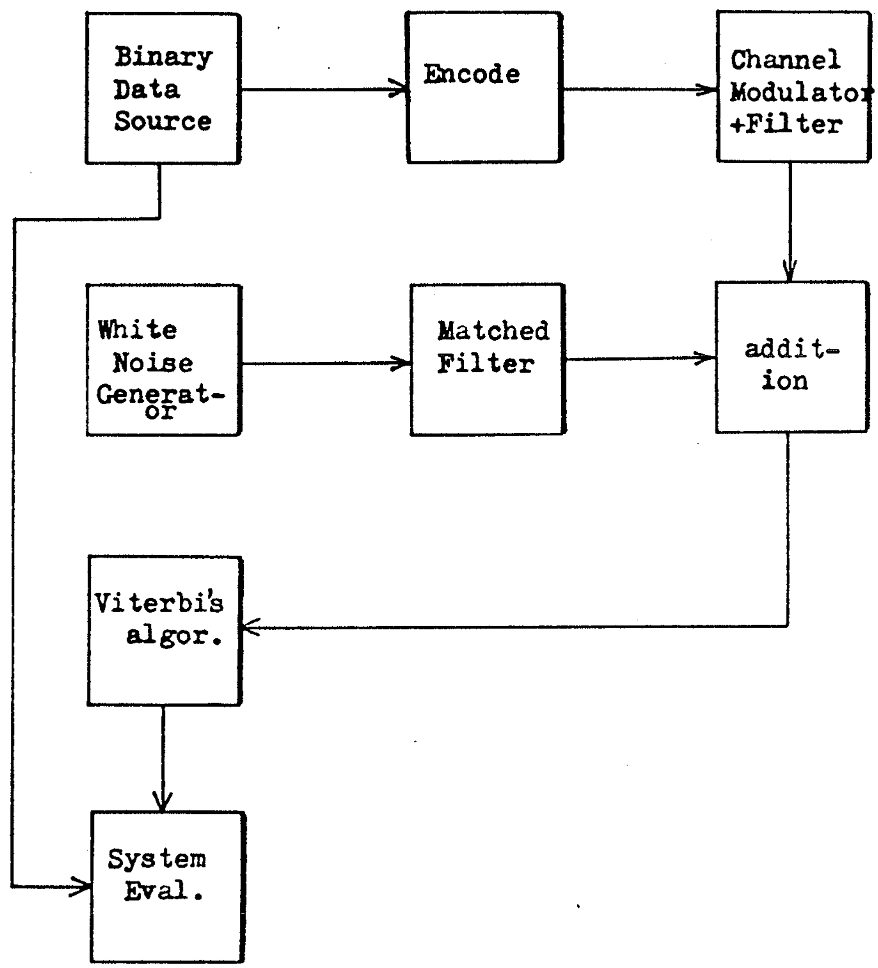


Figure 3.2 Modified Simulation Block Diagram

As mentioned previously the number of states grows exponentially with the number of ISI terms and the number of memory element in the encoder. This means that the time required for computation grows exponentially with the complexity of the system. For this reason simulation runs for the 1/2 CC with MFM and NRZI were not done.

Based on the available facility at the University of Manitoba, a hybrid system was developed. It consisted of hardware and software. The system used is a PDP 11/40 which can run with both compiled high language program and machine program. Although originally the algorithm was written in Pascal, since a Pascal compiler was not available on the PDP 11/40, the simulation was re-written in FORTRAN and the assembly language of the PDP 11/40 system.

To optimize the memory usage and speed, only the main program which makes use of real number manipulation and control was written in FORTRAN, all other subroutines such as source symbol input routine, noise input routine, source symbol buffering, encoding, decoding, source symbol/estimate comparing routines etc., were written in assembly language.

The flowchart for the simulation is shown in Figure 3.3. The original Pascal program is listed in Appendix 1, the new FORTRAN program is listed in Appendix 2, assembly language subroutines are shown in Appendix 3 and the memory allocation listing is shown in Appendix 4.

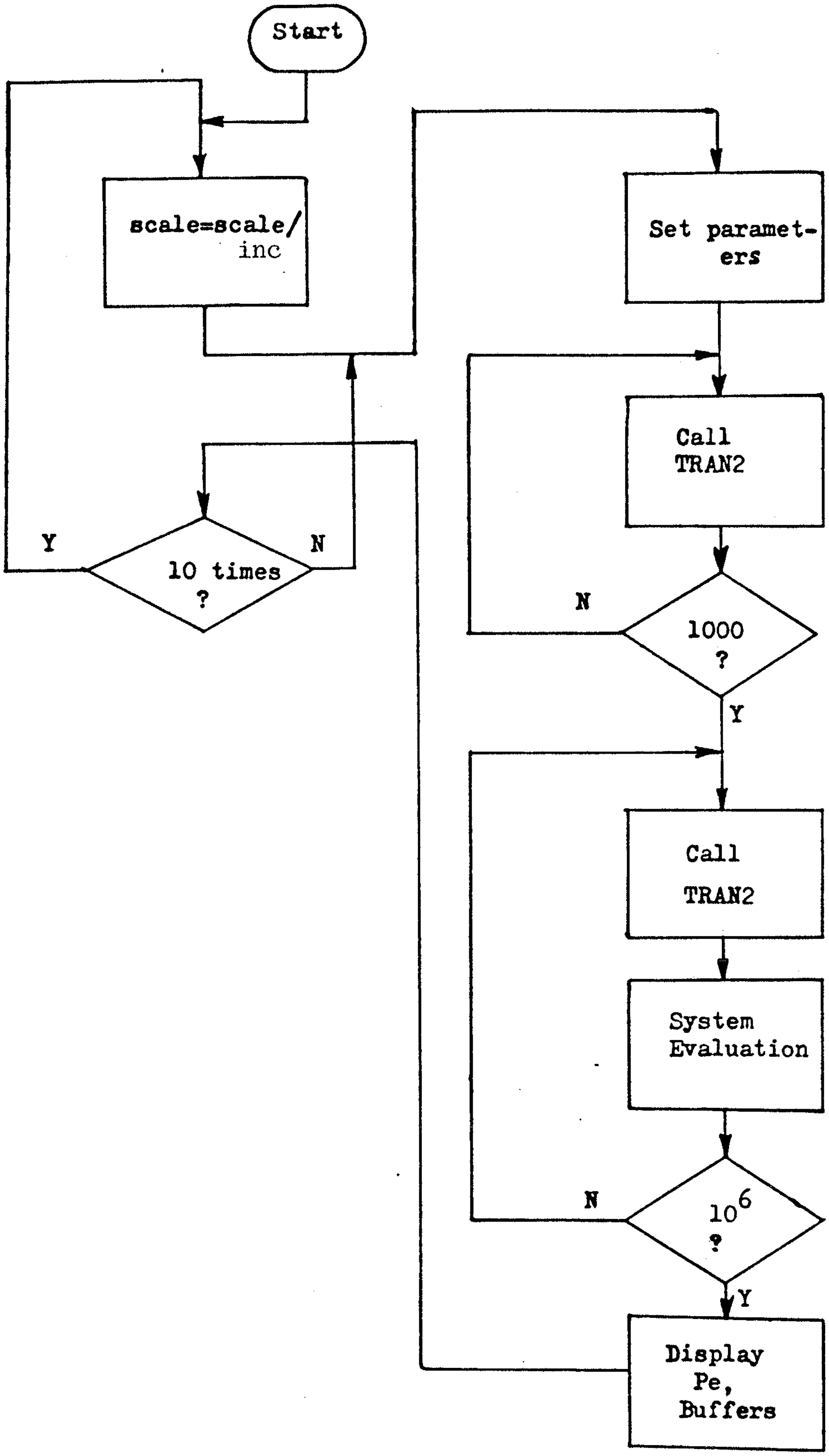


Figure 3.3 Flowchart of Simulation Routine

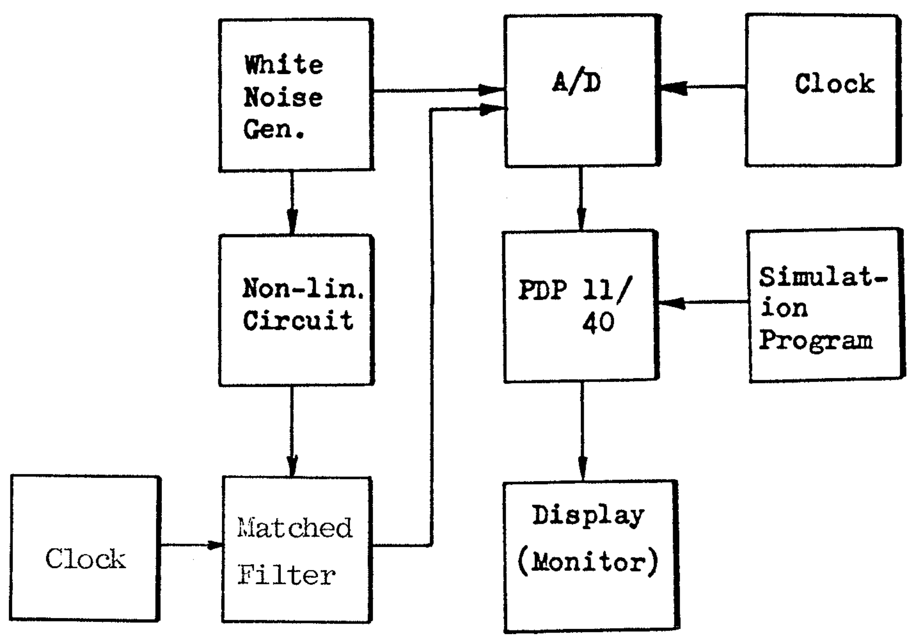


Figure 3.4 Block Diagram of the set-up

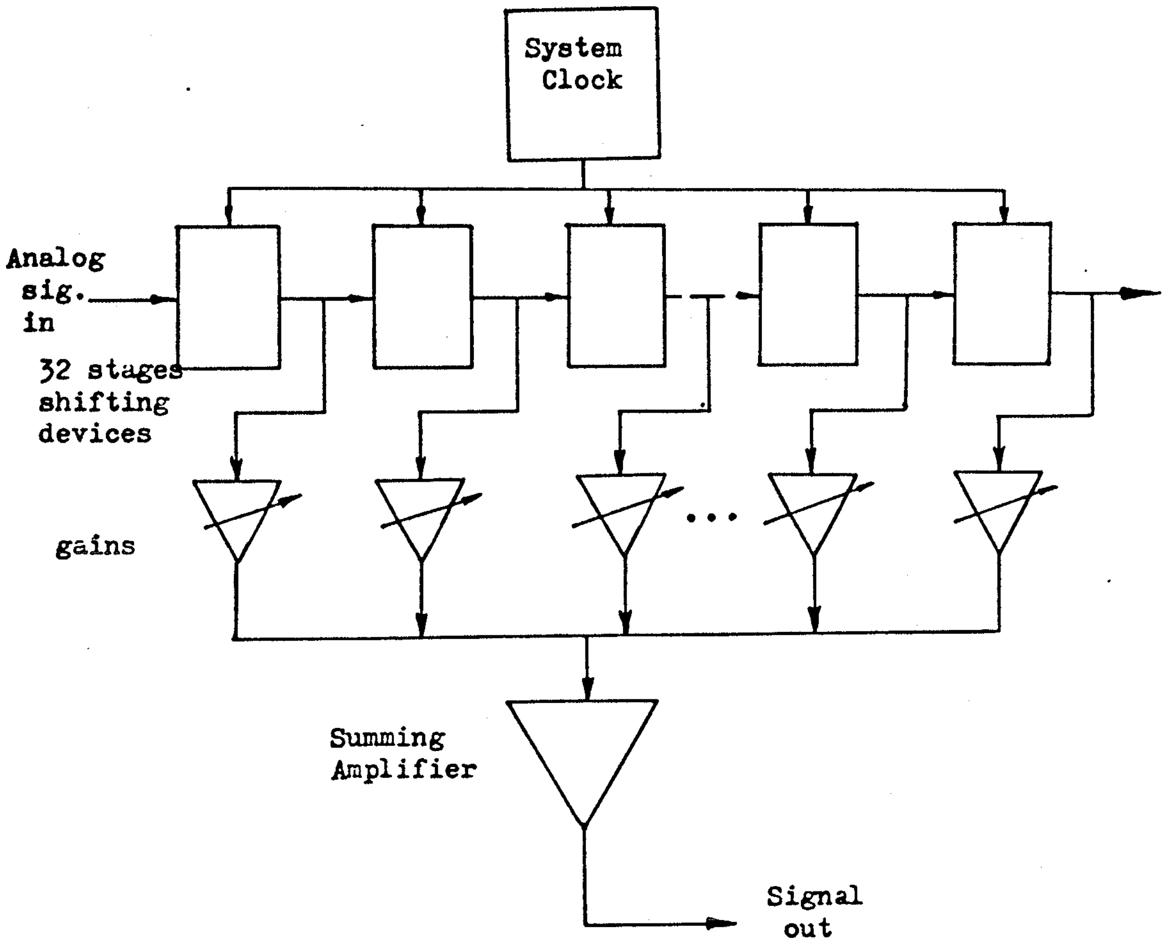


Figure 3.5 Block Diagram of the matched filter implementation

The hardware part of the simulation consists of a white noise generator, a random binary sequence generator (for source), a nonlinear circuit to transform the noise statistics, a matched filter and clocks for timing. A detailed description of the matched filter can be found in the next section. The final system block diagram is shown in Figure 3.4.

3.3 THE MATCHED FILTER IMPLEMENTATION

The matched filter is implemented by using a delayed-line chip TAD-32. It is an analog shifting device which requires a clock to do the shifting. The clock should run at a frequency much higher than the highest frequency of the desired matched filter impulse response. The functional block is shown in Figure 3.5 while the detailed schematic is in Appendix 5.

The chief difficulty with the implementation was the tap gain adjustment to obtain the desired impulse response. Two methods were devised. One way of obtaining a quartic pulse is to observe the matched filter output in an oscilloscope. An assembly program was written to generate a clock to drive the circuit as the system timing. An unipolar (also a bipolar) pulse train is also generated as the signal input for the matched filter for at least 127 clock cycles. The impulse response due to this "impulse" train can be observed through the oscilloscope. Taps are adjusted so that the desired quartic pulse is obtained. Since adjusting the taps upsets the dc bias of the amplifier, constant re-biasing of the amplifer is required. This procedure was time consuming and quite tedious.

The other method is simpler but it required slight modification of the hardware. The weights are first determined by the tap resistors according to the following formula:

$$W = \frac{500 - R}{500} \tag{3.1}$$

where W is the tap weight and $-1<W<+1$ and R is the corresponding resistance.

However, to obtain the weights the tap resistors have to be isolated from each other to prevent loading and are then adjusted simply with an ohmmeter.

There are a few other adjustments for the matched filter circuit. The filter is made up of gates which have to be biased in order to operate in the linear region. To do this a sine wave is applied to the input and the signal output is monitored with an oscilloscope until a symmetrical signal obtained at the output of the feed forward pin. This is then followed by adjusting the input and output dc bias. Also the two phase clocks have to be adjusted to obtain identical amplitude at the output.

3.4 NOISE GENERATION

A HP 3722A white noise generator was used to generate white Gaussian noise. Since two other noises are required, a nonlinear circuit was used to transform a gaussian random variable to the desired random variable. To explain the method for obtaining the nonlinear transformation, consider Figure 3.6.

In general, assuming a monotonic nonlinearity, one can write the following relationship between input x and output y as follows,

$$p(x)dx = p(y)dy \qquad (3.2)$$

If x is uniform over interval (0,1), then

$$\frac{dx}{dy} = p(y)$$

or
$$x = P_y(y)$$

or
$$y = P_y \ (x) \qquad (3.3)$$

where p(y) is the density function of y, p(x) is density function of x, $P_y(.)$ or P(y) is the distribution function of y with variable enclosed in the bracket, or simply distribution function of y.

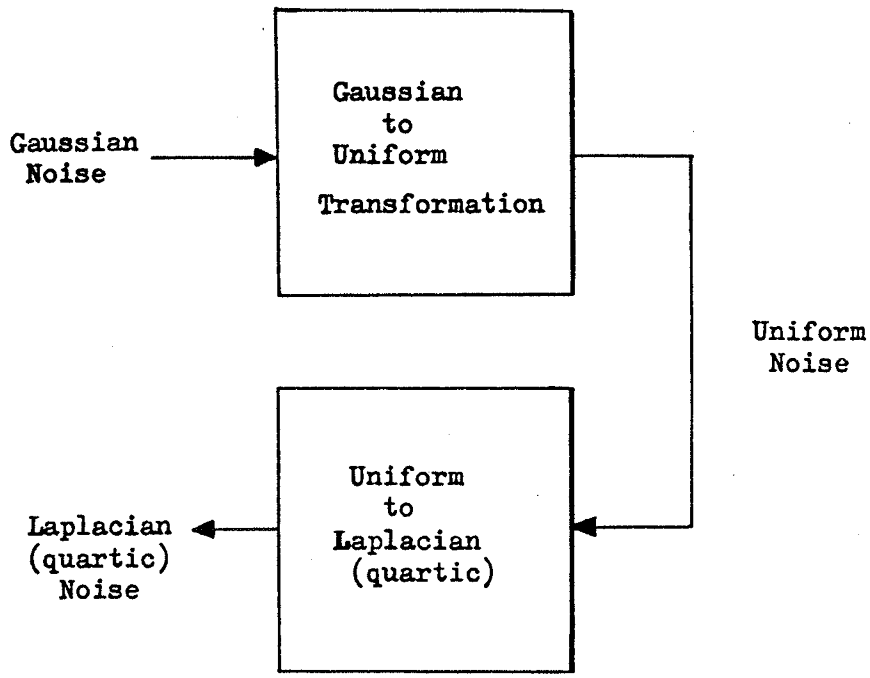


Figure 3.6 Nonlinear transformation block diagram

From equation 3.3, the transformation that maps one random variable with a specified density function to a random variable with uniform density is the inverse of the distribution function of the variable itself. Obviously to transform a Gaussian random variable to Laplacian random variable, one can find the distribution of the gaussian random variable evaluated at x and find the distribution of the laplacian random variable evaluated at y where both x and y have an identical distribution. The corresponding (x,y) pairs are tabulated in Table 3.1 and 3.2 for the nonlinear transformations from gaussian to laplacian and quartic.

A nonlinear circuit based on the above transformation was implemented as shown in Figure 3.7. The nonlinear functions are also shown in Figure 3.8 and 3.9.

3.5 THE SOFTWARE

3.5.1 High Level or Machine?

In order to speed up the simulation, machine language should be used. In the algorithm implementation, the branch metric computation needs to be done by using real number representation, actually in floating point, to eliminate overflow problems. To do this in assembly language requires tedious re-writing of the real number arithmetic routines. A compromise solution is to write the simulation program in both machine and high level languages. The PDP 11/40's FORTRAN compiler was used along with the linker program so that the machine and FORTRAN program could share the same resources.

3.5.2 Main Program

The main program first set up a scale for the signal to noise ratio. Every 10 times, the scale was reduced by a constant. The main routine was repeated independently 10 times in which all parameters and variables were initialized. The inter-state connections were determined by calling a macro subroutine INDn (n = 1, 2, 3, or 4 depending on the coding). Fixed branch metric terms were also calculated for future look-up (they depend also on the coding). The next step was to initialize the state metrics by calling subroutine TRAN2 a 1000 times. The path history depended on the state metrics. Next the simulation subroutine TRAN2 was called again, followed by the system evaluation routine. When either the accumulated errors exceeded 10 or a million runs were achieved, the system displayed the probability of error, input buffer and decoded path buffer and stopped.

One routine worth mentioning is that it pre-calculates the fixed term of the branch metric. The fixed branch metric calculation depends on equation 3.4 which is the fixed term of equation 2.18:

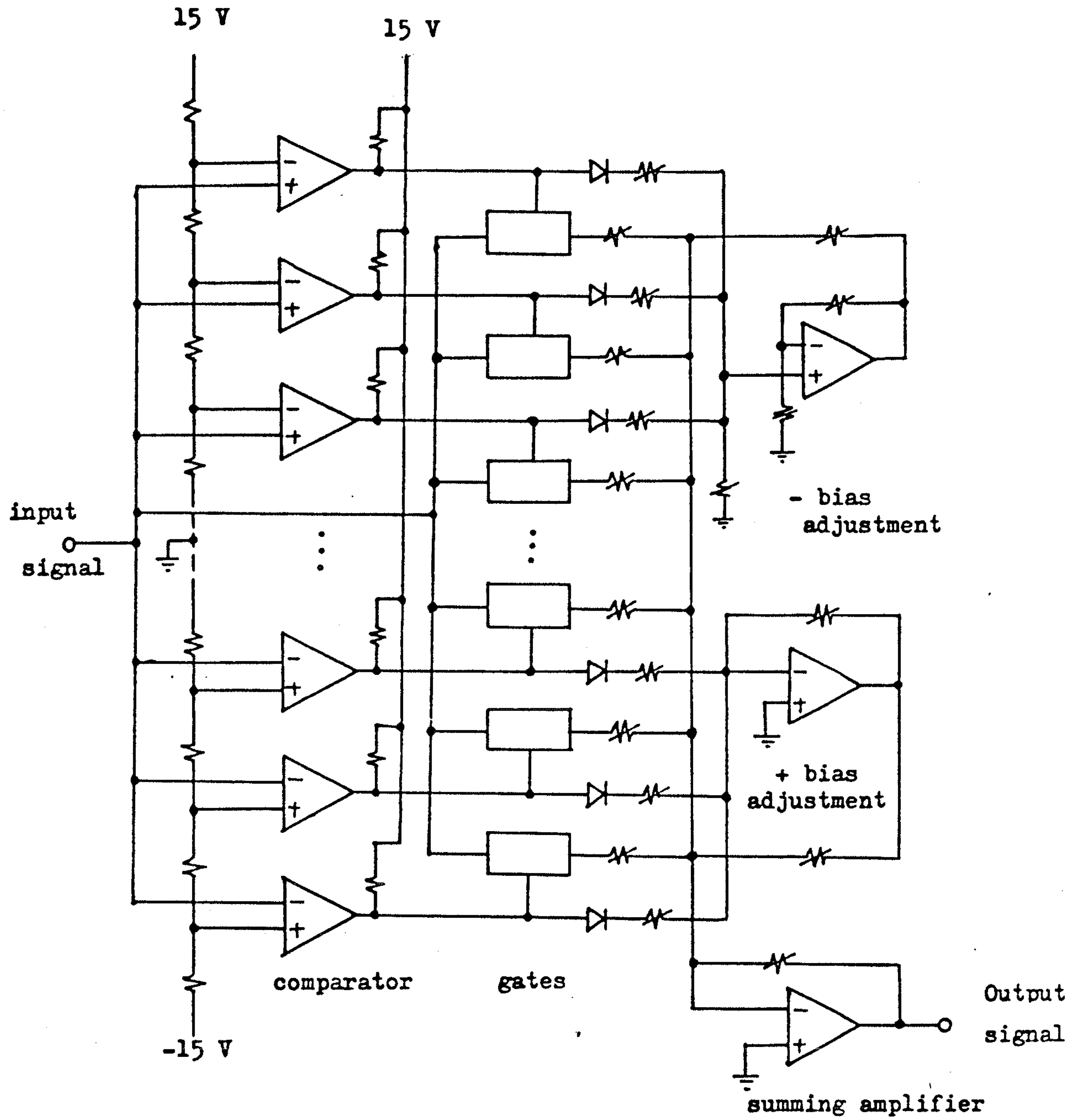


Figure 3.7 Nonlinear circuit implementation

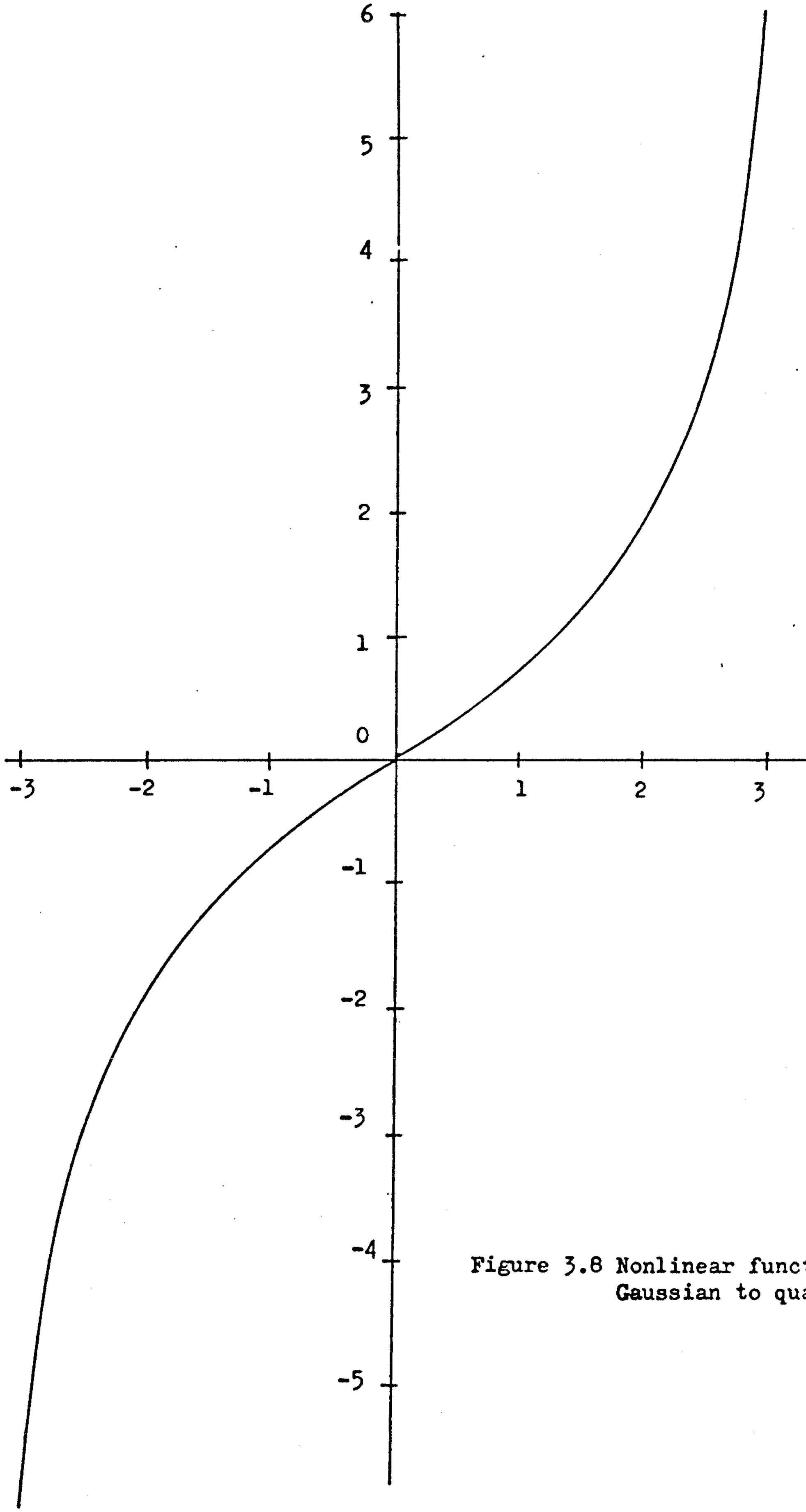


Figure 3.8 Nonlinear function - Gaussian to quartic.

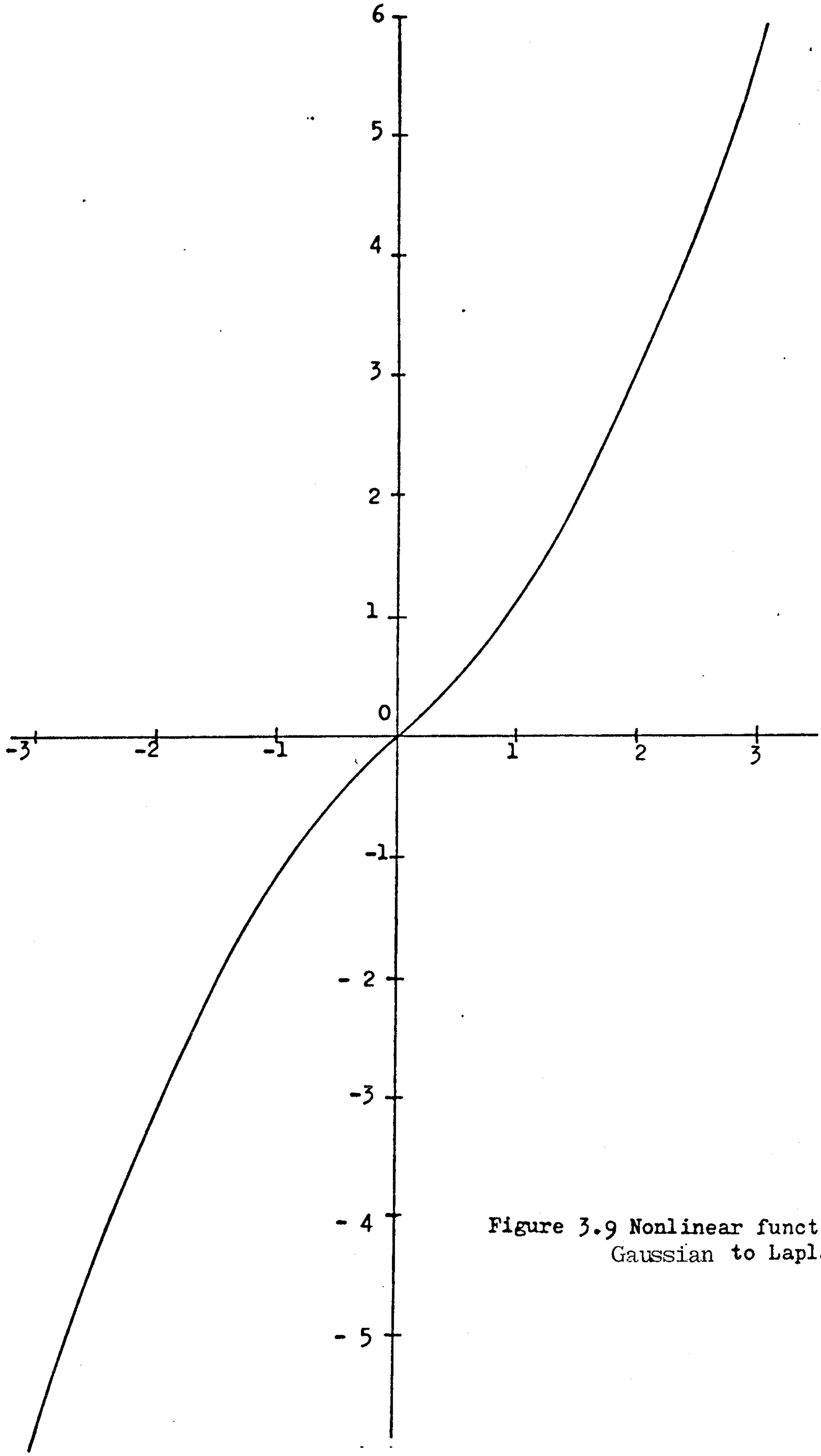


Figure 3.9 Nonlinear function - Gaussian to Laplacian.

TABLE 3.1

Gaussian, uniform, and quartic transformation

Gaussian	Uniform	quartic
-3.00	0.0013	-5.1094
-2.50	0.0062	-3.0234
-2.00	0.0228	-1.8918
-1.50	0.0668	-1.2129
-1.00	0.1587	-0.7462
-0.50	0.3085	-0.3631
0.0	0.5000	0.0000
0.50	0.6915	0.3631
1.00	0.8413	0.7462
1.50	0.9332	1.2129
2.00	0.9772	1.8918
2.50	0.9938	3.0234

TABLE 3.2

Gaussian, uniform, and Laplacian transformation

Gaussian	Uniform	Laplacian
-3.00	0.0013	-5.9145
-2.50	0.0062	-4.3885
-2.00	0.0228	-3.0900
-1.50	0.0668	-2.0128
-1.00	0.1587	-1.1479
-0.50	0.3085	-0.4828
0.0	0.5000	0.0000
0.50	0.6915	0.4828
1.00	0.8413	1.1479
1.50	0.9332	2.0128
2.00	0.9772	3.0900
2.50	0.9938	4.3885

$$\lambda_f = c_k^2 h_o - 2c_k \sum_{i=1}^{L-1} c_{k-i} h_i \tag{3.4}$$

The fixed term of branch metric is different for different codes since they have different ternary digits c_i. Four routines were developed to calculate the fixed branch metric term for the 4 different encoding schemes. These were NRZI, MFM, 1/2 CC with NRZI and 1/2CC with MFM which were named code 1, code 2, code 3, and code 4 respectively.

The connection of the states (present and next state) were determined by calling INDn (n = 1, 2, 3 or 4 corresponding to code 1, code 2, code 3, and code 4) and saved in memory for look-up. This helped to save computation time. This was written in machine language. The connectivity of each individual state was determined by searching technique. (see Appendix 3)

After calling TRAN2, a routine which is discussed in the next section, the system evaluation routine starts to compare the input symbol and the decoded symbol. This is done by first finding the path with maximum state metric. A macro COMP was called to compare the most likely path (path with maximum state metric) at a particular bit position with the last bit of the input buffer. If an error was detected it was passed back to system routine after updating the error count. If the error count exceeded 10 the system calculated the error probability and displayed the results. It then went back to the beginning and repeated the simulation again with another scale factor. Eventually the error probability will be zero for a million trial due to noise reduction.

3.6 SUBROUTINES

The task handled by the main program is primarily the setting of parameters, the initialization of all variables, calling the simulating program for the required number of times, and doing the system performance evaluation, that is, counting errors. Thus it is principally a controller program.

The actual simulation is done by subroutine TRAN2 which is called by the main program. TRAN2 calls other subroutines which are all written in assembly language. They include INPUT, CODEn (n=1,2,3, or 4), GETX, DECODE, EXCHANGE, and PRT. Only one assembly subroutine INDn (n=1,2,3, or 4) is called by the main program TRAN.

TRAN2 is written in FORTRAN because it does real number arithmetic for the branch metric. It starts by calling INPUT which generates a random binary input into the input buffer U. The INPUT subroutine takes the white gaussian input, converts the sample to binary symbol by simple thresholding and stores the symbol into memory for future comparison. Based on the data in the input buffer, TRAN2 calls CODE1(2,3 or 4) to do the encoding. The encoded symbol(s) is(are) stored in buffer V. This represents the binary data on the magnetic medium. Next the subroutine GETX is called. This generates the ternary digits x_i. The digits x_i in the X buffer are taken to generate the signal sample S based on the matched filter coefficient H(I). Received sample Y (Y=S+RNOI, RNOI is a noise sample in real number representation) is then inputted to the receiver for decoding. TRAN2 continues to do the branch computation and then the add, compare and select operation. The DECODE subroutine determines the input by choosing the path with maximum metric. The estimate is stored in the path buffer PATH. The next routine for TRAN2 updates the next state to the present state and the whole process is repeated. Details of the program can be found in Appendix 2 and 3.

3.7 TRUNCATION OF ISI TERMS

When the Pascal program was implemented, the decoding algorithm did not appear to work properly. It was found that the all 1's sequence with NRZI coding, that is, an alternating 1 and -1 sequence in ternary form generated excessive errors. To explain this, consider the second term of equation 2.15, this term is the fixed branch metric mentioned (also in equation 2.18). If a message {1, -1, 1, -1, 1, -1, 1, -1} was sent, the signal c(t) corresponding to this message has the energy term as follows:

$$\int_{-\infty}^{\infty} c^2(t)\, dt = \int_{-\infty}^{\infty} [h(t+4T) - h(t+3T) + h(t+2T) - h(t+T) + h(t) - h(t-T) + h(t-2T) - h(t-3T)]^2\, dt$$

$$= \int_{-\infty}^{\infty} [h^2(t+4T) + h^2(t+3T) + h^2(t+2T) + h^2(t+T) + h^2(t) + h^2(t-T) + h^2(t-2T) + h^2(t-3T)]\, dt$$

$$+ 2 \int_{-\infty}^{\infty} \{h(t+4T)[-h(t+3T)+h(t+2T) - \ldots] - h(t+3T)[h(t+2T)-h(t+T) + \ldots] + \ldots - h(t-2T)h(t-3T)\}\, dt$$

$$\geqslant 0 \qquad (3.4)$$

and for L=2, the following inequality holds:

$$8\, h_0 - 14\, h_1 \geqslant 0$$

or

$$h_0 \geqslant \frac{7}{4}\, h_1 \qquad (3.5)$$

However, due to the truncation (in fact L > 2),

$$h_1 > \frac{4}{7}\, h_0 \qquad (3.6)$$

Thus to prevent errors, the above check on the inequality should be done to ensure that

$$h_0 \geqslant \sum_{i=1}^{L-1} w_i h_i \tag{3.7}$$

where w_i are constants and to be determined for a particular message (of any length).

On the other hand, even without the mentioned truncation error truncating the coefficients h_i will possibly introduce certain random noise to the system which may account for a further increase of error probability.

Chapter IV

RESULTS AND DISCUSSION

4.1 DESCRIPTION OF RUNS

Originally 12 runs were selected for the simulation which included the combinations of 3 different noises and 4 different codes. However the program run time for the convolutional code with intersymbol interference took too long, therefore only 2 codes were simulated. These were NRZI and MFM. The NRZI code was simulated with different ISI to investigate the deterioration in error probability with increased ISI.

As mentioned in Chapter 3, the complexity increases exponentially with ISI and the number of memory elements in the code. This further restricted the simulation to estimating error probabilities of 10^{-6} or greater. Experimentally it was found that the software processed one input bit in about 10 milliseconds for NRZI coding and L=4 ISI terms. For MFM, since there was one more memory element and the whole process had to be done twice for every input bit, this time was found to be 14 milliseconds. Time for processing one bit when 1/2 CC with MFM and ISI were simulated was approximately 100 milliseconds.

For any level of signal to noise ratio the error probability was estimated by inputting bits until ten errors occurred. The error probability was then estimated simply as the ratio of the number of errors (10) to the number of bits processed. Thus for a estimated error rate of 10^{-6} the number of processed bits is 10^{6}. The simulation run times are then 28 hours, 39 hours and 280 hours respectively for NRZI with L=4, MFM, and 1/2 CC with MFM and ISI coding. Also when the simulation is repeated again for 10 times, the required hours will be multiplied as much.

The results of the runs are tabulated in Table 4.1. The probability of bit error is plotted in Figure 4.1 for NRZI with L=4. For MFM and NRZI coding (L=4 for MFM and L=2 for NRZI) it is plotted in Figure 4.2. Interpretation of the results is in next section.

TABLE 4.1

Simulation results

NRZI Code, Gaussian noise, L= 4	
Signal to noise	Probability of error
16.4 db	0.207
19.9	3.3×10^{-2}
23.4	4.03×10^{-3}
26.9	2.59×10^{-4}

NRZI Code with L= 2			
Signal to noise	Gaussian	Laplacian	quartic
9.2	3.45×10^{-2}	3.88×10^{-2}	3.7×10^{-2}
12.4	3.34×10^{-3}	4.2×10^{-3}	3.5×10^{-3}
14.0	5.85×10^{-4}	6.5×10^{-4}	1.2×10^{-4}

MFM Code with L= 4			
Signal to noise	Gaussian	Laplacian	quartic
13.6 db	5×10^{-2}	6.4×10^{-2}	5.8×10^{-2}
15.2	7.76×10^{-4}	4.95×10^{-4}	1.04×10^{-3}
16.8	$< 10^{-6}$	$< 10^{-6}$	$< 10^{-6}$

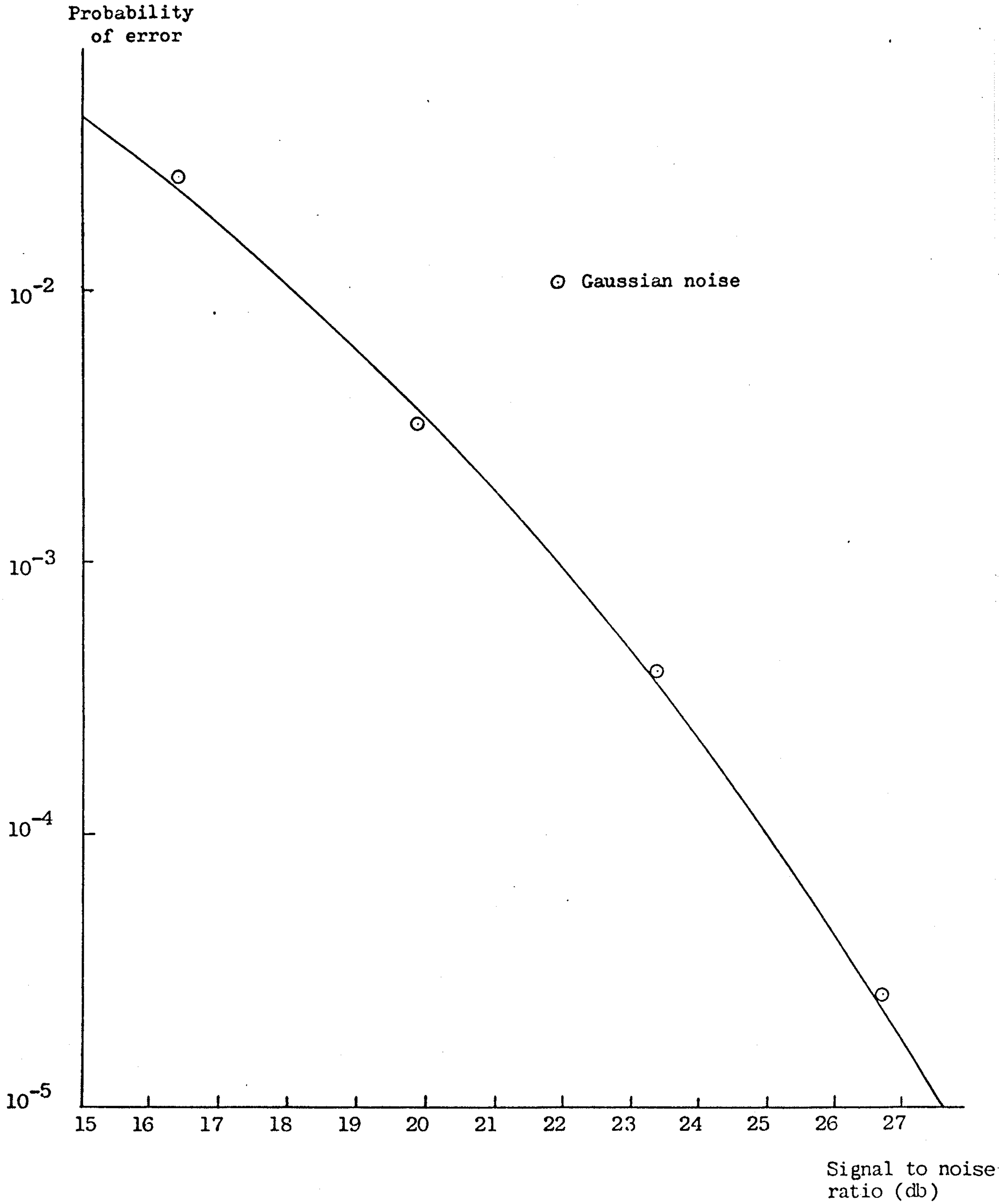


Figure 4.1 Probability of bit error for NRZI with L=4

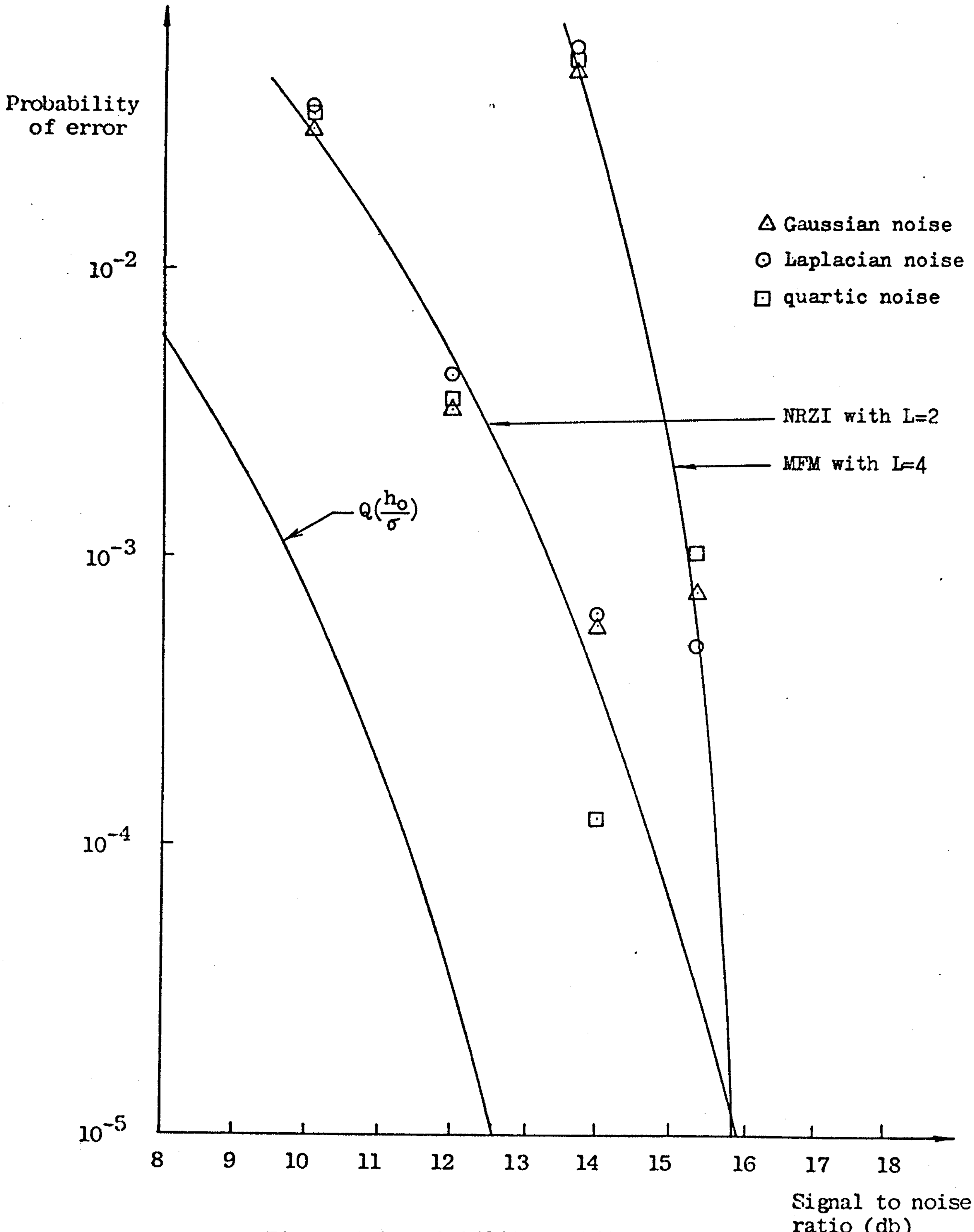


Figure 4.2 Probability of bit error for NRZI and MFM with L=2 and L=4 respectively.

4.2 INTERPRETATION OF THE RESULTS

In general the results are reasonable. Some observations are worth noting. Consider Figure 4.2, at an error rate of 10^{-5}. The deterioration is about 3 db for L=2. Considering Figures 4.1 and 4.2 together, the deterioration is over 14 db for L=4 (in both cases, NRZI coding was referenced). Different noises do not result in any appreciable difference of system performance. The relative performance of MFM and NRZI depends on the SNR. At lower SNR, NRZI has a better error probability. With increasing SNR, MFM becomes better. (The comparision is based on the same input bit rate and ISI)

To determine the confidence limit of the simulation results, consider the analytical approach described in [16]. For an arbitrarily small ε,

$$P\left(|p - \hat{p}| < \varepsilon \right) > 1 - \frac{1}{4 n \varepsilon^{2}} \tag{4.1}$$

where p is the actual probability of error, $\hat{p}$ (or k/n) is the experimental probability of error, n is the number of trials, in this case 10^7, k is the number of errors occured, and ε is the confidence arbitrary range.

Since the confidence limit cannot be less than 0, ε should be chosen properly. For example, if ε is 10 percent of the error probability in interest, then at 10^{-2} error rate, the confidence limit is greater than

$$1 - \frac{1}{4\,(10^{7})[(0.1)(10^{-2})]^{2}} = 97.5\,\%$$

Thus to have the same confidence limit, the number of trials, n should be greater than 10^{15} for the range of error probability 10^{-6} . This however is not feasible at this stage of work. The results however provide insight into the desired estimation of error probability.

4.3 NONLINEAR INTERSYMBOL INTERFERENCE

All the work in the thesis is based on the assumption that ISI is linear and linear superposition holds. With higher densities, linear ISI is no longer valid and a different model is required to represent the system. Viterbi's algorithm however can still be applied provided that the nonlinear model can be established. A basic nonlinear model is introduced in this section along with the metric calculation for it.

Consider a typical bit crowding effect as shown in Figure 4.3. Two assumptions are made: first, the nonlinear effect only affects the amplitude and not the shape of neighbouring pulses; second, only the immediate neighbours are affected.

With the above assumptions, the memory is increased by one. The branch metric calculation can be illustrated by the following example. Consider a channel memory L=2 for the DMRS model as shown in Figure 4.4. The corresponding branch metric is calculated as follows:

State c_1c_0 in linear ISI will be equivalent to state $c_2c_1c_0$ where c_2 is the incoming bit and is equal to c_0 of next state;

$$\lambda_k = 2c_0y_k - c_0^2h_0(c_2,c_1) - 2\sum_{i=1}^{1} c_ih_iw_i$$

where w_i is a constant which depends on the neighbours of c_i. c_i's are assumed ternary symbols.

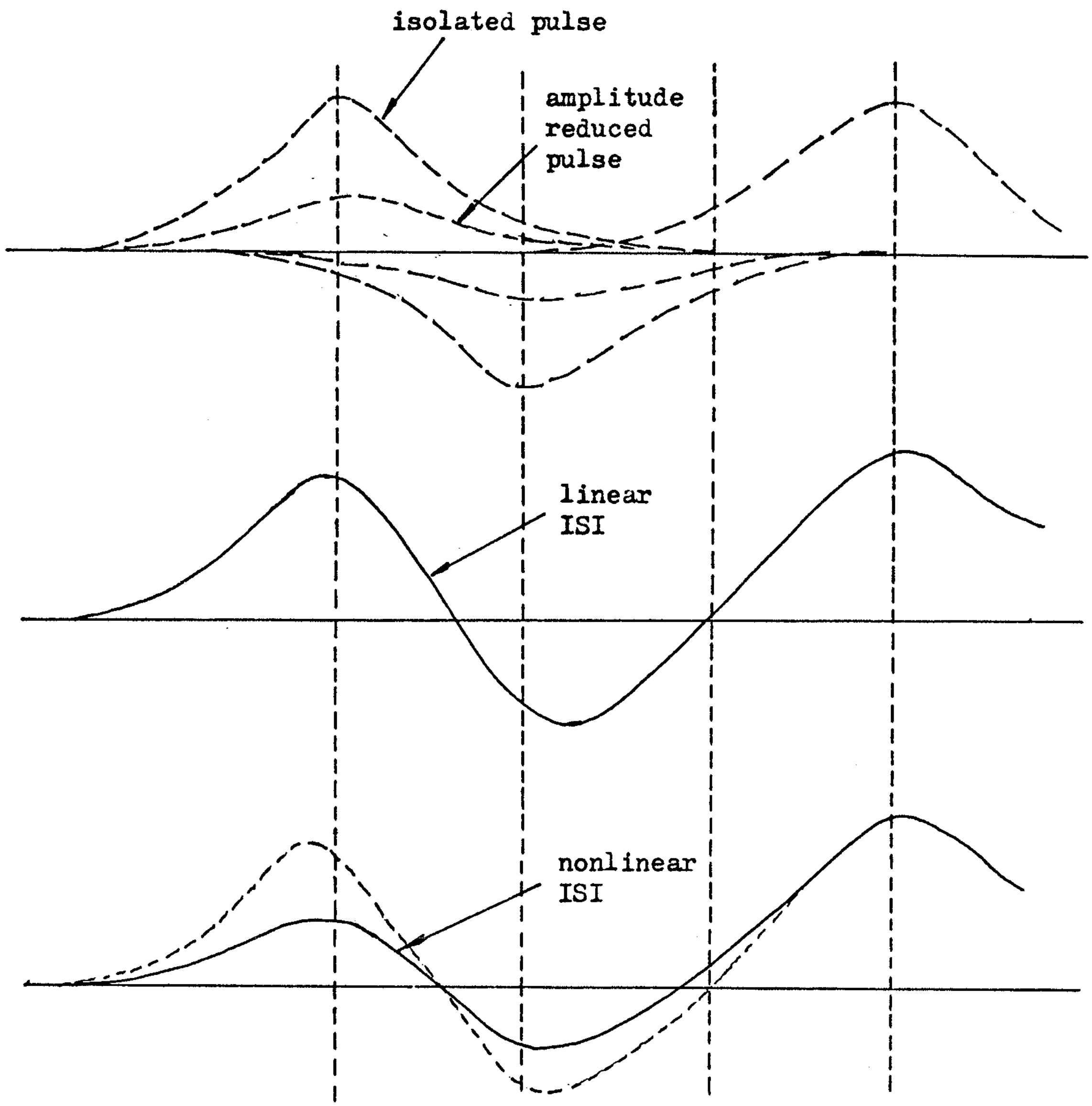


Figure 4.3 Bit-crowding showing nonlinear effect of ISI

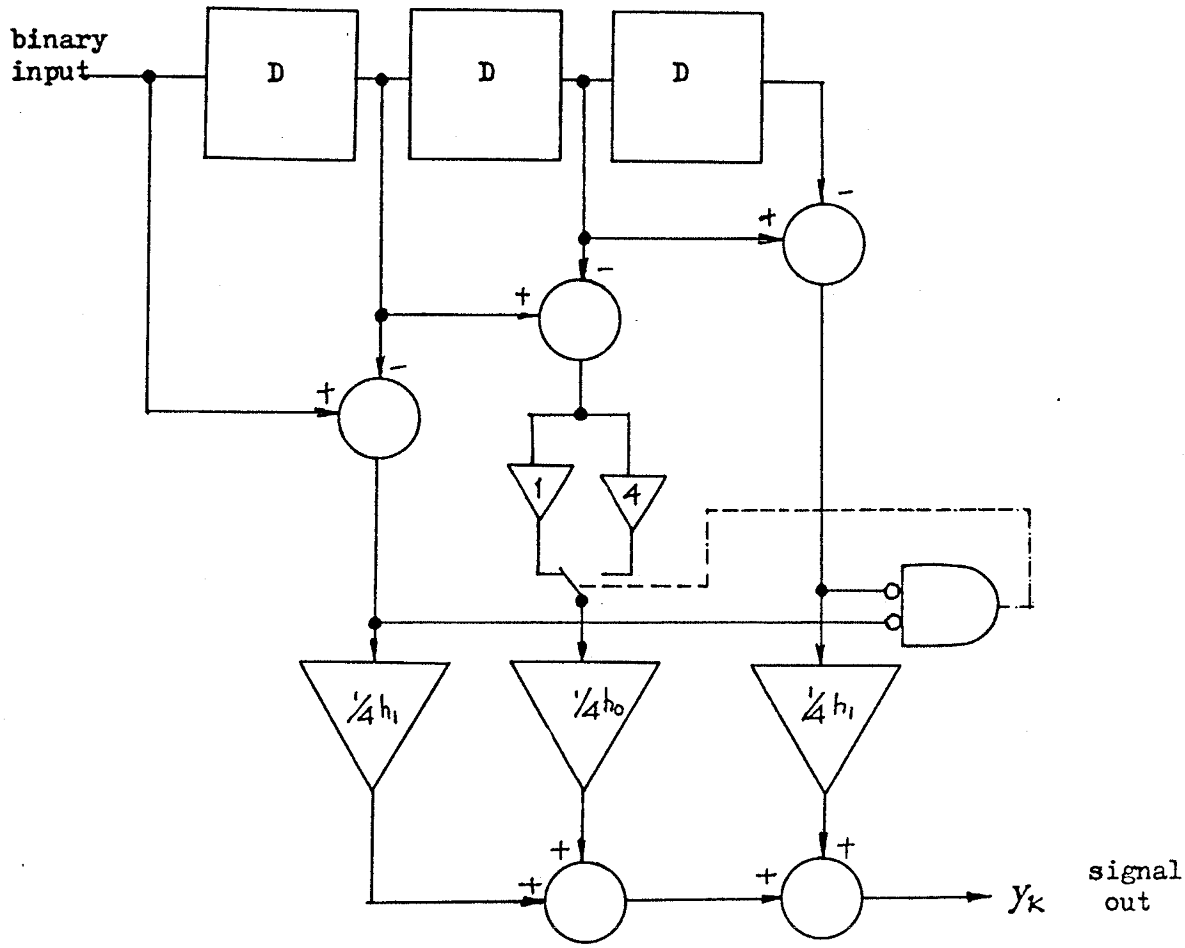


Figure 4.4 Model of Magnetic Channel showing nonlinear ISI effect

The above has illustrated for a simple nonlinear ISI channel how Viterbi's algorithm can be applied. The test of the validity of this model is left for future research.

4.4 SUMMARY

A general communication model has been set up for the DMRS for future research. Software simulation for the model has been implemented on PDP 11/40 and basically written in FORTRAN. Most of the routines have been produced for future use. Depending on the structure of DMRS to be researched after, routines in machine language can be added on for the study. The system is not "user - friendly" but for a user with sufficient background, there should be no difficulty in adjusting parameters such as different code rate and different constraint length, even different codes may be implemented.

Various routines have been isolated so that depending on the system under research, they are available from the calling program. However some routines such as branch metric calculation for different code would require modification as they are imbedded inside the main program.

It would be worthwhile to make use of the present model to develop a "friendly" operating system that would automatically structure the program for the simulation by choosing different code rate and different code, and other parameters required. For example, if a simulation of the DMRS with an NRZI code, constraint length L=3, gaussian noise, S/N = 10 db, is desired, the program would automatically set this up, run and output the system error probability.

There is another way of doing the simulation with speed, that is to have real time simulation. All the components in the simulation can be implemented with hardware. This, however, will limit the flexibility of the system.

Chapter V

CONCLUSION AND RECOMMENDATIONS

This thesis has considered the performance of a digital magnetic recording system from a communication theory point of view. In particular the effect of intersymbol interference and different coding schemes were studied, both analytically and through simulation. In the author's opinion there are three major contributions by the thesis. The first is the model development. Previous models had not dealt with the intersymbol interference. Most have tried to equalize the effect of the differentiation process. In high density DMRS this equalization is redundant. Second is the introduction of inter-state distance concept. Previous distance measures for convolutional code such as d_{free} can only be used to evaluate the performance of a linear code. For a nonlinear code, such as MFM code, which is widely used in DMRS, this measure is not applicable. The inter-state distance measure greatly enhances the analysis of nonlinear code performance. Lastly the program developed can be used for future research on DMRS. For example nonlinear ISI or nonlinear codes can be studied using the developed program.

There are also some recommendations. Since convolutional codes are decoded by Viterbi's algorithm, it would be of interest to evaluate the performance when applied to the DMRS. Although the time required for the simulation holds back running the program, there may be another way such as a hardware simulation to evaluate the system performance. Further for an extremely high density DMRS, it is inevitable that the nonlinear model must be used. Future research on the validity of the model introduced is necessary.

REFERENCES

[1] Sierra, H. M., "An analytical estimate of bit shift and bit crowding in digital magnetic recording", IBM Technical Report, TR02.245, January 24, 1963, pp. 1-17.

[2] Kosters, A. I., and Speliotis, D. E., "Predicting magnetic recording performance by using single pulse superposition", Intemag 1971, IEEE Transactions on Magnetics, September, 1971, pp. 544.

[3] Macintosh, N. D., "A superposition-based analysis of pulse slimming techniques for digital recording", Video and Data Recording Conference, Southampton University, July 1979, pp. 1-3.

[4] Gulak, P. G., "Decision feedback equalization for digital communication over the saturated magnetic recording channel", M. Sc. thesis 1980, University of Manitoba.

[5] Knoll, A. L., "Spectrum Analysis of Digital Magnetic Recording Waveforms", IEEE Transaction on Electronic Computers, vol., EC-16, December, 1967, pp. 732-743.

[6] Mallinson, J.C., and Miller, J. W., "Optimal codes for digital magnetic recording", The Radio and

Electronic Engineer, vol 47, No. 4, April 1977, pp. 172-176.

[7] Horiguchi, T., and Morita, K., "An optimization of modulation codes in digital recording", IEEE Transactions on Magnetics, vol. MAG-12, No. 6, November 1976, pp. 740-742.

[8] Chu, W. W., "Computer Simulation of Waveform Distortions in Digital Magnetic Recordings", IEEE Transactions on Electronic Computers, vol. EC-15, No. 3, June, 1966, pp. 328-336.

[9] Jacoby, G. V., "Signal Equalization in Digital Magnetic Recording", IEEE transactions on Magnetics, vol. MAG-1, No. 3, September, 1968, pp. 302-305.

[10] Schneider, R. C., "An improved pulse-slimming method for magnetic recording", IEEE Transactions on Magnetics, vol. MAG-11, No. 5, September, 1975, pp. 1240-1243.

[11] Viterbi, A. J., "Error Bounds for Convolutional codes and an Asymptotically optimum Decoding Algorithm", IEEE Transactions on Information Theory, vol. IT-13, pp. 260-269.

[12] Forney, G. D., "Maximum-likelihood sequence estimation of digital sequences in the presence of intersymbol interference", IEEE Trans. on Information Theory, vol. IT-18, May 1972, pp. 368-378.

[13] Kobayasi, H., "Application of Probabilistic Decoding to Digital Magnetic Recording Systems", IBM Journal of Research and Development, 1970, pp 64-74.

[14] Viterbi, A. J., and Omura, J. K., "Principles of Digital Communication and Coding", McGraw-Hill Book Company, 1979, pp. 272-286.

[15] Duncan, F. G., "Microprocessor Programming and Software Development", Prentice-Hall International Inc., 1979.

[16] Papoulis, A., "Probability, Random Variables, and Stochastic Processes", McGraw-Hill Book Company, 1965, pp 264.

APPENDIX 1

SIMULATION PROGRAM IN PASCAL LANGUAGE

```
//PASCAL JOB ',,,,R=1024,T=460,L=20,C=0,I=70',PUI
// EXEC PASCCG
//PASC.SYSIN DD *
PROGRAM PROG(INPUT,OUTPUT,NOISE,SIGMA,FLREAL,FLINT,FLBIN,FLTERN);

CONST PI=3.14159265;
      MAXNOFSTATES=255; (* 2 TO POWER OF 8 *)
      MAXINT=2147483647; (* 2 TO POWER OF 31 *)

TYPE BINARY=0..1;
     TERNARY=-1..1;
     BRANGE=1..4;
     NRANGE=1..10;
     KRANGE=0..6;
     PATHRANGE=0..30;
     STORAGERANGE=0..40;
     POSITIVEINTEGER=0..MAXINT;
     STATERANGE=0..MAXNOFSTATES;
     UVECTOR=ARRAY[BRANGE,STORAGERANGE] OF BINARY;
     VVECTOR=ARRAY[NRANGE] OF BINARY;
     MEMORYRANGE=0..6;

VAR B,N,K,L,MEMORY,RULE,LAST : POSITIVEINTEGER;
    EFFECTIVENUMBER,NUMBEROFTIMES,LENGTH,ERROR,TAPEMEMORY: INTEGER;
    Q,NUMBEROFSTATES,LASTSTATE,INDEX4,INDEX1,INDEX2,INDEX3:INTEGER;
    DATA:BINARY;
    GO_ON:BOOLEAN;
    PE,SK,NK,SEED,SIGMA2:REAL;
    STATE1,STATE2 : ARRAY [STATERANGE] OF REAL;
    PATH1,PATH2 : ARRAY [STATERANGE,BRANGE,PATHRANGE] OF BINARY;
    H : ARRAY [-9..9] OF REAL;
    D : ARRAY [-100..100] OF BINARY;
    X : ARRAY [-100..100] OF TERNARY;
    YK : ARRAY [NRANGE] OF REAL;
    U : ARRAY [BRANGE,STORAGERANGE] OF BINARY;
    V : ARRAY [NRANGE] OF BINARY;
    G : ARRAY [NRANGE,BRANGE,KRANGE] OF BINARY;
    NOISE : FILE OF REAL;
    SIGMA : FILE OF REAL;
    FLREAL : FILE OF REAL;
    FLINT : FILE OF INTEGER;
    FLBIN : FILE OF BINARY;
    FLTERN : FILE OF TERNARY;

FUNCTION POWER(NUMBER,EXPONENT:POSITIVEINTEGER):INTEGER;
VAR I:INTEGER;
BEGIN
  IF EXPONENT=0 THEN POWER:=1
  ELSE POWER:=NUMBER*POWER(NUMBER,EXPONENT-1)
END;

FUNCTION ROUNDUP(NUMBER:REAL):INTEGER;
BEGIN
  IF NUMBER=1.0*TRUNC(NUMBER) THEN ROUNDUP:=TRUNC(NUMBER)
  ELSE ROUNDUP:=TRUNC(NUMBER)+1
END;

FUNCTION RANDOM(NUMBER:REAL):REAL;
VAR Y:INTEGER;
BEGIN
  Y:=ROUND(NUMBER/0.4656613E-09)*65539;
  IF Y<0 THEN Y:=Y+MAXINT+1;
  RANDOM:=Y*0.4656613E-09;
END;    THIS ROUTINE GIVES A RANDOM NUMBER 0<N<1 INDEPENDENT OF INPUT N
```

```
PROCEDURE GETDATA(VAR Y:REAL);
BEGIN
  Y:=RANDOM(Y);
  IF Y>=0.5 THEN DATA:=1 ELSE DATA:=0;
END;

PROCEDURE SETPARAMETERS;
VAR I1,I2,I3:INTEGER;
BEGIN
  SEED:=0.21735299441;
  B:=1;
  N:=2;
  L:=3;
  K:=3;
  Q:=POWER(2,B);
  TAPEMEMORY:=L+1;
  EFFECTIVENUMBER:=ROUNDUP((TAPEMEMORY-1)/N);
  LENGTH:=N*EFFECTIVENUMBER;
  MEMORY:=EFFECTIVENUMBER+K;
  H[0]:=1.209;
  H[1]:=0.734; H[2]:=0.316; H[3]:=0.057;
  H[-1]:=H[1]; H[-2]:=H[2]; H[-3]:=H[3];
  NUMBEROFTIMES:=1000000;
  FOR I1:=1 TO N DO
    FOR I2:=1 TO B DO
      FOR I3:=0 TO K-1 DO READ(INPUT,G[I1,I2,I3]);
  NUMBEROFSTATES:=POWER(Q,MEMORY-1);
  LASTSTATE:=NUMBEROFSTATES-1;
  RULE:=5*MEMORY;
  LAST:=RULE+EFFECTIVENUMBER
END;

PROCEDURE INITIALIZE;
VAR I1,I2,I3:INTEGER;
BEGIN
  FOR I1:=0 TO LASTSTATE DO
  BEGIN
    STATE1[I1]:=0.0;
    STATE2[I1]:=0.0;
    FOR I2:=1 TO B DO FOR I3:=1 TO RULE DO
    BEGIN
      PATH1[I1,I2,I3]:=0;
      PATH2[I1,I2,I3]:=0
    END
  END;
  ERROR:=0;
  FOR I1:=-LENGTH TO LENGTH DO D[I1]:=0;
  FOR I1:=-LENGTH TO LENGTH-1 DO X[I1]:=0;
  FOR I1:=1 TO N DO V[I1]:=0;
  FOR I1:=1 TO B DO FOR I2:=0 TO RULE+EFFECTIVENUMBER DO U[I1,I2]:=0
END;

PROCEDURE SHIFTU;
VAR I1,I2: INTEGER;
BEGIN
  FOR I1:=1 TO B DO FOR I2:=RULE+EFFECTIVENUMBER DOWNTO 1 DO
    U[I1,I2]:=U[I1,I2-1]
END;

PROCEDURE SHIFTD;
VAR I: INTEGER;
BEGIN
  FOR I:=LENGTH DOWNTO -(LENGTH-1) DO

    D[I]:=D[I-1]
END;
```

```
PROCEDURE SHIFTX;
VAR I: INTEGER;
BEGIN
  FOR I:=LENGTH-1 DOWNTO -(LENGTH-1) DO
    X[I]:=X[I-1]
END;

PROCEDURE SHIFTPATH2(NS:INTEGER);
VAR I1,I2: INTEGER;
BEGIN
  FOR I1:=RULE DOWNTO 1 DO FOR I2:=1 TO B DO
    PATH2[NS,I2,I1]:=PATH2[NS,I2,I1-1]
END;

PROCEDURE INPUTZERO;
VAR I: INTEGER;
BEGIN
  FOR I:=B DOWNTO 1 DO U[I,0]:=0
END;

PROCEDURE INPUT_U;
VAR I:INTEGER;
BEGIN
  FOR I:=B DOWNTO 1 DO
  BEGIN
    GETDATA(SEED);
    U[I,0]:=DATA
  END
END;

PROCEDURE QTOBINARY(N1:INTEGER;VAR N2:VVECTOR);
VAR I:INTEGER;
BEGIN
  FOR I:=1 TO B DO
  BEGIN
    N2 I :=N1 MOD 2;
    N1:=N1 DIV 2
  END
END;

PROCEDURE INPUTPATH2(NS,PN:INTEGER);
VAR I1: INTEGER;
    BN: VVECTOR;
BEGIN
  QTOBINARY(PN,BN);
  FOR I1:=1 TO B DO PATH2[NS,I1,0]:=BN[I1]
END;

PROCEDURE ENCODE(U1:UVECTOR; VAR V1:VVECTOR);
VAR I1,I2,I3:INTEGER;
BEGIN
  FOR I1:=1 TO N DO
  BEGIN
    V1[I1]:=0;
    FOR I2:=1 TO B DO
      FOR I3:=0 TO K-1 DO
        V1[I1]:=(V1[I1]+G[I1,I2,I3]*U1[I2,I3]) MOD 2
  END
END;

PROCEDURE GETMAXSTATE(VAR MAXSTATE:INTEGER);


VAR I:INTEGER;
BEGIN
  MAXSTATE:=0;
  FOR I:=1 TO LASTSTATE DO
    IF STATE1[MAXSTATE]<STATE1[I] THEN
      MAXSTATE:=I
END;
```

```
PROCEDURE UPDATE_ERROR;
VAR I,NUMBER:INTEGER;
    ADD1,ADD2,ADD3:INTEGER;
BEGIN
  GETMAXSTATE(NUMBER);
  FOR I:=1 TO B DO
    BEGIN
      ADD1:=U[I,LAST];
      ADD2:=PATH1[NUMBER,I,RULE];
      ADD3:=ADD1+ADD2;
      ADD3:=ADD3 MOD 2;
      ERROR:=ERROR+ADD3
    END
END;

PROCEDURE EXCHANGE;
BEGIN
  STATE1:=STATE2;
  PATH1:=PATH2
END;

PROCEDURE SAMPLEYK;
VAR I1,I2,I3 : INTEGER;
BEGIN
  FOR I1:=1 TO N DO
  BEGIN
    SHIFTD; SHIFTX;
    V[I1]:=0;
    FOR I2:=1 TO B DO FOR I3:=0 TO K-1 DO
      V[I1]:=(V[I1]+G[I1,I2,I3]*U[I2,I3]) MOD 2;
    D[-LENGTH]:=V[I1];
    X[-LENGTH]:=D[-LENGTH]-D[-(LENGTH-1)];
    SK:=0.0;
    FOR I2:=-(L-1) TO L-1 DO SK:=SK+H[I2]*X[I2];
    READ(NOISE,NK);
    YK[I1]:=SK+NK
  END
END;

FUNCTION BRANCHMETRIC(NUMBER:INTEGER):REAL;
VAR I1,I2,I3,I4:INTEGER;
    SUM,LAMDAK:REAL;
    UN : UVECTOR;
    TU : UVECTOR;
    VN : VVECTOR;
    DN : ARRAY [0..100] OF BINARY;
    XN : ARRAY [0..100] OF TERNARY;
BEGIN
  FOR I2:=0 TO MEMORY-1 DO
    FOR I1:=1 TO B DO
    BEGIN
      UN[I1,I2]:=NUMBER MOD 2;
      NUMBER:=NUMBER DIV 2
    END;
  I1:=LENGTH;
  FOR I2:=EFFECTIVENUMBER DOWNTO 1 DO

 BEGIN
   FOR I3:=0 TO K-1 DO FOR I4:=1 TO B DO
     TU[I4,I3]:=UN[I4,I3+I2];
   ENCODE(TU,VN);
   FOR I3:=1 TO N DO
   BEGIN
     DN[I1]:=VN[I3];
     I1:=I1-1
   END
 END;
```

```
   FOR I1:=1 TO LENGTH-1 DO XN[I1]:=DN[I1]-DN[I1+1];
   FOR I1:=0 TO K-1 DO FOR I2:=1 TO B DO
     TU[I2,I1]:=UN[I2,I1];
   ENCODE(TU,VN);
   I1:=0;
   LAMDAK:=0.0;
   REPEAT
     I1:=I1+1;
     DN[0]:=VN[I1];
     XN[0]:=DN[0]-DN[1];
     SUM:=0.0;
     FOR I2:=1 TO L-1 DO SUM:=SUM+XN[I2]*H[I2];
     LAMDAK:=LAMDAK+2*YK[I1]*XN[0]-XN[0]*XN[0]*H[0]-2*XN[0]*SUM;
     FOR I2:=LENGTH DOWNTO 1 DO DN[I2]:=DN[I2-1];
     FOR I2:=LENGTH-1 DOWNTO 1 DO XN[I2]:=XN[I2-1];
   UNTIL I1=N;
   BRANCHMETRIC:=LAMDAK
 END;

 PROCEDURE PRINT_V;
 VAR I1: INTEGER;
 BEGIN
   WRITELN('REGISTER OF OUTPUT VECTOR V:');
   I1:=N;
   REPEAT
     WRITE(D[-LENGTH+I1-1]:1);
     I1:=I1-1
   UNTIL I1=0;
   WRITELN
 END;

 PROCEDURE PRINT_U;
 VAR I1,I2: INTEGER;
 BEGIN
   WRITELN('REGISTER OF INPUT VECTOR U:');
   FOR I1:=1 TO B DO
   BEGIN
     FOR I2:=0 TO RULE+EFFECTIVENUMBER DO WRITE(U[I1,I2]:1);
     WRITELN
   END
 END;

 PROCEDURE DISPLAYPATH;
 VAR I1,I2,I3: INTEGER;
 BEGIN
   FOR I1:=0 TO LASTSTATE DO
   BEGIN
     WRITELN('PATH OF STATE: ',I1:3);
     FOR I2:=1 TO B DO
     BEGIN
       FOR I3:=0 TO RULE DO WRITE(PATH1[I1,I2,I3]:1);
       WRITELN
     END
   END;



   WRITELN
 END;

 PROCEDURE PRINTYK;
 VAR I:INTEGER;
 BEGIN
   WRITELN(N:2,' SAMPLES OF YK:');
   FOR I:=1 TO N DO WRITE(YK[I]:7:2,';  ');
   WRITELN
 END;
```

```
PROCEDURE PRINTSTATEMETRIC;
VAR I: INTEGER;
BEGIN
  WRITELN('STATEMETRIC FOR SAMPLES YK');
  FOR I:=0 TO LASTSTATE DO
    WRITELN(' STATE METRIC OF ',I:3,STATE1[I]:7:2);
  WRITELN
END;

PROCEDURE STARTTRELLIS;
VAR I1,I2,I3,I4,I5,I6:INTEGER;
BEGIN
  I1:=0;
  REPEAT
    I1:=I1+1;
    SHIFTU;
    INPUT_U;
    SAMPLEYK;
    FOR I2:=0 TO POWER(Q,I1)-1 DO
    BEGIN
      STATE2[I2]:=STATE1[I2 DIV Q]+ BRANCHMETRIC(I2);
      I5:=I2 DIV Q;
      PATH2[I2]:=PATH1[I5];
      SHIFTPATH2(I2);
      INPUTPATH2(I2,I2 MOD Q);
    END;
    EXCHANGE;
  UNTIL I1=MEMORY-1
END;

PROCEDURE ADDCOMPARESELECT;
VAR I1,I2,I3,I4,SN,SURVIVOR:INTEGER;
    MAX:REAL;
BEGIN
  FOR I1:=0 TO Q-1 DO
    FOR I2:=0 TO POWER(Q,MEMORY-2)-1 DO
    BEGIN
      SN:=I1+I2*Q;
      MAX:=STATE1[SN DIV Q]+BRANCHMETRIC(SN);
      SURVIVOR:=SN DIV Q;
      FOR I3:=1 TO Q-1 DO
        IF MAX<(STATE1[(SN+I3*POWER(Q,MEMORY-1))
        DIV Q]+BRANCHMETRIC(SN+I3*POWER(Q,MEMORY-1)))
        THEN BEGIN
          MAX:=STATE1[(SN+I3*POWER(Q,MEMORY-1))DIV
               Q]+BRANCHMETRIC(SN+I3*POWER(Q,MEMORY-1));
          SURVIVOR:=(SN+I3*POWER(Q,MEMORY-1))DIV Q
        END;
      STATE2[SN]:=MAX;
      PATH2[SN]:=PATH1[SURVIVOR];
      SHIFTPATH2(SN);
      INPUTPATH2(SN,SN MOD Q)
    END;


  EXCHANGE;
END;

PROCEDURE INITREAD;
BEGIN
  RESET(NOISE);
  RESET(SIGMA);
  RESET(FLREAL);
  RESET(FLTERN);
  RESET(FLBIN);
  RESET(FLINT);
END;
```

```
PROCEDURE INITWRITE;
BEGIN
  REWRITE(FLREAL);
  REWRITE(FLINT);
  REWRITE(FLBIN);
  REWRITE(FLTERN);
END;

PROCEDURE STACKUPINFO;
VAR I1,I2,I3 : INTEGER;
BEGIN
  FOR I1:=0 TO LASTSTATE DO
  BEGIN
    WRITE(FLREAL,STATE1[I1]); WRITE(FLREAL,STATE2[I1]);
    FOR I2:=1 TO B DO FOR I3:=1 TO RULE DO
    BEGIN
      WRITE(FLBIN,PATH1[I1,I2,I3]);
      WRITE(FLBIN,PATH2[I1,I2,I3]);
    END
  END;
  WRITE(FLINT,ERROR);
  WRITE(FLINT,INDEX1-1);
  FOR I1:=-LENGTH TO LENGTH DO WRITE(FLBIN,D[I1]);
  FOR I1:=-LENGTH TO LENGTH-1 DO WRITE(FLTERN,X[I1]);
  FOR I1:=1 TO N DO WRITE(FLBIN,V[I1]);
  FOR I1:=1 TO B DO FOR I2:=0 TO RULE+EFFECTIVENUMBER DO
     WRITE(FLBIN,U[I1,I2])
END;

PROCEDURE READSTACK;
VAR I1,I2,I3 : INTEGER;
BEGIN
  FOR I1:=0 TO LASTSTATE DO
  BEGIN
    READ(FLREAL,STATE1[I1]); READ(FLREAL,STATE2[I1]);
    FOR I2:=1 TO B DO FOR I3:=1 TO RULE DO
    BEGIN
      READ(FLBIN,PATH1[I1,I2,I3]);
      READ(FLBIN,PATH2[I1,I2,I3]);
    END
  END;
  READ(FLINT,ERROR);
  READ(FLINT,INDEX1);
  FOR I1:=-LENGTH TO LENGTH DO READ(FLBIN,D[I1]);
  FOR I1:=-LENGTH TO LENGTH-1 DO READ(FLTERN,X[I1]);
  FOR I1:=1 TO N DO READ(FLBIN,V[I1]);
  FOR I1:=1 TO B DO FOR I2:=0 TO RULE+EFFECTIVENUMBER DO
     READ(FLBIN,U[I1,I2])
END;
```

```
BEGIN

   SETPARAMETERS;
   WRITELN('TAP G: ');
   FOR INDEX1:=1 TO N DO
      FOR INDEX2:=1 TO B DO
      BEGIN
         FOR INDEX3:=0 TO K-1 DO WRITE('   G',INDEX1:2,INDEX2:2,INDEX3:2,
            ' = ',G[INDEX1,INDEX2,INDEX3]:2);
         WRITELN;
      END;
   INITIALIZE;  (INITREAD;
   FOR INDEX1:=1 TO 1 DO READ(SIGMA,SIGMA2);
   WRITELN('FOR SIGMA2 =  ',SIGMA2,'   SIGNAL TO NOISE IN DB',INDEX1:4);
   STARTTRELLIS;
   INDEX1:=MEMORY;
   GO_ON:=TRUE;
   WHILE (GO_ON) DO
   BEGIN
      SHIFTU;
      IF INDEX1<=NUMBEROFTIMES THEN INPUT_U ELSE INPUTZERO;
      SAMPLEYK;
      ADDCOMPARESELECT;
      UPDATE_ERROR;
      INDEX1:=INDEX1+1;
      IF ERROR=10 THEN GO_ON:=FALSE;
      IF EOF(NOISE) THEN
         BEGIN
            GO_ON:=FALSE;  (INITWRITE;
            STACKUPINFO;
         END
   END;
   PE:=ERROR/((INDEX1-1)*B);
   WRITELN('ERROR RATE = ',PE,'  TOTAL NUMBER OF ERROR IS',ERROR:4);
   WRITELN('TOTAL NUMBER OF BITS SENT =',(INDEX1-1)*B);
   WRITELN('*********** FINISH ************')
END.
```

APPENDIX 2

FORTRAN PROGRAM FOR DMRS SIMULATION

```
C ****************
C *** TRAN.FOR ***
C ****************

        INTEGER*2 SELECT,INOI(8),NP,PATH1(768),PATH2(768)
        REAL*4 AA,BB,RNOI,BM(512),SCALE
        INTEGER*2 U(3),V(3),X(17),IP
        INTEGER*2 MAXI,ERROR,COUNT1,COUNT2,CM
        INTEGER*2 I,O,C,J,J1(256),J2(256),K1(256),K2(256)
        INTEGER*2 A1,A2,B1,B2,Z(8,512),R(8),RV(8),ID
        REAL*4 Y(8),S,MAX,PE
        REAL*4 SM1(256),SM2(256),BMF(512)
        INTEGER*2 B,N,K,Q,L,NN,M,RULE,NS,W
        REAL*4 H(17)

        COMMON NS,I,L,S,H,X,V,U,Y,SM1,SM2,BMF,J1,J2,K1,K2,Z
        COMMON AA,BB,RNOI,BM,SELECT,INOI,NP,PATH1,PATH2,MAXI
        COMMON SCALE,J,B,N,K,Q

        SCALE=0.0001
1       SCALE=SCALE*2
        DO 1000 W=1,10
C
C       SETUP PARAMETERS FOR TRANSMIT ROUTINE
C
        PRINT*,'TRIAL NUMBER',W,'BEGIN',' WITH SCALE=',SCALE
        PRINT*
        B=1
        N=1
        K=2
        Q=2**B
        L=3
        NN=INT(1.0*(L-1)/N+.99999)
        M=NN+K
        RULE=5*M
        NS=Q**(M-1)
        NS2=NS/2
C
C       PRINT*,'B',B,'N',N,'K',K,'M',M,'Q',Q,'NS',NS
C
        PRINT*,
        COUNT1=1000
        COUNT2=1000
        CM=1000
        NP=2*NS
```

```
C        INITIALIZATION

         H(0)=0.001
         H(1)=0.002
         H(2)=0.002
         H(3)=0.006
         H(4)=0.015
         H(5)=0.047
         H(6)=0.179
         H(7)=0.607
         H(8)=1.0
         H(9)=H(7)
         H(10)=H(6)
         H(11)=H(5)
         H(12)=H(4)
         H(13)=H(3)
         H(14)=H(2)
         H(15)=H(1)
         H(16)=H(0)

C
C        THIS ROUTINE INITIALIZE PATHS CONTENT
C
         DO 4 I=1,768
         PATH1(I)=0
         PATH2(I)=0
4        CONTINUE
C
C        THIS ROUTINE INITIALIZES METRICS AND SETUP SELECTION INDEX
C
         DO 5,I=1,NS
         SM1(I)=0
         SM2(I)=0
         CALL IND1
         J1(I)=A1
         J2(I)=A2
         K1(I)=B1
         K2(I)=B2
         PRINT*,I,A1,A2
5        CONTINUE
         PRINT*
         PRINT*

C
C        BMF AND Z PROCESS FOR NRZI ENCODING
C

         DO 7 I=1,NP
         ID=I-1
         DO 61 J=1,L+1
         R(J)=MOD(ID,2)
         ID=ID/2
61       CONTINUE
C        PRINT*,(R(J),J=1,L+1)
         DO 62 J=1,L
         R(L+1-J)=MOD((R(L+1-J)+R(L-J+2)),2)
62       CONTINUE
```

```
C       PRINT*,(R(J),J=1,L+1)
        BMF(I)=H(8)*(R(1)-R(2))
        Z(1,I)=R(1)-R(2)
        DO 63 J=1,L-1
        BMF(I)=BMF(I)+2*H(J+8)*(R(J+1)-R(J+2))
63      CONTINUE
C       PRINT*,BMF(I)
        BMF(I)=Z(1,I)*BMF(I)
7       CONTINUE

C
C       BMF AND Z PROCESS FOR MFM ENCODING
C

C       DO 7 I=1,NP
C       ID=I-1
C       DO 61 J=1,M
C       R(J)=MOD(ID,2)
C       ID=ID/2
C61     CONTINUE
C       RV(2*(NN+1)+1)=R(M)
CC      PRINT*,(R(J),J=1,M)
C       DO 62 J=M-2,1,-1
C       RV(2*J)=RV(2*J+1)
C       IF (R(J).EQ.0.AND.R(J+1).EQ.0) RV(2*J)=MOD((1+RV(2*J+1)),2)
C       RV(2*J-1)=MOD((RV(2*J)+R(J)),2)
C62     CONTINUE
CC      PRINT*,(RV(J),J=1,2*(NN+1))
C       Z(2,I)=RV(1)-RV(2)
CC      PRINT*,Z(2,I)
C       BMF(I)=H(8)*Z(2,I)*Z(2,I)
C       DO 63 J=1,L-1
C       BMF(I)=BMF(I)+2*H(J+8)*(RV(J+1)-RV(J+2))*Z(2,I)
CC      PRINT*,(RV(J+1)-RV(J+2))
C63     CONTINUE
C       Z(1,I)=RV(2)-RV(3)
CC      PRINT*,Z(1,I)
C       BMF(I)=BMF(I)+H(8)*Z(1,I)*Z(1,I)
C       DO 64 J=1,L-1
C       BMF(I)=BMF(I)+2*H(J+8)*(RV(J+2)-RV(J+3))*Z(1,I)
CC      PRINT*,(RV(J+2)-RV(J+3))
C64     CONTINUE
C7      CONTINUE


C
C       BMF AND Z PROCESS FOR 1/2 CC WITH NRZI
C

C       DO 7 I=1,NP
C       ID=I-1
C       DO 61 J=1,M
C       R(J)=MOD(ID,2)
C       ID=ID/2
C61     CONTINUE
```

```
CC        PRINT*,(R(J),J=1,M)
C         RV(2*(M-3)+1)=R(M)
C         DO 62 J=M-3,1,-1
CC        PRINT*,'J',J,R(J),R(J+1),R(J+2),RV(2*J+1)
C         RV(2*J)=RV(2*J+1)+R(J)+R(J+1)+R(J+2)
CC        PRINT*,RV(2*J)
C         RV(2*J)=RV(2*J)-(RV(2*J)/2)*2
CC        PRINT*,RV(2*J),'MOD'
C         RV(2*J-1)=RV(2*J)+R(J)+R(J+2)
C         RV(2*J-1)=RV(2*J-1)-(RV(2*J-1)/2)*2
CC        PRINT*,RV(2*J-1)
C62       CONTINUE
CC        PRINT*,(RV(J),J=1,2*(M-3))
C         Z(2,I)=RV(1)-RV(2)
CC        PRINT*,Z(2,I)
C         BMF(I)=H(8)*Z(2,I)*Z(2,I)
C         DO 63 J=1,L-1
C         BMF(I)=BMF(I)+2*H(J+8)*(RV(J+1)-RV(J+2))*Z(2,I)
CC        PRINT*,RV(J+1)-RV(J+2)
C63       CONTINUE
C         Z(1,I)=RV(2)-RV(3)
CC        PRINT*,Z(1,I)
C         BMF(I)=BMF(I)+H(8)*Z(1,I)*Z(1,I)
C         DO 64 J=1,L-1
C         BMF(I)=BMF(I)+2*H(J+8)*(RV(J+2)-RV(J+3))*Z(1,I)
CC        PRINT*,RV(J+2)-RV(J+3)
C64       CONTINUE
C7        CONTINUE
C

C
C         BMF AND Z PROCESS FOR 1/2 CC WITH MFM
C

C         DO 7 I=1,NP
C         ID=I-1
C         DO 61 J=1,M
C         R(J)=ID-(ID/2)*2
C         ID=ID/2
C61       CONTINUE
CC        PRINT*,(R(J),J=1,M)
C         RV(2*(NN+1)+1)=R(M)
C         DO 62 J=NN+1,1,-1
C         RV(2*J)=R(J)+R(J+1)+R(J+2)
C         RV(2*J)=RV(2*J)-(RV(2*J)/2)*2
C         RV(2*J-1)=R(J)+R(J+2)
C         RV(2*J-1)=RV(2*J-1)-(RV(2*J-1)/2)*2
C62       CONTINUE
C         RV(4*(NN+1)+1)=R(M-1)
C         DO 63 J=2*(NN+1),1,-1
C         A1=RV(2*J+1)
C         IF (RV(J).EQ.0.AND.RV(J+1).EQ.0) A1=(A1+1)-((A1+1)/2)*2
C         RV(2*J)=A1
C         RV(2*J-1)=RV(2*J)+RV(J)
C         RV(2*J-1)=RV(2*J-1)-(RV(2*J-1)/2)*2
C63       CONTINUE
CC        PRINT*,(RV(J),J=1,4*(NN+1))
C         BMF(I)=0
```

```
C         DO 64 J=1,4
C         Z(5-J,I)=RV(J)-RV(J+1)
C         BMF(I)=BMF(I)+H(8)*Z(5-J,I)*Z(5-J,I)
C         DO 65 A1=1,L-1
C         BMF(I)=BMF(I)+2*H(A1+8)*(RV(A1+J)-RV(A1+J+1))*Z(5-J,I)
C65       CONTINUE
C64       CONTINUE
C7        CONTINUE


C
C         TRANSMITTING ROUTINE
C
          DO 50 C=1,COUNT1
          CALL TRAN2
50        CONTINUE
C
C         THIS ROUTINE BEGINS COUNTING ERRORS
C
          ERROR=0
          DO 90 C=1,CM
          DO 93 O=1,COUNT2
          CALL TRAN2
C
C         SYSTEM EVALUATION FOR EVERY DECODED INPUT(S)
C

          MAX=SM2(1)
          MAXI=1
          DO 80 I=2,NS
          IF(MAX.GT.SM2(I))GOTO 80
          MAX=SM2(I)
          MAXI=I
80        CONTINUE
          I=MAXI
          CALL PRT
          CALL COMP
          IF(ERROR.GT.10)GOTO 99
93        CONTINUE
90        CONTINUE
          O=O-1
          C=C-1
99        C=C-1
          PE=1.0*ERROR/(C*1000+O)
          PRINT *,'PROBABILITY OF ERROR',PE,'ERROR',ERROR
          PRINT *,'COUNTS',C*1000+O
          I=MAXI
          CALL PRT
          STOP
          END
```

```
      SUBROUTINE TRAN2
      INTEGER*2 SELECT,INOI(8),NP,PATH1(768),PATH2(768)
      REAL*4  AA,BB,RNOI,BM(512),SCALE,Y(8)
      INTEGER*2 NS,I,L,V(3),U(3),X(17),Z(8,512)
      REAL*4 S,SM1(256),SM2(256),BMF(512),H(17)
      INTEGER*2 J1(256),J2(256),K1(256),K2(256),J,K,B,Q,N

      COMMON NS,I,L,S,H,X,V,U,Y,SM1,SM2,BMF,J1,J2,K1,K2,Z
      COMMON AA,BB,RNOI,BM,SELECT,INOI,NP,PATH1,PATH2,MAXI
      COMMON SCALE,J,B,N,K,Q

      CALL INPUT
C     PRINT*,'U',U(1)

      CALL CODE1
C     CALL CODE2
C     CALL CODE3
C     CALL CODE4
C     PRINT*,'V',V(1)

C
C     THIS ROUTINE CALCULATE THE SAMPLE FROM ENCODED SEQUENCE
C

      DO 11 J=1,N
      S=0
      DO 10 I=1,15
      CALL GETX
C     PRINT*,J,X(I)
      IF(X(I)) 20,10,30
20    S=S-H(I)
      GOTO 10
30    S=S+H(I)
10    CONTINUE
      RNOI=SCALE*(INOI(J)-2048)/2048
      Y(J)=S+RNOI
C     PRINT*,'J',J,'Y',Y(J)
11    CONTINUE

C
C     THIS ROUTINE CALCULATE BRANCH METRIC FOR ALL PATH
C     BASED ON THE SAMPLED YK
C

      DO 44 I=1,NP
      S=0.0
      DO 43 J=1,N
      S=S+Y(J)*Z(J,I)
43    CONTINUE
      BM(I)=2*S-BMF(I)
44    CONTINUE
```

```
C
C       ADD COMPARE AND SELECT THE PATH WITH MAX ACCUMULATED METRICS
C

        DO 70 I=1,NS
        AA=SM1(J1(I))+BM(K1(I))
        BB=SM1(J2(I))+BM(K2(I))
        IF(AA.GT.BB)GOTO 60
        SM2(I)=BB
        SELECT=J2(I)
        MAXI=K2(I)
        GO TO 72
60      SM2(I)=AA
        SELECT=J1(I)
        MAXI=K1(I)
72      CALL DECODE
70      CONTINUE
74      CALL EXCHG
        RETURN
        END
```

APPENDIX 3

SUBROUTINES IN PDP 11/40 ASSEMBLY LANGUAGE

```
        .TITLE  TRANIO.MAC

DD      = 31026                         ;$DATA OFFSET
DU      = 31252                         ;.$$$$. OFFSET

THRESH  = 2048.                         ;INPUT THRESHOLD VALUE
AD      = 170402                        ;A/D INPUT BUFFER
DA      = 170420                        ;D/A OUTPUT BUFFER
ADSTAT  = 170400                        ;A/D STATUS REGISTER
CON     = 177566                        ;CONSOLE OUTPUT BUFFER

U       = 000166+DU
V       = 000160+DU
J       = 046302+DU
N       = 046306+DU
I       = 000002+DU
A1      = 000114+DD
A2      = 000116+DD
B1      = 000120+DD
B2      = 000122+DD
NS2     = 000146+DD
INOI    = 040252+DU
SELECT  = 040250+DU
PATH1   = 040274+DU
PATH2   = 043274+DU
SM1     = 000234+DU
SM2     = 002234+DU
NS      = 000000+DU
ERROR   = 000100+DD
MAXI    = 046274+DU
X       = 000116+DU

INPUT:: MOV     #U,R2           ;FETCH INPUT BUFFER POINTER
        CLR     @#ADSTAT        ;INITIALIZE STATUS REGISTER
        MOV     #20,@#ADSTAT    ;SET A/D STATUS TO SAMPLE CHAN 1
        MOV     #4095.,@#DA
8$:     BIT     #200,@#ADSTAT   ;TEST A/D STATUS
        BEQ     8$              ;WAIT FOR CONVERSION COMPLETE
        MOV     @#AD,R1         ;FETCH SAMPLE FROM A/D
        CLR     @#DA
        CMP     #THRESH,R1      ;COMPARE AGAINST THRESHOLD
        BPL     9$              ;IF LOWER, SAMPLE IS LOW
        SEC                     ;SAMPLE IS HIGH, SET INPUT BIT
        BR      7$
9$:     CLC                     ;SAMPLE IS LOW, CLEAR INPUT BIT
7$:     ROL     (R2)+           ;SHIFT THE INPUT BUFFER
        ROL     (R2)+
        ROL     (R2)+
        MOV     @#N,R0          ;BEGIN TO TAKE NOISE SAMPLE
        MOV     #INOI,R1
        CLR     @#ADSTAT        ;INIT FOR FIRST SAMPLE
        INC     @#ADSTAT        ;START CONVERSION
```

```
        BR      5$
1$:     CLR     @#DA              ;SET CH X OUTPUT LOW
        MOV     #420,@#ADSTAT     ;SET A/D TO SAMPLE CH1
        MOV     #4095.,@#DA
5$:     BIT     #200,@#ADSTAT     ;TEST A/D STATUS FOR CONVERSION COPMPLETE
        BEQ     5$                ;WAIT FOR CONVERSION COMPLETE
        MOV     @#AD,(R1)+        ;FETCH A/D SAMPLE
        SOB     R0,1$
        CLR     @#DA
        RTS     PC                ;RETURN TO CALLING ROUTINE
        .BLKW   128.              ;BUFFER TO ISOLATE LOCAL LABEL ADDRESSING

CODE1:: MOV     @#U,R0
        MOV     @#V,R1
        XOR     R0,R1
        ROR     R1
        MOV     #V,R0
        ROL     (R0)+
        ROL     (R0)+
        ROL     (R0)+
        RTS     PC
        .BLKW   128.

CODE2:: MOV     #V,R1
        BIT     #1,@#U
        BNE     2$
        BIT     #2,@#U
        BNE     2$
        BIT     #1,@#V
        BNE     3$
        SEC
        ROL     (R1)+
        ROL     (R1)+
        ROL     (R1)+
        MOV     #V,R1
        BR      4$
3$:     CLC
        ROL     (R1)+
        ROL     (R1)+
        ROL     (R1)+
        MOV     #V,R1
        BR      1$
2$:     BIT     #1,@#V
        BNE     5$
        CLC
        ROL     (R1)+
        ROL     (R1)+
        ROL     (R1)+
        MOV     #V,R1
        BIT     #1,@#U
        BNE     4$
        BR      1$
5$:     SEC
        ROL     (R1)+
        ROL     (R1)+
        ROL     (R1)+
```

```
        MOV     #V,R1
        BIT     #1,@#U
        BNE     1$
4$:     SEC
        ROL     (R1)+
        ROL     (R1)+
        ROL     (R1)+
        BR      7$
1$:     CLC
        ROL     (R1)+
        ROL     (R1)+
        ROL     (R1)+
7$:     RTS     PC
        .BLKW   128.

CODE3:: CLR     R3                      ;R3 STORES ENCODED V
        MOV     @#U,R0                  ;R0 CONTAINS INPUT DATA
        MOV     R0,R1                   ;DUPLICATE R0 TO R1
        ROR     R1                      ;OLD DATA AT LSB OF R1
        MOV     R1,R2                   ;DUPLICATE R1 TO R2
        ROR     R2                      ;SECOND OLD DATA AT LSB OF R2
        XOR     R2,R1
        XOR     R0,R1                   ;DO THE XOR TO GET ENCODE DATA
        ROR     R1
        ROL     R3                      ;SAVE ENCODED V1
        XOR     R0,R2
        BIT     #1,R2
        BEQ     1$
        SEC
        BR      3$
1$:     CLC
3$:     ROL     R3                      ;SAVE ENCODED V2
        MOV     #V,R0                   ;R0 POINTS TO THE ENCODED V
        MOV     @#V,R1                  ;R1 CONTAINS DUPLICATE OF V
        MOV     R3,R4                   ;R4 CONTAINS DUPLICATE OF V1 AND V2
        ROR     R4
        XOR     R1,R4
        ROR     R4                      ;SAVE ENCODED NRZI DATA 1
        ROL     (R0)+
        ROL     (R0)+
        ROL     (R0)+
        MOV     #V,R0                   ;RESET R0 TO POINT TO V
        MOV     @#V,R1                  ;R1 CONTAINS DUPLICATE OF V
        XOR     R1,R3
        ROR     R3                      ;SAVE ENCODED NRZI DATA 2
        ROL     (R0)+
        ROL     (R0)+
        ROL     (R0)+
        RTS     PC
        .BLKW   128.

CODE4:: MOV     #1,R3
        MOV     @#U,R5

        MOV     @#U,R1
        ROR     R1                      ;BIT 2 OF U AT LSB
```

```
        MOV     R1,R2
        ROR     R2
        ROR     R2              ;BIT 4 OF U AT LSB
        XOR     R1,R2
        ROR     R2
        ROL     R0              ;R0 <---- OLD CODE

        MOV     @#U,R1
        ROR     R1
        MOV     R1,R2
        ROR     R2
        XOR     R5,R1
        XOR     R2,R1
        ROR     R1
        ROL     R0              ;R0 <== FIRST CODE

        MOV     @#U,R1
        ROR     R1
        ROR     R1
        XOR     R5,R1
        ROR     R1
        ROL     R0              ;R0 <== SECOND CODE

        MOV     @#V,R1
        BIT     #6,R0
        BNE     1$
        XOR     R3,R1
1$:     ROR     R1
        MOV     #V,R1
        ROL     (R1)+           ;FIRST ENCODED V
        ROL     (R1)+
        ROL     (R1)+

        MOV     @#V,R1
        MOV     R0,R2
        ROR     R2              ;BIT 2 OF R0 AT LSB
        XOR     R1,R2
        ROR     R2
        MOV     #V,R1
        ROL     (R1)+           ;SECOND ENCODED V
        ROL     (R1)+
        ROL     (R1)+

        MOV     @#V,R1
        BIT     #3,R0
        BNE     2$
        XOR     R3,R1
2$:     ROR     R1
        MOV     #V,R1
        ROL     (R1)+           ;THIRD ENCODED V
        ROL     (R1)+
        ROL     (R1)+

        MOV     @#V,R4
        XOR     R4,R0
        ROR     R0
```

```
        MOV     #V,R1
        ROL     (R1)+
        ROL     (R1)+
        ROL     (R1)+               ;FOURTH ENCODED V

        RTS     PC
        .BLKW   128.


IND1::  MOV     @#I,R0
        DEC     R0
        CLR     R1                  ;R1 COUNTS NUMBER OF STATES
        CLR     R2                  ;R2 COUNTS A1 OR A2
        CLR     R3                  ;R3 USED AS REGISTER/ACC
14$:    BIT     #1,R0               ;TEST FOR INPUT 1 OR 0
        BEQ     2$
1$:     SEC                         ;INPUT 1/ODD ENTRY
        BR      3$
2$:     CLC                         ;INPUT 0/EVEN ENTRY
3$:     ROL     R3
        BIT     @#NS,R3             ;TEST HIGH ORDER BIT OF STATE
        BEQ     4$                  ;IF 0 THEN NO CHANGE OF NEXT BIT
        BIT     @#NS2,R3            ;TEST  NEXT HIGH ORDER BIT
        BEQ     5$                  ;GO CHANGE IT TO 1
        BIC     @#NS2,R3            ;IT IS 1 THEN CHANGE IT TO 0
        BR      6$
5$:     BIS     @#NS2,R3            ;IT IS 0 THEN CHANGE IT TO 1
6$:     BIC     @#NS,R3             ;CLEAR HIGH ORDER BIT/PROCESS END
4$:     CMP     R0,R3               ;COMPARE PROCESSED STATE WITH I-1
        BEQ     7$
12$:    INC     R1                  ;NEXT STATE TO SEARCH
        MOV     R1,R3
        BR      14$                 ;GO BACK
7$:     MOV     R1,R4               ;SAVE THE RETURN STATE
        INC     R1
        INC     R2
        BIT     #2,R2               ;CHECK A1 DONE
        BNE     8$                  ;GO FOR A2
        MOV     R1,@#A1             ;GO FOR A1
        BR      9$
8$:     MOV     R1,@#A2
9$:     DEC     R1                  ;NOW THE BRANCH B1 AND B2
        ASL     R1                  ;MULTIPLY BY 2
        BIT     #1,R0               ;TEST INPUT 1 OR 0
        BEQ     10$
        INC     R1
10$:    INC     R1
        BIT     #2,R2               ;TEST FOR B1 OR B2
        BNE     11$                 ;GO TO B2
        MOV     R1,@#B1             ;IT IS B1
        MOV     R4,R1               ;RECOVER RETURN STATE
        BR      12$                 ;GO BACK AND SEARCH NEXT
11$:    MOV     R1,@#B2             ;IT IS B2
        RTS     PC
        .BLKW   128.
```

```
IND2::  MOV     @#I,R0
        DEC     R0              ;R0 CONTAINS STATE BE MATCHED
        CLR     R1              ;R1 COUNTS THE STATE TO MATCH
        CLR     R2              ;R2 COUNTS A1 OR A2
        CLR     R3              ;R3 AND R5 USED AS REGISTERS
14$:    BIT     #1,R0           ;CHECK INPUT 1 OR 0
        BEQ     2$
1$:     SEC
        BR      3$
2$:     CLC
3$:     ROL     R3              ;INPUT SHIFTS INTO COUNTING STATE
        MOV     R3,R5           ;SAVE IN REGISTER 5
        ASL     R5              ;CHECK THE BIT BEHIND NS2
        BIT     @#NS2,R5        ;IF THIS BIT IS 0 THEN
        BNE     4$
        BIT     @#NS,R5         ;INVERT NS2 POSITION
        BEQ     16$
        BIC     @#NS,R5         ;IF IT IS 1 SET IT TO 0
        BR      17$
16$:    BIS     @#NS,R5         ;IF IT IS 0 SET IT TO 1
17$:    XOR     R3,R5           ;DO THE EXCLUSIVE OR FUNCTION OF
        BIT     @#NS,R5         ;THE BIT AT NS AND NS*2
        BNE     5$
7$:     BIC     @#NS2,R3        ;RESULT 0 IS STORED IN BIT NS2
        BR      6$
5$:     BIS     @#NS2,R3        ;RESULT 1 IS STORED IN BIT NS2
6$:     BIC     @#NS,R3         ;CLEAR MOST SIG BIT OF OLD STATE
        BR      8$              ;GO TO COMPARISON ROUTINE
4$:     BIT     @#NS,R3         ;OTHERWISE DEPENDS ON BIT AT NS
        BNE     7$
        BR      5$
8$:     CMP     R0,R3
        BEQ     9$
15$:    INC     R1
        MOV     R1,R3
        BR      14$             ;NOT MATCH GO BACK
9$:     MOV     R1,R4           ;SAVE THE RETURN STATE
        INC     R1
        INC     R2
        BIT     #2,R2           ;CHECK A1 DONE
        BNE     10$             ;GO FOR A2
        MOV     R1,@#A1         ;GO FOR A1
        BR      11$
10$:    MOV     R1,@#A2
11$:    DEC     R1              ;NOW THE BRANCH B1 AND B2
        ASL     R1              ;MULTIPLY BT 2
        BIT     #1,R0           ;TEST INPUT 1 OR 0
        BEQ     12$
        INC     R1
12$:    INC     R1
        BIT     #2,R2           ;TEST FOR B1 OR B2
        BNE     13$             ;FOR TO B2
        MOV     R1,@#B1         ;IT IS B1
        MOV     R4,R1           ;RECOVER RETURN STATE
```

```
        BR      15$                 ;GO BACK AND SEARCH NEXT
13$:    MOV     R1,@#B2             ;IT IS B2
        RTS     PC
        .BLKW   128.

IND3::  MOV     @#I,R0              ;R0 CONTAINS STATES TO BE MATCHED
        DEC     R0
        CLR     R1
        CLR     R2
        CLR     R3
14$:    BIT     #1,R0
        BEQ     2$
1$:     SEC
        BR      3$
2$:     CLC
3$:     ROL     R3
        MOV     R3,R4
        ROL     R4
        ROL     R4
        XOR     R3,R4
        BIT     @#NS,R4
        BEQ     4$
        BIS     @#NS2,R3
        BR      5$
4$:     BIC     @#NS2,R3
5$:     BIC     @#NS,R3
        CMP     R0,R3
        BEQ     7$
12$:    INC     R1
        MOV     R1,R3
        BR      14$
7$:     MOV     R1,R4
        INC     R1
        INC     R2
        BIT     #2,R2
        BNE     8$
        MOV     R1,@#A1
        BR      9$
8$:     MOV     R1,@#A2
9$:     DEC     R1
        ASL     R1
        BIT     #1,R0
        BEQ     10$
        INC     R1
10$:    INC     R1
        BIT     #2,R2
        BNE     11$
        MOV     R1,@#B1
        MOV     R4,R1
        BR      12$
11$:    MOV     R1,@#B2
        RTS     PC
        .BLKW   128.

IND4::  CLR     R0                  ;R0 AS COUNTER
        CLR     R3                  ;R3 AS MODIFYING STATES
```

```
31$:    BIT     #1,@#I
        BEQ     1$
        CLC
        BR      2$
1$:     SEC
2$:     ROL     R3              ;INPUTING TO OLD STATE

        MOV     R3,R4
        ROL     R4
        MOV     R4,R5
        ROL     R5
        XOR     R5,R4           ;XOR 1,2 BITS
        ROL     R5
        XOR     R5,R4           ;XOR 1,2,3 BITS
        BIT     @#NS2,R4
        BEQ     11$
        SEC
        BR      12$
11$:    CLC
12$:    ROL     R1              ;SAVE CC CODE1

        MOV     R3,R4
        ROL     R4
        MOV     R4,R5
        ROL     R5
        ROL     R5
        XOR     R5,R4           ;XOR BIT 1,3
        BIT     @#NS2,R4
        BEQ     13$
        SEC
        BR      14$
13$:    CLC
14$:    ROL     R1              ;SAVE CC CODE2

        BIT     @#NS,R3         ;TEST OLD CC CODE EQUALS 0
        BNE     15$
        BIT     #2,R1           ;IS THE INPUT ALSO 0
        BNE     15$
        BIT     @#NS2,R3        ;IF IT IS THEN MFM INPUT IS
        BNE     16$             ;OLD MFM INVERTED
        SEC
        BR      17$
16$:    CLC
        BR      17$

15$:    BIT     @#NS2,R3        ;IF NOT BOTH 0 THEN MFM INPUT
        BNE     4$              ;IS OLD MFM CODE
        CLC
        BR      17$
4$:     SEC

17$:    ROL     R2              ;SAVE MFM CODE1

        MOV     R1,R4           ;XOR INPUT CC CODE1 WITH MFM CODE
        ROR     R4
```

```
        XOR     R2,R4
        ROR     R4
        ROL     R2              ;SAVE MFM CODE2

        BIT     #2,R1           ;SEE IF NEXT CC CODE WITH OLD ONE 0
        BNE     18$
        BIT     #1,R1
        BNE     18$
        BIT     #1,R2           ;IF IT IS THEN INVERTED OLD MFM
        BNE     19$
        SEC
        BR      20$
19$:    CLC
        BR      20$

18$:    BIT     #1,R2           ;IF NOT THEN NEW MFM EQUALS OLD
        BNE     5$
        CLC
        BR      20$
5$:     SEC

20$:    ROL     R2              ;SAVE MFM CODE3

        MOV     R1,R4
        XOR     R2,R4
        ROR     R4
        ROL     R2              ;SAVE MFM CODE4

        MOV     @#NS2,R4
        ASR     R4

        BIT     #1,R1
        BEQ     21$
        BIS     @#NS2,R3
        BR      22$
21$:    BIC     @#NS2,R3
22$:    BIT     #1,R2
        BEQ     23$
        BIS     R4,R3
        BR      24$
23$:    BIC     R4,R3
24$:    BIC     @#NS,R3
        INC     R3              ;NEW STATE GENERATED

        CMP     @#I,R3
        BEQ     25$
30$:    INC     R0
        INC     R0
        MOV     R0,R3
        ASR     R3
        BR      31$

25$:    MOV     R0,R4
        ASR     R4
        BCS     26$
        INC     R4
```

```
        MOV     R4,@#A1
        BR      27$
26$:    INC     R4
        MOV     R4,@#A2
27$:    DEC     R4
        ASL     R4
        BIT     #1,@#I
        BNE     28$
        INC     R4
28$:    INC     R4
        BIT     #1,R0
        BNE     29$
        MOV     R4,@#B1
        INC     R0
        BR      30$
29$:    MOV     R4,@#B2
        RTS     PC
        .BLKW   128.

DECODE::MOV     @#I,R0          ;FETCH INDEX I
        DEC     R0              ;ADJUST FOR 0 OFFSET
        MOV     R0,R1           ;R1 <-- I
        ASH     #1,R1           ;R1 <-- I*2
        ADD     R0,R1           ;R1 <-- I*2+I=I*3
        ASH     #1,R1           ;R1 <-- OFFSET*2 TO GIVE BYTE OFFSET

        MOV     @#SELECT,R0     ;SAME AS FOR I USING SELECT INDEX
        DEC     R0
        MOV     R0,R2
        ASH     #1,R2
        ADD     R0,R2
        ASH     #1,R2           ;R2 <-- BYTE OFFSET OF SELECT

        ADD     #PATH2,R1       ;R1 <-- PATH2+OFFSET(I)
        MOV     R1,R4           ;SAVE PATH2(I) IN R4
        ADD     #PATH1,R2       ;R2 <-- PATH1+OFFSET(SELECT)

        MOV     (R2)+,(R1)+     ;PATH2(I) <-- PATH1(SELECT)
        MOV     (R2)+,(R1)+
        MOV     (R2)+,(R1)+

        MOV     @#MAXI,R0       ;FETCH MAXI
        DEC     R0              ;ADJUST FOR 0 OFFSET
        ROR     R0              ;SHIFT LOW ORDER BIT INTO CARRY

        ROL     (R4)+           ;SHIFT CARRY INTO PATH2(I)
        ROL     (R4)+
        ROL     (R4)+

        RTS     PC              ;RETURN TO CALLING ROUTINE
        .BLKW   128.

EXCHG:: MOV     #PATH1,R1       ;FETCH POINTER TO PATH1
        MOV     #PATH2,R2       ;FETCH POINTER TO PATH2
        MOV     @#NS,R0         ;SET NUMBER OF STATES COUNTER
```

```
        MOV     R0,R3           ;SAVE NS FOR LATER USE
2$:     MOV     (R2)+,(R1)+     ;PATH1 <-- PATH2
        MOV     (R2)+,(R1)+
        MOV     (R2)+,(R1)+
        SOB     R0,2$           ;REPEAT UNTIL ALL PATH1 <- PATH2

        MOV     #SM1,R1         ;FETCH POINTER TO SM1
        MOV     #SM2,R2         ;FETCH POINTER TO SM2
3$:     MOV     (R2)+,(R1)+     ;SM1 <-- SM2
        MOV     (R2)+,(R1)+     ;(TRANFER 2 WORDS FOR EACH REAL*4)
        SOB     R3,3$           ;REPEAT UNTIL ALL SM1 <-- SM2
        RTS     PC
        .BLKW   128.

GETX::  MOV     @#I,R0          ;R0 HOLDS ADDRESS OF X(I)
        DEC     R0
        ASH     #1,R0
        ADD     #X,R0
        MOV     @#N,R1          ;R1 COUNTS THE SHIFT FOR J
        SUB     @#J,R1          ;TO READ NEW DATA ENCODED
        BEQ     2$              ;IT HOLDS N-J NORMALLY IF 0 THEN
        MOV     #V,R5           ;OTHERWISE DO SHIFTING
        MOV     (R5)+,R2        ;FIRST GET ENCODED DATA
        MOV     (R5)+,R3
        MOV     (R5)+,R4
1$:     ROR     R4              ;DO THE CYCLIC SHIFT
        ROR     R3
        ROR     R2
        SOB     R1,1$
        BR      3$
2$:     MOV     @#V,R2          ;IF N=1 THEN R2 HOLDS ENCODED DATA
3$:     CLR     R3              ;R3 HOLDS THE DATA X TO BE CALCULATED
        MOV     @#I,R5          ;R5 COUNTS THE I SHIFTS TO GET X(I)
        INC     R5
        NEG     R5
        ASH     R5,R2
        BIT     #1,R2
        BEQ     4$
        BCS     5$
        DEC     R3
        BR      5$
4$:     BCC     5$
        INC     R3
5$:     MOV     R3,(R0)
        RTS     PC
        .BLKW   128.

COMP::  MOV     @#MAXI,R0
        DEC     R0
        MOV     R0,R1
        ASH     #1,R1
        ADD     R0,R1
        ASH     #1,R1

        ADD     #PATH1,R1
        ADD     #4,R1           ;GET LAST REGISTER
```

```
        MOV     #U,R2
        ADD     #4,R2           ;GET LAST REGISTER

        MOV     @#ERROR,R3

        BIT     #100000,@R2
        BPL     1$
        BIT     #200,@R1
        BEQ     2$
        BR      3$
1$:     BIT     #200,@R1
        BEQ     3$
2$:     INC     R3
3$:     MOV     R3,@#ERROR

        RTS     PC
        .BLKW   128.

PRT::   MOV     #U,R0
        JSR     PC,OUTPUT
        MOV     @#I,R0
        DEC     R0
        MOV     R0,R1
        ASH     #1,R0
        ADD     R1,R0
        ASH     #1,R0
        ADD     #PATH2,R0
        JSR     PC,OUTPUT
        JSR     PC,RETURN
        RTS     PC

OUTPUT: MOV     #48.,R4
        MOV     4(R0),R3
        MOV     2(R0),R2
        MOV     (R0),R1
1$:     ROR     R3
        ROR     R2
        ROR     R1
        BCC     2$
        JSR     PC,WAIT
        MOV     #61,@#CON
        BR      3$
2$:     JSR     PC,WAIT
        MOV     #60,@#CON
3$:     SOB     R4,1$
        JSR     PC,RETURN
        RTS     PC

RETURN: JSR     PC,WAIT
        MOV     #15,@#CON
        JSR     PC,WAIT
        MOV     #12,@#CON
        JSR     PC,WAIT
        RTS     PC

WAIT:   BIT     #200,@#177564
        BEQ     WAIT
        RTS     PC
```

APPENDIX 4

STORAGE MAP/ALLOCATION FOR FORTRAN SIMULATION PROGRAM

FORTRAN IV   Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000222 (73. words)

Name	Type	Offset	Name	Type	Offset	Name	Type	Offset
A1	I*2	000126	A2	I*2	000130	B1	I*2	000132
B2	I*2	000134	C	I*2	000124	CM	I*2	000120
COUNT1	I*2	000114	COUNT2	I*2	000116	ERROR	I*2	000112
ID	I*2	000136	IP	I*2	000110	M	I*2	000152
MAX	R*4	000140	NN	I*2	000150	NS2	I*2	000160
O	I*2	000122	PE	R*4	000144	RULE	I*2	000154
W	I*2	000156						

COMMON Block /   /, Size = 046314 (9830. words)

Name	Type	Offset	Name	Type	Offset	Name	Type	Offset
NS	I*2	000000	I	I*2	000002	L	I*2	000004
S	R*4	000006	H	R*4	000012	X	I*2	000116
V	I*2	000160	U	I*2	000166	Y	R*4	000174
SM1	R*4	000234	SM2	R*4	002234	BMF	R*4	004234
J1	I*2	010234	J2	I*2	011234	K1	I*2	012234
K2	I*2	013234	Z	I*2	014234	AA	R*4	034234
BB	R*4	034240	RNOI	R*4	034244	BM	R*4	034250
SELECT	I*2	040250	INOI	I*2	040252	NP	I*2	040272
PATH1	I*2	040274	PATH2	I*2	043274	MAXI	I*2	046274
SCALE	R*4	046276	J	I*2	046302	B	I*2	046304
N	I*2	046306	K	I*2	046310	Q	I*2	046312

Local and COMMON Arrays:

Name	Type	Section	Offset	------Size-----	Dimensions
BM	R*4	.$$$$.	034250	004000 (1024.)	(512)
BMF	R*4	.$$$$.	004234	004000 (1024.)	(512)
H	R*4	.$$$$.	000012	000104 (34.)	(17)
INOI	I*2	.$$$$.	040252	000020 (8.)	(8)
J1	I*2	.$$$$.	010234	001000 (256.)	(256)
J2	I*2	.$$$$.	011234	001000 (256.)	(256)
K1	I*2	.$$$$.	012234	001000 (256.)	(256)
K2	I*2	.$$$$.	013234	001000 (256.)	(256)
PATH1	I*2	.$$$$.	040274	003000 (768.)	(768)
PATH2	I*2	.$$$$.	043274	003000 (768.)	(768)
R	I*2	$DATA	000000	000020 (8.)	(8)
RV	I*2	$DATA	000020	000020 (8.)	(8)
SM1	R*4	.$$$$.	000234	002000 (512.)	(256)
SM2	R*4	.$$$$.	002234	002000 (512.)	(256)
U	I*2	.$$$$.	000166	000006 (3.)	(3)
V	I*2	.$$$$.	000160	000006 (3.)	(3)
X	I*2	.$$$$.	000116	000042 (17.)	(17)
Y	R*4	.$$$$.	000174	000040 (16.)	(8)
Z	I*2 Vec	.$$$$.	014234	020000 (4096.)	(8,512)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name	Type	Name	Type	Name	Type	Name	Type	Name	Type
COMP	R*4	IND2	I*2	INT	I*2	MOD	I*2	PRT	R*4
TRAN2	R*4								

FORTRAN IV   Storage Map for Program Unit TRAN2

COMMON Block /   /, Size = 046314 (9830. words)

Name	Type	Offset	Name	Type	Offset	Name	Type	Offset
NS	I*2	000000	I	I*2	000002	L	I*2	000004
S	R*4	000006	H	R*4	000012	X	I*2	000116
V	I*2	000160	U	I*2	000166	Y	R*4	000174
SM1	R*4	000234	SM2	R*4	002234	BMF	R*4	004234
J1	I*2	010234	J2	I*2	011234	K1	I*2	012234
K2	I*2	013234	Z	I*2	014234	AA	R*4	034234
BB	R*4	034240	RNOI	R*4	034244	BM	R*4	034250
SELECT	I*2	040250	INOI	I*2	040252	NF	I*2	040272
PATH1	I*2	040274	PATH2	I*2	043274	MAXI	I*2	046274
SCALE	R*4	046276	J	I*2	046302	B	I*2	046304
N	I*2	046306	K	I*2	046310	Q	I*2	046312

Local and COMMON Arrays:

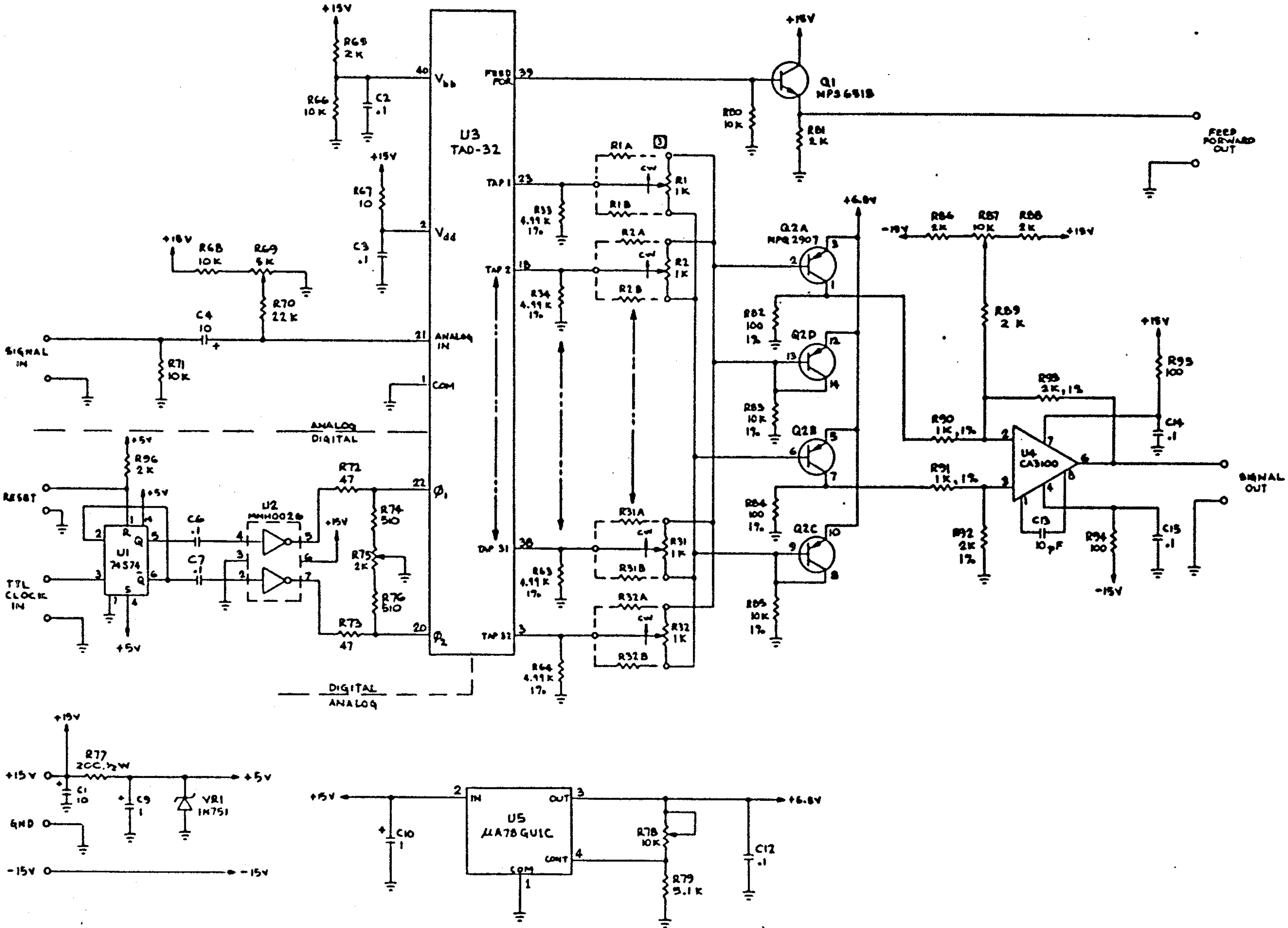
Name	Type	Section	Offset	------Size------		Dimensions
BM	R*4	.$$$$.	034250	004000	(1024.)	(512)
BMF	R*4	.$$$$.	004234	004000	(1024.)	(512)
H	R*4	.$$$$.	000012	000104	(34.)	(17)
INOI	I*2	.$$$$.	040252	000020	(8.)	(8)
J1	I*2	.$$$$.	010234	001000	(256.)	(256)
J2	I*2	.$$$$.	011234	001000	(256.)	(256)
K1	I*2	.$$$$.	012234	001000	(256.)	(256)
K2	I*2	.$$$$.	013234	001000	(256.)	(256)
PATH1	I*2	.$$$$.	040274	003000	(768.)	(768)
PATH2	I*2	.$$$$.	043274	003000	(768.)	(768)
SM1	R*4	.$$$$.	000234	002000	(512.)	(256)
SM2	R*4	.$$$$.	002234	002000	(512.)	(256)
U	I*2	.$$$$.	000166	000006	(3.)	(3)
V	I*2	.$$$$.	000160	000006	(3.)	(3)
X	I*2	.$$$$.	000116	000042	(17.)	(17)
Y	R*4	.$$$$.	000174	000040	(16.)	(8)
Z	I*2 Vec	.$$$$.	014234	020000	(4096.)	(8,512)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name	Type	Name	Type	Name	Type	Name	Type	Name	Type
CODE2	R*4	DECODE	R*4	EXCHG	R*4	GETX	R*4	INPUT	I*2

APPENDIX 5

SCHEMATIC OF THE MATCHED FILTER IMPLEMENTATION


U3
TAD-32
Q1
MPS6518
FEED FORWARD OUT
SIGNAL IN
SIGNAL OUT
RESET
TTL CLOCK IN
U1
74S74
U2
MMH0026
U4
CA3100
U5
μA78GU1C
ANALOG
DIGITAL
Q2A
MPQ2907
Q2D
Q2B
Q2C
VR1
1N751
R77
200, ½W
GND
+15V
-15V
+5V
+6.8V