Loughborough University

This item was submitted to Loughborough University as a PhD thesis by the author and is made available in the Institutional Repository (https://dspace.lboro.ac.uk/) under the following Creative Commons Licence conditions.

creative commons

COMMONS DEED

Attribution-NonCommercial-NoDerivs 2.5

You are free:

- to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

No Derivative Works. You may not alter, transform, or build upon this work.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/

BLLID No. D48155/84

LOUGHBOROUGH
UNIVERSITY OF TECHNOLOGY
LIBRARY

AUTHOR/FILING TITLE
JARMOUKLY, H

ACCESSION/COPY NO.
002364/02

VOL. NO.

CLASS MARK
LOAN COPY

000 2364 02

WEIGHTED DECODING FOR ERROR CORRECTION

By

Hanna Jarmoukly

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the award of

Doctor of Philosophy

of the Loughborough University of Technology

Supervisor: Dr. M.E.Woodward

© Hanna Jarmoukly, June 1983.

Loughborough University
of Technology Library
Oct 83
Class
Acc. No. 002364/02

ACKNOWLEDGEMENTS

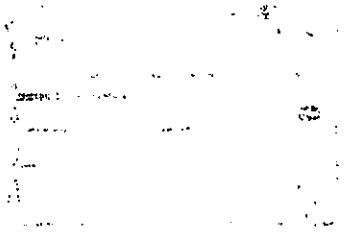
I wish to express my most sincere thanks to my supervisor Dr. M.E. Woodward for his constant guidance and encouragement throughout the course of this work. His comments and criticisms have been appreciated as much as his learned guidance and advise that inspired my work and moulded my thesis.

Many thanks are due to the Syrian SSRC for their grant, and to the staff of the Liason Office in the SSRC for thier care.

I would like to thank the department of Electronic and Electrical Engineering in Loughborough University of Technology for providing all the research facilities.

Many thanks to Mrs. M.A. Hearnden and Mr. G.P. Gerrard from the computer centre for their advice on Plan programming.

My deepest gratitude to my wife and daughter for their moral support. Lastly, I wish to thank Mrs. M.J. Kostic for typing the manuscript of this thesis.



SUMMARY

When digital data is transmitted over a noisy channel, there is always a chance that the received data will contain errors. Usually an error rate is specified above which the received data is considered unusable, and if the channel error rate exceeds this value then error correction coding can be used to reduce the error rate to an acceptable level.

In recent years such coding techniques have become widespread, particularly hard decision decoding which is well established and documented. However, by making use of the additional statistical information in the received signal and making 'soft' decisions, soft-decision decoding can provide improved coding gain and thereby increase the usefulness of a particular code.

Most published results on soft-decision decoding are concerned with performance on random error channels. The present work describes the application of soft decision techniques to burst noise channels and brings to light some of the problems involved.

A new decoding method called parallel threshold decoding is introduced. The resulting decoders are more economical to implement than equivalent soft-decision decoders, yet they are shown to have superior performance on both random and burst noise channels. Performance evaluation was carried out using computer simulation, and also a prototype hardware decoder has been designed, constructed and tested. The improvement using parallel

threshold decoding over conventional hard and soft decision decoding methods predicted by the simulations was verified for the hardware decoder.

The work also includes an investigation into the use of slow microprocessors for implementing error correction coding in fast transmission channels. This leads to the concept of a time shared decoder, where the microprocessor can spend more than the block receive time for decoding an erroneous block. Algorithms which lend themselves to this type of decoding are described and evaluated.

LIST OF MAJOR SYMBOLS AND ABBREVIATIONS

AL1	First soft-decision algorithm
AL2	Second soft-decision algorithm
C	A codeword
E	The error
EPSW	Error-pattern soft weight
E.T.	Error-trapping
g(X)	The generator polynomial
k	No. of information digits in a block
ℓ	The maximum burst length a burst code can correct
n	Block length
NOO	Number of ones in the transmitted codeword
NOZ	Number of zeros in the transmitted codeword
opt	The optimum decoder
PTD	Parallel threshold decoder
q_0	Probability of digit being zero
q_1	Probability of digit being one
R	The received word
S	The syndrome
S.D.	Soft decision
SNR	Signal to noise ratio
SPT	Search parallel threshold
SPTD	Statistical parallel threshold decoder
t	Maximum number of errors a random code can correct
Th	Threshold value
λ	Interlaceing degree

CONTENTS PAGE

Chapter 5

CHAPTER 1

INTRODUCTION

One of the serious problems in a digital data communication systems, is the occurrence of errors in the data transmitted over a noisy channel. The user generally establishes an error rate above which the received data are not usable. If the received data will not meet the error rate requirements, error-correction coding can often be used to reduce errors to a tolerable level. In recent years the use of error-correction coding for solving this type of problem has become widespread. It is now used almost on a routine basis in most new communication systems, because in addition to increasing the energy efficiency of communication links, coding ideas are providing new methods for solving existing problems in communication systems, among many others, the elimination of intersymbol interference caused by either filtering or multipath signals.

Since the appearance of Shannon's classic papers in 1948 and 1949[81], a great deal of research has been devoted to the problem of designing efficient schemes by which information can be coded for reliable transmission across noisy channels. From a practical standpoint, it soon became clear that the real limit on communication rate was set not by Shannon's channel capacity, but by the complexity and the cost of implementing the coding scheme. For this reason during the last

twenty years or so, efforts have been directed towards the design of coding and decoding schemes which could be easily implemented. The first generalised decoding schemes of real significance to emerge were based on algebraic concepts. The algebraic techniques basically involve the simultaneous solution of sets of equation for location and values of errors.[6,7,15]. Nonalgebraic decoding techniques, while accomplishing the same goal, are based upon simple structural aspects of the codes which permit the determination of error-patterns in a more direct fashion[59,61,72]. The introduction of microprocessors, and the dramatic decrease in the cost of solid-state devices, gave designers a higher degree of freedom in implementing more complicated, yet practical error-correction systems. Consequently, researchers have been looking into new ideas to improve the already existing decoding techniques. Although soft-decision decoding was known as far back as 1954[5,82] it has only relatively recently become a practial reality. The additional information provided by the soft-decision in most instances can provide an additional coding gain, which was shown by Wozencraft[94] to be about 2dB, this additional gain can, therefore, significantly increase the usefulness of the code employed. The use of soft-decision with block codes can be divided into two types according to the communication channel noise, firstly, soft-decision decoding for correcting random-errors, and secondly, soft-decision decoding for correcting burst errors. The first type is well studied and

documented, while the second type is not as widely studied due to the complexity of formulating bursty channel statistics. Soft-decision decoding for correcting random errors is particularly effective over a broad range of SNR, starting from low SNR values. Needless to say, at high SNR values the soft-decision decoding performance approaches that of a conventional decoder, up to a point where the soft-decision decoding does not provide any additional gain. It has been shown[18] that for quantization levels of eight or more, no improvement is achieved whether a linear or optimum spacing is used, while an optimum spacing performs slighty better when the number of quantization levels is less than eight. Again it has been shown by many researchers that most of the gain can be achieved by eight level quantization, and that there is no real gain in increasing the number of quantization levels. On the other hand very little is published about soft-decision decoding of burst-errors, and not much is published in particular about the effect of the number of quantization levels on the decoder performance.

The first aim of the work was to introduce an algorithm that can be used by a soft-decision decoder for correcting transmission errors when the transmission is over a bursty channel, then to study the effect of the number of quantization levels on the decoder performance. The soft-decision decoding was expected to perform badly at very low burst SNR values, because

at these SNR values the additional information on which the decoder depends for its improved performance is not correct, due to the high noise power. The next step is to modify the introduced algorithm so it performs better at very low SNR values.

Clearly, the binary channel model[32] is no longer suitable to be used for the soft-decision transmission channel simulation, because its output is either '1' or '0'. Consequently, a new channel model has to be used, where the channel output is a sample of the analogue received signal. In such a channel, a burst is no longer defined by the errors, but by the noise power. Since binary codes are to be used, then it was necessary to make sure that the analogue channel model used for channel simulation in the study of the soft-decision performance is transferable to the binary channel model used in studying binary codes. Obviously, introducing the analogue channel model has to be achieved before continuing to fulfil the first aim.

Most of the soft-decision decoding algorithms are based on the use of microprocessors to execute the complicated decoding algorithm, which in turn complicates the decoder and escalates its cost. Needless to say, that the decoding speed is a deciding factor in a real time decoding. Thus a complicated decoding algorithm, or even a simple one used at high transmission rate channels, might cause decoding problems in real time transmission systems because the lack of speed, which in the best case can be rectified by using faster

microprocessors. Hence escalating the cost further more. The second aim was to look at this problem and find an economical solution, so that slower microprocessors can be used, or very complicated algorithms can be used in high rate transmission channels. Consequently, soft-decision decoding algorithms can be used over a wider range of transmission rates.

The idea of using a slow microprocessor can be explained as the following. During transmission errors tend to happen in some transmitted blocks, the number of these erroneous blocks depends on the noise power value. In general the percentage of these erroneous blocks to the overall transmitted blocks is small. If the decoding process is divided into syndrome calculation, error-detection, and error correction. The decoder is calculating the syndrome only for most of the transmission time, because the received blocks are error-free most of the transmission time, while the decoder has to finish all three phases during the receive time of one block, in order to be usable in real time transmission. The same argument is applicable when a microprocessor is used. Consequently, the microprocessor is idleing there for most of the time, while trying as fast as possible to execute the correction algorithm when an erroneous block is received. Assuming that the micro-processor is allowed to work on a time shared basis, which is, to allow the microprocessor to share the next block correction time for correcting the present erroneous block. Then in real time transmission there is no real

need for the microprocessor to finish the three decoding phases in the receive time of one block. Consequently, the time sharing system can achieve higher transmission rates than the conventional real time decoder using the same decoding algorithm and same microprocessor. Clearly, the time sharing system introduced some delay into the system, and requires buffers at the input and the output of the decoder.

Generally, soft-decision decoding has a number of disadvantages, and a designer must think carefully before using such decoding techniques. As mentioned earlier, the average gain is about 2dB, and for most soft-decision algorithms the use of analogue to digital converters for quantizing the incoming signal is inevitable, so is the microprocessor. These two hardware items are sophisticated, and subsequently costly, and in general their cost increases dramatically with their speed. Hence, although a soft-decision decoder is costly at low transmission rates, it is even more costly at high transmission rates,even without taking into account the software cost. It was found that whatever modifications are carried out on a soft-decision decoding algorithms, these inherited disadvantages, although affected in one way or another, are still there. In view of this, the third aim was to find a new type of decoding that can achieve higher gain, is simple to implement as a hardware, and the use of analogue to digital converters and microprocessors is unnecessary. Such a system will have the advantage that its real time speed is limited only

by the maximum speed of the hardware used and not by the decoding algorithm complexity.

The third aim is achieved in the following way. A hard-decision decoder receives at its input, a binary digit, '0' or '1'. This digit is calculted from the received signal as follows. The demodulator supplies the detector with a sample of the received signal, and the detector set at some threshold value, detects the binary value from this sample. The threshold value is chosen in such a way as to minimize the probability of error in the received symbol, assuming that the probability of a symbol being '1' or '0' is equal. Clearly this strategy works well for the overall transmission, because the probability of '1' and '0' are equal. But it does not minimize the probability of error in a received block, unless the number of '1' and '0' are equal, which could be the case for some codeword, but not for all. Subsequently some additional gain can be obtained if an optimum threshold is used with each block. The soft-decision decoding used the same threshold value as a reference for the confidence number. Thus nothing of the optimum threshold gain is obtained. The parallel threshold decoder is based on the idea of obtaining this additional gain, by using the optimum threshold value of each block for detecting the received samples of that block. Obviously, the use of all possible values for optimum thresholds will require a variable threshold detector, in addition to that the threshold value has to be calculated, which will complicate the decoder. Instead a fixed number of

thresholds are used, the output of each threshold is fed to a subdecoder, each of which consists of a complete decoder that can detect decoding failures. The parallel threshold decoder scans all subdecoders in a preset sequence, and once a codeword or correctable error pattern is detected, the information digits are accepted or corrected accordingly. The scan sequence is chosen so that the parallel threshold decoder tries to minimize the probability of error of the received symbol first, if no correctable error-pattern is found, the decoder attemps to minimize the received word probability of error. The second factor in preseting the scan sequence is to scan all subdecoders according to the weight value of the input threshold, so that the threshold likely to be optimum or near optimum for the highest number of codewords is scaned first.and the threshold likely to be optimum for the next to the highest number is scaned second and so on. The parallel threshold decoder is interesting when error-trapping subdecoders are used, because in addition to the gain descirbed above, an additional gain over the conventional error-trapping decoder can be achieved by converting correctable but untrappable errors into trappable errors, thus reducing the number of decoding failures. The statistical parallel threshold decoder is a further modification of the parallel threshold decoder, it is used for error-correction of burst-errors, where any burst is likely to occur more frequently than any longer burst. The statistical parallel threshold decoder choses the

shortest error-pattern detected by all the subdecoders as the most likely error-pattern to have occured in the channel during transmission, while in turn each subdecoder has chosen previously the shortest possible error-pattern for the received word detected at its threshold. The parallel threshold decoder and the statistical parallel threshold decoder description shows that no analogue to digital convertors or microprocessors need be used, hence overcoming two of the major disadvantages of the soft-decision decoders.

The parallel threshold decoding idea is used in the error-pattern search parallel threshold decoder, and the digit search parallel threshold decoder. The optimum threshold in the error-pattern search parallel threshold decoder is found by searching all the trapped error-patterns and then accepting the one that occured most. While the digit search parallel threshold decoder, corrects on a digit by digit basis, by scanning one bit of all the error-pattern that correspond to a digit of the received word at a time, if the number of bits indicating errors are larger than a certain threshold, that digit is assumed erroneous, and is corrected. And so on for all the received word digits. Here again no analogue to digital convertor is used, but a microprocessor may be required for the error-pattern search parallel threshold decoder.

CHAPTER 2

BINARY CYCLIC CODES AND ERROR-TRAPPING TYPE DECODERS

2.1-Binary Cyclic Codes

During the past ten years, most of the research work done on block codes has been concentrated on a subclass of linear codes, namely, the cyclic codes. There are two reasons for this. Firstly, encoding and syndrome calculation of any cyclic code can be implemented easily by employing simple shift registers with feedback connections[6,55]. Secondly, because of their structure, it is possible to find various simple and efficient decoding methods.

Cyclic codes were first studied by Prange in 1957[70]. Since then, many algebraic coding theorists[6,67,68] studied extensively cyclic codes and their implementations for both random-error correction and burst-error correction.

2.1.1 Description of Cyclic Codes

By definition, an (n,k) linear code is said to be cyclic if for any codeword C, where

$$C = (c_0, c_1, c_2, \ldots\ldots\ldots, c_{n-2}, c_{n-1}) \qquad (2.1)$$

a new word $C^{(1)}$ which is formed by shifting cyclically the components of C once to the right, is a codeword where

$$C^{(1)} = (c_{n-1}, c_0, c_1, \ldots\ldots\ldots, c_{n-3}, c_{n-2}) \qquad (2.2)$$

From the definition, it is clear that

$$C^{(i)} = (c_{n-i}, c_{n-i+1}, c_{n-i+2}, \ldots\ldots, c_{n-1}, c_0, c_1, \ldots$$

$$\ldots, c_{n-i-2}, c_{n-i-1}) \qquad (2.3)$$

is obtained by shifting the codeword C to the right cyclically i places, and is also a codeword.

When the components of the codeword C are treated as coefficients of a polynomial, then

$$C(X) = c_0 + c_1 X + c_2 X^2 + \ldots\ldots + c_{n-1} X^{n-1} \qquad (2.4)$$

if the polynomial C(X) is multiplied by X mod X^n-1, the result

$$XC(X) \bmod X^n-1 = c_{n-1} + c_0 X + c_1 X^2 + \ldots + C_{n-3} X^{n-2} + C_{n-2} X^{n-1} \qquad (2.5)$$

is the polynomial representation of the codeword $c^{(1)}(X)$, and the multiplication by X mod X^n-1 is seen as a cyclic shift to the right of the codeword represented by the polynomial C(X). To obtain the codeword formed by shifting C(X), i shifts to the right, C(X) is multiplied by $X^i C(X)$ mod X^n-1. The result is

$$C^{(i)}(X) = c_{n-i} + c_{n-i+1} X + c_{n-i+2} X^2 + \ldots\ldots + c_{n-1} X^{i-1} +$$

$$c_0 X^i + c_1 X^{i+1} + \ldots\ldots + c_{n-i-2} X^{n-2} + c_{n-i-1} X^{n-1} \qquad (2.6)$$

Conversely, if the codewords of a linear cyclic code are represented by a set of polynomials, then for every codeword polynomial C(X), the code contains all cyclic shifts of C(X). And since the sum of codewords of a linear code is also a codeword of the same linear code, the code must contain all multiples of C(X) mod X^n-1. Which conclude that the codeword polynomials of any linear cyclic code consist of the multiples of some generator polynomial mod X^n-1, where the generator polynomial g(X) is a divisor of X^n-1.

The polynomial representation, enables the development of some important properties[66,67,70,72] for the cyclic codes, which make the simple implementation of encoding and syndrome calculation possible. The cyclic property, and the property that each codeword polynomial is a multiple of the generator polynomial minimize the storage facilities for the encoding dictionary, and make the shift-register devices very easily implemented.

2.1.2 Shortened Cyclic Codes

In certain applications, where the requirements of a system cannot be met by a suitable natural length of a code, it may be desirable to shorten a code to meet these requirements. The shortening is accomplished as follows.

Given an (n,k) cyclic code, assuming the number of information digits required by the system is α, then the β leading information digits of each block are assumed by the encoder and the decoder to be zeros, where

$$\beta = k - \alpha \qquad (2.7\text{-a})$$

It is clear from equation (2.7-a) that for the code to be a shortened code and meaningful, α should satisfy the following condition.

$$k > \alpha > 0 \qquad (2.7\text{-b})$$

Since the β zeros are assumed to be inserted at the encoder and deleted at the decoder, it is easy to see that the shortened code consist of $2^{k-\beta}$ codewords, and that these codewords form an $(n-\beta, k-\beta)$ linear code.

This code is called a shortened cyclic code[67] and is not cyclic. It has at least the same error-correcting capability as the code from which it is derived. The encoding and syndrome calculation for a shortened cyclic code can be accomplished by the same circuits as employed by the original cyclic code. This is because the deleted β zeros do not affect the parity-check calculations. The decoder for the original code can be used for decoding the shortened cyclic code simply by prefixing each received codeword with β zeros. This prefixing can be eliminated, however by modifying the feedback connections[67] of the syndrome register.

Let an (n,k) cyclic code be a shortened code by β, if the correction procedure is unaltered, it would require additional β shifts corresponding to the omitted information digits, before the actual correction process is started. However, decoding can be accomplished more

quickly if, instead of calculating the syndrome of $X^{n-k}R(X)$ as would be done for a cyclic code, the syndrome of $X^{n-k+\beta}R(X)$ is calculated, where $R(X)$ is the received word. This can be achieved by an automatic premultiplication by X^{β} mod $g(X)$. The technique is illustrated in the following example.

Suppose it is necessary to shorten the (15,11) single-error-correcting Hamming code which has the generator polynomial

$$g(X) = 1 + X + X^4 \qquad (2.8\text{-a})$$

by $\beta=5$, so that a new (10,6) single-error-correcting shortened code will result.

The feedback connections for the premultiplication are the remainder obtained by dividing $X^{n-k+\beta}=X^9$ by $g(X)$. The result of the division is

$$P_m(X) = X + X^3 \qquad (2.8\text{-b})$$

The altered Meggitt decoder for the new (10,6) code is shown in fig.(2.1) while the Meggitt decoder for the (15,11) code is shown in fig.(2.2).

It can be seen that feedback connections of $P_m(X)$ in fig.(2.1) are used only when the received word is fed to the syndrome register, otherwise the generator polynomial $g(X)$ feedback connections are used.

The modification advantage over the unaltered

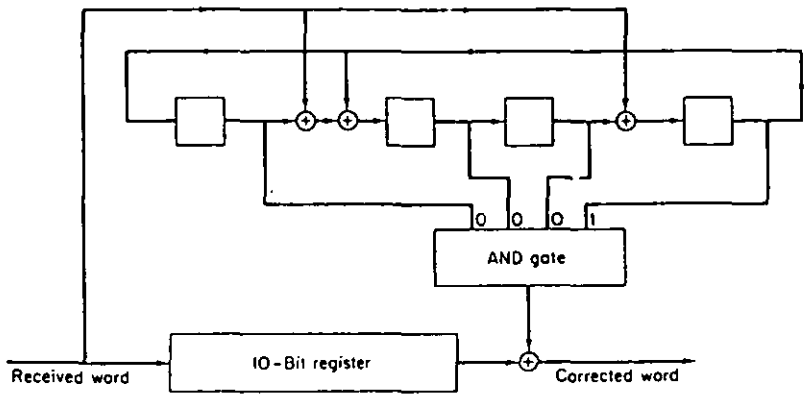


Fig.2.1 A Meggitt decoder for a (10,6) shortened Hamming code.

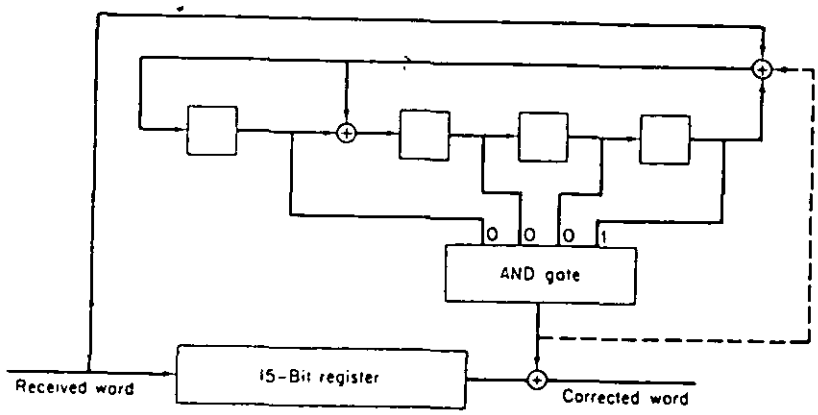


Fig.2.2 A Meggitt decoder for the (15,11) Hamming code.

decoder is the speed of which the correction process is started, and the bigger β the more the speed gain. But that is achieved on the account of complicating the hardware.

2.2-Error-trapping Decoding for Cyclic Codes

Decoding of cyclic codes have been studied extensively[38,66,71,95] by a large number of researchers. The general decoding algorithms for BCH codes given by Peterson[67] are well known. Also, other interesting decoding methods have been given by Meggitt[62], Prange[72], and Kasami[49].

The general decoding method of Meggitt[55,61,62,67] applies to any cyclic code, but refinements are necessary for practial implementations. Error-trapping decoding is based on Meggitt decoding technique, it uses a very simple combinational logic circuit for error detection and correction.

Cyclic codes are divided into two groups. Firstly, cyclic codes for correcting random errors[6,55,67,68]. Secondly, cyclic codes for correcting single-burst errors[3,21,29,36,37,47,48,50,63,77]. Consequently, there are different types of error-trapping decoders, for decoding each group of codes.

2.2.1 Error-trapping Decoding for Correcting Random Errors

Suppose that an (n,k) cyclic code is used for correcting errors in a communication channel, Let C(X) be the transmitted binary codeword and R(X) the received binary word, where C(X) is as in Eq.(2.4)

$$C(X) = c_0+c_1X+c_2X^2+.....+c_{n-1}X^{n-1} \qquad (2.4)$$

and

$$R(X) = r_0+r_1X+r_2X^2+.....+r_{n-1}X^{n-1} \qquad (2.9)$$

depending on the channel noise R(X) may or may not be the transmitted codeword C(X). The decoder calculates the syndrome S(X), which is equal to the remainder resulting from dividing the received word polynomial R(X) by the generator polynomial g(X), i.e

$$R(X) = P(X)\ g(X) + S(X) \qquad (2.10)$$

where S(X) is a polynomial of degree n-k-1 or less. If the syndrome is zero, the received word is a codeword and the decoder will accept the received codeword as the transmitted codeword. If the syndrome is a non-zero vector, the received word is not a codeword, and errors have been detected. Let the error polynomial be E(X) where

$$E(X) = e_0+e_1X+e_2X^2+.....+e_{n-1}X^{n-1} \qquad (2.11)$$

then

$$R(X) = C(X) \oplus E(X) \qquad (2.12)$$

where $\oplus$ represent modulo two addition, since C(X) is a codeword polynomial it must be multiple of the generator polynomial g(X), say

$$C(X) = m(X)g(X) \qquad (2.13)$$

Combining Eqs.(2.10),(2.12), and (2.13), we obtain

$$E(X) = \Big[P(X)+m(X)\Big]g(X)+S(X) \qquad (2.14)$$

let $$P(X)+m(X) = q(X) \qquad (2.15)$$

$$E(X) = q(X)g(X) + S(X) \qquad (2.16)$$

If the errors of E(X) are confined to the n-k parity-check postions of P(X), then E(X) is a polynomial of degree n-k-1 or less. It follows that q(X)=0 in Eq.(2.16) and

$$E(X) = S(X) \qquad (2.17)$$

Thus, correction is done by modulo-2 addition of the syndrome and the n-k received parity check digits. Or, alternatively, by simply outputing the information digits to the data sink if the partiy-check digits are no longer required.

Suppose that the errors are not confined to the n-k parity-check positions of R(X), but are confined to the n-k-1 low-order parity check digits, plus one error in the leading high-order information digits. In such cases the error polynomial E(X) is of degree higher than n-k-1, and correction cannot be done directly. However, if the received word is shifted once to the right cyclically then the shifted error-pattern $E^{(1)}(X)$ is confined to the n-k parity-check digits of the shifted received word $R^{(1)}(X)$. Since from Eq.(2.2) $C^{(1)}(X)$ is a codeword, then the syndrome of $R^{(1)}(X)$ is identical to

the error polynomial $E^{(1)}(X)$ and correction can be made in the same way as in the previous step.

Making use of the cyclic properties of the code the syndrome of $R^{(1)}(X)$ can be calculated from the received word syndrome $S(X)$ by dividing $S(X)$ by the generator polynomial $g(X)$.

Conversely, if the error-pattern is confined to any n-k consecutive position including the end round case, i.e

$$E(X) = e_i X^i + e_{i+1} X^{i+1} + \ldots\ldots + e_{(n-k)+i-1} X^{(n-k)+i-1} \tag{2.18}$$

after n-i cyclic shifts, the error-pattern will be confined to the n-k parity check positions. The corresponding syndrome is calculated by dividing $S(X)$ by the generator polynomial $g(X)$, n-i times. As a result the errors can be corrected.

2.2.2 Error-trapping Decoder for Correcting Random Errors

Given an (n,k) cyclic code, which is capable of correcting all t or fewer random errors in any codeword, an error-trapping decoder is shown in fig.(2.3), the operation of which is described by the following[55,64,65,79].

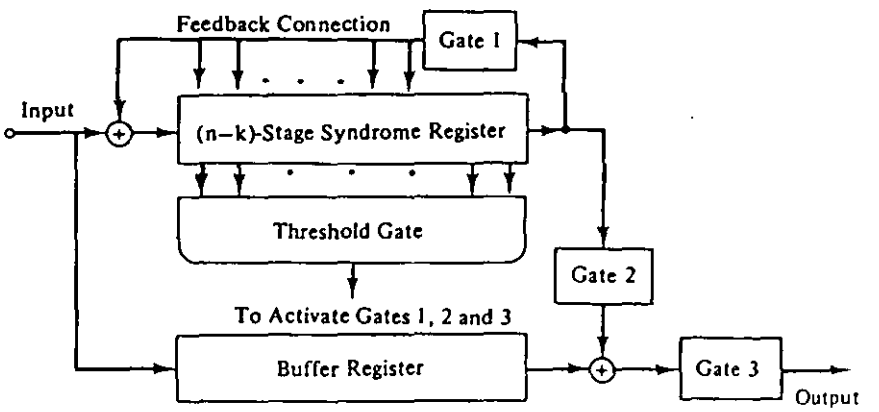


Fig.2.3 An error-trapping decoder

STEP 1. Gate 1 is turned on, Gate 2 and 3 are turned off. The received word R(X) is read into the syndrome register and into the buffer register simultaneously (if the parity-check digits are no longer required, the buffer register has only to store the k received information digits). As soon as the entire received word has been shifted into the syndrome register, the contents of the register is the syndrome of the received word.

STEP 2. The Hamming weight of the syndrome is tested by an (n-k) input threshold gate. The output of this gate is '1' when t or fewer of its inputs are '1'. Otherwise, the output is zero.

STEP 3. a)If the output of the threshold gate is '1', which means either the syndrome is zero and the received word is a codeword, or the Hamming weight of the syndrome is

$$1 \leqslant \omega_h(S) \leqslant t \qquad (2.19)$$

and the errors are confined to the n-k parity-check positions. Consequently, the k received information digits in the buffer register are error-free in both cases. Gate 3 is turned on, and the information digits are sent to the data sink. The syndrome register is set to zero. To correct the next received word go to STEP 1.

b)If the threshold gate output is '0', the syndrome register is then shifted once, with Gate 1 turned on, and Gate 2 and 3 turned off. Go to STEP 4.

STEP 4. a)If the threshold gate output is '1', the errors are confined to the positions $X^{n-1}, X^0, X^1 \ldots X^{(n-k-2)}$ of the received word. The leftmost digit in the syndrome matches the error at the positions X^{n-1} of the received word, the other n-k-1 digits in the syndrome register match the errors at positions $X^0, X^1, \ldots, X^{n-k-2}$ of the received word. The output of the threshold gate turns Gate 1 off and sets a clock to count from 2. The syndrome register is shifted to the right in step with the clock. As soon as the clock has reached n-k, the syndrome register contain '1' in the rightmost position, and zeros in all the rest. The '1' matches the error in position X^{n-1} of the received word. Gates 2 and 3 are turned on, the k information digits are read out of the buffer, while the syndrome register is shifted to the right at the same time. Thus, the first received information digit is corrected by the '1' coming out of the syndrome register, and the decoding is completed. Return to STEP 1.

b)If the threshold gate output is still '0', the syndrome register is shifted once again with Gate 1 turned on, and Gates 2 and 3 turned off. Go to STEP 5.

STEP 5. Step 4b is repeated until the threshold gate output goes up to '1'. If the output is '1' after the i^{th} shift, for $1 \leq i \leq n-k$, the clock starts to count from i+1. At the same time, the syndrome register is shifted with Gate 1 turned off. As soon as the clock

has counted to n-k, the rightmost i digits in the syndrome register match the errors in the first i received information digits in the buffer register. The other information digits are error-free. Gates 2 and 3 are then turned on. The received information digits are read out of the buffer, with shifting the syndrome register to the right at the same time for correction. Return to STEP 1.

STEP 6. If the output of the threshold gate never goes up to '1', by the time the syndrome has been shifted n-k times, Gate 3 is turned on, with Gate 1 still on, and the information digits are read out. At the same time the syndrome register is shifted once for each information digit read out. As soon as the threshold gate output goes up to '1', the contents of the syndrome register match the errors in the rightmost n-k digits in the buffer register. Gate 1 is turned off, and Gate 2 is turned on, the syndrome register is shifted to the right once as every information digit is read out, so that the erroneous information digits are corrected one by one.

STEP 7. The syndrome register is set to zero before starting to calculate the next received word syndrome. Go to STEP 1.

If the threshold gate output never goes up to '1' by the time the k received information digits have been read out of the buffer, then either an uncorrectable error-pattern has occured or a correctable error-pattern with errors not confined to n-k consecutive positions

has occured (untrappable error-pattern).

2.2.3 Error-Trapping Decoding for Correcting Single-Burst Errors

Given an (n,k) cyclic code, which is capable of correcting all single-bursts of length ℓ or less. The error-trapping decoding[1,2,55,67,68] is as described in section 2.2.1 with a slight variation.

Let R(X) be the received word as in Eq.(2.4), and E(X) the error polynomial as in Eq.(2.1), and

$$S(X) = s_0+s_1X+s_2X^2+.....+s_{n-1}X^{n-1} \qquad (2.20)$$

be the syndrome of R(X). If the errors of R(X) are confined to the ℓ high-order parity-check positions $X^{n-k-\ell},.....X^{n-k-2},X^{n-k-1}$ of R(X). Then according to Eq.(2.16), the ℓ high-order bits of S(X) match the errors of E(X), and the remaining n-k-ℓ low-order bits of S(X) are zeros.

Suppose that the errors are not confined to the ℓ high-order digits of the parity-check postions of R(X), but are confined to certain ℓ consecutive positions of R(X). Where

$$E(X) = e_pX^p +e_{p+1}X^{p+1}+.....+e_{p+\ell-1}X^{p+\ell-1} \qquad (2.21)$$

according to the previous discussion, the errors should be confined to the ℓ high-order of an n-k vector, so

that they can be corrected. Let B(X) be a polynomial of degree n-k-1, where the low-order n-k-ℓ coefficients are all zeros, and the high-order coefficients of E(X), then

$$B(X) = 0+0X+0X^{3}+.....+e_{p}X^{n-k-\ell}+....+e_{p+\ell-1}X^{n-k-1} \tag{2.22}$$

the error-pattern E(X) and B(X) can be superimposed if B(X) is shifted j times.

$$E(X) = X^{(j)}B(X) \tag{2.23}$$

where

$$j = p -(n-k-\ell) \tag{2.24}$$

from Eq.(2.16)

$$S(X) = E(X) - q(X)g(X) \tag{2.25}$$

substituting Eq.(2.23), in Eq.(2.25)

$$S(X) = X^{(j)}B(X) - q(X)g(X) \tag{2.26}$$

let

$$i = n-j \tag{2.27}$$

multipling Eq.(2.26), by $X^{(i)}$ yields

$$X^{(i)}S(X) = X^{(i+j)}B(X) - X^{(i)}q(X)g(X) \tag{2.28}$$

by using Eq.(2.27)

$$X^{(i)}S(X) = (X^{n}-1)B(X) - X^{(i)}q(X)g(X) + B(X) \tag{2.29}$$

Since g(X) divides $X^{n}-1$, and B(X) has a degree less than the degree of g(X). Then B(X) is the remainder

error-patterns of length ℓ or less in the high-order part of the syndrome, the decoder detects the n-k-ℓ zeros in the n-k-ℓ low-order part of the syndrome. As soon as n-k-1 zeros are detected, a correctable error-pattern is trapped in the ℓ high-order part of the syndrome, which is of length ℓ or less. The zero detection is accomplished by the use of an n-k-ℓ input OR gate, where the output is '0' when an all zeros input is present, otherwise the output is '1'. The decoding procedure can be described in the following steps:

STEP 1. Gate 1 is turned on, Gates 2 and 3 are turned off. The syndrome is calculated by shifting the received word R(X) into the syndrome register. At the same time, if the parity-check digits are no longer required, the k information digits are shifted into the buffer register.

STEP 2. a)If the OR gate output is '0'. Then, either, the syndrome is zero and the received word is a codeword, or the error is trappable and is confined to the ℓ high-order parity-check digits, which leave the information digits error-free. In both cases, Gate 3 is turned on, with Gate 2 still off, and the information digits are read out to the data sink. The syndrome register is set to zero. Return to STEP 1.

b)If the OR gate output is '1'. Gate 1 is turned on, Gates 2 and 3 are turned off. The syndrome register is shifted. As soon as the OR gate output goes down to '0', the ℓ rightmost stages of the syndrome

register contain the error-pattern. Three phases must be considered, so that the correction can be made.

STEP 3. a)If the OR gate output goes down to '0' after the i^{th} shift for $1 \leqslant i \leqslant n-k-\ell$. Then the errors of the E(X) are confined to the parity-check digits of R(X). Thus, the k received information digits are error-free. Gate 2 is turned on, and the information digits are read out to the data sink. The syndrome register is set to zero. Return to STEP 1.

b)If the OR gate output never goes down to '0' during the first $n-k-\ell$ shifts. Then the burst is not confined to the n-k parity-check digits of R(X).

STEP 4. If the OR gate output goes down to '0' after the $(n-k-\ell+i)^{th}$ shift of the syndrome register for $1 \leqslant i \leqslant \ell$, then the burst is confined to the positions $X^{n-i},\ldots,X^{n-1},X^{0},\ldots,X^{\ell-i-1}$ of R(X). The i bits contained in the ℓ^{th}, $(\ell-1)^{th},\ldots(\ell-i+1)^{th}$ stages of the syndrome register (from the right end) match the error bits at the positions $X^{n-i},\ldots,X^{n-2},X^{n-1}$ of R(X). At this instant, a clock starts to count from $(n-k-\ell+i+1)$. The syndrome register is shifted (in step with the clock) with Gate 1 turned off. As soon as the clock has counted up to n-k, the i rightmost bits in the syndrome register match the error in the first i received information digits in the buffer register. Gates 2 and3 are then turned on. The received information digits are read out of the buffer for correction. The syndrome register is set to zero. Return to STEP 1.

STEP 5. If after the syndrome register has been shifted n-k times, the OR gate output never goes down to '0', then Gate 3 is turned on and the received information digits are read out of the buffer one at a time. At the same time, the syndrome register is shifted with Gate 1 turned on. As soon as the OR gate output goes down to '0'. the contents of the ℓ rightmost stages of the syndrome register match the errors in the next ℓ received information digits to come out of the buffer. Gate 2 is turned on and the erroneous information digits are corrected by the digits coming out from the syndrome register with Gate 1 turned off.

If the OR gate output never goes down to '0' by the time the k information digits have been read out of the buffer, then a burst of length longer than ℓ have been detected.

2.2.5 Properties of Error-Trapping Decoding

Efficiency of error-trapping decoding, depends greatly on the ability to trap the error-pattern, which contaminated the transmitted codeword. If the uncorrectable error-patterns (which are beyond the correction ability of the code used) are excluded, error-trapping decoding decodes effectively all single-error-correcting codes, and single-burst-error-correcting codes. It is also effective for decoding some double-error-correcting codes which have a low rate. But when it is applied to high rate codes, with large error-correcting capability, it becomes very ineffective, and will be able to correct

only a small percentage of the total correctable errors[55]. While the code rate does not affect the performance of the decoder when used to decode burst-error-correcting codes.

The main disadvantage of the error-trapping decoding is the disability to trap all the correctable error-patterns. In general this disadvantage increases the higher the code rate is (excluding the single-error-correcting codes, and single burst-error-correcting codes). The other disadvantage is that unlike some other decoders, the error-trapping decoders will correct only t errors or less for random-error-correcting codes, or a burst of length ℓ or less for burst-error-correcting codes, and will not consider any more than t or longer then ℓ error-patterns even if the code can correct these error-patterns.

On the other hand, the main advantage is that, in the case of decoding failure, where the received word is not a codeword, or is not corrected to a codeword, either because the error-pattern is uncorrectable or is untrappable, the decoder will inform of such an occurrance, and an appropriate action can be taken, i.e. ask for retransmission of the erroneous word, if this facility is available. The other advantage is that the combinational logic circuit is very simple, and inexpensive. So that a complete decoder for a certain code can be built easily on one integrated circuit.

Because of the advantage, a great deal of work

is done to overcome the disadvantages. In an effort to extend the application of the random-error-trapping decoding to multiple-error-correcting codes, several modifications have been devised[49,57,73,79,80,90], while Gallager[31] proposed an optimum decoding method for burst-error-correcting codes based on the burst-error-trapping decoding.

2.3-The Optimum Decoding for Burst-Error-Correcting Codes

Although the decoder described in section (2.2.4) is efficient in correcting all burst error of length ℓ or less, it is still a fraction of the correctable error-patterns (coset leaders).

The number of possible error-patterns of length ℓ or less, in a word of length n is $n2^{\ell-1}$, the total number of the correctable burst errors of length n-k is 2^{n-k}. Then the ratio of the correctable error of length ℓ or less is

$$R_c = \frac{n2^{\ell-1}}{2^{n-k}-1} \qquad (2.30)$$

The most efficient burst-error-correcting codes are the optimal codes, which meet the Reiger bound[77]. They satisfy the condition

$$\ell_{opt} = \frac{n-k}{2} \qquad (2.31)$$

Substituting Eq.(2.31) in Eq.(2.30)

$$R_c = \frac{n2^{\frac{n-k}{2}-1}}{2^{n-k}-1} \qquad (2.32)$$

it can be seen from Eq.(2.32) that for large n, R_c is a small fraction. It is also clear that R_c is even smaller for nonoptimal codes.

Gallager[31] introduced a modification to the burst-error-trapping decoder in such a way that it corrects all the correctable burst errors of length n-k or less; that is, besides correcting all bursts of length ℓ or less, the decoder also corrects those bursts of length $\ell+1$ to n-k, which are used as coset leaders.

An optimum burst-error-correcting decoder for a cyclic code is defined as a decoder which, given the received word R(X), selects C(X) as the transmitted codeword, for which R(X)-C(X) contains the shortest error burst. Such a decoder would minimize the probability of decoding error on a channel for which each burst of any given length is less likely than each burst of any shorter length.

The performance of the optimum decoder is plotted in fig.(2.5). It is interesting to notice that when n-k and ℓ are large, most bursts are corrected for $\ell' < n-k-\log_2 n$ where $\ell' \geq \ell$. It can be seen from fig.(2.5) that increasing the burst correcting capability ℓ raises the flat part of the exponent function of the uncorrectable bursts $e(\ell')$. Decreasing the fraction of uncorrectable bursts in the vicinity of $\ell' = (n-k)/2$.

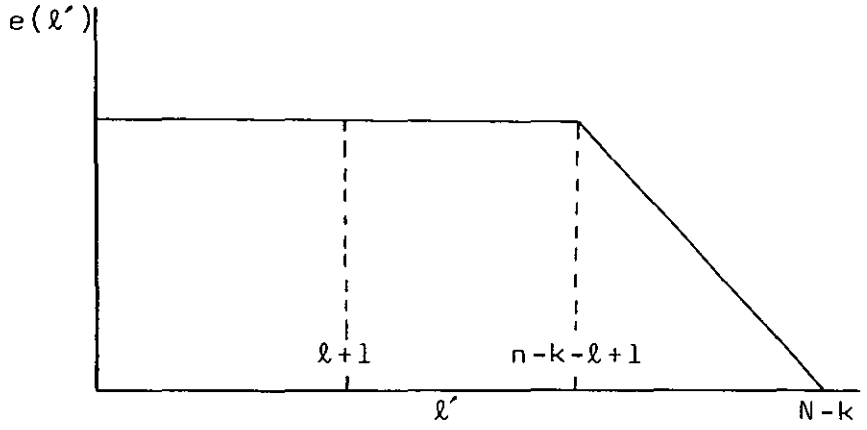


fig.(2.5)* Exponent for fraction of uncorrectable burst of length ℓ′

Unfortunatley, relatively little is known about how to choose the generator polynomial g(X) for a given n and k to maximize ℓ. Fire[26] has developed a large class of cyclic codes with reasonably large values of ℓ, Elspas and Short[21] have published a short table of cyclic codes with optimum values of ℓ. Lin[55] published some efficient cyclic codes and shortened cyclic codes. Kasami[48,50] has also given a table of shortened cyclic codes with optimum value of ℓ.

2.3.1 An Optimum Decoder for Correcting Single-Burst Errors

An optimum decoder is shown in fig.(2.6), assuming that the decoder is used to decode an (n,k) cyclic code, which is able to correct all burst of errors of length ℓ or less. The decoding procedure can be described in the following steps:

STEP 1. Gate 1 is turned on, Gate 2 and 3 are turned off. The received word R(X) is read into the syndrome register, and into the buffer register

---

*This figure is taken from Gallager's book Information Theory and Reliable Communication", ref.31.

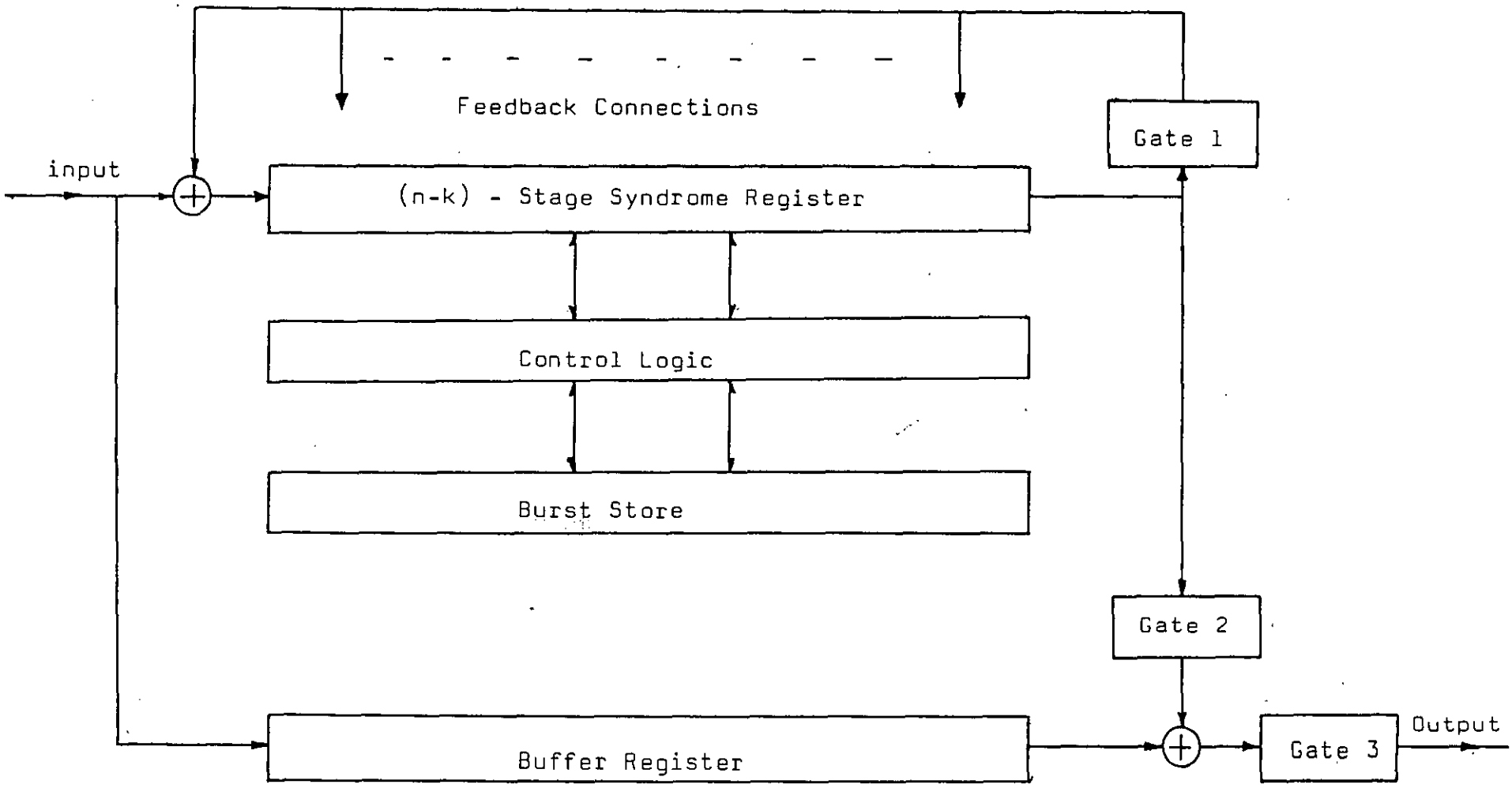


Fig.2.6 An optimum decoder for single-burst-error cyclic code

simultaneously (when the parity-check digits are not required, the buffer register has only to store the k received information digits). As soon as the entire received word has been shifted into the syndrome register, the contents of the register is the syndrome of the received word.

STEP 2. The syndrome is tested, if it is zero, then the received word is a codeword. Gate 3 is turned on, with Gate 2 turned off, and the information digits are read out to the data sink. The syndrome register is set to zero. Return to STEP 1. If the syndrome is not zero, n-k '1' are stored in the burst store, and the syndrome register is shifted once with Gate 1 turned on and Gates 2 and 3 turned off.

STEP 3. The syndrome content is tested by the control logic, if the burst error is not confined to the rightmost digits of the syndrome register go to STEP 4. Otherwise the burst length is calculated. If the length of the burst error which is in the syndrome register is found to be less than the length of the burst error in the burst store, then, the content of the syndrome register is transferred to the burst store. Otherwise, the content of the burst store is kept unchanged.

STEP 4. With Gate 1 turned on, and Gate 2 and 3 are turned off. The syndrome register is shifted n times, STEP 3 is repeated after each shift. After the n shifts, the contents of the syndrome register will be the syndrome of the received word, while the burst store

will contain the shortest burst error.

STEP 5. The syndrome register is shifted, with Gate 1 turned on and Gates 2 and 3 turned off. As soon as the contents of the syndrome register contain the shortest error burst, i.e. its content match the content of the burst store, the correction can be made. Let the syndrome register contain the shortest error burst after the i^{th} shift, then three stages should be considered.

STEP 6.a)If i=0, then the burst errors confine to the n-k parity-check digits, and the information digits in the buffer register are error-free. Gate 3 is then turned on, and with Gate 2 turned off, the information digits are read out to the data sink. Thus the decoding is completed. The syndrome register is set to zero. Return to STEP 1.

b) If $1 \leqslant i \leqslant n-k$, the clock starts to count from i+1. At the same time, the syndrome register is shifted with Gate 1 turned off. As soon as the clock has counted to n-k, the rightmost i digits in the syndrome register match the errors in the first i received information digits in the buffer register. The other information digits are error-free. Gates 2 and 3 are then turned on. The received information digits are read out of the buffer for correction. Return to STEP 1.

c) If the contents of the syndrome register never matched the burst store by the time the syndrome register has been shifted n-k times (with Gate 1 turned on), Gate 3 is then turned on and the received information digits are read out of the buffer one for each shift of

the syndrome register. As soon as the syndrome register is shifted i times, its content match the errors in the rightmost n-k digits in the buffer register. Gate 1 is turned off, and Gate 2 is turned on and the erroneous information digits are corrected by the digits coming out from the syndrome register. As soon as the k information digits are read out of the buffer register, the syndrome register is set to zero. Return to STEP 1.

Depending on the design of the logic circuit, some times it could be better to store the burst-error length and the number of shifts, of the burst to be stored in the burst store instead of storing the actual burst. Another suggestion is to store the burst-error length, and set an n counter to zero, then increment the counter with each shift of the syndrome register. The same burst error will appear in the syndrome register once n shifts are completed, i.e. the n counter returns to zero.

A number of modifications are sometimes desirable in such a decoder.For example, the 'round the end' bursts can be ignored since they are usually much less likely to happen than the ordinary bursts. Such a modification will simplify the decoding process greatly, in addition to the simplification of the control logic. Another modification is to count as a detected error any burst longer than a given length. This modification is of great significance when a retransmission facility is available. Finally, in some cases when two or more bursts of the same length as the shortest burst are

present, the decoder is confused in determining which burst to use. The detection process can be modified so that, the decoder can look at both the number of errors in the burst and the length of the burst to decide which burst is the more likely one.

On one hand, the optimum decoder has the advantage of being able to correct burst errors longer than ℓ. On the other hand, it has the disadvantage of being unable to detect uncorrectable errors. To overcome this disadvantage the second modification described above is used. Let an (n,k) cyclic code be used, where n is large. According to fig.(2.5) the optimum decoder can correct all burst-error of length ℓ' or less where

$$\ell \leq \ell' \leq n-k-\ell+1 \qquad (2.33)$$

depending on Eq.(2.33), the optimum decoder can be modified to accept the corrected word as the transmitted codeword, whenever the detected burst error is of length ℓ' or less. And to inform of the detection of an erroneous word, when the detected burst error is longer than ℓ', so that an appropriate reaction can be taken. A modified optimum decoder in the way described above is more complicated than the ordinary optimum decoder which is in turn more complicated than the error-trapping decoder.

2.4-Soft-Decision Decoding

In a communication system using error-correcting codes for error control, the transmitted bit stream

usually carries a great deal of information about the noise in the channel. Although, in general, when dealing with binary codes, the decoding techniques developed [31,49,55,67] assume a channel whose output is also binary, for many communication applications this assumption is not necessary and furthermore, to make a 'hard' decision without regard to these additional noise information is to throw information away and degrade the performance of the system. This situation was tolerated for a time because it was thought that the loss in performance was justified by the simplicity of the digital decoder. Recently, the great advance in electronic technology, makes this justification come into question, and there have been many proposals for reducing this performance loss through modified decoders [13,22,23,24,25,27,33,34,35] which takes advantage of the additional information by abstraction of the channel measurement information.

Failure correction, or forced erasure detection, or null-zone decoding [8,14, 41,42,45,51,54,58,69,83,84,92] can be considered as the first step in soft-decision decoding. Received signal elements lying on both sides near the threshold level, are passed to the decoder labelled as erasures, so that the decoder has some knowledge of where the errors are likely to be. Because the channel noise added to the transmitted elements is likely to have affected the elements near the threshold more than the far elements from the threshold level.

The next step nearer to soft-decision decoding was the extention of the null-zone detection, to the double null-zone detection, which gave an improvement in the performance[54] over the null-zone detection. The generalisation to more than two null-zones, with the improvement in the performance rise as the number of zones is increased. This general form of null-zone detection is called soft-decision decoding[13,20,34,40, 85,86,87].

An early example of the use of channel measurement information with block codes is given by Wagner decoding[5,82] and its generalization[30], where channel measurement information is used to extend by 1 the error-correcting capabilities of a code whose minimum distance is an even number. Recently considerably more sophisticated approaches for using channel measurement information with block codes have been developed[10,12,13,19,22,23,24, 25,27,33,34,35,59,91].

A block diagram of a communication system using a soft-decision is shown in fig.(2.7), the information digits are encoded to give the transmitted codeword, these binary digits are fed into a data modulator, which determines the transmitted waveform x(t). When a binary channel is assumed, the data demodulator produces a sequence of n binary digits R where r_i are the coefficients of R(X) in Eq.(2.9)

$$R = r_0, r_1, r_2, \ldots, r_{n-1} \qquad (2.34)$$

which are based on the received waveform y(t). In the case of soft-decision decoding the data demodulator will supply the binary sequence R, and in addition, a sequence of n positive numbers denoted by α, where

$$\alpha = \alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{n-1} \tag{2.35}$$

will be supplied. These positive numbers, called the channel measurement information, are used by the decoder to provide a measure of the relative reliability of the received binary digits. Then if

$$\alpha_i > \alpha_j \tag{2.36}$$

the decoder shall assume that r_i is more likely to be correct than r_j. Each value of α coefficients α_i is viewed as a confidence value on the reliability of each received digit. Since the decoder is fed both the received word R(X) and the sequence α, then the decoder is no longer a true binary decoder.

For many applications the abstraction of channel measurement information is relatively a simple matter. For example, if the magnitude of the decision statistic of each received digit is assumed to be monotonically related to the probability that the digit is received correctly, the required channel measurement information can be obtained by simply replacing the 1-bit output device by a J-bit analogue-to-digital converter. These J-bits represent Q quantization levels symmetrically spaced about the hard-decision boundary, where

$$Q = 2^J \tag{2.37}$$

Let us assume that the waveform representing the i^{th} element has entered the demodulator the output is the estimate of the received i^{th} binary digit given by the J-bit word. The output of the quantizer v_i is

$$v_i = v_{i,1}, v_{i,2}, v_{i,3}, \ldots\ldots\ldots, v_{i,J} \tag{2.38}$$

where the first bit $v_{i,1}$ is the hard-decision and the remaining J-1 bits give an indication of the confidence of that estimate. The confidence number α_i of the i^{th} element is defined as

$$\alpha_i = v_{i,2}, v_{i,3}, \ldots\ldots\ldots, v_{i,J} \qquad \text{when } v_{i,1} = 0 \tag{2.39-A}$$

$$\alpha_i = (v_{i,2}, v_{i,3}, \ldots\ldots, v_{i,J}) \oplus (1,1,\ldots\ldots\ldots,1) \qquad \text{when } v_{i,1} = 1 \tag{2.39-B}$$

where $\oplus$ represent modulo-two addition, from Eqs.(2.38) and (2.39) the quantizer output v_i can be represented as

$$v_i = v_{i,1}, \alpha_i \tag{2.40}$$

The confidence number α can be used either in the binary or the decimal form depending on the type of algorithm used for correction.

2.4.1 Improvement Over The Hard-Decision Decoding

First, consider the random-error channel. Let d_h be the hard minimum distance of code, then its bounded-distance hard correcting power is the largest integer.

$$t_h \leqslant (d_h-1)/2 \tag{2.41}$$

In the soft-decision sense codewords (paths) are $\geqslant d_s=(Q-1)d_h$ soft-decision levels apart, and therefore the bounded distance guaranteed soft-decision error correction power in levels is

$$t_s \leqslant (d_s-1)/2 \tag{2.42}$$

and Q/2 or more soft-decision errors will result of a hard-decision error, since the soft-decision decoder can correct t_s soft-decision errors, then the number of hard-decision errors that can be corrected by the use of soft-decision technique is

$$t_h = t_s/(Q/2) \tag{2.43}$$

Substituting Eq.(2.42) in Eq.(2.43)

$$t_h = \frac{2}{Q}\left[(d_s-1)/2\right] \tag{2.44}$$

$$t_h = \frac{1}{Q}\left[(Q-1)d_h-1\right] \tag{2.45}$$

for large Q

$$t_h \simeq d_h \qquad (2.46)$$

Thus, the correction power of the code is doubled.

It should be noted that the doubling of the correction power of a code is an upper bound on the improvement, and will be achieved only at very high SNR. While, at low SNR the average improvement will be significantly less than this.

Considering a Gaussian channel as the transmission media, the soft-decision decoding has an improvement of 3 dB in coding gain[25,94] for a very high SNR. For a noisy channel the improvement will drop to 2 dB. These gains are obtained when the quantization levels are infinite. For more practical values of quantization levels, the improvement is less, but fortunately the degradation in performance due to using fewer quantization levels is not linear so the degradation involved[18,44] in using 8-level equal-spacing quantization is only about 0.2 dB, which gives an improvement of 2.8 dB at high SNR, and 1.8 dB for low SNR values.

In case of Rayleigh channel, the soft-decision decoding is capable theoretically[25] of providing much larger coding gain than in the case of the Gaussian channel. For example at high SNR the soft-decision decoding requires approximately half the SNR in dB to achieve the same output bit error as hard-decision decoding. It must be noted, however

that the expected halving in power requirement will not be achieved at low SNR. In general, the increase in improvement where soft-decision techniques are applied to a non-Gaussian channel is more than the improvement in the Gaussian channel case.

As mentioned earlier the abstraction of channel measurement is done by quantizing the incoming signal into Q levels symmetrially spaced about the hard-decision threshold, but these Q levels, although assumed linearly (equally) spaced up to now, need not be. In fact, it has been shown[40,46,60,76] that a non-linear spacing may be optimum. In the case of a Gaussian channel a 3-level optimum-spacing quantization degrades the code gain by about 1dB only.

If we next consider a burst-error channel, it is not possible to derive a theoretical soft-decision improvement figure for this, because of the lack of a simple burst-noise model. In general, the burst channel can be considered[9] to be a diffused-burst channel in which error bursts are separated by relatively short gaps of low density of errors. Therefore any code used on such a channel must have burst and random-error-correction capability to achieve high improvement. However, one method of evaluating the improvement is by simulation.

Given a random-error-correcting code capable of correcting t random errors, this same code is capable of correcting all burst errors of length t or less. Since the use of soft-decision decoding extend the

random-error-correcting capability to 2t according to Eq.(2.46), then the same code can correct all burst errors of length 2t or less if soft-decision decoding is used.

Conversly, one can assume that the soft-decision decoding doubles the power of burst-error-correcting codes.

2.4.2 Error-pattern Soft Weight Calculation

The data modulator of fig.(2.7) will output two sequences, when the waveform y(t) is received

$$R = r_0, r_1, r_2, \ldots\ldots, r_{n-1} \tag{2.34}$$

which is the hard-decision estimate of the transmitted codeword, and the confidence number α for the received word elements.

$$\alpha = \alpha_0, \alpha_1, \alpha_2, \ldots\ldots, \alpha_{n-1} \tag{2.35}$$

where each element α_i is calculated as in Eq.(2.39)

let
$$\beta = \beta_0, \beta_1, \beta_2, \ldots\ldots, \beta_{n-1} \tag{2.47}$$

where
$$\beta_i = \text{decimal value of } (\alpha_i) \tag{2.48}$$

it can be seen from Eq.(2.39) that β_i is large, when r_i is likely to be correct, and is small, when r_i is likely to be erroneous.

Assuming the error-pattern that corrupted the transmitted codeword is

$$E(X) = e_0 + e_1 X + e_2 X^2 + \ldots\ldots\ldots + e_{n-1} X^{n-1} \tag{2.11}$$

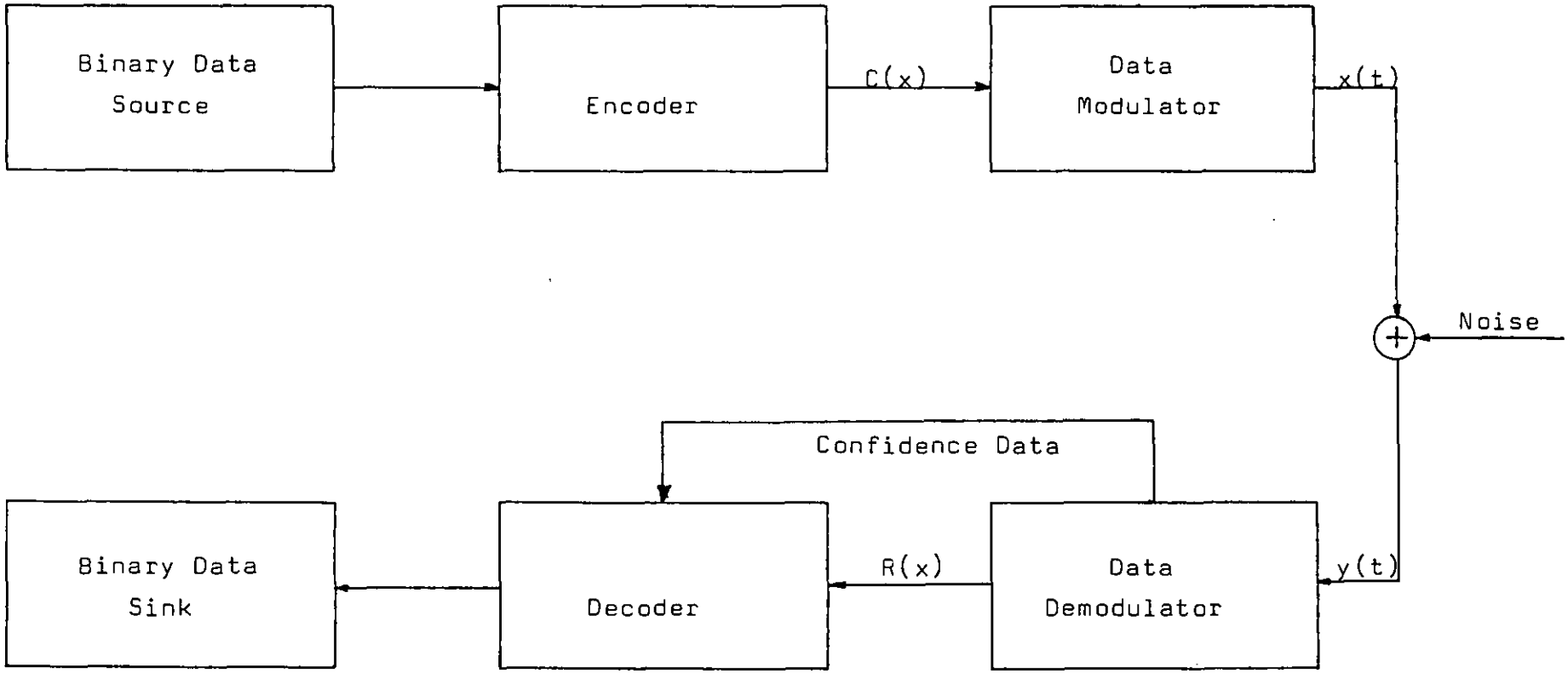


Fig.2.7 Communication system block diagram

then, the error-pattern soft weight (EPSW) is

$$EPSW = \sum_{i=0}^{n-1} e_i \cdot \beta_i \qquad (2.49)$$

it is clear that the value of EPSW is relatively small, since the confidence numbers β_i should be small for all the error values.

Soft-decision is simply calculating the soft weight for all the possible error-patterns, and accepting the error-pattern which has the smallest EPSW as the error-pattern that corrupted the transmitted codeword.

If a cyclic code is used, then all the possible error-patterns are calculated by shifting the syndrome register n times, after calculating the syndrome. In fact, significantly fewer additions than in Eq.(2.49) are required, if one observes that, in such cases, the error-pattern is confined to the n-k syndrome bits only. Assuming that β is shifted cyclicaly once with each shift of the syndrome register (after the syndrome is calculated), then Eq.(2.49) can be rewritten as

$$EPSW = \sum_{i=0}^{n-k-1} s_i . \beta_{i+j} \qquad (2.50)$$

where s_i is the i^{th} coefficient of the syndrome S(X)

$$S = s_0 + s_1 X + s_2 X^2 + \ldots\ldots + s_{n-k-1} X^{n-k-1} \qquad (2.51)$$

and j is an adjustment factor so that β_{i+j} will correspond to the digit represented by s_i

2.4.3 Soft-Decision Decoder

A soft-decision decoder based on the above facts is shown in fig.(2.8). The decoding procedure can be described in the following steps:

STEP 1. Gate 1 is turned on, Gates 2 and 3 are turned off. The syndrome S(X) is formed by shifting the entire received word R(X) into the syndrome register. At the same time, the received word is stored in the buffer register (if parity-check digits are no longer needed, only the k information digits are stored).

STEP 2. If the syndrome is zero, the received word is a codeword, Gate 3 is turned on, with Gate 2 still off. The data is read out to the data sink. The syndrome register is set to zero. Return to STEP 1. Otherwise a decimal number larger than $(n-k)Q/2$ is stored in the EPSW store.

STEP 3. The syndrome content is tested. If the rightmost digit of the syndrome register is '1', then the EPSW of the error-pattern present at the syndrome register is calculated. If the EPSW calculated is smaller than the value stored in the EPSW store, then the new EPSW is stored in the EPSW store, and the error-pattern in the syndrome register is stored in the error-pattern store. Otherwise the EPSW and the error-pattern are kept unchanged.

STEP 4. Gate 1 is turned on, Gates 2 and 3 are turned off. The syndrome register is shifted n times, STEP 3 is repeated after each shift. After the n shifts, the contents of the syndrome register is the syndrome

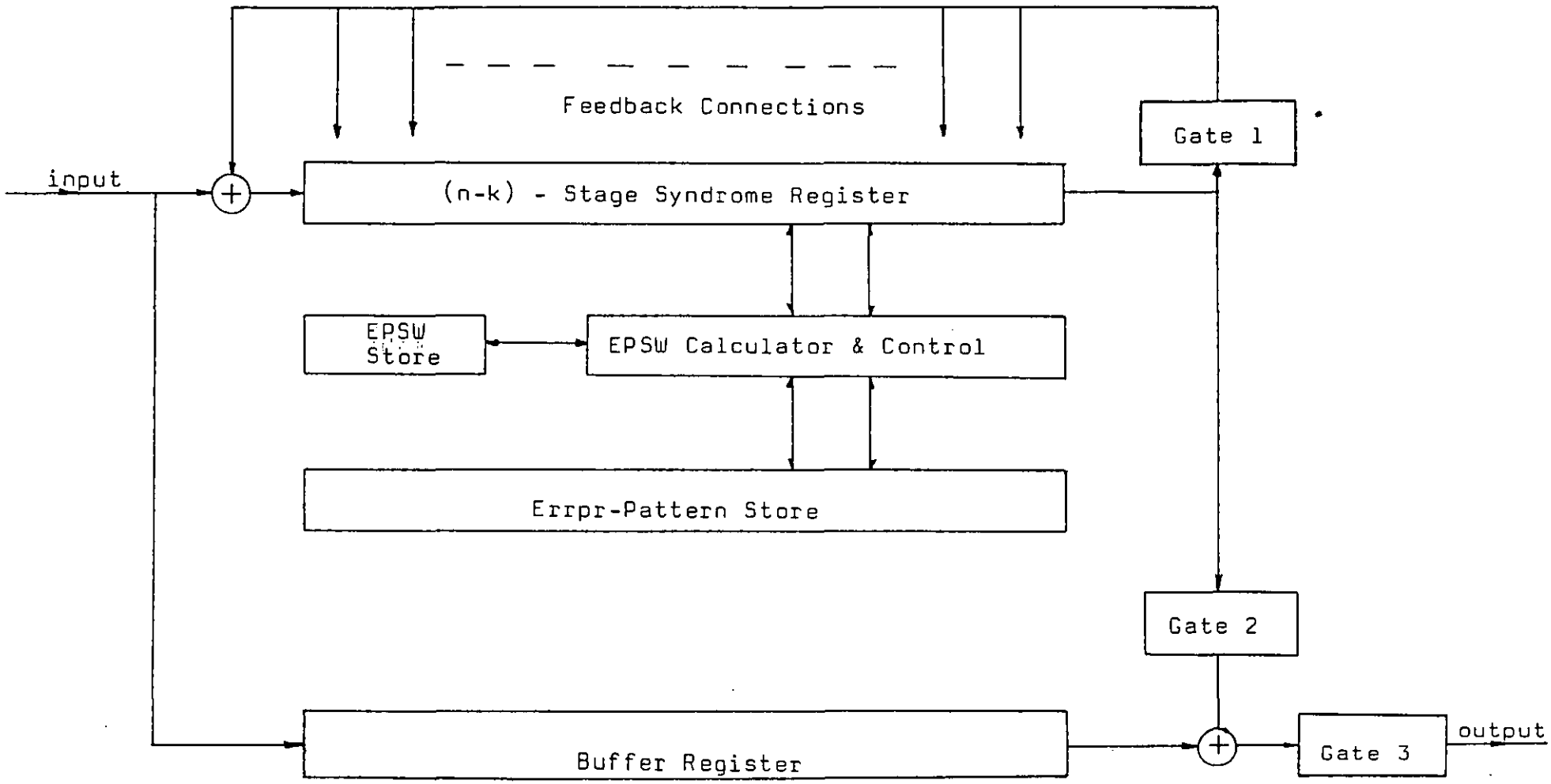


Fig.2.8 A soft-decision decoder for random-error-correction cyclic code

of the received word, while the error-pattern store contains the error-pattern which has the smallest EPSW i.e. the error-pattern that is chosen by the decoder, as the corrupting error-pattern.

STEP 5. Gates 1,2, and 3 as in STEP 4. The syndrome register is shifted,let the contents of the syndrome register match the contents of the error-pattern store after the i^{th} shift. Then three phases should be considered:

STEP 6. a)If i=0. The errors confine to the n-k parity-check digits, and the information digits are error-free. Gate 3 is turned on with Gate 2 turned off. The information digits are read out to the data sink. The syndrome register is set to zero. Return to STEP 1.

b)If $1 \leqslant i \leqslant n-k$. The clock starts to count from i+1. At the same time, the syndrome is shifted with Gate 1 turned off. As soon as the clock count reaches n-k, the rightmost i digits in the syndrome match the errors in the first i received information digits in the buffer register. Gates 2 and 3 are turned on, the information digits are corrected as they are read out of the buffer register. Return to STEP 1.

c)If the contents of the syndrome register never matches the error-pattern store by the time the syndrome register has been shifted n-k times (with Gate 1 turned on), Gate 3 is turned on and the received information digits are read out one for each shift of the syndrome register. As soon as the syndrome register is shifted i times, its contents match the error in the n-k rightmost digits of the buffer register. Gate 1 is

turned off, Gate 2 is turned on, the erroneous information digits are corrected as they are read out of the buffer register by the bits coming out of the syndrome register. As soon as all the received word digits are read out of the buffer register, Gate 3 is turned off. The syndrome register is set to zero. Return to STEP 1.

The increased coding gain achievable with this decoder over a hard-decision decoder is dependent on whether the most trappable error-patterns are confined to an n-k consecutive positions i.e. whether they are present as one of the calculated error-patterns or not. Since the number of error-patterns confined to n-k position is proportional to n-k, then this decoder is most effective for decoding single and double-error-correcting codes and the improvement in coding gain will decrease as the code rate is increased. For high rate codes, it will become very inefficient, and much of the correction power will be lost.

CHAPTER 3

THE COMMUNICATION CHANNEL

A communication channel might represent any medium by means of which the signal is transmitted or stored. A typical transmission channel is a telephone line and a typical storage device is a magnetic-tape unit including writing and reading heads. The channel is usually subject to various types of noise disturbances; for example, time-varying frequency response, and impulsive switching noise, for the transmission channel, while dirt particles and defective tape material are common noise sources for the storage channel. A block diagram of a typical data communication system is shown in fig.(3.1). The source encoder converts the data generated by the source into binary data, the channel encoder attaches parity-check digits to the data digits and this output is a sequence of digits $a_1,\ldots,a_n$. These digits are produced at a fixed rate, say one digit every τ secs. In each interval of τ seconds, the modulator produces waveforms, each of duration τ, $x_1(t),\ldots,x_n(t)$. Each waveform $x_i(t)$ is determined by the digit a_i entering the modulator. These $x_i(t)$ are transmitted through the channel where they are contaminated by noise. At the other end of the channel the demodulator receives the waveform $y_1(t),\ldots,y_n(t)$.

$$y(t) = x(t) + w(t) \qquad (3.1)$$

where

$$y(t) = \Sigma\, y_i(t - i\tau) \qquad (3.2)$$

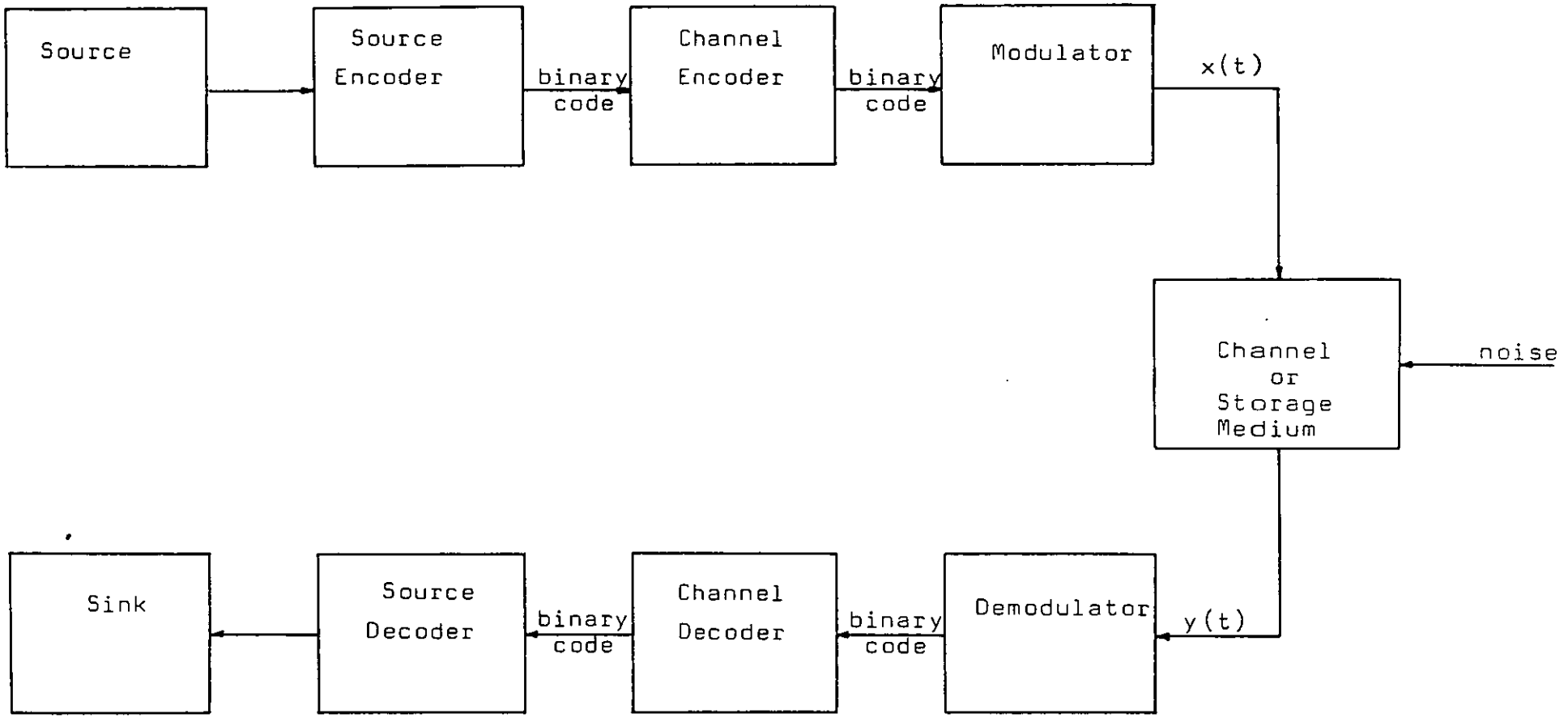


Fig.3.1 Block diagram of a typical communication system

$$x(t) = \Sigma\ x_i(t - i\tau) \qquad (3.3)$$

and

$$w(t) = \Sigma\ w_i(t - i\tau) \qquad (3.4)$$

where $w(t)$ is the channel noise added to the tranmitted waveform. The demodulator takes the received waveform from the channel and converts it into a sequence of digits $b_1,\ldots,b_n$, at a rate of one digit each τ seconds. The channel encoder will produce binary data from the b_i sequence which hopefully will be the same data generated by the source, after the source decoder has converted this sequence.

In more sophisticated cases, the output from the demodulator will contain information about how reliable the b_i sequence is, and in this case the demodulator will output in additional to the sequence b_i another sequence or sequences, which will contain the reliability information.

A channel can be specified in terms of the set of inputs available at the input terminal, the set of outputs available at the output terminal, and for each input the probability measure on the output events conditional on that input.

Considering this definition, a channel can be one of many kinds, for example, in fig.(3.1), the channel can be considered continuous in time, in which the input and output are waveforms i.e. the channel input is the modulator output $x(t)$, the output is the demodulator

input y(t), and the channel is the block named 'channel or storage medium '. On the other hand, the channel can be discrete in time, in which case the input and output are sequences belonging to the set of real numbers i.e. the channel consists of the modulator, the channel, and the demodulator, in which the input is the discrete sequence $a_1,.....,a_n$, and the output is the discrete sequence $b_1,.....,b_n$. The choice of channel is dependent entirely on the stages one is interested in.

If one is interested primarily in the encoder and decoder, then it is convenient to consider the modulator and the demodulator as being part of the channel. On the other hand, if one is interested in a channel in which the input is discrete and the output is continuous, then the appropriate channel to consider is the one which contains the modulator and the channel block, where the input is $a_1,....,a_n$, and the output is y(t).

3.1-Random-errors and burst-errors channels

In order to predict the performance of a code, it is necessary to have precise information about the channel. Though most real communication channels are not accurately represented by the binary symmetric channel (BSC), shown in fig.(3.2), it has been studied extensively[31,81]. For the binary symmetric channel, the probability is q that the same symbol as transmitted will be received and that p is the probability of receiving an erroneous symbol. It is assumed that $q > p$ and that each symbol is

independent of all others (the channel is memoryless). This channel model includes the modulator, channel, and the demodulator of the system in fig.(3.1).

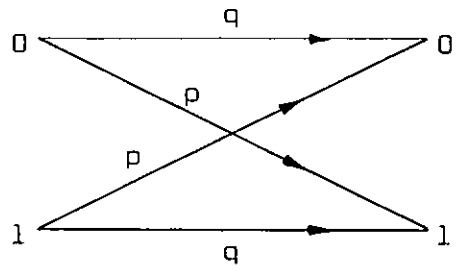


fig.(3.2) The binary symmetric channel

The transmission errors induced on the binary symmetric channel are referred to as random errors. Unfortunately, very few real channels are like the binary symmetric channel. There is usually serious dependence of errors in successive transmitted symbols. The noise disturbance - a strike of lightning or a man-made electrical disturbance - frequently affects several adjacent symbols. Defects on magnetic recording devices also usually affect more than one symbol. Thus, errors occur in bursts.

When each element in the output sequence depends statistically both on the corresponding input and on past inputs and outputs, the channel is a channel with memory. A burst-error channel is such a channel. They have the special property that errors tend to group together,where the error groups being separated by larger error-free groups. A Markov chain with two states can be used as a model for generating bursts[32]. Assuming that the i^{th}

error generated is e_i where

$$e_i = 1 \quad \text{for generating an error} \qquad (3.5\text{-a})$$

$$e_i = 0 \quad \text{for not generating an error} \qquad (3.5\text{-b})$$

The two states are 'A' for generating no error (e_i=0), and state 'B' for generating either e_i=0 or e_i=1, as in fig.(3.3), the directed branches indicate transitions from

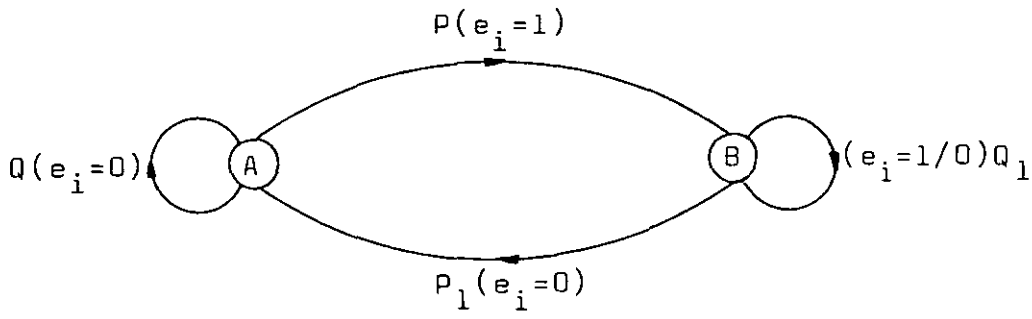


fig.(3.3) Transition diagram for Markov chain

one state to another, with the symbol on each branch representing the probability of that transition, and when mentioned the output is given in parenthesis.

After producing the noise digit e_i, the Markov chain makes a transition to prepare for e_{i+1}. To simulate burst errors, the states 'A' and 'B' must tend to persist i.e. the transition probabilities P (for going from A→B) and P_1 (for going from B→A) are small, and the probabilities Q (for remaining in 'A') and Q_1 (for remainig in 'B') are large where

$$Q = 1 - P \qquad (3.6\text{-a})$$

and $$Q_1 = 1 - P_1 \qquad (3.6\text{-b})$$

The Markov chain model is suitable for investigating binary burst-error channels, where e_i may take either '0' or '1', as in Eq.(3.5), but this model is not suitable for channels where noise information is required i.e. in simulating channels for the study of soft-descision decoders, where information about the actual amplitude value of the noise sample w_i is needed, where the amplitude of w_i can take any value

$$-\infty \leqslant w_i \leqslant +\infty \qquad (3.7)$$

To overcome this problem it is desirable to define a model that can supply the required noise information. To differentiate between the two models the latter will be called an analogue-burst-noise channel model.

3.2- Analogue-burst-noise channel model

In practice, assuming that the noise effect on the channel is only additive noise, two noise types must be considered:

a) Noise of relatively low power, which is added to the signal continuously during the whole transmission period. Its presence is responsible for producing few scattered errors. These errors can be considered random errors. Clearly, this noise is not the main source of errors in a bursty channel and will be called the background noise.

b) Noise of high power, and a relatively short duration (at least longer than one element duration period τ). This burst noise is the main source of errors in the channel and its presence tends to be separated by longer periods of its absence.

Although the background noise is present during the whole transmission time, it is insignificant compared to the burst noise and so can be neglected when an error burst is present. One can thus think of noise as being switched from low power noise to high power noise and vice versa.

A model of the analogue-burst-noise channel based on the above facts is shown in fig.(3.4). This chain will provide information about noise amplitude, where noise amplitude is as defined in Eq.(3.7). Since we are interested in the channel coding and channel decoding performance, then it is more convenient to consider the discrete signals at the input and the output of the channel, so that Eq.(3.1) can be rewritten as

$$y_i = x_i + w_i \qquad (3.8)$$

where y_i, x_i, w_i are samples of the i^{th} received, transmitted, and noise elements respectively.

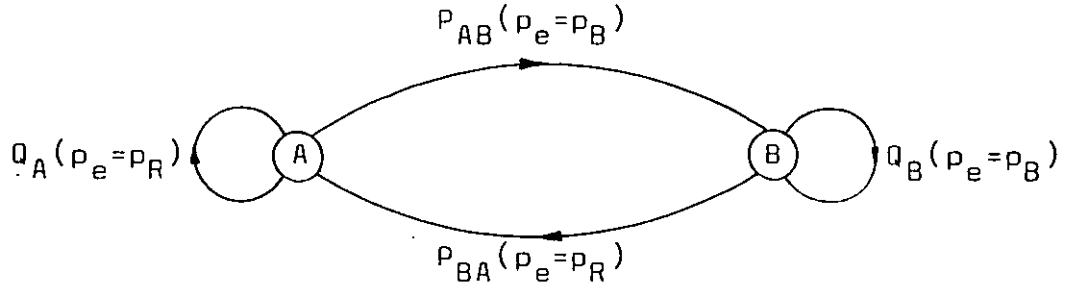


fig.(3.4) Transition diagram for analogue-burst-noise chain

Two states are used for generating noise, state'A' generates the background noise, where the probability of generating error is p_R (for random), and state 'B' for generating burst noise, where the probability of generating error is p_B (for burst). The probabilities Q_A (for remaining in 'A'), and Q_B (for remaining in'B') are large compared to the transition probabilities P_{AB} (for moving from 'A' to 'B'), and P_{BA} (for moving from 'B' to 'A'). Eqs.(3.6) can be rewritten

$$Q_A = 1 - P_{AB} \qquad (3.9\text{-a})$$

$$Q_B = 1 - P_{BA} \qquad (3.9\text{-b})$$

The noise sample w_i is a statically independent random variable, which has one of two probability densities depending on the state that generates w_i.

Assuming the system is in state 'A', although the noise power is low, Eq.(3.7) still applies and errors may be generated. The probability of generating errors p_R, varies according to the probability density function

of the simulated noise, and its power. When the noise power jumps up to a high value, the system will move to state 'B'. Again the probability of generating errors p_B, is dependent on the probability density function and the power of the noise. In this state more errors are generated in a form of a burst. At some stage the cause of the high power noise will cease to exist, noise will then jump down to the previous value, and the system will move back to state 'A', and so on. To define the probabilities of the transitions, one should recall that the system moves from 'A' to 'B' and from 'B' to 'A' as a result of the change in the noise power from low to high value and high to low value respectively. Consider the case of moving from 'A' to 'B'; this case will arise when the noise makes a jump from low power to high power, thus the system has to move from 'A' to 'B', generating one noise sample w_i as it moves. Since the system starts to move after the noise power change, then the probability of generating errors in this transition state is the same as the probability of generating errors in the high power noise state ('B'), namely, p_B. Conversly, the probability of generating errors in the transition from 'B' to 'A' is p_R.

3.3-Properties of analogue-burst-noise channel model

In order to compare the analogue-burst-noise channel model with any other model we must list some of its major properties.

1-Each noise sample w_i is an algebraic value, it

must satisfy Eq.(3.7)

$$-\infty \leqslant w_i \leqslant +\infty \qquad (3.7)$$

independent of the state in which the system is i.e. 'A', 'B', moving from 'A' to 'B', or moving from 'B' to 'A'.

2-Assuming the channel noise is only additive noise, the noise addition is algebraic, and Eq.(3.8)applies

$$y_i = x_i + w_i \qquad (3.8)$$

Given that the modulator of fig.(3.1) will output the sample voltage +V for the binary '1' and the sample voltage -V for the binary '0', then

$$x_i = \pm V \qquad (3.10)$$

then the sample of the received waveform y_i will take value in the range

$$+\infty \leqslant y_i \leqslant -\infty \qquad (3.11)$$

where the addition in Eq.(3.8) is an algebraic addition.

3-Few errors may be generated in the no burst state, (state 'A' and moving from 'B' to 'A'). But these errors are scattered, and are considered random errors.

4-An analogue-burst starts when the noise power jumps up to a high value. The probability of generating errors at the first burst element (the system is moving

from 'A' to 'B') is p_B. Although p_B is large, yet it is less than one, thus, an analogue-burst does not necessarily start with an error, although that has the probability of p_B which is relatively high.

5-The end of the analogue burst is when the noise power jumps down to a low value. The probability of generating errors at the last burst element (the system is still in 'B') is p_B. Which is large, but less than one. So an analogue-burst does not necessarily end with an error, although this has a relatively high probability.

6-The probability distributions of states 'A' and 'B', can be similiar or different depending on the simulated channel.

7-The high noise power of state 'B', can be either constant or variable during a transmission period.

8-The probability distribution of state 'B', can be in theory variable during a burst generation, or during a transmission time.

3.4-Transforming analogue-burst-noise model to an equivalent binary model

The Markov chain representing a binary-burst channel model, shown in fig.(3.3), can be used to study the performance of any binary burst-error-correcting codes. Since binary-error-correcting codes, and the binary correction technique will be used to correct errors, in decoding methods, that make use of the available noise information, then the analogue-burst-noise

model should be transformed to an equivalent binary-burst model to facilitate the application of binary correction rules.

Before considering the transformation from the analogue-burst-noise model to the binary-burst model, it will be beneficial to consider the following facts.

1-The analogue-burst-noise model represents the generation of additive noise, while the binary-burst model represents the generation of errors.

2-Any noise sample w_i generated by the analogue-burst-noise model, may or may not be an error. Let Th be the threshold value on which the demodulator of fig.(3.1), decides from the received sample y_i, whether the transmitted sample x_i was '0' or '1', according to the rules

$$\text{if} \quad y_i < Th \quad \text{then } x_i = 0 \qquad (3.12\text{-a})$$

or

$$\text{if} \quad y_i > Th \quad \text{then } x_i = 1 \qquad (3.12\text{-b})$$

The noise sample w_i will correspond to an error if two conditions are met

$$w_i \equiv e_i = 1 \qquad (3.13\text{-a})$$

$$\text{if} \quad |w_i| > Th \qquad (3.13\text{-b})$$

$$\text{and} \quad \text{sign } w_i \neq \text{sign } x_i \qquad (3.13\text{-c})$$

3-Although theoretically, noise generated by the analogue-burst-noise model is of unlimited value as in Eq.(3.7), it is practically, limited by the receiver to the voltage levels representing '0' and '1'. Let the voltage represent '0' by -V and the voltage represent '1' be +V then Eq.(3.7) can be rewritten

$$-V \leqslant w_i \leqslant +V \qquad (3.14)$$

The term 'analogue model' will be used as an equivalent to the term 'analogue-burst-noise channel model', while the term 'binary model' will be used to represent the 'binary-burst channel model', during the description of the translation process, which is as follows:

STATE 'A' in the analogue model can be considered as two separate cases according to the result of the detection process in the demodulation of fig.(3.1) which uses Eqs.(3.12).

Case 1, state 'A' in the analogue model does not generate an error. This event has the probability $1-p_R$ and so it can be transferred directly to state 'A' in the binary model.

Case 2, state 'A' in the analogue model generates errors, with probability p_R. Assuming that the present noise sample w_i is an error, then state 'A' in the analogue model represents the transition form 'A' to 'B' in the binary model. The next noise sample w_{i+1} may be one

of two possibilities, either w_{i+1} is not an error, consequently state 'A' in the analogue model is represented by the transition from 'B' to 'A' in the binary model, or w_{i+1} is an error, then state 'A' in the analogue model is represented by state 'B' in the binary model. In the later case, 'A' in the analogue model becomes 'B' in the binary model as long as the next consecutive noise samples are errors . As soon as the next noise sample is not an error, state 'A' in the analogue model becomes a transition from 'B' to 'A' in the binary model.

Transition from 'A' to 'B', three cases can be considered depending on the previous noise samples. If w_{i-1} was not an error, then w_i can be one of two values, either w_i is not an error where the probability is $1-p_B$, in which case the transition from 'A' to 'B' in the analogue model is state 'A' in the binary model. Or w_i is an error which has the probability of p_B, thus the transition from 'A' to 'B' in the analogue model is transferred to the transition from 'A' to 'B' in the binary model. On the other hand if the previous noise sample w_{i-1} was an error the transition from 'A' to 'B' in the analogue is state 'B' in the binary model.

STATE 'B' This is best described by considering the noise samples generated in state 'B' as three parts; start, middle, and end.

Transforming the few first samples is dependent on the previous samples. Let the first noise sample generated in state 'B' be w_i; if the previous sample w_{i-1} was an

error, then state 'B' in the analogue model becomes state 'B' in the binary model. Otherwise if w_{i-1} was not an error and w_i is an error, then state 'B' in the analogue model is represented by the transition from 'A' to 'B' in the binary model, while at the next sample w_{i+1}, state 'B' in the analogue model becomes state 'B' in the binary model. If w_{i-1} and w_i are not errors, then state 'B' in the analogue model becomes stat 'A' in the binary model. As soon as a noise sample is an error, state 'B' in the analogue model is transformed to the transition from 'A' to 'B' in the binary model, and at the next noise sample state 'B' in the analogue model becomes state 'B' in the binary model.

In the middle part, state 'B' in the analogue model has been transformed to state 'B' in the binary model, it stays there for all the middle part.

Transformation of the end part is dependent on the future noise samples. Let the present noise sample w_i be the last error in the burst, then state 'B' in the analogue channel becomes state 'B' in the binary model. Since the next noise sample w_{i+1} is not an error, then state 'B' in the analogue model becomes transition from 'B' to 'A' in the binary model, while for the next sample state 'B' becomes state 'A'.

Transition from 'B' to 'A' If the noise sample w_i is an error, the transition from 'B' to 'A' in the analogue model is represented by state 'B' in the binary model, while if the previous noise sample w_{i-1} was an

error, and w_i is not an error, then the transition from 'B' to 'A' in the analogue model becomes a transition from 'B' to 'A' in the binary model. But if w_{i-2} and w_{i-1} and w_i were no error samples, then the transition from 'B' to 'A' in the analogue model becomes state 'A' in the binary model.

The above discribed transformations from the analogue-burst-noise channel model to the binary-burst channel model are summarised in table.(3.1).

3.5-The transforming probabilities of the analogue-burst-noise model

The description of the transform from the analogue-burst-noise channel model to the binary-burst channel in section (3.4) and in table (3.1) does not give a proper idea of how many transformations take place from one state to another. To determine the transformation from any state in the analogue-burst-noise channel model to the corresponding state in the binary-burst channel model, it is essential to look at the probabilities for each individual case.

Before calculating the probabilities, let us define the terms to be used in the calculations. Let $P_{(statel,state2)}$ be the probability of transforming state 1 in the analogue-burst-noise channel model to state 2 in the binary-burst channel model, furthermore let A, represent state 'A', AB the transition from 'A' to 'B', B for state 'B', and BA represent the transition from 'B' to 'A'.

analogue model state	error samples			binary model state		probability of error
	present	next		previous	present	
	e_i	e_{i+1}	e_{i+2}	transform	transform	
A	0			A	A	p_R
	1			A	A to B	p_R
	0			A to B	B to A	p_R
	1			A to B	B	p_R
	0			B	B to A	p_R
	1			B	B	p_R
	0			B to A	A	p_R
	1			B to A	A to B	p_R
A to B	0			A	A	p_B
	1			A	A to B	p_B
				A to B	B	p_B
				B	B	p_B
	0			B to A	A	p_B
	1			B to A	A to B	p_B
B	0			A	A	p_B
	1			A	A to B	p_B
				A to B	B	p_B
			1	B	B	p_B
	0	1	0	B	B	p_B
	0	0	0	B	B to A	p_B
	0	0	0	B to A	A	p_B
B to A	0			A	A	p_R
	1			A	A to B	p_R
	0			A to B	B to A	p_R
	1			A to B	B	p_R
	0			B	B to A	p_R
	1			B	B	p_R
	0			B to A	A	p_R
	1			B to A	A to B	p_R

$e_i = 1$ error

$e_i = 0$ no error

blank do not care

Table (3.1) Transferring analogue model to binary model

Considering state 'A', according to table (3.1) there are four possible places where state 'A' can be transformed, so there are four probabilities corresponding to these four transformations.

The transform from state 'A' in the analogue model to the transition from 'A' to 'B' in the binary model, takes place each time an error is generated, following a no error generation. Since the probability of generating an error in state 'A' is p_R, and the probability of generating no error is $1-p_R$, then

$$P_{(A,AB)} = p_R(1 - p_R) \qquad (3.15)$$

Transforming 'A' in the analogue model to state 'B' in the binary model requires state 'A' to generate an error next to an error generation, or in other words to generate an error following a transform from state 'A' in the analogue model to the transition from 'A' to 'B' or to state 'B' in the binary model. Thus

$$P_{(A,B)} = p_R^2 \qquad (3.16)$$

Assuming state 'A' in the analogue model has been transformed to state 'B' in the binary model as a result of the previous transform, the current transform can be to the transition from 'B' to 'A', if state 'A' in the analogue model did not generate an error i.e. state 'A' in the analogue model will transform to the transition from 'B' to 'A' in the binary model if state 'A' generates

no error after generation of an error. The probability of this transform is

$$P_{(A,BA)} = p_R(1 - p_R) \tag{3.17}$$

The fourth and last probability to be calculated is the probability of the transform from state 'A' in the analogue model to state 'B' in the binary model. This transform takes place if the previous transform was to the transition from 'B' to 'A', or to state 'A' i.e. if two consecutive no errors are generated when the system is in state 'A' in the analogue model, thus the probability is

$$P_{(A,A)} = (1 - p_R)^2 \tag{3.18}$$

To consider the transform probabilities for state 'B', the length of the current burst ℓ should be taken into account. Considering the transform from state 'B' in the analogue-burst-noise channel model to the transition from 'A' to 'B' in the binary-burst channel model, this transform will take place at the first error generation when the system is in state 'B' in the analogue model. The probability of generating at least one error during all ℓ samples, given that the probability of generating an error is p_B, can be calculated as follows. The probability of generating no errors during the whole burst is

$$P_{\text{no errors}} = (1 - p_B)^{\ell} \tag{3.19}$$

since $$\sum_i p_i = 1 \qquad (3.20)$$

then the probability of generating one error or more in the ℓ samples is

$$p_{error} = 1 - (1 - p_B)^{\ell} \qquad (3.21)$$

during the burst generation ℓ transforms take place, the transform from state 'B' in the analogue model to the transition from 'A' to 'B' occurs only once, so its occurence is $\frac{1}{\ell}$ of the whole burst, and since the probability of its occurence is p_{error}, then the probability of the transfer is

$$p_{(B,AB)} = \frac{1}{\ell} \{1 - (1 - p_B)^{\ell}\} \qquad (3.22)$$

The next transform to be considered, is the transform from state 'B' in the analogue channel model to the transition from 'B' to 'A' in the binary model. This transform will occur, if the system in state 'B' in the analogue model has generated at least one error (the system is in state 'B' in the binary model), and the error generated does not confine itself to the ℓ_{th} position of the burst. In fact the transform to the transition from 'B' to 'A' in the binary system will take place, at the next sample to the last error generation, wherever that error is. The ℓ_{th} sample of the burst can be used as an indication of the transform. So if the ℓ_{th}

sample generates an error, then no transform has occured, while if the ℓ_{th} sample generates no error, then a transform has occured, but not necessarily at the ℓ_{th} position. The probability of an no error generated at the ℓ_{th} position is

$$P_{\text{no error}} = 1 - P_B \tag{3.23}$$

The probability of the system being tansformed to state 'B' in the binary model is given in Eq.(3.21). Thus the transform probability is the probability of generating no errors in the ℓ_{th} position given that the system is in state 'B' in the binary model, and since it takes place only once during the burst, then

$$P_{(B,BA)} = \frac{(1-P_B)}{\ell} \{1-(1-P_B)^{\ell}\} \tag{3.24}$$

The third transform is the transform state 'B' in the analogue channel model to state 'A' in the binary channel model. Such a transform may occur at the beginning and the end of the burst. At the beginning of a burst, the system will move from state 'A' to state 'B' in the analogue channel model, once the system is in state 'B' in the analogue model, it will be transformed to state 'A' in the binary model as long as no error is generated. As soon as an error is generated, state 'B' in the binary model is transformed to the transition from 'A' to 'B' and then to state 'B'. On the other hand, near the end

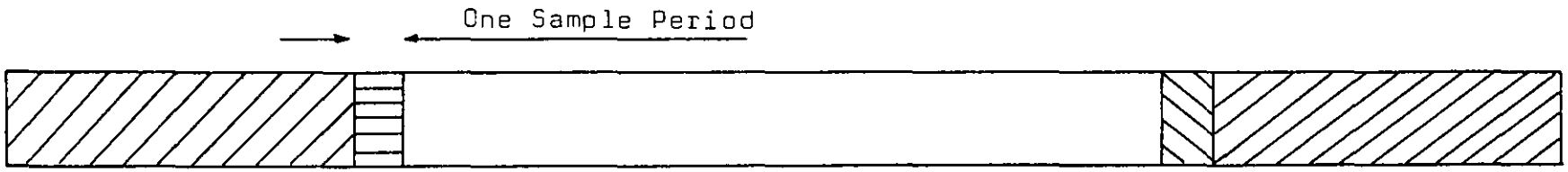
of the burst, state 'B' in the analogue channel is being transformed to state 'B' in the binary model, as long as, the last error in the burst is not generated. After the generation of the last error, state 'B' in the analogue is transformed to the transition from 'B' to 'A', then to state 'A' in the binary model. To reduce the calculation complexity the actual transform from state 'B' in the analogue to state 'B' in the binary model is shifted, so that the transform from 'B' in the analogue channel to state 'A' in the binary channel will occur at the beginning only. Fig.(3.5) shows a burst generated when the system is in state 'B' in the analogue channel model before and after shifting. The probability of transforming state 'B' in the analogue model to state 'A' in the binary model during the 1st sample is the probability of generating no errors, then

$$p_{\text{no error}} = 1 - p_B \tag{3.25}$$

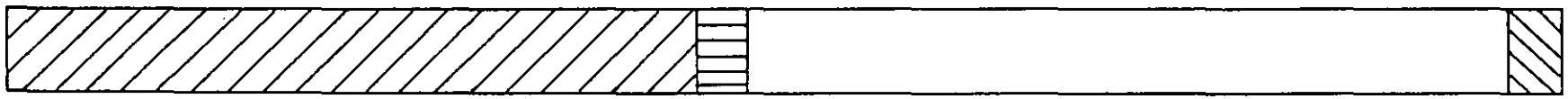
since this transform takes place once during ℓ samples, then

$$p_{1 \text{ no error}} = \frac{1-p_B}{\ell} \tag{3.26}$$

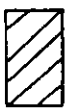
Similarly the probability of generating no errors for the i^{th} sample is the probability of Eq.(3.26) given that all previous samples generate no errors, Hence



a) The Generated Burst



b) The Shifted Burst



Transfer to state 'A'

Transfer to the transition from 'A' to 'B'

Transfer to state 'B'

Transfer to the transition from 'B' to 'A'

Fig.3.5 Shifting a burst

$$P_{i \text{ no error}} = \frac{1}{\ell} (1 - p_B)^i \qquad (3.27)$$

the total probability of transforming state 'B' in the analogue model to state 'A' in the binary model is the sum of all these ℓ probabilities

$$P_{(B,A)} = \frac{1}{\ell} \sum_{i=1}^{\ell} (1-p_B)^i \qquad (3.28)$$

The fourth and last probability, is the probability of transforming state 'B' in the analogue channel model to state 'B' in the binary channel model. According to Eq.(3.20)

$$P_{(B,B)} + P_{(B,A)} + P_{(B,AB)} + P_{(B,BA)} = 1 \qquad (3.29)$$

$$P_{(B,B)} = 1 - \{ P_{(B,A)} + P_{(B,AB)} + P_{(B,BA)} \} \qquad (3.30)$$

Substituting Eqs.(3.28),(3.22),(3.24) in Eq.(3.30) gives

$$P_{(B,B)} = 1 - \left\{ \{ \frac{1}{\ell} \sum_{i=1}^{\ell} (1-p_B)^i \} + \frac{1}{\ell} \{ 1-(1-p_B)^{\ell} \} + \frac{1-p_B}{\ell} \{ 1-(1-p_B)^{\ell} \} \right\} \qquad (3.31)$$

or

$$P_{(B,B)} = 1 - \frac{1}{\ell} \left\{ \{ \sum_{i=1}^{\ell} (1-p_B)^i \} + (2-p_B) \{ 1-(1-p_B)^{\ell} \} \right\} \qquad (3.32)$$

A real transmission channel has the probability values of $p_R = 10^{-2} - 10^{-4}$, and $p_B = 0.1 - 0.5$.

Considering an average channel which has $p_R=10^{-3}$, and $p_B=0.3$, and let the average burst length $\ell=128$ the transform probability from state 'A' in the analogue channel model to state 'A' in the binary channel from Eq.(3.18) is

$$P_{(A,A)} = (1 - 10^{-3})^2 \tag{3.33}$$

$$P_{(A,A)} = 0.998 \tag{3.34}$$

Similarily the transform probability from state 'B' in the analogue channel model to state 'B' in the binary channel model is given in Eq.(3.32)

$$P_{(B,B)} = 1-\frac{1}{128}\left[\{\sum_{i=1}^{128}(1-0.3)^i\}+(2-0.3)\{1-(1-0.3)^{128}\}\right] \tag{3.35}$$

$$P_{(B,B)} = 0.9789 \tag{3.36}$$

Eqs.(3.34) and (3.36) show that 99.8% of the noise samples generated in state 'A' correspond to state 'A' in both channel model, and nearly 98% of the noise samples generated in state 'B' correspomd to state 'B' in both channels. Consequently, the two channel models are interchangeable for 98% of the time, so that the techniques used for signal processing when the binary channel is used can be used for the analogue channel and vice versa, since the transformation error incurred is very small.

The probability of the system being present in state 'A' in both systems is the probability of transform from state 'A' to state 'A' and from state 'A' to the transition from 'B' to 'A', because at the end of the two transforms the system is in state 'A'. So

$$P_A = P_{(A,A)} + P_{(A,BA)} \tag{3.37}$$

Substituting the values of the average channel in Eq.(3.37) gives

$$P_A = 0.998 + 0.00099 = 0.999 \tag{3.38}$$

similarily, the probability of the system being in state 'B' in both systems is

$$P_B = P_{(B,B)} + P_{(B,AB)} \tag{3.39}$$

$$P_B = 0.9789 + 0.0078 = 0.9867 \tag{3.40}$$

The effect of the probabilities of error variation can be seen as follows. It is clear from Eqs.(3.37),(3.18), and (3.17) that state 'A' probabilities is dependent on the background noise probability only. Thus p_R is the only factor that will affect the transform probabilities of state 'A'. Eq.(3.18) shows that $P_{(A,A)}$ will move closer to 1 as p_R decreases, while Eq.(3.17) shows that the value of $P_{(A,BA)}$ decreases as p_R decreases, but since

$P_{(A,A)}$ is the dominant factor and its increase is larger than the decrease of $P_{(A,BA)}$, then P_A moves closer to 1 as p_R decreases and vice versa. These results are expected because as p_R is decreased, fewer random errors are generated, which lead to more transforms from state 'A' in the analogue model to state 'A' in the binary model, hence $P_{(A,A)}$ is larger. On the other hand, since more transforms occur from 'A' to 'A' then fewer transforms occur from'A'to'B' and consequently fewer transforms to the transition from 'B' to 'A', hence $P_{(A,BA)}$ becomes smaller.

State 'B' probabilities are dependent on the burst noise probability only, for a given burst length,according to Eqs.(3.39),(3.32), and (3.22). It can be seen from Eq.(3.32) that $P_{(B,B)}$ increases, as the probability of error p_B is increased, while for a moderate length ℓ $P_{(A,AB)}$ is nearly constant. Thus P_B increases with the increase of p_B, and vice versa. Again this result is expected because as p_B is increased, fewer no errors will be present at the beginning and the end of a burst, hence fewer transforms to 'A'.

3.6-Channel simulation

During the course of the tests, two types of codes were used. Codes for correcting random-errors, and codes for correcting burst-errors. Consequently, there was a need to simulate two types of channels that are suitable for the transmission of each type of codes, namely a

random error channel and a burst-error channel (bursty channel).

3.6.1 The bursty channel

The obvious way to get the data for a bursty channel is to set three random variables, one representing the start point of the burst, the second representing the burst length, while the third represents the errors position in the burst. This will be a simulated channel which is not real, because in reality bursts have a special properties where short bursts tend to occur more frequently than the longer ones, while very long burst are rare. A better way is to smiulate a real transmission channel. It was stated at the beginning of this chapter that a channel can be any medium where data is transmitted or stored. And since our interest is in the decoder itself and not in the channel, therefore the choice of a bursty channel for the simulation is not critical, and can be either a transmission line, communication link, or a storage media.

The data chosen for the simulation is taken from a consultancy work done in Loughborough University of Technology for EMI Central Research Laboratories[93], where tests were carried out for recording and reading digital data on digital audio tapes. Data obtained from the report concerns the burst length and frequency in a one hour recording period at a data rate of 26Kbps for different tape types.

3.6.1.1 The bursty channel simulation

Simulation of the bursty channel is carried out by

calculating values for the three random variables mentioned in section 3.6.1. The first represents the burst distribution during transmission, i.e. it gives the start point of each burst. It was assumed that bursts are randomly distributed during the transmission period. This is in accordance with section 3.2 where no special distribution is assumed for burst generation, and in the absence of further data concerning bursts distribution appears a reasonable assumption to make. The second variable determines the duration of each burst. The data for this variable is obtained from reference (93), although that reference gives a very accurate method for simulating the bursty channel it does not lend itself to computer simulation due to time constraints, and so a simple curve fitting technique will be used for our study. A least-squares approximation using Chebyshev polynomial[30] is used to get the equation of the bursts length, which in turn is used to generate the random length of the simulated bursts. The third variable is the actual noise value added to each of the transmitted digits, and it sign, i.e. the error position in the burst. This variable is usually taken in a binary burst channel model as a '0' or '1' generator, but for our case since we are interested in simulating an analogue burst channel model, the noise added to the transmitted signal must have a algebraic value, and not '0' and '1' only. Although the additive noise introduced by many practical channels does not approximate to Gaussian noise, it is well known that a digital signal having a better

tolerance to additive white Gaussian noise than another signal, will normally also have a better tolerance to the additive noise obtained in practice[16,17]. Therefore an additive white Gaussian noise is used as the random variable that decides the error positions in each burst.

3.6.1.2 Implementation of the simulation

The data from reference (93) shows that 230 burst occured on average during a one hour recording, and that the total number of transmitted digits is 93,600,000 per hour. To start with, 230 random numbers are generated from the curve fitting equation, these numbers represent the length of the 230 burst to be used during the simulated transmission. Then another 230 numbers ranging between 1 and 93,600,000 are randomly generated, these numbers represent the starting point of each burst. Since the computer will simulate the transmission process in a serial mode, therefore these numbers are sorted in an ascending order, so that each burst is in the correct order timewise. The next step was to consider the adjacent burst, to check for a common area between the two burst. When a common area is present between two or more bursts, the following rules are used to deal with that area:

1-If the next burst happens to exist inside the present burst, the next burst is discarded.

2-If two bursts or more happen to overlap, all the bursts are considered as one burst that starts at the

beginning of the first overlapped burst, and ends at the end of the last overlapped one.

3-If any burst extends to the point beyond the end of transmission, it is forced to end at the end of the transmission, and the rest is discarded.

The noise samples are then added to the corresponding digits of the burst, which are then handed to the receiver for decoding. Since we are interested in simulating the analogue burst noise channel described in section 3.2, then two types of noise will be added to transmitted signal in the channel. Firstly, noise of low power, which will be called background noise, this noise is present all the time, so it is added to the signal during all the transmission period, the SNR value at the channel output is high, therefore only few random errors will be generated. Secondly, the burst noise, which has high power, and is added to transmitted signal during the burst periods only, the SNR value at the channel output is low, thus this is responsible for generating most of the errors introduced in the transmission system.

3.6.1.3 Some practical considerations

Interlaceing techniques are expected to be used on any bursty channel to improve the code performance, especially with short length block codes. In order to achieve a practical system simulation, codes were interlaced to a degree of λ. Interlaceing techniques will be discussed in detail in chapter four.

For the sake of running the simulation programme

faster, and using less memory space, some practical aspects were considered during the program writing. These changes do not affect the programme results, but they increase its complexity. These changes are:

1-Since each code tested was interlaced to interlaceing degree λ. A minimum memory space of λn is used to store the transmitted and the received data.

2-To cut the big amount of calculation in the encoding and decoding processes, it was assumed that the background noise generate errors that can be corrected by the decoder. Therefore no errors occur in the blocks not affected by a burst or more. Thus the encoding and decoding process in the simulation takes place only for the corrupted λn blocks. Furthermore, the program monitors each transmitted codeword block of the λ blocks, and excludes any unaffected block or blocks from the decoding process.

3-Because of the policy used to deal only with the corrupted blocks, then the program has to fit each burst in its exact place within the code blocks. In some cases where the burst is not comfined to one block, the noise values are added to the transmitted digits from the place where the burst starts up to the last (λn^{th}) digit of the interlaced block, after which the decoding process is started. Once the decoding is finished the remainder of the burst is added to the digits of the next interlaced block, and the process is repeated as long as the burst is not finished. Once all the burst is added to the transmitted

signal and the decoding process is finished, the next burst is considered

The block diagram of fig.(3.6) shows the flow chart for the analogue-burst- noise channel test system as used in the simulation, while the actual simulation programme is included in appendix D.

3.6.2 The random-error channel simulation

The channel simulation programme for random-error generation is a straightforward programme. Additive white Gaussian noise is used for the same reason mentioned in section 3.6.1.1. Since the noise power is nearly constant in these channels, then a white Gaussian noise is added to the transmitted signal during the transmission period. The SNR value at the channel output is constant, and the number of random errors is dependent on the noise power value. Simulation is acheived by the algebraic addition of a noise sample to each transmitted digit. The encoding and decoding process is carried out for all the transmitted blocks. The simulation programme is included in appendix D.

3.7-Two-Way Channels

The communication channel shown in fig.(3.1) is strictly a one-way channel, where signals are transmitted from one terminal (the source) to another terminal (the sink), through the channel, in one direction only. Very frequently communication systems employ two-way channels, where a terminal (source or sink) can transmit and receive signals from the other terminal (sink or source). Being

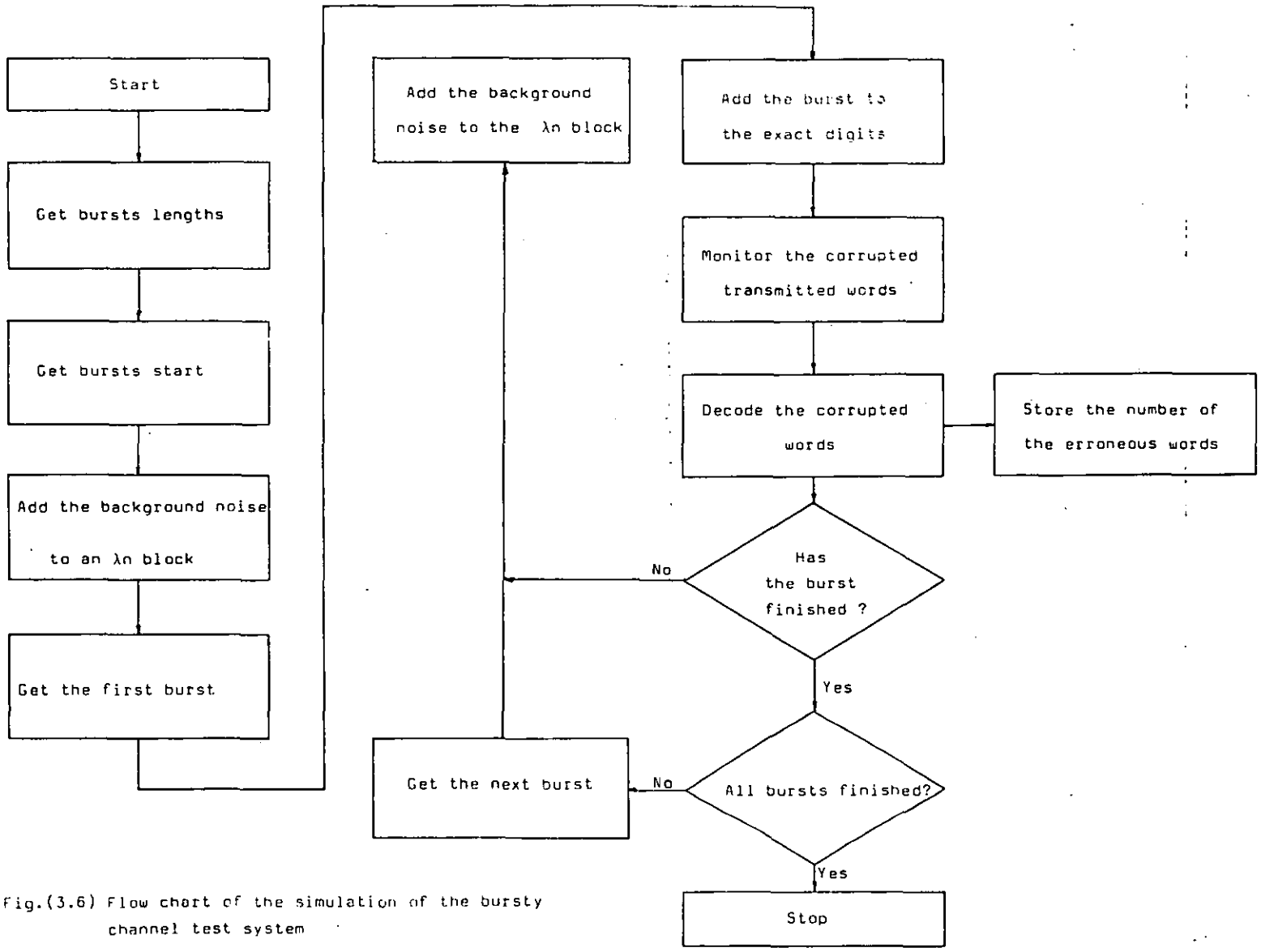


Fig.(3.6) Flow chart of the simulation of the bursty channel test system

error detection and retransmission. Error detection requires much simpler decoding equipment, although encoding for error detection is no more complex than for error-correcting codes. Also, error detection with retransmission is adaptive, i.e. redundancy digits are increased with errors, when no errors are detected, there are no redundancy digits transmitted. Therefore the transmission bit rate under certain circumstances may be lower. Thus this kind of system may perform better than the one-way channel system.

There is a limit to the efficiency of a system that uses simple error detection and retransmission alone. Short error-detecting codes cannot detect errors efficiently, while if extremely long codes are used, retransmission must be done too frequently. A combination of correction of most frequent error-patterns and detection coupled with retransmission for less frequent error-patterns is not subject to the limitations described above, and is often more efficient than either error correction or error detection and retransmission alone. Several systems have been build using the combined error correction and detection with retransmission facilities[53].

Deciding on the best system, that add less redundancy digits, at some error-rate value, is dependent on many factors, i.e. the number of parity-check digits of the detection and correction codes, the lenght of the block, and the channel error rate. A detailed study of the bit rate consideration in detection and correction codes for random-error channels, is included in appendix C.

CHATPER 4

TIME - SHARED

SOFT-DECISION DECODER FOR BURST-ERROR CHANNELS

4.1-Burst Noise

Burst noise problems can be divided into two fundamental types. Firstly, where no information is available to identify single unreliable digits. This type has been well studied and documented over the years, because this problem existed long before soft-decision techniques were available. However, if the error statistics are available, the channel under study can be modelled (as in section 3.1), and a suitable error-correction scheme can be introduced. The second type of problem occurs when noise information is available from the demodulator, thus the reliability identification for each received digit can be calculated. However the use of reliability information for decoding burst-errors, will create new problems at high noise power in that the high noise value may force the transmitted digit to change its value to a value higher than that transmitted. This will happen if the noise sample w_i satisfies two conditions

$$|w_i| \geq 2V \qquad (4.1\text{-a})$$

$$\text{sign } w_i \neq \text{sign } x_i \qquad (4.1\text{-b})$$

where x_i is the transmitted digit, that can be transmitted as +V or -V. The decoder will treat such samples as a correctly received digit, because the confidence number is high. To overcome this problem the received digits are erased during severe noise conditions.

The first type of problem occurs in certain communication media where data is handled digitally such as storage media. Examples are films, magnetic disks, magnetic memories, and sometimes magnetic tapes. Burst error can occur through scratches, defects, ageing, etc. The nature of the error mechanism is such that reliability information is very difficult, if not impossible to obtain. Typical approaches to solving this problem are techniques which correct long bursts of errors, or multiple short bursts, or interlacing codes, so multiple long bursts can be corrected. Burst-error-trapping techniques found by Tong[88] and Gallager[31] are very effective in correcting long bursts provided there is sufficient error-free guard space between bursts. The greatest problem in applying these techniques is getting an accurate data of the burst statistics of the transmission channels.

The use of coding in the second type of burst noise problem can be more effective, because of the use of noise information of the channel (section2.4.1). Often the channel is basically a Gaussian noise channel that is occasionally corrupted by large bursts of noise or interference. The transmission channel is specified by

four probabilities as in fig.(3.4), two transition probabilities P_{AB} and P_{BA} that give the remaining Markov chain probabilities from Eqs.(3.9). The other two probabilities are the background noise probability p_R, which cause the generation of random errors, and the burst noise probability p_B, which is responsible for the burst noise errors generation. A near-optimum strategy in providing likelihood information when the bursts are present is to simply blank the digits effected by the burst noise so that erasures are produced[11]. This strategy will improve the decoder performance, since the erroneous confidence numbers are not fed to the decoder, thus the decoder is not assuming any of the burst digits are correct as opposed to the ordinary soft-decision decoder. However, it is assumed in such decoders that when the burst noise is present, it is considered to have such a large value that it can easily be detected and blanked by the modulator, which will result in a burst of erasures. And that only complete digit or digits are blanked. The performance in an actual system will depart somewhat from the results calculated by computer simulation, because of difficulties in implementing perfect blankers or because the blanker may be approximated by a clipper or a limiter. In addition, the burst noise may not have a sufficient large value to activate the blanker. Because of the degradation in the practical system that uses blankers, compared to the computer simulated system, it was decided not to use blankers in the introduced algorithms keeping in mind that a similar

system using blankers will perform better at low SNR values, The reason for this choice is to get a more relistic idea from comparing the performance of the different computer simulated systems.

4.2-Interlaceing (Interleaving) Techniques

To correct all bursts of errors added to the transmitted signal during transmission through a bursty channel, an error-correcting code which is capable of correcting bursts of length equal at least to the maximum burst length generated by the channel should be used. This will restrict the choice of codes for such channels to long codes. One potential solution involves utilizing a suitable interlacer/deinterlacer pair. Using this approach, codewords from the encoder output are fed into an interlacer prior to transmission, at the receiver end the received word is fed to a deinterlacer prior to decoding. The function of the interlaceing is to distribute error more uniformly at the decoder input, so that codes which have relatively short burst-length correction capability can be used. A block diagram of a system that uses interlacer/deinterlacer is shown in fig.(4.1). Note that if each received digit is quantized to J bits

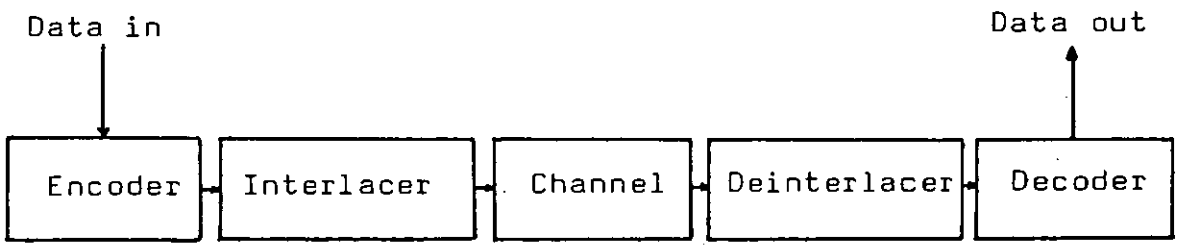


Fig.(4.1) Block diagram of external Interlacer/Deinterlacer

in the demodulator, then the deinterlacer requires a factor of J more memory than the hard decision memory requirements.

By definition, an interlacer is a device that rearranges the ordering of a sequence of symbols in a deterministic manner. On the other hand a deinterlacer is a device that functions exactly in the reverse order of the interlacer, i.e. it applies the inverse ordering to restore the symbols to thier original sequence. These two devices can assume any configurations, as long as a law and its inverse are applied. However interlacers can be divided into two principal classifications. First, the periodic type, which is perferred in many applications because of its simplicity and its low cost. The second, is the pseudorandom type, which offers more robustness than the periodic type. Hence, it may be preferred in certain applications where the burst characteristics of the transmission channel vary substantially.

The interlacer/deinterlacer shown in fig.(4.1) are applied externally to the encoder/decoder hardware, which is the general case. But this is not the simplest implementation for the cyclic codes using error-trapping decoders. The simplest implementation is to apply interlaceing and deinterlaceing internally to the decoder by the use of shift registers[47].

4.2.1 Periodic Interlacers

A periodic interlacer is an interlacer which has a periodic function of time law as the ordering law. Two

types of interlacer are commonly used, symbol interlacers, and convolutional interlacers.

4.2.1.1 Symbol Interlacers

A typical case of a symbol interlacer involves writing the coded digits in the rows of a matrix, which has n columns and λ rows which is called (λ,n) interlacer. Each codeword is written in a row, so in total λ codewords are written in the matrix. The ordering consist of reading these digits out of the matrix by columns prior to transmission. Such an interlacer is called a symbol interlacer of degree λ. At the receiving terminal, the deinterlacer performs the inverse operation, digits are written in columns and read out in rows. The most important characteristics of the symbol interlacers are:

1-Any burst of errors of length $\ell' \leqslant \lambda$ results in single errors at the deinterlacer output, each separated by at least n digits. Since the block code length is n, then for such a case one error may occur in each block. If the code used can correct a burst of ℓ errors in one block, then using this interlacer will enable the code to correct bursts of length $\ell' \leqslant \lambda\ell$. Hence the correction power is increased by λ.

2-Any burst of length ℓ' that statisfy the following equation

$$\ell' = f\lambda \qquad (4.2)$$

where for the moment assuming f>1, will result in bursts of no more than $\lceil f \rceil$ digits length, which are

separated by a space guard no less than $n-\lceil f \rceil$ digits, where $\lceil f \rceil$ represent the nearest largest integer to the value of f. Since it is assumed that the code can correct all bursts of ℓ or less in one block, then for correctable bursts f can be

$$0 \leqslant f \leqslant \ell \qquad (4.3)$$

3-A periodic sequence of K single errors spaced by λ digits, will result in a single burst of length K at the deinterlacer output, while if the errors are spaced by λ/J , they will result in J bursts. On the other hand, even if the errors are not single errors, they will result in a burst and some scattered errors in the interlaced λn digits, as long as there exist in these errors spaced by λ digits.

4-The memory requirement is λn digits of storage in both the interlacer and the deinterlacer provided that the received digits are not quantized. End to end delay can differ depending on the modulating and decoding strategy. If the transmission, and the decoding starts after the whole λn digits are stored then the maximum delay is

$$D_{max} = 2\lambda n \qquad (4.4)$$

On the other hand, if the transmission and the decoding starts once digits are stored in the whole first column, then the delay is minimum and given by

$$D_{min} = 2 \{ \lambda n - (n-1) \} \qquad (4.5)$$

The third characteristic demonstrates clearly that, if there is substantial variation in the burst noise characteristics of the transmission channel, this type of interlacer lacks robustness, and its ability to disperse bursts is degraded.

4.2.1.2-Convolutional Interlacers

This type of interlacer is referred to as a (λ,n) interlacer[28,75], and has properties similar to the (λ,n) symbol interlacer. A shift register version is shown in fig.(4.2). By definition

$$M = \frac{n}{\lambda} \qquad (4.6)$$

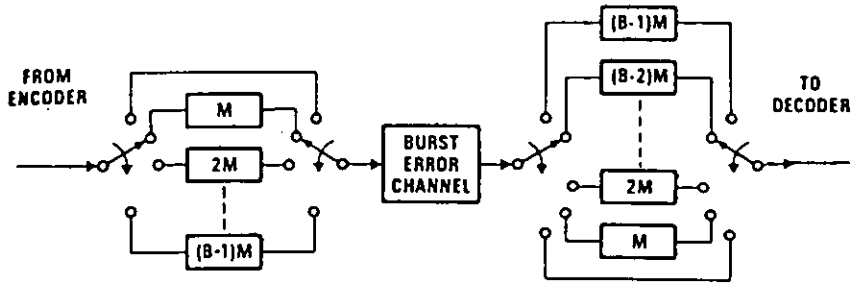


Fig.(4.2) Shift register implementation of a convolutional interlacer/deinterlacer

The interlacer functions in the following way. The encoder output digits are fed to the interlacer input which shifts them sequentially into λ registers. Each digit is fed to one of λ register in turn, as a new digit is shifted in,the oldest digit in the interlacer is shifted out to the channel. The shifting in and

shifting out is done synchronously. The length of the λ registers is increased by a factor of M. The deinterlacer functions in the reverse order, where the λ registers are decreased by a factor of M. Channel output is shifted in one of the registers and the oldest is shifted out at the same time. Clearly a synchronization between the interlacer and the deinterlacer should be present for proper deinterlaceing.

The most important characteristics of this type of interlacer are :

1-Any two digits that are separated by less than n digits at the interlacer input will be separated by at least λ digits. Let a burst consist of ℓ digits which are added to the transmitted digit in the channel, this burst will result at the output of the deinterlacer as ℓ single errors which are separated by at least n digits.

2-Any pattern of errors that is a periodic single errors spaced by n+1 digits results in a burst of length λ at the interlacer output.

3-The total end-to-end delay excluding the channel delay is

$$D = n\,(\lambda-1) \qquad (4.7)$$

while the memory requirement for each terminal is $n(\lambda-1)/2$ digits of storage.

Two points worth noticeing at this stage are, firstly, in this type of interlacer the memory and delay are about

half those of the equivalent symbol interlacer's requirements. Secondly this system lacks robustness in the same way as the symbol interlacer systems.

4.2.2 Pseudorandom Interlacer

This interlacer takes a block of L digits from the encoder output and reorders, or permutes them in a pseudorandom fashion. This can be implemented by writing the L digits into a (1xL) RAM, and then reading them out pseudorandomly. Any pseudorandom sequence can be used to permute the interlacer memory address. To generate the pseudorandom sequence, a pseudorandom number generator can be used, or alternatively the desired permutation can be stored in a ROM and the ROM output is used to address the interlacer[18,78]. The deinterlacer simply performs the inverse permutation. That is, at the deinterlacer the received digits are written into a (1xL) RAM using the same pseudorandom sequence used to read them at the interlacer. Then these digits are read out of the memory sequentially. The reading out of the interlacer and writing in the deinterlacer sequences should be synchronized to get back the same transmitted codeword at the deinterlacer output.

The most important characteristics of the pseudorandom interlacer are the following:

1-This technique provides a high degree of robustness to the variability of the burst noise channel parameters during transmission. Needless to say, such an interlacer is more costly and complex than either the symbol interlacer or the convolutional interlacer.

2-If the same permutation sequence is used for interlaceing each block of L digits, there exist certain error-patterns that can seriously degrade the interlacer performance. In systems where such patterns are likely to occur, permutation should be changed freqeuntly, during the transmission to avoid this problem.

3-The total end-to-end delay exclusive of the channel delay is

$$D = 2\,L \qquad (4.8)$$

And the memory requirement in both the interlacer and the deinterlacer is L digits of storage, excluding the memory control requirements.

4.2.3 Implementation of Interlacers

The two principal classes of interlacers described in sections 4.2.1 and 4.2.2 function using a different strategy. The periodic interlacers strategy attempts to produce maximally spaced errors while the pseudorandom interlacers strategy attempts to produce random errors at the decoder input. It has been widely assumed that the periodic approach will result in superior performance when the interlacer is exactly matched to the transmission channel parameters[18]. But it is much less robust to their changes than the pseudorandom interlacers.

Synchronization is a problem of various complexity according to the interlacer used. The symbol interlacer presents no additional problem when it is used with

block codes, where same block synchronization techniques can be used to synchronize the interlacer and the deinterlacer. The implementation of a convolutional interlacer involves a similar problem to the symbol interlacer, and in some applications a detector is added to detect the unsynchronized codewords[89] so that resynchronization can be achieved faster, or the erroneous codewords can be discarded. In the pseudorandom interlacer there is a fixed relationship between the ROM address counter, or the pseudorandom number generator state and the received digit counter. Once lock is obtained, the deinterlacer is set ready to receive the first digit of the L block digits by standard synchronization technieques.

Although the memory requirements for each type of interlacer are given in the appropriate sections, these are not the exact memory requirements of the system for the symbol and pseudorandom interlacers, while $n(\lambda-1)/2$ is sufficient at each encoder or decoder terminal for the convolutional interlacer. The reason is that for the symbol interlacer and the pseudorandom interlacer, the actual memory management is usually accomplished using two interlacers and two deinterlacers in a ping-pong configuration, where digits are written in one interlacer or deinterlacer while digits are read from the other. Once one interlacer or deinterlacer is full and the other is empty the functions of the two interlacers or deinterlacers are interchanged. So the actual memory requirements are twice those mentioned in sections 4.2.1.1 and 4.2.2.

4.3-The Time-Shared Decoders

There is no doubt that the recent developments in micro-technology, and the fall in the cost of storage devices has attracted too many designers to the microprocessor, due to its falling hardware cost, and its flexibility. In many communication fields as well as the error-correction, field, microprocessors are used widely, and many algorithms have been introduced that give superior improvement to non-microprocessor based systems, but at the cost of more complexity. Some algorithms need a long time to be executed, which forces their application to be limited by the microprocessor speed, i.e. whether the microprocessor can process the input data fast enough to be used as a real-time processor. In most applications it is desirable to use slow microprocessors, because of their cost and power comsumption requirements.

4.3.1 Microprocessors Speed Limitations

Microprocessors can be defined as a program-driven clocked sequential circuits. Their internal organization contains no special circuit or architectual features which do not exist in conventional integrated circuits. The main difference is that the rapid development in recent years has allowed more ciruits to be accommodated in less space. This has created the first major limitation, which is that whereas the capacity of the integrated circuit chips for logic components is very large, the number of pins that can be accommodated mechanically on a chip is limited by its physical size. In the case of

microprocessors, this problem is overcome by time-sharing the input/output pins.[96] This arrangement incurs a speed penalty on the microprocessor compared to a system which used independent input and output pins. The second limitation can be seen by regarding the microprocessor as a clocked sequential circuit whose processing activities are timed by a clock. Clearly the higher the clock frequency, the faster the system. The maximum clock frequency that can be used in a system is determined by the response time of the internal circuits and by the access time of the memory used. With present-day components the limiting factor in practice is usually the memory access time. It is worthy to mention that microprocessor are getting faster all the time, but on the other hand transmission rates are also increasing, thus demanding faster and faster microprocessors to cope with the faster processing requirements for real time processing on these high bit rate channels.

4.3.2 The Use of Microprocessor In Decoding Block Codes

Most decoding algorithms for error-correcting codes are based on the use of a microprocessor to perform the algorithm functions. Although decoding algorithms that use microprocessors were used for error-correction in hard decision decoders, the use of microprocessors become inevitable, due to the decoding algorithms complexity. When a microprocessor is used in the decoding process, the relation between the necessary time for exceuting the algorithm used, and the data receive time is very important

since for a practical system with a real-time decoding it could mean the difference between using a particular algorithm or discarding it. At the other extreme one has the choice of a faster and more expensive microprocessor, or reducing the data transmission rate.

From this point, since we are interested only in block error-correcting codes, we are going to consider those codes only. However, the same argument stands for any sort of codes, but some of the equations developed might need some modification.

To derive the equation that relates data receive time and the algorithm execution time, assume an (n,k) block error-correcting code is used in a transmission system, where the decoder is a microprocessor that executes a certain algorithm for decoding. The data is assumed to be transmitted through a transmission channel that has an error rate of p, and the transmission is assumed to be m bits per second. The time t_1, of received one block of n transmitted digits is

$$t_1 = \frac{n}{m} \text{ seconds} \qquad (4.9)$$

To use a microprocessor that uses the algorithm in real time, the microprocessor should be able to execute the algorithm in a time, say t_m, less or at the most equal to the time of receiving one block of the transmitted digits, that is

$$t_m \leqslant t_l \tag{4.10}$$

4.3.3 Idle Time Index and Usage Efficiency

The algorithm execution time t_m is a good measure of the microprocessor choice. If t_m was, for example, much smaller than t_l, it indicates a badly designed system, because a slower microprocessor can be used, or additional duties can be done by the microprocessor, during the remaining block time, thus cutting down the peripheral hardware cost and complexity. The Idle Time Index I of the microprocessor for one block is, by definition, the fraction time of the total block time t_l, where the microprocessor is idle.

$$I = \frac{t_l - t_m}{t_l} \tag{4.11}$$

The Idle Time Index can vary theoretically from zero, where the microprocessor is used all the time to one, where the microprocessor is not used at all. The microprocessor usage percentage efficiency η , when represented as a percentage of the total time, can be written as

$$\eta = 100 \left(\frac{t_m}{t_l} \right) \% \tag{4.12}$$

Any error-correcting decoder, has to start the decoding process by calculating the syndrome of the received word regardless of whether the decoder is a hardware decoder, a software decoder, of a mixture of

both. Let the syndrome calculation time be t_s, when a hardware decoder is used t_s is negligible, while usually it has a value other than zero for software decoders. The second step is the detection process, where the calculated syndrome is tested, if the test result is zero, i.e. all the syndrome components are zeros, then the received word is accepted as the transmitted codeword, although it could be a codeword other than the transmitted codeword. While if any one or more of the syndrome components are not zeros, then an error is detected in the received word. The third step is to calculate the error pattern that was present in the transmission channel during the block transmission, and correct the erroneous digits accordingly, which is done by executing the algorithm. The last step is to output the corrected digits to the next stage of the system.

Consider an error-correcting decoder which decodes the received block totally by a sotfware algorithm, i.e. the syndrome calculation, error detection, and error correction is done by a microprocessor. Let the total correction time t_m for correcting an erroneous block t_2, and the time to calcultate the syndrome only be t_s. Then the idle time index for an erroneous block I_e, can be calculated from Eq.(4.11)

$$I_e = \frac{t_1 - t_2}{t_1} \qquad (4.13)$$

while the idle time index I_c for a block which has a zero syndrome is

$$I_c = \frac{t_l - t_s}{t_l} \qquad (4.14)$$

In the case of using software algorithms for decoding the received word, I_e is bigger than I_c, the ratio $\frac{I_e}{I_c}$ is dependent on the algorithm used and the code rate. In the all software decoder the ratio $\frac{I_e}{I_c}$ is about 15 for the hard decision system, to 250 for the soft-decision decoders, for code rates of about 0.5, while the ratio is a very large number if hardware syndrome calculation is used with software detection and correction.

4.3.4 Decoder Idle Time

Assuming a microprocessor is used for executing the decoding algorithm, although Eqs.(4.13) and (4.14) give an idea about the use of the microprocessor during each received block, they are not suitable for measuring the overall microprocessor usage during the whole transmission, because some of the received blocks will be correct, while some other will be erroneous. Thus to calculate the overall microprocessor usage, additional information about the correct and erroneous received blocks is required.

Let, by definition, the decoder idle time be the total time during the whole transmission where the microprocessor is idle, which gives a clear idea about

the microprocessor usage. The error probability can provide the necessary information about the received block. Let the probability of generating an error in the transmission channel be p. Then the probability of receiving a block of n digits free of error P_c is, the probability that each of the received digits is correct, that is

$$P_c = (1-p)^n \tag{4.15}$$

and the probability of receiving a block which contains at least one error P_e is

$$P_e = 1 - (1-p)^n \tag{4.16}$$

Actually, since the decoder will treat any erroneous block received as another codeword as a correctly received block, Eq.(4.16) is not very accurate[6,67]. But for simplicity it will be assumed that Eq.(4.16) is accurate enough.

It is reasonable to assume that the correction algorithm is not executed when a correct block is received. This leaves the decoder idle during the decoding of error free blocks except of the syndrome calculation time t_s. Consequently the decoder idle time I_d is

$$I_d = I_e P_e + I_c P_c \tag{4.17}$$

substituting Eqs.(4.13),(4.14),(4.15) and (4,16) in Eq.(4.17) gives

$$I_d = \frac{t_1 - t_2}{t_1} \{1 - (1-p)^n\} + \frac{t_1 - t_s}{t_1} (1-p)^n \qquad (4.18)$$

The first part $I_e P_e$ is a small fraction of the total time, because I_e is small and P_e is small, while $I_c P_c$ is the dominent part. Consequently, the microprocessor is idle mostly because of receiving correct blocks most of the transmission time, and since the probability of receiving correct blocks is relatively high, then I_d is close to 1.

To get a rough idea about the decoder idle time in a real system, consider a channel which has a probability of error p, where $p=10^{-3}$ as it was taken in chapter three. Let the error-correcting code be an (n,k) random code, where the block length n=31. And let the idle time index I_e for erroneous block be $I_e=0.1$, and the idle time index for error-free block be $I_c=0.8$. substituting in Eq. (4.18)

$$I_d = 0.1 \{1-(1-10^{-3})^{31}\} + 0.8(1-10^{-3})^{31} = 0.7786 \qquad (4.19)$$

The decoder idle value of Eq.(4.19) is the worst case value, because the channel is considered a random channel. While if a bursty channel which has the same error rate is considered, the decoder idle time is higher, because for such channel errors tend to be concentrated in a few

blocks. Which leaves most of the other blocks error free.

Any decrease in I_d improves the effective usage of the microprocessor timewise. Such a decrease will result in an increase of the correction time t_2, whether the increase is due to the increase of the syndrome calculation time or the algorithm execution time. For a real time usable decoder any increase in t_2 should allow the microprocessor to execute the correction algorithm in the block receive time t_1, i.e. the erroneous block idle time index should not become negative.

$$I_e \geq 0 \qquad (4.20)$$

there is no need to restrict I_c separately, because since it is included in t_2, then it is restricted indirectly by Eq.(4.20).

4.3.5 The Basic Idea of Time-Shared Decoders

Time-shared decoder is a technique to increase the microprocessor effeciency and decreasing the decoder idle time to some negative value, yet using the decoder in real time transmission, which is done by avoiding Eq.(4.20). When I_e became negative Eq.(4.20) can be rewritten

$$I_e < 0 \qquad (4.21)$$

substituting I_e from Eq.(4.13) and dividing both sides by t_2 gives

$$t_1 < t_2 \tag{4.22}$$

that is allowing the microprocessor to expand the correction time of the erroneous block i into the next i+1 received block correction time $t_{m,i+1}$. If the $i+1^{th}$ block is theerror free then the remaining time after correcting the i^{th} block will be enough to calculate the syndrome and accept the data if

$$t_2 + t_s \leqslant 2t_1 \tag{4.23}$$

otherwise the microprocessor is allowed to share the $i+2^{th}$ calculation time $t_{m,i+2}$. On the other hand if the $i+1^{th}$ block is erroneous the microprocessor is allowed to expand into the $i+2^{th}$ calculation time, and so on.

Assuming that the decoder received j erroneous blocks, then one error-free block, the decoder will finish decoding all the received blocks at the end of the calculation time of the $j+1^{th}$ block $t_{m,j+1}$ if the decoder can satisfy the following equation

$$jt_2 + t_s \leqslant (j+1)t_1 \tag{4.24}$$

fig.(4.3) shows the decoder timings when one and two erroneous blocks are received. It is assumed

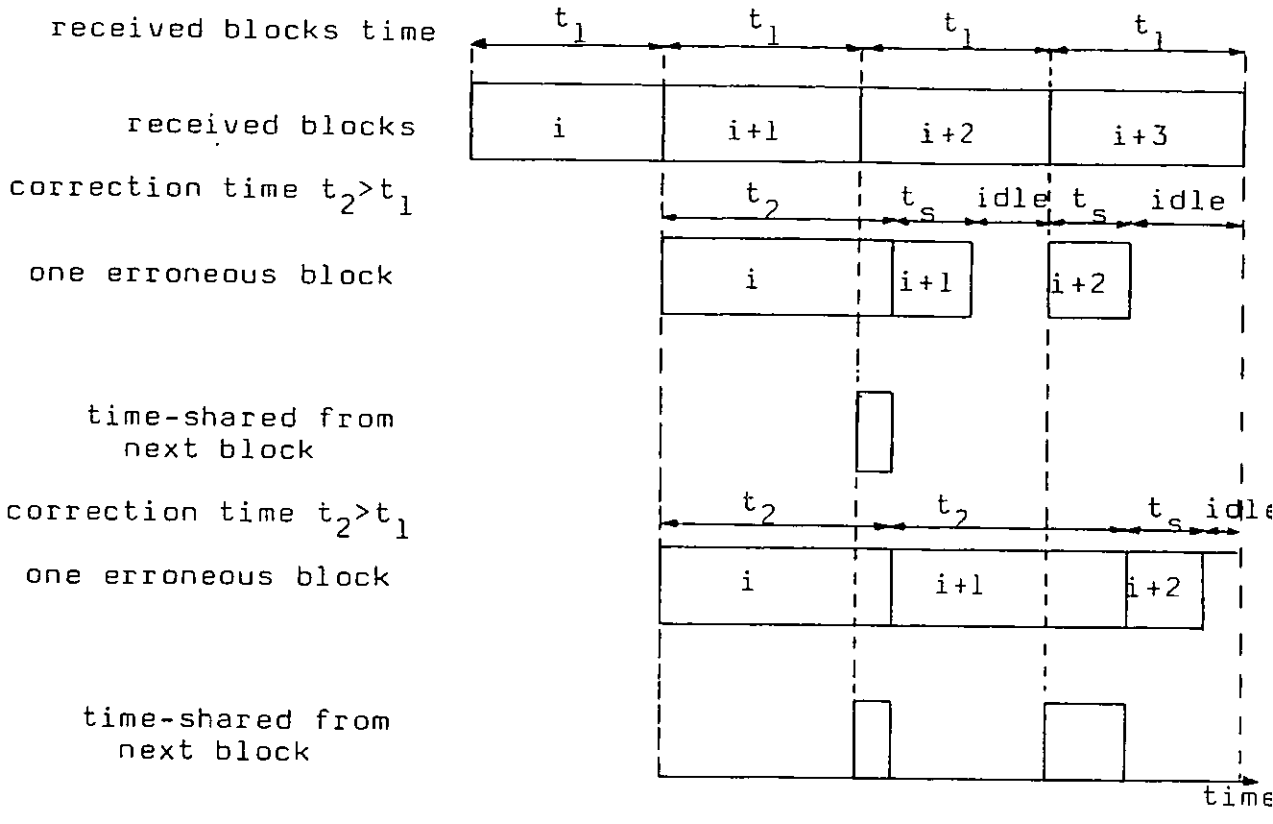


Fig.(4.3) time diagram for time-share and correction

in fig.(4.3) that the decoding process starts after receiving the last digits in the block, furthermore it is assumed that t_s is relatively very long time compared with t_2 for the drawing clarity. The delay introduced by this technique is

$$D = j\,(t_2 - t_1) \qquad (4.25)$$

where j is the number of consecutive erroneous blocks.

4.3.6 Buffering Requirements

Once a delay is introduced into a system some sort of storage is required to cope with this delay. Clearly

a buffer is required in the time-shared decoder system, because, as it can be seen from fig.(4.3), while the microprocessaro is executing the algorithm for correcting the i^{th} block, the $i+1^{th}$ block was received, and the $i+2^{th}$ is received, so unless the $i+1^{th}$ block is stored it will be lost. The storage space required to handle the incoming digits without any loss of information is dependent on the maximum number of the consecutive erroneous blocks.

4.3.6.1 Random-Error Channel Buffer

Consider a time-shared system that is used to correct errors that results from transmission through a random-error channel. Assuming that a buffer of one block storage space is used, it is required to calculate the maximum number of consecutive erroneous blocks that the system can cope with, without any loss of information.

To start with assume that no erroneous blocks are received so far. The buffer is empty because each received digit is read out as soon as its written in. Assume now that an erroneous block is received, all its digits will be read out as before. Because the decoder is executing the correction algorithm the first received digit will not be read out until the decoder has finished correcting the erroneous block. The number of digits in the buffer up to the moment where the decoder has finished the correction process can be calculated as follows. The duration time of one digit is

$$\tau = \frac{1}{m} \qquad (4.26)$$

where m is the transmission rate, and the delay resulting of one erroneous block from Eq.(4.25) is

$$D = t_2 - t_1 \tag{4.27}$$

Then B the number of digits in the buffer is

$$B = \frac{D}{\tau} \tag{4.28-a}$$

$$B = (t_2 - t_1)m \tag{4.28-b}$$

if the next received block is erroneous then an additional B digits are stored in the buffer, and so on. In the general case where j consecutive erroneous blocks are received, the number of digits in the buffer, given that the buffer was empty before receiving the first erroneous block, is

$$B = j(t_2 - t_1)m \tag{4.29}$$

Clearly there is no loss of information as long as all B digits are stored. So for the case where one block storage space is available, the maximum number of consecutive blocks received without any loss in the information is when

$$B = n \tag{4.30}$$

Thus from Eq.(4.30) and (4.29)

$$j = \left\{ \frac{n}{(t_2 - t_1)m} \right\} \tag{4.31}$$

substituting Eq.(4.9) in Eq.(4.31) gives

$$j = \left\{ \frac{t_1}{(t_2 - t_1)} \right\} \tag{4.32}$$

where {f} represent the nearest integer $\leq f$. This is used since no part of an erroneous block can be accepted. In practice there is no need to force the buffer to be one block or multiple of a block, so if the buffer has Z digits store space, then Eq.(4.31) can be rewritten

$$j = \left\{ \frac{z}{(t_2 - t_1)m} \right\} \tag{4.33}$$

4.3.6.2 Bursty Channel Buffer

The use of time-shared decoders to decode transmitted signals through a bursty channel makes the use of buffers inevitable. To find the storage space required for such channels, assume that ℓ_{max} is the length of the longest burst that may occur in the channel, and that an (n,k) code is used. Two cases must be considered.

Firstly, when the code is not interlaced. Any burst will produce a number of consecutive erroneous blocks. Thus the storage requirements can be calculated from Eq.(4.29). Let j_{max} be the maximum number of blocks that

can be affected by the maximum length ℓ_{max}, then

$$j_{max} = \frac{\ell_{max}}{n} + 1 \qquad (4.34)$$

Substituting Eq.(4.34) and Eq.(4.29) gives the minimum buffer requirements

$$B_{min} = (\frac{\ell_{max}}{n} + 1)(t_2 - t_1)m \qquad (4.35)$$

Secondly, the interlaced codes. Assuming that the interlacer memory space is equal to M_{sp} then the number of blocks that can be fitted in the interlacer is

$$u = \frac{M_{sp}}{n} \qquad (4.36)$$

Then the longest burst ℓ_{max} cannot affect more than 2u blocks.

For the convolutional interlacer, the buffer is placed after the deinterlacer, and should be able to cope with 2u consecutive erroneous block, thus from Eq.(4.29) the minimum buffer size required is

$$B_{min} = 2u(t_2 - t_1)m \qquad (4.37)$$

in practice values lower than 2u can be used, because the 2u blocks will not be all erroneous.

For the symbol interlacers and the random interlacers, the case is the same as above and Eq.(4.37) gives the buffer requirement. The buffer can be placed in two

different places, if the buffer storage space is taken as a multiple of M_{sp}

$$B = i.u(t_2-t_1)m \qquad \text{where } i = 1,2 \qquad (4.38)$$

Substituting Eq.(4.9) in Eq.(4.38)

$$B = i.M_{sp}(t_2-t_1)/t_1 \qquad \text{where } i = 1,2 \qquad (4.39)$$

Such buffers can be placed within the deinterlacer. As it was mentioned in section 4.2.3, deinterlacers are in effect two deinterlacers each working in a ping-pong configuration,the additional buffer can be placed with these two deinterlacers giving a total of three or more. The data is written in each one in turn and read out in the same sequence, this configuration simplifies the buffer control circuit. The other case is where i is a real number and not an integer, clearly the previous implementation cannot be used. The buffer is placed after the deinterlacer as in the convolutional interlacers.

4.3.6.3 Decoders Output Buffer

The use of time-shared decoders will introduce an additional delay into the decoder, this delay varies from zero for the error-free blocks to a maximum value depending on the maximum number of consecutive erroneous blocks j.

$$0 \leqslant D \leqslant j\,(t_2-t_1)m + D_{re} \qquad (4.40)$$

where D_{re} is the remaining delay from the previous set

of erroneous blocks, which can be neglected without serously affecting Eq.(4.40).

In some applications the whole block is treated as an individual. In such application the delay isnot a serious problem, and the data sink can wait for the correction delay. In others any delay is not tolerated, where the data should be outputed at a fixed rate, consequently, additional buffer storage is required at the output. The decoder output is written into the buffer until the buffer is full, before reading any data out. Once a delay occurs, the buffer contents will be reduced, but the buffer will be full again as soon as the micrprocessor has recovered from the delay.

Since the decoding time is no longer critical, the microprocessor can also be used to control the input and output buffers, giving a further saving in cost.

4.3.7 Modified Time-Shared Decoder for Bursty Channels

Although the time-shared decoder makes a better use of the microprocessor, the microprocessor is not used efficeiently even if it is decoding during all the transmission time. For example, assuming that our interest is concentrated on the k information digits, the microprocessor is attempting to correct errors confined to the parity-check digits which will be discarded anyway. A more efficient decoder will be one that can detect when the errors are confined to the parity-check digits, and in such a case it will accept the data without executing the correction algorithm, hence saving the delay resulting

from the decoding process, and reducing the queue length in the buffer.

4.3.7.1 Parity-Check Errors Trapping

The parity-check error-trapping is based on the error-trapping decoding idea discussed in chapter two. Assuming that an (n,k) code is used to correct errors in a transmission system, where the transmission channel can be either a random or bursty channel. A decoder that can detect the correct received words and words with the errors confinded to the parity-check digits without too many calculations is the error-trapping decoder. So to achieve the required modified algorithm, the parity-check error-trapping part of the error-trapping decoder is combined with the correction algorithm to be used. A summary of the modified algorithm follows.

STEP 1 After the receiving of the whole block, the syndrome is calculated.

STEP 2 The syndrome is tested and if it is all zeros, the received word is a codeword and the information digits are accepted as the transmitted data and the decoding is complete, otherwise:-

STEP 3 Is dependent on the type of code used. For random-error-correcting codes which can correct all errors of t or less, the syndrome is tested and if its Hamming weight is t or less then errors are confined to the n-k parity-check digits, and the information digits are error-free, which are accepted as the transmitted digits and the decoding is complete. Otherwise go to STEP 4.

For burst-error-correcting codes, which can correct all bursts of length ℓ or less, the leftmost $n-k-\ell$ stages of the syndrome are tested, if they contain zeros, then the error-pattern is trapped in the ℓ rightmost stages, otherwise the syndrome is shifted once again and the syndrome tested. If the $n-k-\ell$ leftmost stages contain zeros at the i^{th} shift, where $0 \leqslant i \leqslant n-k-\ell$, then the error-pattern is confined to the parity-check digits, the information digits are error-free, and are accepted as the transmitted information digits. Otherwise go to STEP 4.

STEP 4 The correction algorithm is executed and errors are corrected.

It should be noted that the algorithm should calculate the possible untrappable errors in the n-k parity-check digits.

4.3.7.2 Probability Of Parity-Check Errors-Trapping For Random Error Channel

When an (n,k) code is used for error correction in a random-error channel, where this code can correct all t or less errors, the probability of t or less errors confined to the parity-check digits can be calculated as follows. Let the probability of error in the transmission channel be p, then the probability of receiving a word containing v errors is the probability of receiving v errors given that the remaining n-v digits are error free, that is

$$p_v = p^v (1-p)^{n-v} \qquad (4.41)$$

The fraction of v error confined to the parity-check digits, to the total number of possible v errors in a codeword is

$$\frac{C_{n-k,v}}{C_{n,v}} = \frac{\frac{(n-k)!}{v!(n-k-v)!}}{\frac{n!}{v!(n-v)!}} = \frac{(n-k)!(n-v)!}{n!(n-k-v)!} \qquad (4.42)$$

from Eqs(4.41) and (4.42) the probability of v errors confining to the n-k parity-check digits is

$$P_v = \frac{(n-k)!(n-v)!}{n!(n-k-v)!} \; p^v \; (1-p)^{n-v} \qquad (4.43)$$

Since all t or less errors are trappable, then the probability of trapping t or less errors confined to the parity-check P_{et} is

$$P_{et} = \sum_{i=1}^{t} \frac{(n-k)!(n-i)!}{n!(n-k-i)!} \; p^i \; (1-p)^{n-i} \qquad (4.44)$$

4.3.7.3 Probability Of Parity-Check Error-Trapping For Bursty Channel

Assuming an (n,k) code is used for error correction in a bursty channel, where the code used can correct all bursts of length ℓ or less. Let the probability of generating errors in the channel is p. Since the channel is a bursty channel then errors tend to affect a number of consecutive digits. So it will be assumed that errors in this channel affect consecutive digits only, and that no random errors will occur outside the burst. The

probability of generating a burst of length ℓ or less, where the burst contain $1 \leqslant v \leqslant \ell$ errors, given that the remaining $n-\ell$ digits are error-free is

$$P_v = p^v (1-p)^{n-\ell} \qquad \text{where } 1 \leqslant v \leqslant \ell \qquad (4.45)$$

The total number of possible bursts of length ℓ in a block of n digits is

$$\gamma_1 = n \qquad (4.46)$$

the number of bursts of length ℓ confined to the parity-check digits is

$$\gamma_2 = n-k-\ell+1 \qquad (4.47)$$

from Eqs(4.45), (4.46) and (4.47), the probability of a burst of length ℓ containing v errors being confined to the n-k parity-check digits is

$$P_v = p^v (1-p)^{n-\ell} \frac{(n-k-\ell+1)}{n} \qquad (4.48)$$

where $0 \leqslant v \leqslant \ell$

Since all bursts of length ℓ or less are trappable, regardless of the number of errors in the burst, then the probability of trapping bursts of length ℓ or less confined to the parity-check digits P_{et} is

$$P_{et} = \sum_{i=1}^{\ell} p^{j} (1-p)^{n-\ell} \frac{(n-k-\ell+1)}{n} \qquad (4.49)$$

4.3.7.4 The Percentage Reduction In The Storage & Delay Due to The Modified Algorithm

The probability of receiveing erroneous blocks is given in Eq.(4.16), thus the percentage reduction in executing the correction algorithm R_{ed} is

$$R_{ed} = \frac{P_{et}}{P_e} 100\% \qquad (4.50)$$

Let the time required to calculate the syndrome and detect the parity-check digit errors be t_e, then for each parity-check erroneous block detected there is a reduction in the delay t_{red}, where

$$t_{red} = t_1 - t_e \qquad (4.51)$$

but according to Eq.(4.25), a delay D resulting from receiving j consecutive erroneous blocks is

$$D = j(t_2-t_1) \qquad (4.25)$$

Consequently, the detection of partiy-check erroneous block will result in reducing the buffer contents by v blocks if the buffer contains v blocks, where

$$v = \frac{t_1 - t_e}{t_2 - t_1} \qquad (4.52)$$

Let b_{red} be the reduction in the buffer size measured in bits, then Eq.(4.52) becomes

$$b_{red} = n \frac{(t_1 - t_e)}{t_2 - t_1} \quad . \qquad (4.53)$$

Conversley, the overall percentage reduction in the storage space is

$$B_{red} = n \frac{(t_1 - t_2)}{t_2 - t_1} \quad \frac{P_{et}}{P_e} \; 100\% \qquad (4.54)$$

where B_{red} is in bits, P_e is as given in Eq.(4.16) and P_{et} is as given in Eq.(4.44) or Eq.(4.49) depending on the channel type. The precentage reduction in the delay can be calculated in the same way as

$$D_{red} = D \frac{P_{et}}{P_e} \; 100\% \qquad (4.55)$$

4.3.7.5 Symbol Interlacer And Parity-Check Error -Trapping

The parity-check error-trapping technique has a big advantage when used with symbol interlaced error-correcting codes. Let us take a closer look at a burst generated in a channel. There are two possible places where a burst may be added, firstly, the whole burst may be added to the contents of one interlacer somewhere between the first and the λn^{th} digit, and in this case the maximum number of consecutive erroneous blocks

cannot exceed λ blocks. Secondly, the burst is affecting the contents of two adjacent interlacers and in this case the burst will affect the end digits of the first interlacer blocks, and the beginning digits of the second interlacer blocks, which are normally the parity-check digits. Assuming that the longest burst ℓ_{max} is

$$\ell_{max} < \lambda(\ell+1) \qquad (4.56)$$

Then any burst that affects two adjacent interlacers will not affect more than the parity-check digits of the second interlacer. Since all these parity-check errors can be trapped, then in this case also the maximum number of consecutive blocks for which the algorithm needs executing cannot exceed λ blocks. Thus Eq.(4.37) that states the minimum buffer requirements so that no information is lost can be rewritten for the symbol interlacer when a parity-check error trapper is implemented in the decoding algorithm, as

$$B_{min} = \lambda(t_2-t_1)m \qquad (4.57)$$

4.3.8 An Algorithm For Decoding Single-Burst Errors

The algorithms described are algorithms for correcting burst errors. The reason for choosing a bursty channel is, firstly to get information about the behaviour of soft-decision decoding for bursty channels, secondly to study the effect of quantization, and thirdly to study the

algorithm performance itself. Although these algorithms as described are for decoding burst-error-correcting codes, they can be easily modified to be used for decoding random-error correcting codes.

4.3.8.1 First Algorithm

This algorithm has a basic structure as the general description of the modified algorithm in section 4.3.7.1, and can therefore trap all errors that are confined to the parity-check digits. Assuming that an (n,k) code which can correct all bursts of length ℓ or less is used to correct the transmission channel bursts, then the decoding algorithm can be described by the following steps.

STEP 1 The syndrome S(X) of the received word is calculated and tested. If the syndrome is all zeros, the received word is a codeword, the information digits are accepted as the transmitted data, and the decoding is finished. Otherwise.

STEP 2 The syndrome is tested for $n-k-\ell$ zeros in the $n-k-\ell$ leftmost stages. If they are detected, then the burst pattern is confined to the ℓ rightmost stages of the syndrome register, and the information digits are error-free. Hence the data is accepted, and the decoding is finished. Otherwise,

STEP 3 The syndrome is shifted with the feedback connection affecting the syndrome to the right once, and its contents tested up to i times, where $1 \leqslant i \leqslant n-k-\ell$. As soon as the $n-k-\ell$ leftmost stages contain only zeros, the ℓ rightmost stages contain the error-pattern, and the error burst is confined to the parity-check section of

the received word. Thus the information digits are accepted as error-free, and the decoding is finished. If at the end of the $n-k-\ell^{th}$ shift the error is not trapped then the burst has affected the information digits and to correct them the correction algorithm has to be executed. To start this, the shifts counter is set to zero.

STEP 4 The syndrome is shifted with the feedback considered to the right once, and the error-pattern soft weight (EPSW) is calculated from Eq.(2.50) for this error-pattern, then the syndrome contents, the EPSW, and information about the syndrome location are stored.

STEP 5 STEP 4 is repeated n times and at each time the EPSW is compared with the EPSW stored, if the new EPSW is found to be smaller then the stored EPSW, then the new syndrome contents, EPSW, and the syndrome location are stored.

At the end of the n shifts, the syndrome store contains the error-pattern which has the lowest EPSW, i.e. the most likely error-pattern to have been added to the transmitted codeword during transmission.

The content of the syndrome, after the $k+\ell^{th}$ shift, is the syndrome $S(X)$ of the received word, thus the EPSW calculated corresponds to the error-pattern that is confined to the parity-check digits, which was not calculated at the beginning of the decoding. It should be noted that the burst length of the error-pattern at the syndrome after the n^{th} shift is longer than ℓ, otherwise the error-pattern would have been detected in STEP 2 or STEP 3.

STEP 6 Using the information stored in the syndrome location store , the syndorme is modulo-two added to the corresponding digits for correction before the data is read out, thus the decoding is finished.

A complete process flowchart is shown in fig.(4.4)

4.3.8.2 Second Algorithm

The second algorithm introduced is based on the same idea as the first algorithm. Three major modifications are carried out to improve the algorithm execution time without degrading the decoder performance as follows.

1-Once the syndrome for the received word is calculated, it is tested for an all zeros syndrome, then it is tested for n-k-ℓ zeros in the leftmost stages. Whenever any of these tests are satisified, the information digits are accepted as the transmitted data. Since, whenever the first test is satisfied the second one will be satisfied, then the first test can be omitted from the algorithm.

2-The EPSW as is given in Eq.(2.50) is dependent on the '1's in the syndrome. Since the '1's in the syndrome do not change unless the last rightmost stages contain '1', then the syndrome will contain the same error-pattern as long as no shift has occured where the last rightmost stage contains '1', i.e. the syndrome contents will be the previous contents shifted once to the right till a '1' at the $n-k^{th}$ stage changes the error patern. So the EPSW will not change its value until after a shift where the last rightmost stage of the syndrome contains '1'. Consequently, the same results can be achieved with less calculation if the EPSW is calculated only when the last

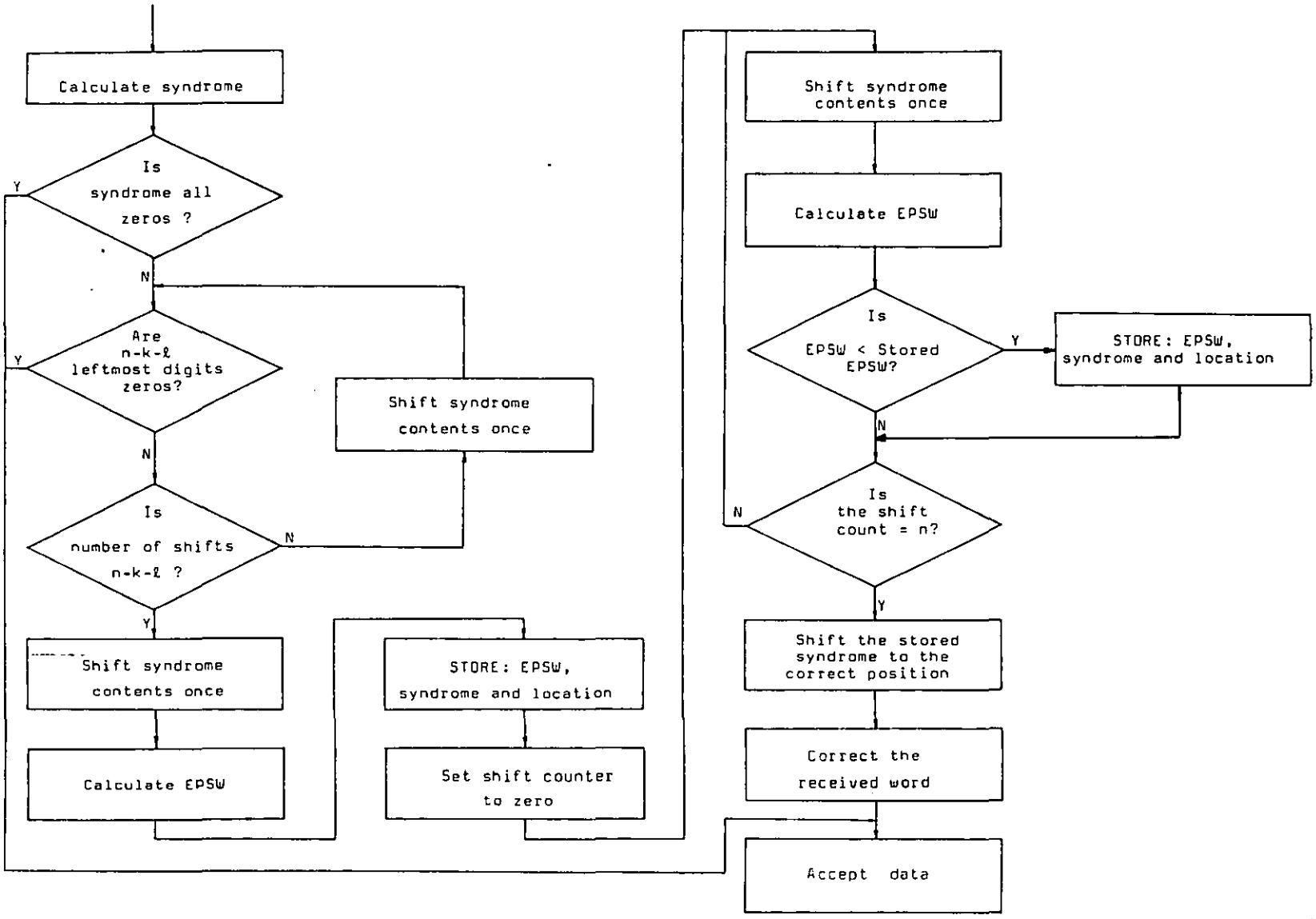


Fig.(4.4) Flowchart of first algorithm

rightmost stage is '1'.

3-To reduce the EPSW calculations further, it was assumed that the transmission channel generates bursts in such a way that each burst of any given length is less likely to be generated, than each burst of any shorter length. Thus the EPSW is more likely to have its smallest value for shorter error-patterns. The decoder stores a number of the shortest error-patterns, then the EPSW is calculated for these error-patterns and the one with the lowest EPSW is chosen as the error-pattern added to the transmitted signal in the channel. Clearly the bigger the number of error-patterns stored, this algorithm will perform similar to the first algorithm. While if the number goes down to one it will perform as the optimum decoder of section (2.3).

Considering the three points mentioned above, the second algorithm can be described in the following steps.

STEP 1 The syndrome S(X) of the received word is calculated, then tested for n-k-ℓ zeros in the n-k-ℓ leftmost stages. If these zeros are detected, then, either the syndrome is zero, i.e. the received word is a codeword, or the burst pattern is confined to the ℓ rightmost stages. In both cases the information digits are accepted as the transmitted data, and the decoding process is finished. Otherwise

STEP 2 With the feedback connection considered the syndrome is shifted to the right once, and its contents are tested. If the n-k-ℓ leftmost stages contain zeros at any stage of the shifting process, where the number of

shifts i is $1 \leqslant i \leqslant n-k-\ell$, the burst pattern is confined to the n-k parity-check digits, and the information digits are error-free. Thus the data is accepted, and the decoding is finished. Otherwise, the shifts counter is set to zero.

STEP 3 The syndrome rightmost digit is tested, if it is '0', the syndrome is shifted to the right once with the feedback connection considered, then STEP 3 is repeated from the beginning, while if the syndrome's rightmost digit is '1', the length of the burst pattern in the syndrome is calculated, and the syndrome, the burst length, and the syndrome location are stored.

STEP 4 STEP 3 is repeated till the shifts counter reaches n. Each time the burst length is calculated, the syndrome , the burst length, and the syndrome location are each stored in a different store.

At the end of the n shifts, the store area contains all the possible burst information for the received word, and includes any burst pattern that maybe confined to the parity-check digits, and which is longer than ℓ, i.e. untrappable.

STEP 5 All the bursts stored in STEP 3 and STEP 4 are sorted according to their length. Assuming that it is required to consider j bursts, then the EPSW of the shortest j bursts is calculated according to Eq.(2.50), and the burst which has the smallest EPSW is accepted as the burst added to the transmitted codeword in the channel.

STEP 6 Using the location information of the chosen burst, the syndrome is modulo-two added to the corresponding

digits for correction, then the data is read out, finishing the decoding cycle.

A flow chart of a complete process is shown in fig.(4.5).

4.3.8.3 Shortened Codes And The First And Second Algorithms

The use of shortened codes with these two algorithms can simplify the two algorithms furthermore. Given that an (n_1,k_1) code is shortened by β, so that the shortened code is the (n,k) code. Assuming that

$$n_1 - k_1 = n-k \qquad (4.58)$$

and

$$\beta > n-k \qquad (4.59)$$

which means that the code will not detect round the end burst, because they are spaced by β digits. Thus they cannot be trapped by the syndrome.

Assuming the the generator polynomial of the (n_1,k_1) code is used without any modification, clearly in this case the syndrome has to be shifted n_1 times instead of n times to get back to the received word syndrome. The algorithms for burst-error-correcting codes can be modified in the following way.

1-After the $n-k-\ell$ shifts the syndrome contents will correspond to the round the end bursts for the next i shifts, where $n-k-\ell<i<n-k$. After the $n-k^{th}$ shift the syndrome contents will correspond to the β inserted zeros until i reaches the $\beta-n-k$. After the next shift,

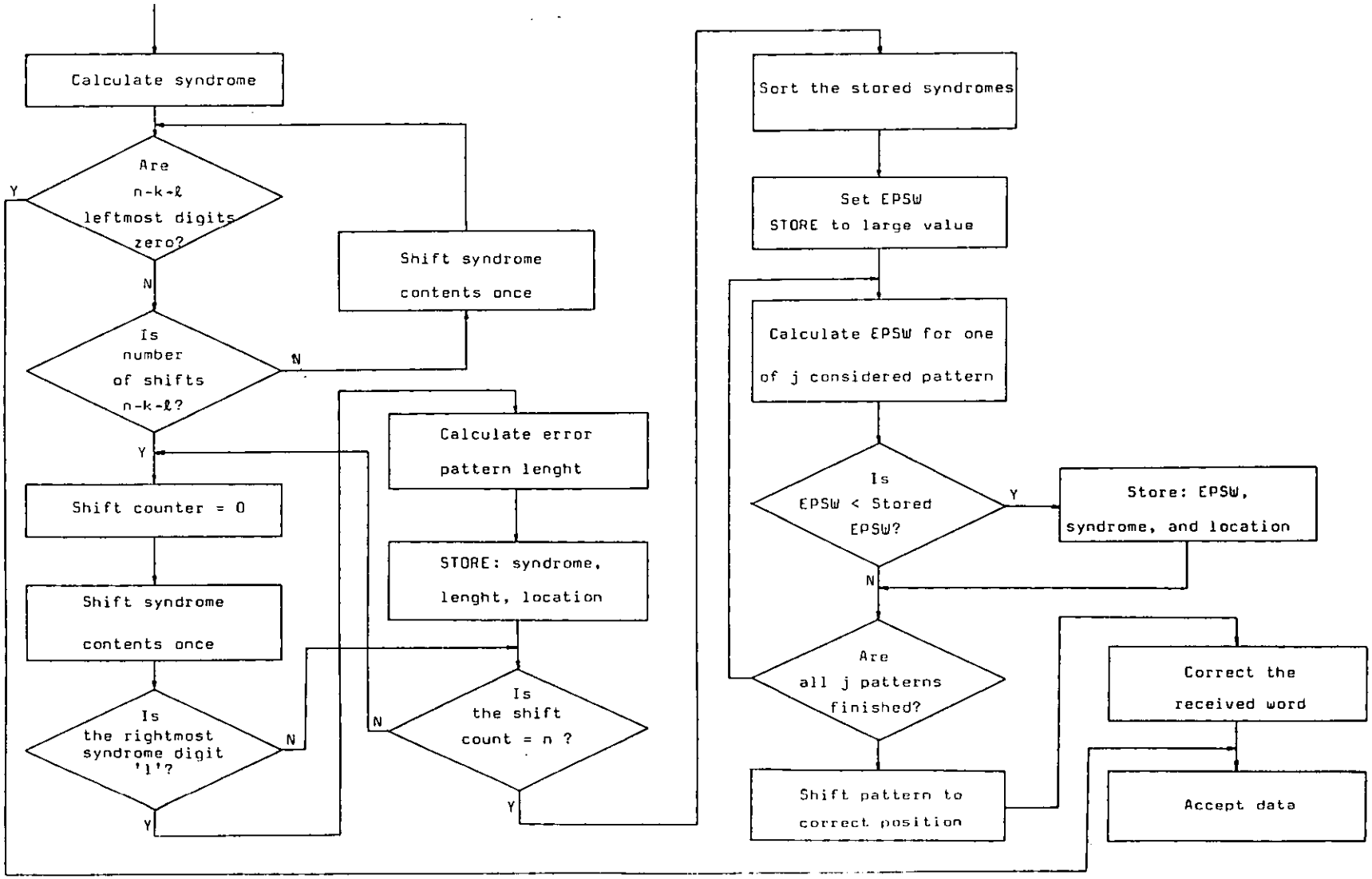


Fig.(4.5) Flowchart of the second decoding algorithm

i.e. the β-n-k+1th shift, the syndrome will correspond to the position X^{n-1} of the received polynomial R(X), and to the zeros in the positions $X^{n},X^{n+1},\ldots,X^{n-k-2}$. The next shifted syndorme will correspond to X^{n-2},X^{n-1} of R(X) and to the $X^{n},X^{n+1},\ldots,X^{n-k-3}$, and so on. Considering the fact that any burst present at X^{n-1} or X^{n-2},X^{n-1}, or....,X^{k+1}, $X^{k+2},\ldots,X^{n-1}$, is present at the positions $X^{k},X^{k+1},\ldots,$ X^{n-1}. Becuase for each case the rest are zeros, and cannot be erroneous. Then it can be seen that the syndrome contents does not give any additional information for all i shifts where

$$n-k-\ell < i < \beta+n-k \qquad (4.60)$$

This will lead to the insertion of additional STEP after STEP 3 in the first algorithm, and after STEP 2 in the second algorithm. The inserted STEP is as follows.

INSERTED STEP The syndrome contents are shifted with the feedback connection considered $\beta+\ell$ times to the right, and the shifts counter is set to zero.

The execution of STEP 4 and STEP 5 in the first algorithm, and STEP 3 and STEP 4 in the second algorithm, k times correspond to calculating the burst patterns that contain at least one information digits, the k+1 corresponds to the burst pattern confined to the parity-check digits. While the rest correspond to round the end burst patterns. Since these last patterns do not exist in the shortened codes, they can be excluded. This will lead to the substituting n in STEP 5 of the first algorithm, and STEP 4 of the second algorithm by k.

Since shortened codes where used in the simulation, these two modifications are included in the programmes of thses two algorithms in appendix D.

4.4-Simulation Results And Discussions

The (34,22) single-burst-error-correcting code is used in the simulation, this code is a shortened code from the (91,79) single-burst-error-correcting code, which has the generator polynomial $g(X)^{(55)}$.

$$g(X) = 1+X+X^3+X^4+X^5+X^6+X^9+X^{11}+X^{12} \qquad (4.61)$$

the generator of Eq.(4.61) is used as the generator polynomial for the shortened code.

Since all codewords are equally likely to be transmitted, then the use of a repeated codeword will result in the same test outcome. To avoid any complication at the transmitter end an all zero codeword is chosen to be transmitted repeatedly. Two types of Gaussian noise is added to the transmitted signal as described in chapter three. The received signal is quantized and fed to the decoder. A symbol interlacer and deinterlacer of degree λ=25 are used with all decoders.

4.4.1 First Algorithm Performance

4.4.1.1 Effect of Parity-Check Error-Trapping

As it was mentioned in section 4.3.8.1 the first algorithm employs a parity-check error-trapping technique. To study the effect of this technique, two decoders were

used to decode the same received signals, a soft-decision decoder and a decoder using parity-check error-trapping. The background SNR is assumed fixed at 9 dB while the burst SNR is varied from -20 to 6 dB, the resultant word error rate for both decoders are plotted infig.(4.6).

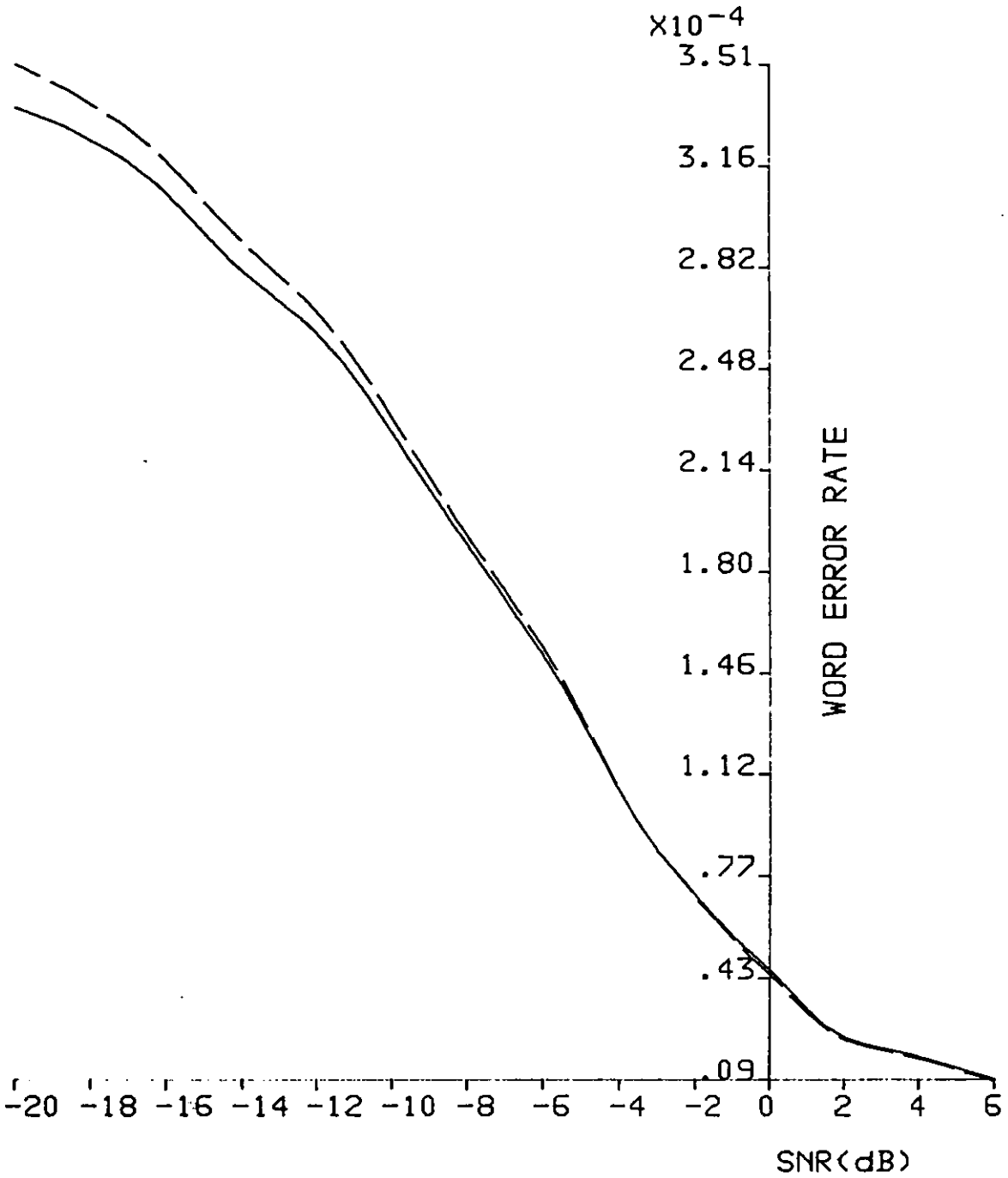
Clearly the curves of fig.(4.6) can be divided into two parts according to the burst SNR values. In the first part which is for burst SNR<-2.25 dB the parity-check error-trapping decoder outperforms the conventional soft-decision decoder and the improvement is increased with the decrease of burst SNR values, e.g. at burst SNR=-3 dB the improvement is 0.05 dB while at burst SNR=-17 dB the improvement is 1.05 dB. In the second part where the burst SNR<-2.25 dB the parity-check error-trapping introduces a degradation of 0.1 dB on average.

The parity-check error-trapping decoder is in effect an error-trapping decoder during the error detection in the parity-check digits only, and the reasons for its performance are discussed later in section 4.4.1.3.

4.4.1.2 Quantization Effects

A soft-decision decoder using the first algorithm is used to study the effect of the number of quantization levels. Linear law quantizers are used for all the tests, the number of quantization levels is varied from 4 levels to 64 levels. The results are plotted in fig.(4.7). As in the previous test the background SNR is set to 9 dB, while the burst SNR is varied from -20 dB to 6 dB.

Although one may expect the decoder performance to improve as the number of quantization levels increases,as



—— AL1 16 LEVELS

— — SD 16 LEVELS

FIG.(4.6) PARITY-CHECK ERROR-TRAPPING PERFORMANCE

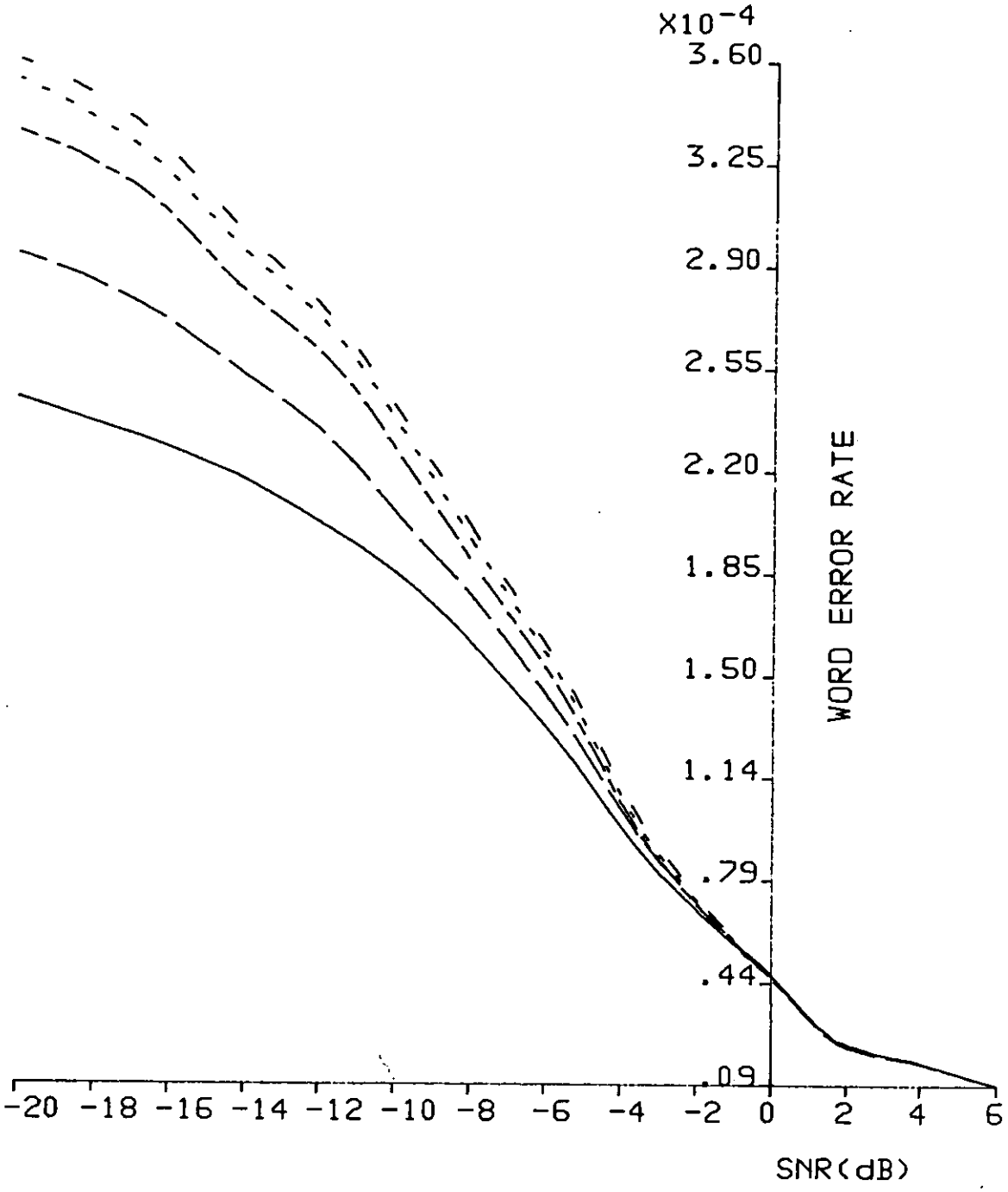


FIG.(4.7) QUANTISATION EFFECTS

it has been shown by a number of researchers at higher SNR values the results shown in fig.(4.7) show the opposite. The reason for this is that at low SNR values the noise imformation obtained by quantization is not correct, and the decoding process will contain a higher number of errors i.e. the larger the number of quantization levels the more the noise information will contain errors. This can be best described in the following example.

Example: Assume that two '0' digits are transmitted through a high noise power channel, i.e. low SNR value. The modulator will output to the channel two values

$$x_j = x_k = -V \tag{4.62}$$

during transmission noise will be added to these signals, let the two noise samples be w_j, w_k where

$$w_j = 2V \tag{4.63-a}$$

$$w_k = 0.8V \tag{4.63-b}$$

The received sampled digits will be according to Eq.(3.8)

$$y_i = x_i + w_i \tag{3.8}$$

Then

$$y_j = -V + 2v = +V \tag{4.64-a}$$

$$y_k = -V + 0.8v = -0.2V \tag{4.64-b}$$

Consider for simplicity that a two level decoder is used to decode the received digits. Such a decoder has a detecting threshold of zero volts, hence y_j will be detected as '1', and y_k as '0'. The confidence number for both digits are equal, thus the decoder will consider both digits to be equally likely candidates for being erroneous.

Next consider a four level decoder to decode the received digits, with equal spacing between the quantization levels. Such a decoder will have three detecting thresholds at +V/2, 0, -V/2 volts. Thus y_j will be detected as '1', with a confidence number of '1', while y_k will be detected as '0', with a confidence number of '0'. Although both digits are detected the same as in the two level detector, the decoder will consider y_k as a more likely candidate for being erroneous than y_j, which is not the case, and may result in erroneous decoding.

Clearly, at high SNR values, such a case will not arise very often, because the probability of the noise value having high power is very small, while the probability will increase as the noise power is increased, i.e. the SNR is decreased, causing more such cases, and eventually increasing the error rate. In the previous example the difference in the confidence number was '1' between the two samples for four quantization levels. If we assume that the number of quantization levels is increased further, clearly the received sample y_j will have the highest confidence number whatever the quantization levels. On

the other hand, the difference between the confidence numbers of y_j and y_k will increase for this case as the number of quantization levels is increased. Consequently, the decoder will treat y_k as a more likely candidate for being erroneous as the number of quantization levels is increased, thus increasing the error rate, as can be seen from fig.(4.7).

4.4.1.3 Decoder Performance

The test conditions for the decoder performance are the same conditions for the previous tests, i.e. the background SNR=9 dB, and the burst SNR is varied from -20 dB to 6 dB. Two decoders used for this test, a four levels decoder and a sixteen levels decoder, the test results are shown in fig.(4.8), where the error-trapping decoder, and the optimum decoder performance are plotted for comparsion.

As it is expected the optimum decoder performs better then the error-trapping decoder, because the channel statistics fits the assumed channel statistics of the optimum decoder. The 16 levels decoder performs better than the optimum decoder up the -5 dB point, and better than the error-trapping decoder up the -7.4 dB point. As the burst SNR value drops below these points, the first algorithm performs worse than the other two decoders, The 4 levels decoders performs in a similar way but the performance intersection points are -13 dB with the error-trapping decoder, and -6.8 dB with the optimum decoder.

When the background SNR is changed to 7 dB and the

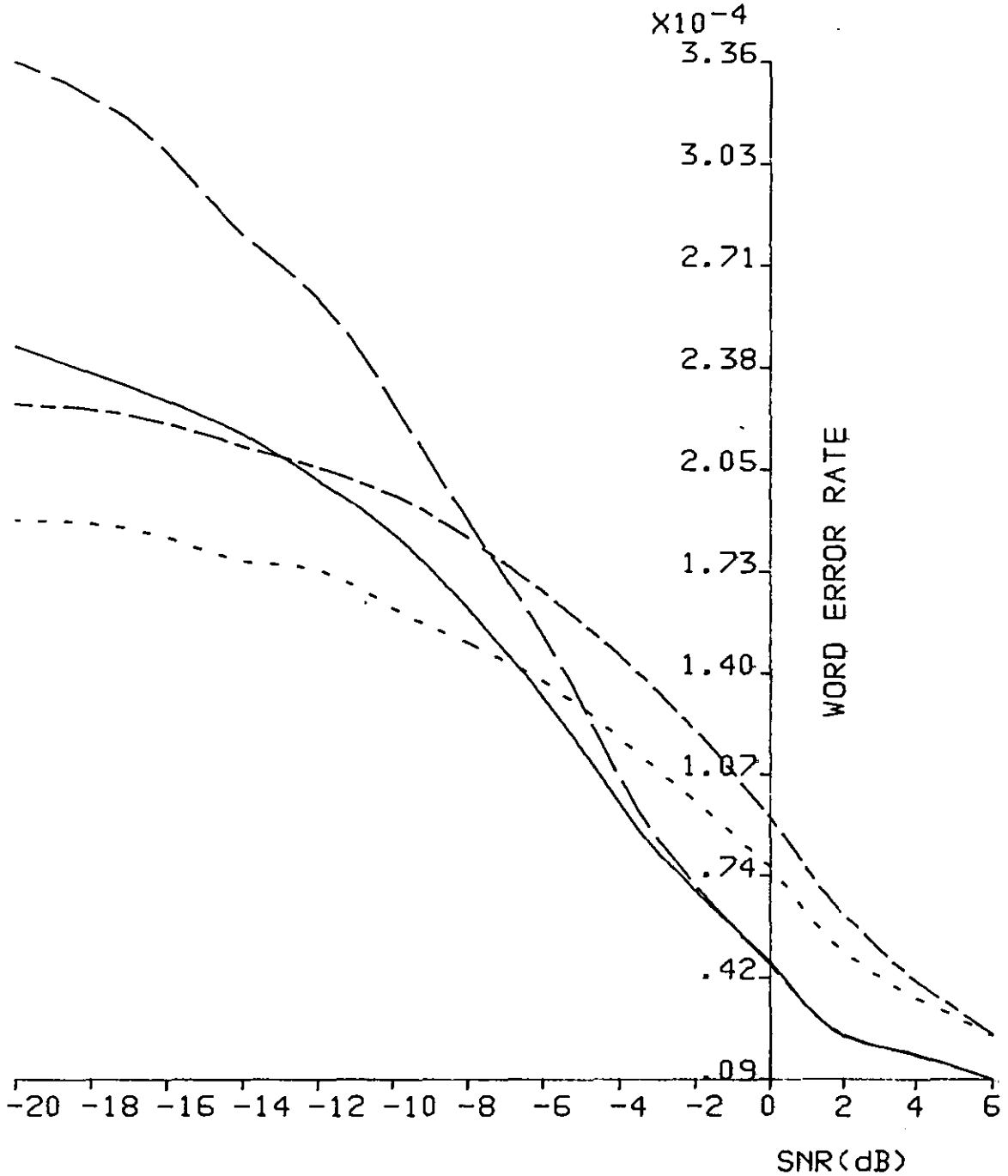


FIG.(4.8) AL1 PERFORMANCE AT BACKGROUND SNR=9 dB

same test is carried out, fig.(4.9) shows that the decoder performance is better at this lower background SNR value, yet the general shape of the curve is as before but the intersection points are shifted. For the 16 levels decoder the intersection points are -11 dB, and -17 dB, with the error-trapping decoder, and the optimum decoder results respectively, while for the 4 levels decoder the intersection point with the optimum decoder is at -19.5 dB, and the 4 levels decoder performs better than the error-trapping decoder for the whole studied range. When the background SNR is changed to 11 dB, fig.(4.10), shows that the intersection points becaome -6 dB, and -4 dB for 16 levels decoder, and -10 dB, and -5.6 dB for the 4 levels decoder with error-trapping deocder and the optimum decoder respectively.

The inferior performance of the decoder using the first algorithm at low burst SNR values is expected because, as mentioned in the previous section, the noise information is no longer correct, hence the calculated EPSW gives unreliable values, which leads to decoding errors. Eventually, any decoder that does not use the noise information will perform better as it is clear from figs.(4.8),(4.9) and (4.10). However as the background SNR is decreased the decoder performance using the first algorithm starts to improve over the error-trapping and the optimum decoders. The reason for this is that as the background SNR is decreased, the number of random errors is increased. In such cases the random errors may extend the burst length. For the error-trapping, if the extended

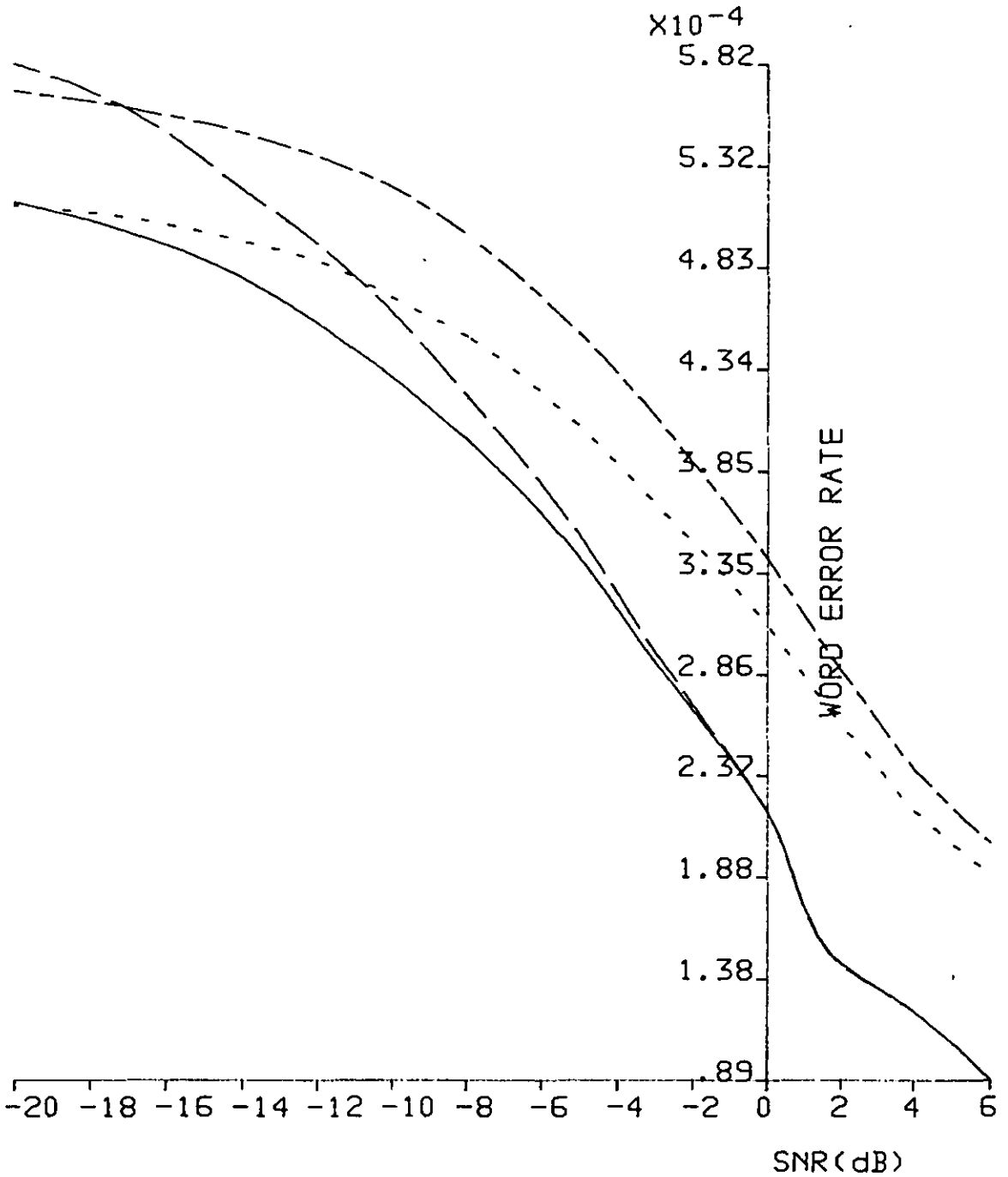
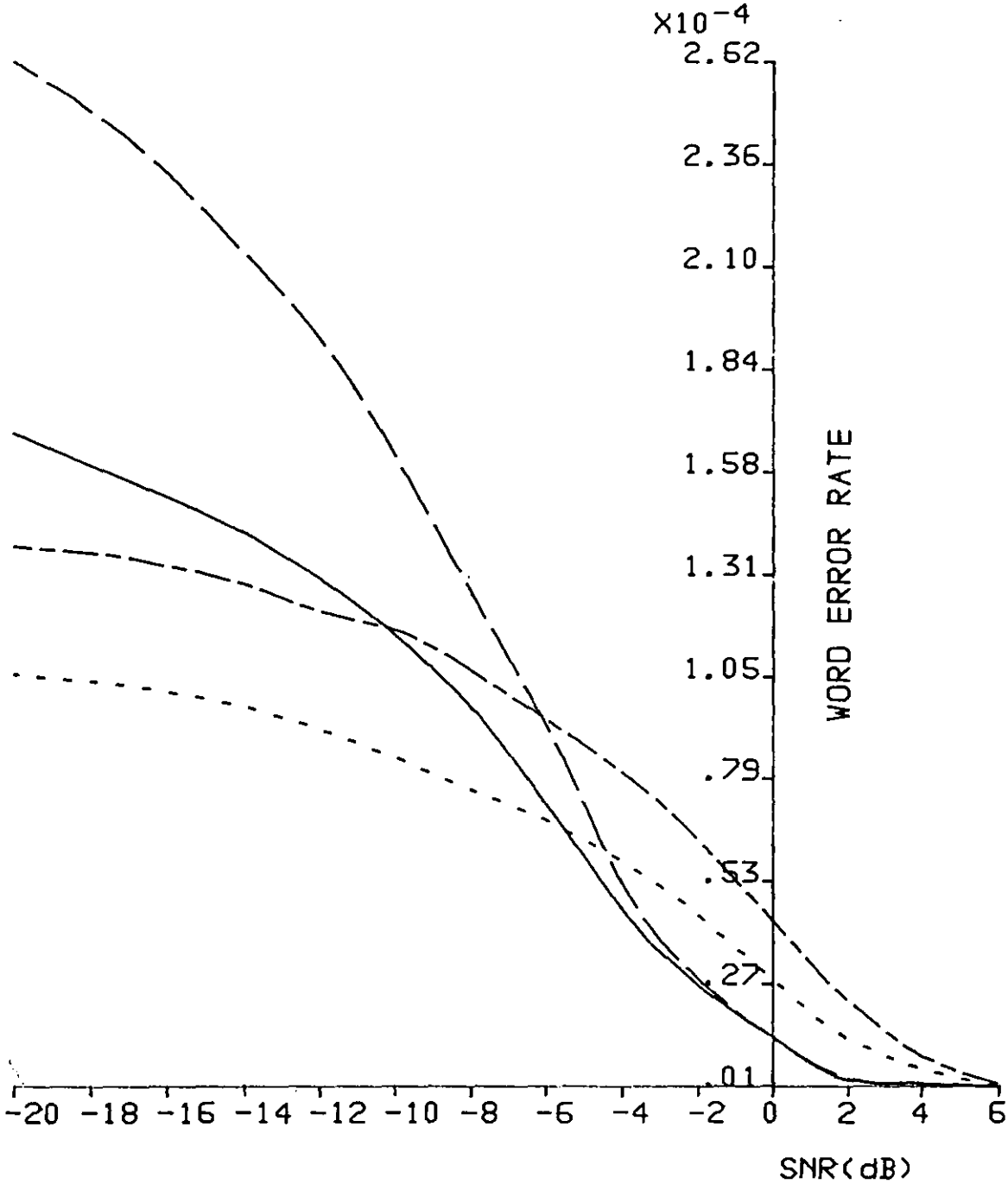


FIG.(4.9) AL1 PERFORMANCE AT BACKGROUND SNR=7 dB



____ 4 LEVELS _ _ E.T.

_ _ 16 LEVELS . . . OPT.

FIG.(4.10) AL1 PERFORMANCE AT BACKGROUND SNR=11 dB

burst is of length ℓ or less, the decoding is correct, otherwise it is either a decoding error or decoding failure. The optimum decoder will decode correctly if the extended burst is the shortest error-pattern detected and is unique. But if there exist two error-pattern of the same length as the extended burst length, the decoder is confused, and may decode erroneously, while if the extended burst is not the shortest detected error-pattern the decoding is erroneous. On the other hand, the first algorithm soft-decision decoder can be regarded as an optimum decoder that evaluates the error-patterns according to their soft-decision weight and not to the error-pattern length. In the cases where the burst SNR is very low, burst errors have in general, high confidence numbers, while since the background SNR is relatively high, random errors have, in general, low confidence numbers. Clearly, the actual error-pattern consists of burst errors and random error or errors, while other error-patterns may point to some burst errors and correct digits. The EPSW for the actual error-pattern is lower than the other error-patterns because of the random error or errors, thus the decoding is correct. But if the random error or errors are indicated by other error-patterns, the decoding may be erroneous. Consequently, for low burst SNR, and low background SNR values, the first algorithm soft-decision decoder performs better than the other two as it is clear from figs.(4.8), (4.9) and (4.10).

Needless to say that the first algorithm improvement is not expected to keep on increasing as the background

SNR is decreased, because as the background SNR is decreased the confidence number is increased. Subsequently, the difference in EPSWs will be smaller, and the decoder will make more decoding errors, thus reducing the improvement.

In general, the use of blankers at low SNR values will improve the algorithm performance. But an algorithm is not simulated because as it was mentioned in section 4.1 the simulated results give a superior performance to a real system, eventually, and the simulation will not be accurate to use for comparsion.

4.4.2 Second Algorithm Performance

4.4.2.1 The Effect of The Number of Error-Patterns Tested

The number of error-patterns tested in the decoding, has a significant effect on the second algorithm performance, as it was described in section 4.3.8.2. To see this, the second algorithm is tested in the following conditions; the background SNR=7 and 9 dB, the quantization levels were taken as 4 and 16 levels, while the number of error-patterns tested is changed in steps, 4,8,12 error-patterns for each of the quantization levels, and the burst SNR is varied from 6 dB to -20 dB.

The test results are shown in figs.(4.11), and (4.12), for burst SNR=7 dB and for 4 and 16 quantization levels respectively. Both figures show that for burst SNR values lower than the values of -6 dB for 16 quantization levels, and -10 dB for 4 quantization levels, the lower the number of the error-patterns tested the higher the improvement.

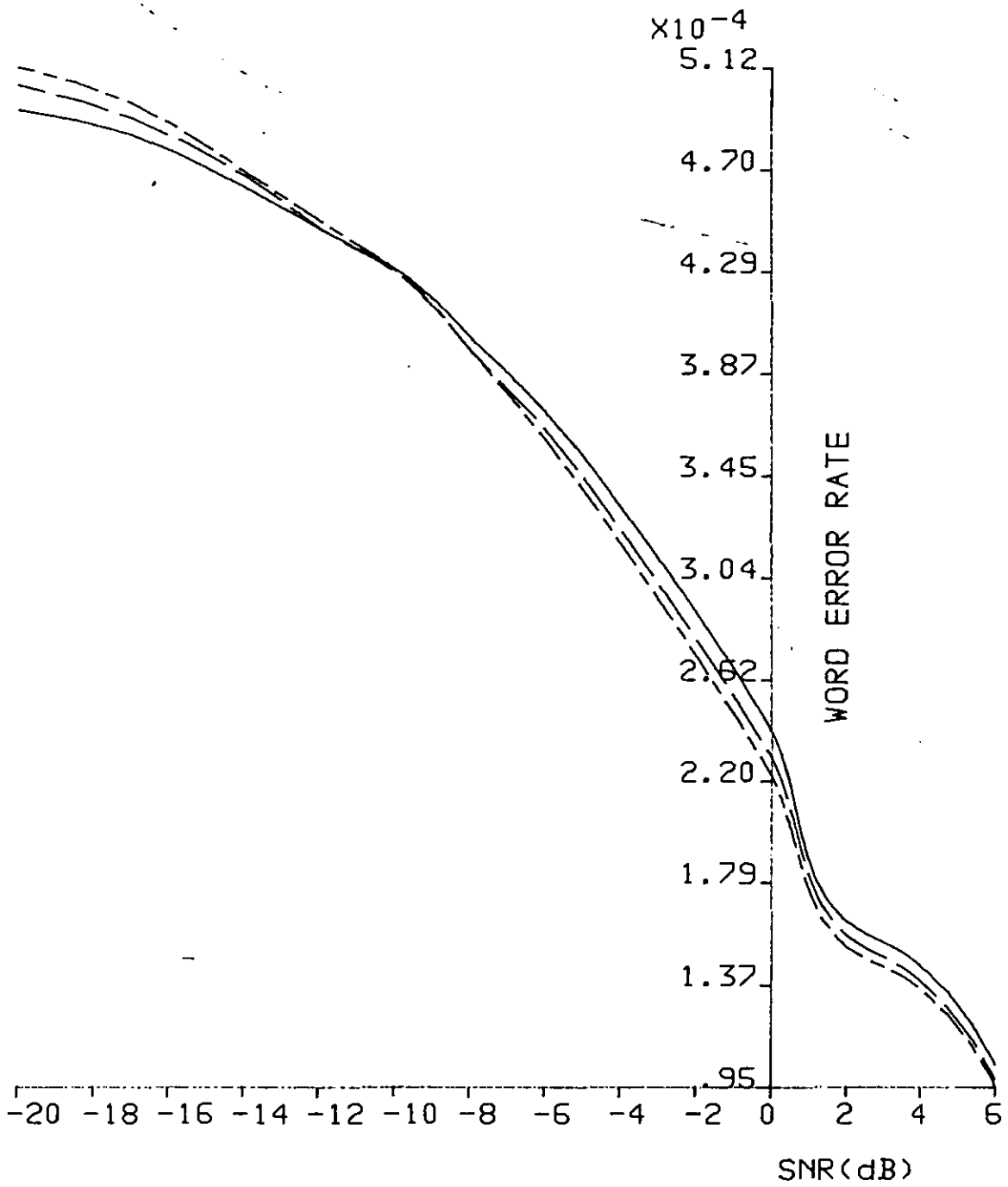


FIG.(4.11) NO. OF ERROR-PATTERNS TESTED EFFECTS

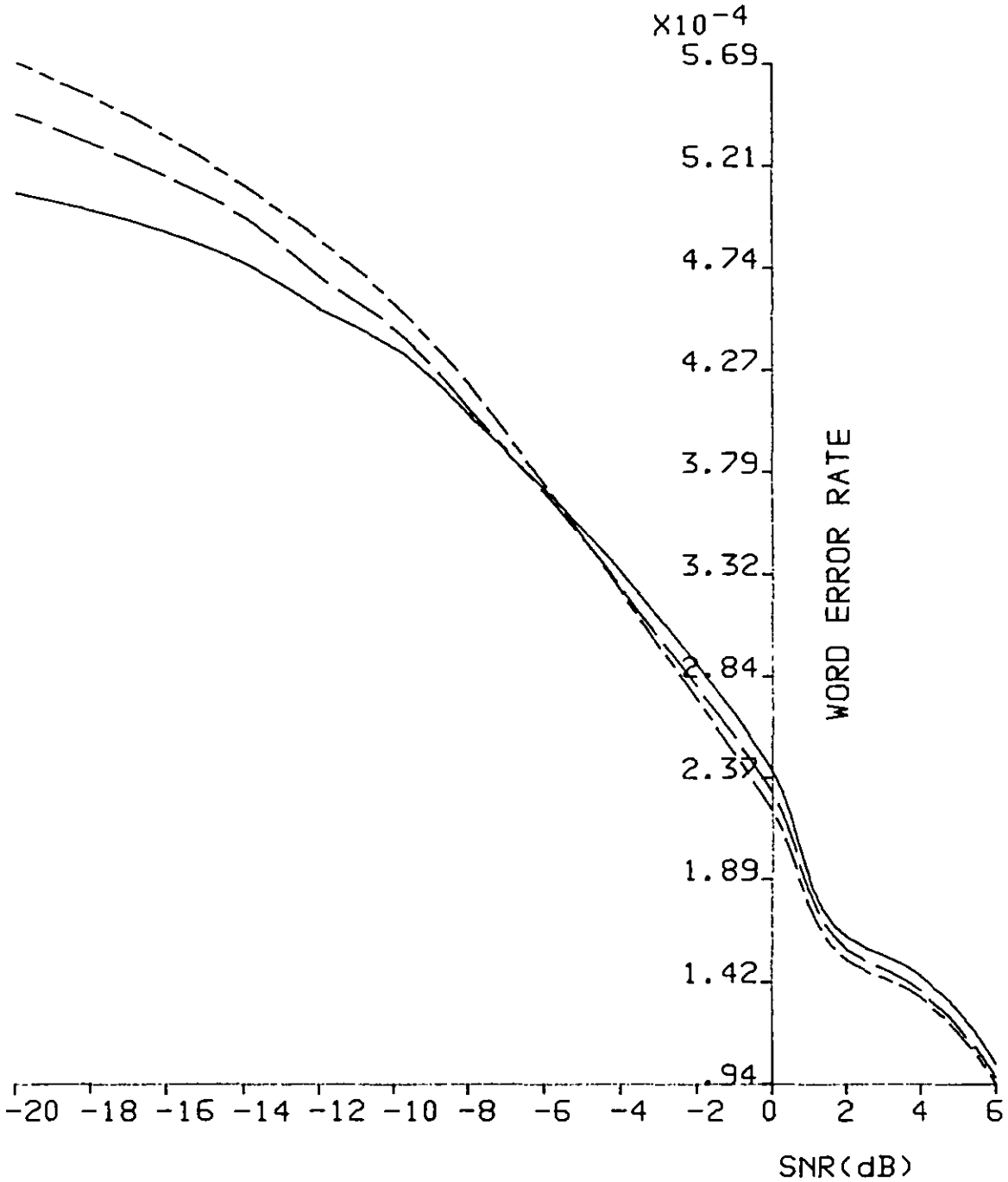


FIG.(4.12) NO. OF ERROR-PATTERNS TESTED EFFECTS

The test results for burst SNR=9 dB, shown in figs.(4.13), (4.14) for 4 and 16 quantization levels respectively, tell the same story, except that the intersection points of the burst SNR values are -2.6 dB, and between -7 dB and -3 dB for 16 and 4 levels respectively.

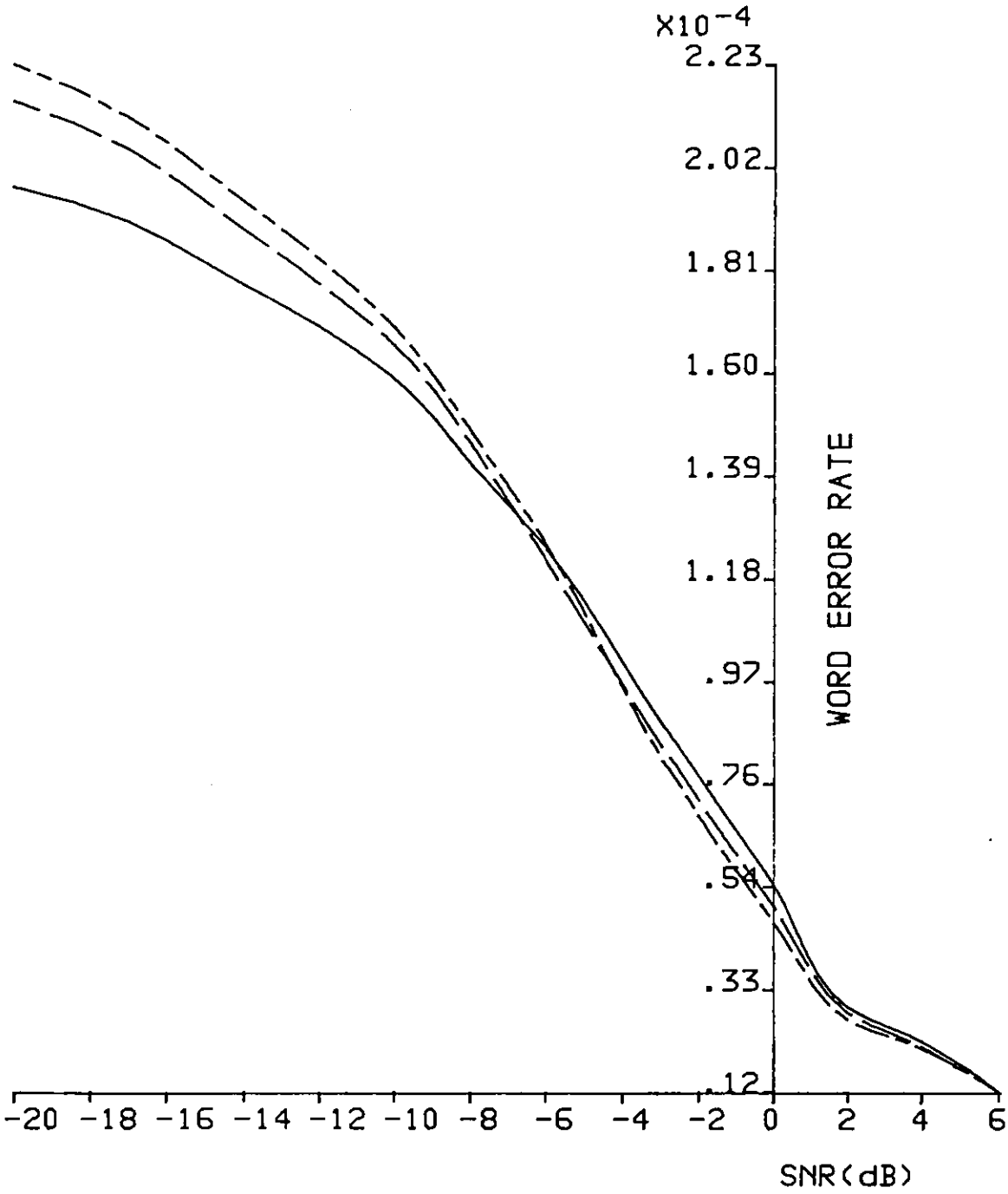
The degradation at high burst SNR values, however, is very small compared with the improvement when the burst SNR values drops to low values during transmission. However, this is not the case if the burst SNR value does not exceed the intersection point.

4.4.2.2 Quantization Effects

To study the quantization effects on the second algorithm performance, a test is carried out under the following conditions; The background SNR=9 dB, 8 error-patterns were tested, the burst SNR is varied from 6 dB to -20 dB, and the quantization levels are taken as 4,8,16, 32, and 64 levels for each burst SNR value. All quantizers used are uniform spaced level quantizers.

The test results are shown in fig.(4.15), where it can be seen that at low burst SNR values, the second algorithm performs better the lower the number of quantizations levels. While at high burst SNR values, say 2 db, a decoder using a 16 levels quantizer performs slightly better than the others.

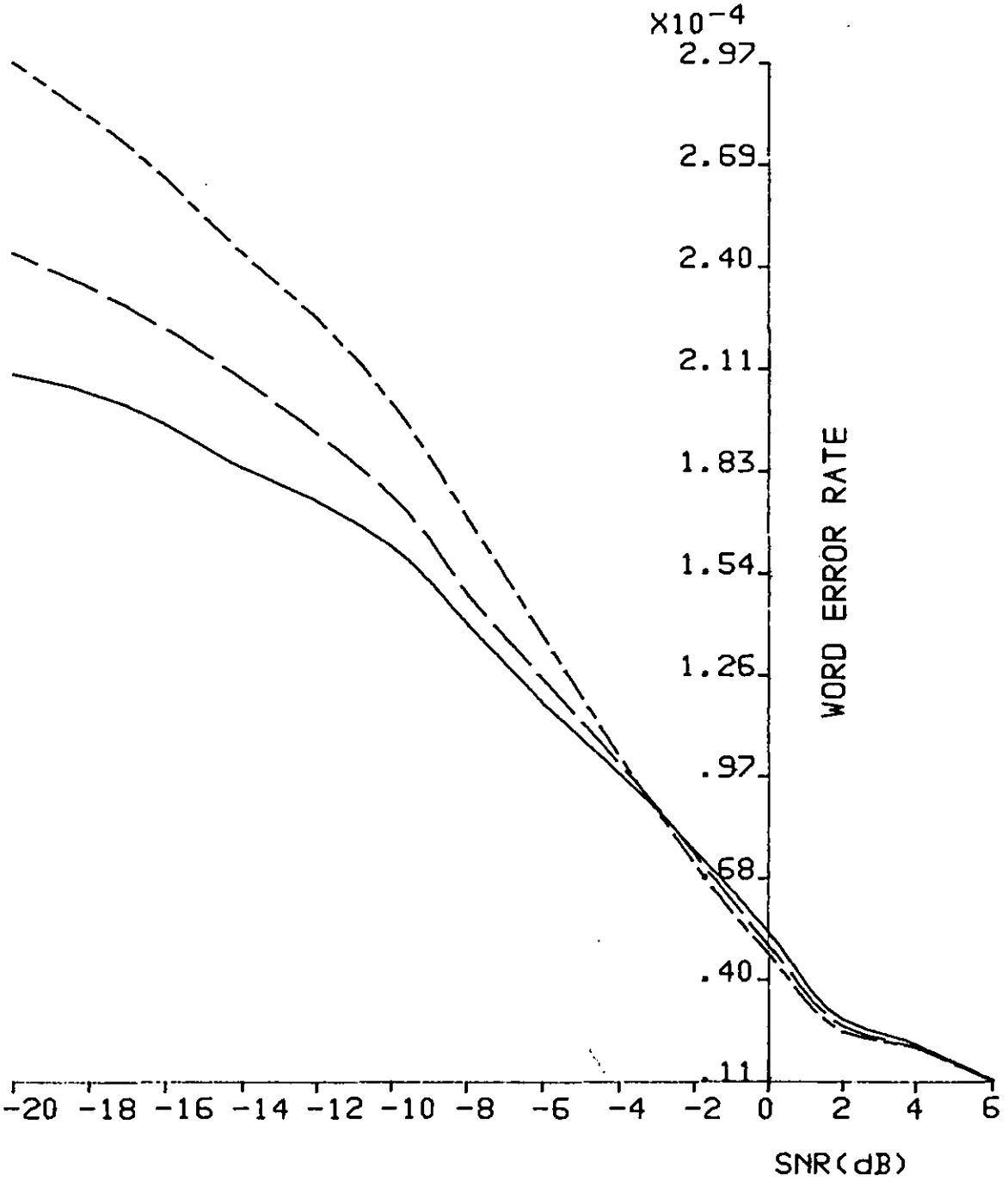
The quantization effects on the second algorithm are very similiar to their effect on the first algorithm, as it can be seen from figs.(4.15) and (4.7). This is somehow expected, because the second algorithm is a



____ 4 EP TESTED ___ 12 P TESTED

__ __ 8 EP TESTED

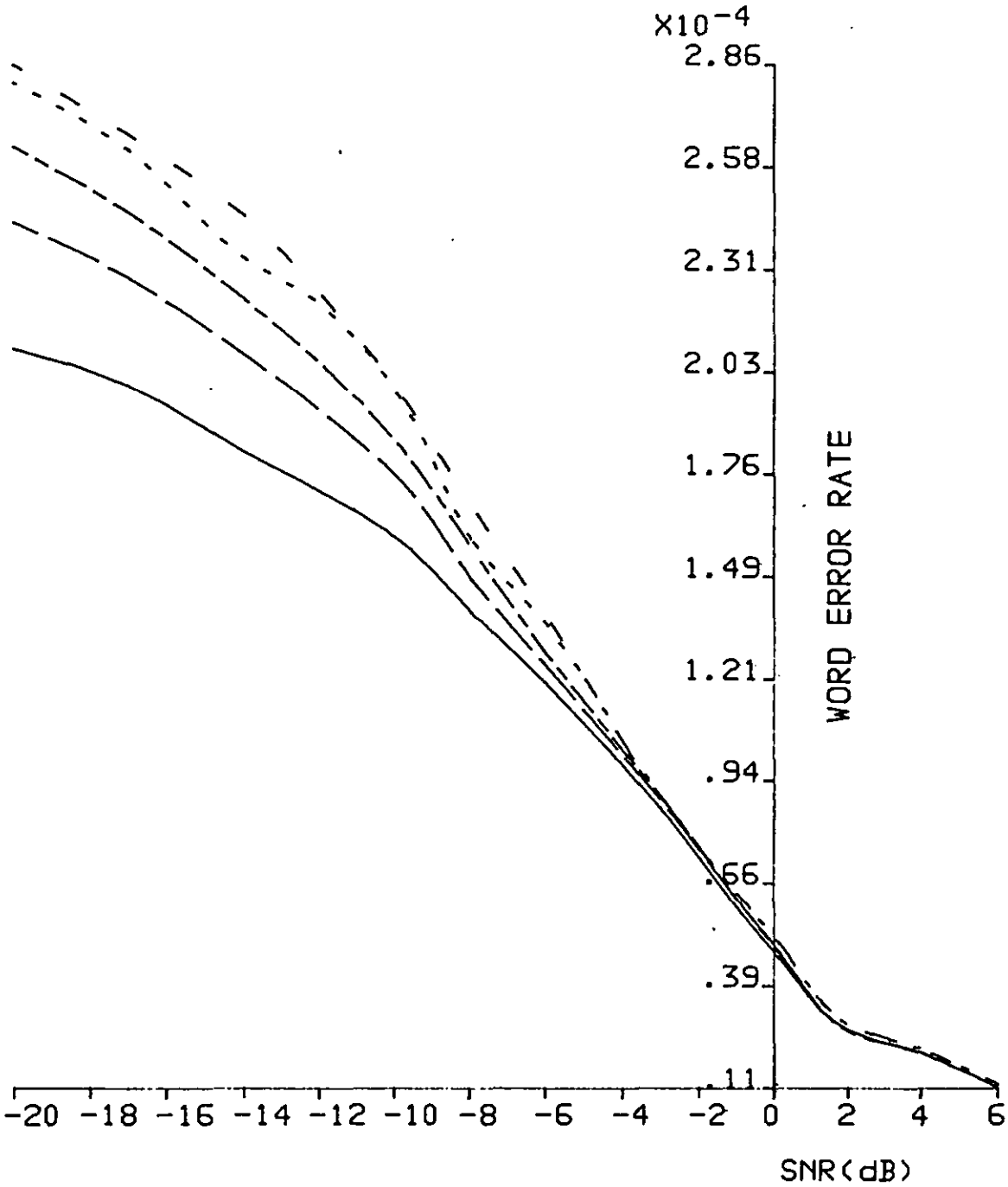
FIG.(4.13) NO. OF ERROR-PATTERNS TESTED EFFECT



____ 4 EP TESTED ___ 12 EP TESTED

__ __ 8 EP TESTED

FIG.(4.14) NO. OF ERROR-PATTERNS TESTED EFFECT



____ 4 LEVELS

___ 8 LEVELS

_ _ _ 16 LEVELS

_ _ _ 32 LEVELS

_ _ 64 LVELS

FIG.(4.15) QUANTISATION EFFECTS

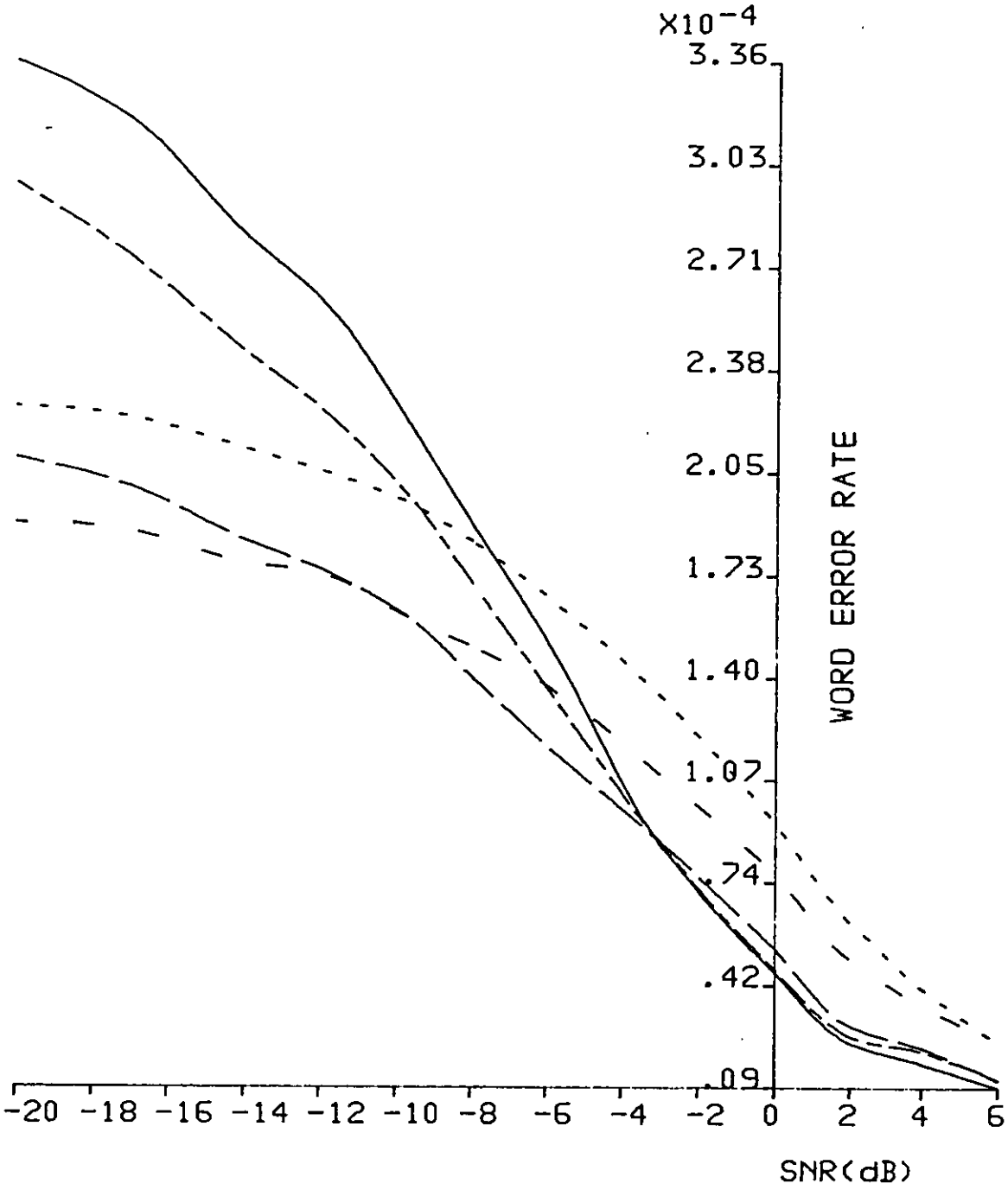
modification of the first one, hence needless to say that the argument for the quantization effect on the first algorithm in section 4.4.1.2 stands also for the quantization effects on the second algorithm.

4.4.2.3 Improvement Over First Algorithm

The second algorithm is designed to perform better than the first algorithm at low SNR values. To verify this, the second algorithm is tested with the background SNR=9 dB. The number of error-patterns tested is taken as 4 and 12 at 16 and 4 quantization levels, and the results are shown in figs.(4.16) and (4.17). The optimum decoder, error-trapping decoder, and the first algorithm decoder results are plotted also for comparsion.

Fig.(4.16) shows the results when the first algorithm and second algorithm decoders,used 16 levels quantizers. For 4 error-patterns, the second algorithm performs better than the first algorithm at burst SNR values lower then -1.9 dB, while its performance is better than the error-trapping decoder for the whole studied range of burst SNR. But worse than the optimum decoder for burst SNR values lower than -11.5 dB. On the other hand the second algorithm decoder using 12 error-patterns performs better than the first algorithm for burst SNR values lower than-1.2 dB. While it performs worse than the optimum decoder and the error-trapping decoder at burst SNR values lower than -6 dB, and -9.5 dB respectively.

Fig.(4.17) shows the same results as fig.(4.16), the only difference here is that the quantization levels



—— AL1 — — 12 EP TESTED — OPT.

— — 4 EP TESTED - - - E.T.

FIG.(4.16) AL2 PERFORMANCE AT 16 LEVELS

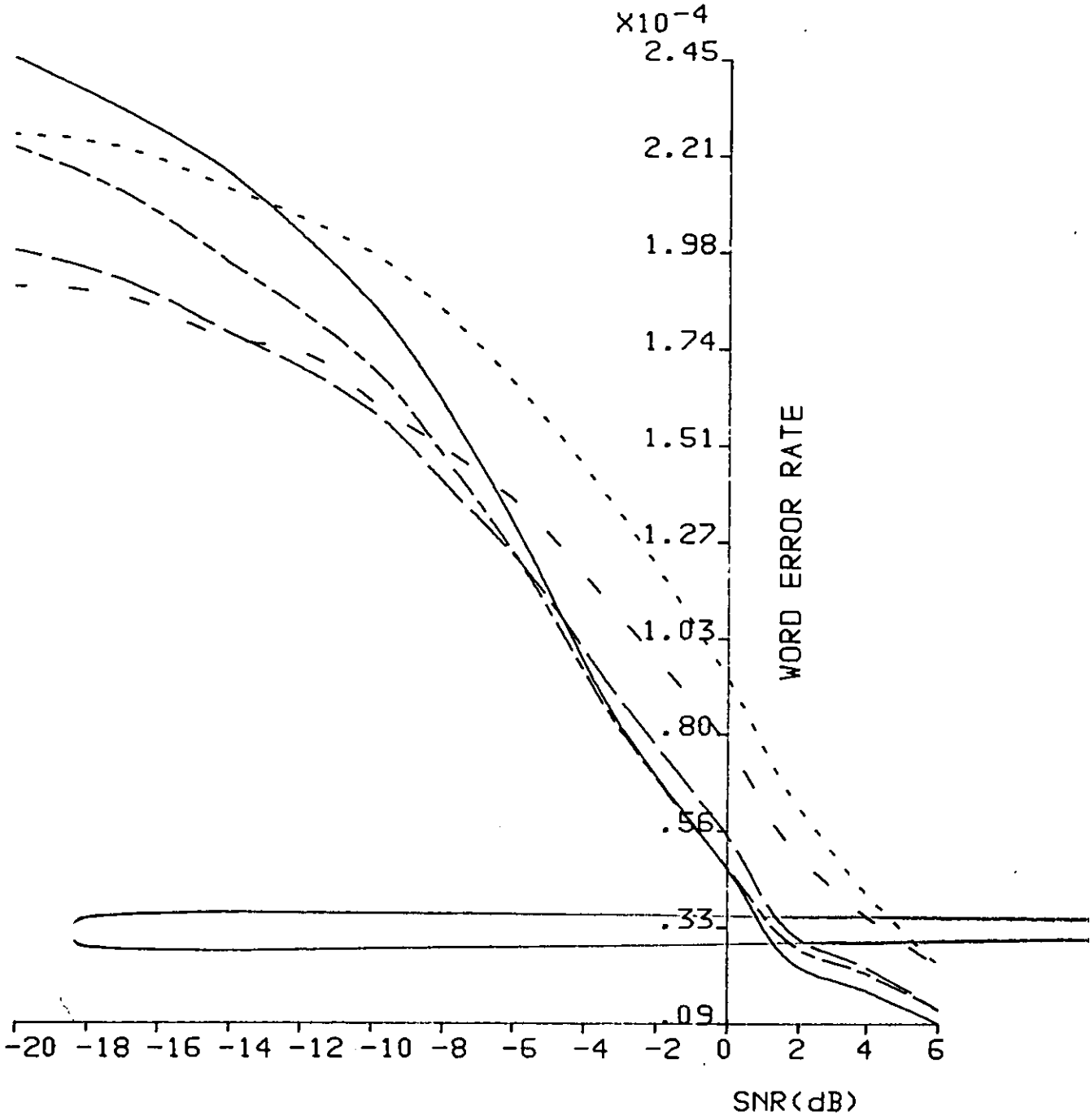


FIG.(4.17) AL2 PERFORMANCE AT 4 LEVELS

are 4. The intersection points for the 4 and 12 error-patterns are -6 dB, and 0 dB respectively, and the intersection points with the optimum decoder results are -13.5 dB, and -8.2 dB for 4 and 12 error-patterns respectively. The second algorithm outperforms the error-trapping decoder for the whole studied range.

The same previous test is repeated for a background SNR=7 dB, the test results are shown in figs.(4.18) and (4.19) for 16 and 4 quantization levels. Here again, as for the first algorithm, the second algorithm performs better at a lower background SNR value. The two figures are very similar to the previous two, the only difference is that the intersection points are shifted to the left.

For the 16 level decoders, fig.(4.18), the intersection points with the first algorithm are -0.5 dB and -4.5 dB for 12 and 4 error-patterns respectively. The intersections with the optimum decoder and the error-trapping decoder for the 12 error-patterns are at -12 dB and -20 dB respectively. While the 4 error-pattern decoder outperforms the optimum and error-trapping decoders for the whole studied range.

The curves of fig.(4.19) are a little bit different, because at 4 quantization levels the second algorithm outperforms the error-trapping decoder, and the optimum decoder over the whole studied range. While the intersection points with first algorithm are -9 dB and -0.8 dB for the 4 and 12 error-patterns respectively.

Clearly figs.(4.16), (4.17), (4.18), and (4.19) show that the second algorithm introduces an improvement over

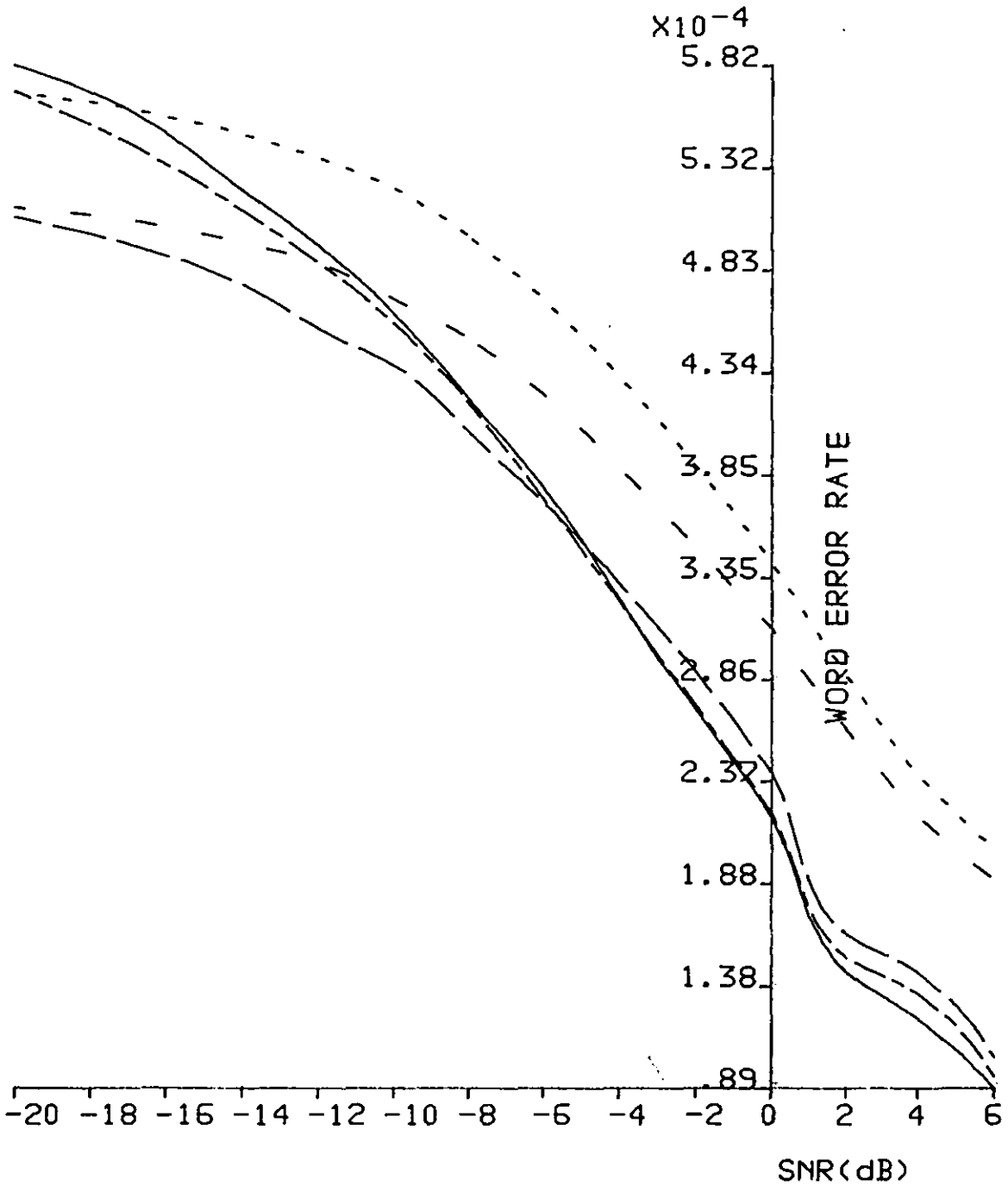


FIG.(4.18) AL2 PERFORMANCE AT 16 LEVELS

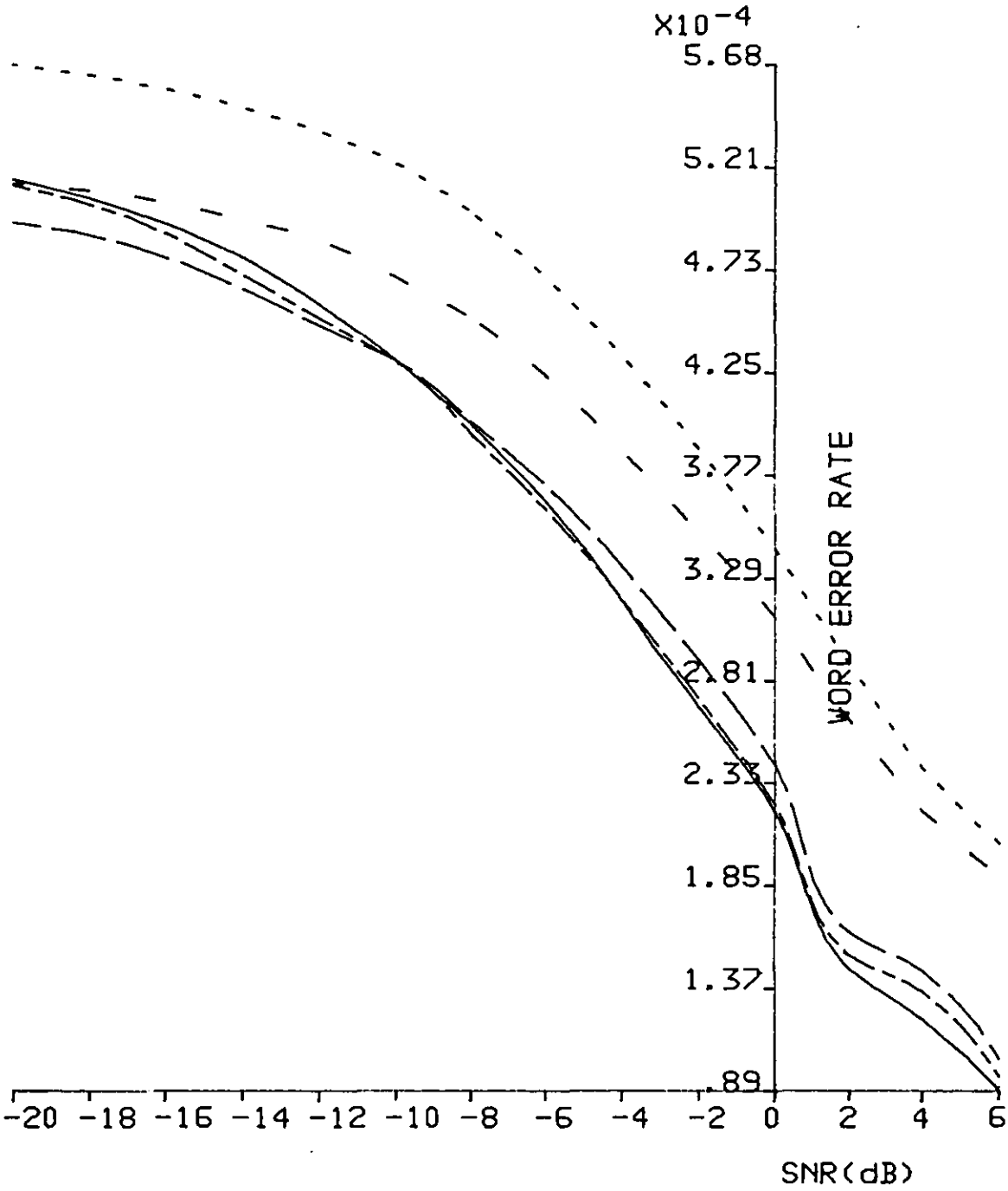


FIG.(4.19) AL2 PERFORMANCE AT 4 LEVELS

the first algorithm at low burst SNR values, but this improvement is at the cost of a small degradation at high burst SNR values. And that the improvement and the degradation are inversely proportional to the number of error-patterns used in the decoding process.

4.5-Conclusions

The time-shared decoding may be the solution for a very high transmission rate system. Any system is evaluated by its complexity, cost, and performance. A time-shared decoder will introduce an additional complexity to the receiver, yet the microporcessor software can be made to simplify the system, i.e. use the microprocessor to handle the buffer control. The cost is also higher because at least the system requires additional storage to be used as buffers. Again some systems may require buffers for usage by other than the error-correcting decoder, hence the existing buffers may be used by the time-shared decoder. Consequently, the time-shared decoder complexity and cost is somehow difficult to evaluate; because it can be different from one system to another. While the time-shared decoder performance is entirely dependent on the decoding algorithm, two algorithms were introduced to be used specifically as time-shared decoding algorithms, although in general any decoding algorithm can be used.

Perhaps the most important conclusion one can draw is the effect of the number of quantization levels on the soft-decision decoder performance. It became clear during the simulation that the decoder performance is not always

improved by increasing the number of quantization levels. In fact the simulation results show that the decoder performance is improved as the number of quantization levels is increased at high burst SNR. But after a certain low SNR value the decoder performance starts to deteriorate as the number of quantization levels is further increased. This represents a special problem because if a small number of quantization levels are used, the decoder will preform well at low burst SNR values, but will not get all the improvement that can be achieved. On the other hand, if a larger number of quantization levels is used, the decoder will perform well at high burst SNR values, and badly at low burst SNR values. In general it is perferable to use larger number of quantization levels, so that the overall decoder performance is good, because in the bursty channel case, the SNR value is high most of the transmission time.

The first algorithm is in fact an ordinary soft-decision algorithm, with the exception that the parity-check error-trapping is added so that it will reduce the execution time, hence reducing delay when used in the time-shared decoder. Although the algorithm is designed to be used in time-shared decoders, there is nothing to stop using it in a conventional soft-decision decoder. The parity-check error-trapping technique can be incorporated in the conventional soft-decision decoder to achieve an additional gain at low burst SNR values. In general the inclusion of the parity-check error-trapper in the decoder can be achieved without additional cost to the hardware

or major modifications in the software.

The second algorithm is a further modification of the first algorithm, it performs better than the first algorithm at low burst SNR values, while the first algorithm outperforms the second algorithm at high burst SNR values. Again the second algorithm can be used in conventional soft-decision decoders with or without the parity-check error-trapping facility. The most important feature of the second algorithm is its change of perfromance as the number of tested error-patterns are changed. When the number of tested error-patterns is large, the second algorithm performs exactly as the first algorithm. If the number of tested error-patterns is equal to one, a second algorithm decoder will perform exactly as the optimum decoder. While if the number of tested error-patterns is in between then, the second algorithm, depending on the SNR values, may perform better than the first algorithm and the optimum decoder.

As it is mentioned earlier, the first algorithm performs better at high burst SNR, while the second algorithm performs better at low burst SNR. Since the second algorithm can be made to perform as the first algorithm by increasing the number of the tested error-patterns. Then an adaptive system that change the number of the tested error-patterns according to the burst SNR value, can be used to achieve a better performance over the whole burst SNR range. Clearly such a system will require a burst SNR calculation circuit, which may be complicated in itself. A simple solution can be based

on the following fact; keeping in mind that the transmitted codewords are interlaced. At low burst SNR values, a burst will cause a number of consecutive received words to be erroneous. As soon as the burst is finished, there should be no consecutive, but random erroneous received words. Instead of the burst SNR calculation circuit, a counter is designated to count the consecutive erroneous words, if the count is larger than certain number a burst is assumed, and the number of tested error-patterns is reduced to a low value, while once an error-free word is received, it is assumed that the burst is finished, and the number of tested error-patterns is increased. Although this solution may not be very elegant, it is reasonably accurate and very simple to implement.

Both algorithms show that for bursty channels the improvement over error-trapping and optimum decoding is increased as the background SNR is decreased. But it is expected that the improvement will start to decrease after reaching a peak value as the background SNR is decreased.

CHAPTER 5.

THE PARALLEL THRESHOLD DECODER

5.1- A General Look

A data stream is transmitted over a transmission channel fig.(3.1), which can handle binary signals. Consider the transmission of a data block from the data stream. Fig.(5.1) shows the different waveforms of a block in various points of the transmission system, the modulator output $x(t)$ is shown in fig.(5.1-a) which represents the transmitted waveform during transmission, noise waveform $w(t)$ is added to the transmitted waveform, to form the received waveform $y(t)$ according to Eq.(3.1)

$$y(t) = x(t) + w(t) \qquad (3.1)$$

The received waveform at the demodulator input $y(t)$ is shown in fig.(5.1-b). At the demodulator the received signal is sampled in time with the sampling pulses fig.(5.1-c), and compared with a present threshold value to detect whether the transmitted digit was '0' or '1'. The output of the demodulator is dependent on the threshold value which in most communication systems is half point between the transmitted values of '0' and '1'. The output of a detector which has a threshold value of zero is shown in fig.(5.1-d), and it can be seen from comparing fig.(5.1-d), and fig.(5.1-a) that the demodulator committed three errors in detecting the 2nd, 6th, and 12th

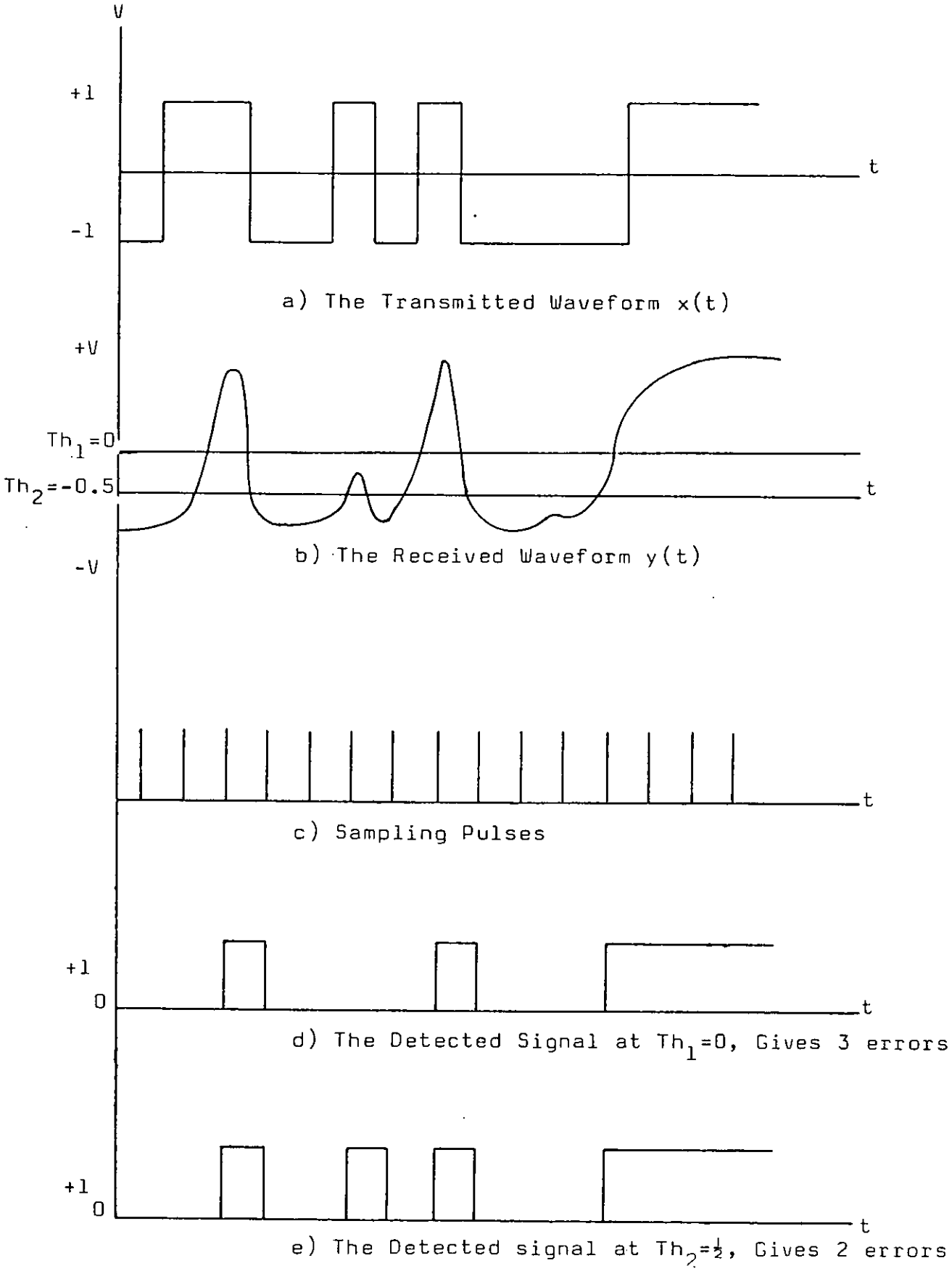


Fig.(5.1) Transmitted, received and detected waveforms

received digits. Let the threshold value be any value other than zero volts, say -V/2 volts. The detector at the demodulator will compare the sampled received waveform with the -V/2 volts threshold, it will assume that the corresponding transmitted digit was '1' if its sampled received waveform value is higher than -V/2, and that the transmitted digit was '0' if the sampled value was lower than -V/2 volts. The output of a detector which has a threshold value of -V/2 volts is shown in fig.(5.1-e), it can be seen from comparing fig.(5.1-e), and fig.(5.1-a), that the demodulator committed two errors in detecting the 2nd, and the 12th received digits. If an error-correcting code is used for error-control in this transmission system, which is capable of correcting two or less errors in the considered block. Then the block decoded by the zero threshold may or may not be corrected, while the block detected by the -V/2 volt threshold will be decoded successfully, and the correct data will be delivered to the sink. Clearly in this particular case the second threshold value is more suitable for the decoding of this particular received waveform than the first threshold value, and this may not be the case if the same block is transmitted again.

5.2-The Optimum Threshold Calculation

The optimum threshold is the value that when used to detect the received signal the detection is optimum. The detection process is optimum in the sense that it minimizes the probability of error in the detection of

the received signal.

It is assumed that no signal distortion is introduced by the transmission channel, so that the signal waveform is shaped entirely at the transmitter and the receiver. The transmission channel, however, introduces additive white Gaussian noise. Although the additive noise introduced by many practical channels does not approximate to Gaussian noise, it is well known[16,17] that a digital signal having a better tolerance to additive white Gaussian noise than another signal, will normally also have a better tolerance to the additive noise obtained in practice. The relative tolerance of different signals to additives white Gaussian noise are therefore usually a good measure of their relative tolerance to additive noise present in a practical channel,

Going back to fig.(3.1) the signal to be transmitted through the channel is generated in the source, the source encoder will change this signal to a stream of '0' and '1's. The channel encoder groups each k digit together and annexes n-k parity check digits to each group, so that a block of n digits is formed. Each block is an independent vector of any other block, and it is a codeword. Let this vector be C, then

$$C = c_1, c_2, c_3, \ldots\ldots, c_n \qquad (5.1)$$

where $c_1,\ldots,c_{(n-k)}$ are the parity check digits, and $c_{(n-k+1)}$, $c_{(n-k+2)}$,.....,c_n are the information digits.

At this stage, since we are interested in the sampled output of the transmission channel. Then a discrete channel will be considered, where discrete noise will be added to the transmitted signal in the channel.

The vector C is converted to the binary polar vector X, and is transmitted through the channel as a two level signal where

$$X = x_1,x_2,x_3,\ldots\ldots,x_n \qquad (5.2)$$

each sample x_i is one of two levels depending on the value of c_i

when c_i = '1' then $x_i = +V$ (5.3-a)

and when c_i = '0' then $x_i = -V$ (5.3-b)

During the transmission through the channel, white Gaussian noise, which has zero mean, and power spectral density of σ^2, is added to the transmitted signal. The received signal is the addition of the transmitted signal and the noise signal. Let the noise vector added to the transmitted vector be W, then

$$W = w_1,w_2,w_3,\ldots\ldots,w_n \qquad (5.4)$$

where w_i is a sample value of a Gaussian process, with zero mean and σ variance .

At the receiver, the received signal is the sampled vector Y, where

$$Y = y_1, y_2, y_3, \ldots\ldots, y_n \tag{5.5}$$

The received vector Y can be written as the addition of the transmitted vector and noise -

$$Y = X + W \tag{5.6}$$

$$y_i = x_i + w_i \tag{5.7}$$

To detect the value of x_i from y_i, y_i is compared with a threshold level of Th,

when $y_i < Th$ x_i is detected as -V (5.8-a)

when $y_i > Th$ x_i is detected as +V (5.8-b)

and when $y_i = Th$ x_i may be detected as +V or-V (5.8-c)

The probability density function of the noise component w_i in Eq.(5.4) is

$$P(w_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) \tag{5.9}$$

The probability density function of the received signal can be obtained from Eqs.(5.9) and (5.7)

$$p(y_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(y_i-x_i)^2}{2\sigma^2}\} \qquad (5.10)$$

Thus y has two possible probability-density functions $p(y_i|_{x_i=-V})$ and $p(y_i|_{x_i=+V})$, its probability density depending upon whether x=-V or x=+V. $p(y_i|_{x_i=-V})$ and $p(y_i|_{x_i=+V})$ are conditional probability densities of y, given that x=-V and x=+V respectively. The two conditional probability densities are shown in fig.(5.2).

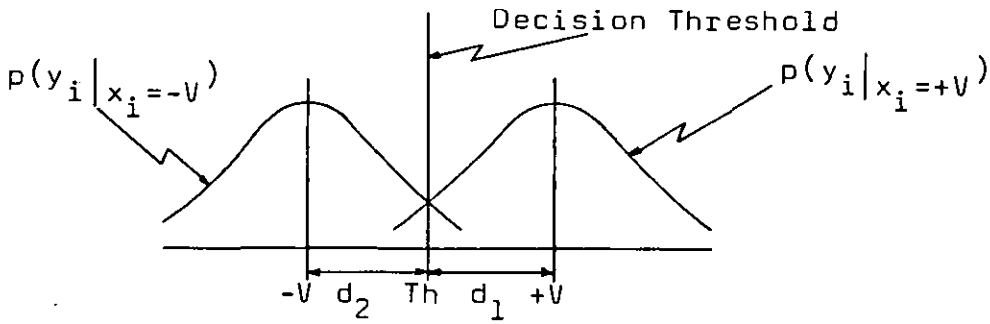


Fig.(5.2) Conditional Probability Density Functions of y

Clearly, the two conditional probability densities can be obtained by substituting the values x_i in Eq.(5.10).

$$p(y_i \mid x_i=-V) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(y_i+V)^2}{2\sigma^2}\} \qquad (5.11)$$

and

$$p(y_i \mid x_i=+V) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(y_i-V)^2}{2\sigma^2}\} \qquad (5.12)$$

Assuming the decision threshold used in the detection process it Th, and the detection rules are the same as Eqs.(5.8). If $x_i=-V$, an error occurs in the detection of x_i when $y_i>Th$, thus the probability of error is

$$p_{e0} = \int_{Th}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(y_i+V)^2}{2\sigma^2}\}\, dy_i \qquad (5.13)$$

If $x_i=+V$, an error occurs in the detection of x_i when $y_i<Th$, thus the probability of error is

$$p_{e1} = \int_{Th}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-\frac{(y_i-V)^2}{2\sigma^2}\}\, dy_i \qquad (5.14)$$

assuming that x_i is transmitted as -V, from Eq.(5.7)

$$w_i = y_i + V \qquad (5.15)$$

Then

$$dw_i = dy_i \qquad (5.16)$$

again assuming that x_i is transmitted as +V, from Eq.(5.7)

$$w_i = y_i - V \tag{5.17}$$

Then

$$dw_i = dy_i \tag{5.18}$$

Substituting Eqs.(5.15) and (5.16) in Eq.(5.13) gives

$$p_{e0} = \int_{Th}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) dw_i \tag{5.19}$$

Similarily substituting Eqs.(5.17) and (5.18) in Eq.(5.14) gives

$$p_{e1} = \int_{-\infty}^{Th} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) dw_i \tag{5.20}$$

Assuming the decision threshold is at a distance d_1 from +V, and d_2 from -V, then from fig.(5.2)

$$d_1 + d_2 = 2V \tag{5.21}$$

Thus if x_i=-V, an error occurs in the detection of x_i whenever $w_i > d_2$. Similarily, if x_i=+V, an error occurs in the detection of x_i whenever $w_i < d_1$. Hence p_{e0} is the probability that the noise component w_i has a value more postive than d_1, and p_{e1} is the probability that w_i has a value more negative than d_2. It follows

that

$$P_{e0} = \int_{d_2}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) \, dw_i \tag{5.22}$$

and

$$P_{e1} = \int_{-\infty}^{d_1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) \, dw_i \tag{5.23}$$

Let the probability of transmitting '1' be q_1, and the probability of transmitting '0' be q_0. Then

$$q_1 + q_0 = 1 \tag{5.24}$$

Consider the transmission of a digit without any knowledge of this digit being '0' or '1'. The probability of decoding this digit erroneously as '1' is

$$P_{e0} = q_0 \, P_{e0} \tag{5.25}$$

$$P_{e0} = q_0 \int_{d_2}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) \, dw_i \tag{5.26}$$

$$P_{e0} = q_0 \int_{\frac{d_2}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{w_i^2}{2}\right) \, dw_i \tag{5.27}$$

$$P_{e0} = q_0 \, \mathrm{erf}\left(\frac{d_2}{\sigma}\right) \tag{5.28}$$

where

$$\mathrm{erf}(y) = \int_{y}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{z^2}{2}\right) dz \qquad (5.29)$$

Conversley, the probability of decoding the transmitted digit erroneously as '0' is

$$P_{e1} = q_1 \, P_{e1} \qquad (5.30)$$

$$P_{e1} = q_1 \int_{-\infty}^{d_1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) dw_i \qquad (5.31)$$

Since $\exp(-\frac{w_i^2}{2\sigma^2})$ is an even function of w, Eq.(5.31) can be rewritten

$$P_{e1} = q_1 \int_{-d_1}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) dw_i \qquad (5.32)$$

$$P_{e1} = q_1 \int_{\frac{-d_1}{\sigma}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{w_i^2}{2\sigma^2}\right) dw_i \qquad (5.33)$$

$$P_{e1} = q_1 \, \mathrm{erf}\left(\frac{-d_1}{\sigma}\right) \qquad (5.34)$$

where erf(y) is defined as in Eq.(5.29).

The probability of decoding a transmitted digit erroneously is

$$P_e = P_{e0} + P_{ei} \qquad (5.35)$$

Substituting Eqs.(5.28) and (5.34) in Eq.(5.35) gives

$$P_e = q_0 \text{ erf } (\frac{d_2}{\sigma}) + q_1 \text{erf}(\frac{-d_1}{\sigma}) \qquad (5.36)$$

To find the distance d_1, that minimizes the probability of error is the received digit the following condition should be satified

$$\frac{\partial P_e}{\partial d_1} = 0 \qquad (5.37)$$

from Eq.(5.21)

$$d_2 = 2V - d_1 \qquad (5.38)$$

Substituting Eq.(5.38) in Eq.(5.26) gives

$$P_{el} = q_0 \int_{2V-d_1}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\ (-\frac{w_i^2}{2\sigma^2}) \quad dw_i \qquad (5.39)$$

from Eqs.(5.35),(5.31), and (5.39)

$$P_e = \frac{1}{\sqrt{2\pi\sigma^2}} \left\{ q_1 \int_{-\infty}^{d_1} \exp(-\frac{w_i^2}{2\sigma^2}) dw_i + q_o \int_{2V-d_1}^{\infty} \exp(-\frac{w_i^2}{2\sigma^2}) dw_i \right\} \qquad (5.40)$$

according to the first fundamental theorm of culculus[4].

$$\frac{\partial P_e}{\partial d_1} = \frac{1}{\sqrt{2\pi\sigma^2}} \left\{ q_1 \exp(-\frac{d_1^2}{2\sigma^2}) - q_o \exp\{-\frac{(2V-d_1)^2}{2\sigma^2}\} \right\} \qquad (5.41)$$

to meet the condition of Eq.(5.37)

$$q_1 \exp\left(-\frac{d_1^2}{2\sigma^2}\right) - q_0 \exp\left\{-\frac{(2V-d_1)^2}{2\sigma^2}\right\} = 0 \qquad (5.42)$$

$$\frac{q_1}{q_0} = \frac{\exp\left\{-\frac{(2V-d_1)^2}{\sigma^2}\right\}}{\exp\left\{-\frac{d_1^2}{2\sigma^2}\right\}} \qquad (5.43)$$

$$\frac{q_1}{q_0} = \exp \frac{1}{2\sigma^2}\left\{(-4V^2+4Vd_1-d_1^2) - (-d_1^2)\right\} \qquad (5.44)$$

$$\log \frac{q_1}{q_0} = \frac{1}{2\sigma^2}(4Vd_1-4V^2) \qquad (5.45)$$

$$\sigma^2 \log \frac{q_1}{q_0} = 2V(d_1-V) \qquad (5.46)$$

$$d_1 = V + \frac{\sigma^2}{2V} \log \frac{q_1}{q_0} \qquad (5.47)$$

Similarily d_2 can be calculated in the same way, to give

$$d_2 = V + \frac{\sigma^2}{2V} \log \frac{q_0}{q_1} \qquad (5.48)$$

or directly from Eq.(5.21).

It is clear from Eqs.(5.47) and (5.48), that the threshold distance from +V or -V (the threshold value), is dependent on the transmitted signal voltage $\pm V$, the noise power σ^2, and the ratio $\frac{q_1}{q_0}$.

The mean power level of the transmitted signal is

$$S = \frac{(+V)^2-(-V)^2}{2} = V^2 \qquad (5.49)$$

The noise power is

$$N = \sigma^2 \qquad (5.50)$$

The signal to noise ratio SNR is

$$SNR = \frac{S}{N} = \frac{V^2}{\sigma^2} \qquad (5.51)$$

Substituting Eq.(5.51) in Eq.(5.47) gives

$$d_1 = V(1+ \frac{1}{2SNR} \log \frac{q_1}{q_0}) \qquad (5.52)$$

which shows that the optimum distance is dependent on the SNR and the ratio $\frac{q_1}{q_0}$.

5.3-The Optimum Threshold of a Continued Transmission

In section (5.2) the optimum threshold value that minimizes the probability of error in each received digit is derived. During a continuous transmission the same equations can be used to minimize the probability of error for each individual received digit. It is wise to assume that the source is an unbiased source, thus each generated digit is equally likely to be either '0' or '1'. Solving Eq.(5.24) for this assumption gives

$$q_0 = q_1 = \frac{1}{2} \tag{5.53}$$

Substituting q_0 and q_1 values in Eqs.(5.47) and (5.48) yields

$$d_1 = d_2 = V \tag{5.54}$$

The decision threshold value for such a system is placed at half-distance between the +V and -V, which is in this case zero volts.

The assumption of an unbiased source is a wise one, because the end result, Eq.(5.54), is independent of the noise value. Consequently, this threshold value is fixed and always optimum regardless of the SNR value in the transmission channel. While the use of any other values for q_0 and q_1 (except 0, and 1) will result of a different threshold value for each SNR value.

5.4-The Optimum Threshold for a Limited Length Block

The term continuous transmission used in section (5.3) implies that a large number of digits are transmitted during each communication process, where the number of '0' and '1' is nearly equal. Unfortunatly, this is not the case when a limited digits number is considered. Since our main interest is block codes, and that each block is independent of the other transmitted blocks. The optimum threshold will be calculated for the received block. The optimum threshold is the value that when used to detect the received block, the detection is optimum.

The detection process is optimum in the sense that it minimizes the probability of error in the detection of the received block.

According to Eqs.(5.47) and (5.48), the optimum threshold value is dependent on q_0 and q_1. To find these values, consider that an (n,k) block-error-correcting code is used for error correction in a channel, the information digits stream is divided into independent groups of k digits in the transmitter. Since the source is generating the '1' and '0' stream according to the information to be transmitted, and each group is independent of the others, then the number of '1's and '0's is different for each group. Furthermore the annexation of the parity-check digits to these groups, will cause a change in the number of '1' and '0' in each block, although the '0's to '1' ratio may or may not change. The relation between the number of '1' NOO, i.e. the Hamming weight and number of '0' NOZ, in each block of length n is

$$NOO + NOZ = n \tag{5.55}$$

the probabilities q_0 and q_1 for th i^{th} block are

$$q_{0i} = NOZ_{i/n} \tag{5.56}$$

and

$$q_{1i} = NOO_{i/n} \tag{5.57}$$

The length of each block is n digits, where n is dependent on the block code used, which is relatively small number compared with the number of digits in a continuous transmission. Consequently, Eq.(5.53) does not apply generally to all blocks, thus, threshold value calculated in Eq.(5.54) is no longer the optimum threshold that minimizes the probability of error in all the received blocks.

The channel input is always a codeword whatever the information digits are, thus the probabilities q_0 and q_1 can take only the values corresponding to the probabilities of the 2^k codewords. Fig.(5.3) shows the frequency distribution of the '0' or '1' in all the possible codewords for the BCH(31,21) cyclic-block code. It can be seen that the 2097152 NOO or NOZ values form a group of 23 values. Fig.(5.3) is plotted by using the NOO values. The NOZ will give a similar figure but rotated about n/2 axis. By scaling down the x axis by n in fig.(5.3), the same figure will represent the frequency of q_0 and q_1. Fig.(5.3) shows the NOO distribution for the BCH(31,21) code, which is typical of the codes used as it can be seen from fig.(5.4) which is the distribution of the (34,22) burst code, fig.(5.5) the (27,20) burst code, fig.(5.6) the (27,17) burst code, and fig.(5.7) the (19,11) burst code. The only exception is the distribution of the BCH(15,7) random-error-correcting code fig.(5.8), which differs in that it has its maximum value at a point other than n/2. It is clear

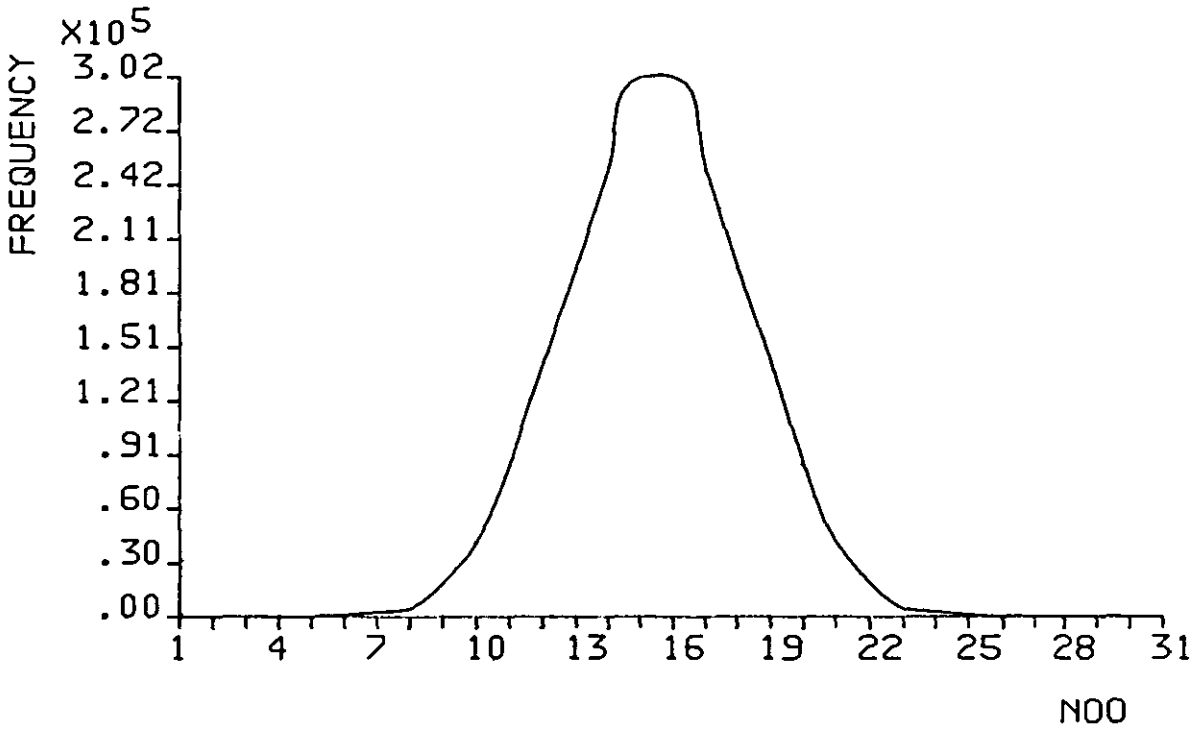


FIG.(5.3) NOO DISTRIBUTION FOR THE (31,21) CODE

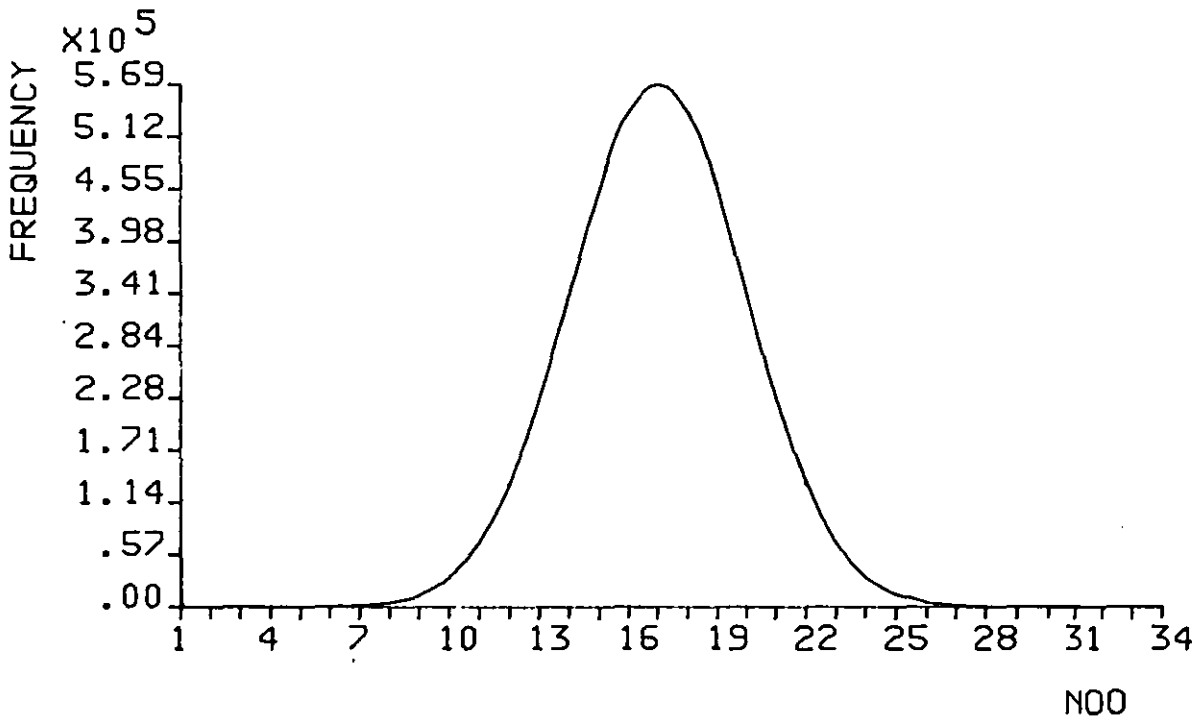


FIG.(5.4) NOO DISTRIBUTION FOR THE (34,22) CODE

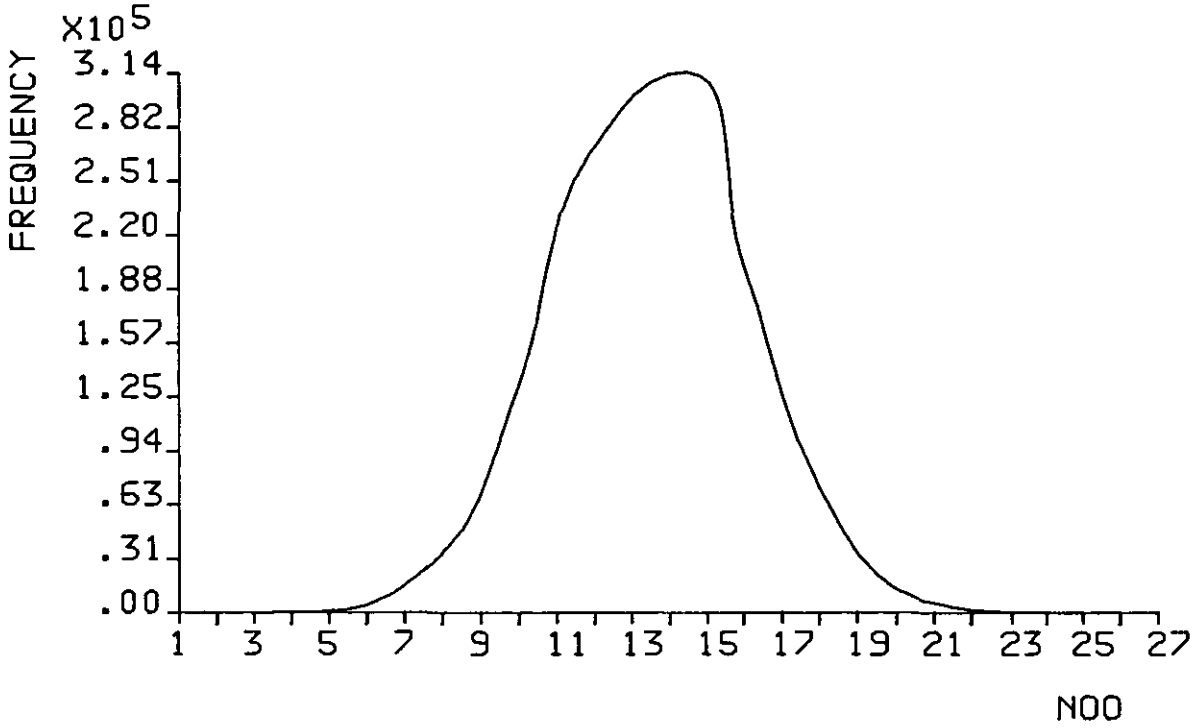


FIG.(5.5) NOO DISTRIBUTION FOR THE (27,20) CODE

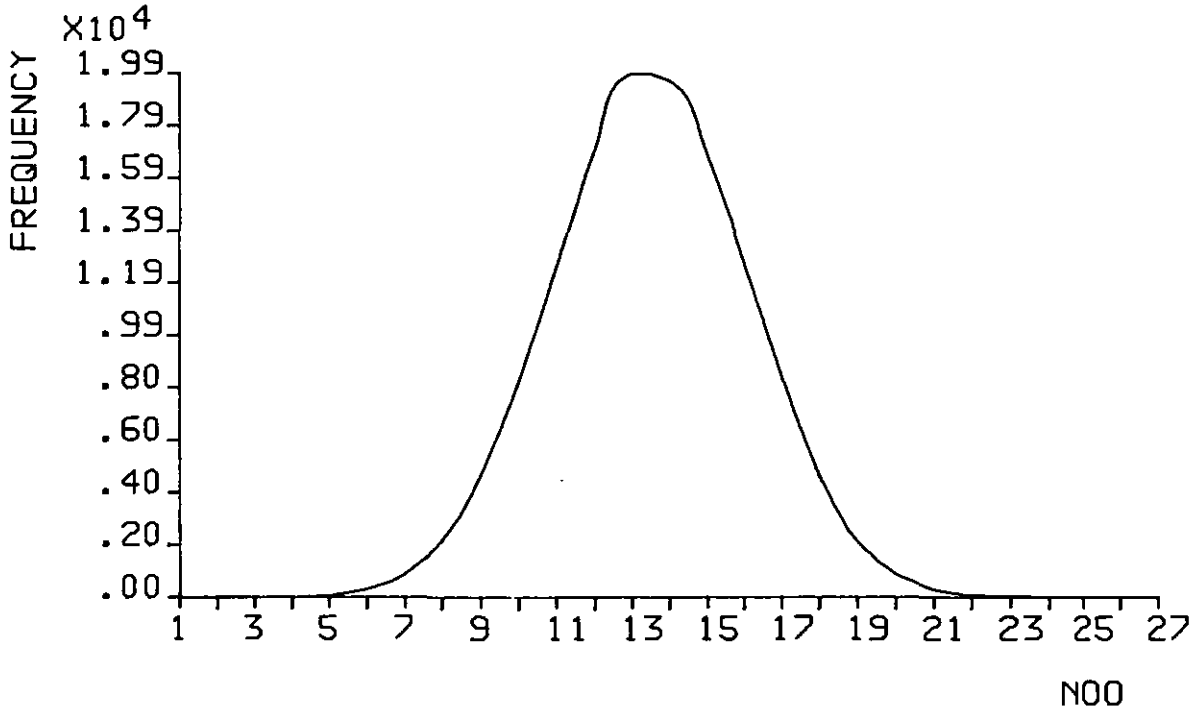


FIG.(5.6) NOO DISTRIBUTION FOR THE (27,17) CODE

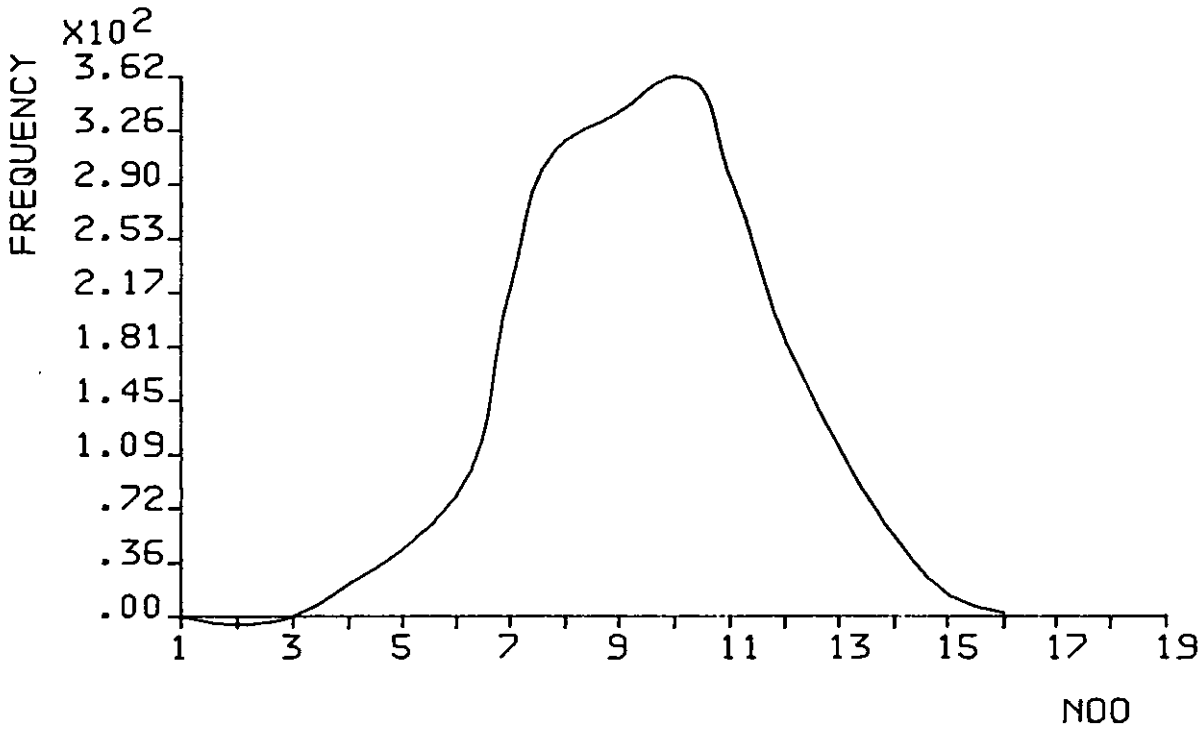


FIG.(5.7) NOO DISTRIBUTION OF THE (19,11) CODE

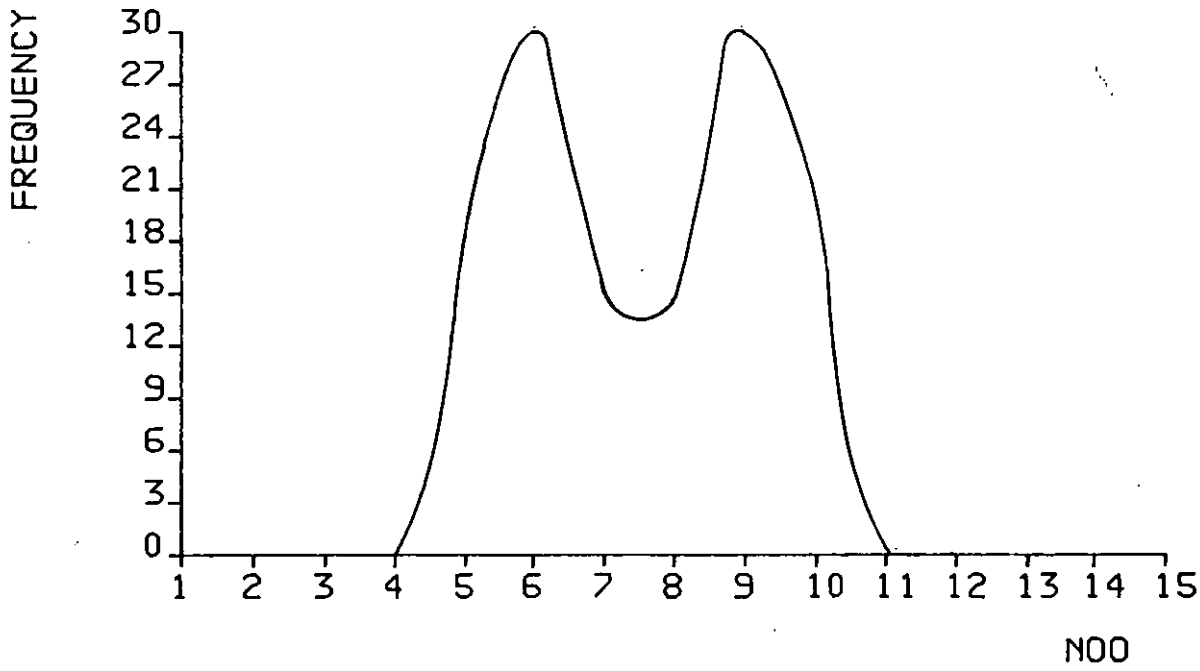
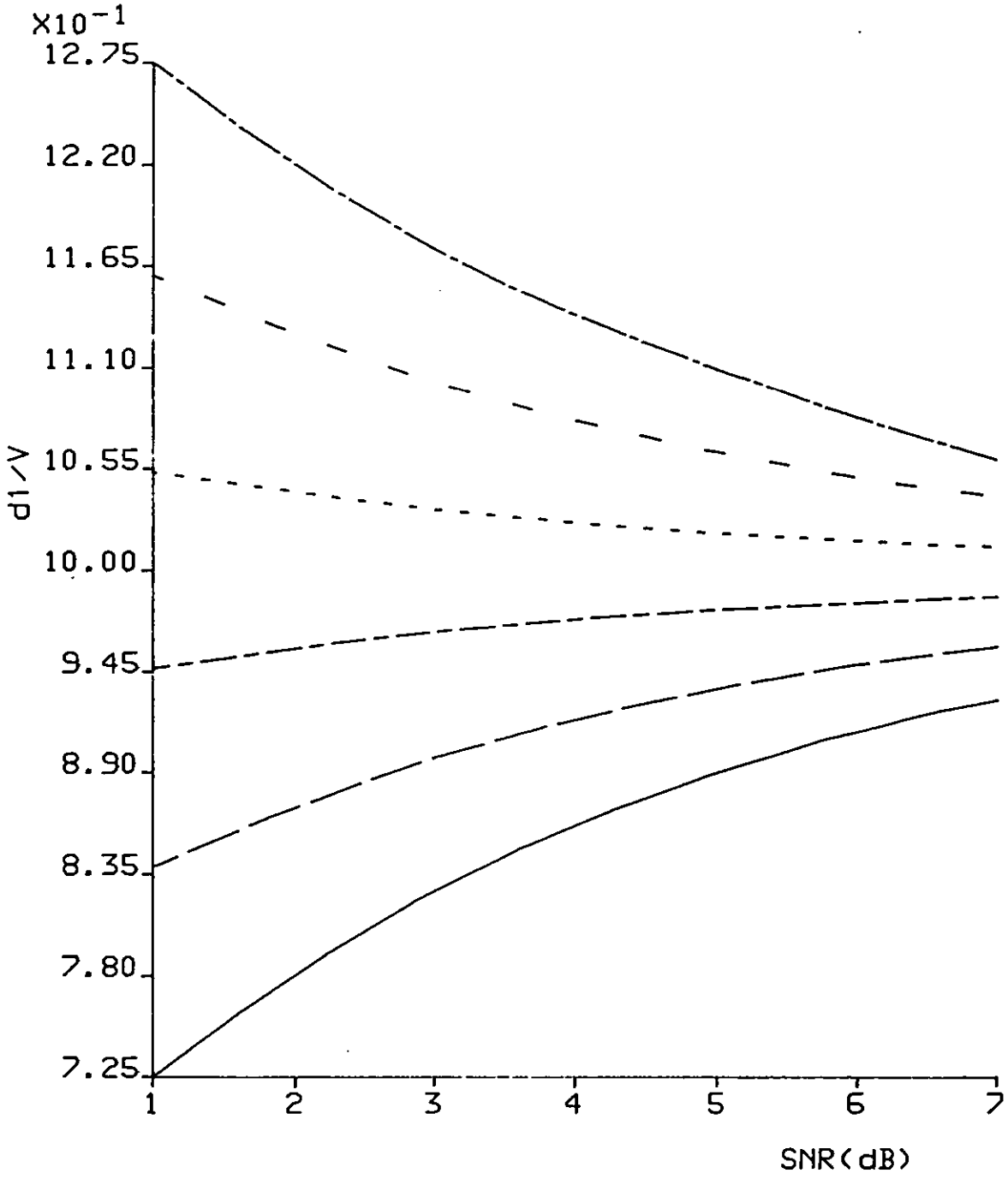


FIG.(5.8) NOO DISTRIBUTION FOR THE (15,7) CODE

from figs.(5.3) to (5.8), that the ratio $\frac{q_0}{q_1}$ is variable from one code to another, furthermore it is variable in the same code from one codeword to another as it is expected.

The other variable that affects the threshold value in addition to the ratio $\frac{q_0}{q_1}$ is the noise power according to Eqs.(5.47) and (5.48), or the SNR as in Eq.(5.52). This variable is channel dependent, and it varies randomly throughout the transmission period. In some cases it can be assumed constant, as in the random error generating channel, where errors are generated randomly (a memoryless channel), but in other cases such as the burst error generating channel (the channel has a memory), its value cannot be predicted, and is not constant throughout the transmission periods. To study the variation of the optimum threshold values with the variation of the noise power of the SNR, the BCH(15,7) random error correcting code was chosen for the study, this choice was made because this code has the smallest number of codewords so, they all can be plotted clearly on the same graph. The optimum threshold value is calculated for each codeword for every SNR value, while the SNR value is increased from 1dB to 7 dB in one dB steps. The study result is plotted in fig.(5.9), the distance $\frac{d_1}{V}$ of Eq.(5.52) represents the optimum threshold values, while the different values of the ratio $\frac{q_1}{q_0}$ represent the different codewords. The all zeros and all ones codewords were left out of the study because d_2 for the



q1/q0=0.5 q1/q0=0.875 q1/q0=1.5

q1/q0=0.667 q1/q0=1.143 q1/q0=2.0

FIG.(5.9) VARIATION OF OPTIMUM THRSHOLD VALUES

all zeros codeword according to Eq.(5.48) is

$$d_2 = V + \frac{\sigma^2}{2V} \log \frac{1}{0} \tag{5.58}$$

$$d_2 = V + \frac{\sigma^2}{2V} (\log 1 - \log 0) \tag{5.59}$$

$$d_2 = - \infty \tag{5.60}$$

which gives an optimum threshold value of $-\infty$ volts. Similarily, from Eq.(5.47) d_1 for all ones codeword is

$$d_1 = - \infty \tag{5.61}$$

which gives an optimum threshold value of $+\infty$ volts. Eqs.(5.60) and (5.61) show that for these particular codewords the optimum threshold is independent of the noise power.

Fig.(5.9) shows that the optimum threshold values are symmetrical around the $\frac{d_1}{V}=1$ value which is the value of d_1 for the ratio $\frac{q_1}{q_0}=1$, and that the threshold values converge to $\frac{d_1}{V}=1$ as the SNR value grows bigger. But the most important result is that there is no fixed value for the optimum threshold of any codeword as the SNR value is varied, apart from the three special values where

$$q_0 = 1 \tag{5.62}$$

or

$$q_1 = 1 \tag{5.63}$$

or

$$q_0 = q_1 = \frac{1}{2} \tag{5.64}$$

It is clear by now that to minimize the probability of error in a received block a variable decision threshold level should be used, where this decision level is dependent on the particular codeword transmitted and on the noise present at the channel during transmission.

5.5-The Variable Optimum Threshold Decoder

The use of Eq.(5.47) and Eq.(5.48) in an actual decoder requires a special arrangements so that the decoder will get all the necessary information about the two variables σ and q_0/q_1, in fact, the knowledge of q_0 or q_1 is sufficient to calculate the ratios q_0/q_1 or q_1/q_0, by solving Eqs.(5.55),(5.56), and (5.57), q_0 or q_1 can be calculated, where

$$q_0 + q_1 = 1 \tag{5.65}$$

Assuming that it is required to build a decoder that will minimize the probability of error in the received block, such a decoder should have a detector which has a controllable threshold level which can be varied finely from $-\infty$ to $+\infty$ to cope with all σ and q_0/q_1 values. The first step is to find the values q_0 or q_1 of the received block, since the decoder will get these values after the decoding process and not before, then one of these two values has to be transmitted, while

the other one will be calculated from Eq.(5.65). To transmit q_0 or q_1 within acceptable accuracy will require the transmission of additional digits with each block, which increase the transmission rate, and add redundancy to the transmitted digits. The second step is to calculate the noise power σ^2, to give the decoder an idea about the noise in the transmission channel, a special sequence known to the receiver has to be transmitted so that an initial value of the noise can be calculated from Eq.(5.7)

$$w_i = y_i - x_i \tag{5.67}$$

The number of digits transmitted in this sequence depends on the accuracy required in the noise power calculation, the longer the sequence the more accurate the calculation will be. Once the initial sequence transmission is ended, the information blocks are tranmitted. At this stage the optimum threshold value for the first block can be calculated, using the initial value of σ^2 and the first block transmitted value of q_0 or q_1, from Eqs.(5.47) or (5.48). At the end of the first block decoding, the decoder has calculated an additional n noise samples, same as in Eq.(5.67), but using the decoded values as x_i. These n samples are used to update the noise power value, so that the updated value is used to calculate the optimum threshold value for the next block and so on.

The optimum threshold value has to be calculated before the information and parity-check block can be

decoded, which will introduce delay to the system.

This decoder may function satisfactorily in some channels where the noise value is constant i.e. random error generating channels, or the slowly noise varying channel, because in the above described decoder the noise information used in the calculation of the present block is the noise value of the channel for the previous period up to the beginning of the present block, so any sudden variation in the noise power during the transmission of the present block, will result in a large difference in the noise power, hence the calculated threshold is no longer optimum. If the decoding of the present block was successful, in which the received block was the same as the transmitted one, then the noise power value is updated, and the threshold value for the next block is optimum. If the decoding block was unsuccessful, then the decoded values of x_i will be erroneous, thus the updated noise value calculated by the use of Eq.(5.67) is wrong which will cause the calculation of the next block threshold to result in a value other than the optimum value. The repetition of this process may cause the propagation of erroneous decoding, but generally the noise power will move nearer to the correct calculated value with every correct block decoding after the sudden change in the noise power. Since the sudden change of noise power will result in the calculation of a nonoptimum threshold value, and may cause an erroneous decoding of the block in which the noise has changed during its

transmission, furthermore the erroneous decoding might spread to the next block or block, then it is clear that this type of decoder is not suitable for decoding transmitted signals over fast noise varying channels, and bursty channels.

5.6-The Multiple Fixed Thresholds Decoder

The decoder of section 5.5 is clearly a complicated one, and has the disadvantages of introducing additional delay, more redundancy, and more complexity into the system. To avoid these disadvantages, a different decoder will be introduced based on the idea of using multiple fixed thresholds as the decision levels. The idea of multiple fixed thresholds can be described as follows. The optimum threshold values of fig.(5.9) shows that for some certain threshold values, one threshold can be optimum or very near optimum for more than one transmitted codeword at different SNR values, i.e. different values of q_1/q_0 and σ^2. For example the threshold value which is at a distance of $d_1/V=0.966$ is optimum for the detection of the codewords which has $q_1/q_0=0.875$ at SNR=3dB, and $q_1/q_0=2/3$ at 7dB, while it is near optimum for the codewords which has $q_1/q_0=2/3$ at SNR=5dB, and $q_1/q_0=0.875$ at SNR=5dB. Although the last two values are not optimum, yet these values are much nearer to the optimum threshold values than the conventional zero volt threshold. Such thresholds can be used for decoding these two codewords at different SNR values to obtain better results than the zero volt threshold. By a careful choice

of threshold values one can serve more than one codeword at different SNR, the number of thresholds to be used can be reduced from an infinitely, large number of optimum thresholds to a few optimum and near optimum thresholds. Fig.(5.9) is derived from the threshold values of the BCH(15,7) random code, and the threshold values mentioned above are applicable to that code only, where the curve values are calculated by Eq.(5.47). In general one should expect curves similar to fig.(5.9) for any code, because the threshold distance d_1 calculated from Eq.(5.47) does not correspond to the code itself but to the ratio q_1/q_0 which can be present in any code. Hence for longer codes there will be more q_1/q_0 values, thus more curves than fig.(5.9), consequently, one chosen threshold may fit many optimum thresholds at different SNR values, and may serve many other near optimum thresholds.

Assuming that for a certain code, thresholds values were found as described above, and the received word is detected by each threshold value, then decoded and corrected. Each received word is detected at least once by a threshold value that is optimum or near optimum regardless of the SNR value in the transmission channel so, all the decoder has to do is to pick the corrected output of the optimum or near optimum. Such a decoder is much less complicated than the decoder of section 5.5 and it has some advantages over other decoders. Firstly, it does not introduce any redundancy digits to the

transmitted bits, thus the transmission bit rate is unchanged. Secondly, the threshold values are fixed, so they can be preset in the decoder. Thirdly, the decoder function is independent of the SNR value in the transmission channel. Fourthly, the probability of error using this decoder is higher than the probability of error of the decoder of section 5.5, yet it is lower than the probability of error of the conventional zero threshold decoder.

To calculate the degradation in the probability of error when a near optimum threshold is used for the detection of a received word, let the transmitted word have the probabilities q_1 and q_0 for the transmitted digit to be '1' and '0' respectively, and let the optimum threshold be at a distance d_1 from +V, and d_2 from -V, the probability of error is given in Eq.(5.36)

$$P_e = q_0 \, \mathrm{erf}\left(\frac{d_2}{\sigma}\right) + q_1 \, \mathrm{erf}\left(\frac{-d_1}{\sigma}\right) \qquad (5.36)$$

Let the near optimum threshold used to detect this received word be at a distance d_3 from +V, and d_4 from -V, where according to Eq.(5.21)

$$d_4 = 2V - d_3 \qquad (5.68)$$

The probability of error for the detection with the near optimum threshold P_{en} is

$$P_{en} = q_0 \, \mathrm{erf}\left(\frac{d_4}{\sigma}\right) + q_1 \, \mathrm{erf}\left(\frac{-d_3}{\sigma}\right) \tag{5.69}$$

Probability of error degradation $P_{e_{deg}}$ is

$$P_{e_{deg}} = P_{en} - P_e \tag{5.70}$$

$$P_{e_{deg}} = q_0 \left[\mathrm{erf}\left(\frac{d_4}{\sigma}\right) - \mathrm{erf}\left(\frac{d_2}{\sigma}\right)\right] + q_1 \left[\mathrm{erf}\left(\frac{-d_3}{\sigma}\right) - \mathrm{erf}\left(\frac{-d_1}{\sigma}\right)\right] \tag{5.71}$$

substituting Eqs.(5.21) and (5.68) in Eq.(5.71) gives

$$P_{e_{deg}} = q_0 \left[\mathrm{erf}\left(\frac{2V-d_3}{\sigma}\right) - \mathrm{erf}\left(\frac{2V-d_1}{\sigma}\right)\right] + q_1 \left[\mathrm{erf}\left(\frac{-d_3}{\sigma}\right) - \mathrm{erf}\left(\frac{-d_1}{\sigma}\right)\right] \tag{5.72}$$

the degradation is shown in fig.(5.10) as the blocked area, while the probability of error for the optimum threshold is the shaded area.

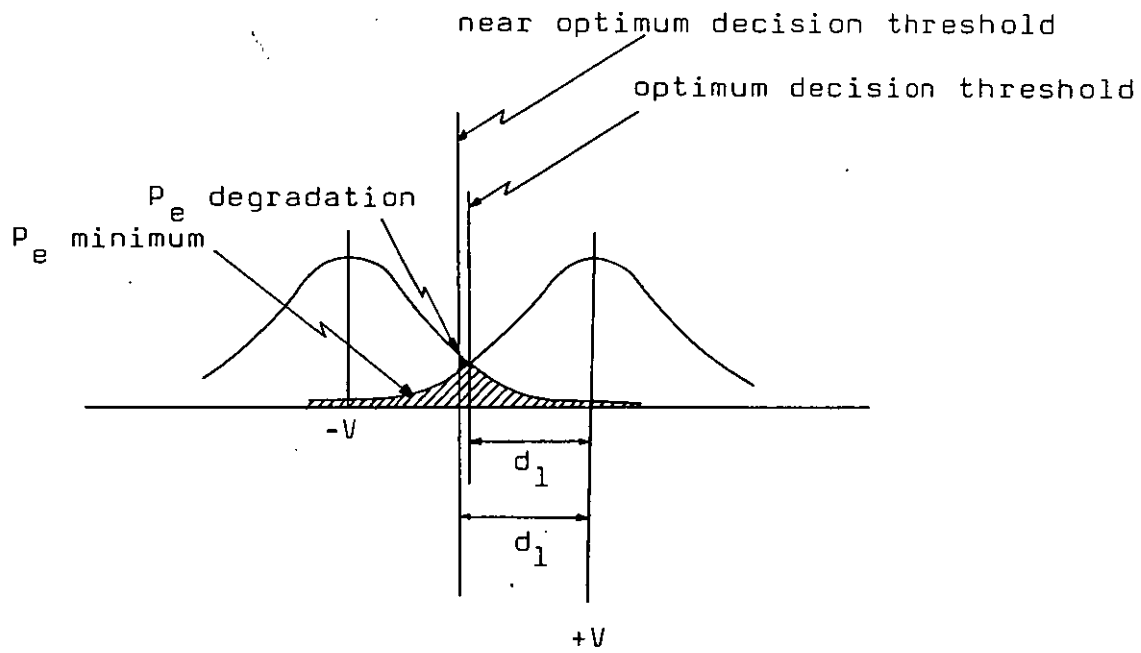


Fig.(5.10) Probability of error and its degradation

When the value of the detection threshold is reasonably close to the optimum detection threshold, then the degradation can be ignored without affecting the probability of error value seriously, as it can be seen from fig.(5.10)

5.7-Thresholds Values Choice

In a practical system it is impossible to prefix the decision thresholds to all the optimum threshold values for all the transmitted codewords and all the possible SNR values in the transmission channel, because the number of these values is so big, it makes the decoder very complicated and expensive to build. To avoid this, a reasonable number of thresholds should be used so that the decoder will give an improved performance at an acceptable cost and complexity. Several methods will be described for choosing decision threshold values; all these methods can be divided into two categories, the code independent thresholds, and the code dependent thresholds. Figs.(5.3) to (5.8) show that all six codes used are symmetrical about the x=n/2 axis. This means that the optimum thresholds of each group will be distributed symmetrically on both sides of the x=n/2 point. Another result of a special interest is the point x=n/2 itself because at that point for any code

$$N0O = N0Z = n/2 \tag{5.73}$$

which gives the '1' and '0's probability from Eqs.(5.56)

and (5.57) as in Eq.(5.53)

$$q_1 = q_0 = \frac{1}{2} \tag{5.53}$$

and results in an optimum thresholds distance of zero volts as in Eq.(5.54)

$$d_1 = d_2 = V \tag{5.54}$$

given that

$$|-V| = |+V| \tag{5.74}$$

It can be said from the above discussion that every group of the optimum thresholds are symmetrically distributed on both sides of the zero volt threshold, which corresponds to the x=n/2 point on the frequency distribution curve.

5.7.1 The Independent Thresholds

These groups of thresholds can be used for the decoding of the received word regradless of the code used. They are symmetrically distributed on both sides of the zero volt threshold. Let the number of thresholds on each side of the zero volt threshold be F, excluding the zero volt threshold. Since the thresholds are symmetrical around the zero volt thresholds, then the total number of thresholds required by the decoder is

$Z = 2F$ if zero volt threshold is not used (5.75-a)

$Z = 2F+1$ if zero volt threshold is used (5.57-b)

Three groups of the independent thresholds will be discussed.

a) The Full Span Thresholds: The Z thresholds are spaced equally between +V and -V, since the distance between every two thresholds is constant, then the voltage difference between every two u is

$$u = 2V/Z-1 \qquad (5.76)$$

If the zero volt threshold is used, then the nearest threshold from each side to the zero volt threshold is at a distance of u volts, while if the zero threshold is not used, the two nearest thresholds to zero volt are at a distance of ±u/2.

b)The Partial Span Thresholds: Fig.(5.9) show that for a large range of SNR values the threshold is not at a distance more than V/4 from the zero volt point (excluding all zero and all ones codewords). Thus the full span thresholds are not economical, because 75% of the thresholds are very far from the optimum threshold, and do not serve any optimum threshold within a reasonable range of the SNR values, so these thresholds are spaced equally between two voltage values, say $+v_1$ and $-v_1$, where these values are greater than the maximum distance of a threshold from the $^{d}1/V=1$ point for the code used. In general v can be chosen as V/3 without regarding the code used. The voltage difference between any two thresholds u is

$$u = {}^{2v_1}/_{Z-1} \tag{5.77}$$

again as in the full span thresholds, the nearest thresholds are at a distance ±u if the zero voltage threshold is used, otherwise, the thresholds are at a distance of ±u/2.

c)The Partial Nonequidistance Thresholds: Furthermore fig.(5.9) show that the threshold values has a special distribution, where they are concentrated near the $\frac{d_1}{V}=1$ point, and become more scattered as the distances grow bigger. To match this, those thresholds are spaced so that the distance is bigger as the threshold is further from $\frac{d_1}{V}=1$ point. Again, to keep the advantages gained by using partial span thresholds, these thresholds cover the central part of the distance between +V and -V, say $+v_1$ to $-v_1$. A good law to set the distance between thresholds is the doubling law, where the distance to the next threshold (which is further from $\frac{d_1}{V}=1$ point) is double the previous threshold (which is nearer to the $\frac{d_1}{V}=1$ point). Here two cases must be considered, firstly, when the zero voltage threshold (at $\frac{d_1}{V}=1$ point) is used, the threshold voltage for the i^{th} threshold from the zero voltage u_i is

$$u_i = \sum_{m=1}^{i} \frac{2^{(m-1)}}{2^F - 1} v_1 \tag{5.78}$$

Secondly, if the zero voltage threshold is not to be used, the threshold voltage for the first threshold

from the zero voltage u_1 is

$$u_1 = \frac{v_1}{2^{F+1}-3} \tag{5.79}$$

the i^{th} voltage value u_i is

$$u_i = \frac{v_1}{2^{F+1}-3} + \sum_{m=2}^{i} \frac{2^m v_1}{2^{F+1}-3} \quad \text{for } 1 \leqslant i \leqslant F \tag{5.80}$$

$$u_i = \frac{v_1}{2^{F+1}-3} \left(1 + \sum_{m=2}^{i} 2^m \right) \quad \text{for } 1 \leqslant i \leqslant F \tag{5.81}$$

fig.(5.11-a) shows the threshold values for F=3, when the zero volt threshold is used, while fig.(5.11-b) shows the thresholds values for F=3, when the zero volt threshold is not used.

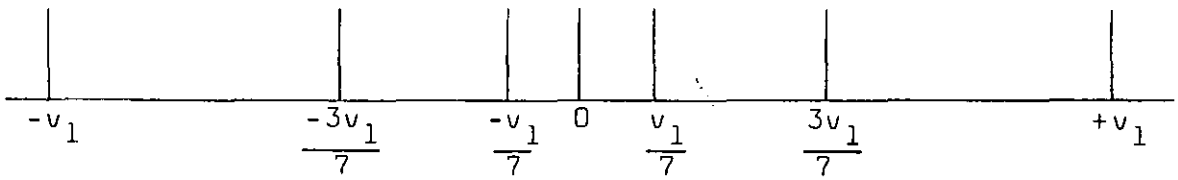


Fig.(5.11-a)Threshold for F=3 with 0 volt threshold

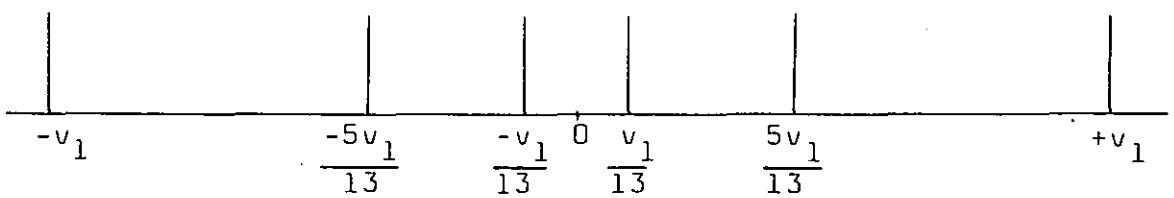


Fig.(5.11-b)Thresholds for F=3

The thresholds values calculated by Eqs.(5.76), (5.77),(5.78).(5.79), and (5.81) are independent of the code used, but they are not the optimum values, although a few of them could be. The probability of having a number of the optimum thresholds in the calculated thresholds becomes bigger as the number of the calculated thresholds grows bigger. When the number of the calculated thresholds reaches infinity, all the optimum thresholds are included in the calculated set. Consequently, the decoder performance will improve as the number of thresholds used is increased. A decoder using a fixed number of thresholds calculated to form a partial span of thresholds will perform much better than the case where the thresholds form a full span, while, if the thresholds were arranged as a partial nonequidistance,the decoder will perform even better, but as the thresholds number is increased the improvment over the other systems become smaller, till at some stage all the three groups perform the same, and no more improvment can be achieved by increasing the threshold numbers beyond this point.

5.7.2 The Code Dependent Thresholds

Each group of these codes are derived to be used with the specific code they are derived for. Three types will be discussed here, two are channel noise dependent, while the third is channel noise independent. All three will be derived from code dependent equations or curves.

a)The Optimum Thresholds Groups: These groups are code and channel noise dependent, and the number of

thresholds is determined by the code used, where for each different value of the ratio $\frac{q_1}{q_0}$, one threshold exists. The thresholds values are calculated by Eq.(5.47)

$$d_1 = V + \frac{\sigma^2}{2V} \log \frac{q_1}{q_0} \qquad (5.47)$$

noise power values σ^2 are taken as the average noise power in the transmission channel.

For long codes, the number of thresholds calculated according to this group is large. In some applications, it may require from a practical point of view to use a fewer number of thresholds in the decoder. In such cases the furthest thresholds on both sides are omitted till the required number is reached.

These groups of thresholds are the best to use (if no thresholds are omitted), because all the thresholds are optimum. The big disadvantage is that the only way to control the threshold number without omitting any threshold is by using a different code. The other disadvantage is that the code itself dictates the cost and complexity of the decoder through the number of thresholds. Consequently, code choice for these type of threshold decoders is somewhat critical.

b)The Optimum Spaced Groups: These groups are also code and channel noise dependent and have the advantage of having a flexible number of thresholds in each group, but at the cost of the thresholds used being optimum or near optimum.

Thresholds are derived from the distance frequency of NOO or NOZ curve. Distance d_1 is calculated from Eq.(5.47) for each value of the ratio q_1/q_0, at the average noise power σ^2, the results are plotted against the frequency of NOO or NOZ, which can be called the frequency of the ratio q_1/q_0, or the frequency of thresholds. Fig.(5.12) shows this curve for the BCH(31,21) random code, at SNR=3 dB.

Assuming that one threshold is required (F=0), a line which has the equation

$$y_1 = f_{max} \tag{5.82}$$

is drawn where f_{max} is the maximum frequency value, this line will intersect with the curve at the point d_1, giving the distance of the threshold required, which is for this code

$$d_1 = V \tag{5.83}$$

which is the zero volt threshold in this case. Now consider the case where F=1, the intersection line equation is

$$y_2 = f_{max}/2 \tag{5.84}$$

This line will give two thresholds values symmetrical or nearly symmetrical about the zero volt threshold. Similarily for F=2 two equations for the intersection lines exist, where

$$y_1 = f_{max}/3 \qquad (5.85\text{-a})$$

$$y_2 = 2f_{max}/3 \qquad (5.85\text{-a})$$

which gives four threshold values. In general, when Z thresholds are required, the intersection equations are F, when no zero volt threshold is used, and F+1, when zero volt threshold is used, the intersection equations are given by

$$y_j = j \frac{f_{max}}{F+1} \qquad (5.86)$$

where for zero volt threshold

$$j = 1 \quad \text{and} \quad F = 0 \qquad (5.87\text{-a})$$

and for the other thresholds

$$j = 1,2,....F \quad \text{when} \quad F > 0 \qquad (5.87\text{-b})$$

fig.(5.12) shows the intersection lines for the BCH(31,21) code and the threshold distances for Z=5.

Some codes have a curve shape other then the shape of fig.(5.12). For these codes Eqs.(5.82) to (5.87) does not apply, but the principle of dividing the y axis into equal segments to get the threshold distance still applies, and is used to find the threshold values. An example of these codes is the BCH(15,7) random code, where fig.(5.13) shows the distance frequency curve, and the intersection lines for finding thresholds for Z=6.

These two methods of finding thresholds are suitable only for channels where the SNR value does not deviate too much from the average value, otherwise the threshold

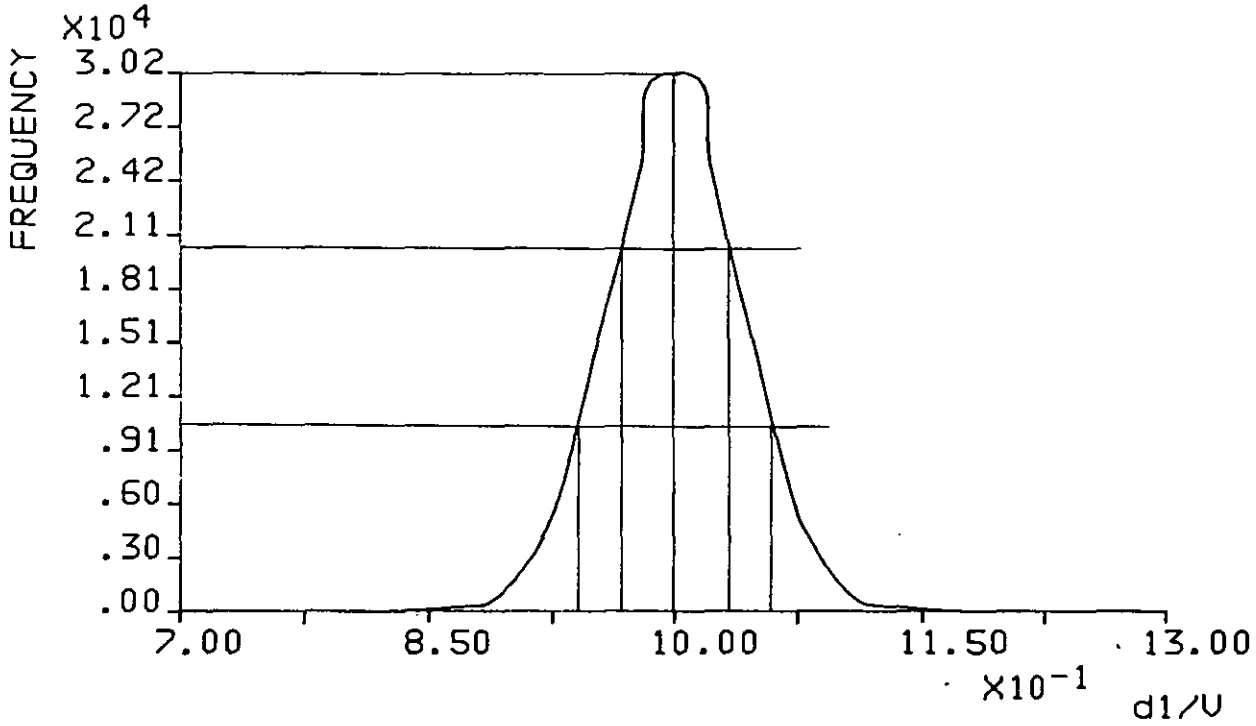


FIG.(5.12) DISTANCE VS FREQUENCY CURVE AT SNR= 3 dB

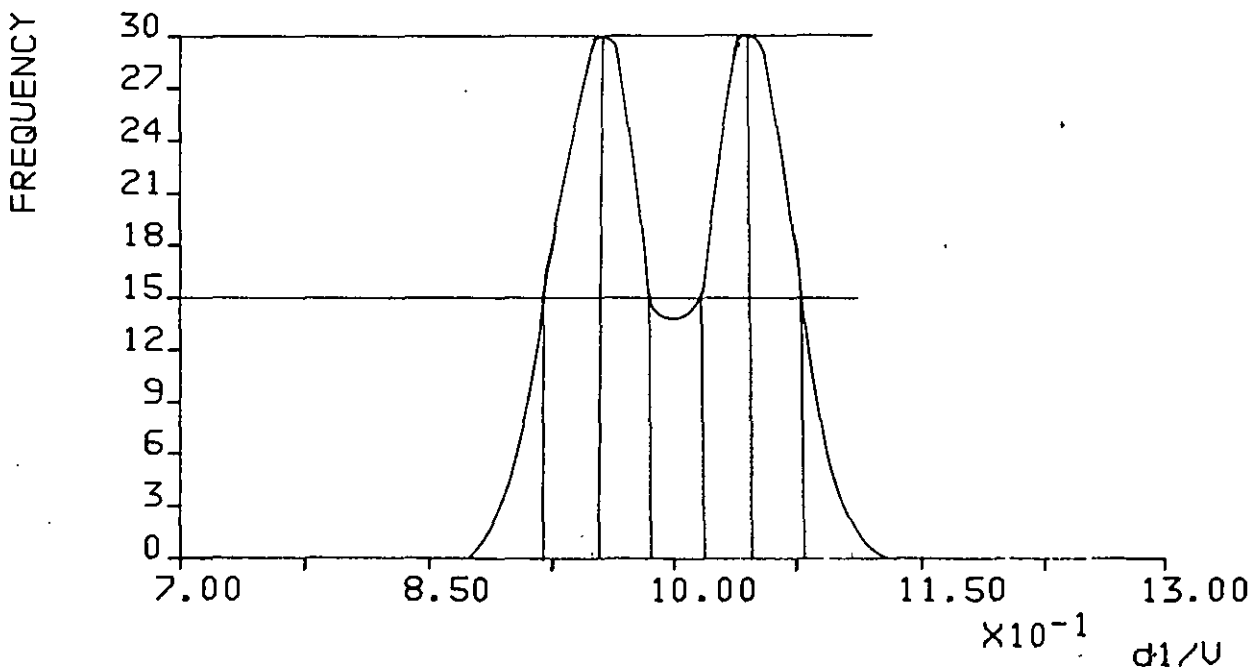


FIG.(5.13) THRESHOLD VALUES FOR Z=6

values used will no longer be optimum. Thus these two methods are suitable for random-error channels, and are not optimum for the burst-error channels.

c)The Practical Spaced Groups: The disadvantage of the average noise power or the SNR being relatively constant, in the optimum thresholds groups, and the optimal spaced groups, hence the degraded improvement when these groups are used in a decoder for decoding burst-noise channels, led to thinking in a way to isolate the noise effect on the thresholds values. The obvious way to achieve this is by taking the SNR at a lower value than the average SNR in the transmission channel. This process will shift the thresholds from their optimum values, but they will still be placed in good places for the decoder to achieve a good improvement. This arrangement will work satisfactorily at both low and high SNR values. At low SNR values, threshold values are very close to the optimum values, because the chosen SNR value is close to the low SNR value. Thus the decoder will perform well giving a good improvement. On the other hand, at high SNR values, threshold values are not optimum, but it can be seen from fig.(5.9) that threshold values are close together near the $\frac{d_1}{V}=1$ point, and they become further apart near the bottom of the curve, furthermore it can be seen that as the SNR values increase the threshold values draw closer and closer. Hence at high SNR, the threshold values chosen by the arrangement described above near the $\frac{d_1}{V}=1$ point will serve as near

optimum thresholds and the decoder will perform satisfactorily. A practical value for lowering the average SNR value can be taken as 3-10 dB, although bigger values can be used for channels with very sharp noise power changes.

Another simple method is based on the observation that the NOO vs the frequency curve of fig.(5.3) is similar to the threshold distance curve, so the frequency curve is used for finding the threshold values. The $\frac{d_1}{V}$ values on the x axis are determined, based on the three following facts

a)At the ratio $\frac{q_1}{q_0}=1$, the threshold value is zero volts, and this corresponds to the $\frac{d_1}{V}=1$ point.

b)At the ratio $\frac{q_1}{q_0}=0$, the threshold value is at $-\infty$ volt, but since in a practical system the incoming signal is limited to the logical '0' value, then the threshold value is at -V volt, which corresponds to the point $\frac{d_1}{V}=0$ on the x axis.

c)Conversely, the practical threshold value for the ratio $\frac{q_1}{q_0}=\infty$, is +V, which corresponds to the point $\frac{d_1}{V}=2$ on the x axis.

After defining the x axis, thresholds values can be found from the frequency curve using Eqs.(5.86), and (5.87). The advantage of the latest method is its independence of the noise channel values, which save the inaccurate choice of the SNR values yet it is code dependent.

Threshold values for these groups are calculated from the corresponding curve, as in the optimal spaced groups, by finding the intersection line equation or

equations from Eqs.(5.82) to (5.87).

To compare the improvement when each group is used, the BCH(31,21) random code and the (34,22) burst code are used, in a five threshold parallel threshold decoder. Five groups are used in the comparsion. Firstly the full span thresholds. Secondly the partial span thresholds ($\pm v_1 = \frac{V}{2}$). Thirdly the partial nonequidistance thresholds, where $\pm v_1 = \frac{V}{2}$, and the threshold values calculated from Eq.(5.78). The fourth group is the parallel spaced group, where the trhesholds are calculated at SNR value of 1 dB for the random code, and -5 dB for the burst code. The last group is again the practial spaced group, but the threshold values are derived from the NOO vs frequency curve. The test results are shown in fig.(5.14) for the random code, and fig.(5.15) for the burst code. Where it can be seen from fig.(5.14) that all thresholds groups perform nearly the same, and better than the full span thresholds. Fig.(5.15) show again that the full span thresholds perform worse than the others, while the practical spaced groups followed both nearly the same curve that performs better than the rest, the remaining two performed nearly the same.

5.8-Choosing A Suitable Error-Pattern

A decoder which has multiple fixed thresholds, say j fixed thresholds, will have at the end of a decoding cycle j received words. The received word is detected by the j thresholds, the output of each threshold is fed to

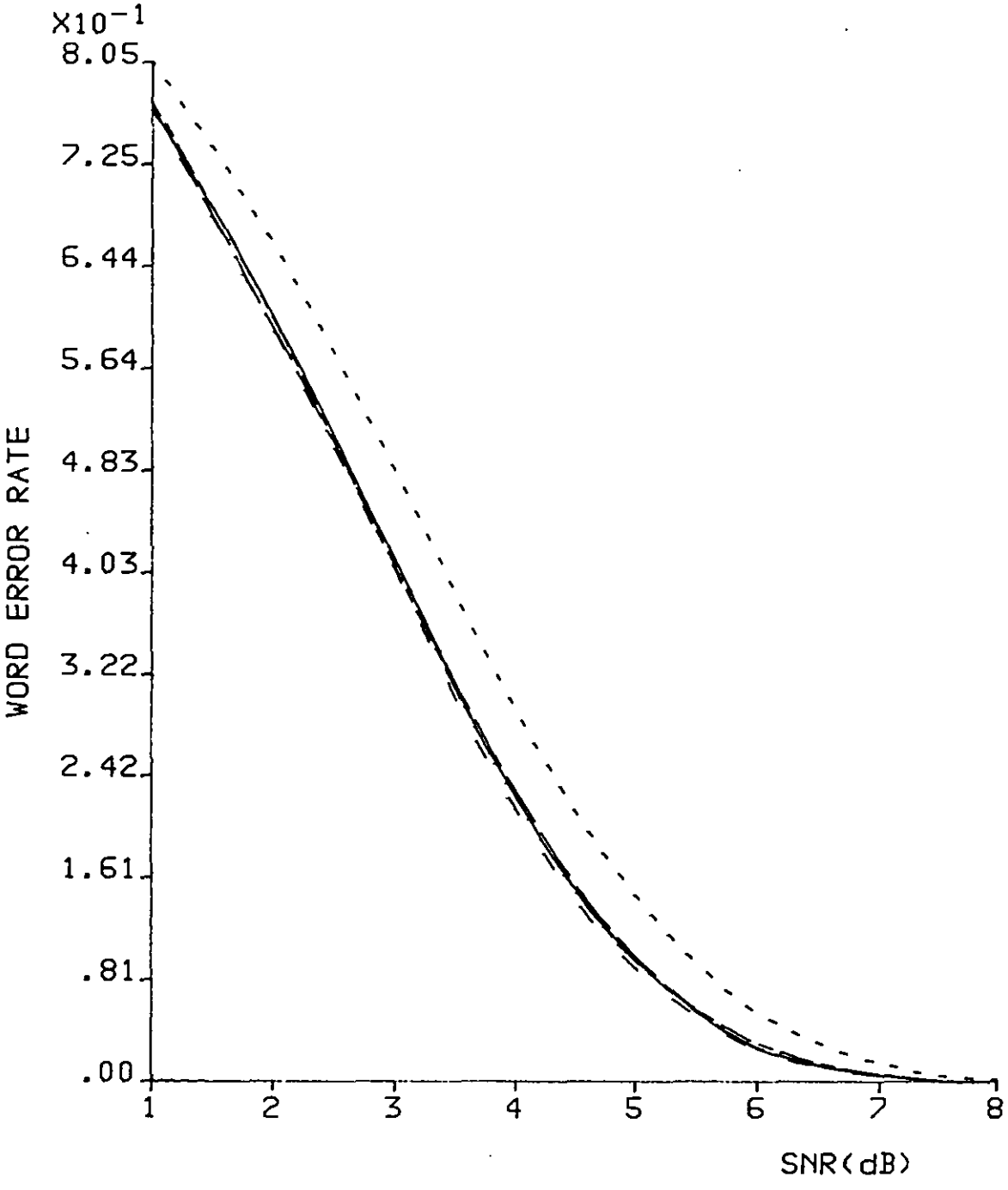


FIG.(5.14) PERFORMANCE OF DIFFERENT THRESHOLD GROUPS

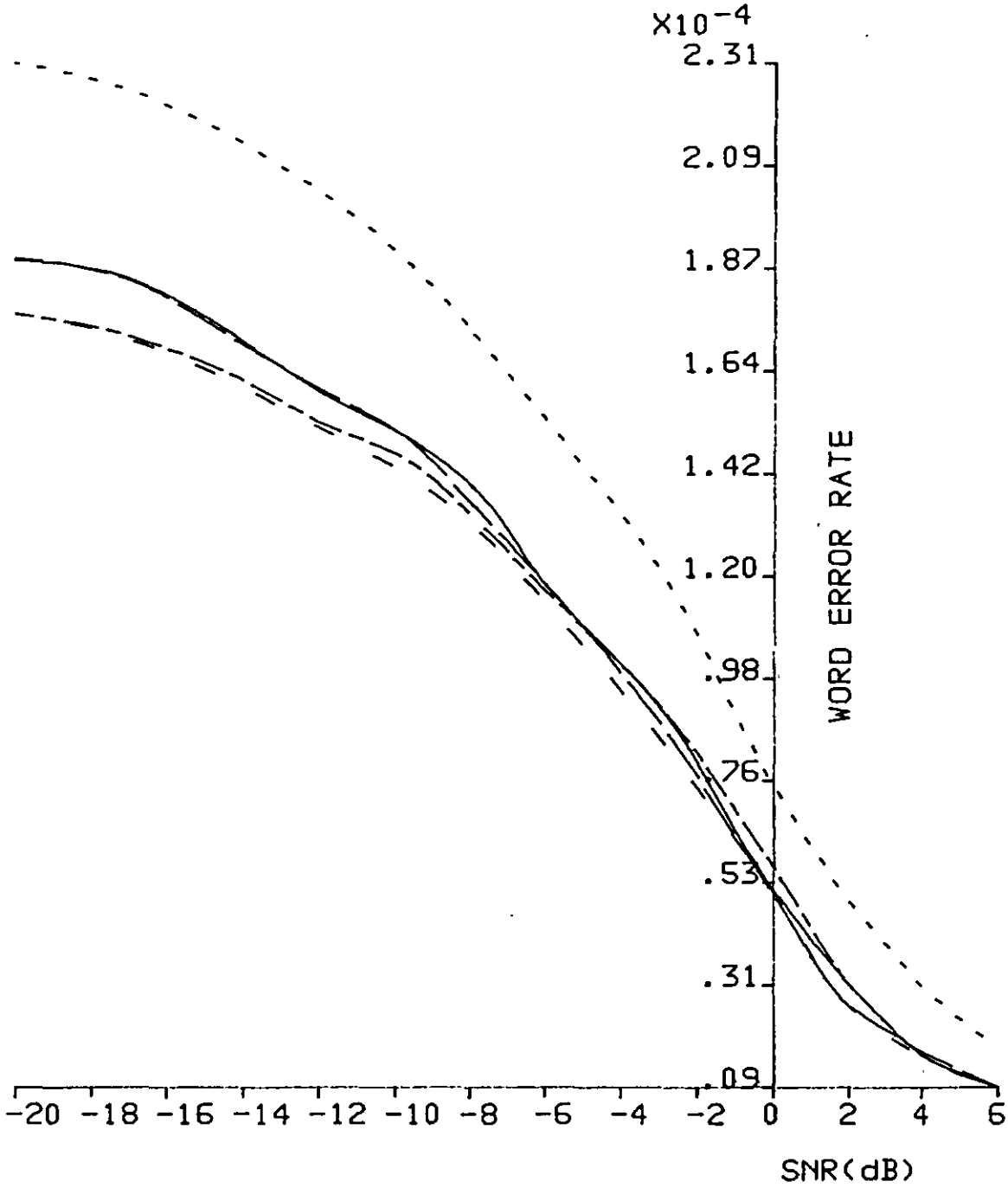


FIG.(5.15) PERFORMANCE OF DIFFERENT THRESHOLD GROUPS

a subdecoder, which will calculate an error-pattern, dependent on the value of its output. Hence for the j thresholds there will be j vectors of error-patterns

$$\left.\begin{aligned} E_1 &= e_0\ e_1 \cdots\cdots e_{n-k-1} \\ E_2 &= e_0\ e_1 \cdots\cdots e_{n-k-1} \\ &\vdots \\ E_j &= e_0\ e_1 \cdots\cdots e_{n-k-1} \end{aligned}\right\} \qquad (5.88)$$

One of these vectors, say the i^{th}, is a result of the detection of the received word by an optimum or near optimun threshold, Consequently the i^{th} vector is the most likely error-pattern to be added to the transmitted codeword in the channel. The i^{th} received word, which is the result of the received word detection by the i^{th} threshold, is corrected by the i^{th} error-pattern vector. The corrected word is the most likely transmitted codeword, where the probability of error for the received word is minimum according to Eq.(5.36). The point is that the decoder has no means of calculating Eq.(5.36) to decide which of the j thresholds is the optimum or near optimum one. Thus the decoder cannot make any decision on which one of the j corrected words is the transmitted codeword.

5.8.1 Probability of a Threshold Being Optimum During a Transmission

In general the data generated at the source is random data, and since it is divided into blocks of k digits each, then the contents fo each block is randomly distributed and can be any of the 2^k possible combinations. To each block the encoder annexes n-k parity-check digits, so the resultant n digits are unique for each of the 2^k blocks. Since the k information digits in a block can equally likely be any combination, then the transmitted n digits block can be equally likely any of the 2^k codewords. The probability of an error-pattern E_i, being the error-pattern added to the transmitted signal in the channel, is the probability of the transmitted codeword being one of the codewords that has the i^{th} threshold as an optimum or near optimum threshold.

The frequency distribution is unique for every code, and is universal frequency distribution, because the test is carried out for all the possible codewords. The discrete probability density function for the '1's or '0's can be calculated from the frequency distribution

$$p_i = \frac{f_i}{2^k} \quad \text{for } i = 1,2,\ldots,n \qquad (5.89)$$

where p_i is the probability of the transmitted word having i number of '1's or '0's in its n digits, and f_i is the frequency of the codewords which has i number of '1's or '0's. It can be seen from Eq.(5.89) that the

discrete probability distribution is the discrete frequency distribution scaled down by the factor of 2^k. Since the probability distribution function of the '1's or '0's is of similar shape of the frequency distribution, then it can be seen easily that the thresholds nearer to the $\frac{n}{2}$ point in fig.(5.3) has a higher probability of being optimum or near optimum than other thresholds, and the probability of thresholds being optimum or near optimum becomes smaller as the distance between the threshold and the $\frac{n}{2}$ point grows bigger.

5.8.2 Threshold Weight Distribution Function

The discrete probability density function of Eq.(5.89) determine the probability of an optimum threshold being used during a transmission. But according to section 5.7 thresholds used in an actual decoder need not be always the optimum thresholds, but may be near optimum thresholds. Since the discrete probability density function exists only where an optimum threshold exists, then it is not a clear indication of the probabilities for the thresholds other than the optimum. Therefore, some sort of continuous indication is required. This indication is called the threshold weight distribution, which is to assign a weight value for each threshold value that indicates the probability of a threshold being used during a transmission. That is, the threshold weight distribution is the continuous probability density function for the '1's or '0's, assuming it exists outside the integer values. Since the

integer values in the discrete probability density function is increased by one for each new value, then the relation between distributions is

$$W_i = \int_{i-0.5}^{i+0.5} f(w) = p_i = \frac{f_i}{2^k} \qquad \text{for } i = 1,2,\ldots,n \tag{5.90}$$

where W_i is the weight of the threshold that is optimum for detecting a received word which has i '1's or '0's in the transmitted codeword, and $f(w)$ is the threshold weight distribution function. The threshold weight distribution for the BCH(31,21) random code is shown in fig.(5.16) where it is clear that whatever the threshold value within the range, a weight values does exist for that threshold.

Going back to the error-patterns detected by the decoder Eq.(5.88), the threshold weight value W_i for the i^{th} threshold used, is an indication of the probability that the error-pattern E_i detected by the use of the i^{th} threshold is optimum. Using the threshold weight distribution function, which is unique for each code, the decoder can choose the error-pattern which is more likely to be the one added to the transmitted signal during the transmission through the channel.

5.8.3 The Decoding Process

The threshold weight distribution function, will always point to the same threshold (or thresholds) as the

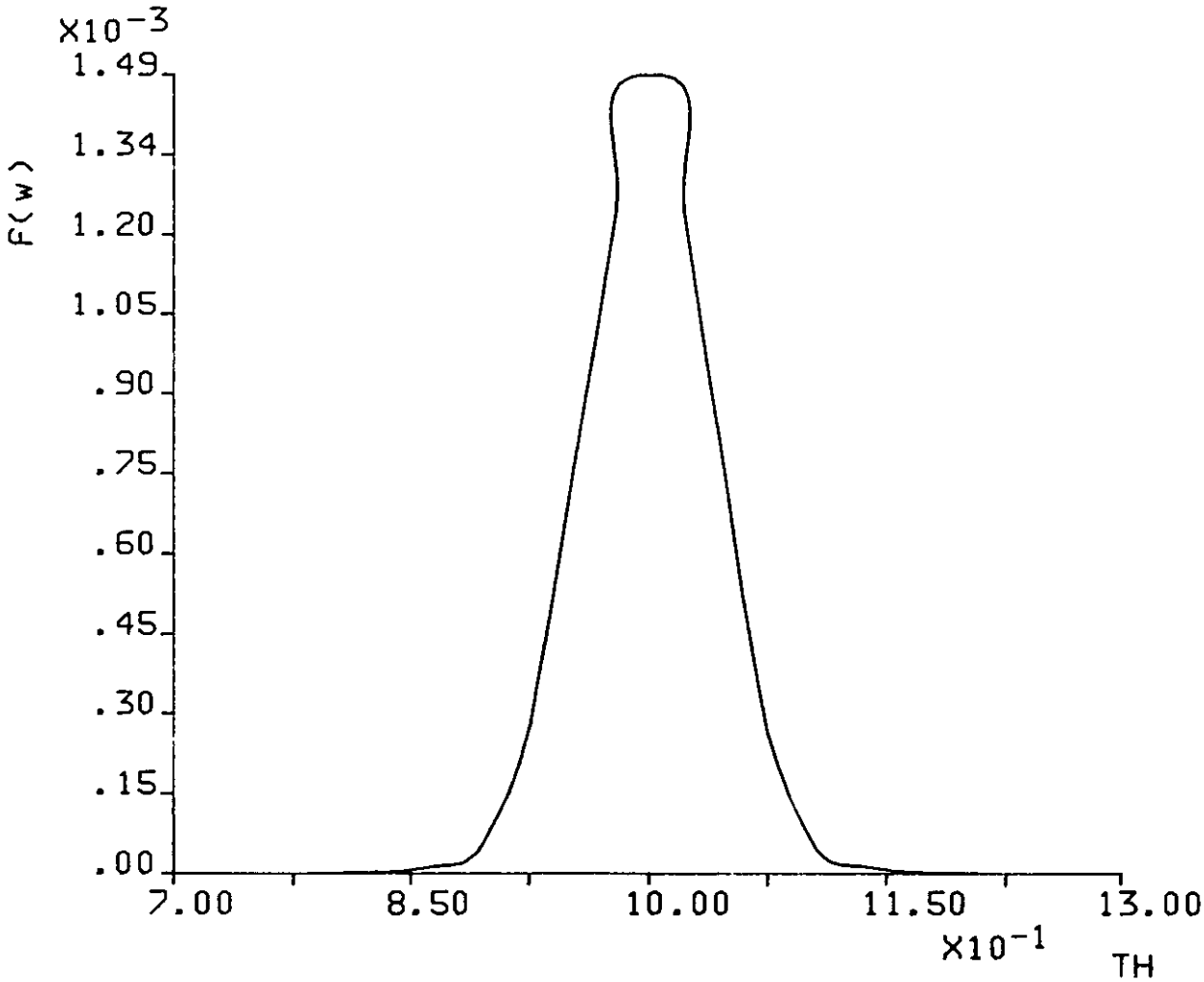


FIG.(5.16) THRESHOLD WEIGHT DISTRIBUTION

most likely optimum. This threshold (or thresholds) will be used for decoding every received word during the whole transmission if the decoder has no means of choosing a threshold for decoding other than using the weight value. Such a decoder will be a single threshold decoder, and any received word which has an optimum threshold other than the one used will not be detected, so that the probability of error is not minimum. Thus the decoder has to use an additional rule to choose a suitable threshold accordingly.

One method is to use an error-trapping decoder as a subdecoder, because the error-trapping decoder has the advantage of detecting whether a correctable error-pattern is trapped, or untrappable error-pattern is added to the transmitted signal in the transmission channel. The decoder described in what follows makes use of the threshold weight distribution function and the decoder's ability to detect the untrappable error-patterns.

Consider a decoder which is using a number of thresholds, say j thresholds for decoding the received word. Such a decoder consists of j identical subdecoders (each subdecoder is the syndrome computer of an error-trapping decoder) and a flag. Each flag is connected to the syndrome computer in such a way that as soon as a trappable error-pattern is detected at the syndrome register, the flag is set, otherwise it remains reset as before. The received word is detected by the j thresholds, and the output of each threshold is fed to its corresponding

subdecoder. At the end of the error-pattern detection phase, some of the j flags will be set, where trappable error-pattern are trapped, while other flags may remain reset. The decoder starts to search for a set flag among all the flags by testing each flag, starting by the flag of the threshold that has the biggest weight value, and ending with the threshold flag that has the smallest weight value, according to the decreasing weight values of their corresponding thresholds. Once a set flag is encountered, its threshold value is accepted as the optimum threshold. Hence the trapped error-pattern in the syndrome register is accepted as the error-pattern added to the transmitted signal during transmission in the channel, and the correction is made accordingly.

Due to the symmetry in the threshold weight distribution curve fig.(5.16), there will be pairs of thresholds which have equal weight values. In such cases it does not make any difference to the overall results, whichever threshold flag is tested first. Since the threshold weight distribution curve fig.(5.16) is a unique and constant curve for each code, then the flags test sequence can be preset, at the decoder, at the time the thresholds values are determined. This sequence remains the same as long as the thresholds values are not changed.

5.9-Parallel Threshold Decoders Performance

Up to now the choice of thresholds values can be one of two options, whatever the chosen thresholds weight

distribution, namely, the use of odd or even number of thresholds. This is, according to Eqs.(5.75), the choice of having or not having a threshold at zero volt. The first option is more appealing for the following reasons:

a)At high SNR values Eq.(5.52) can be rewritten as

$$d_1 \simeq V \tag{5.91}$$

where

$$\frac{1}{2SNR} \log \frac{q_1}{q_0} << 1 \tag{5.92}$$

and can be neglected without any serious error, where SNR value is as given in Eq.(5.51). Thus, the zero volt threshold is an optimum one regardless of the received codeword.

b)The zero voltage threshold is the optimum threshold that minimizes the probability of error for the whole transmission, as in section 5.3. Thus the inclusion of this threshold will give the decoder the ability to try to minimize the bit probability of error as well as the block probability of error.

c)The subdecoder that processes the zero threshold output will give the same results as the conventional error-trapping decoder, and by giving the zero threshold a weight higher than the maximum weight value, regardless of whether the threshold weight curve justifies this or not. The decoder tests the zero threshold subdecoder first, thus all trappable errors by the conventional decoder are accepted by the parallel threshold decoder.

Consequently, the parallel threshold decoder that uses this arrangement should not perform worse than the conventional decoder.

From this point on, it will be assumed that, the parallel threshold decoder has a threshold at zero volts, and that it has the maximum weight value.

5.9.1 The Random-Error Parallel Threshold Decoder Performance

As mentioned above, the parallel threshold decoder will test the zero voltage threshold first and it will accept any trappable error-pattern as the error-pattern added to the transmitted signal in the channel, thus it will perform as any ordinary error-trapping decoder. But once an untrappable error-pattern is encountered, the parallel threshold decoder tries to convert it to a trappable error-pattern, and then correct the erroneous digits.

Untrappable errors can be due to two reasons, firstly, the error-pattern contains more errors than the error-correcting capability of the code, in this case the decoder tries to reduce the number of errors to the correctable number of errors by the code used. Secondly, because of the error-trapping technique nature, the error-pattern could be correctable but untrappable. In such cases the decoder tries to convert the untrappable error-pattern to a trappable one, hence correction can be achieved. The latter source of untrappable errors is dependent on the code rate, where the number of untrappable errors becomes smaller as the code rate decreases. The

relation between the untrappable errors and the code rate for a block code of n=15, where the error correcting capability of the code is assumed to be t=3, is shown in fig.(5.17).

Consequently, the parallel threshold decoder is expected to perform slightly better with low code rates because the source of improvement is correcting the uncorrectable errors only. The performance is expected to improve with the increase of the code rate, because the number of untrappable but correctable errors increases with the increase of the code rate.

5.9.2 The Burst-Error Parallel Threshold Decoder Performance

Two types of errors are usually present in bursty channels,random errors, and burst errors. The code used should be able to cope with the burst errors. But because of the burst error definition, a random error at some distance from a burst, may change a trappable error-pattern to an untrappable one. In general an untrappable error-pattern will occur only if the overall burst length of the error-pattern is greater than the burst-error-correcting-capability ℓ of the code. When an interlacer is used then the expression 'overall burst length' represents the overall burst length that is confined to a particular block.

Untrappable bursts in bursty channels can be due to any of three sources. Firstly, a burst of length less than the correctable burst length ℓ, but a random error at some distance forces the burst length to become

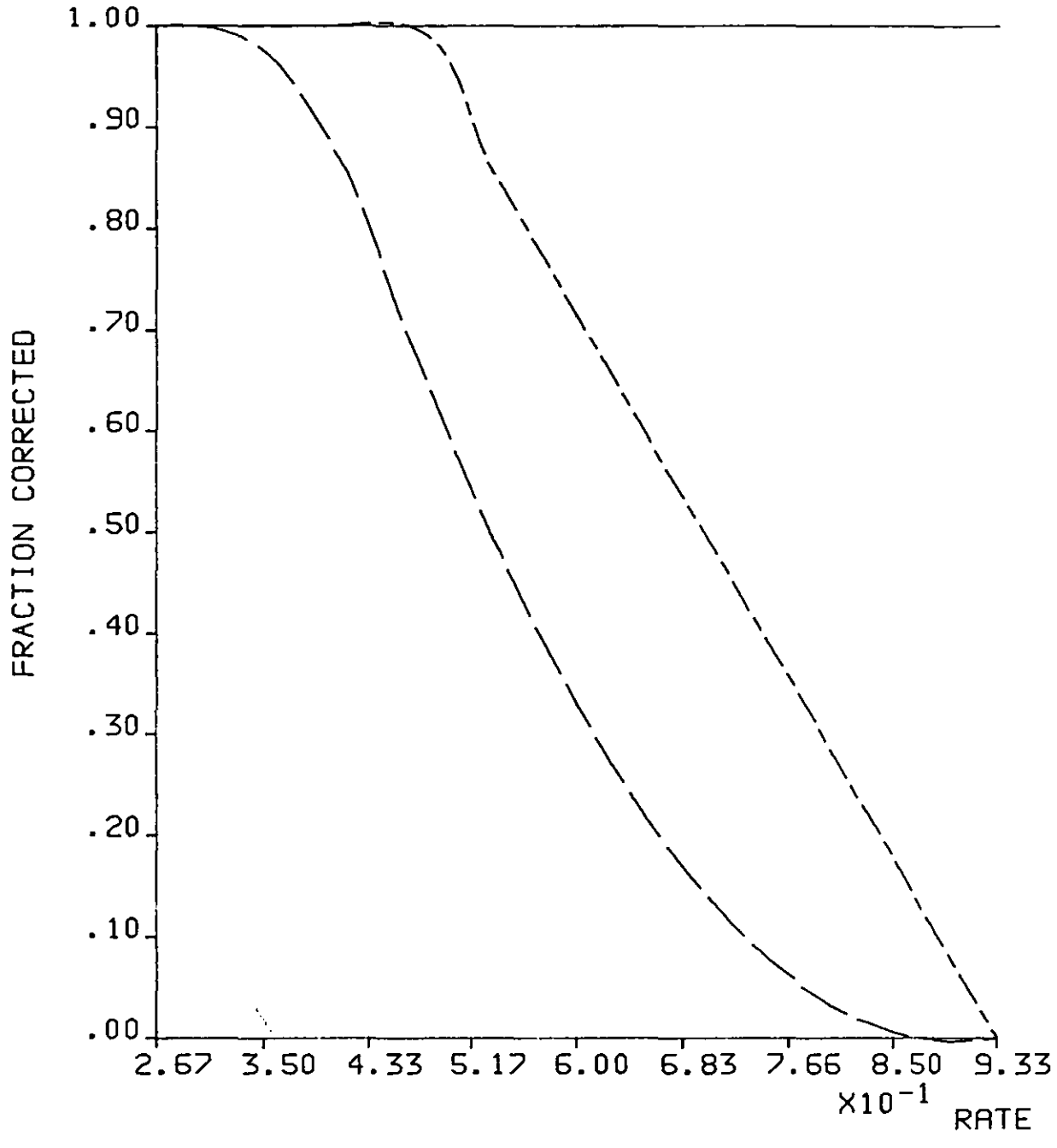


FIG.(5.17) FRACTION OF ERRORS CORRECTED VS CODE RATE

larger than ℓ. Secondly, the burst length is bigger than ℓ, and thirdly, the burst is longer than ℓ, and a random error makes it even longer.

Let us consider the probabilities of error when a threshold of a value, say v is used for decoding an error generated by the background noise which has the power σ_L, and an error generated by the burst noise which has the power σ_H. Let the probability of error for the background noise detection be P_{ea}, and for the burst noise be P_{eb}. Then from Eq.(5.36) it can be seen that, for any threshold value v and the same codeword

$$P_{ea} < P_{eb} \tag{5.93}$$

that is, the probability of detecting an error generated by the background noise correctly is greater than the probability of detecting correctly an error generated by the burst noise. Thus the parallel threshold decoder is able to trap a good deal of the error-patterns generated by the first source. Some error-patterns generated by the second source, and a little bit less of the third source.

5.10-Block Diagrams of the Parallel Threshold Decoders

Two block diagrams will be discussed, any one of them can be used for random-error correction or burst-error correction, keeping in mind that the appropriate error-trapping decoders should be used as subdecoders in each case. When the interlaceing technique is used

it is assumed that the deinterlacer is placed outside the decoder, although nothing can stop the use of an internal deinterlacer in the first block diagram, apart from the cost of too many shift registers.

5.10.1 First Block Diagram

A j parallel thresholds decoder consists of j branches of similar threshold and subdecoders circuits, correction circuits, and the control circuit, as shown in fig.(5.18). The sampled signal is fed to the j threshold detectors, where it is compared with the preset threshold value of each detector, the output of each threshold is as defined in Eqs.(5.8),but limiting the -V to '0', and the +V to '1'. Hence the outputs of the threshold detectors are binary symbols of either '0' and '1'. The output of the threshold detector is fed to a subdecoder, which is a complete error-trapping decoder excluding the correction circuit, which is shown in fig.(5.19). The k information digits are shifted into the k bit shift register, simultaneously the syndrome computer will be calculating the syndrome, as soon as the n digits are shifted in the syndrome computer, the syndrome is present at the syndrome computer. The syndrome contents are tested for trappable-errors, and then shifted. If during the next n shifts, an error-pattern is trapped, the shifting stops, and the flag is set, otherwise the flag remains reset. By this time the flag scan and gate control circuit monitor the zero volt threshold subdecoder. If at the end of the subdecoder n shifts the flag is set, the gate control

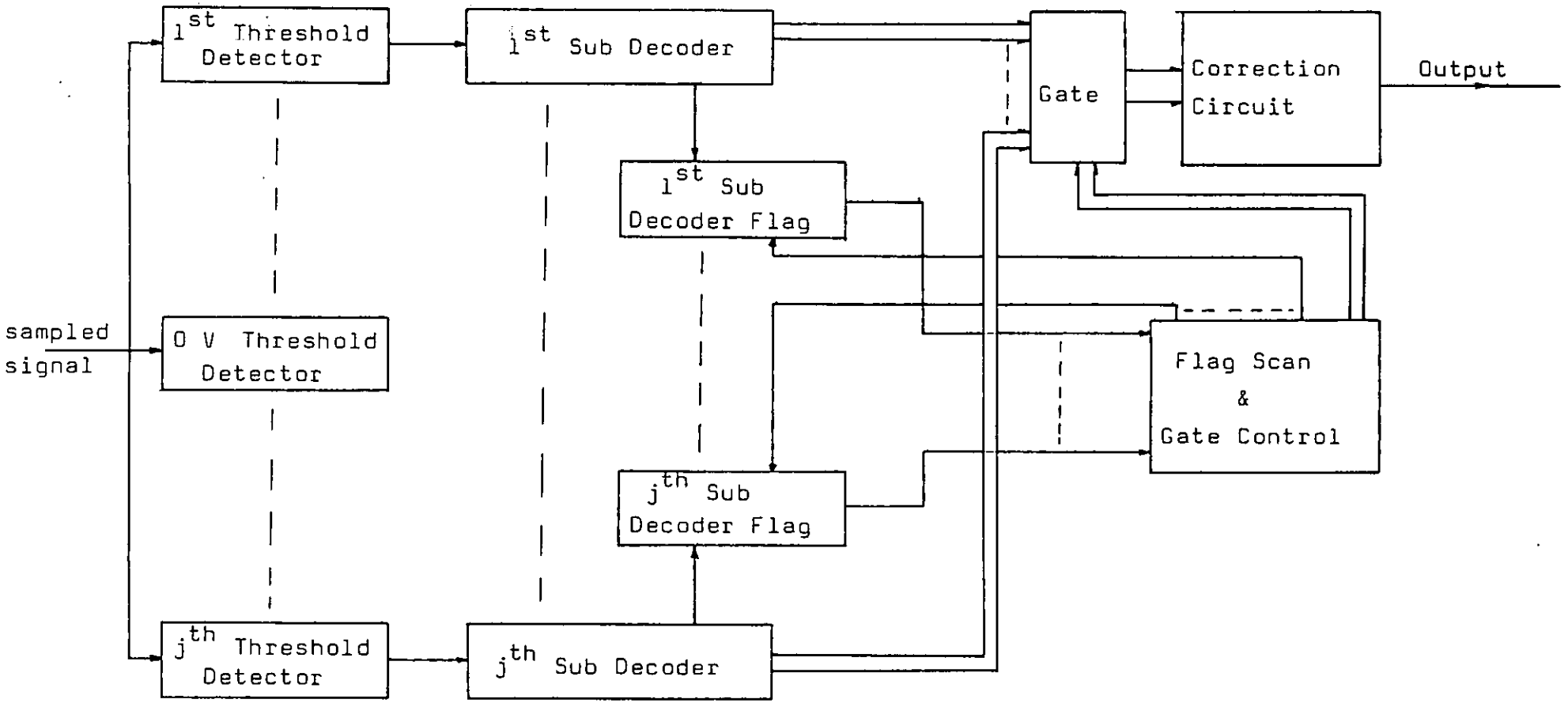


Fig.5.18 Block diagram of the PTD

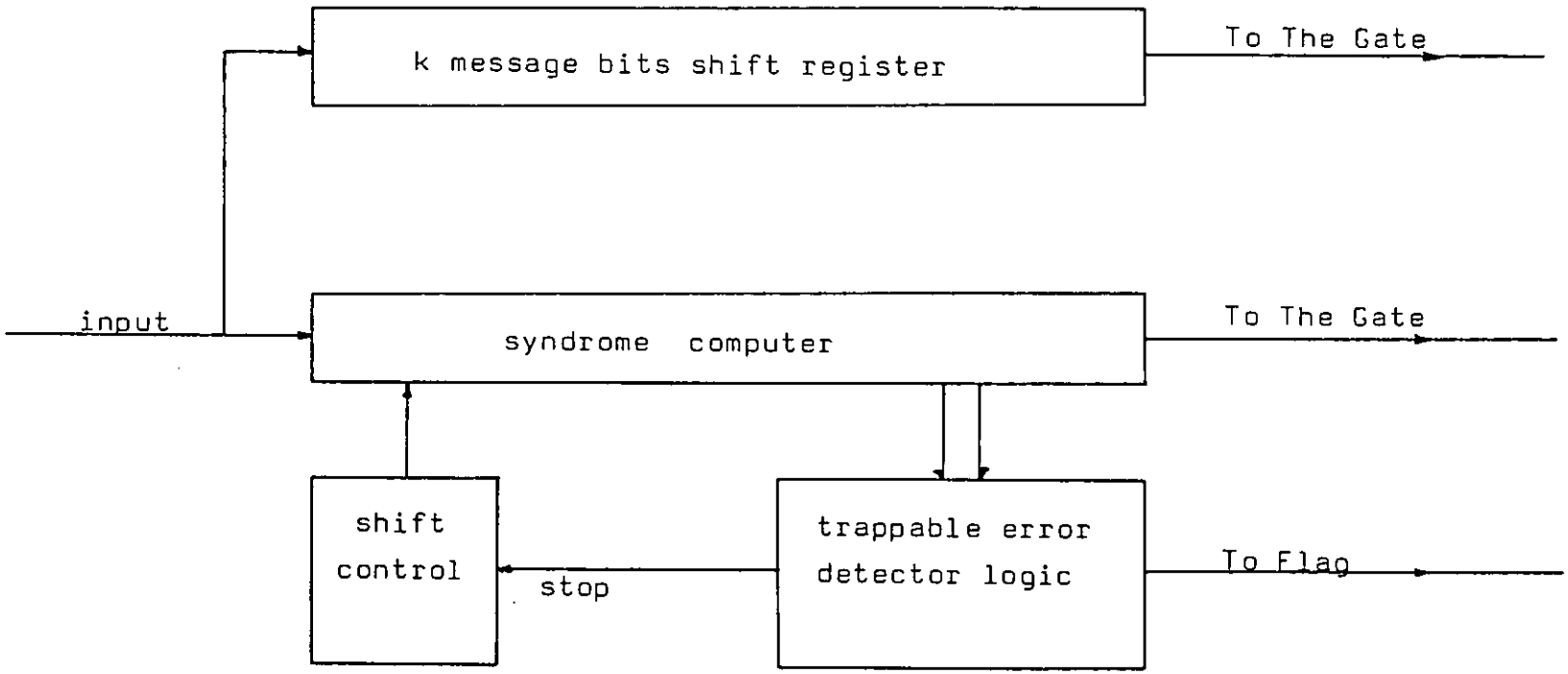


fig.(5.19) Block diagram of A Subdecoder

allows the gate to pass the zero volt subdecoder information digits and the error-pattern to the correction circuit for correction. On the other hand, if the zero volt flag is not set, the flag scan circuit starts to scan the flags according to the preset scan sequence, testing for a set flag, once detected, its message and error-pattern are passed to the correction circuit. If no flag is found set, a decoding failure is assumed. Once the correction is achieved, all flags are reset, and all syndrome registers are set to zero, so the decoder is ready for decoding the next received word.

5.10.2 Second Block Diagram

An alternative to the decoder described above is the decoder of fig.(5.20). This decoder differs from the previous one in the block arrangement, where the correction circuit is moved from the last stage of fig.(5.18), and placed in each of the subdecoders fig.(5.21). The information digits are stored and the syndrome is calculated as in the decoders of fig.(5.18). During the next n shifts, if an error-pattern is trapped, the flag is set, and the correction process is carried out. Otherwise the flag remains reset. Then unlike the typical error-trapping decoder, the corrected information digits are fedback to be stored in the k bit information register. At the end of the correction cycle the shift register should contain the information digits in the right order. Once the n syndrome shifts are finished, a correction time is allowed, then the flag scan and gate control tests the flags

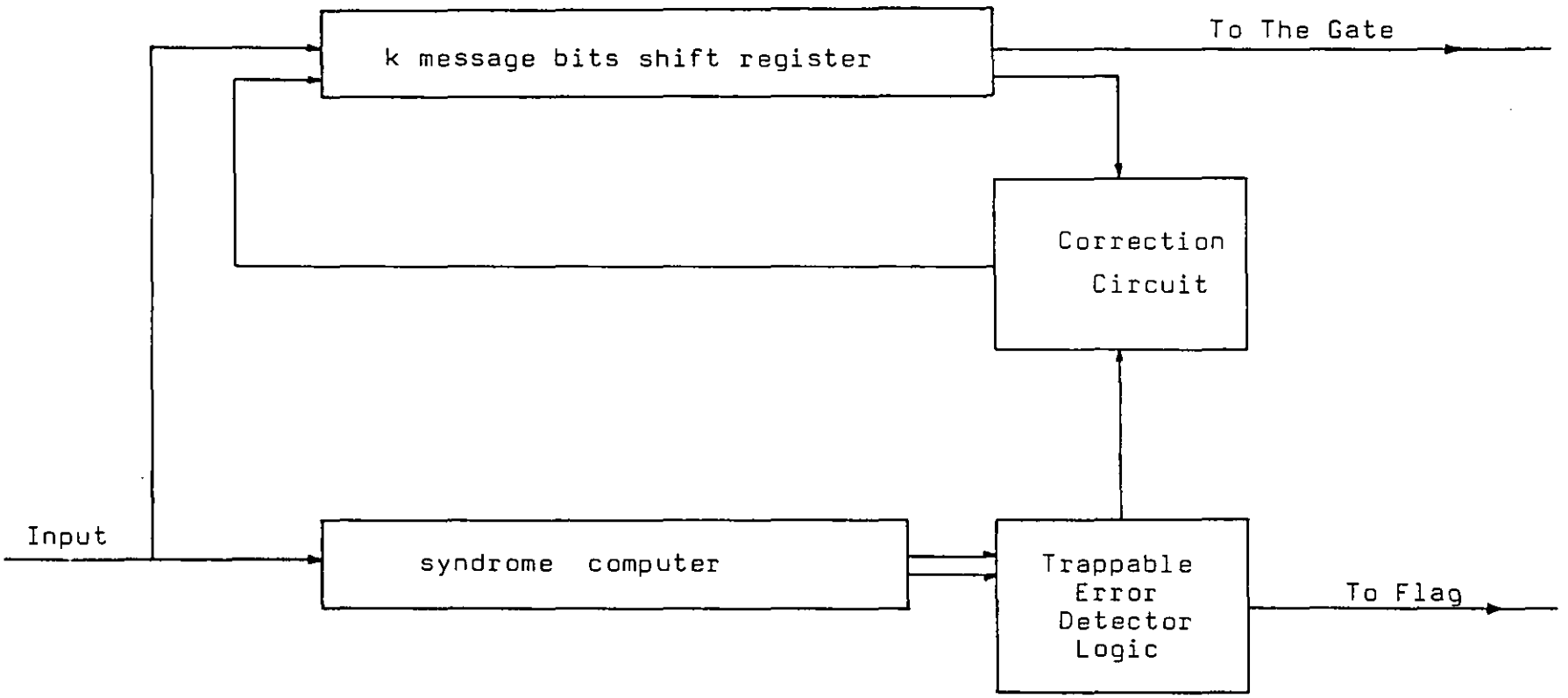


fig.(5.21) Block diagram of A Subdecoder

according to the preset sequence, once a set flag is found, its information digits are read out through the gate. If no set flag is found then a decoding failure is assumed. The last stage is to reset all flags, and set all syndromes to zero. Thus the decoder is ready for decoding the next received word.

The second decoder is more expensive because of the additional j-1 correction circuits, but assuming that the scan is done very fast and can be neglected compared to overall correction time, then the whole correction cycle is achieved in a constant time, thus synchronization is less complex.

5.11-The Statistical Parallel Threshold Decoder For Bursty Channels

The statistical decoders are of special interest, because their performance is better when the channel statistics match the decoder assumed statistics. The statistical parallel threshold decoder assumes that in the transmission channel, each burst of any given length is less likely than each burst of any shorter length. This information is used to choose the most likely error-pattern that is added to the received word in the channel.

The statistical parallel threshold decoder function can be described as the following. The sampled received word is fed to j thresholds, the output of each threshold is fed to a subdecoder, which generates all the possible error-patterns, and chooses the shortest error-pattern as the most likely error-pattern for the received word

detected at the particular value of the subdecoder threshold. At the end of the subdecoder detection cycle, the j syndrome registers will contain the shortest j error-pattern at the different values of thresholds. Since according to the assumed channel statistics, the shortest error-pattern are the most likely to occur, thus the decoder chooses the shortest error-pattern from these j error-patterns, and then the correction can be achieved accordingly.

Any subdecoder function is similar to the optimum decoder[33] described in chapter two, thus it can be said that the statistical parallel threshold decoder is the prallel decoder version of the optimum decoder.

An important point to notice is that the thresholds limits are of great importance to the statistical parallel threshold decoder performance. Where the limits are far apart the farthest thresholds will strat to introduce errors at the high SNR (the background noise), because although these thresholds are optimum or near optimum for detecting words received in the presence of burst noise, they are very far from optimum for detecting words received in the presence of background noise. In practice it was found that limiting the threshold values to ±0.5 V gives acceptable results.

5.11.1 Block Diagram of the Statistical Parallel Threshold Decoder

A j threshold statistical parallel threshold decoder consists of j branches of a similar threshold

and subdecoder circuit, decision and control circuit, and a correction circuit, as in fig.(5.22). The sampled received signal is fed to the j threshold detectors in parallel, the binary output of each threshold detector is fed to a subdecoder, fig.(5.23). At the subdecoder the information digits are stored in the k message bit shift register, simultaneously the syndrome is computed. The error-pattern length store is set to a number larger than n-k. The syndrome content length is compared with the error-pattern length store, if it is less, the syndrome contents are transferred to the shortest error-pattern store and its length is stored in the error-pattern length store. Otherwise the stored values remain unchanged, the syndrome is shifted n times and the test is repeated. At the end of the n^{th} shift the shortest error-pattern store contains the shortest burst, while the error-pattern length store contains its length. The decision and control circuit tests the error-pattern length stores for the shortest error-pattern, which is allowed to pass through the gate with its k information digits to the correction circuit, where the erroneous information digits are corrected.

Although all j stored error-patterns are tested, the test should be carried out in the threshold weight sequence, because in the case of two equal error-pattern lengths, the first one is kept in the store, thus when the scan is done according to the threshold weight, the error-pattern which has the higher weight of the equal error-patterns is used for correction. Other schemes can

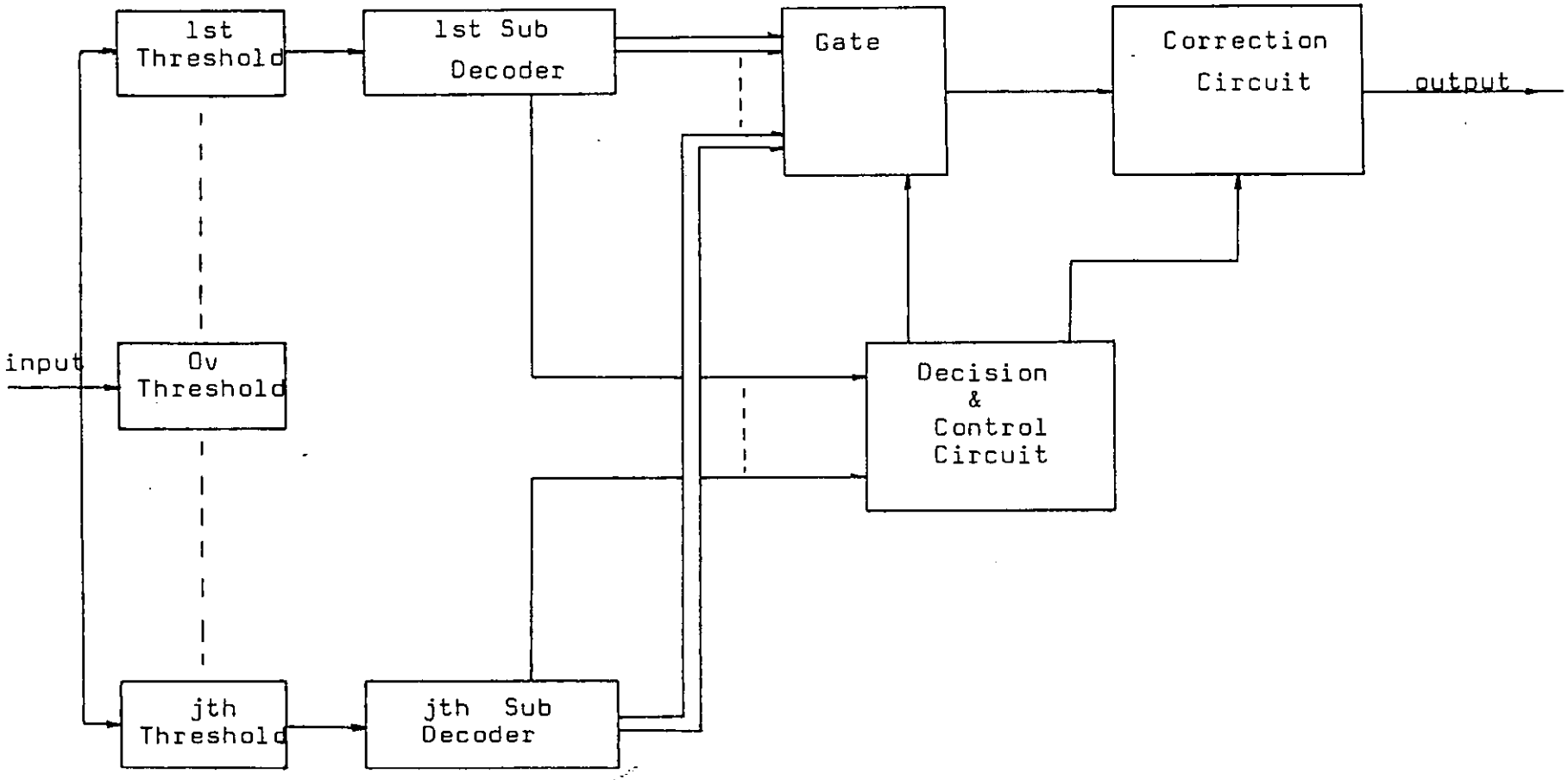


fig.(5.22) Block diagram of The SPTD

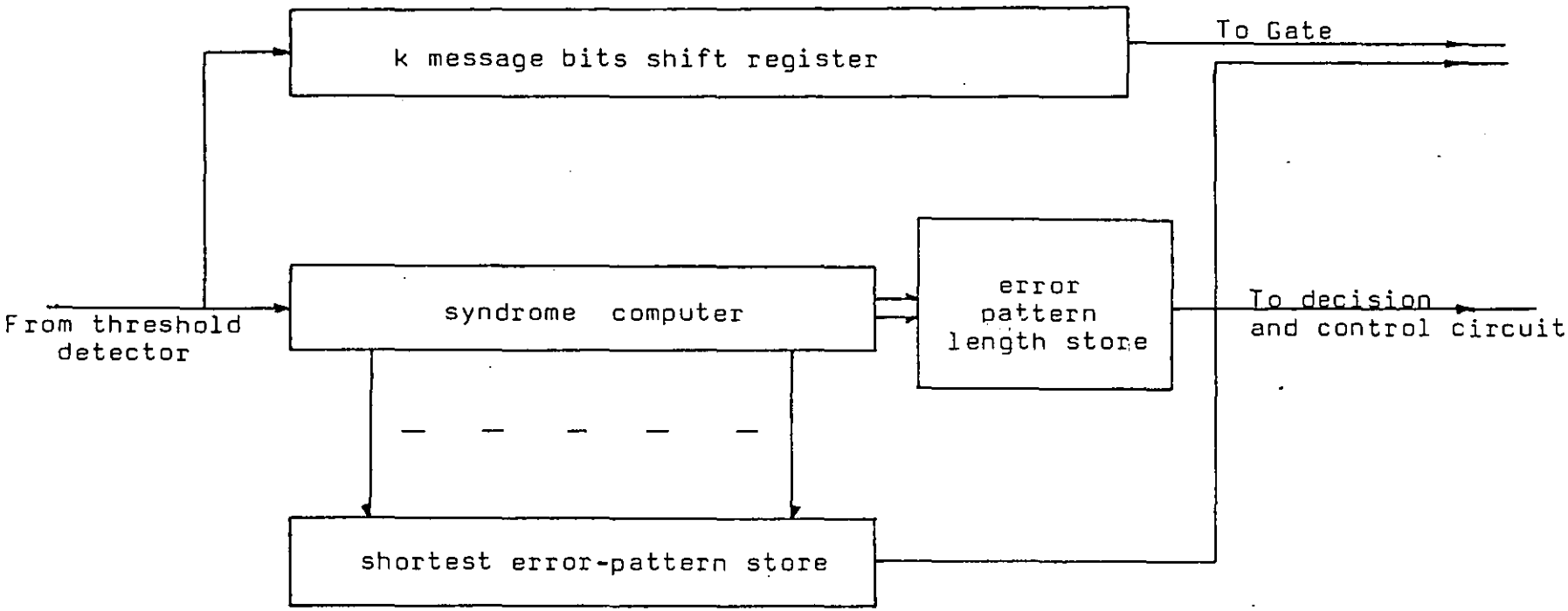


fig.(5.23) Bolck diagram of A subdecoder

be used for choosing one of two or more error-patterns of equal length, e.g. the error-pattern which has the lowest Hamming weight can be used, if the threshold weight method is not to be used.

5.12-Results and Discussions

The repetitive transmission of the same codeword does not give accurate results, when the parallel threshold decoders are used, because if an optimum threshold value happens to be used the results will be better than the actual performance. To eliminate such bias a random data is used for each codeword. This requires an additional complexity of the simulation program , where a random number generator is used to simulate the random information digits, and a simulated encoder to get the transmitted codeword.

Six codes were used to test the performance of the parallel threshold decoders. Two of them are random-error-correcting codes, the first is the BCH(31,21) code, which has a generator polynomial

$$g(X) = 1+X^3+X^5+X^6+X^8+X^9+X^{10} \qquad (5.94)$$

the second code is the BCH(15,7) code, which has a generator polynomial

$$g(X) = 1+X^4+X^6+X^7+X^8 \qquad (5.95)$$

Both these codes can correct two or less random errors in any codeword.The other four codes are single-burst-

error-correcting codes. The first one is the shortened (34,22) code, which is shortened from the (91,79) code, and can correct all bursts of ℓ=6 or less, thus it is an optimal code. Its generator polynomial is

$$g(X) = 1+X^2+X^3+X^4+X^5+X^6+X^9+X^{11}+X^{12} \qquad (5.96)$$

The second code is the (27,17) burst code, which is again an optimal code, and can correct all bursts of ℓ=5 or less. It is a shortened code derived from the (341,331) code, which has a generator polynomial

$$g(X) = 1+X^3+X^4+X^5+X^7+X^8+X^{10} \qquad (5.97)$$

The third code is the (19,11) burst code, which is an optimal code that can correct all bursts of length ℓ=4 or less, and is shortened from the (217,211) code, (see appendix B). The generator polynomial used is

$$g(X) = 1+X^2+X^4+X^7+X^8 \qquad (5.98)$$

the last code is the (27,20) burst code, which is not an optimal code, and can correct all bursts of length ℓ=3 or less. It is shortened from the (62,55) code, which has a generator polynomial

$$g(X) = 1+X^3+X^6+X^7 \qquad (5.99)$$

Symbol interlacing of degree λ=25 is used with all burst

codes to enhance the code performance. The interlacing degress has no effect on the decoder performance because in each case the results are compared with the corresponding results for the same code interlaced to the same degree. The simulation programmes are shown in appendix D.

5.12.1 Number Of Thresholds Choice

To study the effect of the number of thresholds on the decoder performance, two types of threshold spacing is used. the full span thresholds, and the practical spaced group. These two spacings are chosen because they represent the two extremes, as it can be seen from fig.(5.14). Consequently all other spacing schemes will fit in between these two spacings. This test is carried out for all codes using the corresponding parallel threshold decoder, while the test is carried out for all the burst codes using the statistical parallel threshold decoder for the practical spaced thresholds only, because as it was mentioned in section 5.10, the full span will result in a degraded performance. Typical specimen results are given, and if any code behaves differently its results will be shown as well.

5.12.1.1 The Full Span Threshold Spacing

The (31,21) random code is chosen to represent the typical performance of the parallel threshold decoder when a different full span threshold spacing is used. 3000 codewords are transmitted in each test, where the SNR value is changed from 1-8 dB, and for each test the threshold values are varied from 5-17 thresholds. The

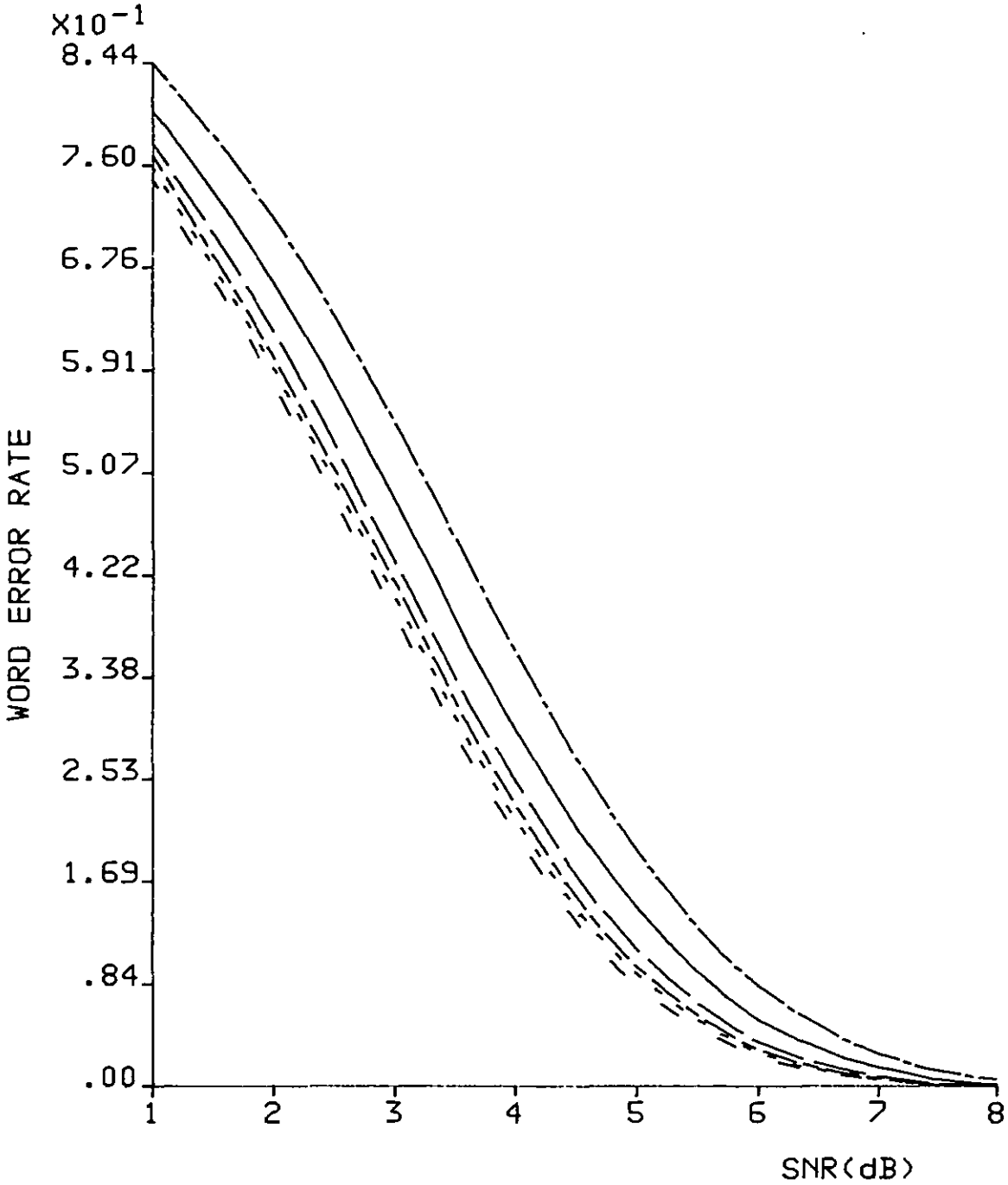
error rate is plotted against the SNR values for the different threshold values in fig.(5.24), in addition the performance of a conventional error-trapping decoder is included for comparsion.

It can be seen clearly that the parallel threshold decoder outperforms the conventional error-trapping decoder for any number of thresholds used. The improvement in performance increases with the increase of the number of thresholds used. A big improvement is added with an increase in the number of thresholds, while the use of more than 9 thresholds results in an additional smaller, and steady improvement with the increase of the thresholds number.

5.12.1.2 The Practical Spacing Thresholds

The (34,22) burst code decoded by a parallel threshold decoder is chosen to represent the parallel threshold decoder typical performance, when a practical threshold spacing is used. The background noise is assumed 9 dB, while the burst noise is changed between 6 dB and -20 dB. Here the treshold number is varied from 3 to 17. The results are shown in fig.(5.25). Again for comparsion, the conventional error-trapping decoder performance is plotted, as well as the parallel threshold decoder (PTD) using 17 full span spacing thresholds.

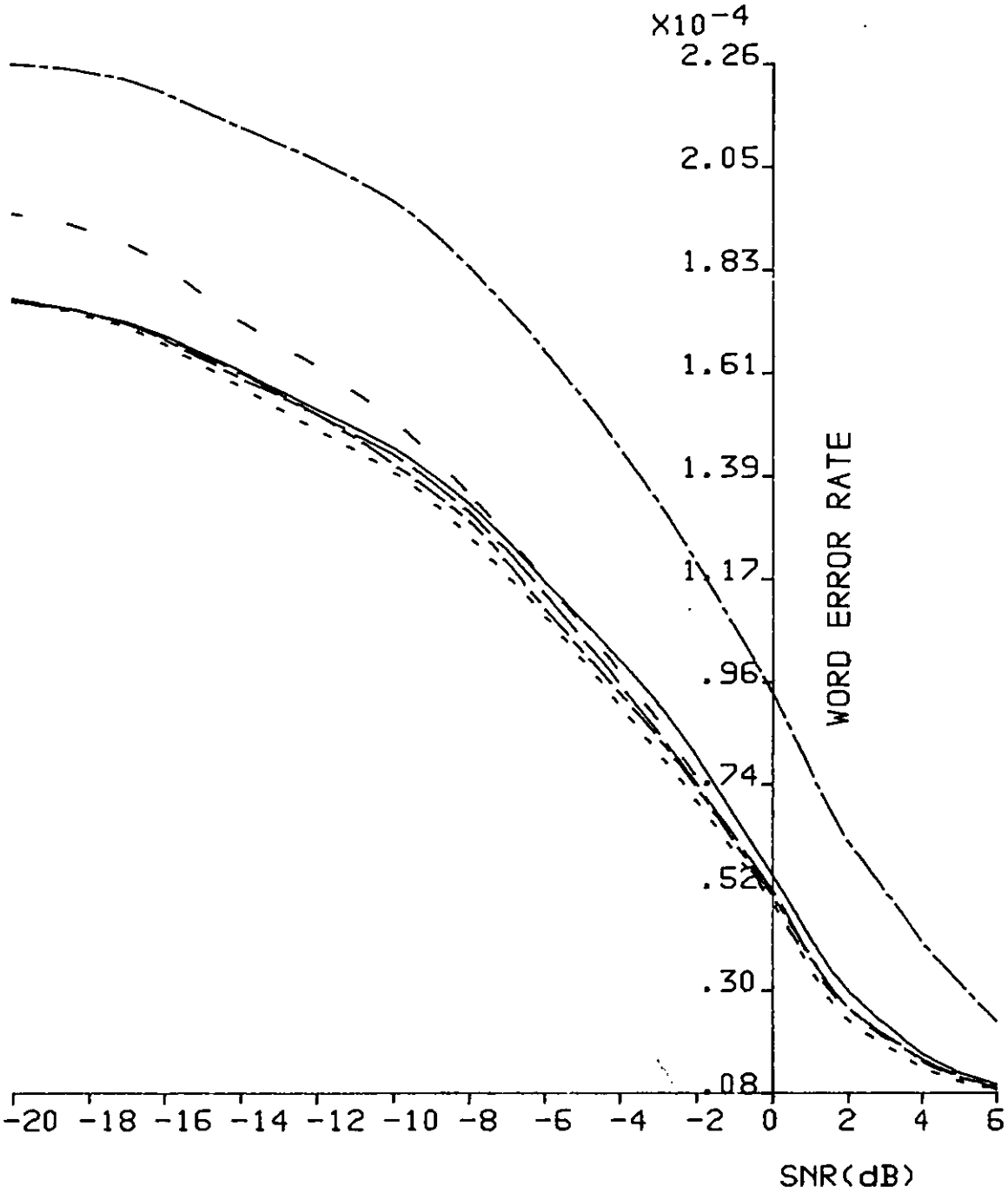
Fig.(5.25) shows clearly that the parallel threshold decoder performs better over the whole studied range, and that the practical spacing results in a lower additional improvement as the number of thresholds is increased, i.e.



____ 5 TH. ___ 9 TH. _ 17 TH.

___ 7 TH. - - - 11 TH. ___ E.T.

FIG.(5.24) FULL SPAN SPACING AT SNR= 9 dB



____ 3 TH. ___ 7 TH. _ _ 17 FULL SPAN

__ __ 5 TH. . . . 17 TH. ___ E.T.

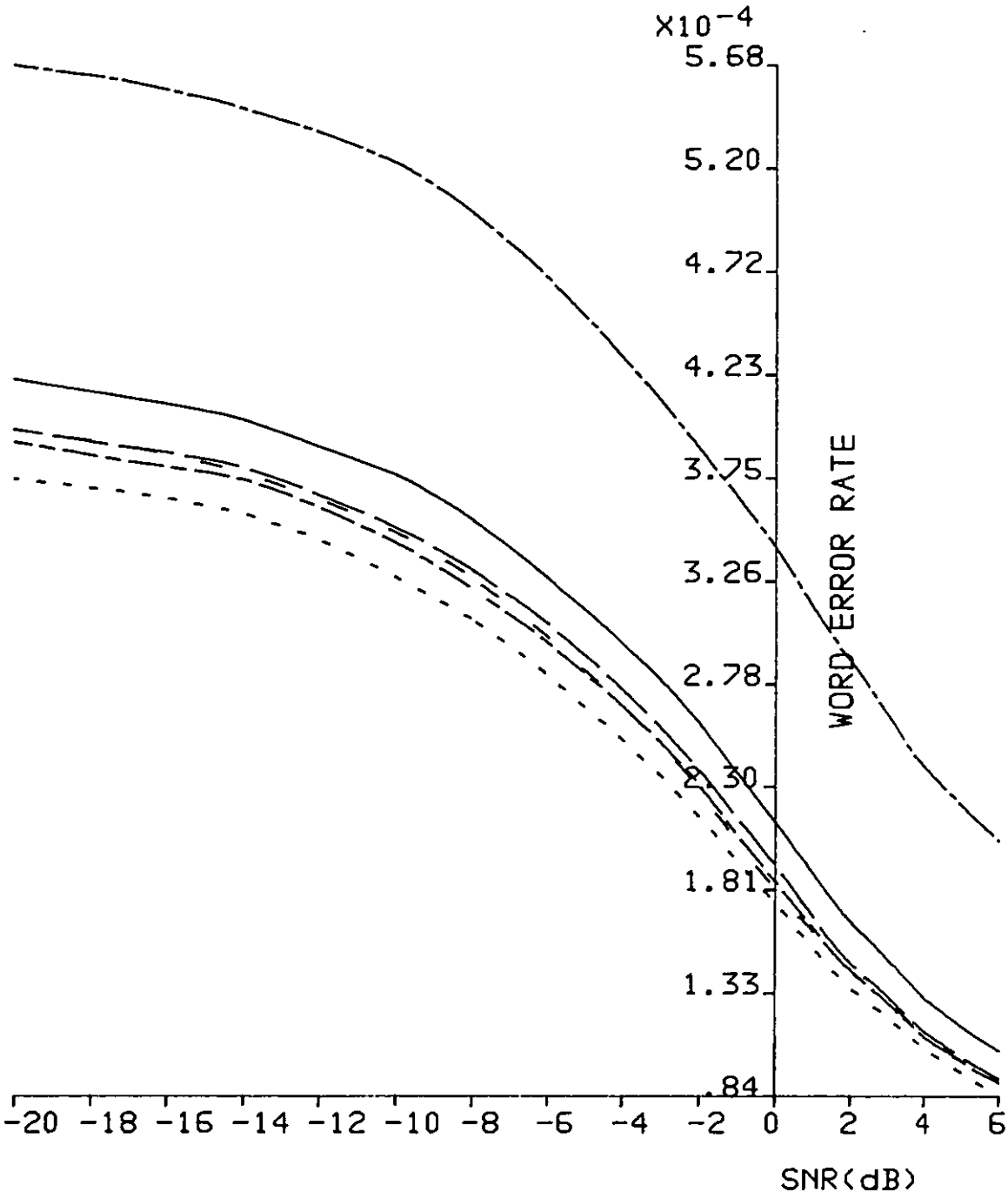
FIG.(5.25) PRACTICAL SPACING AT SNR= 9 dB

a big improvement is achieved when three thresholds are used, less additional improvement is achieved by going from three to five thresholds, while very little improvement can be achieved by going to more than five thresholds. As it is expected the practical spacing outperforms the full span spacing, and it can be seen that a parallel threshold decoder using three practical spaced threshold performs better than the same decoder using 17 full span spaced thresholds especially at low SNR burst noise values.

5.12.1.3 Effect Of Background Noise On The Number Of Thresholds

The effect of varying the burst noise on the parallel threshold decoder can be observed in fig.(5.25), but not the effect of varying the background noise. To study its effect the same graphs of fig.(5.25) are produced for two background noise values, fig.(5.26) is for background noise of 7 dB, while fig.(5.27) is for background noise of 11 dB.

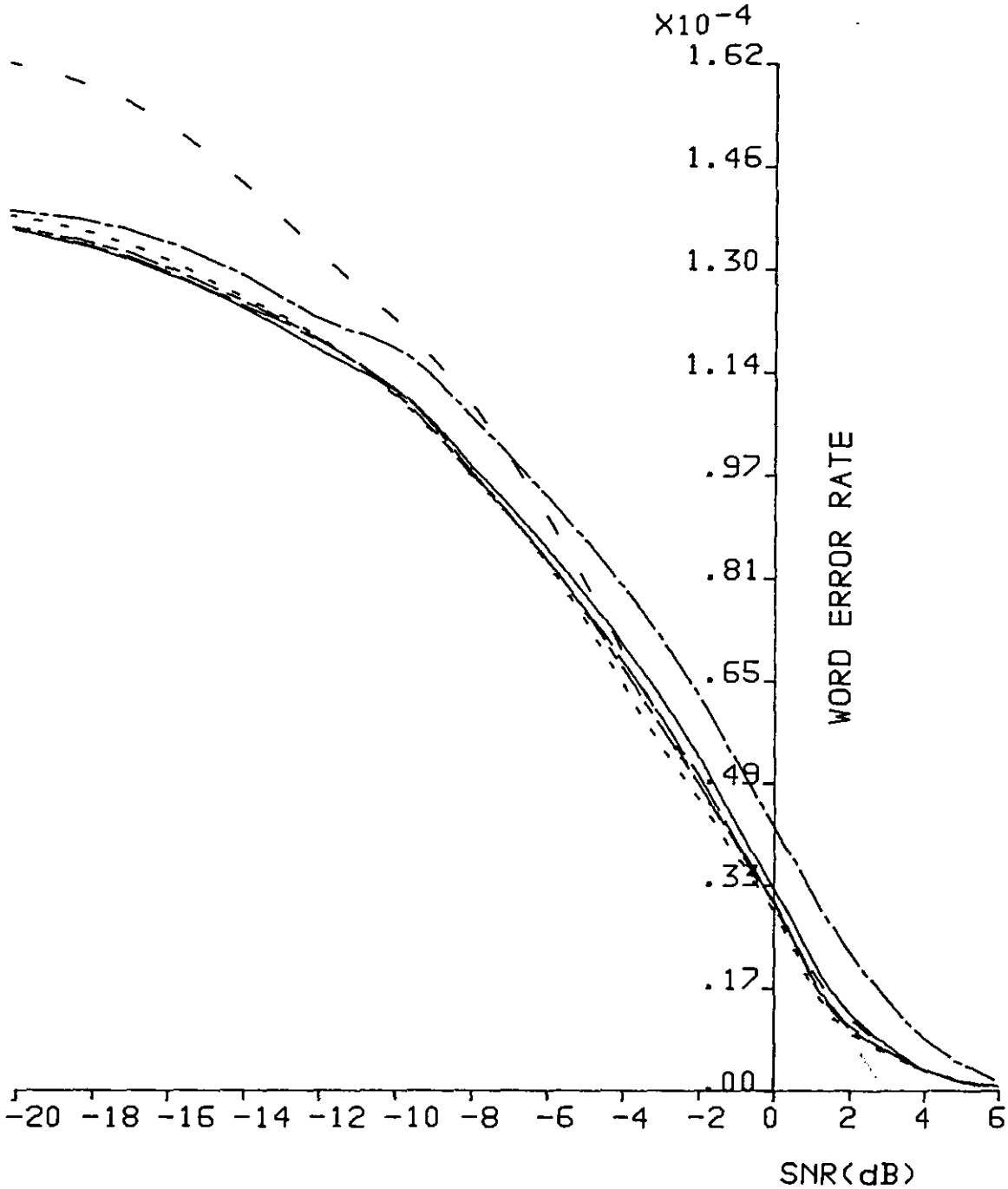
The results shown in fig.(5.26) confirm the results of fig.(5.25), the only difference is that the improvement for increasing the thresholds number from three to five is bigger than the improvement shown in fig.(5.25). Fig.(5.27) however shows a different result. The parallel threshold decoder using practical spaced thresholds is still performing better than the error-trapping decoder, but the improvement is very small when three thresholds were used, and hardly any additional improvement can be



____ 3 TH. _ _ _ 7 TH. _ 17 FULL SPAN

_ _ 5 TH. . . . 17 TH. ___ E.T.

FIG.(5.26) PRACTICAL SPACING AT SNR= 7 dB



____ 3 TH. __ _ 7 TH. _ 17 FULL SPAN

_ _ 5 TH. _ _ _ 17 TH. ____ E.T.

FIG.(5.27) PRACTICAL SPACING AT SNR= 11 dB

achieved by increasing the number of thresholds. On the other hand the parallel threshold decoder using 17 full span thresholds is performing worse than the error-trapping decoder despite the fact that the parallel threshold decoder is designed to perform the same as the error-trapping decoder in the worst case, as it is shown in section 5.9. In reality, the parallel decoder is not performing worse, but the reason for these results can be seen as the following.

The transmission channel is assumed to be a one way channel where no retransmission facility is available, furthermore the receiver is assumed to accept the information digits without correction in the case of decoding failures so that maximum improvement can be achieved. In the simulation, the received imformation digits only are compared with the transmitted digits, and any missmatch is counted as an erroneous word. Let the received word contain more than t errors for the random-errors decoder, and an error-pattern longer than ℓ for the burst correction channel. And let these errors be confined to the parity-check digits only. The error-trapping decoder will decode such a received word as a decoding failure, and will accept the information digits, the simulation programme will count this word as a correct word although it is a decoding failure. The parallel threshold decoder will try to convert this decoding failure to a trappable-error, if the conversion is successful the performance is the same, otherwise the error-trapping is decoding better for such received words.

Let the error probability in the transmission channel during the occurance of such a word be p, and the code used be an (n,k) code. Then the probability of decoding failure of correct information digits P_{cf} for the random-error channel is from Eq.(4.44)

$$P_{cf} = \sum_{i=t+1}^{n-k} \frac{(n-k)!(n-i)!}{n!(n-k-i)!} p^i (1-p)^{n-i} \qquad (5.100)$$

and P_{cf} for the burst error channel, from Eq.(4.49) is

$$P_{cf} = \left(\frac{1}{n} \sum_{i=1}^{n-k} p^i (1-p)^k\right) - \left(\frac{n-k-\ell+1}{n} \sum_{i=1}^{\ell} p^i (1-p)^{n-\ell}\right) \qquad (5.101)$$

this is present of course at low SNR value, but they are unnoticeable because of the superior performance of the parallel threshold decoders at lower SNR values. However, in the error-trapping decoders that disregard the received word whenever a decoding failure is detected, the parallel threshold decoder will perform equal or better whatever the SNR value.

At high background SNR values, the poor improvement over the error-trapping is justified, because at high background SNR values, there are only few random errors, hence the improvement is from detecting burst-errors correctly, which according to Eq.(5.93) is not as successful as detecting random errors.

The three figs.(5.25),(5.26),and (5.27), show that the improvement increases with the increase of the number of thresholds. For a background SNR of 11 dB 3 thresholds are sufficient to get most of the improvement that can be

achieved, while for background SNR of 9 dB, 5 thresholds is enough, and for background SNR of 7 dB, 5 threshods is acceptable, but 7 thresholds will give a little additional improvement. In general it can be said that, to get most of the improvement, the number of thresholds should be increased with the decrease of the background SNR value.

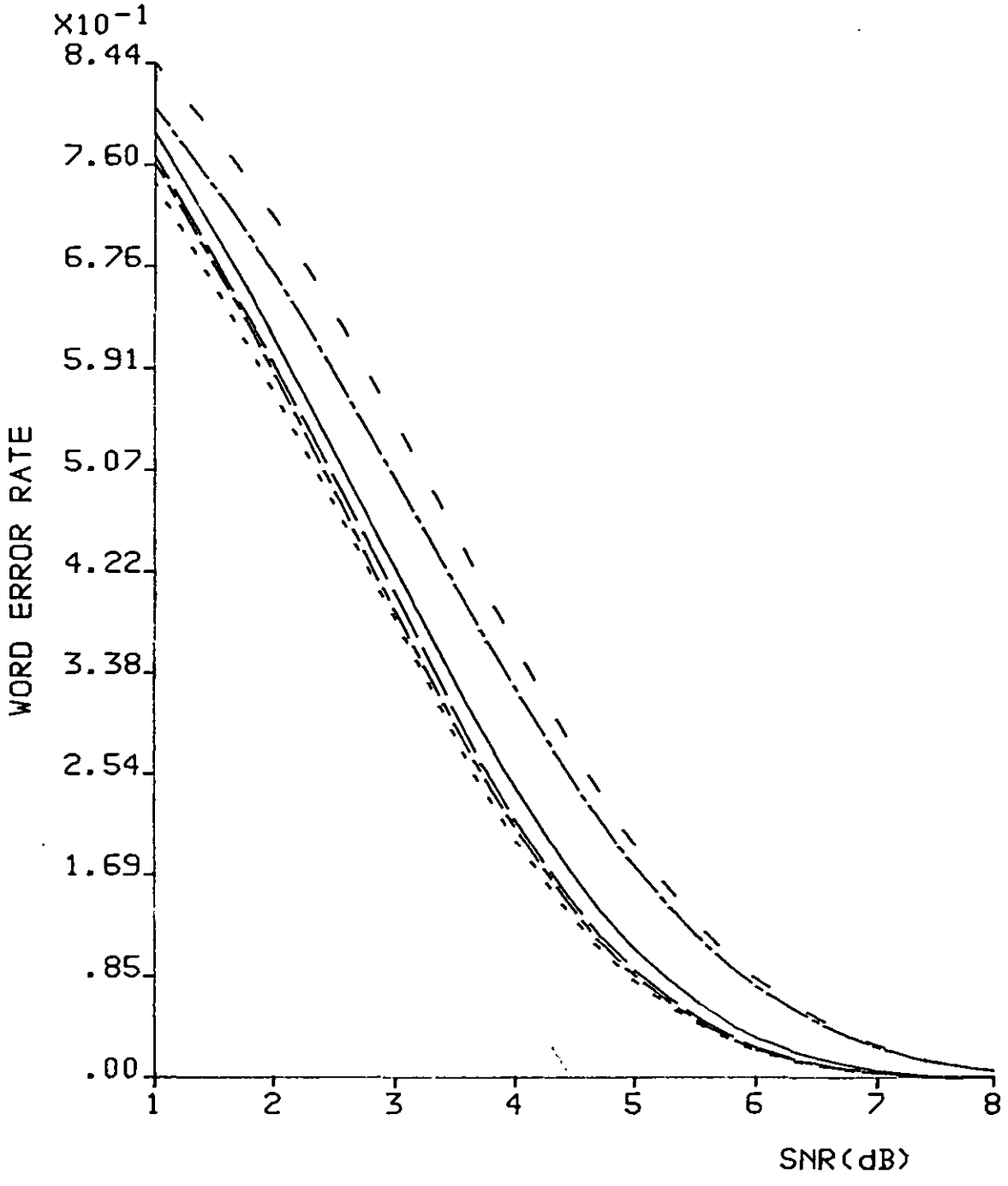
5.12.2 Random-Error Parallel Threshold Test Results

5.12.2.1 Simulation Results

Two codes were used in the simulation results, the BCH(31,21), and the BCH(15,7). The results given in fig.(5.28) and fig.(5.29) show the results of both codes, when 3,5,7, and 17 practical spaced thresholds are used for the detection. The simulation results for the conventional error-trapping decoder and soft-decision decoder are plotted for comparsion. The soft-decision decoder used a 16 level qunatizer, so that most of the possible improvement is achieved.

The (31,21) code performance of fig.(5.28), is self explanatory. Soft-decision decoding outperforms the error-trapping decoding and an improvement of 0.28 dB is obtained at SNR=3 dB, and 0.1 dB at SNR=6 dB. The parallel threshold decoder perform better than both, an improvement of 0.67 dB at SNR=3 dB, and 1 dB at SNR=6 dB is achieved over the error-trapping decoding, and 0.5 dB at SNR=3 dB, and 1 dB at SNR=6 dB over the soft-decision decoding, for the 5 threshold parallel threshold decoder.

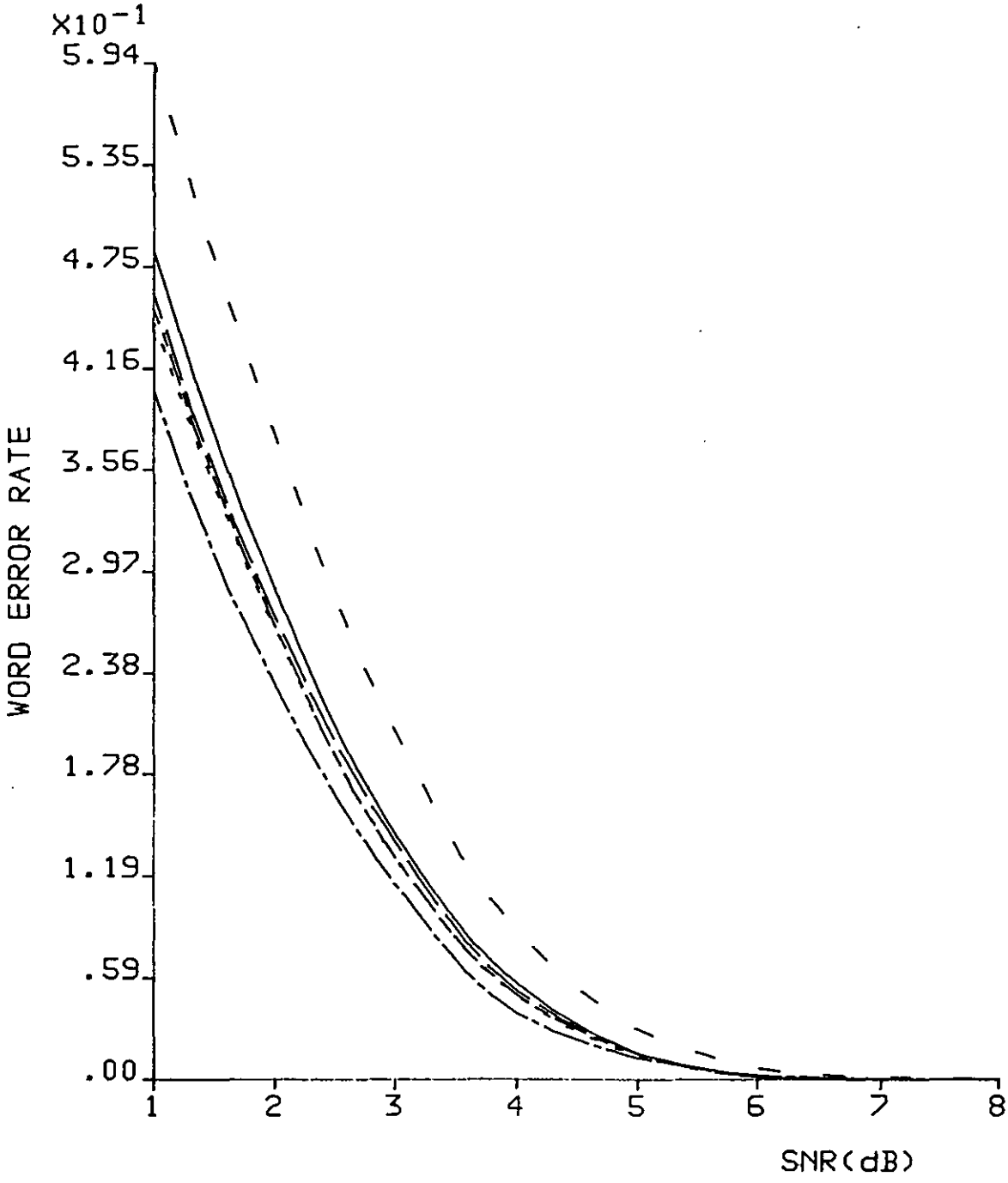
The (15,7) code performance, fig.(5.29) is different in that the error-trapping decoder using this code will trap all correctable errors, hence any decoding failure



—— 3 TH. — — 7 TH. - - E.T.

— 5 TH. - - - 17 TH. —— S.D.

FIG.(5.28) PTD PERFORMANCE USING BCH (31,21) CODE



____ 3 TH. —.— 7 TH. _ E.T.

_ _ 5 TH. - - - 17 TH. ___ S.D.

FIG.(5.29) PTD PERFORMANCE USING BCH (15,7) CODE

will guarantee that the received word contains at least 3 errors. The soft-decision decoding has an advantage in such a case, since all correctable errors are trappable, then the soft-decision decoder will calculate the EPSW for all the possible error-patterns, and not for the trappable error-patterns only, which give the soft-decision decoder a superior performance over codes in which the n-k parity-check digits cannot trap all correctable errors. On the other hand, the parallel threshold decoder is at a disadvantage, because as it is mentioned in section 5.9.1, the only source of improvement for such codes, is for the parallel threshold decoder to detect correctly all the received errors but two, in the received word. Consequently the poor performance of the parallel threshold decoder for the (15,7) code is not surprising. Fig.(5.29) shows that the parallel threshold decoding performs better than the error-trapping decoding, an improvement of 0.5 dB is achieved at SNR=3 dB, and 1.5 dB at SNR=6 dB when a 5 thresholds decoder is used. While the overall soft-decision decoding outperforms both decoding methods, where an improvement of 0.9 dB is achieved at SNR=3 dB, and 1.5 dB at SNR=6 dB over error-trapping decoding. And 0.2 dB at SNR=3 dB, and no improvement at SNR=6 dB over the parallel threshold decoding.

5.12.2.2 Hardware Test Results

A hardware parallel threshold decoder has been built and tested. Three and five practical thresholds

were used, and the test results are very similiar to those obtained by the simulation. The hardware and simulation results, for errer-trapping decoding, 3 and 5 thresholds parallel threshold decoder are shown in fig.(A.8). The circuit diagram, and a full discussion of the parallel threshold decoder function, and the results are included in appendix A.

5.12.3 Burst-Error Decoding Results

The simulation results considered here will be for the parallel threshold decoding, and the statistical parallel threshold decoding. In each case the four burst codes are used. The background SNR values is taken as 9 dB during all the tests, and an interlaceing degreee λ=25 is taken for all codes. To assess the performance of the decoding, the error-trapping decoding, the optimum decoding, and the soft-decision decoding is plotted, with the performance of 3, 5, and 17 practical threshold spacing parallel threshold decoding results.

5.12.3.1 Parallel Threshold Decoding Results

The first code to be considered is the (34,22) code, its performance results are shown in fig.(5.30). As it is expected the optimum decoding performs better than the error-trapping decoding, because the assumed channel statistics match the decoding statistics. The average improvement over the studied range is about 5 dB. Soft-decision decoding does not give any improvement over the error-trapping decoding for burst SNR values lower than -7.4 dB, and lower than -4.25 dB for the optimum decoding. On the other hand the parallel threshold decoding outperforms

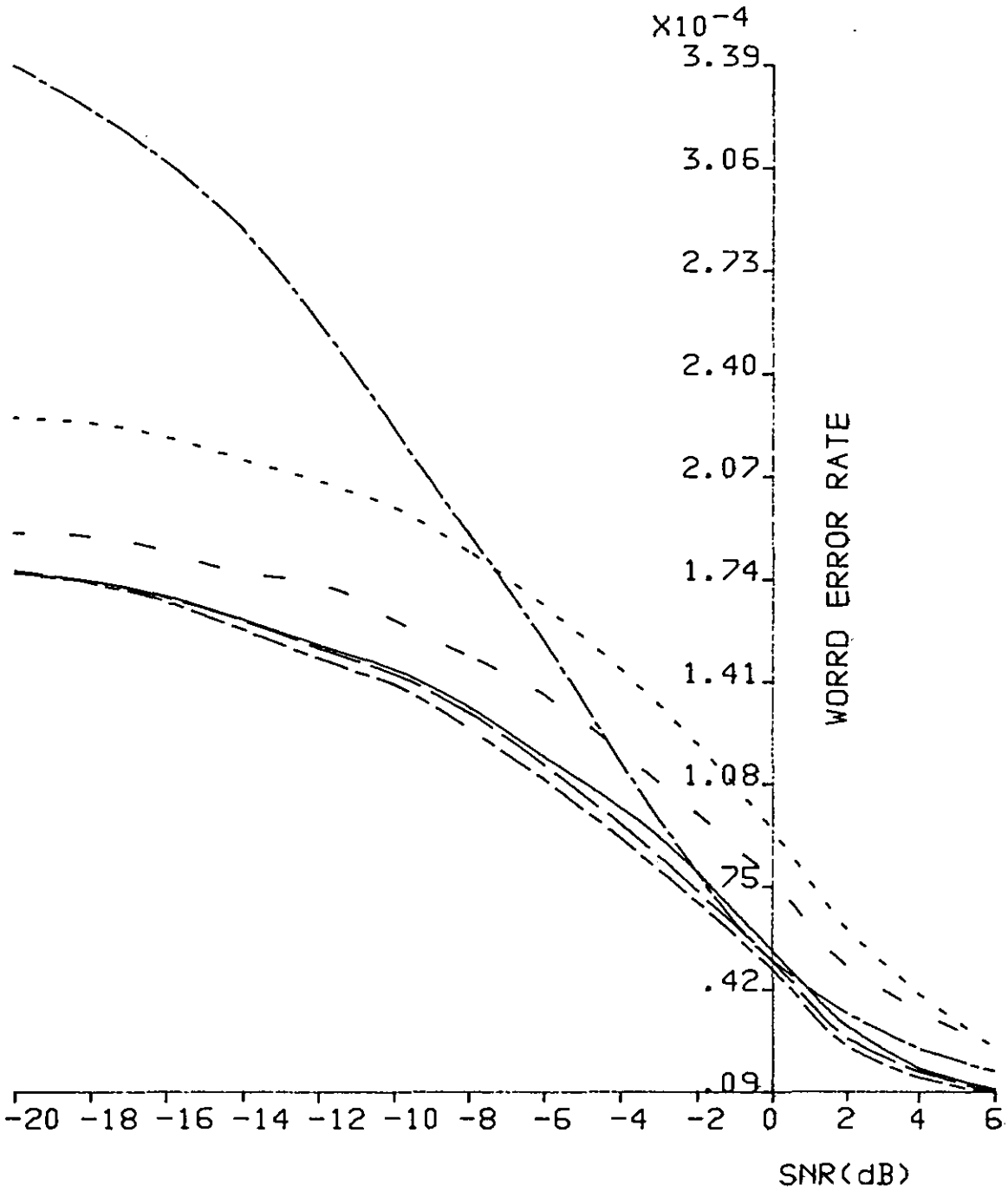


FIG.(5.30) PTD PERFORMANCE USING BURST (34,22) CODE

all other decoding methods. At burst SNR = -3 dB the five thresholds parallel threshold decoding, improves by 3 dB, 2.25 dB, and 0.7 dB, over error-trapping, optimum, and soft-decision, respectively, while at burst SNR=-17 dB, the improvement is 10.5 dB, 5.4 dB, and 10 dB.

Using the (27,17) code, the results of fig.(5.31) show similar performance to the (34,22) code, at burst SNR=-3dB a five threshold PTD achieved an improvement of 3.3 dB, 1.16 dB, and -1 dB, over error-trapping, the optimum, and soft-decision decoding respectively, while at burst SNR=-17 dB the improvement is 9 dB, 3.15 dB, and 7.8 dB.

The (19,11) is a powerful code and because of its low code rate only a few correctable errors are untrappable. This can be seen from fig.(5.32), where the word error rate is lower then the two previous codes, thus under these conditions the parallel decoder using this code is not expected to give a big improvement. The optimum decoding gives a small improvement over the error-trapping decoding, while the soft-decision is worse most of the range. A five thresholds PTD at burst SNR=-3 dB performs slightly better, the improvement is 1.1 dB, 0.34 dB, and 1.27dB, over error-trapping, the optimum, and soft-decision decoding respectively, while at burst SNR=-17 dB, the PTD achieves an improvement of 9.7 dB over the soft-decision decoding, and 0.36 dB over error-trapping, but introduces a degradation of 9.7 dB over the optimum decoder.

The PTD using the (27,20) code does not achieve a big improvement, the reason being that the (27,20) is

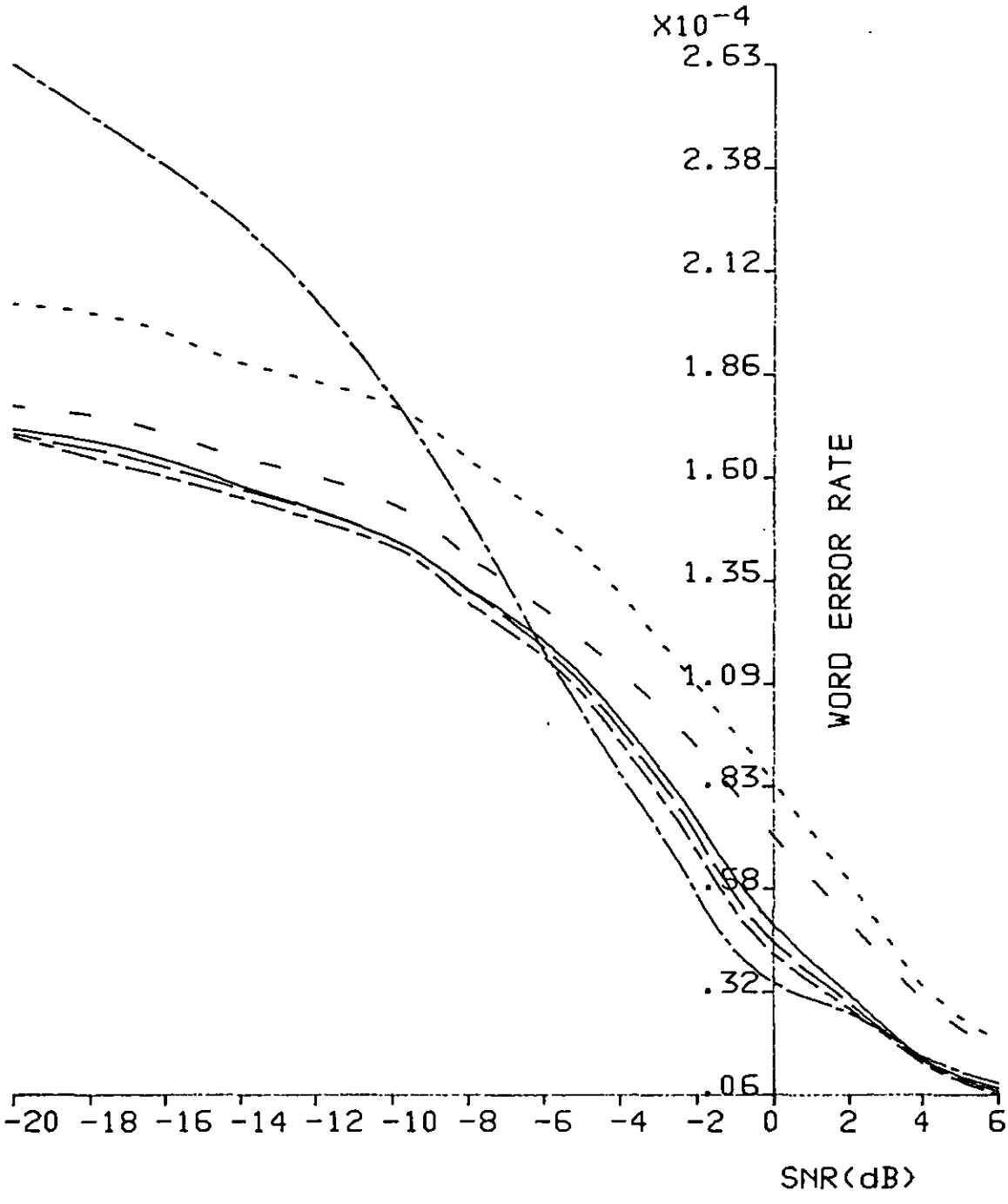


FIG.(5.31) PTD PERFORMANCE USING BURST (27,17) CODE

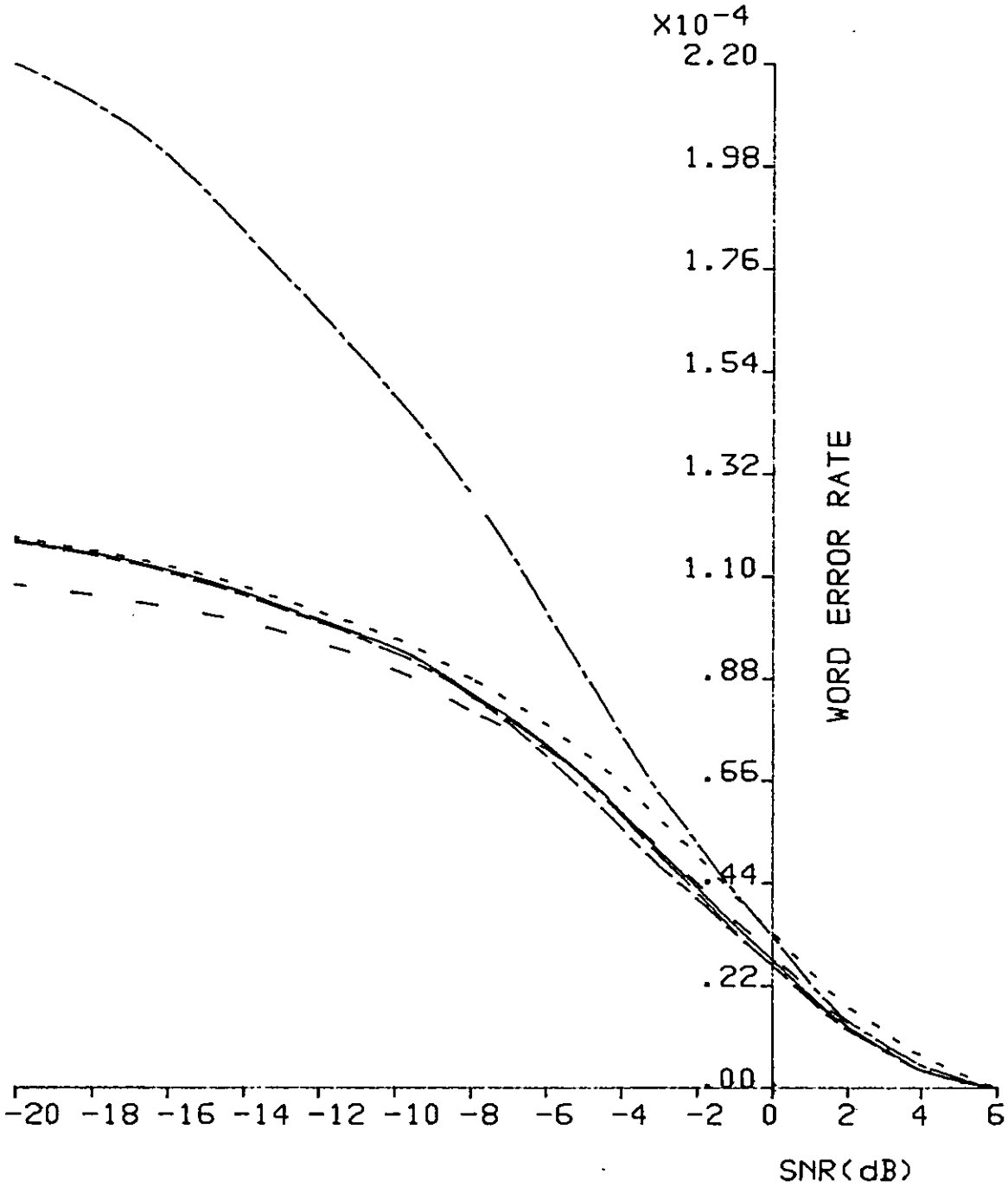


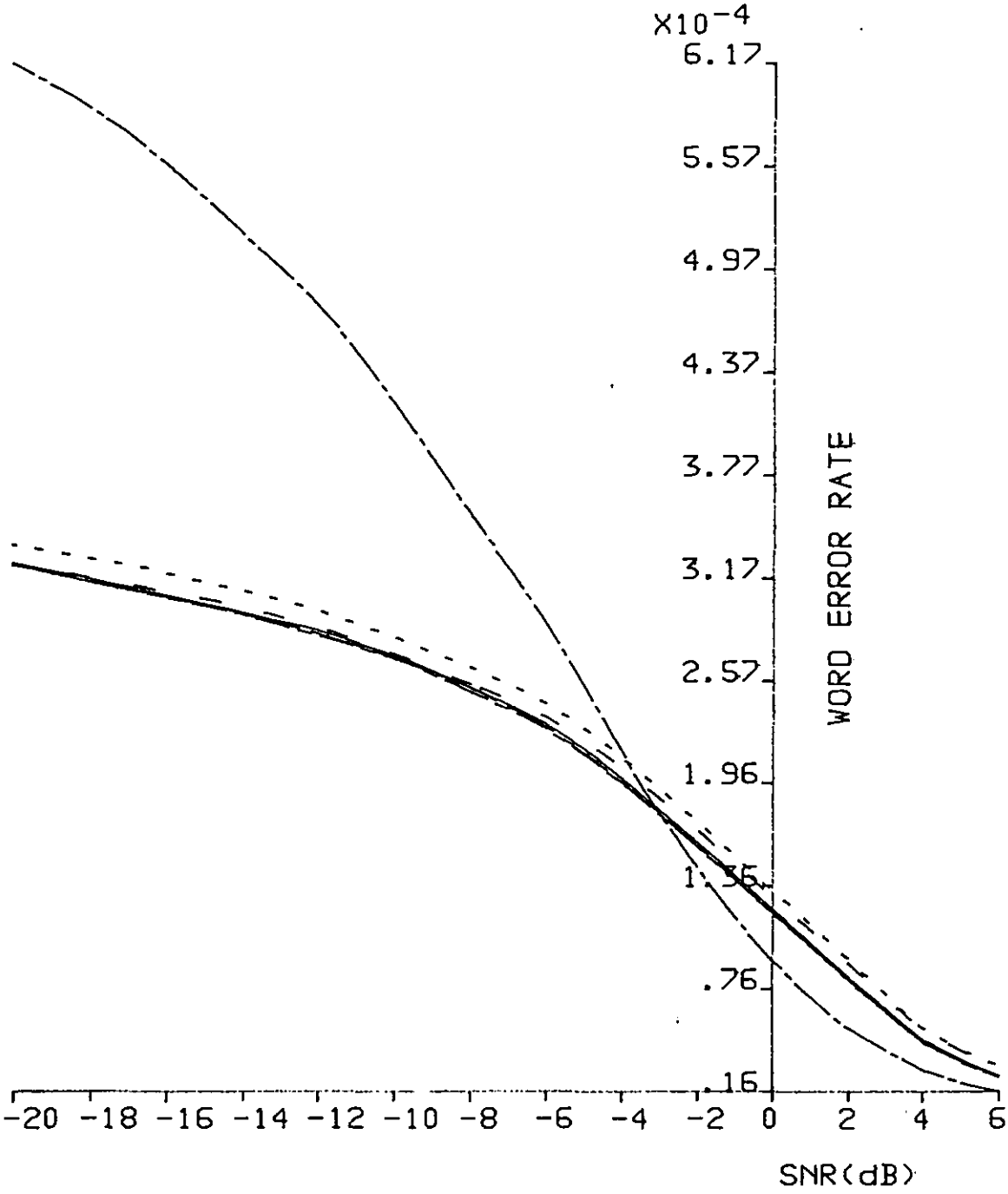
FIG.(5.32) PTD PERFORMANCE USING BURST (19,11) CODE

not a powerful code, due to its low correction ability, as it is seen from fig.(5.33) where the word error rate is high. A five threshold PTD achieves at burst SNR=-3 dB an improvement of 0.7 dB, 0.5 dB, and 0 dB over error-trapping, the optimum, and soft-decision decoding respevtively, while at burst SNR=-17 dB the improvement is 2.5 dB, 0.3 dB, and 10.3 dB.

5.12.3.2 The Statistical Parallel Threshold Decoder (SPTD)

The decoding performance for the (34,22) code is shown in fig.(5.34). As it is expected the SPTD performs even better than the PTD, because the assumed channel statistics matches the decoder statistics. The five thresholds SPTD outperforms the error-trapping, the optimum, and soft-decision decoding by 4.7 dB, 3.9 dB, and 2.1 dB, at burst SNR=-3 dB, and by 13.3 dB, 10.7 dB, and 11.7 dB, at burst SNR=-17 dB, respectively. From a comparsion of fig.(5.34) and fig.(5.30) it can be seen that the SPTD introduces an additional improvement of 1.7 dB, at burst SNR=-3 dB, and 6.4 dB, at burst SNR=-17 dB, over the five thresholds PTD.

The decoding performance using the (27,17) code is very similar to the (34,22) performance fig.(5.35). The five thresholds SPTD gives an additional gain over error-trapping, the optimum, and soft-decision decoding of 3.78 dB, 3 dB. and 0 dB, at burst SNR=-3 dB, and 11.8 dB, 8.4 dB, and 9.3 dB, at burst SNR=-17 dB respectively. Again the improvement over the PTD can be seen from comparing fig.(5.35) and fig.(5.31), where a gain of 7.1 dB, and 0.6 dB, is achieved at burst SNR=-3 dB and



____ 3 TH. ___ 17 TH. _ OPT.

_ _ 5 TH. . . . E.T. ___ S.D.

FIG.(5.33) PTD PERFORMANCE USING BURST (27,20) CODE

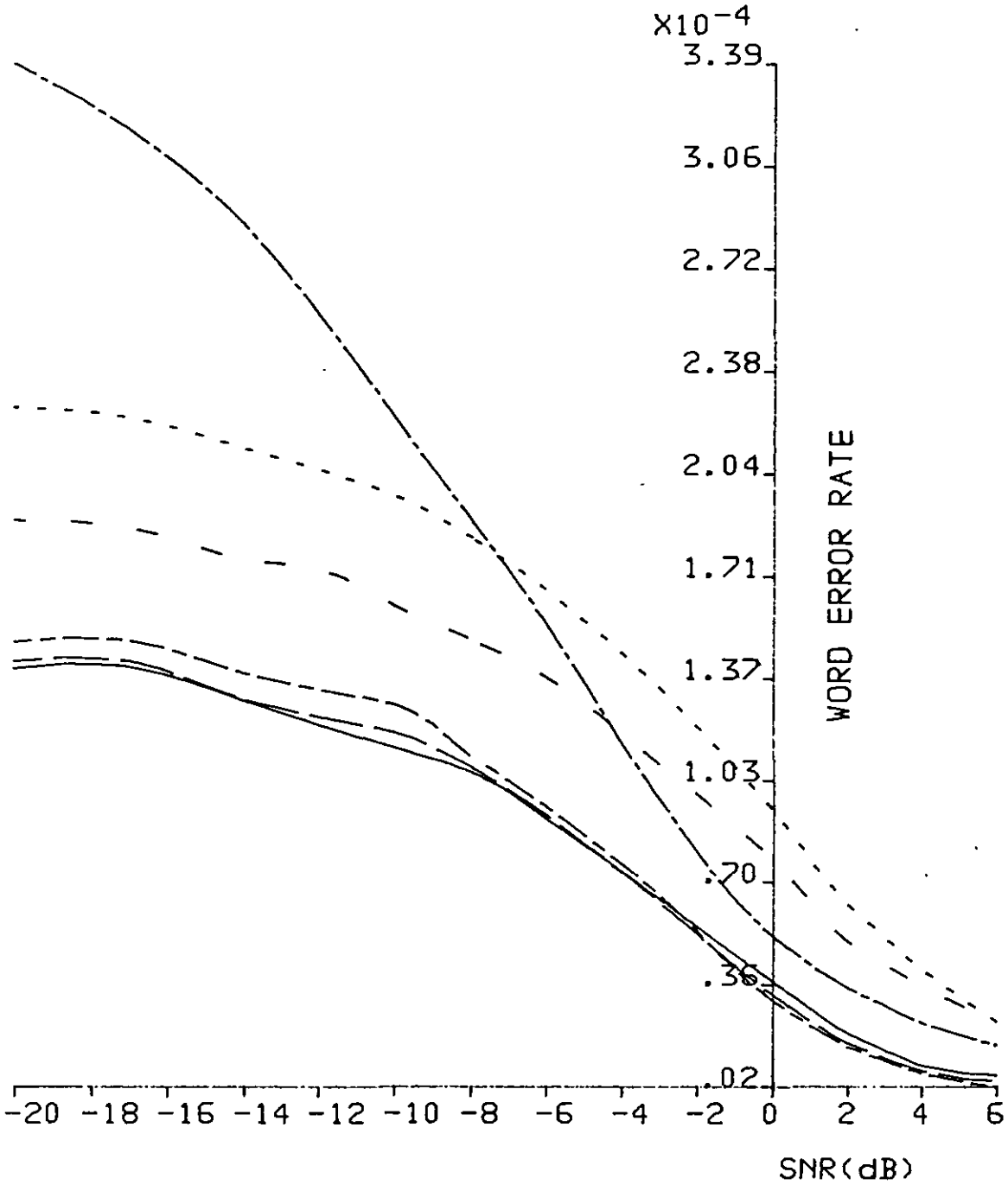


FIG.(5.34) SPTD PERFORMANCE USING BURST (34,22) CODE

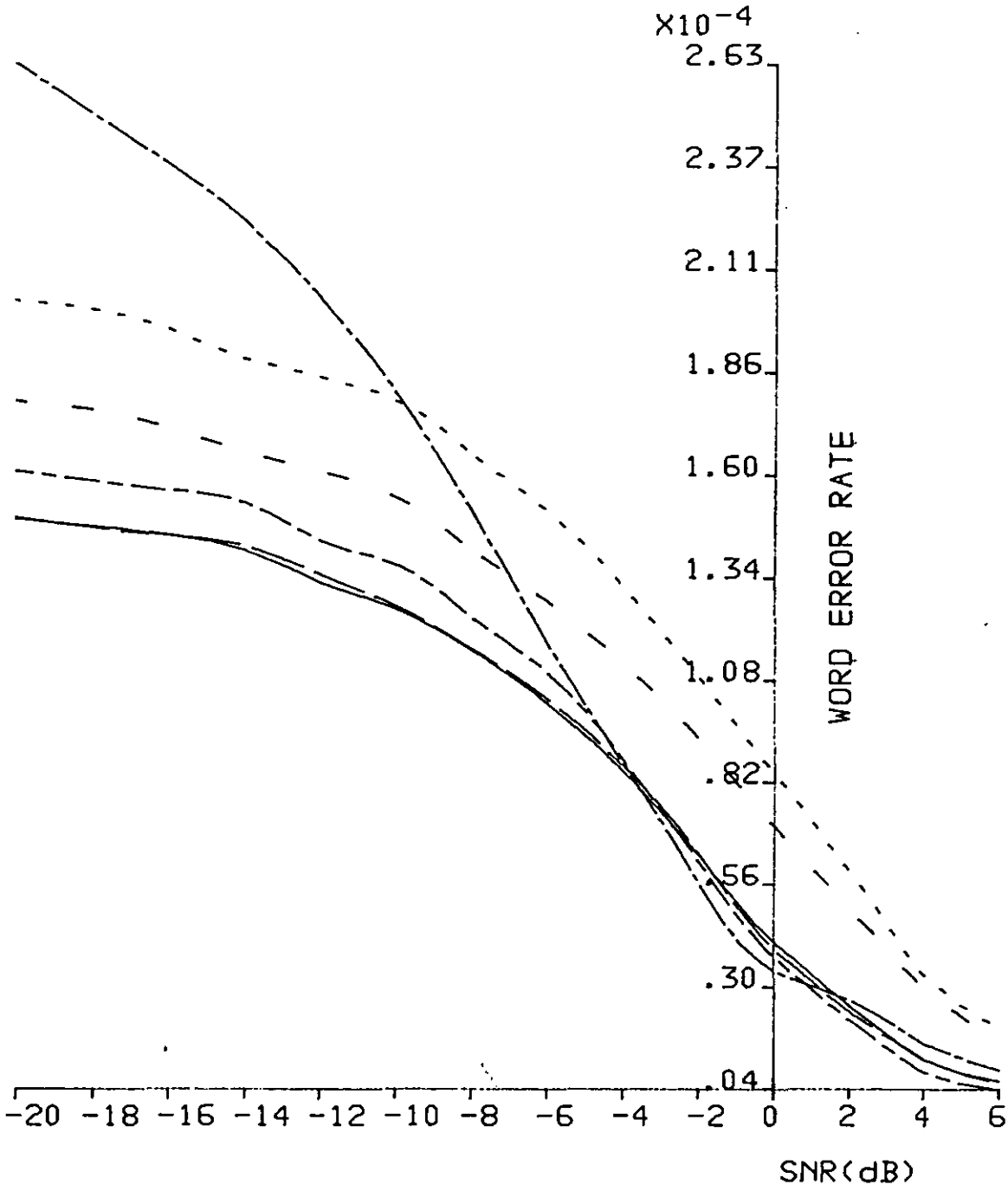
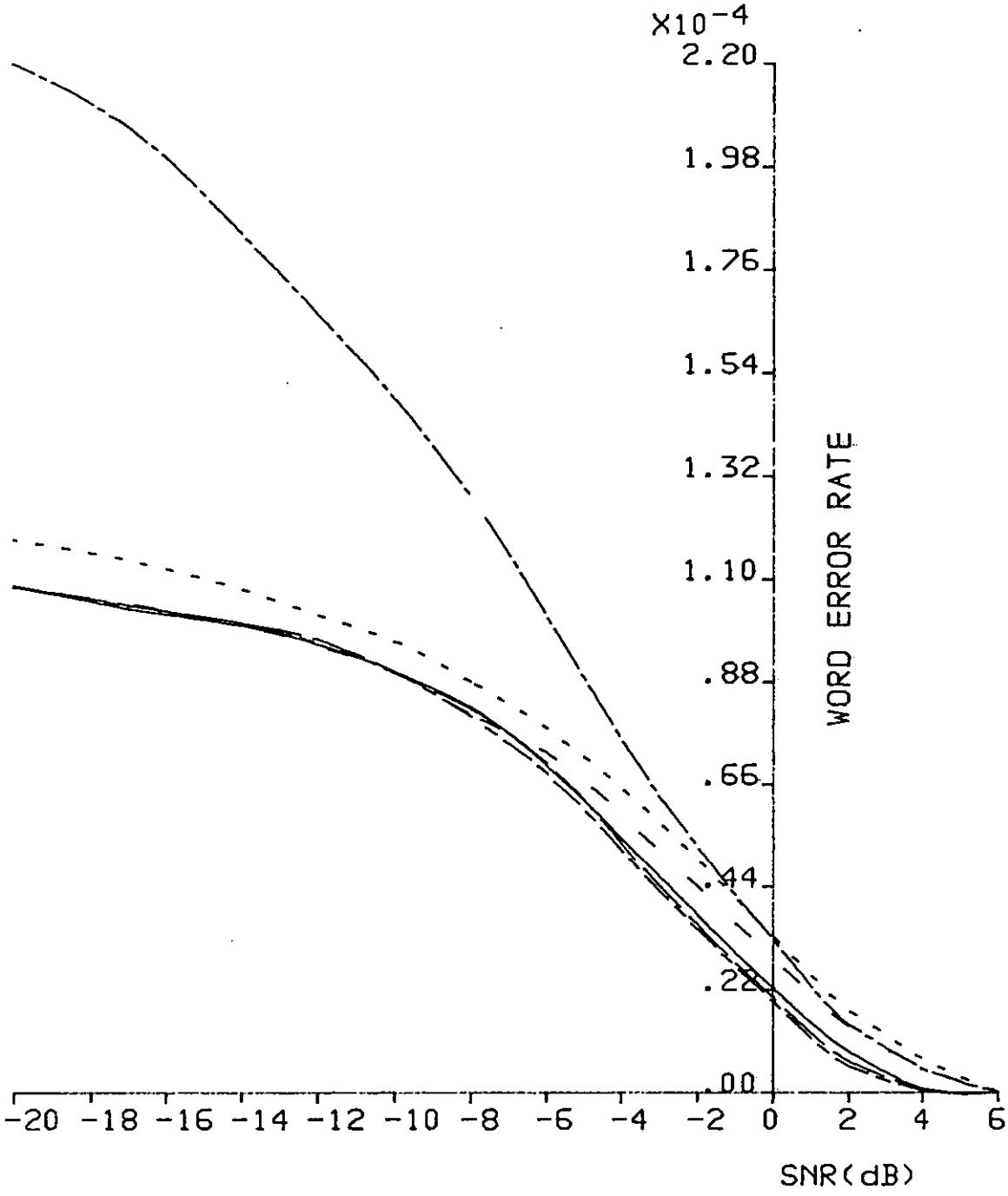


FIG.(5.35) SPTD PERFORMANCE USING BURST (27,17) CODE

SNR=-17 dB respectively.

The SPTD performs poorly when the (19,11) code is used, for the same reasons that cause the PTD poor performance. The SPTD same as the PTD introduces some improvement over the optimum decoding for the burst SNR>-6 dB, while hardly any improvement for burst SNR<-6 dB as in fig.(5.36). A five threshold SPTD achieves an additional improvement of 1.8 dB, 1.15 dB, and 2 dB, at burst SNR=-3 dB, and 4.4 dB, 0 dB, and 10.8 dB, at burst SNR=-17 dB over error-trapping, the optimum and soft-decision decoding, while the improvement over the PTD is 0.9 dB, and 3.9 dB, at burst SNR=-3 and -17 dB respectively.

On the other hand, the performance of the SPTD is very poor when the (27,20) code is used as it is shown in fig.(5.37). A five thresholds SPTD introduces a very small improvement at low burst SNR values, and a degradation at very low burst SNR values except over the soft-decision decoding. The improvement at burst SNR=-3 dB is 1.6 dB, 1.4 dB, and 0.6 dB, over error-trapping, the optimum, and soft-decision decoding. While at burst SNR=-17 dB, the only improvement is over the soft-decision, which is 9.8 dB, the degradation over error-trapping is 2.4 dB, and over the optimum decoding is 4.2 dB. The SPTD introduces a reasonable gain over the PTD at burst SNR>-8 dB, but it performs worse for a burst SNR<-8 dB. For example, the improvement at burst SNR=-3 dB is 0.8 dB, while the degradation at burst SNR=-17 dB is 3.9 dB.



____ 3 TH. __ __ 17 TH. _ OPT.

_ _ 5 TH. . . . E.T. ___ S.D.

FIG.(5.36) SPTD PERFORMANCE USING BURST (19,11) CODE

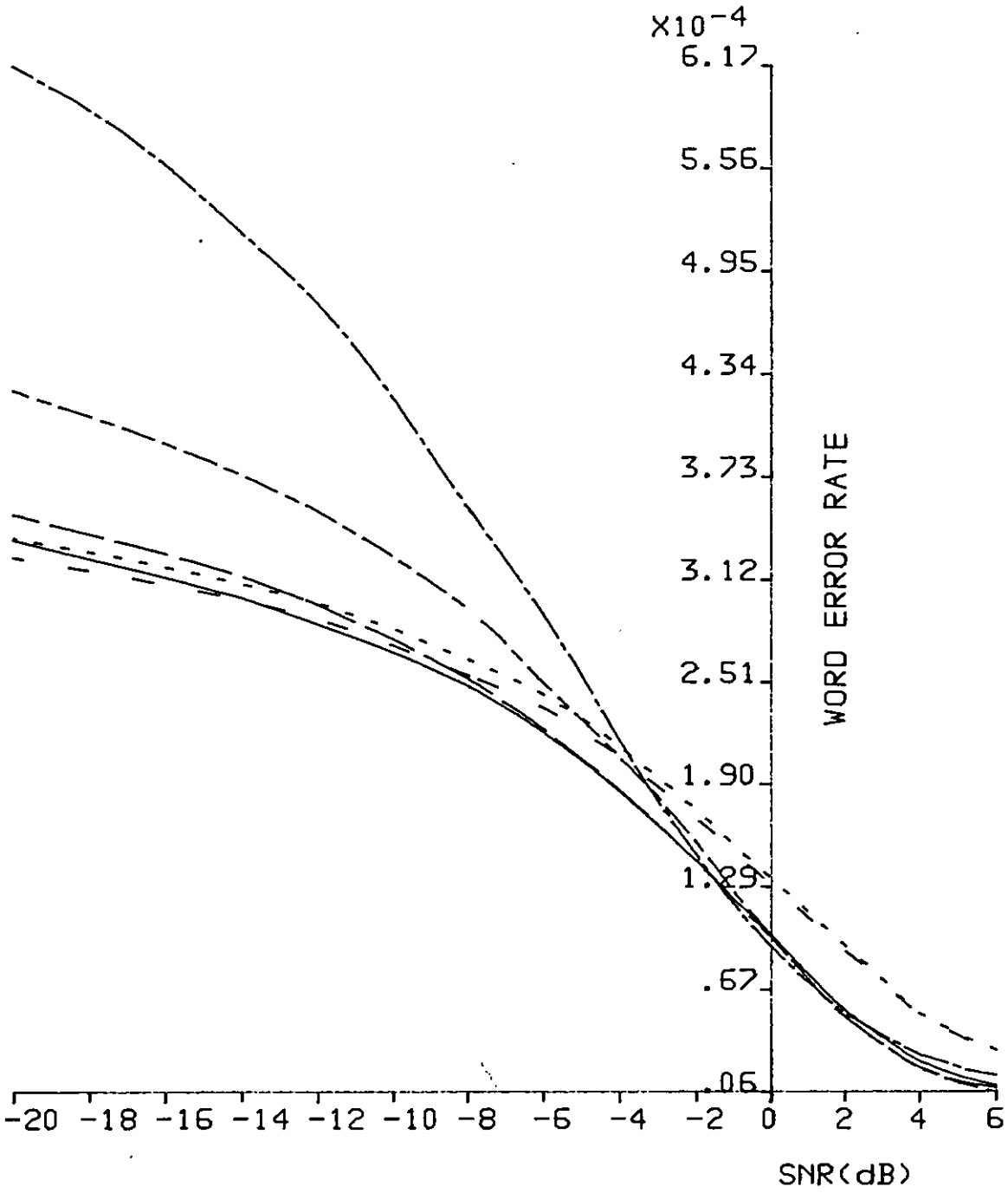


FIG.(5.37) SPTD PERFORMANCE USING BURST (27,20) CODE

5.12.4 The Effect of Code Rate On The Performance Of Parallel Threshold Decoders

The study of the effect of code rate on the PTD performance is a difficult problem to solve. On one hand different codes with different code rates have to be used, which involves the simulation of all chosen codes. On the other hand all the codes used should have the same correction power, i.e. the error-correction ability of all codes (t for random codes, and ℓ for burst codes) should be the same for the same number of perity-check digits.

An easier solution to this problem is to use a code that can be shortened and lengthened. Shortening a code represents no problem since any code can be shortened. But lenghtening codes cannot be done unless the code itself is a shortened code. It is worth mentioning that if an optimum burst code is shortened or lengthened, it does not necessarily mean the code is still optimum, although an optimal code will remain optimal. Clearly the shortening of a code will decrease its code rate, and lengthening it will increase its code rate, and if the same code is shortened and lengthened, then its correction power is the same.

The shortening and lengthening idea is used to study the code rate effects on the PTD performance. The shortened (34,22) burst code is used for the study. A parallel threshold decoder with practical spacing is used. The background SNR=9 dB, and the burst SNR is varied from -20 dB to 6 dB, while the code rate is changed from

0.52 to 0.666. It would be more comprehensive to increase the code rate to a higher value but the computer word length was a deciding factor to stop at this value. For each value the PTD and the error-trapping decoder are tested. The PTD improvement over the error-trapping decoder in terms of word error rate is calculated for all burst SNR values. It was found that the results curve have a similar shape, hence only three curves for burst SNR 2,-8, and-17 dB are plotted in fig.(5.38).

The curves of fig.(5.38) show that the code rate has a big effect on the PTD improvement. At low code rates the improvement is low, then as the code rate is increased the improvement becomes greater until it reaches a maximum value, then it starts to drop as the code rate is further increased. Two other conclusions can be drawn from fig.(5.38); firstly, the improvement is higher at lower burst SNR, which means that the PTD performs better as the burst SNR drops; secondly, the peak of the curve is flater at low burst SNR than at higher burst SNR, which means that to get the maximum improvement the code rate is less critical at low burst SNR values than at higher burst SNR values.

The above results were somehow expected because of the relation between the code rate and the number of trappable error-patterns, as previously discussed in sections 5.9.1, 5.9.2. Since the number of the correctable but untrappable errors increase with the increase in the code rate, and since the PTD achieves some improvement by correcting these untrappable error-patterns, then the

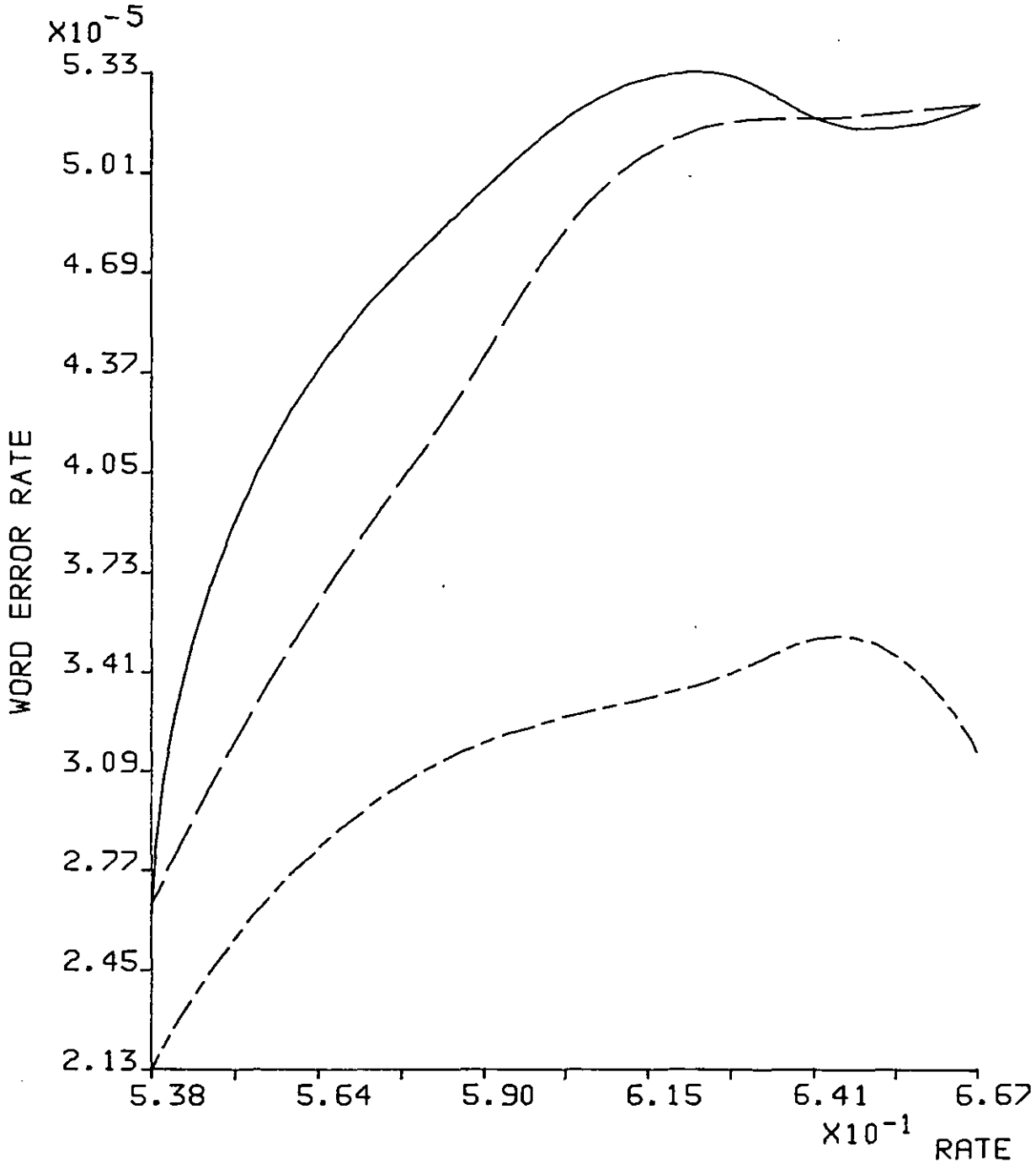


FIG.(5.38) CODE RATE EFFECT ON PTD IMPROVEMENT

improvement increases with the increase in the code rate, which is responsible for the left hand side of the curve from the maximum point. As the code rate is increased, the number of untrappable but correctable errors become so large that the code itself becomes powerless to correct most error-patterns, hence the drop at the right hand side of the curve.

5.13- Advantages Of The Parallel Threshold Decoding

The use of any decoding method is dependent on two principle factors. Firstly, the total decoder cost, i.e. hardware, implementation, software, testing, etc., secondly, the decoder's ability to function in real time, i.e. its speed, for a given performance achieved.

The parallel threshold decoder is very attractive from these points of view, its total cost is not much more than that of the error-trapping decoder, and much less than the soft-decision decoder. Once a subdecoder is built and tested, the remaining subdecoders are just duplicates, in a practical system hardly more than five threshold need to be used thus the hardware cost of the subdecoders is very cheap, because they consist of shift registers and gates only. While the threshold circuits are ordinary threshold gates unlike the expensive analogue-to-digital converter used in soft-decision decoders. Again because the parallel threshold decoder is easily and economically implemented in hardware, no software expense need be involved, thus on the whole the parallel threshold decoder gives a very good cost-

performance trade of. Also, since the whole decoder is economical to implement directly in hardware its speed as a real time decoder is limited only by the maximum speed of the hardware used.

Consequently the two main advantages of the parallel threshold decoder are its low cost, and it speed, which makes it very competitive with other decoders. Although the parallel threshold decoder provides a big advantage when implemented as hardware decoder, yet there is nothing to stop its implementation by the use of microprocessors, although by doing this at the present state of thechnology most of its advantages will be lost.

5.14-Conclusions

The parallel threshold decoding is a decoding technique that attempts to minimize the symbol probability of error during the whole transmission, or to minimize the block symbol probability of error, or both of them.

The parallel threshold decoder should in general perform equal to or better than the error-trapping decoder, the amount of improvement is dependent on the SNR ratio or ratios, the lower the SNR values the more the improvement. The improvement is also dependent on the code itself, the largest improvement is achieved at moderate code rates 0.629 and 0.647, while for a powerful code, at moderate background SNR value, the improvement is small because the parallel threshold decoder is not able to add too much to the code correction power. At the other extreme, the parallel threshold decoder is unable to enhance the performance of a weak code.

is dependent on the code rate and the SNR values, thus the statistical parallel threshold decoder, will follow a similar improvement curve achieving higher improvement, where the parallel threshold decoder improvement is high, but at high code rates not achieving the same improvement as the parallel threshold decoders.

Although in theory the number of thresholds can be very large, in practice there is hardly any need to include more than five thresholds or at most seven thresholds, since most of the improvement is achieved at the first few thresholds.

The parallel threshold decoders can be used in a two-way channel system, provided a small number of thresholds are used i.e. three or five thresholds at most. A retransmission is requested once a decoding failure is detected, i.e. no trappable error-pattern is found in all subdecoders. The number of thresholds is very important for such a system because as the number of thresholds is increased the improvement is increased, so is the number of erroneous decoded words, while the number of the decoding failures in all subdecoders, hence the retransmission is decreased, and visa versa. Consequently it is safer to use a small number of thresholds, and more retransmissions, so that the overall error rate is reduced.

CHAPTER 6

THE SEARCH PARALLEL THRESHOLD DECODER

6.1- The Decoder Limitation

Decoding techniques for block codes can be divided into two general categories, algebraic and nonalgebraic. The algebraic techniques basically involve the simultaneous solution of set of equations for the location and values of the errors. The nonalgebraic techniques, while accomplishing the same goal, are based upon simple structural aspects of the codes which permit the determination of the error-patterns in a more direct fashion.

In general the algebraic decoders are more complicated in terms of the hardware than the nonalgebraic decoders, but they are usable with many error-correcting code. While the nonalgebraic decoders have some built in disadvantages. Since we are interested in the error-trapping decoders which are nonalgebraic decoders, we will consider their disadvantages. As a practical matter, although their hardware implementation is reasonably simple, the complexity of these decoders increases rapidly as the number of error to be corrected grows. In addition to this, for any error-pattern to be correctable by these decoders, it has to be trappable, and since any correctable error-pattern is trappable only if the error-pattern is confined to n-k consecutive digits, the decoder may not correct all correctable error-patterns. Hence they

are generally not useful for correcting more than 3 random errors or for correcting more than a single burst of errors. Clearly, as the number of random errors, or the length of the single burst grows, the algebraic decoders becomes more efficient than the error-trapping decoders, because the number of the correctable but untrappable error-patterns becomes larger. Consequently the algebraic decoders have to be used.

In general any error-correcting code can correct all errors-patterns of L or less. Where L=t the maximum number of random errors that a random-error-correcting code guarantee to correct regardless of their position in the codeword, or L=ℓ the maximum burst length of which the code is capable of correcting regardless of its position in the codeword. On the other hand the code cannot correct all the error-patterns of L+1 or more, but it will correct some of them.

In order to guarantee the correction of all error-patterns of L or less, decoders in both categories are limited to the correction of t random errors or burst length ℓ, although that is below the code error-correction ability.

6.2- Expanding the Limitations

When a one-way channel is used as the transmission channel, some application may be required to correct as much data as possible at the expense of increased decoding errors. No matter which decoder is used all error-patterns of L or less will be decoded correctly,

while all error-patterns of L+1 or more will be signaled as decoding failure. If an error-trapping decoder is used, then the untrappable error-patterns if any, will be signaled as additional decoding failures. Being one-way channel the receiver has no choice other than discarding the erroneous received word, or accepting it knowing it is erroneous. In the later case the word error rate will be lower if the full correction capability of the code is used. Because all error-patterns of L or less will be corrected as before, while part of the previous decoding failures will be corrected. But in such a case the receiver will have no means of telling which received word is correct and which is a decoding failure i.e. it may make a decoding error.

6.3- The Search Parallel Threshold Decoding (SPT)

The search parallel threshold decoder is a decoder of the later type described above, it tries to correct as many as possible of the erroneous received words. It is basically a modified error-trapping decoder of which the decoder has no means of detecting failures and uses the principle of the parallel thresholds to decode the received word, but in a completely different way to the parallel threshold decoding.

One way for decoding using the SPT decoder is for the decoder to use certain weighing rules to decide which threshold is optimum or near optimum, and accept its output as the received word, where the decoding is carried out for that word, and the corrected output is accepted

as the transmitted information digits. The weighing is done for all the trapped error-patterns in the syndrome registers for all thresholds.

An alternative method is for the SPT decoder to use the same concept of the optimum and near optimum thresholds as above, but the weighing is done in a way that the SPT decoder uses one digit at a time of the trapped error-pattern in every syndrome register to correct the bit errors in the received word, i.e. the weighing is done on a digit by digit basis for the trapped error-patterns.

6.3.1 The Trapped Error-Patterns

Consider a decoder which has j threshold detectors at its input, the output of each of these j detectors is fed to a syndrome calculator. When a received word is fed into this decoder, each detector will output the received word according to its threshold value. Once all the received words are fed into the receiver, j syndromes are formed in the j syndrome registers. Then each of these syndromes is shifted until a trappable error-pattern is found, and if at the end of the n shifts no trappable error-pattern is found, the syndrome register is reset to zero. Let the j vectors of the error-patterns be

$$\left.\begin{aligned} E_i &= e_{0,1}e_{1,1}\cdots\cdots e_{n-k-1,1} \\ E_2 &= e_{0,2}e_{1,2}\cdots\cdots e_{n-k-1,2} \\ &\vdots \\ E_j &= e_{0,j}e_{1,j}\cdots\cdots e_{n-k-1,j} \end{aligned}\right\} \qquad (6.1)$$

Assuming that the i^{th} threshold is an optimum or near optimum threshold, then the i^{th} trapped error-pattern is the most likely error-pattern to have been added to the transmitted codeword in the channel.

The distance d_1 between the '1', voltage +V, and the optimum threshold is given in Eq.(5.47)

$$d_1 = V+\frac{\sigma^2}{2V} \log \frac{q_1}{q_0} \qquad (5.47)$$

where σ^2 is the noise power, q_1 and q_0 is the number '1' and '0' in the transmitted codeword. Fig.(5.9) shows the variation of the optimum threshold distance with the variation of SNR values for all possible codewords except the all zeros and all ones codewords for the BCH(15,7) random-error-correcting code.

Although Eq.(5.47) gives the distance for all zeros and all ones codewords as $+\infty$ and $-\infty$, which correspond to threshold values of $-\infty$ and $+\infty$ respectively. Fig.(5.9) shows that for all codewords other than the all zeros and all ones codewords, thresholds are concentrated around the 0 volt value, and for reasonable SNR values the thresholds are within a small distance from the 0 volt, e.g. in fig.(5.9) the distance is $\pm\frac{V}{2}$. However, no matter what the threshold values are, usually they will be limited by the hardware to $\pm V$. One should notice that for any code there are 2^k different codewords, hence the ratio $\frac{q_1}{q_0}$ in general will have more values as k grows, consequently the number of the optimum thresholds will increase as k

increases. Such an increase will lead to a higher concentration of thresholds around the 0 volt when a figure similar to fig.(5.9) is plotted for a code with higher k value. So even if using the optimum threshold for the (15,7) code may be reasonable, for longer codes one has to use one of the threshold distributions described in section 5.7.

Assuming that the number of thresholds used j in the above described decoder are large, and that a codeword other than the all ones and all zeros codeword is transmitted, furthermore, let the optimum or near optimum threshold for detecting the received word be the i^{th} threshold, then the most likely error-pattern to be added to the transmittted codeword in the channel will be E_i, where E_i is the i^{th} trapped error-pattern in Eq.(6.1). Since j is a large number, then the $i+1^{th}$ and $i-1^{th}$ threshold will be near optimum thresholds, hence the corresponding error-patterns E_{i+1} and E_{i-1}, are likely to be the error-pattern added to the transmitted codeword. Again the $i+2^{th}$ and $i-2^{th}$ thresholds may be close enough to be near optimum thresholds, thus the corresponding error-patterns E_{i+2} and E_{i-2}, may be the error-pattern added to the transmitted codeword, but they are less likely than E_{i+1} and E_{i-1}, which in turn are less likely than E_i. The same thing can be said about E_{i+3} and E_{i-3}, and so on.

The reason for excluding the all ones and all zeros codewords from the above argument is that these two codewords represent the extreme cases, and since

their optimum thresholds are at $-\infty$ and $+\infty$, then their optimum thresholds are not included in the thresholds used in the decoder, because usually the thresholds span is -V to +V. Thus the nearest thresholds to -V and +V are the near optimum thresholds for these two words. Going back to the j thresholds decoder, these two thresholds will be the 1^{st} and j^{th} thresholds. Consequently, for the 1^{st} threshold no 0^{th} threshold exists, and for the j^{th} threshold no $j+1^{th}$ threshold exists. Needless to say what applies to the thresholds applies to the error-patterns, hence no E_0 or E_{j+1} exists. Clearly for such cases no symmetry exists around the optimum or near optimum threshold, and the error-patterns to be considered are $E_1, E_2, E_3, \ldots$ or $E_j, E_{j-1}, E_{j-2}, \ldots$. Apart from that all the above discussions apply to all thresholds. In general the decoder can consider as many error-patterns as required, provided they do exist.

6.3.2 The Error-Pattern - A Different Look

Consider the i^{th} error-pattern E_i in Eq.(6.1), which could be any trapped error-pattern resulting from the detection by the i^{th} threshold,

$$E_i = e_{0,i} e_{1,i} \cdots\cdots e_{n-k-1,i} \qquad (6.1\text{-a})$$

The error-pattern consists of n-k digits, each digit is either '0' or '1' and can be regarded as a flag, where the flag is indicating an error if it is set, i.e. it is '1' or indicating no error if it is reset, i.e. it is '0'.

In addition to that the error-pattern corresponds to a specific n-k digits in the received word, each digit of the error-pattern components corresponding to a unique digit in the received word. Consequently the decoder knows exactly which bits are erroneous according to E_i.

Assume that the i^{th} error-pattern is the optimum or the nearest to the optimum threshold, then the most likely error-pattern to have been added to the transmitted codewrod is E_i. But since the $i+1^{th}$ and $i-1^{th}$ thresholds are near optimum, then E_{i+1} and E_{i-1} will represent the same error-pattern as E_i, and all error-patterns will correspond to the same n-k digits of the received word. Thus

$$E_i = E_{i-1} = E_{i+1} \tag{6.2}$$

from Eq.(6.2) and Eq(6.1-a)

$$e_{m,i} = e_{m,i-1} = e_{m,i+1} \text{ where } m=0,1,2,..,n-k-1 \tag{6.3}$$

If any other threshold is near enough to be considered a near optimum threshold, then its error-pattern is the same as E_i and it will correspond to the same n-k digits in the received word. In general Eqs.(6.2) and (6.3) can be written

$$E_i = E_r \tag{6.4}$$

$$e_{m,i} = e_{m,r} \text{ where } m=0,1,2,\ldots,n-k-1 \tag{6.5}$$

where r is the number fo the r^{th} threshold that can be

considered as near optimum threshold to the i^{th} threshold.

Clearly as the number of thresholds used j is increased, the distance between thresholds becomes smaller. Consequently, the number of near optimum thresholds for any optimum threshold grows larger. In other words as j is increased the range of r becomed larger, and according to Eq.(6.4) more error-patterns become the same as the optimum threshold error-pattern E_i. Up to now no distribution of any kind is assumed for the thresholds used. If a special distribution is used where the thresholds are more concentrated in the area where the optimum thresholds are, then the range of r in Eqs(6.4) and (6.5) will be even larger for the same number of thresholds used j. In fact one can use one of the distribution described in chapter five, section 5.7.

6.3.3 The Decoding Strategy

The basic decoding idea is to detect the added error- attern to the transmitted codeword from the j error-patterns using Eq.(6.4). Or to detect the bits of the error-pattern added by the use of Eq.(6.5). Each type of detection will give a different implementation of the decoder. The first type of detection, involves the scanning of all j error-patterns of Eq.(6.1), and looking for the largest number of equal error-patterns that correspond to the same n-k digits of the received word and accepting this error-pattern as the error-pattern added to the transmitted codeword, where the correction can be achieved as in the conventional

error-trapping decoder. The second type of detection, involves the scanning of one bit at a time of all j error-patterns of Eq.(6.1) looking for set error flags that correspond to the first digit in the received word, if the number of detected flags is greater than a certain threshold (the error-threshold) value, an error in that bit is assumed, otherwise it is assumed error-free, then the whole scan is repeated for the second digit, and so on. The detection correction process is finished when all the n digits of the received word are finished, i.e. n scans are finished. At the end of this detection correction process the corrected word may contain an erroneous bit, thus it is corrected in another error-trapping decoder to give the final decoder output.

6.4-The SPT Decoders

6.4.1 The Error-Pattern SPT Decoder

The error-pattern SPT decoder is the decoder that uses Eq.(6.4) for decoding. A block diagram of the decoder is shown in fig.(6.1), and the decoder functions as follows:

The received word is fed to the decoder input bit by bit, where each bit is compared with all j thresholds. The output of each threshold is fed to its 'Syndrome Calculator' and to the received word storage register. By the time all the received word is fed completely into the decoder, the syndrome calculator contains the syndrome, while the k stage 'Information Digits Storage Register' contains the received information digits detected by

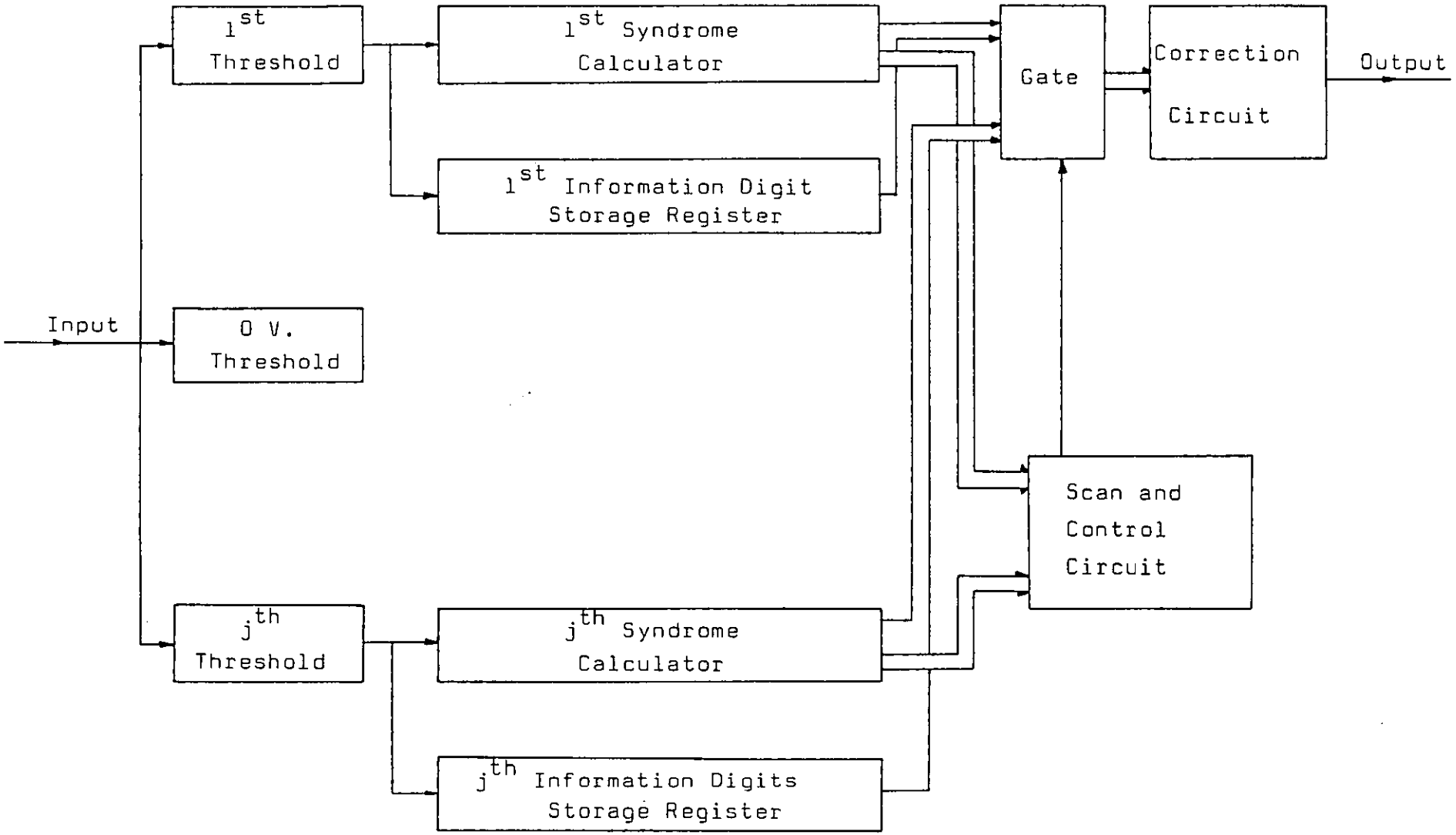


Fig.6.1 A block diagram of the error-pattern SPT decoder

its threshold. Then the syndrome calculator is shifted until a trappable error-pattern is detected. If at the end of n shifts no trappable error-pattern is detected, the syndrome register is reset to zero. At the end of n shifts all j syndrome registers will contain either trappable error-patterns , or zeros. The 'Scan and Control Circuit' starts to scan all error-patterns from the first to the j^{th} in turn, looking for the largest number of consecutive equal error-patterns. The term equal error-patterns means that for any two equal error-patterns, say E_i, E_s

$$E_i \oplus E_s = 0 \qquad (6.6)$$

where $\oplus$ represent a modulo-two addition, and that the two error-patterns correspond to the same n-k digits in the received word. After scanning the j^{th} error-pattern the 'Scan and Control Circuit' knows the place of the largest number of equal error-patterns area, say from E_i to E_s. At this stage the detection phase is finished, and the decoder accepts any of these equal error-patterns (E_i to E_s) as the actual error-pattern added to the transmitted codeword. For correction, the 'Scan and Control Circuit' selects one of the accepted error-patterns, and the corresponding received information digits, opens the 'Gate' and passes them through to the 'Correction Circuit' where the correction is done and the corrected information digits are outputed to the next stage.

A further simplification of the decoder can be achieved by observing that the accepted error-pattern is present in large number in adjacent thresholds trapped error-pattern, when the two error-patterns are equal the two detected received words are equal. Thus the decoder is simplified by omitting every second and third 'Information Storage Register' so that when a group of error-patterns are chosen, an error-pattern that has an 'Information Digits Storage Register' is passed through the 'Gate' to the correction circuit.

6.5-The Digit SPT Decoder

The digit SPT decoder is the decoder that uses Eq.(6.5) for decoding. A block diagram of the decoder is shown in fig.(6.2). The decoder function is as follows:

The received word is fed to the decoder, where it is detected at each of the j thresholds and the output of each threshold is fed to its 'Syndrome Calculator'. The output of the zero threshold is fed also to the 'Received Word Storage Register', which consists of n stages, so that the whole received word is stored including the parity-check digits. By the time the whole received word is fed into the decoder the j syndromes arecalculated and each syndrome calculator is shifted until a trappable error-pattern is detected, if no trappable error-pattern is detected after n shifts the syndrome register is set to zero. Once the time of n shifts has elapsed, i.e. no syndrome calculator is shifted any longer, the 'Scan and Control Circuit' detects any erroneous digits according

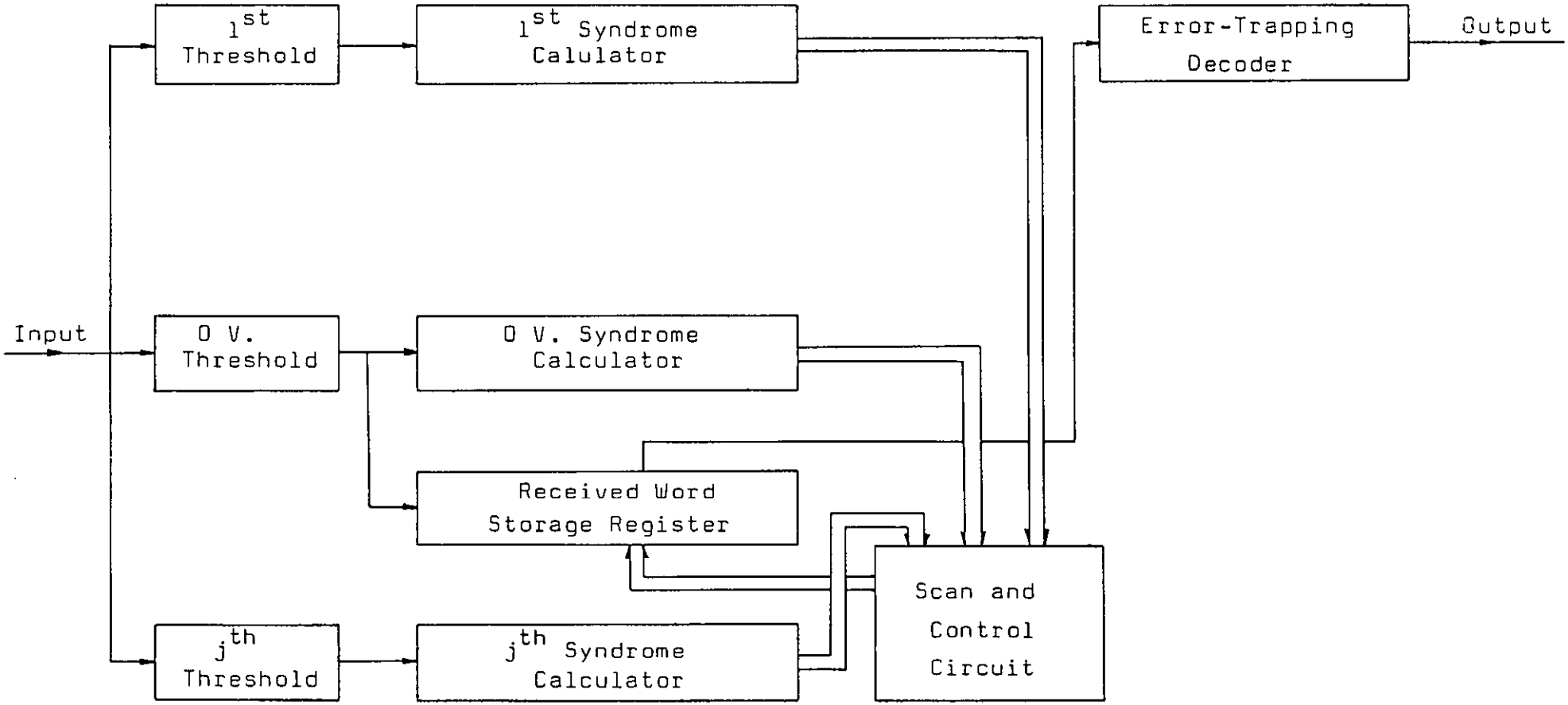


Fig.6.2 A block diagram of the digit SPT decoder

to the following rules.

When a received word is detected by an optimum or near optimum threshold and the adjacent near optimum thresholds, the j trapped error-patterns will correspond to different sections of n-k digits of the received word. If one considers any one digit of the received word, then there is at most j flags pointing to that digit. If the considered digit is erroneous, then according to Eq.(6.5) there are at least r set flags. The error detection process is simply counting all set flags which correspond to a particular digit, if the number of flags is equal or larger than r (the error-threshold value), then the digit is assumed erroneous, otherwise the digit is assumed error-free. Since the error-threshold number is varied according to the channel noise, then it is taken as a preset number, determined by simulation.

Going back to the 'Scan and Control Circuit', once the shifts are ended, the 'Scan and Control Circuit' starts scanning all flags corresponding to the first digit of the received word, counting the set flags. If no flag is found in an error-pattern, i.e. no digit in that error-pattern corresponds to the first digit of the received word. then a reset flag is assumed. At the end of the first scan cycle if the number of the set flags is larger than the error-threshold, the first digit is changed. This process is repeated n times for all n digits. The last phase of correction is to feed the corrected received word which is stored in the

'Received Word Storage Register' to be corrected in an error-trapping decoder. The necessity of this last phase is that there is a small probability that error-patterns other than the error-patterns resulting from the optimum or near optimum thresholds, will set enough flags so that a digit in the received word, not erroneous according to the accepted error-pattern is inverted. And since the code itself can correct such errors, it is used for correction in the last phase.

6.6-The Statistical Decoders

The statistical decoders are the decoders that can correct all error-patterns of L or less and some error-patterns of L+1 or more. An example of these is the optimum decoder (discussed in chapter two) for correcting burst of errors. In general the statistical decoders depend on special statistical information of the transmission system to perform their decoding.

The optimum decoder can be considered as a statistical error-trapping decoder, because the error-pattern is trapped in the syndrome register, yet that requires the calculation of all the possible error-patterns for the received word. Although the optimum decoder can be used effectively with a large number of single-burst error-correcting codes, unfortunatly its idea can not be extended to the random error-correcting codes for two basic reasons; Firstly the optimum decoder is used with channels which have special statistical properties, unlike these channels random channels are memoryless

channels, hence no special statistical properties can be assumed. Secondly, in the optimum decoder the detected error-pattern is extneded to include burst lengths of $L \geqslant \ell+1$, this cannot be done with decoders for random error-correcting codes, because of the following.

Consider a linear (n,k) code, which has a minimum distance d_{min}, where

$$2t + 2 \geqslant d_{min} \geqslant 2t + 1 \qquad (6.7)$$

Let C be a transmitted codeword, R the received word, and U any other codeword. The Hamming distances among C, U, and R satisfy the following inequality

$$d(C,R) + d(U,R) \geqslant d(C,U) \qquad (6.8)$$

if an error-pattern of L error occurs, where

$$L \leqslant t \qquad (6.9)$$

Then the Hamming distance between the transmitted codeword C and the received word R is

$$d(C,R) = L \qquad (6.10)$$

Since

$$d(C,U) \geqslant d_{min} \geqslant 2t + 1 \qquad (6.11)$$

Then from Eq.(6.8)

$$d(U,R) \geqslant d(C,U) - d(C,R) \qquad (6.12)$$

Substituting Eqs.(6.10) and (6.11) in Eq.(6.12) gives

$$d(U,R) \geq 2t + 1 - L \tag{6.13}$$

from Eq.(6.9)

$$2t + 1 \geq L + t + 1 \tag{6.14}$$

$$2t + 1 - L \geq t + 1 \tag{6.15}$$

substituting in Eq.(6.13) gives

$$d(U,R) \geq t + 1 \tag{6.16}$$

from Eqs.(6.9) and (6.16)

$$d(U,R) > L \tag{6.17}$$

The inequality of Eq.(6.17) says that, if an error-pattern of t or fewer errors occurs, the received word R is closer to the actual transmitted code vector C than to any other codeword U, thus the decoding will be correct. On the other hand, the decoder cannot correct all patterns of L>t+1 errors, for there is at least one error-pattern which is closer to a codeword other than the transmitted codeword.

The result of Eq.(6.16) is of special interest, because it states that for any error-pattern of L errors that satisfies Eq.(6.9), there exists at least one codeword where the received word is at a Hamming distance

of t+1 from this codeword. Eq.(6.10) shows clearly that the extension of the correction power of an error-trapping decoder using a random-error-correcting code from L=t to L=t+1 will result in more decoding errors, because for every received word at a distance t or less from a codeword, there is another codeword at distance t+1 from the received word. And since the error-trapping decoder will accept the first error-pattern at a distance of L or less as the error-pattern added to the transmitted codeword in the channel, then the decoder will accept the first of either codewords as the transmitted codeword. If the decoder identifies the codeword at distance t or less as the transmitted codeword, the decoding is correct, otherwise the decoder will make an incorrect decoding. Although the decoder will decode correctly some error-patterns of L=t+1, but the degradation will be much more than the improvement.

Needless to say that further extension of the correction power, say to L>t+1, will result in more degradation, because the number of the codewords at distance L will become larger as L grows larger.

6.7-Error-Trapping Decoder With Extended Correction Power

An extended correction power error-trapping decoder must satisfy the following rules;

1-Its performance must be at least equal to the performance of an error-trapping decoder, i.e. the extended error-trapping decoder should be able to correct all trappable error-patterns of L or less,

where $L \leqslant t$.

2-It should be able to correct some error-patterns of L or more, where $L \geqslant t+1$.

An error-trapping decoder can be modified to be an extended error-trapping decoder as the following. After calculating the syndrome, the error-trapping decoder will calculate all the possible error-patterns added to the transmitted codeword. Once a trappable error-pattern is detected, it is assumed that the trapped error-pattern is the one added to the transmitted codeword in the channel and the correction is done accordingly. The modified error-trapping decoder does the same, except that while calculating the possible error-patterns added to the transmitted codeword, and looking for a trappable error-pattern, the decoder looks also for error-pattern of $L=t+1$. Once such an error-pattern is detected the error-pattern and its location are stored, and the calculation of the remaining error-patterns is continued. If a trappable error-pattern (of $L \leqslant t$) is detected, the correction is carried out as with the error-trapping decoder. On the other hand, if no trappable error-pattern is detected at the end of the detection cycle, the stored error-pattern (of $L=t+1$) is used for correction using the error-pattern location information.

Another method of achieving the correction is by storing the error-pattern only. At the end of the detection cycle, if no trappable-error is found, the shifting process is continued until the error-pattern

in the syndrome register matches the stored error-pattern, then the correction can be carried out in the same way as with the trappable error-pattern.

The error-correction power of the modified error-trapping decoder can be extended so that the code can correct some error-patterns of L where

$$t + 1 \leqslant L \leqslant n-k \qquad (6.18)$$

Assuming that the decoder is required to correct some error-pattern of L_i where

$$t + 1 \leqslant L_i \leqslant t + i \qquad (6.19)$$

in addition to correcting all error-patterns of $L \leqslant t$. In such cases the decoder will function as described above, but it will look for and stored error-patterns of $t+1, t+2, \ldots . t+i$. If no trappable error-pattern is detected, the error-pattern which has the smallest number of errors is used for correction as described above.

6.7.1 The Extended Error-Trapping Decoder Performance

Consider the case where the error-pattern is trappable, i.e. $L \leqslant t$. The extended error-trapping decoder will consider the trapped error-pattern as the error-pattern added to the transmitted codeword in the channel, and will correct accordingly same as the error-trapping

decoder, no matter how many error-patterns of $L \geq t+1$ are stored. Thus for $L \leq t$ the extended error-trapping decoder will perform exactly as the error-trapping decoder. If an untrappable error-pattern of $L \leq t$ is added to the transmitted codeword, the error-trapping decoder will signal that an untrappable error has occured, and since that is decoding failure generally, the output is an erroneous block of information digits. While the extended error-trapping decoder will consider the stored error-pattern, and correct accordingly, thus resulting in an erroneous block of information digits. The only difference between the two decoders so far is that the error-trapping decoder gives an indication when a decoding failure has occured.

Consider now the other case, where the error-pattern is of $L=t+1$, the Hamming distance between a codeword U and the received word R is given in Eq.(6.13)

$$d(U,R) \geq 2t + 1 - L \tag{6.13}$$

substituting $L=t+1$ in Eq.(6.13) gives

$$d(U,R) \geq t \tag{6.20}$$

the inequality of Eq.(6.20) shows clearly that if an error-pattern of $L=t+1$ has occured the received word is closer to a codeword other than the transmitted codeword, thus the decoding is erroneous. But under certain conditions these codewords are not detected, i.e. the t errors are

untrappable, thus the decoder will detect a pattern of t+1 errors, if the detected error-pattern is the actual one, then the decoding is correct.

To study the performance of the error-trapping and the extended error-trapping decoders, when an error-pattern of L=t+1 is added to the transmitted codeword, two cases must be considered; Firstly, when different error-pattern is trappable, in such a case the error-trapping decoder, and the extended error-trapping decoder will accept the error-pattern of t errors as the actual error, and the decoding will be carried out accordingly, resulting in a decoding error in both decoders. Secondly, when the error-pattern of t errors is untrappable. In this case an error-pattern of L=t+1 is present, and the closest codeword or codewords are at a Hamming distance of t+1 from the received word. In general the error-trapping decoder will detect an untrappable error-pattern, signal a decoding failure, and the decoding result is an erroneous block of information digits. On the other hand the extended error-trapping decoder will store an error-pattern of L=t+1. According to Eq.(6.10) the received word is at Hamming distance of t+1 from the transmitted codeword, while Eq.(6.20) shows that there could be another one or more codewords at the same Hamming distance from the received word. Assuming that there are B_{t+1} codewords at Hamming distance of t+1 from the received word. Since the decoder has no means of evaluating which one is the transmitted codeword, the extended decoder will choose

one of these codewords as the transmitted codeword. In other words the extended error-trapping decoder will find B_{t+1} error-patterns of L=t+1, and has to choose one of them as the actual error-pattern. Since any of the B_{t+1} error-patterns can be the actual error-pattern and the decoder has no additional information about which one to choose, then the decoder can be forced to make a certain choice, i.e. first or last error-pattern, so that the decoder hardware is less complicated.

The probability of choosing the actual error-pattern added to the transmitted codeword in the channel, hence the probability of correct decoding P_{cd} is

$$P_{cd} = \frac{1}{B_{t+1}} \tag{6.21}$$

Since the decoder will choose an error-pattern of L=t+1 only if no error-pattern of L$\leq$t+1 is detected, then Eq.(6.21) can be rewritten

$$P_{cd} = \frac{1}{B_{t+1}} \quad \text{iff} \sum_{i=0}^{t} B_i = 0 \tag{6.22}$$

where B_i is the number of error-pattern with L=i. In general

$$P_{cd} = \frac{1}{B_L} \quad \text{iff} \sum_{i=0}^{L-1} B_i = 0 \tag{6.23}$$

where L is defined in Eq.(6.18).

In general, extending the error-correction power

of an error-trapping decoder to more than t+1 will not give an additional improvement. To see the reason, one should go back to Eq.(6.13), substituting the value of t+2 in L shows that error-patterns of $\geqslant$t-1 are present, and substituting t+3 in L, shows that error-patterns of $\geqslant$t-2 are present, and so on. Clearly, the chances of the decoder choosing the actual error-pattern between the present error-patterns of $t-1 \leqslant L \leqslant t+2$ for the first case, and $t-2 \leqslant L \leqslant t+3$ in the second case are slim indeed. Another way of looking into this is by considering the Hamming distance between the received word and the accepted codeword as the transmitted one. As this distance is increased by increasing L as it can be seen from Eq.(6.10), more and more codewords become within the range of L. And since the decoder will choose one of these codewords, then the probability of correct decoding becomes smaller and smaller.

6.8- Simulation Results and Discussions

Unfortunately, because of the time limitations, the results included here are the first stage of the test results for the digit SPT decoder only. Although these results represent the beginning of the test.work for this chapter, and may not be the same as the final result after including all the possible improvements, yet they are of great importance because they show clearly that the basic theoretical ideas are correct and can be implemented. It was felt that the error-pattern SPT

decoding will provide higher improvement than the digit SPT decoding, thus it was decided to work on the digit SPT decoding in the remaining time, because the success of the digit SPT decoding idea will give some assurance that the error-pattern SPT decoding is also a success.

6.8.1 Choosing The Error-Threshold

To study the effect of the error-threshold on the digit SPT decoder, two types of codes were used the (31,21) random-error-correcting code, and the (34,22) single-burst-error-correcting code. The first code used with a random error-generating channel, the decoder is the error-trapping decoder type described in section 6.5.1. The thresholds are taken 17 practically spaced thresholds. Several runs for different SNR values are tried, and they all prove to have the same general shape, although the error-threshold value that minimizes the word error rate is different for different SNR values. The curve of fig.(6.3) is a typical curve, where the word error rate is plotted against the value of the error-threshold. The error-trapping decoder results are plotted also for comparsion. The SNR value used in fig.(6.3) is 3 dB.

The second code used is the (34,22) single-burst-error-correcting code. The decoder used for the test is similar to the decoder described in section 6.5.1, but the basic unit is an optimum decoder instead of error-trapping decoder. Again the thresholds used are 17 thresholds practically spaced. The background SNR is set to 9 dB, while the burst SNR is varied for different

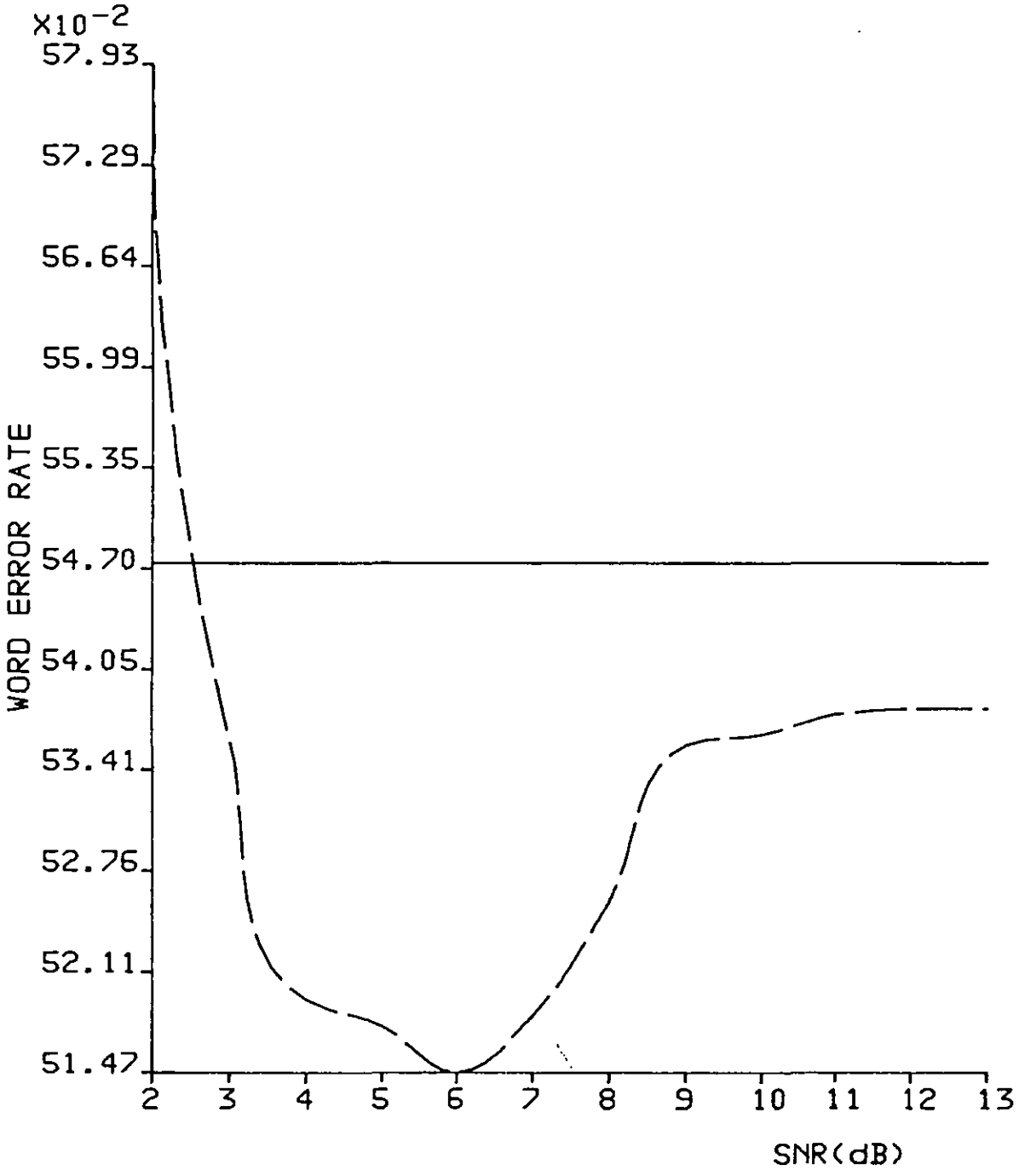


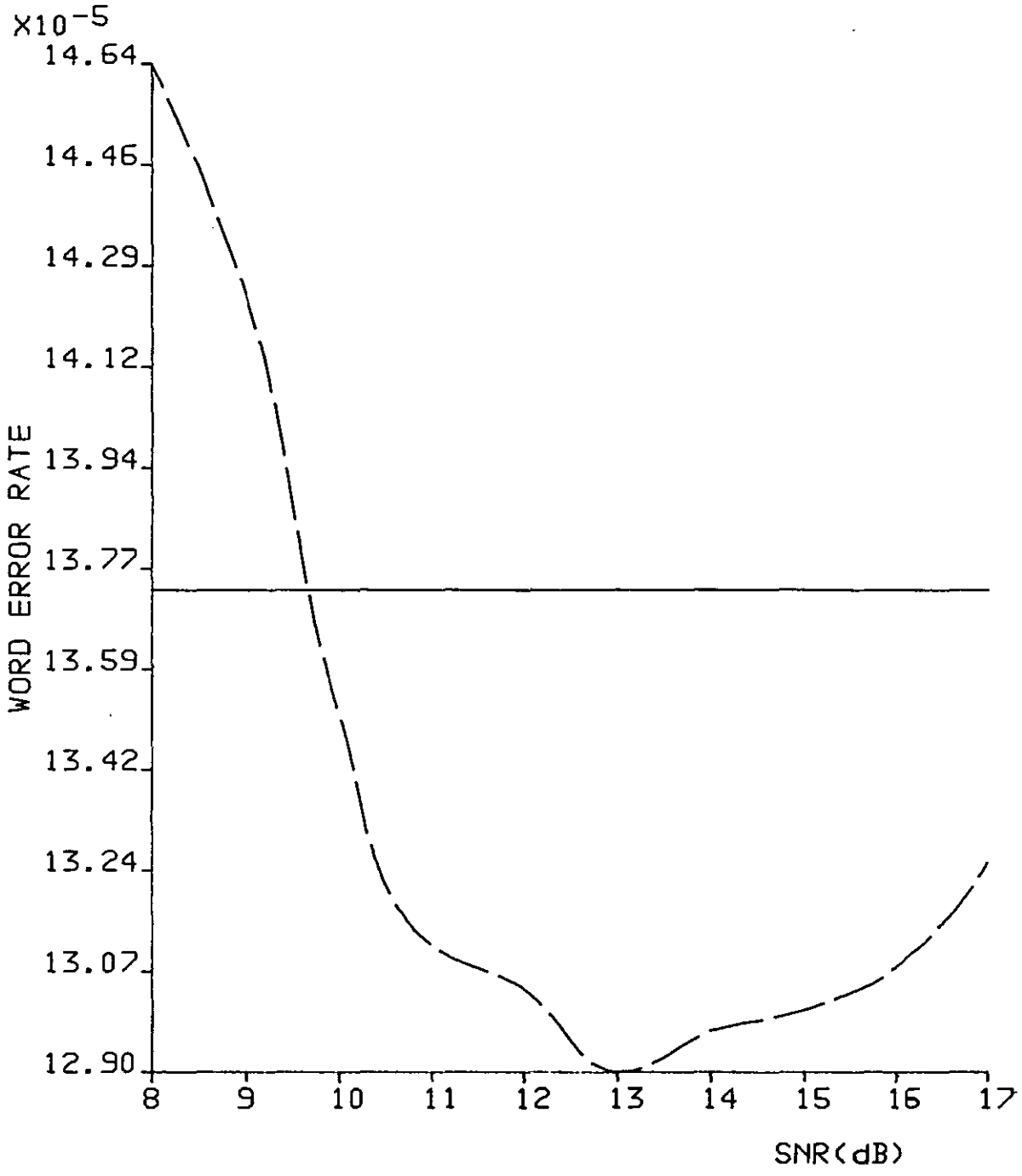
FIG.(6.3) THE EFFECT OF ERROR-THRESHOLD VALUE

runs. All runs prove to have the same general curve, furthermore they all have a very close values for the error-threshold that minimize the word error rate. A typical curve is plotted in fig.(6.6), where the burst SNR=-8 dB. The optimum decoder results are plotted also for comparsion.

Figs.(6.3) and (6.4) show clearly that at low error-thresold values, the word error rate shoots up. This is somehow expected, because at low error-threshold values almost every error flag present in the j error-patterns of Eq.(6.1) has affected a digit, which makes a proper correction nearly impossible. Then for error-threshold values larger then the intersection value the decoder introduces an improvement over the error-trapping or the optimum decoders. This intersection value is nearly fixed for different values of burst SNR for the burst code decoder, while its value changes with the SNR for the random code. The variation suggest that al low SNR a lower value of error-threshold should be used and visa versa.

6.8.2 The Digit SPT Decoder Performance

The digit SPT decoder performance is studied using the above mentioned two codes, namely the (31,21) random code, and the (34,22) burst code. The thresholds used with the two codes are 17 thresholds practically spaced. Three runs are carried out for each test, the SNR values for the (31,21) code are 1, 3, and 6 dB. The first test is for a decoder that has a fixed error-threshold value,



—— OPT.

— — DIGIT SPT

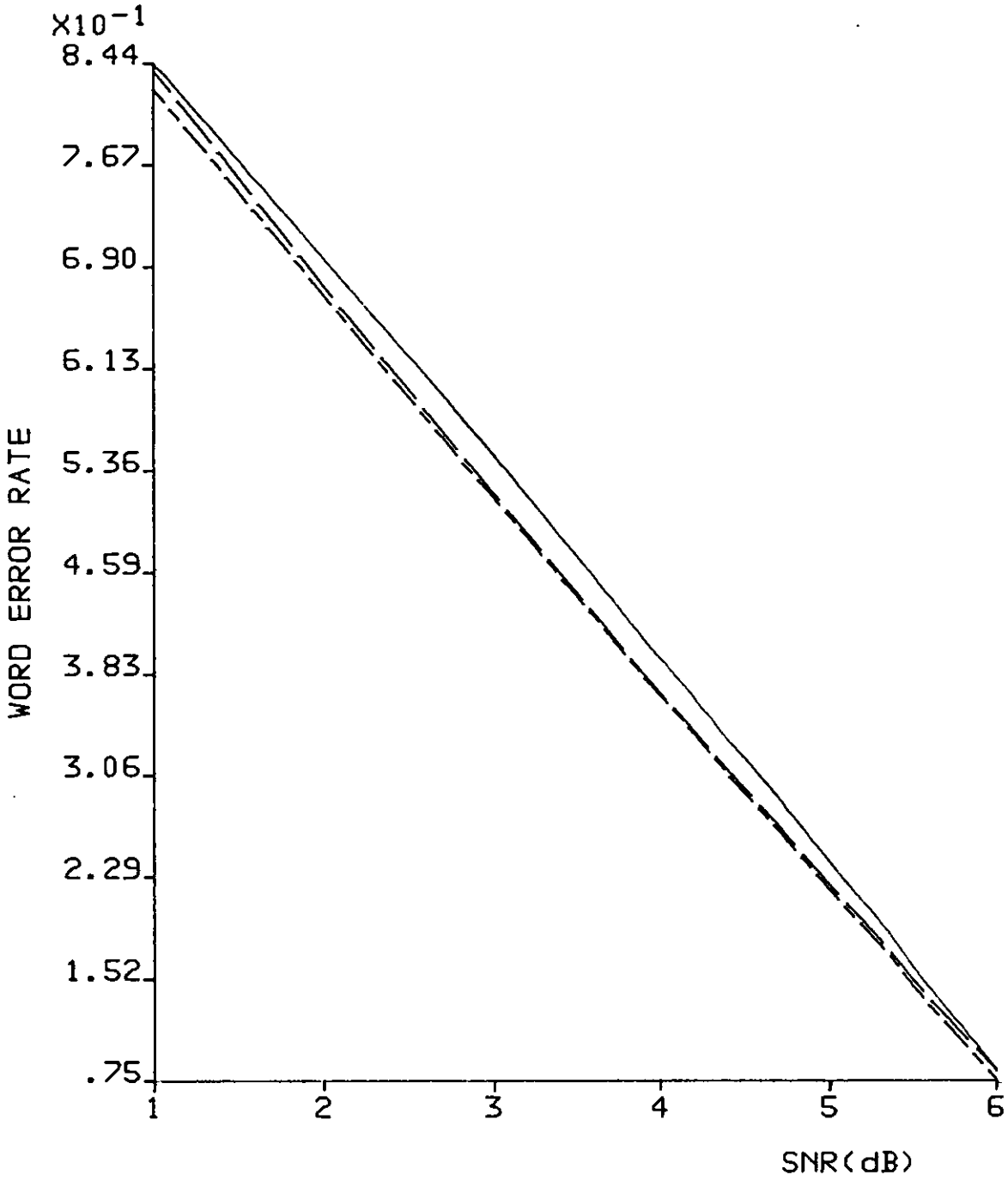
FIG.(6.4) THE EFFECT OF ERROR-THRESHOLD VALUE

the error-threshold value is set to 7. The second test is carried out with a varaible error-threshold values, these values are chosen so that the maximum improvement is achieved for each SNR value. The error-threshold values are 4 for 1 dB, 6 for 3 dB, and 7 for 6 dB. The last test is carried out to show the improvement in case an adaptive error-threshold value scheme is used. The two test results are plotted in addition to the error-trapping decoder results in fig.(6.5). It can be seen that the digit SPT decoder will introduce an improvement of 0.15 dB for the fixed error-threshold value and 0.157 dB for the adaptive error-threshold value over the error-trapping decoder at 3 dB, while the improvement is 0.086 dB over the error-trapping decoder for both decoders at 5 dB.

The SNR values for the (34,22) code are -20, -8,and, 0 dB. The test is carried out for a fixed error-threshold value of 13, there was no need for adaptive decoder tests because the chosen value minimize the word error rate for the three chosen SNR values. The test results and the optimum decoder results are plotted in fig.(6.6). It can be seen that the digit SPT decoder performs better than the optimum decoder for all the region and produces an improvement of 3.6 dB at burst SNR of -17 dB, and 0.64 dB at burst SNR of -3 dB over the optimum decoder.

6.9-Conclusions

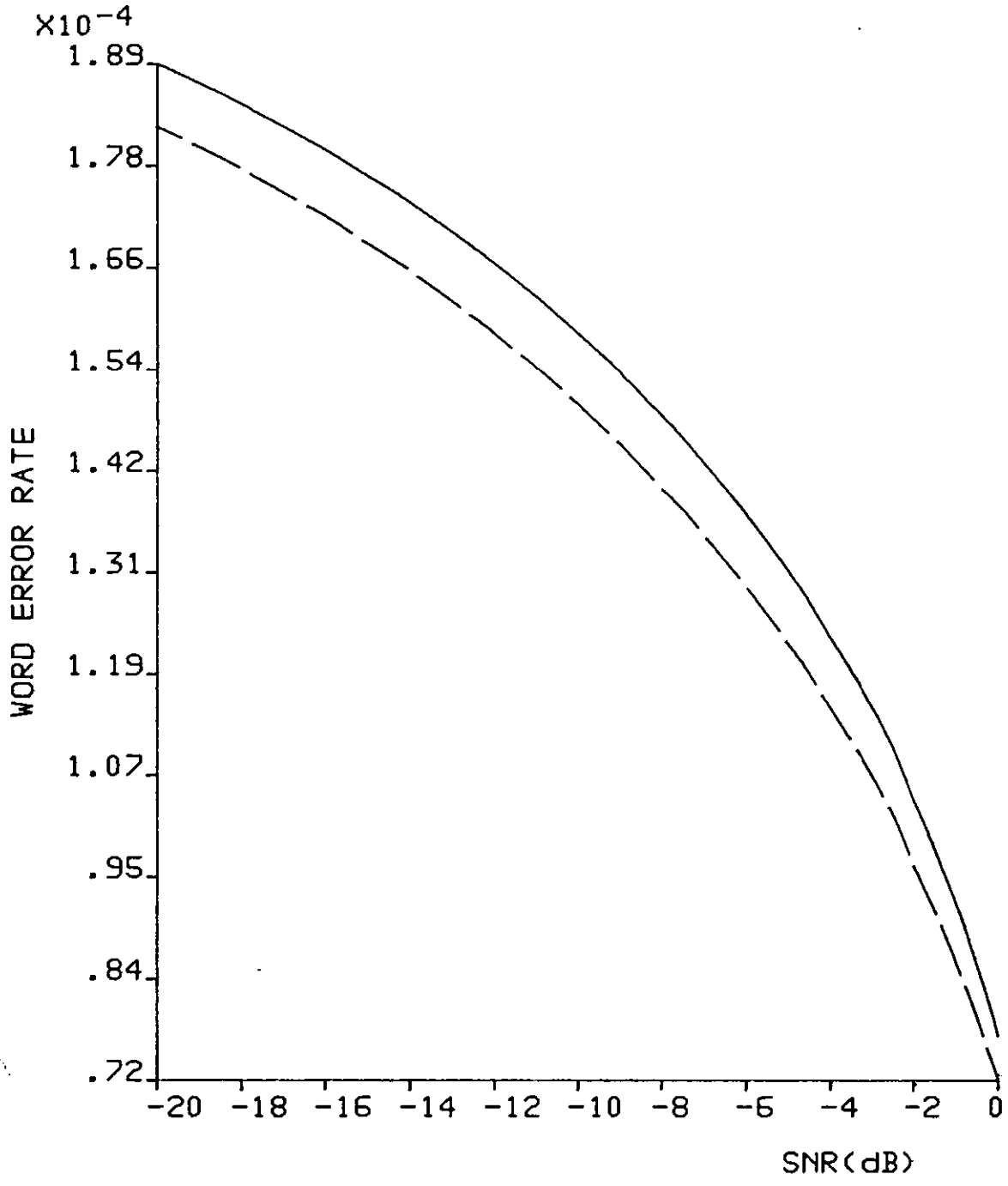
It would be very unwise to draw major conclusions from the results obtained so far. However the results



____ E.T. ___ ADAPTIVE SPT

__ __ FIXED SPT

FIG.(6.5) DIGIT SPT DECODER PERFORMANCE



____ OPT.

_ _ DIGIT SPT

FIG.(6.6) DIGIT SPT DECODER PERFORMANCE

obtained give the assurance that the basic idea is correct, and that a wide variety of modifications can be incorporated with the decoder to provide a better improvement some of these modifications are suggested in the next section.

Nevertheless, considering the digit SPT decoder, the results of section 5.6.1 shown in figs.(6.3) and (6.4) suggest that there exists some inversely proportional relation between the error-threshold value and the SNR value. That is, when the SNR value is high, lower value for error-threshold should be used, and visa versa. According to the simulation results, it seems that the penalty paid in increasing the word error rate by using higher error-threshold value is very small. Hence it can be beneficial to use a higher error-threshold value to prevent a performance deterioration if the channel SNR value dropped to a value lower than the expected.

The digit SPT decoder for random error-generating channels test results suggests that an adaptive system is preferred in the case of changing SNR values especially if the SNR value is low. While error-threshold value is preferred otherwise because the decoder is less complicated.

The simulation results show that at low SNR values the improvement obtained from the digit SPT decoder is higher than at high SNR values, yet there is a small improvement at high SNR. Which suggests that these types of decoders can be used over a wide range of SNR values.

The only thing to say about the error-pattern SPT decoding, if any, is that the success of the digit SPT

decoding enhances the belief that the error-pattern SPT decoding will prove a success also.

6.10-Suggestions For Further Study

Two types of suggestions will be considered for further study of the SPT decoding techniques. The first type is based on the observation of the digit SPT decoders behaviour and concerns only the digit SPT decoders. Although the second type is based on the digit SPT decoders, but they are general suggestion which can apply to both types of SPT decoders. These suggestions are mostly concerned with the improvement of the decoder performance in general, because it was believed that the improvement resulted from these suggestion is worth considering. However many areas of unfinished work needs further study, but it was felt that these are very obvious to mention.

6.10.1 The Digit SPT Decoder

1-The use of adaptive error-threshold value. The significance of using an adaptive error-threshold value is clear from fig.(6.5), where a higher improvement can be achieved. The use of adaptive error-threshold value is limited to the random code decoders, because no real improvement can be achieved for the burst code decoders. Furthermore, they should be used only when they are needed, i.e. when the SNR variation is large. In case of small SNR variation, the error-threshold value can be increased, so that the drop in SNR value will not cause a deterioration in the decoder preformance.

A simple method for calculating the value of the error-threshold, is to start with a fixed value and monitor the number of the inverted digits in a received block, if the number is larger than a set value, say 2t, then the error-threshold is increased. On the other hand if the number of the inverted digits is small, then the error-threshold value is decreased.

Needless to say that the adaptive digit SPT decoder is more complicated to implement then the digit SPT decoder.

2-Extending the correction power of the decoder. This modification is applicable only to the random error decoder. The extended correction power decoder will introduce an additional improvement over the conventional decoder, only if the error-trapping decoder can not trap all correctable error-patterns. As it is shown in section 6.7.1, the correction power should not be extended to more than t+1 errors. However, the improvement expected from the correction power extention is not very big, but it can be considerable in some special cases, i.e. when some codewords are not used.

6.10.2 The SPT Decoder

This modification is applicable to digit SPT decoder, and error-pattern SPT decoders. Its aim is to force the SPT decoder to function the same as the decoder it is derived from in the worst case. The modification is as the following:

The SPT decoder is forced to look at the 0 volt threshold error-pattern before starting the decoding process.

If the error-pattern is zero, a correct received word is assumed, while if the error-pattern is trappable, the trapped error-pattern is assumed to be the actual error-pattern added to the transmitted codeword, and the correction is done accordingly. While if no trappable error-pattern is detected, the SPT decoding process is continued.

For the digit SPT decoder, this modification has the effect of pushing the parts above the horizontal line, figs.(6.3) and (6.4), of the conventional decoder down to coincide with the horizontal line. When such modification is incorporated in the decoder, the error-threshold value no longer has a deteriorating effect on the decoder, although it still has an affect on the decoder improvement.

APPENDIX A

PARALLEL-THRESHOLD-DECODER HARDWARE TEST SYSTEM

A.1-General

The ideal system for testing the parallel threshold decoder is a real transmission system which has a block digram as in fig.(3.1) where the demodulater delivers a sampled value of the demodulated signal (not a binary value), and the 'Channel Decoder' represents the parallel threshold decoder. Such a system is ideal but too complicated and too expensive to be built for testing purposes. The same results can be obtained by simlifying the system of fig.(3.1) as follows.

1. Using a binary data 'Source' and 'Sink',the 'Source Encoder' and 'Source Decoder' can be eliminated from the diagram.

2. By assuming that the channel can handle binary signals and that noise is an additive noise only, the 'Modulator' can be discarded, while the 'Demodulator' can be replaced by a sampling circuit.

After, these two modifications the system will be as in fig.(A.1)

A.2-Repeated Same Word Transmission

Further simplification can be achieved by transmitting the same codeword repeatedly because the 'Encoder' can be excluded. On the other hand the 'Source' has to generate the repeated codeword (including the parity-check digits) instead of generating a random sequence of binary digits.

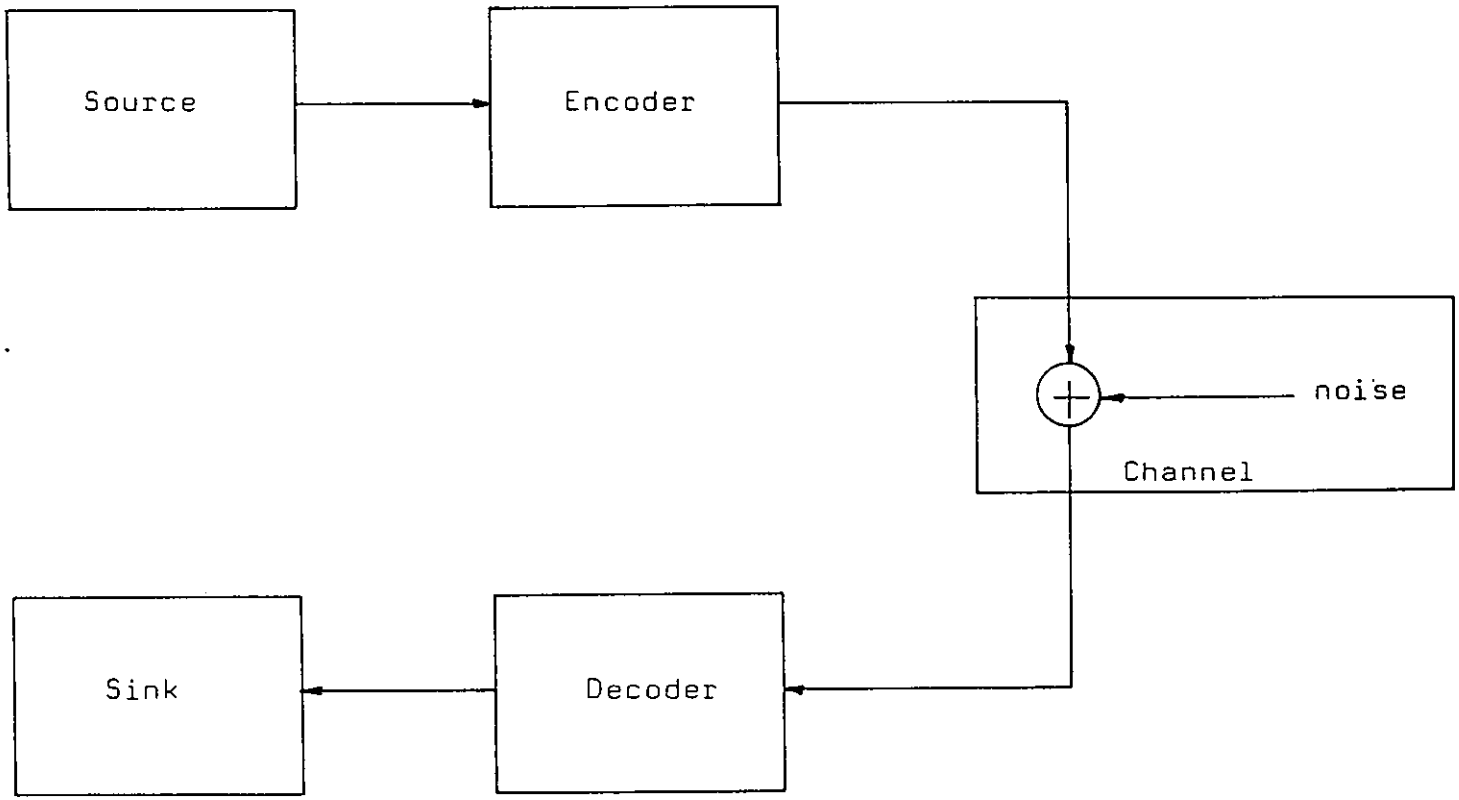


Fig.(A.1) Block diagram of the modified communication system

This in turn adds to the complexity of the 'Source' circuit in the test system. One method of overcoming this complexity is by using an all ones word or all zeros word, if these are codewords. So that the whole upper branch of figs.(3.1) or (A.1) is a wire connected to the positive or negative voltage to represent the all ones or all zeros word respectively.

In general, since all codewords are equally likely to be transmitted,then the use of repeated transmission of the same codeword will result in the same test outcome. Unfortunately, this is not the case with the parallel threshold decoder, because if the optimum threshold for that codeword is used, then the outcome is much better than the outcome of the transmission of random codewords. This false improvement depends on the transmitted codeword and the distance between the threshold used and the optimum threshold value. The maximum false improvement will be achieved in the special case where an all ones or all zeros word is transmitted repeatedly.

Consider the case where and (n,k) code is used, it is required to calculate the probability of errors when an optimum threshold is used to decode an all ones transmitted codeword. Eq.(5.47) gives the distance of the optimum threshold.

$$d_1 = V + \frac{\sigma^2}{2V} \log \frac{q_1}{q_0} \qquad (5.47)$$

$$d_1 = V + \frac{\sigma^2}{2V} (\log q_1 - \log q_0) \qquad (A.1)$$

Since the transmitted codeword is all ones, then

$$q_1 = 1 \tag{A.2}$$

and

$$q_0 = 0 \tag{A.3}$$

Substituting in Eq.(A.1) yields

$$d_1 = V + \frac{\sigma^2}{2V} (\log 1 - \log 0) \tag{A.4}$$

$$d_1 = -\infty \tag{A.5}$$

Since the reference point for the threshold distance is at +V volts and the direction of d_1 is opposite the X axis direction, then the threshold value is

$$Th = +\infty \text{ volts} \tag{A.6}$$

The probability of error is given by Eq.(5.31)

$$P_{e_1} = q_1 \int_{-\infty}^{d_1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-v^2}{2\sigma^2}\right) dv \tag{5.31}$$

Substituting Eq.(A.5) in Eq.(5.31)

$$P_{e_1} = q_1 \int_{-\infty}^{-\infty} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-v^2}{2\sigma^2}\right) dv \tag{A.7}$$

$$P_{e_1} = 0 \tag{A.8}$$

Although, an error-trapping decoder will not give the results of Eq.(A.8), a parallel threshold decoder with

three thresholds placed at zero and $\pm\infty$ will give a very close error probability to Eq.(A.8), provided the $+\infty$ threshold output is decoded before the $-\infty$ threshold output.

Similarily, the same results can be derived for the all zeros transmitted codeword.

A.3-Choosing The Code And The Codeword

To be able to compare the hardware test results with the computer simulation results, it is clear that one of the simulated codes should be used. Although some burst-error-correcting codes perform very well and give large improvements, it was decided against using one of these codes for two reasons.

1-A burst-noise generator is not available, and building one is a costly process and will add to the system complexity.

2-The use of burst-error-correcting code requires deinterlacers at the input, which again adds to the system cost and complexity, while if the code is not interlaced it will lose much of its correction power.

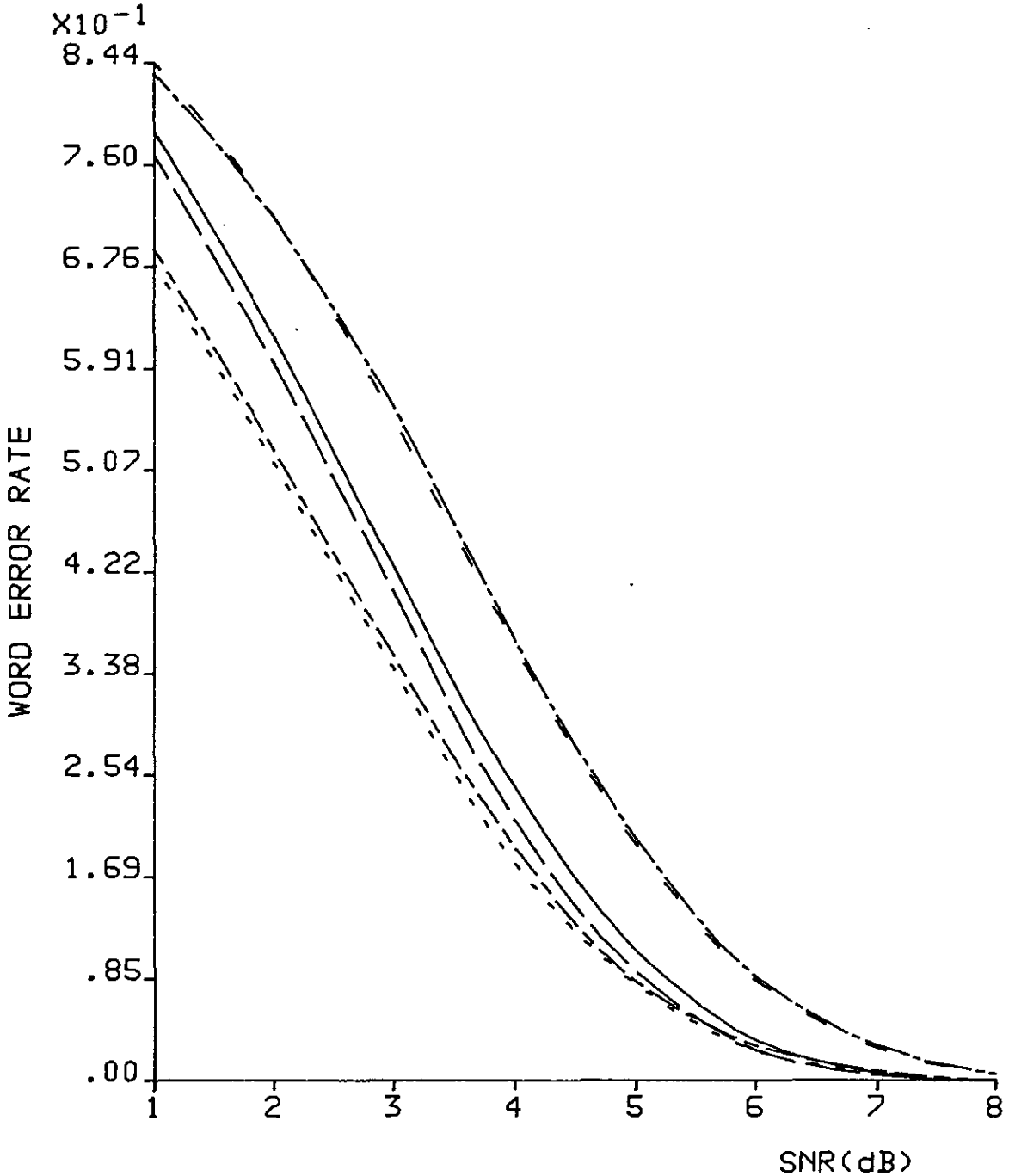
With the elimination of the burst-error-correcting codes, two random error-correcting codes are left to choose from. It was decided to use the (31,21) random code because it preforms better than the (15,7) random code.

The other point to decide is the codeword. For sake of simplicity it was decided to use a repeated transmission of the all ones codeword. To accommodate for the false

improvement, a repeated transmission of all ones codeword is simulated, the results of simulating repeated all ones codeword and random codewords is ploted in fig.(A.2), where it can be seen that an improvement of 0-0.5dB, and 0-0.6dB has taken place for the 5, and 3 thresholds parallel decoder respectively, while the error-trapping decoder degradation can be ignored.

A.4-The Test System

A block diagram of the test system is shown in fig.(A.3). The input signal is provided by connecting the channel input to a ligical '1'. So that all ones codeword is generated. At the channel, noise is added to the channel input signal and the output is the received signal. To start the test, the 'Error Words Counter' and the '3000 Word Counter' are reset to zero, the later will cause 'Gate 1' to turn on allowing the received signal to be fed to the 'Parallel Threshold Decoder'. As soon as the received word has entered the 'Parallel Threshold Decoder', a signal is sent to the '3000 Word Counter' to advance its count by one. At the end of the decoding, if the decoded word is not the all ones word, the 'Error Words Counter' advances its count by one, otherwise it remains as it was. As soon as the decoding is ended, the next received word is fed to the 'Parallel Threshold Decoder'. Once the 3000th word is fed in, the '3000 Word Counter' turns 'Gate 1' off, indicating the end of the test process. The number of the test words is chosen to be 3000 (93000 bits), so that it will be the same number used in the computer simulation.



3 TH.
3 TH. ALL 1s
E.T.
5 TH.
5 TH. ALL 1s
E.T. ALL 1s

FIG.(A.2) FALSE IMPROVEMENT FOR ALL '1's CODEWORD

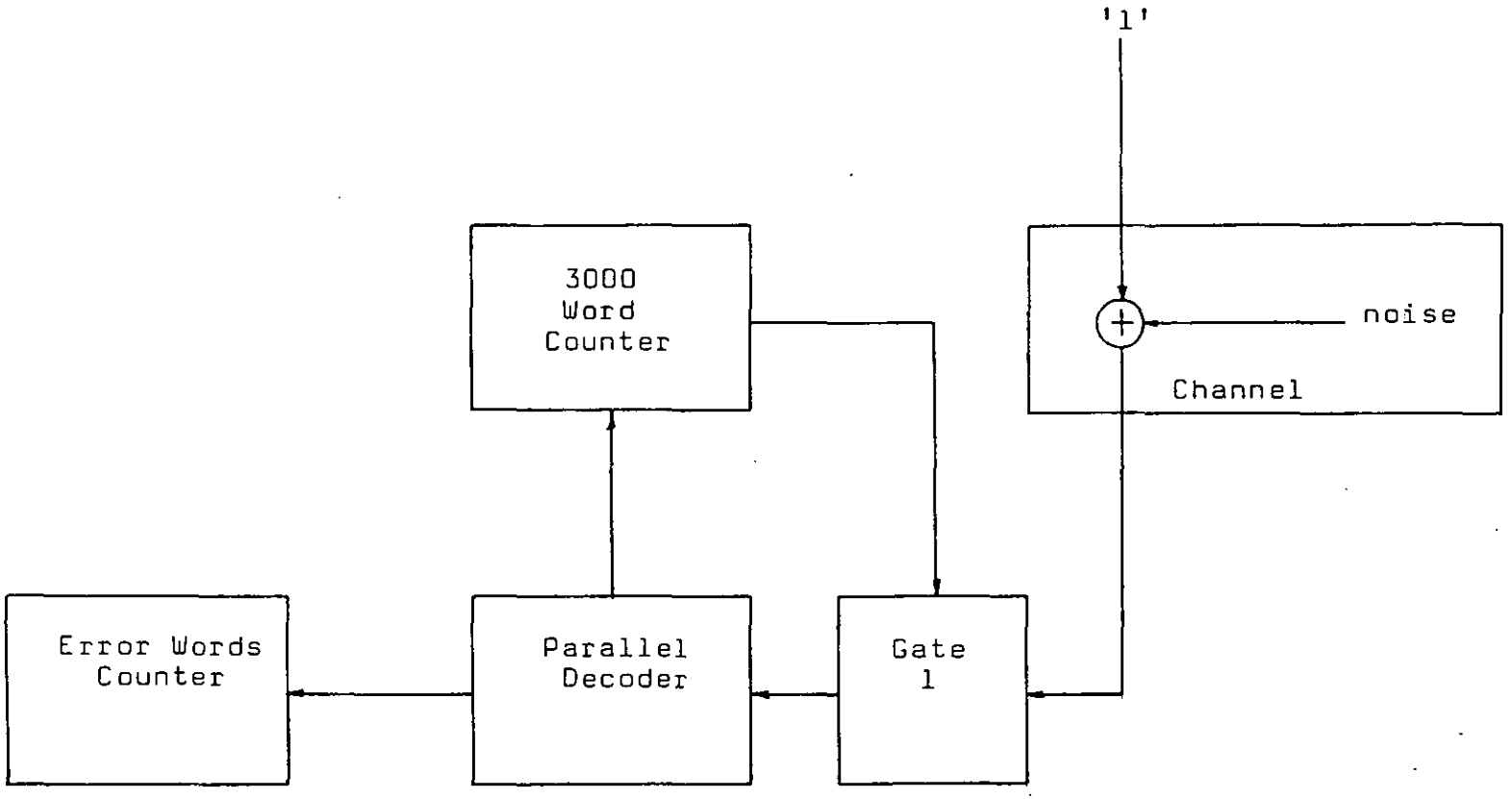


Fig.(A.3) Block diagram of the test system

The test system arrangement is shown in fig.(A.4), which is an expanded version of fig.(A.3). The difference between the two figures is :-

1-As the 'Parallel Threshold Decoder','3000 Word Counter' and 'Error Words Counter' are digital circuits, they require a system clock to drive them, this clock is introduced in fig.(A.4). Since without clock pulses, the decoding system will stand still. Then a clock stop signal is used to stop the system, by inhibiting the clock, after the decoding of 3000 words. Using this arrangement, 'Gate 1' of fig.(A.3) is no longer required.

2-The 'Channel' block in fig.(A.3) is expanded to include noise generation and addition blocks. The noise generator is an analogue random Gaussian noise generator. Gaussian noise is used to match the noise used in the simulation. The noise generator has a maximum value of 1 volt at its output, and since higher output values are required, an amplifier is used to get a suitable noise output. Again the noise generator has an output noise bandwidth of 20Hz-20MHz, so a low pass filter is used to band limit the noise. At the output of the filter an RMS voltmeter is used to measure the noise to be added to the signal in the channel. The addition ciruit is an analogue addition circuit, which adds the generated noise to the signal at the channel input.

A.5-The Parallel Threshold Decoder Circuit

The parallel threshold decoder circuit is of special interest because it gives an idea of the decoder complexity and so the circuit will be discussed in some details. The

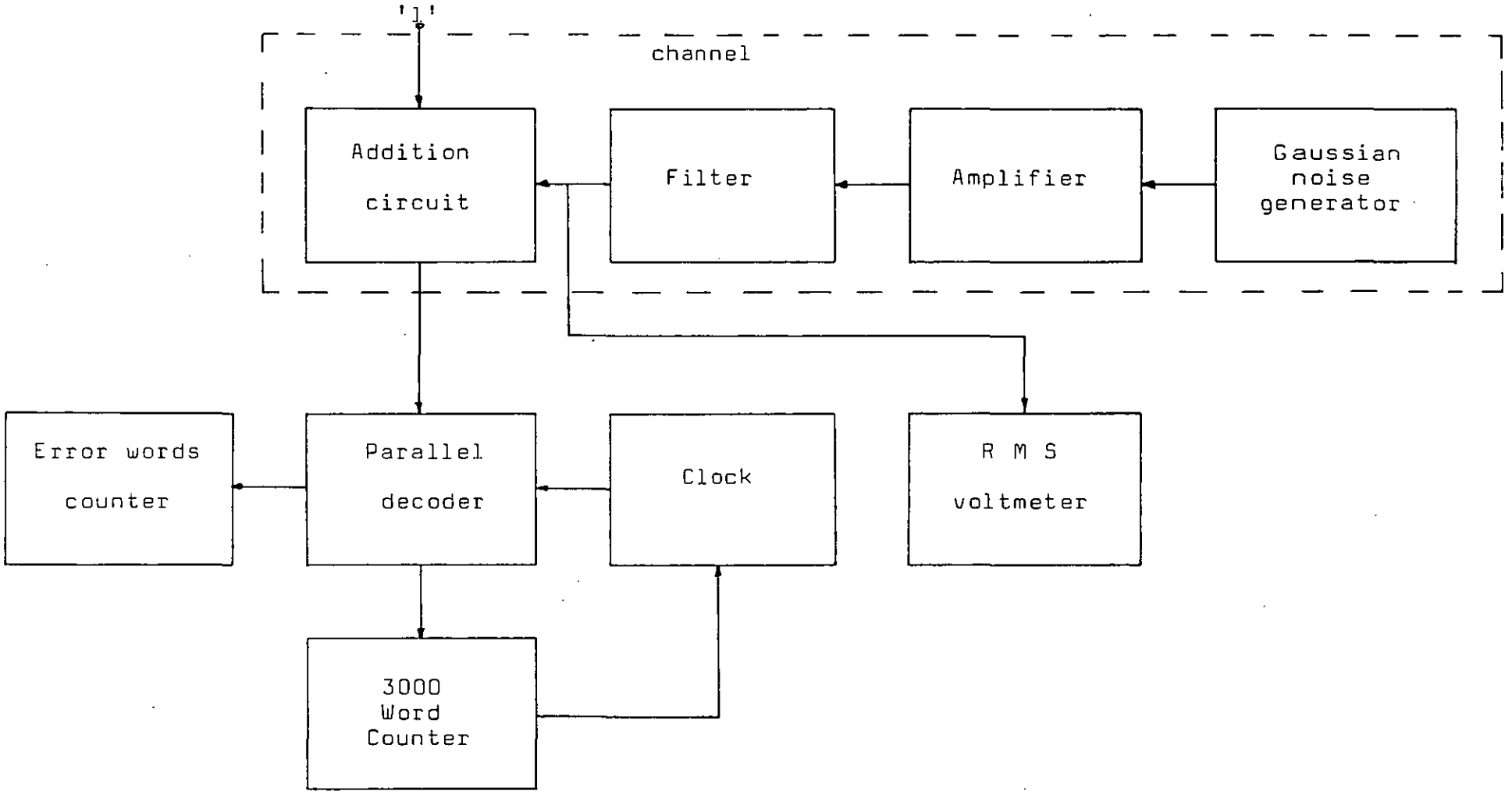


Fig.(A.4) Test system arrangement

circuit used is a modification of a circuit designed by Mr. D.G.King[52], although its function is exactly the same as a parallel threshold decoder, it works in a serial mode. This mode incurs a time penalty over the parallel system, but it requires fewer devices to implement, while the control circuitry is slightly more complicated. A block diagram of the decoder is shown in fig.(A.5). The channel output signal is fed to the 'Input Threshold Detectors', where it is compared with the various threshold values, the output of the third threshold (0 volt threshold) is fed directly to the error-trapping decoder, while the output of the other thresholds are stored in the four auxiliary registers 1,2,3, and 4. At the error-trapping decoder, the information digits are stored in the '21 Stage Main Data Register', at the same time the complete received word is shifted into the syndrome register. As soon as the whole word is shifted in, the syndrome is calculated at the syndrome register and if it is all zeros the received word is correct and is passed to the decoder output. If the syndrome is not all zeros, it is shifted cyclicaly until a trappable error-pattern is detected, and the received word is corrected. If at the end of 21 cyclic shifts of the syndrome register no trappable error-pattern is detected, the syndrome register is reset to zero, and the next threshold output, which is stored in an auxiliary register, is fed to the error-trapping decoder, and new correction is started. If at the end of this correction phase no trappable error is detected, the next threshold output is considered, and so on.

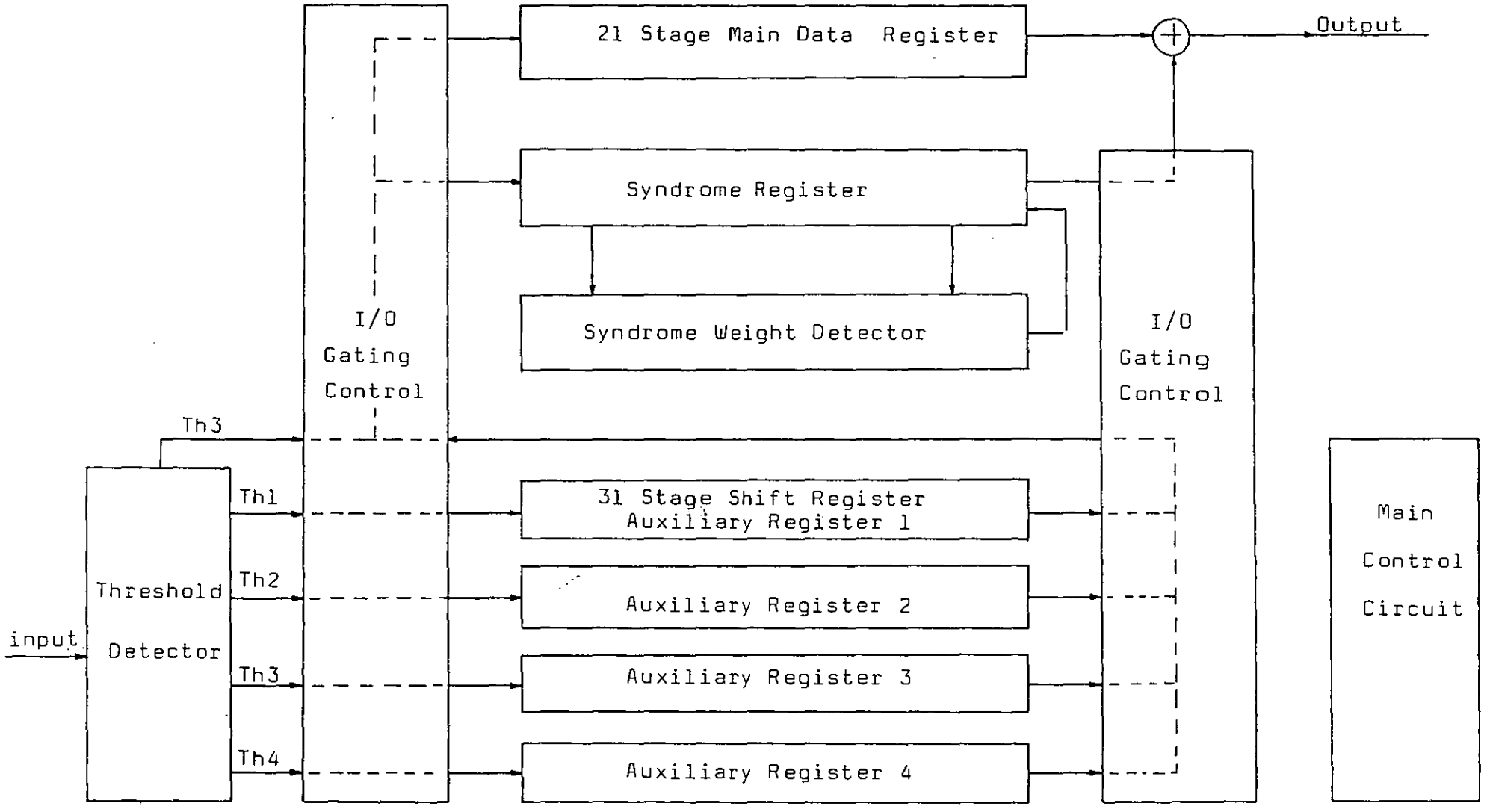


Fig.(A.5) Block diagram of the parallel threshold decoder

If at the end of the five correction phases no trappable error is detected, an erroneous decoded word is assumed. Once a trappable error is detected, and the received word is corrected, no matter which correction phase the decoder is in, the whole system is reset, and a new received word is accepted. The threshold outputs are considered in a fixed sequence 3,2,4,1,5 each time a new received word is fed in from the channel.

The parallel threshold decoder circuit diagram is shown in fig.(A.6). Before going into the circuit details it is worthy to consider the EPROM (A38) outputs fig.(A.7) because it is the main control circuit. The EPROM is an (64x8) memory device, each output being allocated a specific task in controlling the circuit.

E1 : is the 'Information Digits Input Control' and is used to turn gate A34a on for the first 21 clock waveforms of each detection phase, to allow the first 21 digits of the processed word to be stored in the main data register (A23 and A24).

E2 : is the 'Syndrome Input Control', it is used to turn gate A34b on at the beginning of each phase, to allow the whole processed word to be shifted in the syndrome register (A25, A26, A27, and A28). Once the 31 bit word is in (during the first 31 clock waveforms), gate A34b is turned off.

E3 : during the whole decoding process, in any phase, the syndrome Hamming weight is constantly being tested by the weight detector. To stop any action being taken if a trappable error is detected during the syndrome

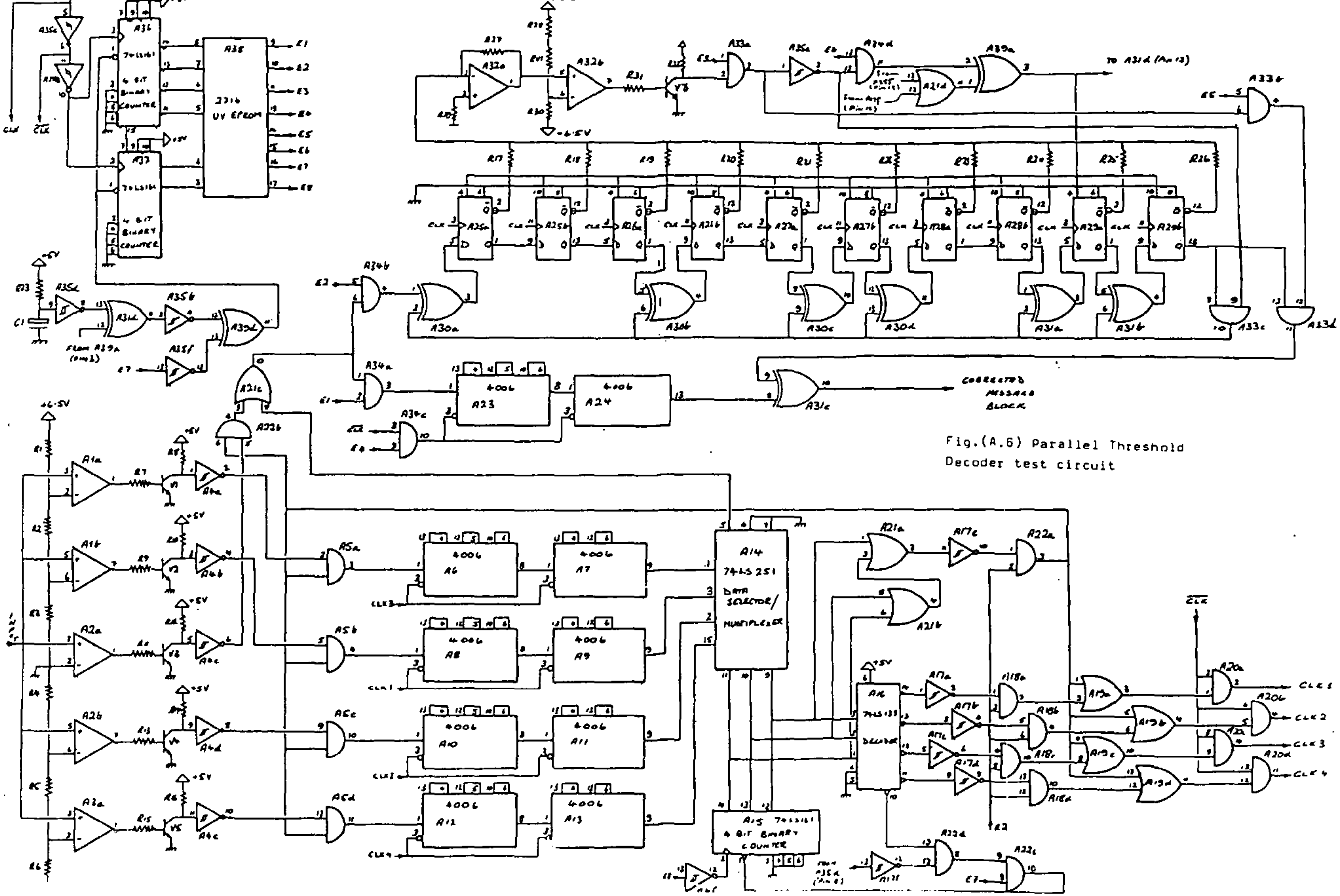


Fig.(A.6) Parallel Threshold
Decoder test circuit

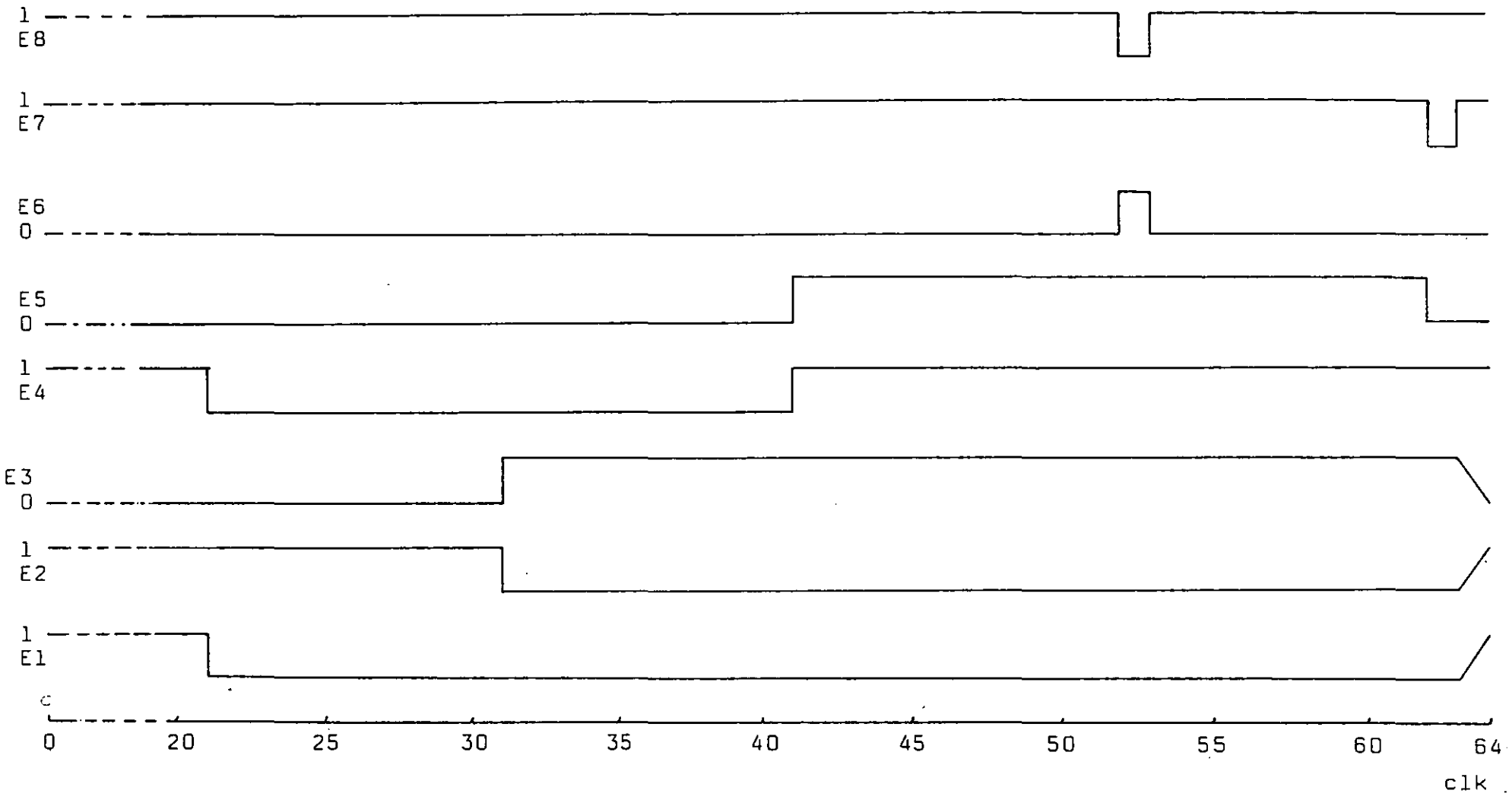


Fig.A.7 EPROM output waveforms

calculation, E3, the 'Weight Control Output' turns gate A33a off for the first 31 clock waveforms, and then on for the rest of the decoding phase.

E4 : the 'Information In/Out Control', is used to control the shift clock in the main data register. It turns A34c on during the first 21 clock waveforms to allow the information digits to be shifted in. Again it turns gate A34c on during the 41st clock to enable the read out of the information digits.

E5 : the 'Correction Control', is used to route the output of the syndrome register to the correction gate A31c, which adds the error-pattern to the information digits as they are simultaneously read out of their respective registers after correctable errors have been detected. Since the correction process may start after the 41st clock waveform, E5 turns gate A33b on during the 41st clock, and turns it off at the end of the correction process, during the 62nd clock waveform.

E6 : is used as an 'Untrappable Error Pattern Reset', which turns gate A34a on during the 52nd clock cycle of each detection phase, causing a reset pulse to be generated if no trappable error is detected by then. The reset pulse ends prematurely the current detection phase and starts the next phase.

E7 : or the 'Trappable Error Pattern Reset'. When a trappable error pattern is detected, E7 generates a reset pulse after the end of the correction process, to reset the whole system. So that the next received word decoding can start. The reset pulse will be generated

whenever a trappable error pattern is detected and corrected during any of the five detection phases.

E8 : is used as a 'Phase Advance Clock', to increase A15 count by one at the end of each detection phase, so in the next phase the received word output is connected to the error-trapping decoder input.

The best way to describe the function of the serial mode parallel threshold decoder of fig.(A.6) is to follow a received word from the input throughout all the storeing, detection, and correction processes.

Once the power supply is switched on a reset pulse is generated by R33 and C1, resetting the EPROM address counters A36 and A37 to zero through A35a, A31d, A35b, and A39d, and to reset A15 to zero through A35a, A17f, A22d, and A22c while the syndrome calculator A25....A29 are reset to zero through A35a, A17f, A21d, and A39a. By this time the system clock is inhibited. As soon as the supply voltage is on, the input circuit is functioning, although no clock is present. The channel output signal is fed to the decoder input, which is feeding the five threshold gates A1, A2, A3a in parallel. Each threshold detector will output '1' if the input signal is larger than its referance voltage, transistors V1, V2, V3, V4, and V5 convert the threshold detectors output to a signal compatible with logic circuits, while invertors A4a, A4b, A4c, A4d, and A4e invert the signals to the threshold detectors polarity. It can be seen that no sampling is associated with the threshold detectors input, so the output follows the input signal.

The zero address at the EPROM input causes gates A34a, and A34c, to turn on, so that information digits can be read in. Also gate A34b is turned on to enable the received word to be shifted in. Simultaneously, the zero count at the output of A15, and E2 output of the EPROM, turns gate A22b on allowing the output of the third threshold detector (0 voltage threshold) to be shifted into the error-trapping decoder, and turns on gates A5a, A5b, A5c, and A5d to allow the storage of the first, second, fourth, and fifth threshold detector output in A6, A7, A8, A9, A10, A11, and A12,A13 shift registers respectively.

The signal present at the threshold detectors in synchronism with the clock are accepted as the same outputs of the sampled signal at the output. Once the clock system is enabled, the signal at the output is shifted into A6....A13 registers, the main data register A23 and A24, and into the syndrome register A25, A26, A27, A28, and A29. By the time the first 21 digits are shifted in, Gates A34a and A34c are turned off by E1 and E4, to stop the parity check digits being shifted into the main data register A23 and A24, while the shifting in is continued for the other registers. As soon as the whole word (31bits) is shifted in gates A5a, A5b, A5c, A5d, A22b, and A34b are turned off by E2. By this time the syndrome is calculated by the syndrome register A25....A29. The Hamming weight of the syndrome is tested by the weight testing circuit A32a where each stage output of the syndrome register A25a... A29b is fed through the same registers R17....R26

to an analogue addition circuit A32a, its output being linearly proportional to the Hamming weight of the syndrome contents, which is compared with the Hamming weight of three. When the syndrome weight is less than three the comparator A32b will output a negative voltage, which is translated to logical '1' by V6. Once the weight goes to two or less after the syndrome is calculated, E3 turns gate A33a on allowing the weight detector output to pass throught to turn gate A33c off, so that the error-pattern is trapped in the syndrome register. If the error-pattern is trappable during the first ten shifts of the syndrome register after calculating the initial syndrome (between 31-41st clock cycles), shifting the syndrome is continued to the 41st clock cycle; so that the trapped error-pattern will be in the right place for the correction. On the other hand if an error-pattern has not been trapped before the 42nd clock cycle gates A34a and A34c are turned on by E1 and E4 and the information digits are read out, E5 will turn gate A33b on to allow the weight detector output to pass through. The information digits are read out in synchronism with the clock and at the same time the syndrome register is shifted. As soon as an error-pattern is trapped, the weight detector turns gate A33d on through gate A33b allowing the erroneous digits to be corrected by A31c. By the time the syndrome register is shifted 21 times after calculating the syndrome (at the end of the 52nd clock cycle) E8 will advance A15 count by one preparing for the next phase, if no error-pattern is trapped, the decoding process is prematurely terminated

by reseting to zero the syndrome register by E6 through A34d and A39a, and the EPROM address counters by E6 through A34d, A39a, A31d, A35b, and A39d.

The next phase starts after a decoding failure in the previous phase. Assuming that the current phase is phase two, then the output of A15 is one as a binary number, which will connect the second auxiliary register A8 and A9 the output of the second threshold detector through A14 to the error-trapping decoder input A21c, and cause the output of A16 to turn gate A20a on through A17a, A18a, and A19a to allow the clock (CLK2) for reading out the contents of the auxiliary store, to pass through. CLK1 will shift the third threshold detector output stored in A8 and A9 only into the error-trapping decoder, where a new decoding phase is started. At the end of the second detection phase, if no error-pattern is trapped, phase two is terminated after advancing A15 by one, and the third detection phase is started, where the content of the third auxiliary register (the output of the fourth threshold detector) is fed to the error-trapping decoder, and so on.

If an error-pattern is trapped during any phase, the correction process is allowed to be continued until all the 21 information digits are read out. The decoding process is terminated by reseting the whole system, which is done by E1, the EPROM address counters are reset through A35f, and A39d, the syndrome register is reset through A35f, and A39a, while A15 is reset through A22c.

On the other hand if no error pattern is trapped in the last detection phase, the system reset is different.

Since the phase will be terminated prematurely, then the EPROM output E6 will clear the syndrome register through A34d, and A39a, and the EPROM address registers through A34d, A39a, A31d, A35b, and A39d. The system reseting requires the additional reseting of A15, which is achieved by reseting A15 by the appropriate output from A16 through A22d, and A22c. Using this arrangement makes it possible to use this same decoder as a five threshold parallel-decoder, by connecting output five (pin10) of A16 to A22d (pin13), while a three threshold parallel-decoder is achieved by connecting output three (pin12) of A16 to A22d (pin13). Again the decoder can be used as a conventional error-trapping decoder by connecting output one (pin14) of A16 to A22d (pin13). These connections have the effect of terminating the decoding after the fifth, third, and first decoding phase respectively.

A.6-Test Results

The noise RMS voltage is measured by a slow RMS voltmeter, it is connected to the system as in fig.(A.4). The signal to noise ratio is given by

$$SNR = 20 \log \frac{S}{N} \qquad (A.9)$$

where SNR is the signal to noise ratio in dbs, S signal voltage, and N noise voltage. Given for the test that

$$S = 5 \text{ volts} \qquad (A.10\text{-a})$$

$$N = \text{RMS volts} \qquad (A.10\text{-b})$$

Eq.(A.9) becomes

$$SNR = 20 \log \frac{5}{RMS} \qquad (A.11)$$

Eq.(A.11) is used to calculate the RMS values for the required SNR values, where

$$RMS = \frac{5}{\log^{-1}\frac{SNR}{20}} \qquad (A.12)$$

Five test runs were carried out for each SNR value, the number of erroneous words were counted for each run, and the average number calculated. These tests were repeated for five thresholds, three thresholds, and the conventional error-trapping decoder. The threshold values used for the five threshold parallel-decoder were 1.43, 0.8, 0, -0.8, -1.43 volts, and the values for the three thresholds parallel-decoder were 1.12, 0, -1.12 volts. The results of these runs are shown in tables (A.1-3)

The computer simulation results, and the hardware test results are ploted together in fig.(A.8). Both results show nearly the same improvement over the error-trapping decoding, but it can be seen that the hardware test results are shifted by about 2dB towards a lower SNR. This is because although a slow RMS voltage meter was used, it was very difficult to accommodate for the noise variations, so the noise voltage was set to be at the correct value for most of the run time, but that would not compensate for the sudden increase in the noise voltage.

SNR(dB)	WORD ERROR RATE					AVERAGE ERRORS
1	2950	2980	2979	2955	2960	2964.8
2	2695	2802	2844	2792	2822	2811
3	2599	2594	2602	2605	2593	2598.6
4	2293	2296	2279	2280	2278	2285.2
5	1671	1683	1678	1666	1670	1673.6
6	1147	1125	1117	1098	1130	1123.4
7	583	585	603	590	592	590.6
8	258	255	263	261	242	255.8
9	76	64	70	68	62	68
10	17	19	17	18	16	17.4

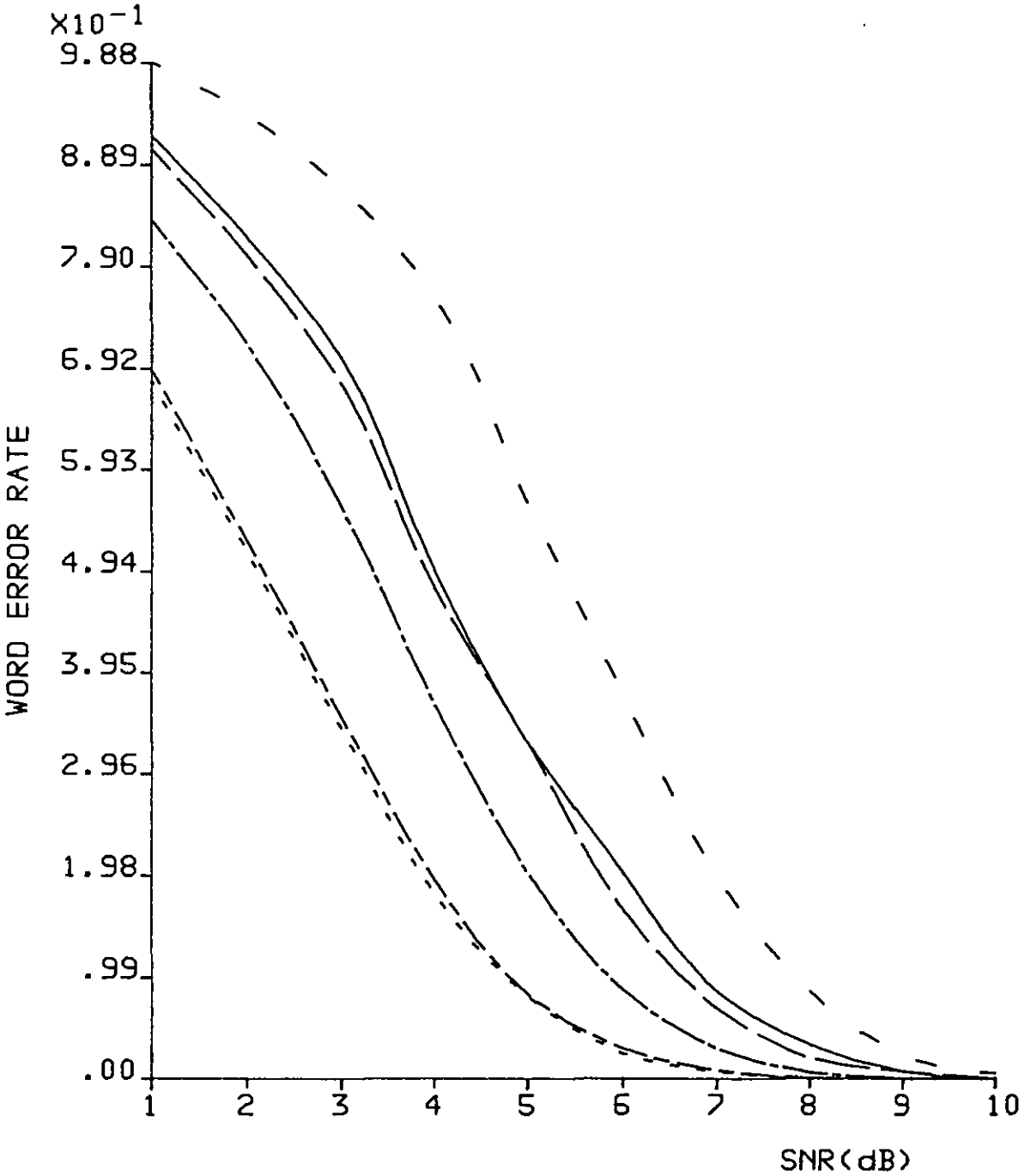
TABLE (A.1) ERROR-TRAPPING DECODER TEST RESULTS

SNR(dB)	WORD ERROR RATE					AVERAGE ERRORS
1	2765	2780	2769	2709	2707	2750.2
2	2455	2459	2450	2461	2426	2454
3	2091	2108	2108	2125	2080	2102.4
4	1509	1471	1480	1485	1469	1482.8
5	963	988	983	976	976	977.2
6	604	597	590	609	595	599
7	263	260	243	250	247	252.6
8	105	101	103	99	87	99
9	21	20	18	19	23	20.2
10	2	0	0	1	2	1

TABLE (A.2) THREE THRSHOLDS PTD TEST RESULTS

SNR(dB)	WORD ERROR RATE					AVERAGE ERRORS
1	2742	2701	2727	2684	2694	2709.6
2	2410	2396	2423	2392	2389	2402
3	2049	2031	2025	2027	1998	2026
4	1416	1426	1427	1435	1448	1431
5	960	951	989	974	1005	975.8
6	472	502	495	512	480	492.2
7	201	215	230	200	181	205.4
8	53	70	62	55	65	61
9	30	19	25	12	14	20
10	1	2	0	1	0	0.8

TABLE (A.3) FIVE THRSHOLDS PTD TEST RESULTS



____ 3 TH. HARD. __ _ 3 TH. SIM. _ _ E.T. HARD.

_ _ _ 5 TH. HARD. _ _ _ 5 TH. SIM. ___ E.T. SIM.

FIG.(A.8) SIMULATION AND HARDWARE TEST RESULTS

APPENDIX B

THE SHORTENED CYCLIC BURST-ERROR-CORRECTING CODE (19,11)

During the test stages of the parallel threshold decoder by computer simulation, the shortened cyclic Burst-error-correcting code (19,11), was chosen to be used in the test process. This code was listed in two references, as an optimal code, which is capable of correcting all bursts of length ℓ or less, where

$$\ell = \frac{n-k}{2} \tag{B.1}$$

$$\ell = \frac{19-11}{2} = 4 \tag{B.2}$$

but, the generator polynomial g(X), was different in each reference from the other. In reference(48) it was given as

$$g(X) = 1+X^2+X^4+X^7+X^8 \tag{B.3}$$

while in reference(55) it was given as

$$g(X) = 1+X^3+X^5+X^6+X^9 \tag{B.4}$$

it was clear form the beginning that Eq.(B.4) is not the correct one, because the degree of g(X) is nine while it should be of degree n-k, which is eight.

Going back to reference (48)showed clearly that the author of reference (55) miss copied the generator polynomial. He copied the generator polynomial of the shortened cyclic burst-error-correcting code (38,29),

which is a sub optimal code, with a capability of correcting all burst of four errors or less, and has a burst-correcting efficiency Z.

$$Z = \frac{2\ell}{n-k} \tag{B.5}$$

$$Z = \frac{2\times 4}{38-29} = \frac{8}{9} \tag{B.6}$$

while the burst-correcting efficiency Z, for any optimal code is one, from Eq.(B.1)

$$2\ell = n-k \tag{B.7}$$

Substituting Eq.(B.7) in Eq. (B.5) gives

$$Z = \frac{n-k}{n-k} = 1 \tag{B.8}$$

In spite of the knowledge that the generator polynomial of Eq.(B.3) is the correct one; it was decided to search for all the generator polunomials suitable for the shortened code (19,11). The search was done by the use of a computer.

Using the knowledge that the generator polynomial is of degree n-k, a program was written to verify which of the 127 generator polynomial possibilities is a suitable generator polynomial for a shortened (19,11) burst-error-correcting code, which is capable of correcting all burst length of four or less. The computer

test produced six generator polynomials:

1- It was found that the (19,11) code can be derived from the (255,247) burst cyclic code, by inserting $\beta = 236$ zeros. Two generator polynomials were found for such a code.

$$g_1(X) = 1+X^2+X^3+X^4+X^8 \qquad (B.9)$$

$$g_2(X) = 1+X^4+X^5+X^6+X^8 \qquad (B.10)$$

2- The (19,11) code can be derived from the (217,209) code, by inserting $\beta = 198$ zeros. Again two generator polynomials were found for such a code.

$$g_3(X) = 1+X+X^4+X^6+X^8 \qquad (B.11)$$

$$g_4(X) = 1+X^2+X^4+X^7+X^8 \qquad (B.12)$$

it can be seen then $g_4(X)$ is the same $g(X)$ given in Eq.(B.3).

3- The (19,11) code can be derived from the (127,119) burst code, by inserting $\beta = 108$ zeros. The generator polynomial for this code is

$$g_5(X) = 1+X+X^3+X^4+X^7+X^8 \qquad (B.13)$$

4- The (19,11) code can be derived from the (84,76) burst code, by inserting $\beta = 65$ zeros. The generator

polynomial for this code is

$$g_6(X) = 1+X+X^2+X^3+X^4+X^8 \qquad (B.14)$$

To assess the performance of the (19,11) code generated by various generator polynomials, a transmission is simulated for each of the generator polynomials using the optimum decoder. The simulation results are plotted in fig.(B.1). The optimum decoder is used since it is a decoder that will show any improvement in the performance due to the use of different generator polynomials, because the decoder will try to correct any burst of errors even if they are longer than ℓ, and that is the area where each code performance will differ, while all the codes will perform the same for bursts of length ℓ or less.

As a result, the generator polynomial given in Eq.(B.3) and Eq.(B.12) was used in the simulation because the code performs better when this generator is used for a wide range of SNR values.

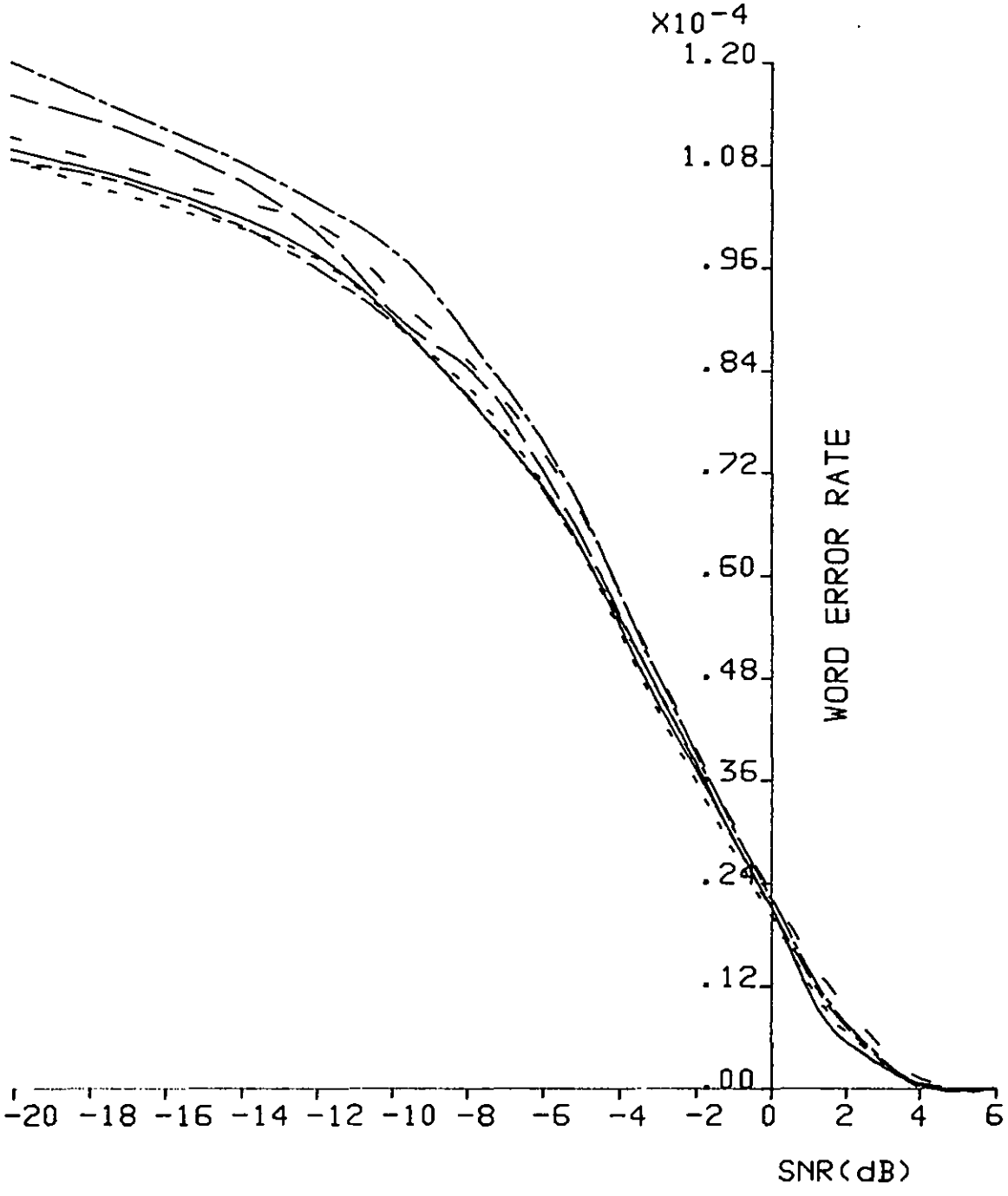


FIG.(B.1) PERFORMANCE OF DIFFERENT g(X)

APPENDIX C

Bit Rate Consideration in Detection and Correction Codes

The performance of error-correction techniques on one-way channels are clearly far superior to the performance of error-detection techniques, because the latter will not result in any improvement in error rate and any detected erroneous information is ignored. On the other hand, both techniques give improvement on two-way channels, but this improvement is at the cost of increasing the transmission bit rate. So it would be reasonable to compare the performances of both techniques at the same constant increase in the transmission bit rate.

Consider the transmission of information digits which are grouped in a block of k digits each, through a transmission channel. This channel can be either a one-way channel or two-way channel, which has in either case a probability p of generating errors. To start with assume that an error-detection and retransmission system is used. Let the error-detection code be (n_d,k) code, where n_d-k digits are annexed for error-detection pruposes. And let the retransmission request require the transmission of n_r digits for each request.

The probability of receiving a whole transmitted block correctly, p_{dc} is the probability of receiving every one of the n_d digits correctly, which is

$$p_{dc} = (1-p)^{n_d} \qquad (C.1)$$

The probability of receiving an erroneous block p_{de} is

$$p_{de} = 1 - (1-p)^{n_d} \qquad (C.2)$$

For simplicity, assume that the transmitter receives all retransmissions and the retransmission requests correctly the average increase in digits per block due to a retransmission δ_r is

$$\delta_r = p_{de}(n_d+n_r)/n_d \qquad (C.3)$$

substituting Eq.(C.2) in Eq.(C.3)

$$\delta_r = \left[1 - (1-p)^{n_d}\right] \left(1 + \frac{n_r}{n_d}\right) \qquad (C.4)$$

Let δ_d be the increase in bits per block due to the addition of the error-detection parity-check bits. Then

$$\delta_d = n_d - k \qquad (C.5)$$

The total increase in bits per block due to the parity-check and retransmission digits is

$$\delta_D = \delta_d + \delta_r \qquad (C.6)$$

from Eqs.(C.4) and (C.5)

$$\delta_D = (n_d-k) + \left[1-(1-p)^{n_d}\right] \left(1+\frac{n_r}{n_d}\right) \qquad (C.7)$$

It is worthy to note that with the detection and retransmission system that the receiving of all the information digits error free is not guaranteed, because any codeword corrupted in the channel to be received as another codeword, will be detected as a correct received word. The probability of error for these cases is given[6,67] as

$$p_{ed} = \sum_{i=1}^{n_d} A_i \, p^i (1-p)^{n_d-i} \qquad (C.8)$$

where A_i denotes the number of codewords of weight i in the block code used.

Now consider the other case, where error-correction is carried out on the received blocks, which are transmitted through the same channel described above. Let the error-correction code used be an (n_c,k) code, where the number of parity-check digits is n_c-k. The total increase in the number of bits per block due to the parity-check digits, which is the only source of increasing the channel bit rate in this case, is

$$\delta_C = n_c - k \qquad (C.9)$$

Two sources of errors are present in this case. Firstly, as in the error-detection case a corrupted codeword can be received as a different codeword, where the decoder will accept the erroneous data as a correct data. The probability of such occurance is

$$p_{ec} = \sum_{i=1}^{n_c} A_i \, p^i (1-p)^{n_c-i} \qquad (C.10)$$

where A_i represent the number of codewords of weight i in the block code used. Secondly, the received words which has erroneous digits more than the code correction ability may be decoded erroneously, which is called failure. The probability of decoding failure, whenever t the maximum number of errors the code guarantees to correct, is less than half the code's minimum distance[6,56]

is

$$p_{ef} = 1-\sum_{k=0}^{n_c}\sum_{j=0}^{t}\sum_{i=0}^{n_c} A_i W_{i,k}^{(i,n_c-i)} p^k (1-p)^{n_c-k} \qquad (C.11)$$

where A_i is the number of codewords of weight i in the block code used, and $W_{j,k}^{(i,n-i)}$ denote the number of words of weight k which are at distance j from any given word of weight i and length n. The actual probability of decoding failure is may be less than the probability calculated by Eq.(C.11), because Eq.(C.11) assumes a decoder that decodes all patterns of up to t errors and nothing more.

If it can be assumed that the error probabilities for the two systems described above are within the acceptable limits, then the increase in the transmission bit rate can be evaluated by calculating the expression that gives the same increase in the bit per block for the two systems, that is

$$\delta_D = \delta_C \qquad (C.12)$$

Let the error probability value that result equal increase be p_{lim}, then from Eqs.(C.9) and (C.7)

$$(n_d-k) + \left(1+\frac{n_r}{n_d}\right)\{1-(1-p_{lim})^{n_d}\} = n_c-k \qquad (C.13)$$

Clearly if the left hand side is smaller, the detection and retransmission system will introduce less increase in the transmission bit rate than the error-correction

system, and vice versa.

Once the error-detction and the error-correction codes are chosen, the deciding factor on which of the two codes will increase the transmission bit rate less, is the channel probability of generating errors p. The probability of error for the equal increase in the bit rate can be calculated from Eq.(C.13)

$$(1-p_{lim})^{n_d} = 1 - \frac{n_c - n_d}{(n_d+n_r)/n_d} \qquad (C.14)$$

$$1-p_{lim} = \left(1 - \frac{n_c - n_d}{(n_d+n_r)/n_d} \right)^{\frac{1}{n_d}} \qquad (C.15)$$

$$p_{lim} = 1 - \left(1 - \frac{n_c - n_d}{(n_d+n_r)/n_d} \right)^{\frac{1}{n_d}} \qquad (C.16)$$

Eq.(C.16) gives the limit for the use of either codes on the channel. When the probability of error in the transmission channel is less than p_{lim}, the use of error-detection and retransmission system will produce less increase in the transmission bit rate. While if the error probability is higher, then the use of error-correction system will result in lower transmission bit rate. In the case that the probability of error in the transmission channel is equal to p_{lim}, it may be beneficial to use the error-detection and retransmission

system, because although it will not give any reduction in the transmission bit rate over the other system, yet it may give some improvement in the error rate.

Consider the special case where the number of parity-check digits for both codes is the same i.e. the thransmitted block length is the same, then

$$n_c = n_d \tag{C.17}$$

susstituting Eq.(C.17) in Eq.(C.16) gives

$$P_{lim} = 0 \tag{C.19}$$

Thus, whenever the codes used are of equal length, the error-correction system will equal the increase in the transmission bit rate, when the channel is error-free with the error-detection system, and will increase the transmission bit rate less for any value of the channel error probability other than zero.

APPENDIX D

SIMULATION PROGRAMS

Sample programs are included in this appendix, where the burst (34,22) code is used for the burst channel samples, and the BCH(31,21) code is used for random channel samples. Programs included are written in Fortran, but some subroutines are written in Plan[43] to speed execution. The first and second soft-decision algorithms are not written in standard Fortran, because they contain logical operations MASK, OR, XOR, AND, SHIFT,..., hence these subroutines have to be modified or used with the appropriate computer, if they have to be used.

```
      MASTER ERORGN
C
C     THIS IS THE RANDOM ERROR CHANNEL SIMULATOR WHERE
C     THE BCH (31,21) CODE IS USED THE NUMBER OF TRANSMITTED
C     CODEWORDS IS 3000 .  THE CODEWORDS GENERATED ARE RANDOM.
C
      DIMENSION D(31),IT(31)
      DIMENSION IR(31,17)
      CALL G05CBF(0)
      SIG=0.447
      DO 7 L=1,3000
C
C     RANDOM DATA GENERATION AND ENCODING
C
      DO 3 I=11,31
      A=G05CAF(A)
      IT(I)=0
3     IF (A.GT.0.5) IT(I)=1
      CALL IENCODE(IT(1),MESS)
      DO 8 I=1,31
      D(I)=-1.0
8     IF (IT(I).GT.0) D(I)=1.0
C
C     THE ADDITION OF NOISE (SIGMA=SIG)
C
      DO 9 I=1,31
9     D(I)=G05DDF(0.0,SIG)+D(I)
C
C     THE QUANTISATION : THE QUANTISER IS A IS A LINEAR QUANTISER
C     THE QUANTISATION LEVELS ARE 2*LEV LEVELS
C
      LEV=8
      DO 1 I=1,31
      Q=LEV*(D(I)+1)
      IF (Q.LT.0) GOTO1
      K=Q+1
      IF (K.GT.(2*LEV)+1) K=(2*LEV)+1
      DO 2 J=1,K
2     IR(I,J)=1
1     CONTINUE
      IF (IPARALLEL(IR(1,1)).NE.MESS) IERR=IERR+1
7     CONTINUE
      WRITE (2,100) IERR
      STOP
100   FORMAT (1H ,'NO. OF ERROR WORD RECEIVED IS ',I10)
      END
      FINISH
****
```

```
      MASTER MAIN
C
C     THE MAIN PROGRAM IS A BURST-CHANNEL SIMULATOR
C     THE CODE USED IN THIS PROGRAM IS THE BURST (34,22)
C     WHERE SIG AND SIG1 ARE THE NOISE SIGMA FOR
C     THE BACKGROUND AND THE BURST NOISE RESPECTIVELY
C
      DIMENSION IX(231),X(231),D(35,25),IT(35)
      DIMENSION NSCAN(2,25)
      N=231
      KODE=34
      IX(N)=1
      X(N)=93650000.0
      LAMDA=25
      SIG=0.355
      SIG1=1.0
      CALL G05CBF(LAMDA)
C     GET CHANNEL DATA
      CALL START (IX,X)
      NSC=1
      NSCAN(1,1)=1
      NSCAN(2,1)=LAMDA
C
C     THE DATA GENERATION AND THE ENCODER
C
      DO 302 I=1,25
      DO 301 J=13,34
      A=G05CAF(A)
      IT(J)=0
      IF (A.GE.0.5) IT(J)=1
301   CONTINUE
      CALL IENCODE(IT(1),IT(35))
      DO 303 J=1,34
      D(J,I)=(2*IT(J))-1
303   CONTINUE
      D(35,I)=IT(35)
302   CONTINUE
C
C     BACKGROUNG NOISE ADDITION
C
      DO 3 I=1,LAMDA
      DO 5 J=1,34
      D(J,I)=G05DDF(0.0,SIG)+D(J,I)
5     CONTINUE
3     CONTINUE
      LL=1
      LBLO=LAMDA*KODE
      CALL BURST(IX,X,ISTAR1,LL,LAMDA,LBLO,NBLO1,IBL1,IST1)
2     LL=LL+1
      IF (MORE.EQ.2) GOTO 28
      IRST1=IX(LL-1)
      IRST=IX(LL-1)
      ISTAR=ISTAR1
      IBL=IBL1
```

```
      IST=IST1
      GOTO 30
28    IF (LONGMO.EQ.0) IRST2=IX(LL-1)
      LONGMO=0
      IRST=ISTAR1-ISTAR-IRST1+IRST2
      IST=IMST
      ISTAR=ISTAR+IRST1
30    NBLO=NBLO1
      MORE=0
      CALL BURST(IX,X,ISTAR1,LL,LAMDA,LBLO,NBLO1,IBL1,IST1)
105   IF (ISTAR+IRST.GT.LBLO) LONGER=1
100   IF (NBLO.LT.NBLO1) GOTO 102
      IF (LONGER.EQ.0) GOTO 106
      IF (ISTAR1+IX(LL).GT.LBLO) GOTO 101
      LL=LL+1
      GOTO 106
101   IRST2=IX(LL)+ISTAR1-LBLO-1
      ISTAR1=1
      IBL1=1
      IST1=1
      NBLO1=NBLO1+1
      LONGMO=1
      GOTO 102
106   IF (ISTAR+IRST-ISTAR1) 103,103,104
104   MORE=1
103   MORE=MORE+1
102   IF (IRST.LE.0) GOTO 32
      LIMIT=LAMDA
      IF (IST+IRST-1.LT.LAMDA) LIMIT=IST+IRST-1
      IF (IRST.LT.LAMDA) GOTO 107
      LSCAN=1
      GOTO 108
107   NSC=NSC+1
      NSCAN(1,NSC)=IST
      NSCAN(2,NSC)=LIMIT
C
C      BURST NOISE ADDITION
C
108   DO 1 I=IST,LIMIT
      IRST=IRST-1
1     D(IBL,I)=G05DDF(0.0,SIG1)+D(IBL,I)
      LNS=LNS+LIMIT
      IMST=LIMIT+1
      IF (IRST.LE.0) GOTO 32
      IF (IBL.EQ.KODE) GOTO 32
8     IF (IRST.LT.LAMDA) GOTO 14
      IBL=IBL+1
      DO 7 I=1,LAMDA
      IRST=IRST-1
7     D(IBL,I)=G05DDF(0.0,SIG1)+D(IBL,I)
      LNS=LNS+LAMDA
      IF (IBL+LONGER.EQ.KODE+1) GOTO 24
      GOTO 8
14    IF (IRST.LE.0) GOTO 32
```

```
      IBL=IBL+1
      NSC=NSC+1
      NSCAN(1,NSC)=1
      NSCAN(2,NSC)=IRST
      DO 20 I=1,IRST
20    D(IBL,I)=G05DDF(0.0,SIG1)+D(IBL,I)
      LNS=LNS+IRST
      IMST=IRST+1
32    IF (MORE.NE.0) GOTO 25
24    NSC1=2
      CALL LIMITS(NSCAN,NSC,LSCAN)
      IF (LSCAN.EQ.1) NSC1,NSC=1
C
C     THE DECODER
C
      DO 109 LO=NSC1,NSC
      IBE=NSCAN(1,LO)
      IEN=NSCAN(2,LO)
      DO 4 L=IBE,IEN
      IF (ICOR(D,L,SIG).NE.0) MERR=MERR+1
      NWORD=NWORD+1
    4 CONTINUE
109   CONTINUE
      NSC=1
      LSCAN=0
      IRST1=IX(LL-1)
      IF (LONGER.EQ.0) GOTO 25
      IRST1=IRST
      ISTAR=1
      IBL=1
      IST=1
      LONGER=0
      MORE=0
      NBLO=NBLO+1
      GOTO 105
25    IF (LL.LT.N) GOTO 2
31    WRITE (2,17) MERR
      WRITE (2,35) NWORD
      WRITE (2,122) LAMDA
C
C     SNR(DB) CALCULATION
C
      ABC=10*ALOG10(1/(SIG*SIG))
      WRITE (2,124) ABC
      ABC=10*ALOG10(1/(SIG1*SIG1))
      WRITE (2,125) ABC
      ABC=((SIG*(93600000.0-LNS))+(LNS*SIG1))/93600000.0
      ABC=10*ALOG10(1/(ABC*ABC))
      WRITE (2,126) ABC
123   FORMAT (1H1)
124   FORMAT (1H ,'BACKGROUND S/N RATIO IS ',F10.2)
125   FORMAT (1H ,'BURST S/N RATIO IS ',F10.2)
126   FORMAT (1H ,'OVERALL S/N RATIO IS ',F10.2)
17    FORMAT (' NUMBER OF DECODING FAILURE IS',I10)
```

```
35    FORMAT (' NO. OF ERROR WORD RECEIVED ',I10)
122   FORMAT (' LAMDA =',I10)
      MERR=0
      NWORD=0
      LNS=0
121   CONTINUE
      STOP
      END
```

```
      SUBROUTINE BURST(IX,X,ISTAR1,LL,LAMDA,LBLO,NBLO1,IBL1,IST1)
C
C     THIS SUBROUTINE CALCULATES THE START AND END POINTS OF EACH
C     BURST IN THE INTERLACED BLOCK AND WETHER IT IS CONFINED TO
C     THIS SAME BLOCK OR NOT
C
      DIMENSION IX(231),X(231)
      B=X(LL)
      B=B/LBLO
      NBLO1=B
      I=B
      A=(B-I)*LBLO
      A=A/LAMDA
      IBL1=A
      I=A
      A=(A-I)*LAMDA
      IST1=A+1
      ISTAR1=IST1+(IBL1*LAMDA)
      IBL1=IBL1+1
      RETURN
      END
```

```
      SUBROUTINE LIMITS(NSCAN,NSC,LSCAN)
C
C     THIS SUBROUTINE CALCULATES THE LIMITS OF THE ERRONEOUS
C     RECIVED WORDS FOR CORRECTION
C
      DIMENSION K(2,24),NSCAN(2,25)
      IF (LSCAN.EQ.1) GOTO 151
154   IF (NSC.LE.2) GOTO 151
      KFLA=0
      K(1,1)=NSCAN(1,2)
      K(2,1)=NSCAN(2,2)
      N=1
      DO 150 I=3,NSC
      DO 153 J=1,N
      IF (NSCAN(1,I).GT.K(2,J)) GOTO 153
      IF (NSCAN(2,I).LT.K(1,J)) GOTO 153
      IF (NSCAN(1,I).LT.K(1,J)) K(1,J)=NSCAN(1,I)
      IF (NSCAN(2,I).GT.K(2,J)) K(2,J)=NSCAN(2,I)
      GOTO 150
153   CONTINUE
      N=N+1
      K(1,N)=NSCAN(1,I)
      K(2,N)=NSCAN(2,I)
      KFLA=1
150   CONTINUE
      DO 152 I=1,24
      NSCAN(1,I+1)=K(1,I)
      NSCAN(2,I+1)=K(2,I)
152   CONTINUE
      IF (NSC.EQ.N+1) GOTO 151
      NSC=N+1
      IF (KFLA.EQ.1) GOTO 154
151   RETURN
      END
```

```
      SUBROUTINE START(IX,X)
C
C     THIS SUBROUTINE CALCULATES THE START POINTS OF THE
C     BURSTS AND THEIR LENGTHS
C
      DIMENSION IX(231),X(231)
      N=230
      DO 75 I=1,N
78    A=G05CAF(A)
      IF (A.LT.0.0003) GOTO 78
      B=(2* A  -1.0003)/0.9997
      B=ACOS(B)
      C=1.8084-1.8609*COS (B)+1.0185*COS (2*B)-0.97088*COS(3*B)
75    IX(I)=IFIX (64*C)
      DO 76 I=1,N
      A=G05DAF(1.0,93600000.0)
76    X(I)=A
      K=0
      CALL M01ANF( X,1,N,K)
      IF (K.NE.0) WRITE (2,77) K
77    FORMAT (' SORT FAIL K=',I3)
      A=IX(N)+X(N)
      IF (A.GT.93600000.0) IX(N)=IX(N)+93600000.0-A
      RETURN
      END
```

THE ENCODER

THIS SUBROUTINE CALCULATES PARITY-CHECK DIGITS FROM THE
THE INFORMATION DIGITS FOR THE BURST CODE (34,22)

```
#PROGRAM        /IENCODE(DBM,22AM)
#LOWER
                ADDRESS(2)
#PROGRAM
      OBEY      0(1)
      LDN   3   0(3)
      STO   3   ADDRESS
      OBEY      1(1)
      LDN   3   0(3)
      STO   3   ADDRESS+1
      LDX   3   ADDRESS
      ADN   3   12
      LDN   7   22
      LDN   5   0
L2    SLL   5   1
      ORX   5   0(3)
      ADN   3   1
      BCT   7   L2
      LDX   3   ADDRESS+1
      STO   5   0(3)
      LDN   7   22
L3    STO   5   3
      ANDX  3   '#1'
      BZE   3   L1
      ERX   5   '#15712'
L1    SRL   5   1
      BCT   7   L3
      LDN   7   12
      LDX   3   ADDRESS
      ADN   3   11
L4    STO   5   6
      ANDX  6   '#1'
      STO   6   0(3)
      SBN   3   1
      SRL   5   1
      BCT   7   L4
      EXIT  1   2
#END
```

THE DECODER

THIS SUBROUTINE DECODES THE OUTPUT OF THE 17 THRESHOLDS
TO FIND THE ERROR-PATTERN THAT IS MOST LIKELY ADDED TO
THE TRANSMITTED CODEWORD FOR THE BURST CODE (34,22)

```
#PROGRAM        /IPARALLEL(DBM,22AM)
#LOWER
                CHECK,MESSAGE
                RETURN,ADDRESS(18)
#PROGRAM
        OBEY    0(1)
        STO  1  RETURN
        LDN  3  0(3)
        LDN  1  17
        STO  3  ADDRESS+1
        SBN  1  1
        STO  3  2
L17     SBN  2  34
        STO  2  ADDRESS(1)
        SBN  1  1
        ADN  3  34
        STO  3  ADDRESS(1)
        BCT  1  L17
        LDN  1  17
L14     LDX  3  ADDRESS(1)
        STO  3  ADDRESS
        LDN  7  12
        LDN  5  0
L3      SLL  5  1
        ORX  5  0(3)
        STOZ    0(3)
        ADN  3  1
        BCT  7  L3
        STO  5  CHECK
        LDN  7  22
        LDN  5  0
L4      SLL  5  1
        ORX  5  0(3)
        STOZ    0(3)
        ADN  3  1
        BCT  7  L4
        STO  3  ADDRESS
        STO  5  MESSAGE
        STO  5  6
#MONITOR        0/110
        LDN  7  10
        ANDX 6  '#7777'
        CALL 2  DIVIDE
        LDN  7  12
        LDX  5  CHECK
        SLL  5  12
#MONITOR        0/115
        CALL 2  DIVIDE
```

```
       BZE    6  COMP
       CALL   2  FAST
#MONITOR         0/103
       LDN    7  22
L11    STO    6  3
       ANDX   3  '#1'
       BZE    3  L8
       CALL   2  BRSLEN
#MONITOR         0/104
       BZE    4  L13
L8     CALL   2  SHIFT
       BCT    7  L11
       LDN    7  12
CHDG   STO    6  3
       ANDX   3  '#1'
       BNZ    3  L12
       SRL    6  1
       BCT    7  CHDG
L12    CALL   2  BRSLEN
       BZE    4  COMP
       BCT    1  L14
       LDX    6  MESSAGE
       BRN       L18
L13    NGX    3  7
       BZE    3  COMP
       SLL    6  22(3)
       LDX    5  MESSAGE
       ERX    6  5
       ANDX   6  '#17777777'
       BRN       L5
COMP   LDX    6  MESSAGE
L5     SBN    1  1
       BXL    1  '#1',L18
L 17   LDN    7  34
       LDX    3  ADDRESS(1)
L15    STOZ      0(3)
       ADN    3  1
       BCT    7  L15
       BCT    1  L17
L18    LDX    1  RETURN
#MONITOR         0/106
       EXIT   1  17
#CUE             DIVIDE
SYND   SRL    5  1
       STO    5  4
       ANDX   4  '#4000'
       STO    6  3
       ANDX   3  '#1'
       SRL    6  1
       ORX    6  4
       BZE    3  L1
       ERX    6  '#6745'
L1     BCT    7  SYND
       EXIT   2  0
```

```
#CUE                SHIFT
        STO    6    3
        ANDX   3    '#1'
        SRL    6    1
        BZE    3    L2
        ERX    6    '#6745'
L2      EXIT   2    0
#CUE                FAST
        LDN    7    69
L7      STO    6    3
        ANDX   3    '#1'
        SRL    6    1
        BZE    3    L6
        ERX    6    '#6745'
L6      BCT    7    L7
        EXIT   2    0
#CUE                BRSLEN
        STO    6    4
        ANDX   4    '#7700'
        EXIT   2    0
#END
```

```
      THIS SUBROUTINE IS THE PARALLEL THRESHOLD DECODER
      FOR THE BCH (31,21) CODE WHERE THE INPUT IS THE
      RECEIVED SIGNAL DETECTED AT 17 THRESHOLDS

PLAN(CR)
#PROGRAM        /IPARALLEL(DBM,22AM)
#LOWER
                CHECK,MESSAGE
                RETURN,ADDRESS(18)
#PROGRAM
      OBEY      0(1)
      STO   1   RETURN
      LDN   3   0(3)
      LDN   1   17
      STO   3   ADDRESS+1
      SBN   1   1
      STO   3   2
L17   SBN   2   31
      STO   2   ADDRESS(1)
      SBN   1   1
      ADN   3   31
      STO   3   ADDRESS(1)
      BCT   1   L17
      LDN   1   17
L14   LDX   3   ADDRESS(1)
      STO   3   ADDRESS
      LDN   7   10
      LDN   5   0
L3    SLL   5   1
      ORX   5   0(3)
      STOZ      0(3)
      ADN   3   1
      BCT   7   L3
      STO   5   CHECK
      LDN   7   21
      LDN   5   0
L4    SLL   5   1
      ORX   5   0(3)
      STOZ      0(3)
      ADN   3   1
      BCT   7   L4
      STO   3   ADDRESS
      STO   5   MESSAGE
      STO   5   6
#MONITOR        0/110
      LDN   7   11
      ANDX  6   '#1777'
      CALL  2   DIVIDE
      LDN   7   10
      LDX   5   CHECK
      SLL   5   10
#MONITOR        0/115
      CALL  2   DIVIDE
      BZE   6   COMP
```

```
L16    CALL   2  HAMWEI
#MONITOR         0/103
       BXL    4  '#3',COMP
       CALL   2  SHIFT
       LDN    7  10
L6     STO    6  3
       ANDX   3  '#1'
       BZE    3  L7
       CALL   2  HAMWEI
#MONITOR         0/111
       BXL    4  '#3',L12
L7     CALL   2  SHIFT
       BCT    7  L6
       LDN    7  21
L11    STO    6  3
       ANDX   3  '#1'
       BZE    3  L8
       CALL   2  HAMWEI
#MONITOR         0/104
       BXL    4  '#3',L13
L8     CALL   2  SHIFT
       BCT    7  L11
       BCT    1  L14
       LDX    6  MESSAGE
       BRN       L18
L12    LDN    4  1
       NGX    3  4
       ADX    3  7
       SRL    6  0(3)
#MONITOR         0/101
       LDX    5  MESSAGE
       ANDX   6  '#7777777'
       ERX    6  5
       BRN       L5
L13    NGX    3  7
       BZE    3  COMP
       SLL    6  22(3)
       LDX    5  MESSAGE
       ERX    6  5
       ANDX   6  '#7777777'
       BRN       L5
COMP   LDX    6  MESSAGE
L5     SBN    1  1
       BXL    1  '#1',L18
L17    LDN    7  31
       LDX    3  ADDRESS(1)
L15    STOZ      0(3)
       ADN    3  1
       BCT    7  L15
       BCT    1  L17
L18    LDX    1  RETURN
#MONITOR         0/106
       EXIT   1  17
#CUE             DIVIDE
```

```
SYND   SRL    5   1
       STO    5   4
       ANDX   4   '#1000'
       STO    6   3
       ANDX   3   '#1'
       SRL    6   1
       ORX    6   4
       BZE    3   L1
       ERX    6   '#1133'
L1     BCT    7   SYND
       EXIT   2   0
#CUE               SHIFT
       STO    6   3
       ANDX   3   '#1'
       SRL    6   1
       BZE    3   L2
       ERX    6   '#1133'
L2     EXIT   2   0
#CUE               HAMWEI
       STOZ       4
       LDN    5   10
L10    LDN    3   1
       ANDX   3   6
       BZE    3   L9
       ADN    4   1
L9     SRC    6   1
       BCT    5   L10
       SLC    6   10
       EXIT   2   0
```

```
      SUBROUTINE ALG(ICOR,D,L,SIG,LEV)
C
C     THIS SUBROUTINE IS THE FIRST ALGORITHM SOFT-DECISION
C     DECODER WHERE
C           SIG IS THE NOISE SIGMA
C           LEV IS 1/2 THE QUATIZER LEVELS
C           D THE RECEIVED WORD
C           L THE LOCATION OF THE RECEIVED WORD IN THE MATIX
C           ICOR THE DECODER OUTPUT 0 IF CORRECT DECODED CORRECTLY
C
      DIMENSION D(34,25)
      DIMENSION K(34)
      DIMENSION KW(34)
      ICOR=0
C
C     THIS IS A LINEAR QUANTISER OF 2*LEV LEVELS
C
      DO 12 I=1,34
      K(I)=0
12    IF (D(I,L).GT.0) K(I)=1
      DO 122 I=1,34
      KW(I)=(LEV*ABS(D(I,L)))+1
      IF (KW(I).GT.LEV) KW(I)=LEV
      D(I,L)=G05DDF(0.0,SIG)-1.0
122   CONTINUE
      ICON=4096
      IG=7114
      IR=0
      IS=0
      DO 1 I=1,22
      IR=SHIFT(IR,1)
1     IR=OR(IR,K(I))
      DO 2 I=23,34
      IR=SHIFT(IR,1)
      IR=OR(IR,K(I))
      IS=SHIFT(IS,1)
2     IS=OR(IS,K(I))
      DO 3 I=1,22
      J=23-I
      IF (K(J).EQ.1) IS=OR(IS,ICON)
      IF ((IS.A.1B).EQ.1) IS=XOR(IS,IG)
3     IS=SHIFT(IS,-1)
C
C     THE PARITY-CHECK ERROR-TRAPPING
C
      DO 4 I=1,7
      IF ((IS.A.7700B).NE.0) GOTO 8
      IF ((IR.A.17777777B).NE.0) ICOR=1
      GOTO 114
8     IF ((IS.A.1B).EQ.1) IS=XOR(IS,IG)
4     IS=SHIFT(IS,-1)
      DO 5 I=1,62
      IF ((IS.A.1B).EQ.1) IS=XOR(IS,IG)
5     IS=SHIFT(IS,-1)
```

```
      IWEIT=10000
C
C     EPSW CALCULATION
C
      DO 6 I=1,22
      IF ((IS.A.1B).EQ.0) GOTO 7
      J=35-I
      IW=0
      IW=IW+KW(J)
      IF ((IS.A.0002B).NE.0) IW=IW+KW(J-1)
      IF ((IS.A.0004B).NE.0) IW=IW+KW(J-2)
      IF ((IS.A.0010B).NE.0) IW=IW+KW(J-3)
      IF ((IS.A.0020B).NE.0) IW=IW+KW(J-4)
      IF ((IS.A.0040B).NE.0) IW=IW+KW(J-5)
      IF ((IS.A.0100B).NE.0) IW=IW+KW(J-6)
      IF ((IS.A.0200B).NE.0) IW=IW+KW(J-7)
      IF ((IS.A.0400B).NE.0) IW=IW+KW(J-8)
      IF ((IS.A.1000B).NE.0) IW=IW+KW(J-9)
      IF ((IS.A.2000B).NE.0) IW=IW+KW(J-10)
      IF ((IS.A.4000B).NE.0) IW=IW+KW(J-11)
      IF (IW.GE.IWEIT) GOTO 9
      ISHORT=IS
      IWEIT=IW
      ISH=I
9     IS=XOR(IS,IG)
7     IS=SHIFT(IS,-1)
6     CONTINUE
      IW=0
      I=23
      J=35-I
      IF ((IS.A.0001B).NE.0) IW=IW+KW(J)
      IF ((IS.A.0002B).NE.0) IW=IW+KW(J-1)
      IF ((IS.A.0004B).NE.0) IW=IW+KW(J-2)
      IF ((IS.A.0010B).NE.0) IW=IW+KW(J-3)
      IF ((IS.A.0020B).NE.0) IW=IW+KW(J-4)
      IF ((IS.A.0040B).NE.0) IW=IW+KW(J-5)
      IF ((IS.A.0100B).NE.0) IW=IW+KW(J-6)
      IF ((IS.A.2000B).NE.0) IW=IW+KW(J-7)
      IF ((IS.A.0400B).NE.0) IW=IW+KW(J-8)
      IF ((IS.A.1000B).NE.0) IW=IW+KW(J-9)
      IF ((IS.A.2000B).NE.0) IW=IW+KW(J-10)
      IF ((IS.A.4000B).NE.0) IW=IW+KW(J-11)
      IF (IW.GE.IWEIT) GOTO 10
      IF ((IR.A.17777777B).NE.0) ICOR=1
      RETURN
      IS=XOR(IS,IG)
      IS=SHIFT(IS,-1)
      ISH=ISH-1
      IS=SHIFT(ISHORT,ISH)
      IR=XOR(IR,IS)
      IF ((IR.A.17777777B).NE.0) ICOR=1
114   DO 113 I=1,22
      IF ((IR.A.1B).NE.0) NBIT=NBIT+1
      IR=SHIFT(IR,-1)
```

```
113   CONTINUE
      RETURN
      END
```

```
      SUBROUTINE ALG(ICOR,D,L,SIG,LEV,LLV)
C
C     THIS SUBROUTINE IS THE SIMULATION OF THE SECOND
C     SOFT-DECISION ALGORITHM DECODER . WHERE
C        D    IS THE RECEIVED WORD
C        L    ITS LOCATION IN THE MATRIX
C        SIG  THE BACKGROUND NOISE SIGMA
C        LEV  1/2 THE NO. OF QUANTISATION LEVELS
C        LLV  THE NO. OF ERROR-PATTERNS TESTED
C        ICOR THE DECODER OUTPUT, 0 IF THE DECODING IS CORRECT
C
      DIMENSION D(34,25)
      DIMENSION K(34)
      DIMENSION KW(34)
      DIMENSION ISTA(2,34)
      DIMENSION LAB(34),LABEL(34),LWO(5),LOW(5)
      ICOR=0
C
C     THE QUAMTISER IS A UNIFORM QUANTISER OF 2*LEV LEVELS
C
      DO 12 I=1,34
      K(I)=0
12    IF (D(I,L).GT.0) K(I)=1
      DO 122 I=1,34
      KW(I)=(LEV*ABS(D(I,L)))+1
      IF (KW(I).GT.LEV) KW(I)=LEV
      D(I,L)=G05DDF(0.0,SIG)-1.0
122   CONTINUE
      ICON=4096
      IG=7114
      IR=0
      IS=0
C
C     SYNDROME CALCULATION
C
      DO 1 I=1,22
      IR=SHIFT(IR,1)
1     IR=OR(IR,K(I))
      DO 2 I=23,34
      IR=SHIFT(IR,1)
      IR=OR(IR,K(I))
      IS=SHIFT(IS,1)
2     IS=OR(IS,K(I))
      DO 3 I=1,22
      IF (K(J).EQ.1) IS=OR(IS,ICON)
      IF ((IS.A.1B).EQ.1) IS=XOR(IS,IG)
3     IS=SHIFT(IS,-1)
C
C     THE PARITY-CHECK ERRRE-TRAPPER
C
      DO 4 I=1,7
      IF ((IS.A.7700B).NE.0) GOTO 8
      IF ((IR.A.17777777B).NE.0) ICOR=1
      GOTO 114
```

```
8     IF ((IS.A.1B).EQ.1) IS=XOR(IS,IG)
4     IS=SHIFT(IS,-1)
      DO 5 I=1,62
      IF ((IS.A.1B).EQ.1) IS=XOR(IS,IG)
5     IS=SHIFT(IS,-1)
      IWEIT=10000
      IND=0
C
C     THE DECODING PROCESS
C
      DO 6 I=1,22
      IF ((IS.A.1B).EQ.0) GOTO 7
      IND=IND+1
      II=IS
      DO 13 J=1,12
      II=SHIFT(II,-1)
      IF ((II.A.7777B).EQ.0) GOTO 14
13    CONTINUE
14    LAB(IND)=J
      ISTA(2,IND)=I
      ISTA(1,IND)=IS
9     IS=XOR(IS,IG)
7     IS=SHIFT(IS,-1)
6     CONTINUE
C
C     SORTING THE ERROR-PATTERNS ACCORDING TO THEIR LENGTH
C
      LKING=5
      IFAIL=0
      CALL MO1ALF(LAB,LWO,LABEL,LOW,IND,LKING,IFAIL)
      IF (IFAIL.NE.0) WRITE (2,99)
C
C     EPSW CALCULATION FOR THE TESTED ERROR-PATTERNS
C
      DO 15 I=1,LLV
      IS=ISTA(1,LABEL(I))
      J=35-ISTA(2,LABEL(I))
      IW=0
      IW=KW(J)
      IF ((IS.A.0002B).NE.0) IW=IW+KW(J-1)
      IF ((IS.A.0004B).NE.0) IW=IW+KW(J-2)
      IF ((IS.A.0010B).NE.0) IW=IW+KW(J-3)
      IF ((IS.A.0020B).NE.0) IW=IW+KW(J-4)
      IF ((IS.A.0040B).NE.0) IW=IW+KW(J-5)
      IF ((IS.A.0100B).NE.0) IW=IW+KW(J-6)
      IF ((IS.A.0200B).NE.0) IW=IW+KW(J-7)
      IF ((IS.A.0400B).NE.0) IW=IW+KW(J-8)
      IF ((IS.A.1000B).NE.0) IW=IW+KW(J-9)
      IF ((IS.A.2000B).NE.0) IW=IW+KW(J-10)
      IF ((IS.A.4000B).NE.0) IW=IW+KW(J-11)
      IF (IW.GE.IWEIT) GOTO 15
      ISHORT=IS
      IWEIT=IW
      ISH=ISTA(2,LABEL(I))
```

```
15    CONTINUE
C
C     THE CORRECTION PROCESS
C
      ISH=ISH-1
      IS=SHIFT(ISHORT,ISH)
      IR=XOR(IR,IS)
      IF ((IR.A.17777777B).NE.0) ICOR=1
114   DO 113 I=1,22
      IF ((IR.A.1B).NE.0) NBIT=NBIT+1
      IR=SHIFT(IR,-1)
113   CONTINUE
      RETURN
99    FORMAT (1H ,'SORT ERROR ')
      END
```

REFERENCES

1. Abramson,N.M., *"A Class of Systematic Codes for Non-Independent Errors"*, IRE Trans. on Information Theory, IT-5, pp.150-157, December, 1959.
2. Abramson,N.M., and B.Elpas, *"Double-Error-Correcting Coders and Decoders for Non-Independent Binary Errors"*, presented at the UNESCO Information Processing Conference in Paris, France, 1959.
3. Abramson,N.M., *"Error Correcting Codes from Linear Sequential Networks"*, Proc. 4th London Symposium on Information Theory, C.Cherry,Ed., Butterworths, Washington, D.C., 1961.
4. Apostol,T.M., *"Calculus"*, Blaisdell, Vol.1 1967, Vol.2 1969.
5. Balser,M., and Silverman,R.A., *"Coding for constant-data-rate systems—Part II. Multiple-error-correcting codes"*, Proc. IRE, vol.43, June 1955, pp. 728-733.
6. Berlekamp,E.R., *"Nonbinary BCH Decoding"*, presented at the IEEE international Symposium on Information Theory, San Remo, Italy, 1967.
7. Berlekamp,E.R., *"Algebraic Coding Theory"*, McGraw-Hill, New York, 1968.
8. Bloom,F.J. et al: *"Improvement of binary transmission by null-zone reception"*, Proc. IRE, vol.45, p 963, 1957.
9. Brayer,K. and Cardinale,D., *"Evalulation of error correction block encoding for high speed HF data"*, ibid., 1967, COM-15, pp. 370-382.

10. Cahn,C.R.; *"Binary Decoding Extended to Nonbinary Demodulation of Phase Shift Keying"*, IEE Trans, Vol COM-17, NO 5 (Oct), p583, 1969.

11. Cain,J.B., and Boyd,R.W., *"Convolutional code performance with PSK signaling in nonstationary Gaussian noise"*, NTC '78 Conference Record, pp.2.5.1-2.5.6. December 1978.

12. Chase,D. and Harper,R.C., *"An investigation of coding and diversity techniques for a selective fading channel"*, presented at the IEEE Int. Conf. Communication, Boulder, Colo., June 1969.

13. Chase,D., *"A class of algorithms for decoding block codes with channel measurement information"*,IEEE Trans, Vol IT-18, No 1 (Jan.), p170, 1972.

14. Chase,D., *"A combined coding and modulation approach for communication over dispersive channels"*, IEEE Trans, Vol COM-21, No 3 (March), pp 159-174, 1973.

15. Chien,R.T.,*"Cyclic decoding procedures for the Bose-Chaudhuri-Hocquenghem codes"*, IEEE Trans, Inf. Theory. IT-10. 357-363, October 1964.

16. Clark,A.P.,*"Principles of Digital Data Transmission"*, Pentech Press, London, 1976.

17. Clark,A.P., *"Advanced Data-Transmission Systems"*, Pentech Press, London, 1976.

18. Clark,G.C.Jr. & Cain,J.Bibb., *"Error Correcting Codes for Digital Communications"*, Plenum, N.Y.,1981.

19. Dorsch,W.R., *"Maximum likelihood decoding of binary group codes for the Gaussian channel"*, presented at IEEE, Int.Symp. Information Theory, Noordwijk, the Netherlands, June 1970.

20. Einarsson,G. & Sundberg,C.E., *"A note on soft-decision with successive erasure"*, IEEE Trans, Vol IT-22, No. 1 (Jan.), p88, 1976.

21. Elspas,B, and R.A.Short., *"A Note on Optimum Burst-Error-Correcting Codes"*, IRE Trans. on Information Theory, IT-8, pp. 39-42, January, 1962.

22. Farrell,P.G.& Munday,E., *"Economical practical realisation of minimum-distance soft-decision decoding for data transmission"*, Proc. Zurich Int. Seminar on Digital Communications, March 1976, pp. 135.1-6.

23. Farrell,P.G.,Munday,E., and Kalligeros,N., *"Digital communications using soft-decision detection techniques"*, AGARD symposium on digital Communication in avionics, Munich, June 1978.

24. Farrell,P.G., and Munday,E.,*"Soft-decision detection of HF signals"*. Conference on recent advances in HF communication systems and techniques, IEE, London, Feb. 1979.

25. Farrell,P.G., and Goodman,R.M.F., *"Soft-Decision Error Control for HF Data Transmission"*, IEE Proc., Vol. 127, pp. 389-400, October 1980.

26. Fire,P., *"A Class of Multiple-Error-Correcting Binary Codes for Non-Independent Errors"*, Sylvania Report RSL-E-2. Sylvania Electronic Defense Laboratory, Reconnaissance Systmes Division, Mountain View, California, March, 1959.

27. Forney,G.,*"Generalised minimum distance decoding"*, IEEE Trans, Vol IT-12, No. 2 (April, 1966), pp 125-131.

28. Forney,G.D., *"Burst-correcting codes for the classic bursty channel"*, IEEE Trans. Commun. Technol., COM-19, 772-781, October 1971.

29. Foulk,C.R.,*"Some Properties of Maximally-Efficient Cyclic Burst-Correcting Codes and Results of a Computer Search for Such Codes"*, File No. 375, Digital Computer Lab., University of Illinois, Urbana, Illinois, June 12, 1961.

30. Froberg,C., *"Introduction to Numerical Analysis"*, Addison-Welsley, 1974.

31. Gallager,R.G., *"Information Theory and Reliable Communication"*, John Wiley, New York, 1968.

32. Gilbert,E.N., *"Capacity of a Burst-Noise Channel"*, Bell Sys. Tech. Journal, 39, pp. 1253-1265, Sept. 1960.

33. Goodman,R.M.F.& Green,A.D.,*"Microprocessor controlled soft-decision decoding of error-correcting block codes"*, Proc. IERE Conf. on Digital Processing of Signals in Communications, No.337, pp 337-349, Loughborough, England, 1977.

34. Goodman,R.M.F., and Green,A.D., *"Microprocessor-controlled permutation decoding of block error-correcting codes"*, IERE Conference proceedings 41, Sept. 1978.

35. Goodman,R.M.F., Green,A.D., and Winfield,A.F.T., *"Soft-decision error-correction coding schemes for HF data transmission"*, Conference on recent advances in HF comunication systems and techniques, IEE, London, Feb. 1979.

36. Gross,A.J., *"Binary Group Codes which Correct in Bursts of Three or Less for Odd Redundancy"*, IRE Trans. on Information Theory, IT-8, pp 356-359, October, 1962.

37. Gross,A.J.,*"A Note on Some Binary Group Codes which Correct Errors in Bursts of Four of Less "*, IRE Trans. on Information Theory, IT-8,p 384, October, 1962.

38. Green,J.H.,Jr., and R.L.San Soucie.,*"An Error-Correcting Encoder and Decoder of High Efficiency"*, Proc. IRE, 46, pp 1741-1744, 1958.

39. Hackett,C.M.,Jr., *"Word error rate for group codes detected by correlation and other means"*, IEEE Trans. Inform. Theory, Vol. IT-9, Jan. 1963, pp 24-33.

40. Harrison,C.N.,*"Application of soft decision techniques to block codes"*, Proc. IERE Conf. on Digital Processing of Signals in Communications, Loughborough, England, No. 37, pp 331-336, 1977.

41. Heller,R.M.,*"Forced-erasure decoding and the erasure reconstruction spectra of group codes"*, IEEE Trans. Vol.COM-15, No. 3 (June), p 390, 1967.

42. Hobbs,C.F., *"Universality of blank-correction and error detection"*, IEEE Trans, Vol. IT-13, No.2 (April), p 342, 1967.

43. ICL,Software Distribution Department, *"Plan Reference Manual, 1900 Series"*, Second Edition, July 1972.

44. Jacobs,L.M., *"Sequential decoding for efficient communication from deep Space"*, ibid., 1967, COM-15, pp 492-501

45. Jayant,N.S., *"An erasure scheme for atmospheric noise burst interferences"*, Proc. IEEE, Vol. 54, No.12 (Dec), p1943, 1966.

46. Jelinek,F., *"A fast sequential algorithm using a stack"*, IBM Jour. Res. Dev., Vol. 13, Nov., pp 675-685, 1969.

47. Kasami,T., *"Cyclic Codes for Burst-Error-Correction"*, J. Inst. Elec. Commun. Eng. Japan, 45, p. 9-16, January, 1962.

48. Kasami,T., *"Optimum Shortened Cyclic Codes for Burst-Error-Correction"*, IEEE Trans. on Information Theory, IT-9, pp. 105-109, April 1963.

49. Kasami,T., *"A Decoding Procedure for Multiple-Error-Correcting Cyclic Codes"*, IEEE Trans. on Information Theory, IT?10, pp. 134-139, April 1964.

50. Kasami,T., and S.Matoba., *"Some Efficient Shortened Cyclic Codes for Burst-Error-Correction"*, IEEE Trans. on Information Theory, IT-10, pp. 252-253, July, 1964.

51. Kazakov,A.A., *"A method of improving the noise immunity of redudnant binary code reception"*, Telecoms. (trans. of Elec. & Radioteknika) Vol.22 No.3 (March), p. 51, 1968.

52. King,D.G., *"Design and Evaluation of a Practical Multi-Threshold Level Error Correction System"*, M.Sc. Proj., Loughborough University of Technology, September, 1982.

53. Lebow,I.L., et al., *"Application of Sequential Decoding"*, IEEE Trans., IT-9, pp. 124-126, 1963

54. Leung,K.S.,and Welch,L.R., *"Erasure Decoding in Burst-Error Channels"*, IEEE Trans., Vol. IT-27, pp. 160-167, March, 1981.

55. Lin,S., *"An Introduction to Error-Correcting Codes"*, Prentice-Hall, N.J., 1970.

56. MacWilliams,F.J., *"A Theorm on The Distribution of Weight in a Systematic Code"*, Bell System Tech. J., 42, pp. 79-94, 1963.

57. MacWilliams,F.J., *"Permutation Decoding of Systematic Codes"*, Bell Systems Tech. J. 43, Part 1, pp. 485-505, January, 1964.

58. Marquart,R.G., *"The performance of foreced erasure decoding"*, IEEE Trans., COM-15, No.2 (June), p.397, 1967.

59. Massey,J.L., *"Threshold Decoding"*, MIT Press, 1963.

60. Massey,J.L, *"Coding and modulation in digital communications"*, Proc. Zurich Int. Seminar on Digital Cimmunications, pp. E2(1)-(4), 1974.

61. Meggitt,J.E., *"Error Correcting Codes for Correcting Burst of Errors"*, IBM J. Research Develop., 4, pp. 329-334, July, 1960.

62. Meggitt,J.E., *"Error Correcting Codes and Their Implementation"*, IRE Trans. on Information Theory, IT?7, pp. 234-244, October, 1961.

63. Melas,C.M., *"A New Group of Codes for Correction of Dependent Errors in Data Transmission"*, IBM J. Research Develep., 4, pp. 58-64, January, 1960.

64. Mitchell,M.E., et al., *"Coding and Decoding Operations Research"*, G.E. Advanced Electronics Final Report on Contract AF(19(604)-1183, Air Force Cambridge Research Labs., Cambridge, Massachusetts, 1960.

65. Mitchell,M.E. "Error-Trap Decoding of Cyclic Codes", G.E. Report No. 62MCD3, General Military Communications Department, Oklahoma City, Oklahoma, December, 1962.

66. Peterson,W.W., "*Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes*", IRE Trans. on Information Theory, IT-6, pp. 459-470, September, 1960.

67. Peterson,W.W. "*Error-Correcting Codes*", The M.I.T. Press, Cambridge, Massachusetts, and John Wiley, New York, 1961.

68. Peterson,W.W., and E.J.Weldon,Jr., "*Error-Correcting Codes*", 2nd Edition, The M.I.T. Press, Cambridge, Massachusetts, 1970.

69. Pettit,R.H., "*Use of the null-zone in voice communications*", IEEE Trans. Vol. COM-13, No.2 (June), p.175, 1965.

70. Prange,E., "*Cyclic Error-Correcting Codes in Two Symbols*", AFCRC-TN-57, 103, Air Force Cambridge Research Center, Cambridge, Massachusetts, Sept.,1957.

71. Prange,E., "*Some Cyclic Error-Correcting Codes with Simple Decoding Algorithms*", AFCRC-TR-58-156, Air Force Cambridge Research Center, Cambridge, Mass., April, 1958.

72. Prange,E., "*The Use of Coset Equivalence in the Analysis and Decoding of Group Codes*", AFCRC-TR-59-164, Air Force Cambridge Research Center, Cambridge, Massachusetts, June, 1959.

73. Prange,E., *"The use of Information sets in decoding cyclic codes"*, IEEE Trans. on Information Theory, IT-8, pp. 85-89, September, 1962.

74. Preparata, E.P., *"A class of optimum nonlinear double error correcting codes"*, INF. Cont., 13, 1968, pp. 378-400.

75. Ramsey,J.L., *"Realization of optimum interleavers"*, IEEE Trans. Inf. Theory, IT-16, 338-345, May, 1970.

76. Rappaport,S.S. & Kurz, L., *"Optimal decision thresholds for digital signalling in non-Gaussian noise"*, IEEE Int. Conv. Rec., Part 2, p 198, 1965.

77. Reiger, S.H., *"Codes for the Correction of 'Clustered' Errors"*, IRE Trans. on Information Theory, IT-6, pp. 16-21, March, 1960.

78. Richer,I., *"A simple interleaver for use with Viterbi decoding"*, IEEE Trans. Commun., COM-26, 406-408, March, 1978.

79. Rudolph,L., *"Easily Implemented Error-Correction Encoding-Decoding"*, G.E. Report 62MCD2, General Electric Corporation, Oklahoma City, Oklahoma, December, 1962.

80. Rudolph,L., and M.E.Mitchell., *"Implementation of Decoders for Cyclic Codes"*, IEEE Trans. on Information Theory, IT-10, pp. 259-260, July, 1964.

81. Shannon,C.E. and W.Weaver., *"A Mathematical Theory of Communication"*, University of Illinois Press, Urabana, Illnois, 1949. (The first part of this book is a reprint of Shannon's paper, *"A Mathematical Theory of Communications"*, Bell Systems Tech.J., 27, 1948.)

82. Silverman,R.A., and Balsar,M., *"Coding for Constant-Data-Rate Systems-part I.A New Error-Correcting Code"*, Proc. IRE, Vol. 42, pp. 1428-1435, Sept., 1954.

83. Smith,J.S., *"Error control in duobinary data systems by means of null-zone detection"*, IEEE Trans. Vol. COM-16, No. 6 (Dec.), p. 825, 1968.

84. Sullivan,N.J., and Heaton,A.G., *"Transient Frequency Response of Transmittance Peaked I.F. Filters with Application to Null-zone Detection"*, Elec. Letters, Vol.5, p.423, Sept., 1969.

85. Sundberg,C.E., *"Reliability numbers matching binary symbols for Gray-coded MPSK and MDPSK signals"*, Dept. Telecom. Th., Lund. Univ., Sweden, Tech. Rep. TR-66, 1975.

86. Sundberg,C.E., *"One-step majority logic decoding with symbol reliability information"*, IEEE Trans., IT-21, pp. 235-242. No.2 (March), 1975.

87. Sundberg,C.E., *"Asymptotically optimum soft-decision decoding algorithms for Hamming codes"*, Elec. Letters, Vol.13, No.2, p.38, 20th Jan, 1977.

88. Tong.S.Y., *"Burst trapping techniques for a compound channel"*, Bell Telephone Laboratories Technical Memorandum, 1968.

89. Viterbi,A.J., Odenwalder,J.P., Bar-David,I., and Kumm,K.M., *"RFI/coding sensitivity analysis for tracking and data relay satellite "*, (TDRSS), Phase II Final Peport, Linkabit Corporation, January,1979.

90. Weldon,E.J.,Jr., *"A Comparsion of an Interleaved Golay Code and a Three-Dimensional Product Code"*, Final Report, USNELC Contract NO095368M5345, Aug.1968.

91. Weldon.E.J.,Jr., *"Encoding and decoding for binary input, M-ary output channels"*, presented at the IEEE Int. Symp. Information Theory, Noordwijk, the Netherlands, June, 1970.

92. White,H.E., *"Failure-correction decoding"*, IEEE Trans, Vol. COM-15, No.1(Feb.), p.23, 1967.

93. Woodward,M.E., and Tsigiroglou,K., *"Performance Evaluation of Single-Burst-Error Correction Cyclic Block Codes on a Digital Audio Channel"*, Report on EMI Consultancy, 69183 Consultancy/AEA/AF, Jan. 1979.

94. Wozencraft,J.M., and Jacobs.I.M., *"Principles of Communications Engineering"*, New York, Wiley,1965.

95. Zieler,N., *"On Decoding Linear Error Correcting Codes-1"*, IRE Trans. on Information Theory, Vol. IT-6, pp. 450-459, September, 1960.

96. Zissos,D., *"System Design with Microprocessors"*, Academic Press, 1968.