Developments in Channel Coding for Error and Spectral Control

A thesis submitted for the degree of

Doctor of Philosophy

in Electronic and Electrical Engineering

March 1997

Simon Fragiacomo

Department of Electrical and Electronic Engineering

University College London

Summary

This thesis is concerned with channel coding. Channel coding consists of error control coding and *line coding* (LC). The basic definitions and concepts of both *error correcting codes* (ECCs) and LC are initially examined, followed by the presentation of a number of existing coding algorithms. Where appropriate, computer simulation is used to establish their limitations. Certain new codes are then devised which offer improved performance.

Finally, following on what is now an established trend, error correcting and line codes are combined to form *error correcting line codes* (ECLCs), which may offer superior performance compared to the use of a cascaded scheme.

Specifically, the first chapter contains some basic definitions, the thesis outline and a summary of the contributions.

Chapter 2 introduces the basic concepts behind *error correcting codes* and *soft decision decoding* (SDD) together with a brief description of the well-established Chase algorithms. Their advantages and limitations will be examined and used to generate novel SDD algorithms in chapter 5.

The basic concepts of line coding are introduced in chapter 3. In addition, a new family of single added bit line codes is also presented. This offers reasonably good line characteristics with very small compromise to rate.

Chapter 4 is concerned with simulation as a means of evaluating the performance of coded systems. A conventional simulation technique is initially presented and used for assessing the performance of BCH codes. This proves inadequate for

simulating the very low error rates of modern communication systems, especially when SDD is used. Two novel simulation acceleration algorithms are therefore introduced to alleviate this problem. These will only simulate code words that affect the *residual bit error rate* (RBER) and simply calculate the effects of the code words which are correctly decoded. The novel simulator algorithms are used in subsequent chapters to determine the performance of the proposed new codes.

Chapter 5 introduces the new *generalised Chase* (GC) algorithms, followed by the *adaptive immediate decision* (AID) and *test pattern elimination* (TPE) algorithms. These can be used to offer near *maximum likelihood* (ML) performance with minimum increase in complexity.

Chapter 6 is concerned with combined EC and line codes to form ECLCs. These can offer both tight line coding characteristics and good decoding performance. Some emphasis is placed on implementation appropriate to very high bit rate systems.

Finally chapter 7 brings the thesis to a conclusion and provides recommendations for future work.

Statement of Originality

Unless otherwise stated in the text, the work presented in this thesis was carried out by the candidate. It has not been presented previously for any degree, nor is at present under consideration by any other degree awarding body.

Candidate:

Simon Fragiacomo

Director of Studies:

Professor J. J. O'Reilly

Acknowledgements

I wish to express my gratitude to Professor John J. O'Reilly for his guidance and support throughout the course of this study and the preparation of this thesis. I would also like to thank him for his friendship and moral support, for which I am deeply indebted.

I would also like to thank a number of people for their contributions, discussions and encouragement: Drs Andrew Popplewell and Yi Bian for their patience and advice. Also a special 'thanks' should go to Chris 'there's at least six ways you can do this' Matrakidis for many fruitful conversations.

I would also like to thank all my friends at UCNW, BT labs and everybody at the UCL Telecoms group, for making these last years so enjoyable.

Last, but certainly not least, I would like to thank my family for their support throughout all these years.

To my Father and Mother

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

A major aim of communication theory is to devise methods with which signals can be transmitted though imperfect media with the highest possible degree of reliability and efficiency.

In order to achieve this, the signal format at the transmitting end must be chosen so that it will offer maximum resistance to the channel impairments. Additionally, the receiver must be able to recover as much of the original signal as possible. The application of coding is one method of achieving increased reliability.

Generally, coding is a form of mapping whereby a given string of information bits is converted into another sequence which may have added redundancy. We can divide coding into three major areas:

1. Source coding, which has two functions: first to transform a message source into a string of digital symbols and second, to remove any redundancy from

the word thus increasing efficiency.

2. Cryptographic coding, which is used to increase the security level of a signal by concealing the message. This is done to ensure that only authorised recipients can decode and obtain the transmitted information.

3. Finally, channel coding which aims to increase the reliability of a channel by using added redundancy.

This thesis is concerned with channel coding, which can be further divided into two main sub-areas:

1. *Line coding* (LC), in which the extra information is used to tailor the transmitted signal in such a way as to match the characteristics of the communications link.

   Line coding can be used, for example, to modify the spectral characteristics of the signal, (e.g. by placing constraints on the *running digital sum* (RDS)), or to place bounds on the runlength.

2. Error control coding, whereby extra bits introduced at the transmitter are utilised at the receiver to detect and possibly correct any errors that may have occurred. This is further sub-divided into *automatic repeat request* (ARQ) and *forward error correction* (FEC). Both ARQ and FEC will be examined in more detail in the next chapter.

Figure 1.1 summarises the structure of the coding field, as described above.

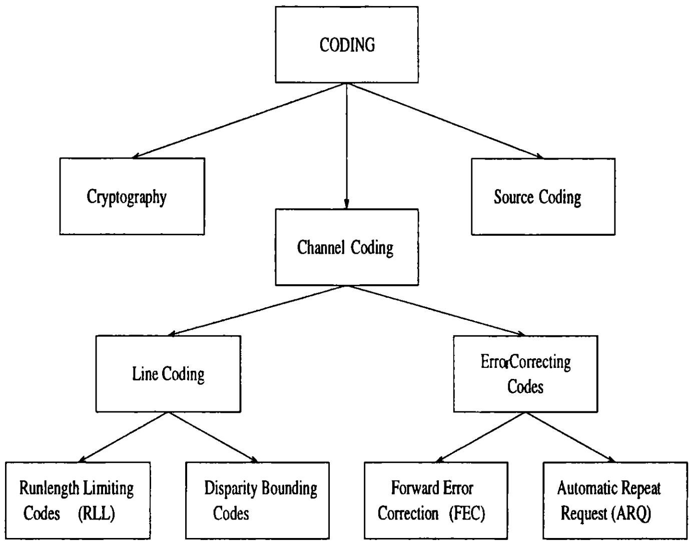


Figure 1.1: Coding tree diagram.

Even though line and error control coding are two distinct operations they both aim to increase the reliability of a system [1, 2, 3]. With the introduction of digital systems in all aspects of modern life, the issue of data integrity is becoming increasingly more important. Channel coding is a very attractive way of achieving this.

This thesis concentrates on channel coding. A number of existing line and error correcting codes will be presented and simulated, to identify their shortcomings. In a number of cases, new codes will be introduced addressing these limitations. The codes used will be very general in nature and simple to implement.

The basic concepts behind channel coding are initially presented. These are then used to build more complex line and error control codes. However, it is not uncommon for a designer to incorporate both aspects of channel coding in a system. Thus, if a conventional cascaded scheme is used, the data goes first through an *error correcting* (EC) encoder and then through a line encoder. The process is reversed at the decoder. It will be shown that this cascaded scheme can be inefficient and reduces the overall decoding performance. For these reasons combined *error correcting line codes* (ECLCs) have been devised which offer improved performance [4, 5].

Finally, the concept of *soft decision decoding* (SDD) will also be introduced. This is a technique which combines the demodulation and decoding processes. A number of the codes presented in this thesis can be improved by the use of SDD. The latter increases the complexity of the receiver but offers increased decoding power without adding extra redundancy.

A particular feature of the thesis is an exploration of the benefits that arise from employing SDD with ECLCs. This enables significant improvements in both rate and decoding power to be realised. There is currently considerable interest in channel coding for high bit rate systems, such as undersea optical telecommunications transmissions, which provided the original motivation for this study. For this reason only low complexity codes (which allow high bit rates to be realised) will be considered.

1.2 Thesis Organisation

The structure of this thesis can be briefly summarised as follows:

Following this introduction, chapter 2 contains the fundamentals of error control coding together with examples of various simple codes. This chapter is used as a basis for the introduction and development of more advanced codes which are presented in later chapters. In addition, the well-established Chase SDD algorithms are extensively examined due to their critical role in the following chapters, and especially in chapter 5.

Chapter 3 presents the basics of line coding together with a family of simple 'added bit' codes. These offer tight runlength and disparity bounds while being very simple to implement. A number of the added bit codes are used in chapter 6 to form error correcting line codes.

In chapter 4, the need for simulation in channel coding is presented together with a basic ECC simulation. The latter is used as a basis for developing two new simulation acceleration algorithms which are used throughout the thesis to validate theoretical results. These work by 'eliminating' a large number of code words without affecting the accuracy of the simulation.

Chapter 5 introduces three novel SDD algorithms which offer significant performance improvements. Specifically, the *generalised Chase* (GC) utilises an increased number of *test patterns* to achieve improved decoding performance; the AID algorithm then uses threshold decoding to reduce the average number of test patterns without affecting the decoding performance. Finally, the TPE algorithm reduces the number of test patters required for SDD, by eliminating those that produce the same *estimated error pattern* (EEP).

In chapter 6, existing concatenated and ECLCs are presented together with the reason behind the need for error correcting line codes. These are used as a basis for the generation of novel codes, which also utilise SDD to provide better performance with minimal increase in complexity.

Finally, chapter 7 brings the thesis to a conclusion and provides some recommendations for future work.

1.3 Summary of Main Contributions

The research presented in this thesis examines channel coding schemes suitable for high bit rate systems. The major contributions fall into four main interlinked areas and can be summarised as follows:

- New line codes: a new family of 'added bit' line codes was introduced, which resulted in the harmonisation into a single identifiable family of the disparate $nB1X$ codes.
- Improved simulation techniques, where two novel simulation acceleration techniques are developed. These significantly reduce the amount of time required for obtaining statistically accurate results without compromising accuracy.
- Novel SDD algorithms: a number of novel SDD algorithms are introduced which offer improved decoding performance and increased decoding speed without significantly increasing the complexity of the decoder.
- New error correcting line codes: SDD was combined with conventional error correcting line codes thus producing novel codes which can offer both acceptable decoding performance and reasonably good line coding characteristics.

A number of conference papers have been accepted for presentation and publication. These are the following:

1. Y. Bian, J. O'Reilly, A. Popplewell, S. Fragiacomo: "New Simulation Techniques for Evaluation Telecommunications Transmission Systems with FEC", *Fifth IEE Conference on Telecommunications*, March 1995, UK.

2. S. Fragiacomo, Y. Bian, A. Popplewell, J. O'Reilly: "A New Low Complexity Near ML Soft Decision Decoding Algorithm for Linear Block Codes", *IEE Singapore International Conference on Communication Systems*, IEEE ICCS/ISPACS '96, 25-29 November 1996, Singapore.

3. Y. Bian, A. Popplewell, J. O'Reilly, S. Fragiacomo, R. Blake: "FEC for Future Trans-Oceanic Optical Systems", *Fifth IEE Conference on Telecommunications*, Brighton, March 1995, UK.

4. S. Fragiacomo, C. Matrakidis, J. O'Reilly: "Exploiting Soft Decision Decoding for Error Correcting Line Codes", *IEE Singapore International Conference on Communication Systems*, IEEE ICCS/ISPACS '96, 25-29 November 1996, Singapore.

5. S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A Novel Error Correcting Line Code", *Third Communication Networks Symposium*, 8-9 July 1996, Manchester, UK.

6. S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A New Error Correcting Line Code", *ITS/IEEE ROC&C '96 International Telecommunications Symposium*, October 28-31 1996, Acapulco, Mexico.

7. S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A Novel Error Correcting Line Code", *Networks and Optical Communications (NOC) - Post-Deadline Ses-*

sion, June 25-28 1996, Heidelberg, Germany.

8. S. Fragiacomo, C. Matrakidis, J. O'Reilly: "Soft Decision Error Correcting Line Code for Optical Data Storage", *9th Annual Meeting, LEOS 96*, 18-21 November 1996, Boston, USA.

9. S. Fragiacomo, C. Matrakidis, A. Popplewell, Y. Bian, J. O'Reilly: "An Accelerated Simulation Technique for Evaluating Communication Systems Utilising FEC", *Networks and Optical Communications (NOC)*, June 17-20 1997, Antwerp, Belgium.

10. S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A Class of Low Complexity Line Codes", *International Symposium on Information Theory, ISIT 97*, 29 June 1997, Ulm, Germany.

11. S. Fragiacomo, C. Matrakidis, J. O'Reilly: "Performance Aspects of a Class of Low Complexity Line Codes", *International Conference on Signal Processing, ICSPAT 97*, September 1997, San Diego, USA.

In addition, a number of journal papers have been submitted. The next chapter begins by presenting some basic concepts of error control coding and soft decision decoding.

Bibliography

[1] K. W. Cattermole: "Mathematical Foundations for Communication Engineering", Volume 2, Pentech Press, 1986.

[2] S. Lin, D. Costello: "Error Control Coding", Prentice Hall, 1983.

[3] "Algebraic Coding Theory and Applications", Edited by G. Longo, Springer-Verlag.

[4] A. Popplewell: "Combined Line and Error Control Coding", *PhD Thesis, University of Wales*, Bangor, 1990.

[5] A. Kokkos: "Contributions to Modulation and Coding", *PhD Thesis, University of Wales*, Bangor, 1990.

Chapter 2

Channel Coding

2.1 Introduction

In this chapter a brief overview of *forward error control* (FEC) and *soft decision decoding* (SDD) will be presented. As discussed before, channel coding can be divided into two main categories: FEC and *line coding* (LC). This chapter introduces the basic concepts behind FEC, while chapter 3 presents the basic concepts of line coding.

2.2 Error Control Codes

In general, error control codes can be divided into two types [1]:

1. *Automatic repeat request* (ARQ), whereby the receiver can only detect errors. If these occur, then a feedback path is used to request the re-transmission of the erroneous data. ARQ systems can be divided into two main categories: stop-and-wait ARQ and continuous ARQ. With stop-and-wait ARQ

the transmitter will only send the next word if the received word contains no errors. Continuous ARQ will send words and receive acknowledgements continuously. If errors are detected the offending code word will be re-transmitted. ARQ systems can be simple to implement since they only require error detecting codes and are efficient in high *signal to noise ratio* (SNR) communication links. However, a number of disadvantages are also present:

- They require a feedback path to send the repeat request. This means that stop-and-wait ARQ requires a half-duplex channel while continuous ARQ requires a full-duplex channel.
- They can be very inefficient, especially if high channel error rates exist because a high number of repeat requests will be made.
- In high bit rate or long distance systems where a significant delay exists, continuous ARQ will be used. This can be more efficient but will require large buffering systems.

For the above reasons ARQ systems are not considered here.

2. The second error control strategy is *forward error correction* (FEC). This utilises codes which can detect and correct errors at the receiver, termed *error correcting codes* (ECCs). Such codes are more complicated to implement but do not require feedback paths. In addition, they are better suited to relatively low SNR applications. For these reasons, FEC is examined in more detail in this study.

2.2.1 Forward error correction

The theoretical basis for error correcting codes was developed during the late 1940's by Shannon [2]. He suggested that the elimination of errors in a received digital bit stream was possible, if the latter was properly encoded. Encoding usually requires the introduction of redundancy. Shannon also proved that any number of errors can be corrected, provided the block length is large enough.

The challenge in coding theory is to discover codes which can correct a large number of errors while minimising redundancy and complexity.

Error correcting codes can be divided into binary and multi-level codes. If the encoded data stream at the transmitter can only obtain two distinct values then our code is a binary one, otherwise it is a multilevel one. Both these types of code can be further sub-divided into block or tree codes. Block codes, which are of interest here, were introduced by Hamming in the 1950's [3]. They differ from tree codes in that there is no 'memory' during the encoding process and the produced bits depend only on the current information word.

Finally, either of these codes can be linear or non-linear. A binary block code is linear if the modulo-2 sum of any two of its code words is also a code word. The best known linear block codes are cyclic, where a cyclic shift of any code word also produces a code word. The most frequently used ones are the BCH codes which include the Reed-Solomon and Hamming families. The most common tree codes are convolutional codes, where the check bits are mixed with the information bits in a continuous manner.

Figure 2.1 presents the family of error correcting codes.

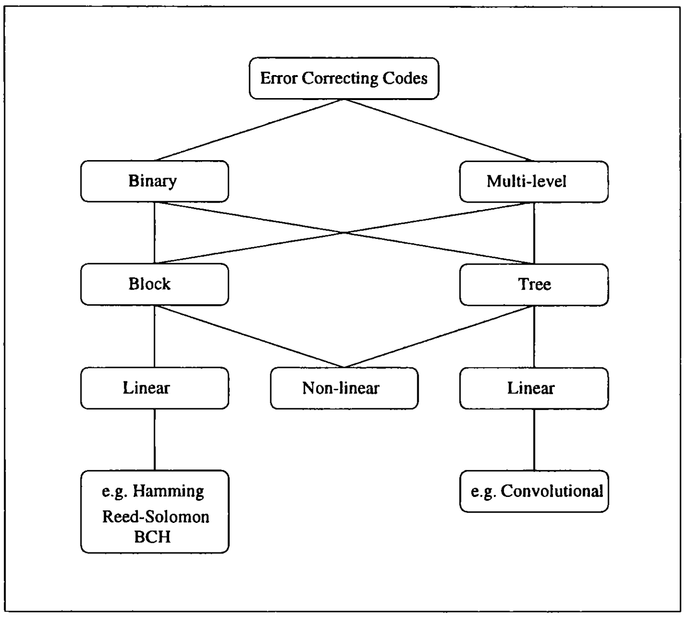


Figure 2.1: Error correcting code tree diagram.

This thesis aims to develop error correcting and line codes appropriate for modern high bit rate systems. For this reason, only linear binary block codes will be considered as they are likely to be the simplest to implement and usually require a minimal amount of decoding time. A widely used group of such codes are the *Bose-Chaudhuri-Hocqenghem* (BCH) codes, introduced in 1960 [4]. These will be used as an example whenever a specific code example is required.

2.3 Generating Linear Binary Block Codes

Figure 2.2 is intended to make linear block error correcting codes easier to appreciate. This represents the block diagram of a complete communications system, utilising an (n, k) ECC encoder.

The digital data source will generate a continuous stream of information bits. The error correcting code encoder will divide this stream into blocks of k bits each.

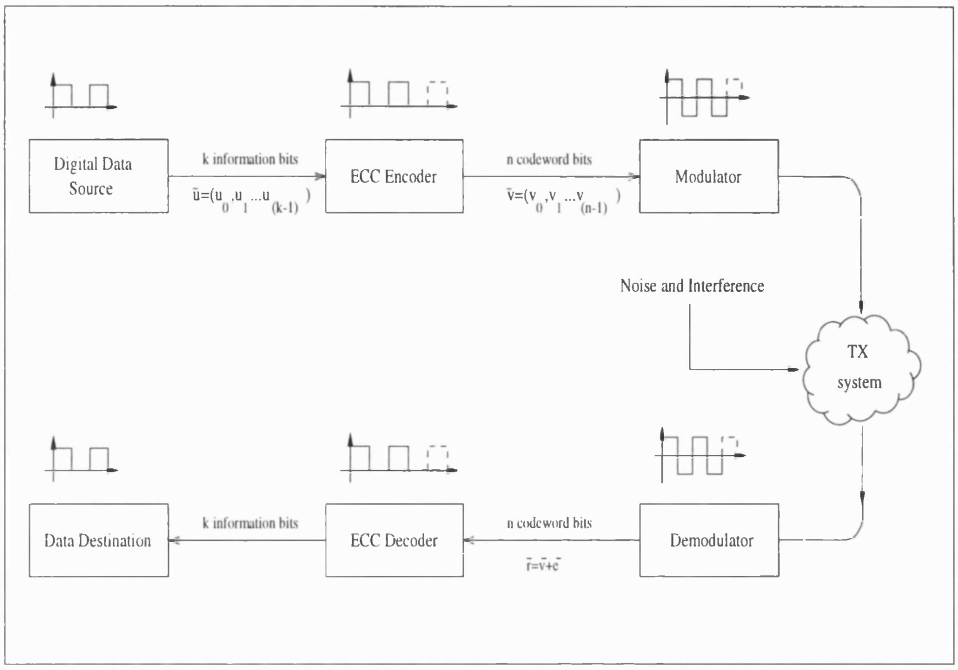


Figure 2.2: Block diagram of a communications system.

These are represented by a binary $k-tuple$ $\bar{u} = (u_0, u_1, \ldots, u_{k-1})$ called a *message*. This implies that for a binary code there are 2^k possible messages.

The encoder 'contains' the generator matrix G which has as rows a set of k linearly independent code words. These are called the 'basis code words' and make the row space of G the linear code. If all the linear combinations of the basis code words are taken ($\bar{v} = \bar{u} \cdot G$) then 2^k n-bit code words will be generated.

The error correcting capabilities of a code depend on the Hamming distance (d) separating the code words. The Hamming distance of two code words of length n (where $k < n$) is defined as the number of places in which their bits differ [5]. The minimum distance (d_{min}) of a code is defined as the smallest possible Hamming distance between any two code words. For BCH codes (d_{min}) is an odd number. Such a code can detect and correct up to $t = \frac{(d_{min}-1)}{2}$ errors and t is therefore termed the 'error correcting capability' of the code.

At the output of the encoder the k-bit information vector will therefore be transformed to an n-bit code word which now has error correcting properties. The added $(n-k)$ bits are called the parity check digits, or parity bits. The n-bit code word is then modulated and transmitted. The same process is repeated for the next k information bits. An (n, k) code of minimum distance (d_{min}) has therefore been generated.

At the receiver, the received waveform is initially demodulated. This produces a binary stream of code words. If no errors have occurred then the received vector $\bar{r}$ will equal the transmitted one, i.e. $\bar{r} = \bar{v}$. If errors have occurred, then it can be assumed that they will have formed the error vector $\bar{e}$. In such a case $\bar{r} = \bar{v} \oplus \bar{e}$, where $\oplus$ indicates modulo-2 addition.

It is up to the decoder to attempt to detect and possibly correct the errors that may have occurred. To do this it uses the parity check matrix H. This is a $((n-k) \times n)$ matrix where any vector in the row space of G is orthogonal to the rows of H. Therefore, for any $\bar{v}$ we get $\bar{v} \cdot H^T = 0$, where H^T is the transpose of H. In order to detect if errors have occurred, $\bar{r}$ is multiplied by H^T. If $\bar{r} \cdot H^T = 0$ then either no errors have occurred, or the error correcting capability (t) of the code has been exceeded.

If error correction (as opposed to error detection) is also required, then the previous formula must be expanded. At the receiver $\bar{r}$ is multiplied by H^T, i.e.

$$\bar{r} \cdot H^T = (\bar{v} + \bar{e}) \cdot H^T = \bar{v} \cdot H^T + \bar{e} \cdot H^T$$

Since $\bar{v}$ is always a valid code word, $\bar{v} \cdot H^T = 0$, so $\bar{r} \cdot H^T = \bar{e} \cdot H^T$. This product is called the *syndrome* (S) and only depends on the error pattern. If the syndrome is zero then no errors have occurred; $2^{(n-k)}$ unique syndromes exist, each corresponding to a specific error vector. Using the syndrome information the error position can be located. Since our codes are binary there are only two possible distinct bit states, a logic 1 or a logic 0. Therefore if the positions of the errors in the code word are known, error correction is also possible by performing a simple inversion of those positions.

2.3.1 A simple ECC example

To provide an illustrative framework for some basic definitions, an example of a very simple linear binary $(7, 4)$ code with a minimum Hamming distance $d_{min} = 3$ will be considered. This has $k = 4$ information bits encoded into $n = 7$ code word bits. Therefore, 2^k possible distinct messages will be encoded into 2^n code words using a one-to-one correspondence. The generator matrix G of such a code is of the following form:

$$G = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ g_3 \end{bmatrix} = \begin{bmatrix} 1\ 1\ 0\ 1\ 0\ 0\ 0 \\ 0\ 1\ 1\ 0\ 1\ 0\ 0 \\ 1\ 1\ 1\ 0\ 0\ 1\ 0 \\ 1\ 0\ 1\ 0\ 0\ 0\ 1 \end{bmatrix} \tag{2.1}$$

while the parity check matrix H will be the following:

$$H = \begin{bmatrix} 1\ 0\ 0\ 1\ 0\ 1\ 1 \\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \end{bmatrix} \tag{2.2}$$

Assume that a message $\bar{u} = (1101)$ must be transmitted using the above code. To achieve this, the message must be multiplied with the generator matrix so that the code word vector $\bar{v}$ is produced, i.e.

$$\bar{v} = \bar{u} \cdot G = (1101) \cdot G = 1 \cdot g_0 + 1 \cdot g_1 + 0 \cdot g_2 + 1 \cdot g_3 =$$
$$(1101000) + (0110100) + (1010001) = 0001101$$

Thus the transmitted EC code word will be 0001101. Because the $(7,4)$ code is a systematic one, the four last bits contain the original message in the correct order, while the initial three bits are the parity bits required for error detection and correction at the receiver. Table 2.1 contains the mappings for all possible four bit information words.

2.4 Decoding a Simple ECC

In the previous section, a 4-bit information message was encoded into a 7-bit code word using a linear block code. In this section, the received word will be decoded and up to t errors will be corrected. Since $t = \frac{(d_{min}-1)}{2}$ and for our code $d_{min} = 3$ then $t = 1$, i.e. it is a single error correcting code.

Assume that $\bar{r} = (r_0, r_1, r_2, r_3, r_4, r_5, r_6)$ is the received vector. The syndrome S is defined as

Information word	$(7,4)$ code words
0000	000 0000
0001	101 0001
0010	011 0010
0011	010 0011
0100	011 0100
0101	110 0101
0110	010 0110
0111	001 0111
1000	110 1000
1001	011 1001
1010	001 1010
1011	100 1011
1100	101 1100
1101	000 1101
1110	010 1110
1111	111 1111

Table 2.1: Linear block code with $k = 4$ information bits and $n = 7$ code word bits.

$$S = \bar{s} = \bar{r} \cdot H^T = (s_0, s_1, \, \dots \, , s_{(n-k-1)}). \tag{2.3}$$

From this it is deduced that the syndrome is simply the sum of the received parity digits and those re-computed at the decoder. Thus if $S \neq 0$ an error has been detected.

Using equation 2.3 in our example $(7,4)$ code the following equation is obtained:

$\bar{s} = (s_0, s_1, s_2) = \bar{r} \cdot H^T = (\bar{u} + \bar{e}) \cdot H^T = \bar{u} \cdot H^T + \bar{e} \cdot H^T = 0 + \bar{e} \cdot H^T = \bar{e} \cdot H^T$

Assuming that the vector $\bar{r} = (1001001)$ has been received, i.e. a single error exists in bit position 5 the syndrome will be equal to:

$$S = \bar{r} \cdot H^T = (1001001) \cdot \begin{bmatrix} 100 \\ 010 \\ 001 \\ 110 \\ 011 \\ 111 \\ 101 \end{bmatrix} = (111) \tag{2.4}$$

Therefore $S \neq 0$ and an error has been detected. In order to detect the error position it is noted that

$$S = (s_0, s_1, s_2) = (1,1,1) = \bar{e} \cdot H^T \Rightarrow$$

$$(111) = (e_0, e_1, e_2, e_3, e_4, e_5, e_6) \cdot \begin{bmatrix} 100 \\ 010 \\ 001 \\ 110 \\ 011 \\ 111 \\ 101 \end{bmatrix} \tag{2.5}$$

.

Using equations 2.4 and 2.5 the following error vectors are derived:

$$1 = e_0 + e_3 + e_5 + e_6 \tag{2.6}$$

$$1 = e_1 + e_3 + e_4 + e_5 \tag{2.7}$$

$$1 = e_2 + e_4 + e_5 + e_6 \tag{2.8}$$

A number of possible solutions exist that satisfy the above equations. If maximum likelihood decoding is used for error location then the error pattern with the minimum number of corrected positions is selected. In this case, this is the (0000010) error vector. If this is added to our received vector the transmitted vector will be obtained.

In chapter 4, Berlekamp's iterative algorithm will be briefly presented. This is an algorithm for generating the error locator polynomial, appropriate for more complex and powerful codes. It is a well-established technique which lends itself to software implementation.

2.5 Soft Decision Decoding

In the previous sections it was demonstrated how the information bits could be encoded and decoded so that error detection and correction were possible. At the receiver, only the algebraic properties of the code were utilised to locate any possible errors. This is termed *hard decision decoding* (HDD).

In this section, the basic principles of *soft decision decoding* (SDD) are presented. SDD uses the analogue information to try to increase the error correcting capability of the code. As has been mentioned before, a code of minimum distance d_{min} can correct up to t errors. The use of SDD allows this limit to be exceeded, under certain conditions. The penalty is the increased complexity of the decoder and the fact that the t-error correcting capability may not be guaranteed anymore.

SDD techniques are not suitable for very noisy systems. However, modern telecommunication and data storage systems usually have very low error rates and can thus benefit from the application of such techniques.

A modified communications system which includes SDD, is shown in figure 2.3.

Here extra information is provided to the decoder in the form of the analogue values of the received bits. These values are provided by the demodulator and are used to facilitate the decoding process by indicating possible error positions.

SDD algorithms can be grouped into two broad classes:

(a) Minimum distance (or minimised sequence error rate) decoding algorithms, based on [6].

(b) Trellis decoding algorithms, adapted for block codes.

A number of SDD algorithms exist, such as [8, 9, 10, 11], but the most widely

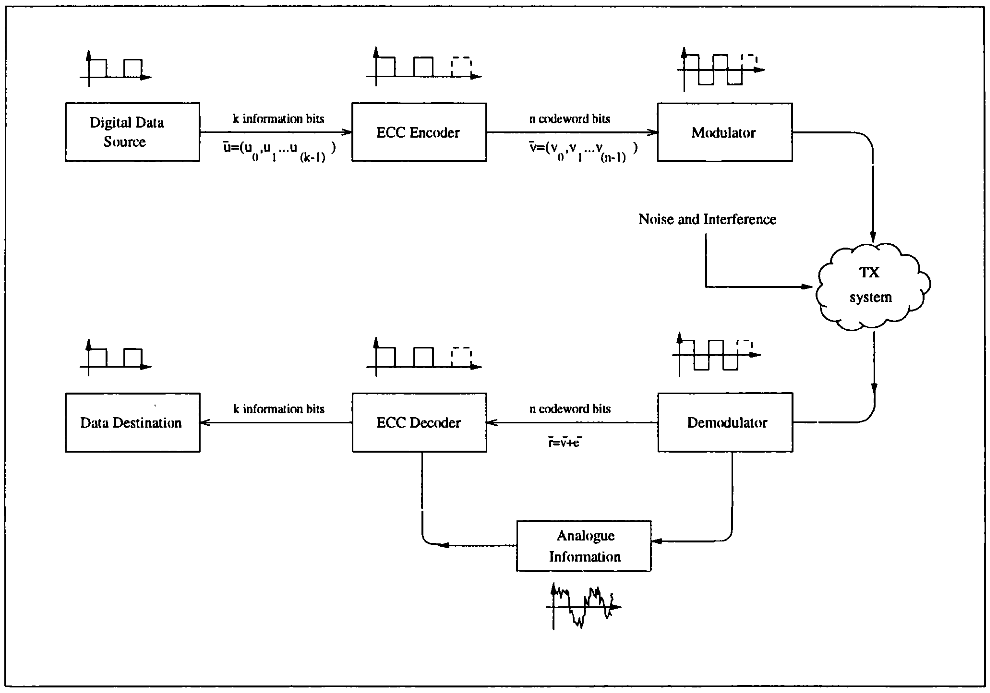


Figure 2.3: Communications system (with SDD) block diagram.

used are the Chase algorithms. This is because they can offer a balance between decoding power and complexity. In the following sections the Chase algorithms will be examined in detail. Their shortcomings will be identified and then addressed in chapter 5.

2.5.1 The Chase algorithms

In 1972, D. Chase [7] suggested three different SDD algorithms of type (a), each of which provided a different number of test patterns, allowing a trade-off between performance and complexity. Specifically, the Chase decoder generates a set of possible error patterns, called the *test patterns* (TPs). These are then sequentially perturbed with the received word and taken through a conventional HDD decoder. This may result in a possible error pattern being generated, termed an *estimated* *error pattern* (EEP). The EEP is used to indicate the bit positions which are deemed to contain an error and each is assigned a confidence value, provided by the analogue information. After HDD of all possible patterns has taken place, the one with the highest confidence value is selected and added to the received word to provide the corrected word.

Chase suggested three different algorithms, each examining a different number of TPs. The TPs are used to invert a number N of *least confidence bit* (LCB) positions. These are defined as the bit positions closest to the decision threshold. The reason behind the use of TPs is that if the voltage value of a received bit is very close to the decision threshold, this bit is very likely to be in error. Thus if the syndrome calculations indicate that the received code word contains errors, these are more likely to be in the LCB positions than anywhere else. By systematic inversion of different combinations of these positions, the errors within a code word could be corrected.

An ECC utilising HDD can correct upto $\lfloor \frac{d_{min}-1}{2} \rfloor$ errors. The use of an SDD algorithm, in conjunction with an ECC, allows a higher average number of errors per code word to be corrected. However, unlike a conventional ECC, the use of any SDD algorithm may not guarantee the correction of up to t errors within a code word.

The flow diagram of the decoding process for the Chase algorithms is shown in figure 2.4 and is explained in more detail in the following section.

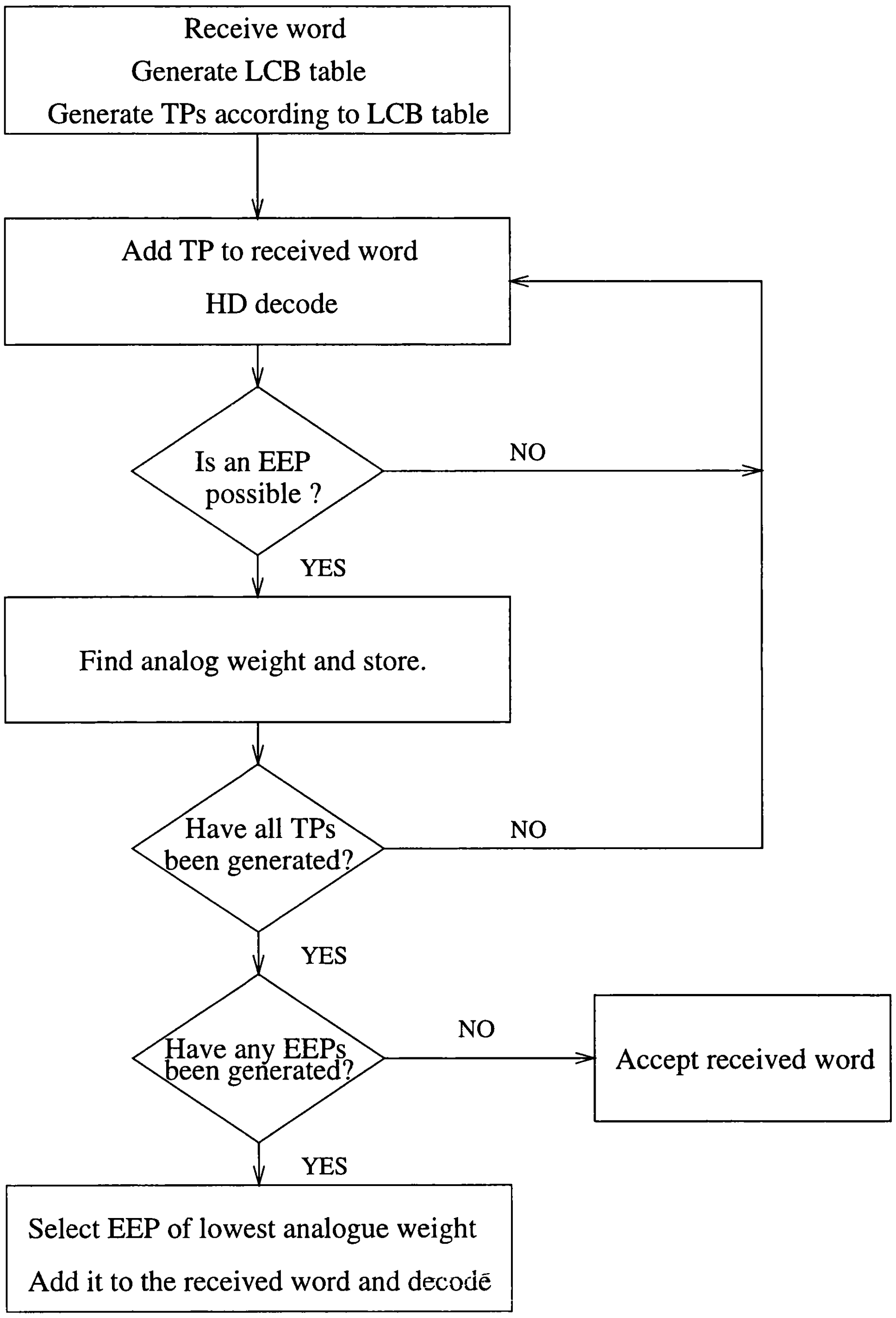

Receive word
Generate LCB table
Generate TPs according to LCB table
Add TP to received word
HD decode
Is an EEP possible ?
NO
YES
Find analog weight and store.
Have all TPs been generated?
NO
YES
Have any EEPs been generated?
NO
Accept received word
YES
Select EEP of lowest analogue weight
Add it to the received word and decode

Figure 2.4: Flow diagram of the Chase algorithms.

2.5.2 Mechanics of the Chase algorithms

Assume that for a given communication system the transmitted word is defined as $\overline{v} = v_0, v_1, \ldots v_{(n-1)}$ and the received word as $\overline{r} = r_0, r_1, \ldots r_{(n-1)}$, where $r_i = v_i + no_i$ for $0 \leq i \leq (n-1)$ and no_i is defined as the noise amplitude. Let t be the error correcting capability of the (n, k) ECC.

The resultant *error pattern* (EP) indicates the bit positions where $\overline{v}$ and $\overline{r}$ differ. $\overline{e} = e_0, e_1, \ldots e_{n-1}$ is termed the error vector and is defined by $e_i = v_i \oplus r_i$ for $0 \leq i \leq (n-1)$ while the binary weight of $\overline{e}$ is termed $W(\overline{e})$ and defined as the total number of non-zero elements in the sequence. A decoder will try to find a code word that satisfies $W(\overline{e}) \leq (\frac{d_{min}-1}{2})$ where d_{min} is the minimum Hamming distance of the code.

In the Chase algorithms, a test pattern $TP = tp_0, tp_1, \ldots tp_{(n-1)}$ for $0 \leq i \leq (n-1)$ of length n is initially generated. This is done by selecting a number (N_{Chase}) of LCBs and producing a set of n-bit sequences containing possible permutations of these N_{Chase} positions. This generates a set of upto $2^{N_{Chase}}$ TPs. The initial TP will always be the all zero word.

Each TP is then modulo-2 added to the received word $\overline{r}$ and the resultant sequence $\overline{r}' = \overline{r} \oplus TP$ is conventionally decoded using any suitable HD decoder. Since the addition of the TPs has the effect of inverting a number of LCB positions it is likely that some of the resultant sequences will have a reduced number of errors compared to the received code word.

If a conventional HD error correcting decoder was used and the initial number of errors exceeded t then the decoding operation would fail. With the addition of the TP procedure, it is possible that the total number of errors is reduced (or even eliminated) thus allowing conventional decoding to take place.

The received word with the added TP will be taken through the EC decoder, which will either produce a *decoder error pattern* (DEP) or fail to decode. The DEP will then be added to the TP to produce an $n-$bit sequence termed *estimated error pattern.* Thus, $EEP = (eep_0, eep_1, \ldots eep_{(n-1)}) = TP \oplus DEP$. If the addition of the TP has introduced more errors (instead of eliminating them) it is possible that the decoding will fail. In such a case no EEP will be produced. This is the reason for always starting with the all zero word as the initial TP; if the error pattern contains t or less errors the all zero TP will allow the decoder to operate properly. If not, then only the removal of errors will allow decoding. The latter can only be accomplished by using appropriate TPs and for this reason the process is repeated for the whole TP set.

Once all possible EEPs have been produced, one must be selected. It should be noted that not every TP will create an EEP and not all EEPs will be distinct. The selection will be done using the analogue weight of each EEP which is defined as:

$$aw_{EEP} = \sum_{i=0}^{n-1} a_i \times eep_i \tag{2.9}$$

where a_i is the analogue value of the $i-th$ bit position of the EEP.

The EEP with the smallest analogue weight will be modulo-2 added to the received word and the resulting sequence will be accepted as the corrected word. The EEP of minimum analogue weight is selected because equation 2.9 determines the pattern which has inverted the least confident bits. If all errors have been corrected then the EEP should equal the EP.

2.5.3 Test pattern set generation

It should be clear that the generation of the TP set is a very important task. If this is correctly generated then the total number of errors in a code word may be reduced, otherwise it will be increased. For this reason, the introduction of channel measurements may not *guarantee* anymore the correction of a minimum number of t received errors. However, if the total number of errors is above t, the decoder may have the ability to correct them. The three algorithms for selecting the TPs as suggested by Chase are the following:

1. The first algorithm (Chase 1) examines all possible TPs. Therefore, for an n-bit code with a minimum distance between code words of d_{min}, a total of $\binom{n}{\lfloor \frac{d_{min}}{2} \rfloor}$ possible combinations exist. For other than very small values of n this is impractical to implement in hardware or indeed to simulate down to the RBERs of interest. A considerable reduction in complexity is obtained if the test patterns that produce identical error patterns are ignored, but even with this improvement, the algorithm is not very efficient.

2. The second algorithm (Chase 2) considers only the set of error patterns containing $N_{Chase} = \lfloor \frac{d_{min}}{2} \rfloor$ lowest channel measurements (i.e. the bits with the highest probability of error). The test patterns generated in this case are those where any combination of inverted positions is allowed within the set of lowest measurements. Thus $2^{\lfloor \frac{d_{min}}{2} \rfloor}$ TPs are examined.

3. Chase 3 examines $N_{Chase} = (\lfloor \frac{d_{min}}{2} \rfloor + 1)$ possible patterns. Once more the inverted positions are assigned to the i positions of lowest confidence. If d_{min} is even, i takes the values $i = (0, 1, 3, \ldots d-1)$. If d_{min} is odd then $i = (0, 2, 4, \ldots d-1)$. This algorithm gives best results for codes with large values of d_{min}.

   Chase does not give the reason for differentiating between odd and even values of d_{min} and work done on the simulator indicates that it is unnecessary. Specifically, the performance of a code decoded using Chase 3 with $i = (0, 2, 4, \ldots d-1)$ never exceeds the performance of the same code using Chase 3 but with $i = (0, 1, 3, 5, \ldots d-1)$.

2.5.4 Performance comparison of the Chase algorithms

Each of the three Chase algorithms utilises a different sized set of TPs for decoding. Clearly, if the TPs are sensibly chosen, the larger the set of TPs the more decoding power the code will have. However a large set of TPs will require more time to decode since a decoding operation for each TP is required.

Clearly, for large n (even for a relatively small d_{min}) algorithm 1 is too complex to employ in practice, since it will examine $\binom{n}{\lfloor \frac{d}{2} \rfloor}$ possible error patterns. The second Chase algorithm will examine $2^{(\lfloor \frac{d}{2} \rfloor)}$ possible error patterns. Finally, using the third Chase algorithm $(\lfloor \frac{d}{2} \rfloor + 1)$ possible test error patterns must be examined.

The above figures indicate that the complexity of Chase 2 and 3 depends on the minimum distance d only and not on the code word length. Therefore they offer clear benefits whenever long code words are used. These results can be seen in table 2.2 where the total number of TPs needed for each Chase algorithm are shown for various values of d_{min}, for an $n = 63$ BCH code.

t	Chase 1	Chase 2	Chase 3
1	63	2	2
2	1953	4	3
3	39711	8	4
4	595665	16	5
5	7028847	32	6

Table 2.2: Number of TPs for a $n = 63$ BCH code with various values of d_{min} decoded using the Chase algorithms.

In terms of decoding power, Chase 1 will provide results which will closely approach the *maximum likelihood* (ML) limit. The latter is the best possible decoding result soft decision decoding can offer. Simulation results indicate that Chase 2 performs reasonably well when compared to Chase 1, particularly at the low error rates of importance for this study. A significant reduction in the number of TPs is effected while an acceptable decoding performance is maintained. Chase 3 involves a loss in performance compared to the second algorithm due to the reduced number of TPs.

A disadvantage of all 3 algorithms is the fact that the complete set of TPs needs to be generated and used before one can be selected. This can prove to be too time consuming, especially for high bit rate operations.

2.6 Summary

In this chapter, the basic definitions and concepts of error control coding (which included both hard and soft-decision decoding algorithms) were introduced. The Chase SDD algorithms were then described in detail and a critical assessment of their performance was made. Thus the second Chase algorithm was found to be the most promising choice for implementation in practical high rate systems. This is because it offers a balanced solution in terms of decoding power and simplicity. The latter is very important both because it allows improvements to be made and also because of the potential application on high bit rate systems.

Having introduced some key ideas relating to error correcting codes we now turn our attention in the following chapter to line coding. We will begin by reviewing the basic principles and then progress to the consideration of some novel low-complexity codes suitable for high bit rate applications.

Bibliography

[1] S. Lin, D. Costello: "Error Control Coding", Prentice Hall, 1983.

[2] C. Shannon: "The Mathematical Theory of Communication", *Bell Systems Technology Journal*, 1948.

[3] R.W. Hamming: "Error detecting and Correcting Codes", *Bell Systems Technology Journal*, Vol. 29, pp. 147-160, April 1950.

[4] R. Bose, D. Ray-Chaudhuri: "On a Class of Error Correcting Binary Group Codes", *Information and Control* , Vol 3, March 1960.

[5] The Open University: "Codes", *TM 361 14*, 1982.

[6] G. D. Forney,"Generalised Minimum Distance Decoding", *IEEE Transactions on Information Theory*, Vol IT-12, pp. 125-131, April 1966.

[7] D. Chase: "A Class of Algorithms for Decoding Block Codes With Channel Measurement Information", *IEEE Transactions On Information Theory*, Vol IT-18, pp. 167-170, January 1972.

[8] O. Olanyian: "Implementable Soft-Decision Decoding Schemes", *International Journal Of Electronics*, Vol 66, 1989.

[9] J. Eiguren, I. Dumer, P. Farrell: "Split Syndrome Soft-Decision Decoding For Block Codes", *Coding and Cryptography Conference*, December 1993.

[10] J. Wolf: "Efficient Maximum Likelihood Decoding Of Linear Block Codes Using a Trellis", *IEEE Transactions On Information Theory*, Vol 24, January 1978, pp. 76-80.

[11] W. H. Thesling, F. Xiong: "Pragmatic Approach to Soft decision Decoding of Linear Block Codes", *IEE Proceedings on Communications*, Vol 142, No. 1, pp. 40-47, February 1995.

Chapter 3

Line Coding

3.1 Introduction

In the previous chapter, the basic concepts of *forward error correction* (FEC) were presented. *Soft decision decoding* (SDD) was then introduced which offered greater *error correcting* (EC) capabilities but at the expense of increased complexity. A number of existing SDD algorithms were presented and critically evaluated.

In this chapter, *line coding* (LC) will be introduced. Specifically, the need for LC will be presented together with some basic definitions and concepts. Those will be followed by the presentation of a new family of line codes which are very simple to implement, yet offer reasonably tight bounds for both the maximum runlength and the disparity. Since they are single added bit codes, the overall rate is not significantly reduced which makes them well suited to high bit rate applications.

The chapter begins with a presentation of the basics of line coding, a simple line code being used as an example to illustrate the main concepts. The $nB1X$

family of line codes is then introduced followed by a detailed examination of its performance. We conclude by presenting two methods for determining the power spectrum of a line code.

3.2 Line Coding

The principal function of a line code is to match the transmission signal to the communication channel characteristics. Therefore, line coding is introduced to overcome the physical impairments of the channel used. This is usually achieved by limiting or eliminating the low frequency content of a signal and/or by reducing the maximum runlength. Both of these factors require added redundancy.

The low frequency content must be restricted since most channels cannot achieve sufficient signal to noise ratio in that area of the spectrum [1]. For example, magnetic recorders do not respond well to low frequency signals, so that a signal that contains such components will have an increased number of errors. These can be corrected by using FEC. However, a simpler technique is to code the data so that distortion is minimised. This can be achieved by line coding [2].

In addition, for practical reasons, many channels are AC coupled. This implies that the DC and low-frequency content of a signal must be suppressed [3, 4]. This can be achieved by bounding the *digital sum variation* (DSV), which is the difference between the largest and smallest values of the *running digital sum* (RDS) [5], defined as

$$RDS = \textstyle\sum_{i=1}^{k} d_i$$

where d_i is defined as the disparity (i.e. the difference in ones and zeros) of each one of the k code words.

Additionally, it is common for the receiver to be synchronised by extracting timing information from the received waveform. It is thus necessary to have an adequate number of transitions within a given amount of time. This is achieved by limiting the maximum runlength (RL_{max}) of a bit stream. The latter is defined as the maximum number of consecutive identical bits in a code word.

Recent experimental data suggest that significant gains in error performance can be achieved in some systems by limiting the maximum runlength of a transmitted word. Specifically, in a 232km optical fibre system, $4dB$ of equivalent coding gain was present simply by limiting the maximum runlength from $RL_{max} = 31$ to $RL_{max} = 7$ [6].

Generally speaking, a line code will map a block of k symbols which have p levels into a block of m symbols with r levels. In most cases the use of the line code will introduce some redundancy, so that $r^m \geq p^k$. The rate R of a code is defined as

$$R = \frac{k\ log_2 p}{m\ log_2 r}.$$

A disadvantage of most line codes is known as 'error extension'. This is a phenomenon whereby errors along the channel give rise to a larger number of errors in the decoder. An example of error extension will be presented in the following sections with the introduction of the bi-modal codes.

The following evaluation factors can be used to compare various line coding schemes:

1. Power Spectrum: This usually is one of the most important factors, indicating the extent of any DC or low frequency contents. The low frequency region of the power spectrum is related to the running digital sum bounds.

2. Synchronisation: In most applications, the receiver utilises signal transitions to synchronise itself with the transmitter so that optimal sampling is effected. Therefore, a code which offers a high number of transitions is preferred. Synchronisation is determined by the maximum runlength.

3. Signal degradation: Very frequent transitions can in some instances (e.g. bandwidth limited channel) cause *inter-symbol interference* (ISI) between adjacent symbols.

4. Reduction in the overall bit rate R, due to the introduction of the line code.

5. Complexity of implementation and cost.

Various existing and new line codes will be presented in this chapter with the areas mentioned above forming a basis for assessing their suitability for use.

3.2.1 A simple line code

One of the simplest line codes is the *alternate mark inversion* (AMI) code. Such a code is not very useful for optical transmission systems since the three transition levels required are not suited to on-off keying techniques. However, AMI will be used to demonstrate, by way of example, the basic concepts behind line coding. The AMI code will encode a binary zero as a ternary 0 and a binary one as a ± 1

alternately. Thus long runs of ones will be avoided while long runs of zeros can still exist. The encoding dictionary for this code is shown in table 3.1.

Information Word	Code word	
	RDS=0	RDS=1
0	0	0
1	+1	-1

Table 3.1: Encoding dictionary for the AMI code.

Figure 3.1 presents the state diagram of the AMI code. A state diagram is a convenient way of showing the possible states a code can have (represented by a circle) and the probability of transferring from one state to another. Figure 3.1 indicates that in the AMI code there is an equal probability of transferring between different states or remaining in the same state.

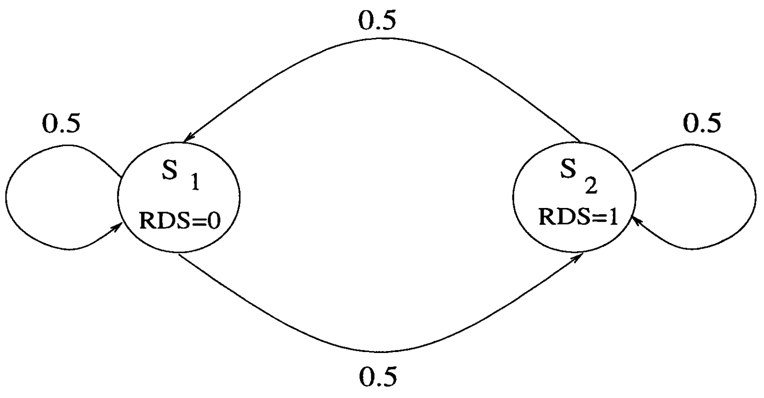


Figure 3.1: AMI state diagram.

Finally, the power spectrum of a line code is very important. As has been mentioned before, one of the reasons behind the use of a line code is the suppression of the low frequency content and the elimination of the DC component. The power spectrum presents the power spectral density versus frequency for a given code. In our case this was achieved by using the Cariolaro and Tronca algorithm [7]. The power spectrum of the AMI code is illustrated in figure 3.2, where it is seen that indeed the lower frequencies are suppressed and the DC content is zero.

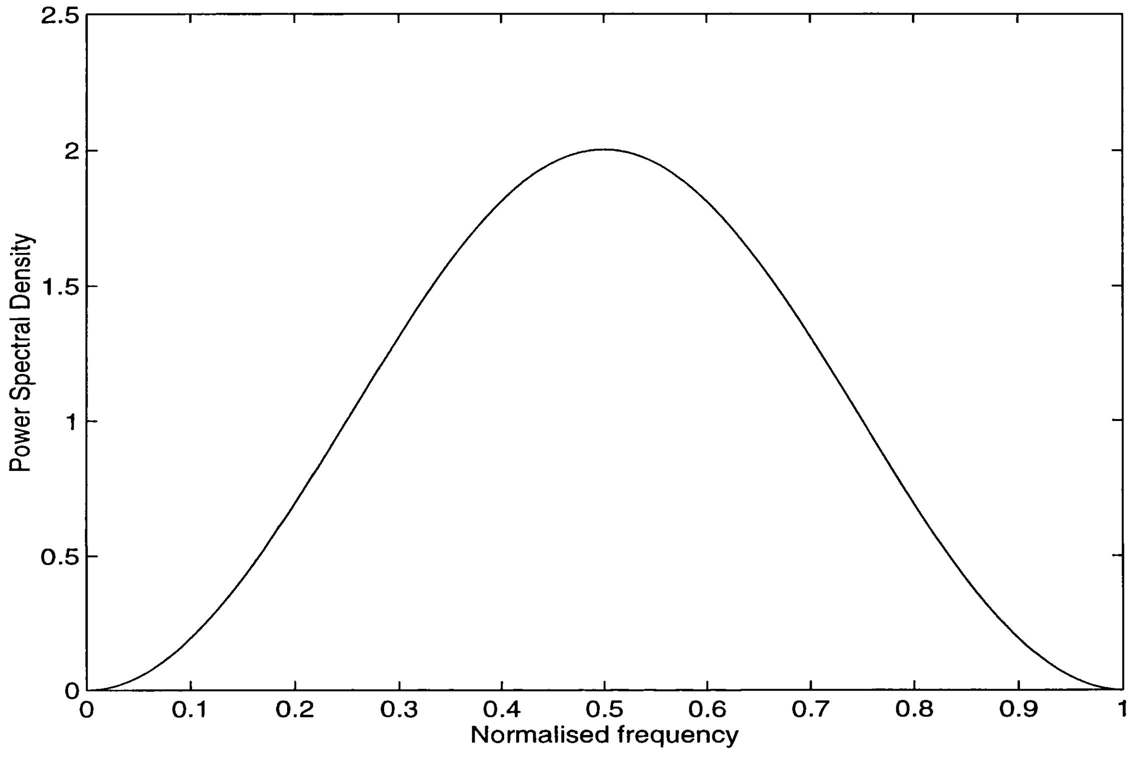


Figure 3.2: AMI power spectrum.

In the following sections a unified family of line codes, termed $nB1X$, is examined. This is simple to implement and can offer very good line coding characteristics.

3.3 The $nB1X$ Class of Line Codes

A binary block line code can be considered as a member of the $nBmB$ family. Such codes have n binary information bits which are encoded to m binary code word bits. A widely used subset of these codes are those where $m = (n+1)$, i.e. a single bit has been added to every n information bit block. The added bit can be used to achieve either error detection or line coding properties. The $nB1X$ line codes introduced here are a subclass of this family [4, 8, 10]. These are very simple to implement and relatively effective but, like all line codes, they reduce the overall code rate, which becomes equal to $R = (\frac{n}{m}) = (\frac{n}{n+1})$, and under certain circumstances, may introduce error extension.

The major advantages of the $nB1X$ codes can be summarised as follows:

- They can provide tight runlength and disparity bounds.
- The reduction in rate (especially for large values of n) is minimal.
- They can be very simple to encode and decode.
- They use state-independent decoding. This means that the decoder does not utilise the RDS information so error propagation between line code words is impossible.
- Finally, most added bit line codes can be converted into *error correcting line codes* (ECLCs).

This chapter introduces a number of novel single added bit codes. These are combined with existing line codes, to form the $nB1X$ family. Each member of this family is presented in detail, together with any possible improvements, in the following sections.

3.3.1 The $nB1P$, $nB1C$ and $nB1I$ line codes

The $nB1P$ code is a single bit insertion code where the added bit is a parity bit. It offers odd number error detection by forcing the code word to have either odd or even parity. If an odd number of errors is present then this will be detected and the word can be ignored or a re-transmission requested.

The $nB1P$ code can not place bounds on disparity but it can place runlength constraints if odd parity is used when $n = odd$. In such a case the maximum runlength will be equal to $2n$. If $n = even$ and even (odd) parity is used then there are no runlength bounds since the number of zeros (ones) is unbounded, while the number of ones (zeros) is limited to $2n$. An example $4B1P$ encoded word, using odd parity, is shown in figure 3.3 (a).

The $nB1C$ code [11] inserts an extra bit at the end of each word, the value of which is the inverse (complement) of the value of the previous bit. The runlength is thus limited to a maximum value of $(n+1)$. Figure 3.3 (b) uses a $4B1C$ encoded word as an example.

It should be noted that if $n = 1$ on the $nB1C$ code, then each bit is followed by its inverse, i.e. a 'Manchester' dipulse code has been derived. This has zero DC content, suppressed low frequencies and is very simple to implement. In addition, it is suitable for use as an ECLC and is thus presented in more detail in chapter 6.

Finally, the $nB1I$ [12] is a bi-modal code where the added bit indicates whether the word is inverted or not. Inversion is effected if both the code word disparity (d_{cw}) and the RDS have the same sign, i.e. if $d_{cw} \times RDS > 0$. This ensures that RDS bounds are placed. The initial value of the flag bit is set to zero. If the word

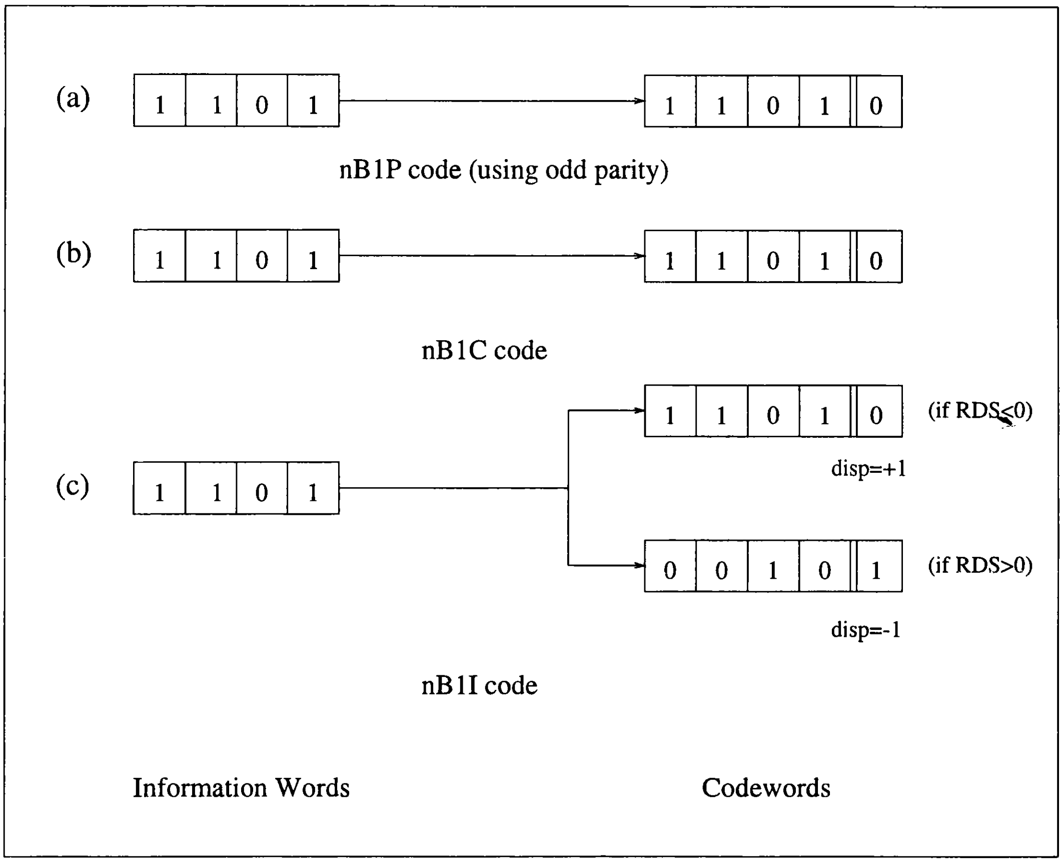


Figure 3.3: Examples of the P, C and I codes, for $n = 4$.

is subsequently inverted then the value of the flag will become equal to one and this will indicate to the receiver that inversion has taken place. This code offers the best line coding performance of all three codes presented in this section, in terms of disparity reduction. Figure 3.3 (c) presents a $4B1I$ encoded word as an example.

The RDS of this code can be between the values of $(-n-1)$ and $(n-1)$ if n is odd and between $(-n-1)$ and (n) if n is even. The maximum runlength for ones is $\frac{(5n+3)}{2}$ and for zeros $\frac{(5n+1)}{2}$, if n is odd. The equivalent numbers if n is even are both $\frac{(5n+2)}{2}$. A disadvantage of the $nB1I$ code is the fact that error extension is present. If, for example, an error occurs on the flag bit then the information bits will be incorrectly decoded.

3.3.2 Improving the $nB1I$ code

The state diagram of the $nB1I$ code was used to determine the word combination that generated the worst runlength of ones, if n is odd. This was found to occur when zero disparity words existed and since these can be inverted without affecting the RDS, a modified algorithm was created, termed $nB1I$ Improved (I). The latter will invert zero disparity words if the first $\frac{(n+1)}{2}$ bits of the current zero disparity word equal the last bit of the previous one. This technique slightly reduces the maximum runlengths of ones down to $\frac{(5n+1)}{2}$ if n is odd. The maximum runlength of zeros and the DSV were not affected. By placing an extra constraint a reduction in the maximum runlength without reducing the rate has been achieved.

3.3.3 The $nB1D$ and $nB1R$ line codes

The $nB1D$ code consists of an n bit (where n is odd) information word coupled to a single bit flag. The value of the flag depends on the disparity of the information word and aims to reduce it. As an example consider that the all-zero information word is to be transmitted in continuous blocks of three bits each. If the $nB1D$ code is to be used, a flag bit with a value of 1 will be added in every block. Thus a four bit block will be generated which will have a reduced disparity compared to the uncoded version. The flag bit primarily aims to reduce the average DSV value but since it consists of a single bit, it will never manage to place bounds on the RDS. Figure 3.4 (a) presents an example of a $5B1D$ code.

It should be also noted that while this code primarily aims to reduce the average DSV this will also have the effect of reducing the maximum possible runlength, see Table 3.4.

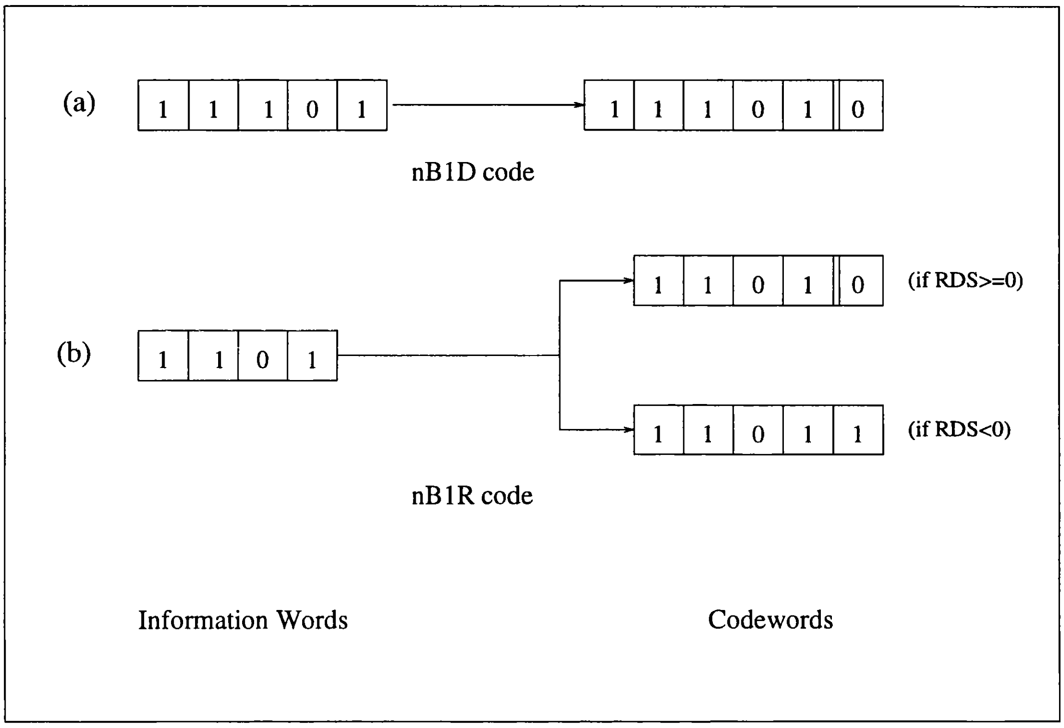


Figure 3.4: Examples of the D and R codes.

The $nB1R$ code aims to reduce the overall RDS value by using a flag bit of the opposite sign, i.e. if $RDS > 0$ then the flag bit is equal to zero, otherwise it equals one. The advantage of this coding scheme is that the *average* DSV is more tightly bound than before. However, no bounds are placed on either the runlength or DSV. Figure 3.4 (b) presents an example of a $4B1R$ code.

3.3.4 The $nB1DR$ line code

The $nB1DR$ is the final code of this class [13]. Similarly to the $nB1I$ algorithm presented previously, it also is a bi-modal code but uses the flag bit to either reduce the disparity of the code word or to indicate that an inversion of the information bits has taken place. The encoder flow diagram for this code is shown in figure 3.5.

Initially the disparity of the information block is calculated and the extra bit is appended at the end. The value of the latter aims to reduce the overall disparity of the code word, in a similar manner to the $nB1D$ code. If n is even it is possible that the code word disparity will be zero. In such a case the flag will be set to zero as well.

The disparity of the whole word (including the flag bit) termed d_{cw} in figure 3.5 is then calculated. If $(d_{cw} \times RDS) > 0$ (i.e. the code word disparity has the same sign as the RDS) then the information bits are inverted before transmission, otherwise the word is simply transmitted. The value of the RDS is then updated and the process is repeated.

This concludes the algorithm if n is odd. However if n is even, then an extra step must be added which compensates for possible zero disparity words. The information bit disparity once inversion has taken place is re-evaluated and if it is found to be equal to zero then the flag is inverted as well.

At the receiver the disparity of the information word is re-calculated. If there is a 'rule violation' between the disparity of the information bits and the flag, the information bits are inverted before being accepted. A 'rule violation' occurs when either both the disparity of the information bits and the flag bit have the same sign, or when the information bit disparity is 0 and the flag is 1, which only occurs if n is even.

As an example, consider that an all-zero information word is to be transmitted

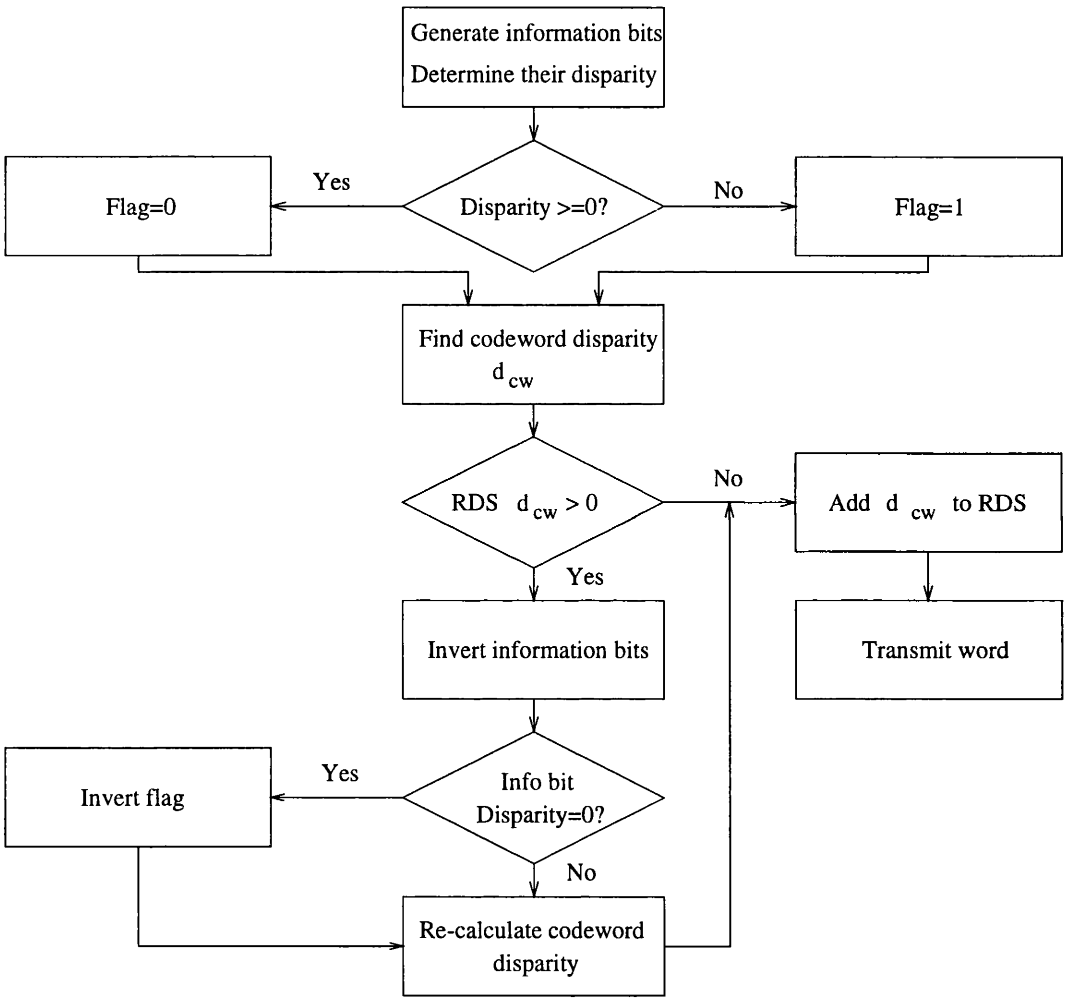


Figure 3.5: Flow diagram of the $nB1DR$ code.

in continuous blocks of three bits each. In such a case, the flag for the initial block will be determined according to the disparity of the information bits, and in this case will be equal to 1. The overall disparity of the resultant code word will be -2 and since the initial value of the RDS is zero, the word is transmitted. At the receiver, if no errors have occurred, the disparity of the first three bits is equal to -3. The flag has a bit disparity of $+1$ and for this reason the word is accepted.

The second block of three bits is initially encoded exactly as before, i.e. the code word 0001 is generated. However, because both the disparity and the *RDS* are equal to -2 the information bits are inverted. The (1111) code word is therefore transmitted, which produces an RDS value of $(-2+4) = +2$. At the receiver, the disparity of the initial three bits is $+3$ and the flag bit disparity $+1$. Thus a 'rule violation' exists and the information bits are inverted. The whole process is then repeated for the third block of data which will be encoded in exactly the same way as the first one. The RDS will thus become equal to zero once more.

The code word table for the $3B1DR$ code is presented in table 3.2. Each code word has two alternative representations depending on the state of the RDS at the time of transmission. The figures in brackets indicate the disparity of each word.

3.3.5 Improving the $nB1DR$ code

If n is even, the $nB1DR$ code can be further improved in terms of runlength performance. This is achieved if the word is not inverted when the RDS has the next closest to zero allowable value and provided that the disparity bounds are not exceeded. In such a case the disparity will remain within the bounds while the

Information Word	Code word	
	Initial	Inverted
000	000 1 (-2)	111 1 (+4)
001	001 1 (0)	110 1 (+2)
010	010 1 (0)	101 1 (+2)
011	011 0 (0)	100 0 (-2)
100	100 1 (0)	011 1 (+2)
101	101 0 (0)	010 0 (-2)
110	110 0 (0)	001 0 (-2)
111	111 0 (+2)	000 0 (-4)

Table 3.2: State table for the $3B1DR$ code. The resultant disparity of each code word is shown in the brackets.

maximum runlength will be reduced. This code is termed $nB1DR(I)$.

The explanation for the above procedure becomes clear if the worst case for the runlength is examined using the $4B1DR$ code as an example. The case for a maximum runlength of ones is investigated but this also applies if zeros were to be used.

The code words shown in the upper part of figure 3.6 represent a set of original words, i.e. words where no inversion has been effected. Words shown in the lower part of figure 3.6 represent inverted ones. The worst case for the runlength occurs if an all-one word (Word 3, lower) is preceded by a word with n ones (Word 2, lower) and followed by a word with $\frac{n}{2}$ ones (Word 4, upper). This situation can exist if a code word of disparity equal to +3 has initially occurred (Word 1, upper), together with an RDS of +1. The code word was therefore inverted causing the RDS to become equal to −4 (Word 1, lower). This allowed two further code word inversions to take place which resulted in the worst case situation.

If the $nB1DR(I)$ code had been used instead, the inversion at 'Word 1' would not have taken place. The RDS would have been equal to −5 which is within the bounds and therefore the maximum runlength would not have occurred. Using the $nB1DR(I)$ algorithm the maximum runlength is thus reduced down to $\frac{5n}{2}$, if n is even.

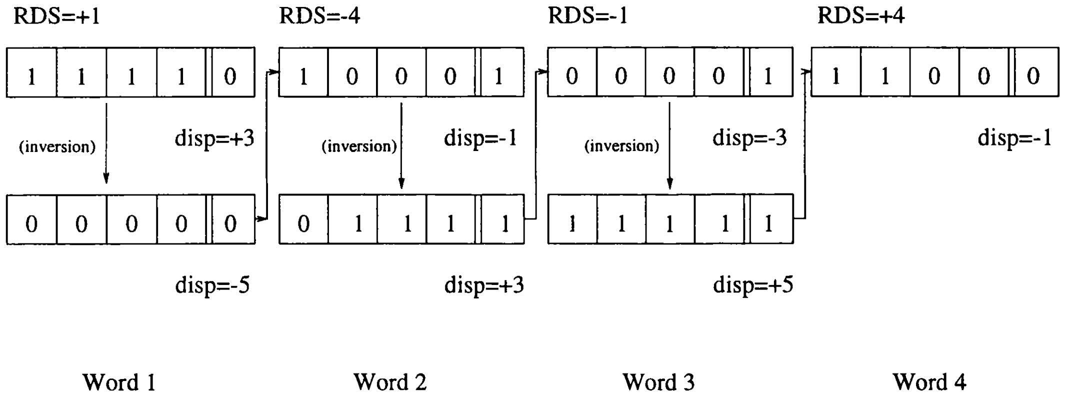


Figure 3.6: Worst case runlength for the $4B1DR$ code.

A similar analytical technique can be used to reduce the runlength, if n is odd. Once more, the worst case is analysed and a slightly more complicated algorithm is developed.

Specifically, the worst case occurs if a zero disparity word follows an all-zero or all-one word. If, therefore, the last and first bits of these two words are identical and provided that the *RDS* limits are not exceeded, the zero disparity word can be inverted so that the maximum runlength is reduced.

The maximum runlength is therefore reduced to $RL_{max} = \frac{5n}{2}$ for all values of n, at the expense of encoder simplicity. The RDS bounds and the decoder complexity are not affected.

3.4 Summary of Code Performance

Table 3.3 presents the RDS bounds for the $nB1X$ class of codes while table 3.4 indicates the maximum possible runlengths for the same codes. These were determined using the state diagrams of each code and verified using computer simulation.

From both the above tables, it was concluded that the $nB1I$ and $nB1DR$ are the most powerful codes. Between the two, the latter offers the best performance in terms of disparity and runlength. At the same time the encoding and decoding algorithm is kept at a reasonably low complexity level.

3.4.1 Decoding performance of the single added bit codes

The decoding performance of the non-bi-modal added bit line codes is exactly the same as that of the information bits. If, for example, a 'parent' error correcting code was used to encode the information bits, thus generating a simple *error correcting line code* (ECLC), then the decoding performance of the latter would equal that of the ECC. This is because the flag bit is only used to offer line coding properties and is discarded before decoding at the receiver.

The decoding performance of the bi-modal added bit codes is affected by error extension. The latter occurs when the number of errors at the decoders' output exceeds the number of errors that have occurred in the channel. Using the $nB1I$

	Running Digital Sum (RDS)	
Code Used	$n = odd$	$n = even$
$nB1P$	Unbounded	
$nB1C$	Unbounded	
$nB1I$	$(-n-1) \leq RDS \leq (n-1)$	$(-n-1) \leq RDS \leq n$
$nB1I$ (I)	$(-n-1) \leq RDS \leq (n-1)$	$(-n-1) \leq RDS \leq n$
$nB1D$	Unbounded	-
$nB1R$	Unbounded	
$nB1DR$	$(-n+1) \leq RDS \leq (n-1)$	$(-n) \leq RDS \leq n$
$nB1DRI$	$(-n+1) \leq RDS \leq (n-1)$	$(-n) \leq RDS \leq n$

Table 3.3: RDS bounds for the $nB1X$ class of codes.

	Maximum Runlength (RL_{max})		
Code Used	$n = odd$ (1/0)		$n = even$ (1/0)
$nB1P$	see text		
$nB1C$	$(n+1)$		
$nB1I$	$\frac{5n+3}{2}$	$\frac{5n+1}{2}$	$\frac{5n+2}{2}$
$nB1I$ (I)	$\frac{5n+1}{2}$		$\frac{5n+2}{2}$
$nB1D$	$\frac{3n+1}{2}$		-
$nB1R$	Unbounded		
$nB1DR$	$\frac{5n+1}{2}$, 12 if $n = 5$		$\frac{5n+2}{2}$
$nB1DRI$	$\frac{5n}{2}$		

Table 3.4: Maximum runlength characteristics of the $nB1X$ class of codes.

code as an example, if an error occurs at the flag bit then the whole word will be wrongly decoded, since that particular bit indicates whether inversion has taken place or not. Similar results exist for the $nB1DR$ code [14]. The decoding performance of both codes can be improved by using ECCs (combined with soft decision decoding, if required) to encode the information bits. This way, the effects of error extension are minimised, at the expense of simplicity and rate. Such ECLCs are presented in more detail in chapter 6.

3.5 Spectral Properties of Selected $nB1X$ codes

The spectral properties of selected $nB1X$ codes are indicated by the power spectra presented in figures 3.7 and 3.8. These were obtained by using the Cariolaro and Tronca algorithm [7].

Figure 3.7 presents the power spectrum of a $nB1D$ code with $n = 3$. Such a code will suppress the low frequency content, but since no bounds on disparity are placed, the DC content will not be equal to zero.

Figure 3.8 indicates that both the $nB1I$ and $nB1DR$ codes have a DC null and suppress the low frequency content, thus allowing AC coupling to take place. Since the $nB1DR$ code has a slightly tighter DSV bound, it offers a small improvement in the suppression of low frequencies, compared to the $nB1I$ code.

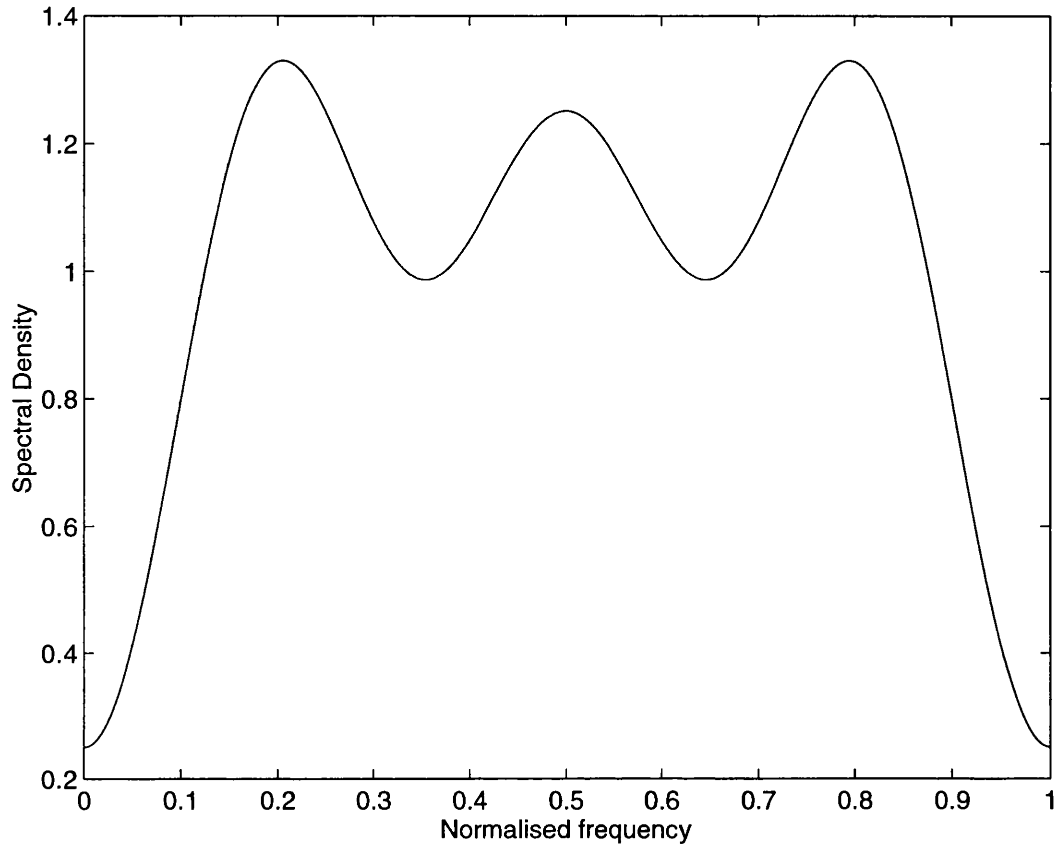


Figure 3.7: Power spectrum for the $nB1D$ code with $n = 3$.

3.6 Direct Calculation of the Power Spectrum

For the $nB1C$, $nB1D$ and Manchester codes, the spectral characteristics may be obtained directly using the formula:

$$S_{yy}(f) = R_{yy}(0) + 2\sum_{k=1}^{n} R_{yy}(k)cos(2\pi fk)$$

where $S_{yy}(f)$ is the power spectrum and $R_{yy}(k)$ is the auto-correlation function of the given code. This formula is derived by obtaining the Fourier transform ($\mathcal{F}$) of the auto-correlation function, i.e.

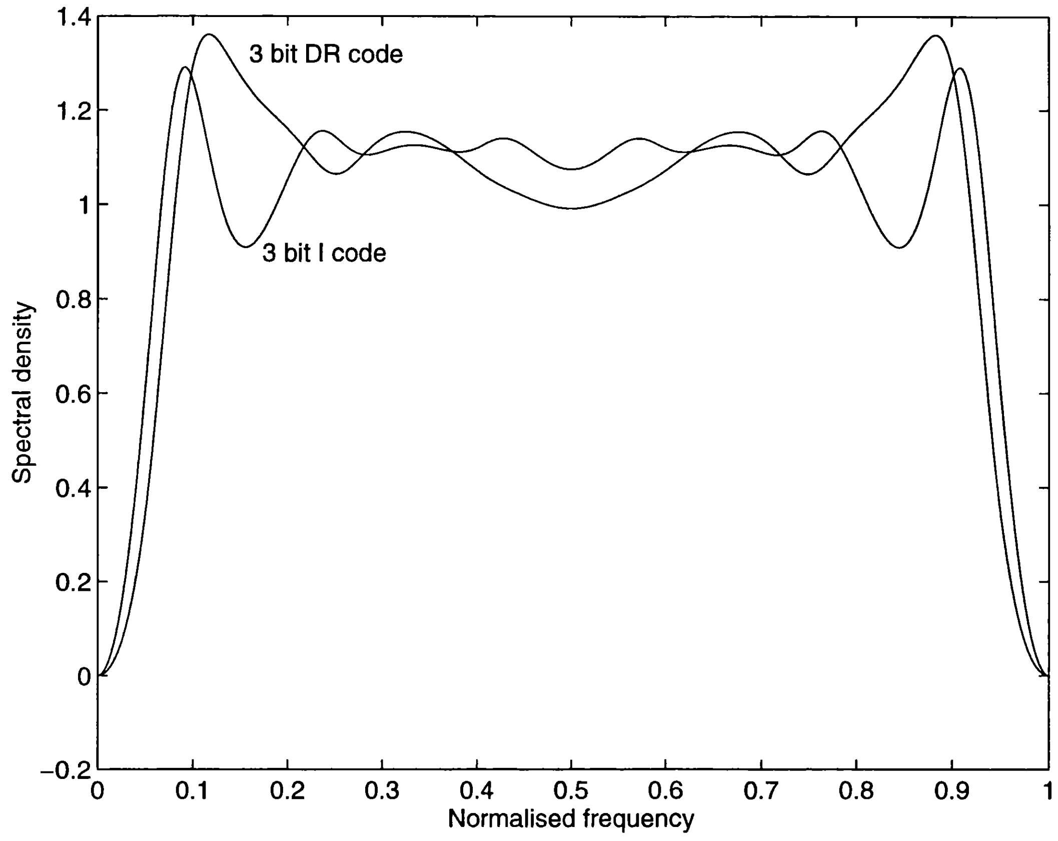


Figure 3.8: Power spectrum for the $nB1DR$ and $nB1I$ codes.

$$S_{yy}(f) = \mathcal{F}(R_{yy}(\tau)) = \mathcal{F}(R_{yy}(0)\delta(\tau) + \sum_{k=1}^{n}(\delta(\tau \pm k)R_{yy}(k)) \tag{3.1}$$

where k is the number of information bits (excluding the parity bit). We therefore need only to determine the values of $R_{yy}(j)$ for both $j = 0$ and $j \neq 0$ to obtain the power spectrum. Where this is possible, such an analysis has the advantage of being more general in nature because it can analytically accommodate varying code lengths and provide for relatively simple direct calculation for specific codes.

Two examples using this technique are presented: the first demonstrates the basic analytic technique with reference to the familiar single information bit Manchester code. The second example applies the same principles in a new context, to obtain the power spectrum of the $nB1D$ code. The technique develops from the approach outlined by O'Reilly [15] for the $nB1C$ code. We assume that the information bit positions are from 1 to n, the parity bit is the $n+1$ bit, and a logic 1 and 0 are presented as ± 1 respectively.

3.6.1 Analysing the Manchester code

To indicate the nature of the analytic approach let us first examine a familiar case and obtain the power spectrum for the Manchester code. This can be viewed as a very short $nB1C$ or $nB1P$ code with $n = 1$. We must first determine the auto-correlation function $R_{yy}(k)$ at $j = 0$ and $j \neq 0$, where $0 \leq j \leq n$. For $j = 0$ we obtain:

$$R_{yy}(0) = \frac{1}{(n+1)} \sum_{k=1}^{n+1} (P[X(k) = 1])(1)^2 + (P[X(k) = -1])(-1)^2 \Rightarrow$$

$$R_{yy}(0) = \frac{1}{(n+1)} \sum_{k=1}^{n+1} 1 = 1$$

$R_{yy}(j)$ is equal to:

$$R_{yy}(j) = \frac{1}{(n+1)} \sum_{k=1}^{n+1} \frac{1}{2}(P[X(k) = 1|X(k-j) = 1]) -$$

$$\frac{1}{2}(P[X(k) = -1|X(k-j) = 1]) - \frac{1}{2}(P[X(k) = 1|X(k-j) = -1]) +$$

$$\frac{1}{2}(P[X(k) = -1|X(k-j) = -1])$$

For a Manchester code all information bits will have equal probabilities of occurrence which will be equal to $\frac{1}{2}$. However, any bit k within a code word will be followed by its inverse. Therefore the above equation can now be expressed as follows:

$$R_{yy}(j) = \frac{1}{2}(\frac{1}{2} - \frac{1}{2} + 0 - 1) \Rightarrow R_{yy}(j) = -\frac{1}{2}$$

Substituting the above values for $Ryy(j)$ into equation 3.1 we obtain:

$$R_{yy}(\tau) = \delta(\tau) - \frac{1}{2}\delta(\tau \pm 1)$$

Applying the Fourier transform to $R_{yy}(\tau)$ the power spectrum $S_{yy}(f)$ is obtained, which for this example is:

$$S_{yy}(f) = 1 - cos(2\pi f)$$

The power spectrum of a Manchester code is the same shape as that of the AMI code shown in figure 3.2; a more detailed analysis can be found in appendix A.

3.6.2 Analysing the $nB1D$ code

A similar analysis to the one described above is performed for the $nB1D$ code, where n is odd. $R_{yy}(\tau)$ is thus determined as being equal to:

$$R_{yy}(\tau) = \delta(\tau) - \frac{1}{n+1}\frac{\begin{pmatrix} n-1 \\ \frac{n-1}{2} \end{pmatrix}}{2^{n-1}}\sum_{k=0}^{n}\delta(\tau \pm k)$$

If the Fourier transform is once more applied on $R_{yy}(\tau)$ the power spectrum $S_{yy}(f)$ is obtained:

$$S_{yy}(f) = 1 - \frac{1}{n+1} \frac{\begin{pmatrix} n-1 \\ \frac{n-1}{2} \end{pmatrix}}{2^{n-1}} \sum_{k=1}^{n} cos(2\pi f k)$$

The power spectrum of an $nB1D$ code for various values of n is presented in figure 3.9. Clearly, for $n = 3$ the waveform matches that of figure 3.7 (the latter obtained using the Cariolaro and Tronca algorithm). A detailed analysis for the $nB1D$ code can be found in appendix A.

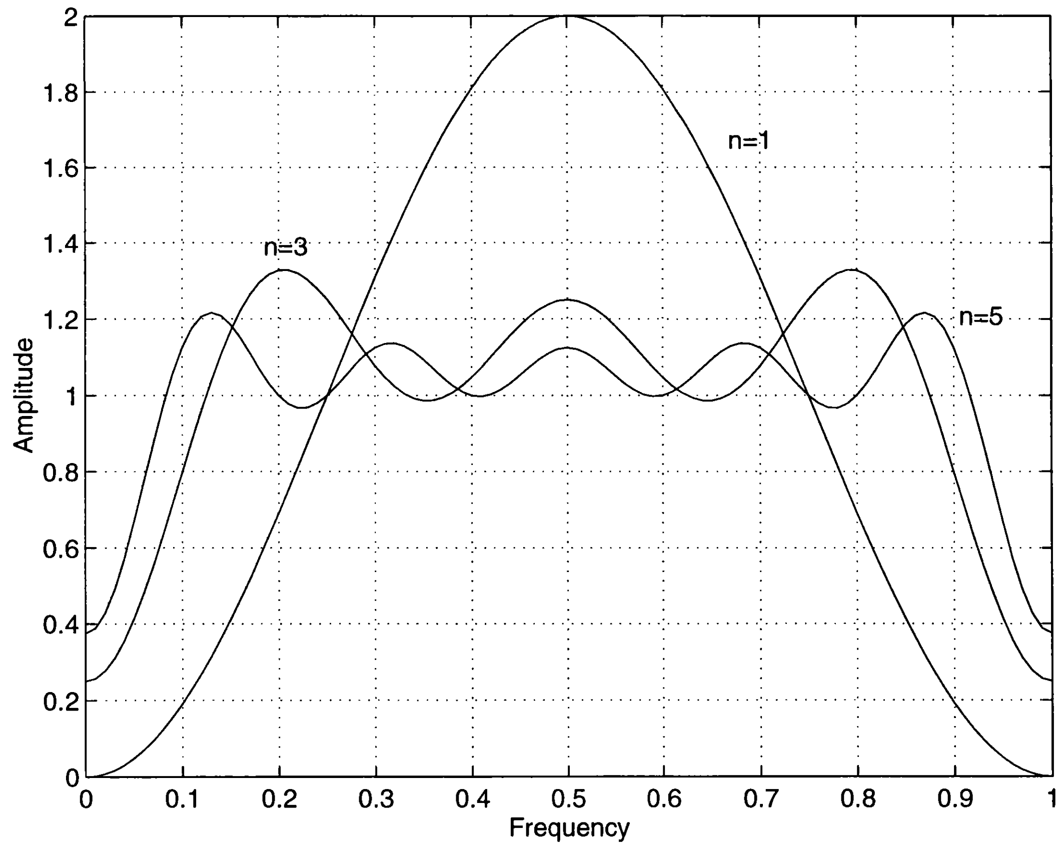


Figure 3.9: Power Spectrum of the $nB1D$ code with various values of n.

3.6.3 Analysing the $nB1C$ code

Finally, similar analysis obtained from [15] yields a power spectrum described by the following formula:

$$S_{yy}(f) = 1 - \frac{1}{n+1} cos(2\pi f)$$

This is presented in figure 3.10. If $n = 1$ then a Manchester code has been created, where each bit is followed by its inverse. This means that both the disparity and the runlength are bounded thus suppressing the low frequencies. As the values of n are progressively increased no disparity bounds exist, which is indicated by the DC content present.

3.7 Summary

In this chapter, the basic definitions and concepts of line coding were introduced. These were followed by an examination of a family of added bit codes, the $nB1X$ codes.

The $nB1X$ codes are all quite simple to implement and reduce the code rate by only a very small amount. However, they can place relatively tight bounds on both disparity and runlength. With small modifications they can be also used as error correcting line codes (see chapter 6).

Finally, the power spectrum for a number of line codes was also presented. This was obtained either by utilising the Cariolaro and Tronca algorithm or by performing a direct mathematical analysis, the latter being more general in nature.

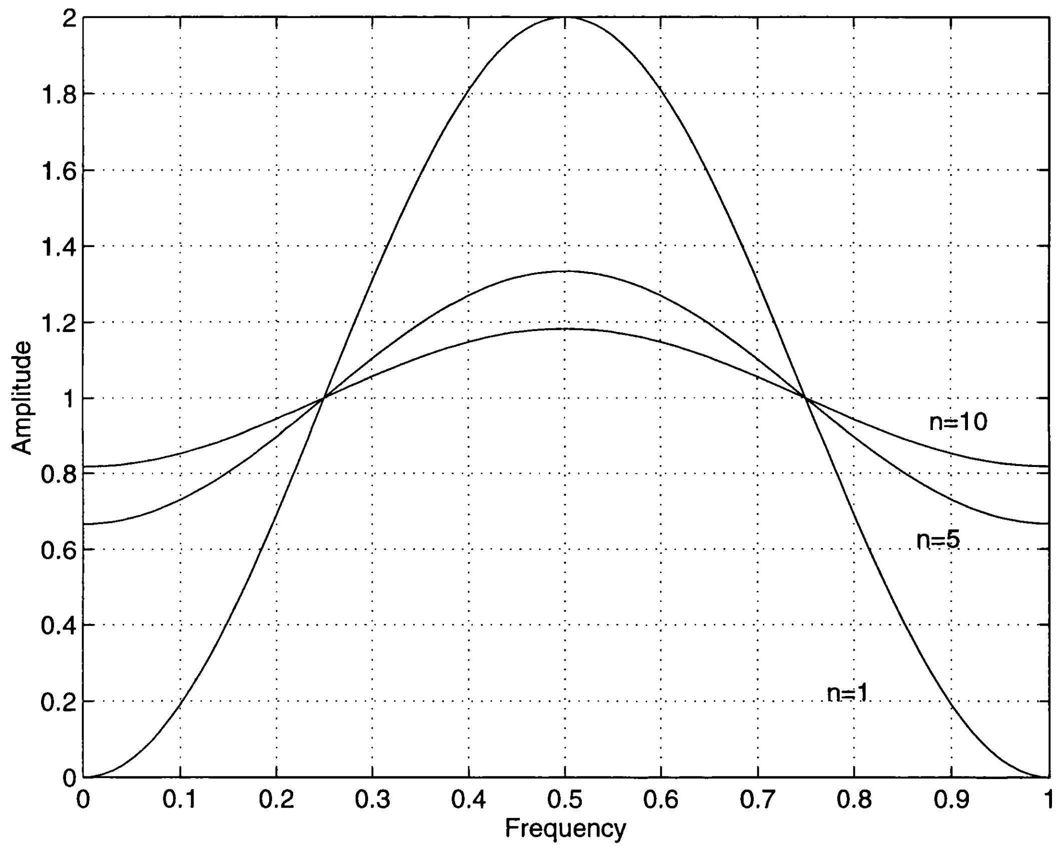


Figure 3.10: Power Spectrum of the $nB1C$ code with various values of n.

Bibliography

[1] K. A. S. Immink: "Properties and Constructions of Binary Channel Codes", *PhD Thesis, Technical School of Eindhoven*, 1985.

[2] K. A. S. Immink: "Runlength-Limited Sequences", *IEEE Proceedings, Vol 78, No 11*, pp 1745–1759, November 1990.

[3] A. S. Helberg, W. Clarke, H. Ferreira, A. Han Vinck: "A class of DC-Free Synchronisation Error Correcting Codes", *IEEE Transactions on Magnetics, Vol 29, No 6*, pp 1243–1250, November 1993.

[4] D. R. Smith: "Digital Transmission Principles", *Van Nostrand Reinhold, Second edition* 1993.

[5] R.W. Tkach, R.M. Derosier, F. Forghieri, A.H. Gnauck, A.M. Vengsarkar, D.W. Peckham, J.L. Zysking, J.W. Sulhoff and A.R. Chraplyvy, "Transmission of eight 20–Gb/s channels over 232 km of conventional single–mode fiber," *IEEE Photon. Technol. Lett.*, vol. 7, no. 11, pp. 1369–1371, Nov. 1995.

[6] G. L. Cariolaro, G. P. Tronca: "Spectra of Block Coded Digital Signals", *IEEE Transaction in Communications, COM-22*, No 10, pp 1555–1564, Oct. 1974.

[7] S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A Class of Low Complexity Line Codes", *Communications Research Symposium*, 29-30 July 1996, UCL, London, UK.

[8] S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A Class of Low Complexity Line Codes", *International Symposium on Information Theory, ISIT 97*, 29 June 1997, Ulm, Germany.

[9] N Yoshikai, K. Katagiri, T. Ito: "mB1C Code and its Performance in an Optical Communication System", *IEEE Transactions on Communications, Vol. Com-32*, February 1984.

[10] R. O. Carter: "Low Disparity Binary Coding System", *Electronics Letters*, Vol 1, No 3, May 1965.

[11] W. Krzymien: "Transmission Performance Analysis of a new Class of Line Codes for Optical Fiber Systems", *IEEE Transaction On Communications*, Vol 37, No 4, pp 402–404, Apr 1989.

[12] S. Fragiacomo, C. Matrakidis, J. O'Reilly: "Performance Aspects of a Class of Low Complexity Line Codes", *International Conference on Signal Processing, ICSPAT 97*, September 1997, San Diego, USA.

[13] J. O'Reilly: "Further Note on $mB1C$ Code Spectra ", *Electronics Letters*, Vol 21, No 20, pp 20, Sep 1985.

[15] J. O'Reilly: "Further Note on $mB1C$ Code Spectra ", *Electronics Letters*, Vol 21, No 20, Sep 1985.

Chapter 4

Simulation

4.1 Introduction

Almost all the theoretical work presented in this thesis was validated through computer simulation. In this chapter, the basic concepts behind modelling and simulating a communications system will be presented.

A model is a description of a system intended to predict its behaviour should certain events take place. Almost always a model will be more clearly defined than the actual system it describes because it will be simplified and idealised. However, all relevant behaviour and properties should represent reality as closely as possible and should be determinable in a practicable way, e.g. by numerical analysis. Simulation means driving the model with suitable inputs and observing the outputs [1].

A computer simulator is a very valuable tool in system investigations which complements advantageously conventional experiments for the following reasons:

1. It is very easy to modify since only software changes are required.
2. It can be very accurate.
3. Results are repeatable.
4. 'Rare' events which would normally require long time intervals to occur can be programmed to happen at user-defined intervals.

Two different types of simulation exist:

- Synchronous simulation, whereby the events of interest occur at regular intervals.
- Asynchronous simulation, whereby the events of interest may occur at any time.

In this thesis, three distinct code performance evaluation programs were developed and used to validate existing and proposed codes. The first is a simulation program used for evaluating the decoding performance of error correcting codes and error correcting line codes. Decoding performance is measured in terms of *residual bit error rate* (RBER) versus *signal to noise ratio* (SNR) and either *hard decision decoding* (HDD) or *soft-decision decoding* (SDD) can be used.

The model of a communications system utilising *error correcting codes* (ECCs) is initially presented and implemented using asynchronous simulation. This requires significant amounts of computational time if low RBERs are to be achieved. This problem is addressed by the introduction of two novel synchronous simulation acceleration techniques which reduce the time required without affecting the accuracy of the simulation.

A second simulation tool evaluates the line coding performance of various codes. This is a small and hence very fast simulation which will indicate runlength and disparity average and maximum values. Finally, the third program implements the Cariolaro and Tronca algorithm [2] to obtain the power spectrum versus frequency for a coded sequence. In the following sections the main concepts behind each program will be presented.

4.2 A Communications System Model

The modelling of a communications system used to determine its decoding performance is usually achieved by attempting to mimic the encoding and decoding operations which would be implemented in practice. Using an (n, k) BCH binary block code combined with HDD as an example, the main steps may be summarised as follows:

1. Generate a random k-bit block $(\overline{u})$ of binary data. This is termed the information or message word.

2. Encode the data block to generate an n-bit code word $(\overline{v})$. This will contain both the k information and $(n - k)$ parity bits. In addition, a second copy of the encoded word is made and stored for performance measurement purposes.

3. Generate the channel noise and add this to the transmitted code word. This will generate the received code word $(\overline{r})$.

4. Decode the received code word so that any errors can be located and corrected.

5. Compare the resultant decoded message with the copy of the original message.

6. Measure the decoded bit errors and calculate the RBER.

7. Repeat the whole procedure using further blocks of information bits until the estimated RBER stabilises within limits corresponding to the required accuracy of the simulation.

Figure 4.1 graphically represents all of the above operations.

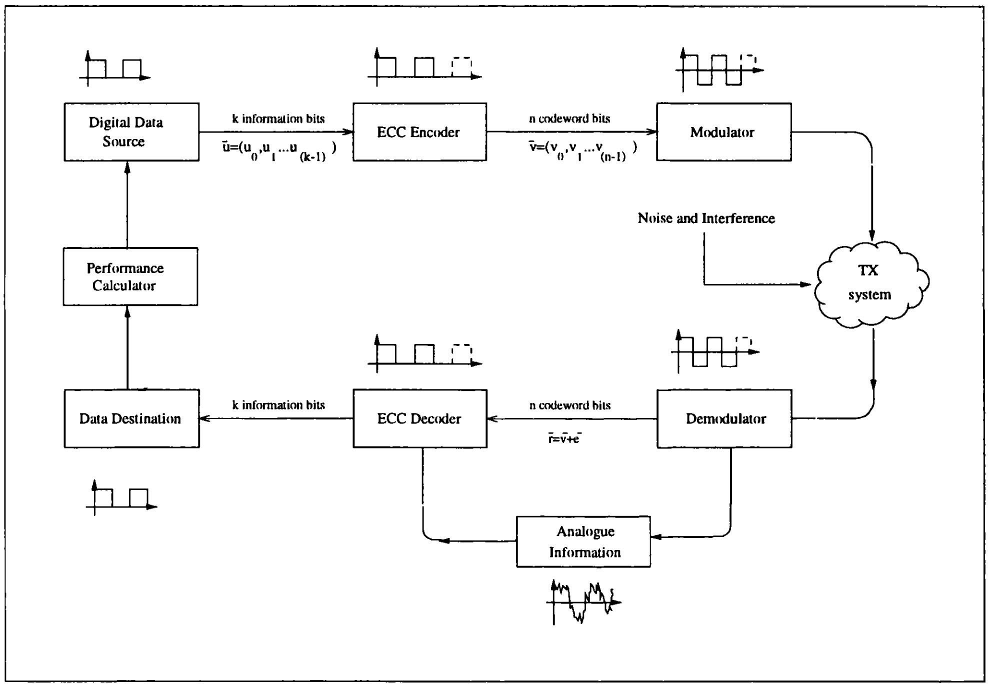


Figure 4.1: Communication system simulator outline.

The aim of this simulation is to determine the decoding power of a particular ECC, measured by the number of errors causing decoder failure. Clearly, the longer the simulation time, the more accurate the final result, due to the larger amount of residual errors. In this study, a number of about 1000 residual errors was deemed to provide sufficient accuracy. This figure can be justified by assuming that error events occur totally at random so that they can be approximately modelled as a Poisson process. The relative level of 'uncertainty' (defined as how close the simulation result is relative to the theoretical value) can be analytically expressed by the following equation:

$$relative\ uncertainty \simeq \frac{Standard\ Deviation}{mean}$$

However, for a Poisson distribution, both the variance and the mean have equal values, i.e. $\sigma^2 = \mu \Rightarrow \sigma = \sqrt{\mu}$. Substituting these values in the above equation we obtain:

$$relative\ uncertainty \simeq \frac{Standard\ Deviation}{mean} = \frac{\sigma}{\mu} \Rightarrow$$
$$\Rightarrow relative\ uncertainty = \frac{\sqrt{\mu}}{\mu} = \frac{1}{\sqrt{\mu}}$$

In our case μ is the number of residual bit errors; if this is set to be $\mu = 100$ then a 10% uncertainty level is achieved (i.e. the result will be within 10% of the theoretical value); If $\mu = 1000$ then the result will be within 3% of the theoretical value which is taken here as an acceptable level of accuracy.

The components needed to realise the model outlined in figure 4.1 are presented in more detail in the following section.

4.2.1 Model components

There are three basic components of a communications system simulator and these are as follows:

1. A data source together with a channel encoder.

2. The transmission channel which could include memory, attenuation, noise and interference effects, non-linear effects, etc.

3. A channel decoder capable of performing hard or soft decision decoding and a data sink.

Each component is now examined in more detail.

4.2.2 Data source and encoder

The first task of a communications channel model is to generate a data source. This is usually a random k bit sequence that should offer all possible bit permutations. The random number generator used at this stage is not very critical as far as the validity of the simulation is concerned. For this reason an $ANSI - C$ standard library function was used to generate pseudo-random binary information bits. This is a linear congruential generator and as such is quite fast; it can exhibit sequential correlations on successive calls but at a level which is unimportant in this context. Since this generator only creates the information bits true 'randomness' is not necessary. An added feature of pseudo-random data is the ability of the user to define the transmitted bits so that repeatability is possible. This way the effects of a particular parameter can be isolated and examined using the same set of information bits.

The next step is to encode the information bits using an ECC. As mentioned before, a (n, k) BCH code will be used and therefore its parameters must be defined. These include the following:

1. The desired code length (n).

2. The information vector length (k).

3. The error correcting capability of the code (t).

4. The power of the corresponding Galois Field .

5. The generator polynomial coefficients of the code.

Once all relevant data are entered, the parity bits are calculated (using the generator polynomial) and appended at the end of the information bits. The complete code word vector $\overline{v}$, is then used to produce two copies: The first one is used for 'transmission' while the second copy is stored so that it can be used for performance measurements.

Finally, the code word is passed through a modulator; this will further encode the bits by assigning a 'voltage' value to each 'logic' value. Thus, for example, a 'logic' 1 becomes a 'voltage' $+1$ and a 0 becomes a -1. The simulated voltage values are then passed through to the transmission channel model.

4.2.3 Channel simulation

The second task is to simulate the communications channel, or more specifically for this study, the noise content of the channel. The mechanics of the transmission medium are not of interest in this particular case so interference, attenuation, non-linear and other similar effects are not considered. In contrast, the performance of the system for a given signal-to-noise ratio expressed in decibels (dB) is of interest. Thus only the noise parameters for a given SNR need to be calculated.

These are described by using the *additive white Gaussian noise* (AWGN) model.

The term 'additive' indicates that the noise is signal independent and its amplitude is added to that of the variable representing the signal. The term 'Gaussian' indicates that the signal has a bell-shaped *probability density function* (PDF) which can be mathematically expressed as:

$$PDF = \frac{1}{\sqrt{(2\pi)}\sigma} e^{-\frac{(x-m)^2}{(2\sigma^2)}}$$

where x identifies possible values of the random variable, m is the mean of the function and σ is the standard deviation.

Finally the term 'white' means that the noise is uncorrelated and therefore the noise amplitudes are statistically independent with a zero mean, i.e. a memoryless channel is used.

The AWGN noise model is a reasonably accurate description of a realistic communications channel and makes calculations tractable. A good noise simulator must represent the AWGN model as accurately as possible. For this reason, the noise model requires high accuracy for the relevant parameters, e.g. PDF, 'randomness' etc.

This is achieved by implementing the generator in two distinct stages: First a uniformly distributed random number is generated, ranging in value from 0 to 1. This is then used by a second function to generate a suitable number (with a Gaussian PDF) representing the noise amplitude.

The inbuilt random number generator varies between different ANSI-C compilers. Its use would make the simulation program non-portable between various compilers, and for this reason the 'minimal standard ' random number generator of Park and Miller with Bays-Durham shuffle (as described in Knuth [3]) is used. This creates a number ranging in value from 0 to 1 with the shuffling procedure required to break up any sequential correlations.

This number is then used as a 'seed' to produce a normally distributed deviate with zero mean and unit variance (n_i). This is a Gaussian deviation function which produces the characteristic bell-like shape centered around zero. The algorithm used to achieve this is based on the Box-Muller method for generating random deviates. The variance is user controlled so that different SNRs can be accommodated. Since the definition of a SNR (expressed in dBs) is

$$SNR = 10\, log\, (\tfrac{average\ signal\ power}{average\ noise\ power})$$

the above equation can be solved for σ thus defining a suitable PDF.

In our case the average noise power is equal to σ^2 and assuming that the average signal power is equal to 1, solving for σ we get:

$$SNR = 10\, log\, (\tfrac{1}{\sigma^2}) \Rightarrow \tfrac{1}{\sigma} = \sqrt{10^{\frac{SNR}{10}}} \Rightarrow \sigma = \frac{1}{\sqrt{10^{\frac{SNR}{10}}}}$$

A low SNR corresponds to a large variance of the Gaussian distribution which results in high probability of error for a given signal amplitude. The opposite is true for a high SNR.

The above procedure thus produces a Gaussian distribution which represents the noise amplitude. This is then added to the modulated bit value so that the complete received bit r_i is created (the original bit with noise added to it, $r_i = v_i + n_i$).

The addition of noise to the transmitted bit implies that errors may be present at the receiver. On figure 4.2, the probability of errors occurring is indicated by the areas where the two functions overlap. This is because in that case the noise amplitude n_i is greater and of opposing sign to the encoded bit amplitude v_i. This will force the analogue value of the signal plus the noise to cross the decision threshold which will cause the decoder to erroneously decode the received bit.

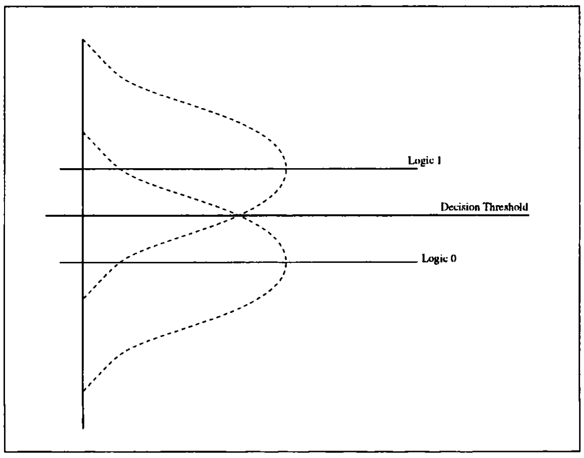


Figure 4.2: Noise function shape.

4.2.4 Hard decision decoder

Upto this point an information bit sequence has been generated, EC encoded and transmitted. This section presents the demodulation and error correcting algorithms required so that the original message is recovered.

The first component of a receiver is the Q level quantiser, where Q=2 for a binary HDD system. This will make a simple initial decision on the value of the received bits based on their analogue value with respect to the threshold. Since the modulator in our example has assigned analogue values of ±1 for a logic one and zero respectively, the output of the quantiser (rq_i) is decided according to the following rule:

$$rq_i = \begin{cases} +1 & if\, r_i \geq 0 \\ -1 & otherwise \end{cases}$$

The next component of a receiver is the demodulator which makes the translation from voltage values to logic values. In our case it simply decides whether the accepted signal ra_i is a logic 1 or 0 using the following simple rule:

$$ra_i = \begin{cases} 1 & if\, rq_i = +1 \\ 0 & if\, rq_i = -1 \end{cases}$$

After demodulation has taken place, the received word passes through the error decoder. The first task is for the syndrome S of the received code word to be calculated. If this is zero then it is assumed that no errors have occurred or if they have, they cannot be detected. In either case, the code word is accepted and the parity bits are discarded, thus producing the message. If the syndrome is not zero then it is assumed that errors have occurred.

Three choices are then available depending on the theoretical error correcting capability (t) of the code, all of which were included in the computer model:

1. If t is specified as being equal to one, only a single error can be corrected. The error position is equal to the power of the received syndrome element S_1

and the corresponding bit in the received code word is inverted. This is the simplest algorithm for correcting single errors.

2. If t is specified as being equal to two then up to two errors can be corrected. First it must be decided whether a single error is present or not. If the syndrome elements S_1 and S_3 are equal, then a single error is present and its position is equal to the power of S_1. If $S_1 \neq S_3$ then multiple errors have occurred and they are located at the positions given by the roots of the equation $x^2 + S_1x + S_3S_1^{-1} + S_1$. If no solution exists then more than two errors are present and the received code word can not be corrected. It must therefore be accepted as received. As before, this is the simplest algorithm for correcting double errors.

3. Finally if $t \geq 3$, then up to three errors can be corrected and in our case this is achieved using Berlekamp's iterative algorithm [4], as presented in Lin and Costello [5]. This algorithm is the most general in nature and can also be used for $t < 3$ BCH codes, at the expense of simplicity. It consists of three main steps:

   (a) Use the received data polynomial to compute the syndrome S.

   (b) Determine the error location polynomial $\sigma(X)$ from the syndrome elements.

   (c) Calculate the roots of $\sigma(X)$ which are the error location numbers and use them to correct the corresponding bit positions.

Once error correction is finished, the parity bits are discarded and the information bits are used to determine the RBER.

4.3 Simulation Verification and Validation

In the previous section, the basic communications system model with ECC was introduced. The accuracy of the decoding performance results must now be determined. In general, this is done using two checks, namely *verification* and *validation* [1].

Verification is the process whereby the correct implementation of the model is checked, which means that our model is examined to determine whether it does what is intended. The latter is achieved by carefully examining each stage of the code for correct operation. After debugging has taken place, a number of tests are performed such as stress testing (where inputs that will cause the model to fail are applied) and sensitivity testing (where only one parameter at a time changes).

The second test is validation. This is the process whereby every aspect of our simulation against a real system is compared, or in our case against specific numerical results. It has already been mentioned that due to the various assumptions made while designing our model, there will not be an *exact* matching of the two. However it must be ensured that any differences will not have practical significance. Validation is the most important test one can perform and for this reason a whole range of BCH codes were tested against known results. These include (but are not limited to) the following:

- (7,4) and (15,11) (15,7) (15,5) BCH codes. These are very useful codes since they offer different error correcting capabilities and any result can be manually checked against the simulation.

- $n = (127, 255, 511, 1023)$ single, double and triple error correcting codes. These codes were simulated because they could be potentially used in a realistic system.

All of the above codes were tested in conjunction with all possible configurations, namely HDD and *soft decision decoding* (SDD) (the latter using the algorithms introduced in the next chapter).

Figure 4.3 provides an example of validation by comparing the simulated decoding performance of a (127, 106) triple error correcting BCH code with the theoretical values. Signal to noise ratios between 1 and 6 dB, combined with hard decision decoding were used, until 1000 residual bit errors were obtained. The continuous line indicates the theoretical results and the dashed line the simulated results. It can be seen that a very close match exists between the two. These results were repeated for other code lengths.

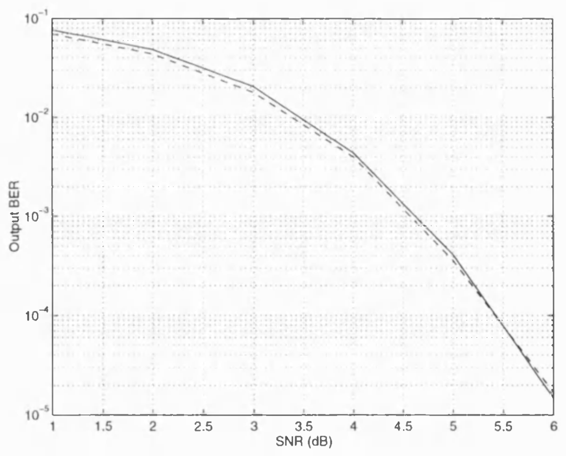


Figure 4.3: Validation of (127, 106) BCH code.

Higher SNRs were (at this stage) not simulated due to time limitations.

A final validation test was performed by comparing the decoding results obtained by the use of the three different HDDs. Specifically, the $t = 3$ algorithm was used to detect and correct single and double errors against the results produced by the $t = 1$ and $t = 2$ algorithms. These were found to be exactly the same, as expected.

4.4 The Need for Accelerated Simulation Techniques

It has been mentioned before that a number of residual errors must be present at the decoder's output to ensure that a statistically accurate result has been obtained. As the SNR progressively improves, the number of channel errors (and therefore the number of residual errors) progressively decreases. Thus, longer simulation runs are necessary before the required number of residual errors is present, a process that can be very time consuming.

Table 4.1 gives some indicative simulation times required for obtaining decoding performance results of a $(127, 106)$ BCH code while using a SUN ULTRA-SPARC platform. These are based on obtaining 1000 residual errors. The first column indicates the SNR (expressed in dBs), the second the number of words required to achieve the presented RBER, the third one the achieved RBER and the final column indicates the time taken. The times presented here are not very precise; they are only given as an indication of the exponential increase of computing time required with each 1 dB increment in SNR. The main reason for the inaccuracy of the times is the fact that the processor in a workstation usually divides its time so that other essential tasks can be performed as well. Therefore, at any given time, only a percentage of processor time is used for the actual simulation. This percentage can vary in time according to the demands from other users or the network, thus affecting the timing operation.

(127, 106) BCH HDD code			
SNR	Number of Simulated Words	RBER	Simulator Time Required
1	154	7.8×10^{-2}	55 sec
2	249	5.6×10^{-2}	59 sec
3	593	3.2×10^{-2}	1:31 min
4	2931	4.4×10^{-3}	3:43 min
5	32760	4.0×10^{-4}	25:50 min
6	929085	2.1×10^{-5}	7:14 hours

Table 4.1: Number of words and simulator time required for obtaining 1000 residual errors for varying SNRs.

The next section presents two simulation acceleration techniques which reduce the total number of code words that must be simulated, without affecting the accuracy of the produced results.

4.5 Simulation Acceleration Techniques

As explained in the previous section, the simulation efficiency decreases progressively with increasing SNR since more and more time is spent on encoding and decoding error-free blocks which offer no contribution to simulation accuracy.

Whilst state of the art workstations allow for effective simulations of low SNR channels, it is impracticable to obtain an accurate estimate of RBERs below 10^{-6} without excessive computational time.

Modern fiber-optic communications systems operate at very low residual bit error rates which would make simulation a very time consuming process. For this reason, the simulation program described above was used as a basis for the development of a new simulation acceleration technique [6, 7]. The latter is also used as a basis for developing a second acceleration technique [8], which further reduces the computational time required. Both techniques are described in the following sections.

4.6 First Acceleration Technique

As has been noted earlier, a conventional simulation will perform three distinct operations, i.e. encode, transmit and decode. Once all three are completed, the decoding performance of the code for various SNRs will be assessed.

In order to measure decoding performance a given number of residual bit errors (usually 1000 errors are sufficient) are required to be present at the output of the decoder. However, code words with a number of errors less than, or equal to t, do not contribute anything to the simulation since it is certain that they will be completely corrected. Nevertheless they will still have to go through the whole simulation process which can be time consuming. In order to enhance efficiency and reduce the total computational time required, a new technique was proposed. This uses knowledge of channel noise statistics to generate the occurrence times of bit errors without generating noise samples.

To demonstrate this point, a block code will be used where the occurrence times of possible bit errors are known. These are defined as the bit positions where the corresponding noise amplitude exceeds the modulation amplitude, which in our case is ± 1. If the corresponding bit has the same sign as the noise amplitude then no error will occur, otherwise an error will be present. Since the number of bits per word is known, it follows that the number of possible bit errors in each block is also known. Thus for a given error correcting capability t the number of words with less than t errors is known. These will therefore be calculated as having been successfully transmitted free of error. The remaining blocks which may add to the RBER will have to go through the whole encoding and decoding process.

However, if such a scheme is used, even though the encoding and decoding processes will remain the same as before, the way in which the noise is generated will have to be different compared to a conventional simulation. The reason for a different noise generation requirement is that the positions of possible bit errors have already been defined. Specifically, noise samples which cause bit errors and those which do not, are distinguished and generated separately according to their respective conditional probability distribution functions. To further illustrate this point, the effect of an additive white Gaussian noise channel on a transmitted signal is considered:

The first task is to determine the occurrence times of possible bit errors by generating samples of a random variable T_{EF}. This is defined as the number of error-free bits between two adjacent bit errors and has a probability function

$$P_i = \text{Prob}\{T_{EF} = i\} = p(1-p)^i \qquad i = 0, 1, 2, \ ...$$

where

$$p(X) = \frac{1}{\sqrt{2\pi}\sigma} \int_1^\infty \exp^{-x^2/2\sigma^2} dx$$

is the transition probability of the channel with σ^2 being the variance of the channel noise.

The next task is to generate the amplitude of the noise samples in code words with more than t possible errors in them. To generate a noise sample which causes a bit error, a random variable x_e is considered for which the conditional PDF when either a +1 or -1 was transmitted was shown to be:

$$f(x_e|+1) = \begin{cases} \frac{1}{\text{p}\sqrt{2\pi}\sigma} \exp^{-x_e^2/2\sigma^2} & x_e \le -1 \\ 0 & x_e > -1 \end{cases}$$

and

$$f(x_e|-1) = \begin{cases} \frac{1}{\text{p}\sqrt{2\pi}\sigma} \exp^{-x_e^2/2\sigma^2} & x_e \ge +1 \\ 0 & x_e < +1 \end{cases}$$

In contrast, to generate a noise sample which does not cause a bit error another random variable x_{ne} is considered with conditional PDF given by:

$$f(x_{ne}| + 1) = \begin{cases} \frac{1}{(1-p)\sqrt{2\pi}\sigma} \exp^{-x_{ne}^2/2\sigma^2} & x_{ne} > -1 \\ 0 & x_{ne} \leq -1 \end{cases}$$

and

$$f(x_{ne}| - 1) = \begin{cases} \frac{1}{(1-p)\sqrt{2\pi}\sigma} \exp^{-x_{ne}^2/2\sigma^2} & x_{ne} < +1 \\ 0 & x_{ne} \geq +1 \end{cases}$$

Thus, the distributions for both the time interval between adjacent error events and the amplitude distribution of the noise for these events have been calculated. Sufficient information is now available to simulate code performance without having to simulate each individual bit. The new simulation technique for block codes with AWGN channels may be summarised as follows:

1. Generate samples of the random variable T_{EF} which indicates the interval between adjacent error events.

2. Determine the code word blocks which have a number of errors exceeding the error correcting capability. Perform steps 3 to 5 only on these words.

3. Generate a random message for each word and conventionally encode.

4. Generate samples of x_e and x_{ne}. These represent the noise amplitudes for all the bits of interest. Add them to the corresponding code word bits.

5. Conventionally decode and measure the residual bit error rate.

As before, the whole process is repeated until the estimated residual error rate stabilises within specified limits.

The new simulation technique was incorporated in the previous simulator and the results were validated. Both the conventional and the accelerated simulations provided statistically identical results. The new technique was then tested to see if the expected improvements in time efficiency were obtained. These results are shown in Table 4.2, together with the conventional simulation times from Table 4.1, illustrating that a very significant improvement exists.

(127, 106) BCH HDD code		
SNR (dB)	Accelerated Simulator Time Required	Conventional Simulator Time Required
1	23 sec	55 sec
2	26 sec	59 sec
3	28 sec	1:31 min
4	32 sec	3:43 min
5	1:13 min	25:50 min
6	10:02 min	7:14:07 hours

Table 4.2: Simulator time required for obtaining 1000 residual errors at varying SNRs using the first acceleration algorithm, compared with conventional simulation results.

4.7 Second Acceleration Technique

The first acceleration technique reduced the computational time required by a significant amount. However, there are a number of cases where an even larger decrease of simulation time is required, e.g. optical systems with very low RBERs. For this reason a further acceleration technique was devised [6].

The main concept behind the previous algorithm was that words with a total possible number of errors below the error correcting capability t of the code, do not need to be encoded and decoded since they do not contribute to the RBER. Therefore the occurrence times of possible bits in error were initially determined. If the number of possible error occurrences exceeded t within the time frame of a single word, the corresponding noise function and code word were generated and decoded. However, if the noise amplitude and the information bit have the same sign then an error will not be generated. Assuming random information bits, only 50% of the possible error events will actually generate errors. Even so, if the number of possible error events exceeds the error correcting capability of the code the code word will have to be generated.

The second acceleration technique takes this idea a step further and completely eliminates the words with less than t errors (as opposed to possible error occurrences of the first acceleration technique), by using channel noise statistics. The main difference compared with the first acceleration technique is that if the word does not contribute to the RBER it is only calculated and not simulated.

To describe the second technique a white noise channel is considered once more. Let p be the bit error probability. Therefore p^u is the probability of having exactly

u errors in u bits, while the probability of no errors in $(n-u)$ bits is $(1-p)^{(n-u)}$.

It is now clear that if all possible combinations of having u errors in a total of n bits are considered, then the probability that a code word of length n contains u transmission errors is

$$P_u = \frac{n!}{(n-u)!u!} p^u (1-p)^{(n-u)}$$

Probability p can be determined theoretically for various SNRs. Therefore, using this formula, the percentage of code words from a user defined initial set which contain u errors can be statistically determined. If a high enough initial number of code words is selected, it can be analytically determined what percentage of those words will have more than t errors within a span of n bits and simulate only these. The rest of the words (with t or less errors) are of no interest since they will always be correctly decoded. Thus they are not simulated but are only used in the final calculation of the RBER.

Simulation of the code words of interest is effected using the same techniques as described in the previous section. So, for example, if a $(127, 106)$ BCH code at a SNR of 5 dB is used with 2.85×10^7 transmitted words, only 2×10^5 code words will contain more than $t = 3$ errors and will need to be simulated, see figure 4.4. The words with fewer than t errors will be corrected by the decoder and are therefore ignored.

Finally, once the complete set of words has been decoded, the RBER needs to be calculated because the words with less than t errors were not included in the

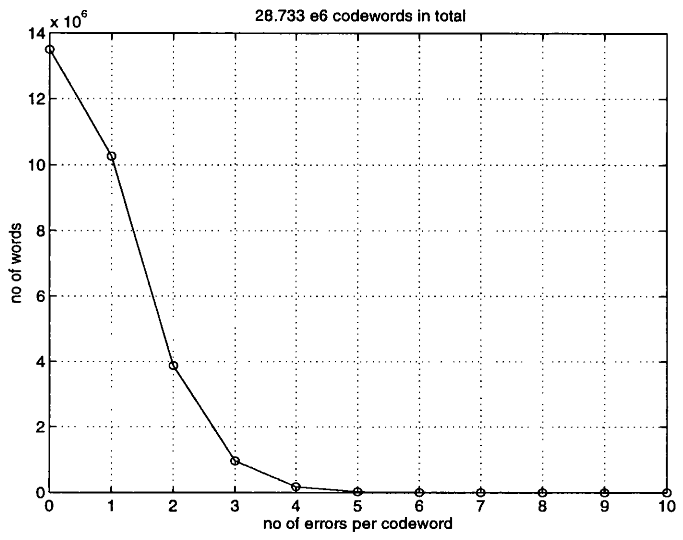


Figure 4.4: BCH (127,106) at SNR=5 simulated error distribution.

simulation. Since RBER is defined as

$$RBER = \frac{Number\ of\ information\ bits\ in\ error\ after\ EC}{Total\ number\ of\ information\ bits}$$

it is clear that the non-simulated information must be added to the denominator if a correct result is to be obtained.

The second acceleration technique can therefore be summarised as follows:

1. Select the total number of code words to be transmitted, which should be chosen to provide a sufficient number of residual errors.

2. Determine the number of code words with more than t errors and determine their distribution.

3. Generate a message and conventionally encode to create a code word. Then modify a given number of bit positions so that they are in error, as determined from the distribution obtained in step 2.

4. Generate the remaining noise samples (as described in the previous section).

5. Conventionally decode the code word.

6. Measure the number of residual errors and calculate the RBER.

The results of this second acceleration technique were also included in the simulator and were validated. Near identical results to a conventional simulation were achieved but at the same time RBERs of 10^{-9} or even lower could be simulated within a reasonable amount of time. Table 4.3 indicates the approximate times required to achieve 1000 residual errors for various SNRs.

(127, 106) BCH HDD code		
SNR	Number of words Simulated	Simulator Time Required
1	137	20 sec
2	172	21 sec
3	190	29 sec
4	220	31 sec
5	249	50 sec
6	260	1:05 min

Table 4.3: Number of words and simulator time required for obtaining 1000 residual errors at varying SNRs.

It should be noted that in this case the total number of words simulated remains virtually constant and does not increase exponentially with increasing SNR as with a conventional simulation. This is because higher values of SNRs imply that progressively larger numbers of error-free words are generated, which are not simulated. Thus, the total number of code words contributing to the RBER remains relatively stable regardless of the SNR value, which explains the very low simulation times present.

In this section, the second novel simulation acceleration technique was presented which has enabled the simulation of very low RBER levels within a reasonable amount of time and without any loss in accuracy. This is achieved by concentrating only on those code words that do contribute to the RBER and not simulating any other words which will be corrected by the decoder.

4.8 SDD and the Simulation Acceleration Techniques

Both the simulation acceleration techniques reduce the amount of computational time required by not simulating code words containing t or less errors since these are certain to be corrected by the decoder. While this is true for HDD, it may not always be the case for SDD. Specifically, the use of SDD does increase the average error correcting capability of a code over t, but may not guarantee that less that t errors will always be corrected. Three possible situations, depending on the number of errors (e) present within a code word may exist:

1. A code word has no errors, i.e. $e = 0$. In such a case the syndrome will be zero, the received code word will be immediately accepted and no decoding (HDD or SDD) will take place. The accuracy of the accelerated simulations will not be affected.

2. A code word contains more than t errors, i.e. $e > t$. In such a case, the code word will be simulated and therefore the accuracy is not affected.

3. The number of error events is one or more, upto (and including) t, i.e. $1 \leq e \leq t$. In this case it is possible that the use of the SDD may result in errors being present at the output of the decoder. For this reason these code words must be simulated, if the accuracy is not to be affected.

   However, the probability of producing residual errors in this instance is very small, especially for ECCs with large values of t. SDD of code words with less than or equal to t errors can only fail if the real error bits do not belong to the *position of least confident bit* (PLC) set. This is even more unlikely for SDD algorithms that utilise extended PLC sets, such as the *generalised Chase*, introduced in the next chapter. For this reason, the effects on the simulator accuracy of the accelerated techniques combined with the use of SDD can be ignored. This can be proven if the same ECC is used to determine the RBER utilising both the accelerated and the conventional techniques, since both results are virtually identical.

It can therefore be concluded that, for all practical purposes, the accelerated simulation techniques may be used in conjunction with both HDD and SDD.

4.9 Other Performance Evaluation Programs

In this section two more performance evaluation programs will be briefly described. These are used for obtaining the line performance and power spectrum of a given line code.

4.9.1 Line code performance

In the previous section, a computer simulation that modelled a complete communications system was introduced. This can, on principle, be modified to determine the line properties of interest such as the maximum runlength and the disparity bounds but would be impractical because of the excessive computation required. This is especially true since none of the acceleration techniques presented before can be used when line signal sequence characteristics are to be assessed. A more elegant solution is therefore the introduction of a new dedicated simulation program for determining line performance. This is briefly summarised in the following steps:

1. Generate a set of information bits.
2. Map these into line-encoded words e.g. by determining any flag bits that may be required.
3. Calculate and store the maximum runlength (RL_{max}) and disparity of the complete word.
4. Use the runlength information of the previous word to determine if a maximum runlength is generated by two consecutive words. If this is the case, then display the relevant code words.
5. Update the *running digital sum* (RDS) value and repeat from step 1.

Such a program is very useful since it allows the user to determine the chain of events leading to a line bound being reached. The code can then be modified to ensure that this will not occur again. In addition it is very fast and therefore almost all possible combinations can be examined within a reasonable amount of time.

4.9.2 Power spectrum

The final program implements the algorithm of Cariolaro and Tronca [2] which allows the power spectrum versus normalised frequency of a given code to be determined. It was introduced in 1974 and is one of the simplest ways of obtaining the power spectrum of a code. As mentioned earlier the *power spectral density* (PSD) of a code provides useful information, such as its response at DC and at low frequencies.

Unfortunately the calculation of the PSD is a very complex and computationally intensive process. In addition, it depends on the statistics of the code words and the coding rules. Therefore, a complete state transition diagram together with the associated probabilities is required for each simulated code. In practice this precludes the use of this technique for complex and/or long codes.

4.10 Summary

This chapter has described the basic steps behind generating a model for simulating BCH codes, both with hard and soft decision decoding. It was then demonstrated that in certain cases (such as when low residual bit rates are required, or the number of test patterns is large) conventional simulation proves to be too computationally intensive for practical purposes, requiring long simulation runs.

To overcome this difficulty two new simulation acceleration techniques were introduced which utilised channel statistics to reduce the number of words that need to be simulated. These have achieved significant savings in the simulation time required without affecting the accuracy of the results produced.

To complement the main performance evaluation simulation program, two software tools were described; the first one enabled line-code symbol sequence characteristics such as the RDS and RL_{max} to be determined, while the second used the Cariolaro and Tronca algorithm for coded signal power spectrum calculations.

All of the programs described in this chapter will be used for both the SDD algorithms (presented in the following chapter) and the ECLCs presented in chapter 6.

Bibliography

[1] P. Bratley, B. Fox, L. Schrage: "A guide to simulation", *Springer-Verlag*, New York, 1983.

[2] G. L. Cariolaro, G. P. Tronca: "Spectra of Block Coded Digital Signals", *IEEE Transaction in Communications, COM-22*, No 10, pp 1555–1564, Oct. 1974.

[3] D. E. Knuth: "The art of computer programming", *Addison-Wesley*, 1981.

[4] E. R. Berlekamp: "Algebraic Coding Theory", *McGraw-Hill*, New-York, 1968.

[5] S. Lin, D. Costello: "Error Control Coding", *Prentice Hall*, 1983.

[6] Y. Bian, J. O'Reilly, A. Popplewell: "Novel Simulation Techniques for Assessing Coding System Performance", *Electronics Letters*, Vol 30, No. 23, pp. 1920-21, November 1994.

[7] Y. Bian, J. O'Reilly, A. Popplewell, S. Fragiacomo: "New Simulation Techniques for Evaluation of Telecommunications Transmission Systems with FEC", *IEE Brighton Conference in Telecommunications*, March 1995.

[8] S. Fragiacomo, C. Matrakidis, A. Popplewell, Y. Bian, J. O'Reilly: "An Accelerated Simulation Technique for Evaluating Communication Systems Util-

ising FEC", *Networks and Optical Communications (NOC)*, June 17-20 1997, Antwerp, Belgium.

Chapter 5

Generalised Chase and the AID Algorithm

5.1 Introduction

In chapter 2, the three Chase algorithms were introduced which set the standard for all *soft decision decoding* (SDD) algorithms, due to their simplicity and reasonable performance. However, they also have a number of disadvantages (especially when used in high-speed systems) such as long decoding times due to the number of *test patterns* (TPs) that need to be examined and limited decoding performance.

In this chapter, the deficiencies of the Chase algorithms will be addressed by introducing three new algorithms: the *generalised Chase* (GC), the *adaptive immediate decision* (AID) and the *test pattern elimination* (TPE) algorithm.

Of the three Chase algorithms, the second one offers a good balance between decoding power and implementational complexity. Nevertheless, both of these attributes could be usefully improved, and to this end a new variation of the Chase algorithms termed *generalised Chase* (GC) has been devised. Furthermore, the *GC* algorithms may be combined with a new EEP selection technique termed *adaptive immediate decision* (AID) algorithm which reduces the average number of TPs used per code word. Finally, both the Chase and GC algorithms can be further improved by the introduction of the TPE algorithm. All of these developments are presented in the following sections.

5.2 Generalised Chase Algorithms

In chapter 2, it was determined that both the decoding power and complexity of the Chase algorithms are mainly dependent on the number (N_{Chase}) of *least confidence bit* (LCB) positions examined. Higher values of N_{Chase} imply an improved decoding performance coupled with increased complexity (in terms of TPs required), and vice-versa. Chase placed boundaries which determine the possible values N_{Chase} may obtain thus defining the performance of the SDD algorithm.

In this chapter, the bounds for the possible values of N (the number of PLCs) are relaxed thus achieving a more versatile SDD algorithm. The latter is termed *generalised Chase* (GC) and the number of possible PLC values is represented by N_{GC} [3, 4].

Since increasing the value of N_{GC} over a certain boundary effectively reduces the decoding performance of the code while increasing its complexity, the limits of

N_{GC} must be clearly defined. In addition, further restrictions must be applied to the set of resultant TPs generated by a given N_{GC} if improved performance is to be maintained. All these problems are examined in more detail in the next section.

5.2.1 N_{GC} selection and TP generation

N_{GC} selection is constrained by the existence of an optimum value. Further increases of this value will not improve performance and will result in an inefficient algorithm. This is because the use of higher values of N_{GC} means that progressively more confident bit position combinations will be examined.

Another issue is that not all $2^{N_{GC}}$ possible combinations give different estimated error patterns. This is especially true at high SNRs where the average number of errors in a code word is small. In such cases, the first few TPs are usually successful while the remaining ones (which examine bit positions that are progressively more confident) either are rejected or produce the same EEP as before.

In addition, for any given N_{GC}, only discrete permutations of all possible bit positions must be used. In all cases, the algebraic sum of the inverted positions in each TP must be less than t, i.e.

$$\sum_{i=1}^{n} TP_i \leq t \tag{5.1}$$

where $(TP)_i$ is the sum of the binary values of the TP elements. Thus, if N_{GC} least confident positions are examined, the total number of TPs will be determined by

$$\sum_{i=0}^{t} \frac{N_{GC}!}{(N_{GC}-i)!i!} \tag{5.2}$$

instead of

$$2^{N_{GC}} \tag{5.3}$$

If, for example, $N_{GC} = 5$ and $t = 3$ then only 26 TPs need to be generated instead of $2^5 = 32$.

The reason for requiring the total number of inverted positions in a TP to be t or less is best explained using an example. For a single error correcting code, numbers are assigned to all possible test patterns, as shown in table 5.1.

TP number	LCB 1	LCB 2
0		
1	invert	
2		invert
3	invert	invert

Table 5.1: Defining TPs according to the inverted LCB positions.

For example, the second TP (TP 2) has the second least confidence bit position inverted, while TP 3 has both the first and second LCBs inverted. The decoding performance of our code is now examined if a number of errors (e) ranging from 0 to 4 occurs within a single code word. Referring to table 5.1 the probability of erroneously decoding a code word will be determined for two different cases:

- Case (a): decode using the proposed rule, as expressed by equation 5.2, using all possible combinations of one LCB position, i.e. TP 0 and TP 1.
- Case (b): decode using more LCBs, e.g. using 4 TPs (TP 0,1,2,3), as defined by equation 5.3. This set includes TP 3 which contains a number of inverted positions exceeding t.

All permutations of errors and TPs are examined in tables 5.2 and 5.3 where an 'X' indicates a failure to decode while a $\surd$ signifies a successfully decoded word. Table 5.2 indicates the results using case (a), i.e. the performance of a single ECC where a single position of least confidence is used by the SDD algorithm, as indicated by the proposed rule. Table 5.3 indicates the performance of the same code using case (b), i.e. two LCBs which correspond to four possible TPs. In both cases, the first column presents the number of errors in the code word and the second the test pattern used (using the conventions indicated in figure 5.1). The following columns indicate whether a word with a given number of errors and a specific TP would be correctly decoded or not.

Assume that the probability of erroneously decoding a code word for various numbers of errors is equal to P_e. In such a case the resultant overall probability of erroneously decoding a word using either 2 or 4 TPs is shown in table 5.4.

Adding the corresponding probabilities for both cases produces a $P_e(a) = 0.5$ for 2 TPs and a $P_e(b) = 0.56$, where $P_e(a)$ is the probability of erroneously decoding a code word if case (a) is used, and $P_e(b)$ if case (b) is used. Therefore the use of all possible $2^{N_{GC}}$ TPs not only increases the decoding time required but also increases the probability of erroneously decoding a code word. Even though the

No of errors		LCB matches error		LCB does not match error
0	TP 0	√		√
	TP 1	√		√
1	TP 0	√		√
	TP 1	√		X
2	TP 0	X		X
	TP 1	√		X
3	TP 0	X		X
	TP 1	X		X

Table 5.2: Single error correcting performance using 2 TPs.

above example considered a simple ECC, similar results occur regardless of the code used.

Another reason for constraining the number of inverted positions in a TP using equation 5.1, is to avoid exceeding the minimum Hamming distance d_{min} of a code word. If the latter does occur, it is possible that our received word may be erroneously decoded as an adjoining (in terms of Hamming distance) one.

To summarise, it is clear that an optimum maximum value of N_{GC} exists above which the number of TP increases significantly without a comparable increase in decoding power. Furthermore, not all possible TP combinations can be used once a value for N_{GC} is defined. Experimental results using a computer simulation have clearly demonstrated that the best choice is $N_{GC} = (N_{Chase2} + 2)$ where $N_{Chase2} = \lfloor \frac{d}{2} \rfloor$. At this value of N_{GC} not only is most of the coding gain recovered,

Errors	TP	No Match	LCB 1 Match	LCB 2 Match	LCB 1/2 Match
0	0	√	√	√	√
	1	√	√	√	√
	2	√	√	√	√
	3	X	X	X	X
1	0	√	√	√	√
	1	X	√	X	X
	2	X	X	√	√
	3	X	√	√	√
2	0	X	X	X	X
	1	X	√	X	√
	2	X	X	√	√
	3	X	X	X	√
3	0	X	X	X	X
	1	X	X	X	X
	2	X	X	X	X
	3	X	X	X	X

Table 5.3: Single error correcting performance using 4 TPs.

Number of errors in word	P_e for 2 TPs	P_e for 4 TPs
0	0	$\frac{4}{16}$
1	$\frac{1}{4}$	$\frac{6}{16}$
2	$\frac{3}{4}$	$\frac{11}{16}$
3	1	$\frac{15}{16}$

Table 5.4: Probability of incorrectly decoding a word for various numbers of received errors.

but the number of TPs remains reasonably low. If this value of N_{GC} is exceeded, the increased number of TPs is not justified by any RBER gain that *may* be present.

The fact that N_{GC} is defined as having the value $N_{GC} = (N_{Chase2} + 2)$ indicates that a larger set of TPs is examined compared to the standard Chase 2 algorithm. This implies that improved decoding performance will be present. Since N_{GC} examines fewer TPs than Chase 1 it is also implied that the decoding performance may not match that of Chase 1. However, the GC performance is very close to the ML limit which makes it a very attractive algorithm. This will be demonstrated by the simulation results presented in the following section.

5.2.2 Simulation results

The generalised Chase algorithm was simulated in order to determine the decoding performance for various SNRs and the optimum values for N_{GC}. The introduction of the GC algorithms made the need for accelerated simulation techniques as described in [5] even more acute. This is because the use of large values of N_{GC}

and the corresponding large set of TPs that need to be examined increases the simulation times beyond acceptable limits.

In this section, the decoding performance of a $(127, 106)$ BCH code will be used as an example (see figure 5.1). This is a triple error correcting code and if the conventional Chase 2 algorithm was used N_{Chase} would be equal to three, i.e. three LCBs would be examined which would correspond to eight TPs. In this example, values of N_{GC} ranging from one to five will be used.

The upper solid line of figure 5.1 (labelled 'Uncoded') indicates the decoding performance of the system when no error correcting code has been used and the data are transmitted as they are generated from the source.

The solid line below it (labelled 'HDD') indicates the performance when a triple error correcting code has been included and the decoder utilises hard decision decoding. The next two dashed lines indicate the performance of the same code but decoded using soft decision and the generalised Chase algorithm with $N_{GC} = 1$ (i.e. $2^1 = 2$ test patterns) and $N_{GC} = 2$ (i.e. $2^2 = 4$ test patterns).

The solid line below that indicates the $N_{GC} = 3$, 8 TP conventional Chase 2 results while the next two dashed lines are the results for $N_{GC} = 4$ (16 TPs) and $N_{GC} = 5$ (26 TPs). The final solid line (labelled 'Ideal SD') indicates the best possible SDD performance, ML.

For the above code if $N_{GC} = 5$, a 0.5dB increase in coding gain compared to the Chase 2 algorithm is offered. The penalty for such a decoding improvement is a corresponding increase from 8 to 26 test patterns.

Figure 5.1 clearly indicates that by using all the combinations of 5 LCBs de-

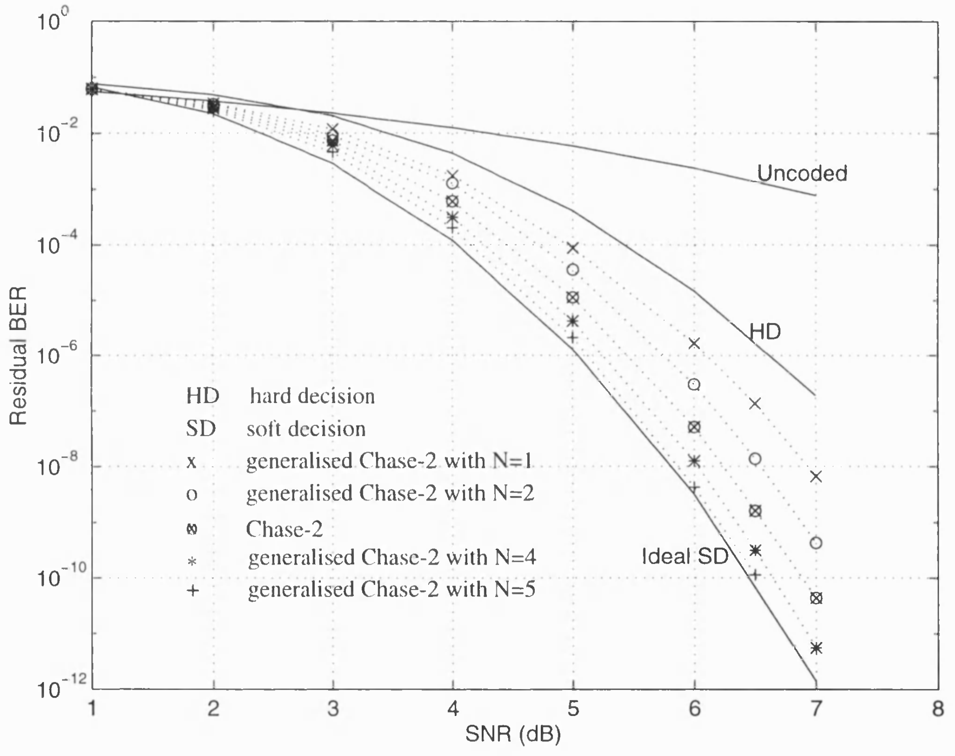


Figure 5.1: Simulation of a BCH (127,106) error correcting code with various values of N.

coding power is very close to the ML bound. Any further increase over $N_{GC} = 5$ will not significantly increase decoding performance. The decoding results once the optimum value of N_{GC} is exceeded are demonstrated in the following examples:

In the first example a (63,51) double EC BCH code is used with $N_{Chase} = 2$ and $N_{GC} = 4$. The decoding results for a SNR of 4dB are shown in Table 5.5.

The resulting RBER steadily improves with increasing values of N_{GC} until $N_{GC} = 4$. For $N_{GC} = 5$ an insignificant improvement exists and for $N_{GC} > 5$ the RBER deteriorates.

In the second example, a $(127, 106)$ triple ECC is used and in this case $N_{Chase} = 3$ and $N_{GC} = 5$, as shown in Table 5.6. Again, the RBER progressively improves until $N_{GC} = 5$ and thereafter remains almost constant. Similar results are found with any combination of BCH code and SNR.

5.2.3 Generalised Chase 3

In the previous sections, the generalised Chase algorithm was introduced. This is an extension of the Chase 2 algorithm whereby a larger set of TPs is examined. The method of generating this extended set remains the same as that of Chase 2. In this section, a possible extension of the conventional Chase 3 called generalised Chase 3 (GC-3) is developed.

The GC-3 will examine (depending on whether d_{min} is odd or even) a set of test patterns with i positions of least confidence inverted. Specifically if d_{min} is odd, Chase 3 dictates that $i = 0, 1, 3, ..., d - 1$ and if d_{min} is even then $i = 0, 2, 4, 6, ..., d - 1$. The (GC-3) simply extends the values of i for both cases, gener-

N	1	2	3	4	5	6
RBER (10^{-4})	34.9	5.9	2.33	2.0	2.0	2.28
TPs	2	4	7	11	16	22

Table 5.5: BCH (63,51) ECC at SNR=4 dB with various values of N (channel BER = 1.2×10^{-2}).

N	1	2	3	4	5	6
RBER (10^{-6})	88.1	35.6	11.2	4.2	2.0	1.9
TPs	2	4	8	15	26	42

Table 5.6: BCH (127,106) ECC at SNR=5 dB with various values of N (channel BER = 5.9×10^{-3}).

ates the appropriate TPs and then performs a conventional hard decision decoding. However, unlike GC-2, GC-3 was found to not offer any significant improvements in performance with increasing values of i and was therefore not investigated further.

5.3 Adaptive Immediate Decision Algorithm

In the previous sections, it was demonstrated how the decoding power of the Chase algorithms can be increased. This was achieved by using a larger set of TPs for decoding by setting $N_{GC} = (N_{Chase2} + 2)$. Both the standard and the generalised Chase algorithms generate and examine a complete set of test patterns before selecting the corresponding estimated error pattern of minimum analogue weight.

However, this will result in increased decoding time which will place bounds on the maximum bit rate allowable. A new algorithm, the *adaptive immediate decision* (AID) has been developed to address this problem, which reduces the average number of TPs required without significantly affecting performance. In this section, the AID algorithm will be presented and used in conjunction with both the Chase and generalised Chase codes. This will make the GC a more attractive option for use in high bit rate systems.

5.3.1 Description of the AID algorithm

The AID algorithm offers an improved TP selection procedure compared to Chase. It is based on the observation that for stationary noise channels the analogue weight of the selected EEP (aw_{EEP}) is another stationary random variable. This is because the algebraic sum of the absolute voltage levels of each erroneous bit in a selected *estimated error pattern* (EEP) (as described in equation 2.9) should have statistically stable values.

Furthermore, for white noise channels and for a relatively large code word length n, the analogue weight of the selected estimated error pattern aw_{EEP} is concentrated around its mean value $M_{aw_{EEP}}$. It is therefore possible to devise a threshold of analogue weight, e.g. $T_{aw_{EEP}} = M_{aw_{EEP}}$, so that an EEP selection decision is made immediately, based on the calculated aw_{EEP}. Such an algorithm reduces the *average* number of required TPs for correctly decoding a received word. This offers a significant reduction in the required decoding time per word, thus allowing higher bit-rate coded systems to be realised.

5.3.2 Threshold decision

The concept behind the AID algorithm is the fact that the complete TP set may not need to be generated for each code word. This will reduce the average number of TPs examined, thus reducing the decoding time and complexity required. This technique is effected by calculating the analogue weight of each EEP (aw_{EEP}) after decoding has taken place. If $(aw_{EEP}) \leq (T_{aw_{EEP}})$, then the EEP will be immediately accepted otherwise the next test pattern is examined and the process repeated. If the analogue weights of the complete set of EEPs are greater than the threshold, i.e.

$$(aw_{EEP_i}) \geq (T_{aw_{EEP}}) \quad \forall \ 1 \leq i \leq \textstyle\sum_{i=0}^{t} \frac{N_{GC}!}{(N_{GC}-i)!i!}$$

then the threshold condition will not be satisfied for any TP. In such a case conventional Chase decoding will take place. This will result in the EEP of minimum analogue weight being selected. In either case, the analogue value of the selected EEP aw_{EEP} will also be added to the value of $T_{aw_{EEP}}$. The flow diagram of the AID algorithm is shown in figure 5.2.

Defining the threshold values

For a stationary channel, the noise content will remain almost constant regardless of how many words have been transmitted. In such a case, after a statistically significant sample of initial words has been decoded using a conventional Chase 2 or GC-2 algorithm, $T_{aw_{EEP}}$ can be determined. The latter will remain reasonably stable and therefore will not need to be re-evaluated. Experimental results suggest that a suitable number of initial words is about 100.

For a fluctuating noise channel however, $T_{aw_{EEP}}$ is not constant. Therefore it must be constantly re-calculated otherwise the threshold level will be invalid. In such a case the average number of TPs required will be increased, or erroneous results will be produced. For such channels, a windowed averaging technique consisting of a number (m) of most recent words is used to calculate $T_{aw_{EEP}}$. The window width (m) needs to be large enough to ensure statistically accurate results; if it is too large then the fluctuations of the noise channel will not be closely followed. An optimum width must therefore be determined which depends on the stability of the noise channel. A rapidly fluctuating channel would require a narrow window for accurate noise tracking; a more stable channel could utilise a larger window likely to offer more statistically accurate results. The threshold weight will

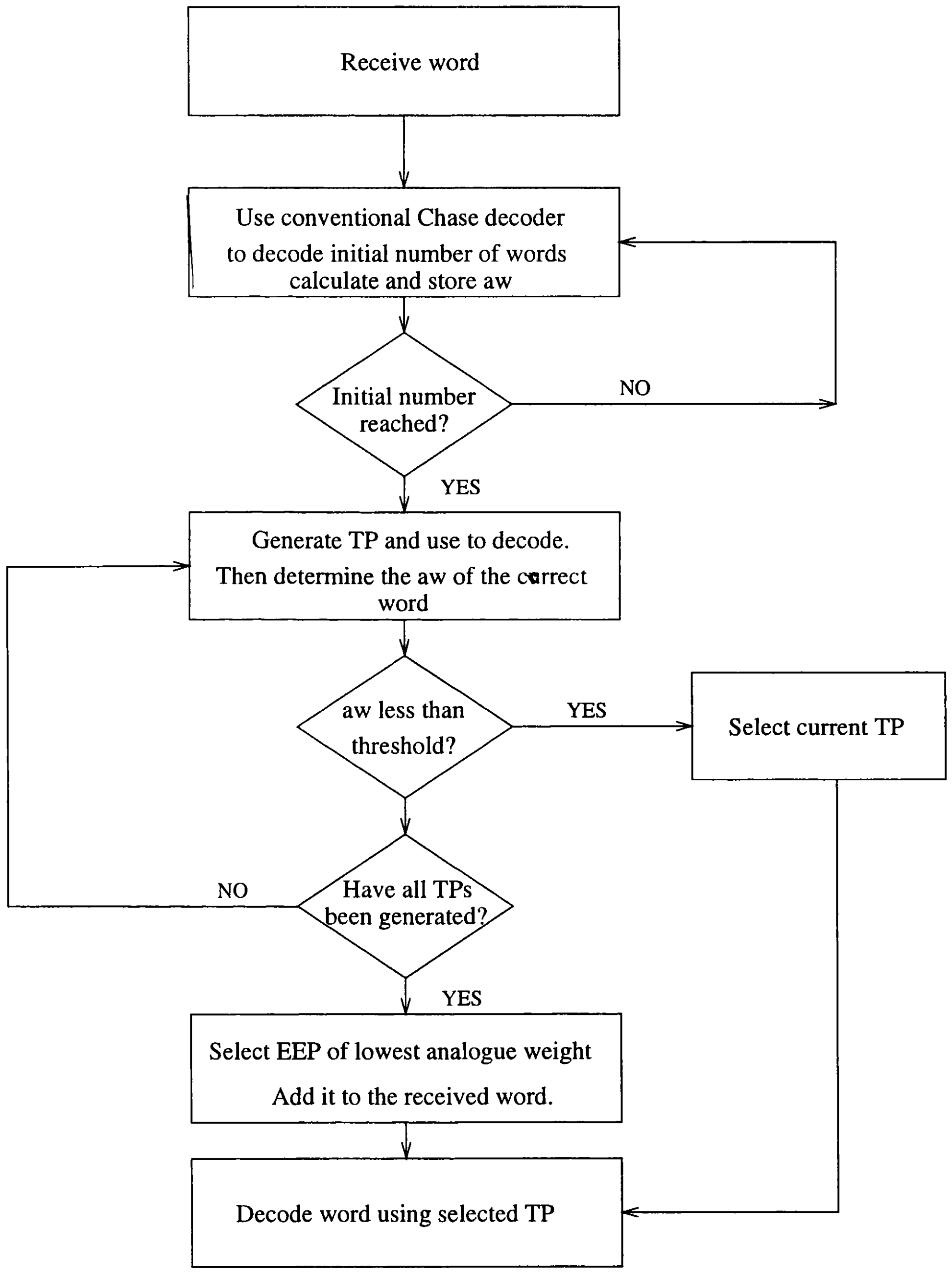

Receive word
Use conventional Chase decoder to decode initial number of words calculate and store aw
Initial number reached?
NO
YES
Generate TP and use to decode. Then determine the aw of the correct word
aw less than threshold?
YES
Select current TP
NO
Have all TPs been generated?
YES
Select EEP of lowest analogue weight Add it to the received word.
Decode word using selected TP

Figure 5.2: AID algorithm flow diagram.

therefore be defined as

$$T_{aw_{EEP}} = \frac{\alpha}{m}\sum_{j=1}^{m}(aw_{EEP})_j \tag{5.4}$$

where α is a factor that determines the decoding power versus speed and $(aw_{EEP})_j$ is the analogue weight of the *jth* most recent word. The factor 'α' is introduced to provide more flexibility by artificially raising or lowering $T_{aw_{EEP}}$ thus allowing a trade-off between decoding speed and power.

5.3.3 Optimal value of α

From formula 5.4 it is clear that the values of α are very significant since they dictate the decoding power and corresponding time required by the algorithm.

For example, if $\alpha \leq 0$ the code words are decoded using a conventional GC algorithm since the condition $aw_{EEP} \leq T_{aw_{EEP}}$ is never satisfied except for the zero-error case. If α is a large positive number (e.g. greater than 2) then the condition $aw_{EEP} \leq T_{aw_{EEP}}$ is easily satisfied and therefore the average number of examined EEPs is reduced. However, in such a case, an increased probability of error exists. The opposite is true if α is a small positive number such as 1 or 2, because more test patterns are examined.

In order to determine the optimal value of α, the simulator was used to provide a number of examples which are presented in the following section in Tables 5.7 and 5.8. In all cases the optimal value of α was determined to be between 2 and 3 since at this level most of the decoding power was present while the required number of TPs was reasonably low.

The formula for calculating $T_{aw_{EEP}}$ allows the algorithm to be adaptive in nature so that it can be used in situations where the SNR levels may vary with time. It can also be used in stationary channels without compromising performance. Since the complete set of TPs is not always required, a major improvement in the decoding time can be achieved, up to an order of magnitude, without significant loss in decoding power.

Sample results for a triple error correcting BCH(127,106) code at various stationary and fluctuating SNRs are included in the following section.

Chase's threshold decoding

In [1], Chase also introduced the concept of threshold decoding. It should be noted though that this is completely different to the threshold decoding arrangement presented here. The technique introduced by Chase uses a threshold to examine whether an *error pattern* (EP) (as opposed to an EEP) will be accepted for correcting a word. This means that the complete set of TPs must first be created and examined. Therefore, in Chase, the decision is not immediate (after every EEP) and if the accepted EP has an analogue weight above the threshold then the accepted word is taken to be the received word without any corrections. Chase therefore uses his threshold weight to examine the validity of an EP.

A second difference between Chase's threshold decoding and the AID algorithm is that Chase provides a different definition for the threshold analogue weight $aw_{EEP}(T_K)$. The latter is defined as the analogue weight of a TP containing K ones, where K is a user specified arbitrary number that can take values $1 \leq K \leq n$, where n is the code word length.

5.3.4 Simulation results for the AID algorithm

In the previous sections, the AID algorithm was introduced. The main advantage of this algorithm is that it will achieve a reduction in the average number of TPs examined per code word without a significant reduction in decoding performance.

In this section, the performance improvements will be demonstrated, using the simulation techniques introduced in the previous chapter and in [5, 6]. A BCH $(127, 106)$ triple error correcting code with $N_{GC} = 5$ will be used as an example but similar performance was present when other codes were used.

The main aim of the AID algorithm is the reduction of the number of TPs used and this is especially useful in conjunction with the GC codes, due to the increased number of least confidence positions they examine. It has been noted before that near maximum likelihood performance for a BCH code exists if the generalised Chase algorithm examines $N_{GC} = N_{Chase2} + 2$. Since this creates a relatively large set of TPs the introduction of the AID algorithm should offer substantial advantages in reducing the total simulation time required.

Table 5.7 presents the simulation results of a $(127, 106)$ triple error correcting code with $N_{GC} = 5$ while simulated at a $SNR = 4$.

The first column indicates the value of the factor α which determines the decoding power used. Decreasing the value of α improves decoding performance. Increasing the value of α however, decreases the average number of TPs and the associated decoding time required. The second column indicates the decoding performance of the code (the RBER) while the third column indicates the average number of TPs used to achieve the corresponding decoding performance. The theoretical RBER of this particular code at this SNR is 1.2×10^{-4}.

From table 5.7 it is evident that there is a small difference in the ML limit and the actual RBER while using the GC with $N_{GC} = 5$. The latter is shown in the first row (where $\alpha = -1$) since with this value of α the threshold condition is never satisfied. The total number of TPs is equal to 26 in accordance with equation 5.2. Progressively increasing the value of α decreases both the decoding performance and the number of TPs used. However, at a value of $\alpha = 3$, while the reduction in the decoding performance is insignificant, the average number of TPs examined per word has dropped by an order of magnitude, from 26 to an average of about 2.

Table 5.8 presents the results for a $(63, 51)$ double error correcting BCH code with $N_{GC} = 4$ simulated at a SNR of 5 dB. This will create a set of 16 TPs and will have a theoretical RBER of 5.6×10^{-6}. Once more the optimum value of α is three, and a minimal reduction in decoding performance exists. Similar results were observed for all combinations of codes and SNRs.

The reduction in the number of required TPs can be demonstrated by setting an arbitrary time limit and decoding the largest number of code words within that limit. For example, in the same time interval that a GC algorithm with $N_{GC} = 5$ at a SNR of 3 dB examines 1000 words, AID will have examined over 8000 words. For comparison, if a conventional Chase 2 decoder was used, about 3000 words would have been examined in the same time interval, but with a much reduced decoding performance.

(127, 106) BCH ECC		
α	Decoding performance	Average number of TPs
-1	2.3×10^{-4}	26
1	2.4×10^{-4}	9.7
2	2.5×10^{-4}	3.8
3	2.6×10^{-4}	1.9
4	3.0×10^{-4}	1.4

Table 5.7: Decoding performance versus average number of TPs for a (127, 106) BCH ECC code at SNR=4dB.

(63, 51) BCH ECC		
α	Decoding performance	Average number of TPs
-1	6.2×10^{-6}	16
1	6.4×10^{-6}	3.5
2	6.7×10^{-6}	2.2
3	6.8×10^{-6}	1.4
4	9.1×10^{-6}	1.1

Table 5.8: Decoding performance versus average number of TPs for a (63, 51) BCH ECC code at SNR=5dB.

5.3.5 Variable noise simulations

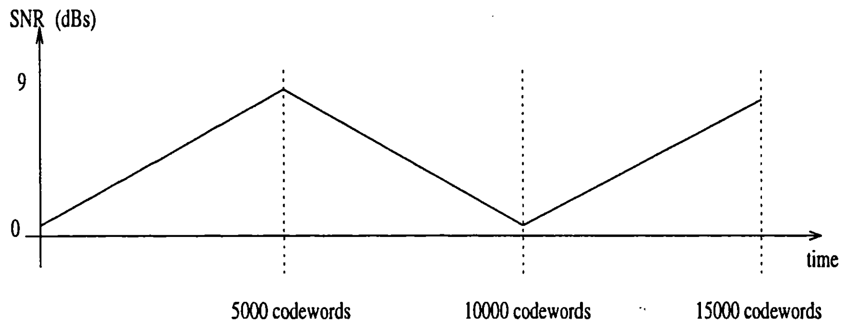
The previous simulations have demonstrated that the AID algorithm has managed to reduce by an order of magnitude the number of TPs required, compared to the GC algorithms, when the SNR remained reasonably constant. The latter is a condition not necessarily representative of a real system. In this section a variable SNR source is used in order to determine the performance of the algorithm.

Two different types of non-constant amplitude noise were used to model the behaviour of a non-stationary system while using the AID algorithm. The first one involved randomly variable noise figures between code words to simulate a rapidly fluctuating transmission channel. This was implemented by allowing the SNR to take random but discrete values. The performance achieved was not significantly worse compared to that of a constant SNR channel. Specifically, the total average number of TPs used increased from an average of 3.4 (for a stationary channel) to about 5.5 while the decoding performance remained at similar levels, as shown in table 5.9.

(127, 106) BCH ECC			
α	Channel Error Rate	RBER	Average number of TPs
-1	1.2×10^{-3}	1.7×10^{-4}	26
1	1.2×10^{-3}	2.1×10^{-4}	11
2	1.1×10^{-3}	2.4×10^{-4}	5.15
3	1.2×10^{-3}	3.8×10^{-4}	3.8

Table 5.9: Code performance simulation results.

The second series of tests used a sloping noise function with SNRs increasing and decreasing with time as illustrated in figure 5.3. The resultant RBER was similar when using both algorithms (AID and GC-2) but AID used an average of 3 TPs as opposed to 26. This represents a significant reduction in decoder complexity and decoding time required. Both of the above tests provide an indication of the potential of the AID algorithm if the values for N_{GC} and α are chosen appropriately.



SNR improving with respect to time between every bit

	Generalised Chase-2	AID with α=3
Channel Error Rate	1.7689e-2	1.778e-2
Residual Bit Error Rate	1.52e-3	1.73e-3
Number of Test Patterns Used	26	3.08

Figure 5.3: Performance comparison between the GC-2 and AID for a (127, 106) BCH code, with sloping noise functions.

5.4 Improving the Chase and AID Algorithms

The introduction of the GC algorithm established the fact that the use of an increased number of test patterns will result in improved decoding performance. The AID algorithm was then used to decrease the number of TPs without significantly affecting the performance of the ECC. In this section, a further improvement called *test pattern elimination* (TPE) will be introduced. The TPE algorithm significantly reduces the decoding time per code word required by reducing the number of TPs that are taken through the EC decoder, as first suggested by Chase. This is achieved by eliminating TPs that produce the same EEP, thus not affecting the decoding performance of the 'parent' algorithm.

The TPE algorithm can be briefly described as follows: In both the Chase and AID algorithms, a TP is added to the received code word and the result is passed through the EC decoder. The latter can correct up to t errors and will produce an *estimated error pattern* (EEP). The Chase algorithm will then select the EEP of minimum analogue weight once a complete set has been generated, while the AID algorithm will immediately select an EEP if its analogue value is below a set threshold.

Observation of the decoding stage however, indicates that a large number of TPs will eventually give the same EEP. This is a very serious drawback of both algorithms, since the EC decoder is the most power intensive component of the decoder.

The TPE algorithm suggests that if the Hamming distance (d) between the current TP and any existing EEP is less or equal to t, i.e.

$$\exists\ i < n : d(TP_i, EEP_n) \leq t$$

then the same EEP will result. Therefore, such TPs will be immediately rejected without going through the decoder. The reason behind this technique is the fact that the decoding algorithm used will always return a 'valid' code word, i.e. a code word with zero syndrome. If the distance between a TP and any one of the existing EEPs is less or equal to the error correcting capability t of the code, then the use of the EC decoder will produce as a result an existing EEP.

Simulation results using both the Chase and AID algorithms with TPE have indicated that a significant reduction in the number of times the decoder is used exists, without affecting the decoding performance at all. Figure 5.10 indicates the reduction in the use of the decoder using a BCH $(127, 106)$ triple error correcting code at a SNR of 3 dB as an example.

Code type	Decoder usage	Max TP number	Average TP number
Chase	7.8	8	8
Chase and TPE	3.8	8	8
AID	10.3	26	11.3
AID and TPE	8.4	26	11.3

Table 5.10: Number of times decoder is used for a BCH $(127, 106)$ ECC at a SNR of 3 dB.

The first column of figure 5.10 presents the type of SDD algorithm used, the second indicates the number of code words that went through the EC decoder, the third column presents the maximum number of TPs allowable while the final column indicates the average number of TPs used.

Table 5.10 indicates that the use of the TPE algorithm combined with Chase decoding offers a very significant reduction (of the order of 50%) in the number of times the decoder is used. This reduction is not so significant when combined with the AID algorithm since the use of the immediate decision threshold will usually select a code word before a significant amount of repetition in the EEPs is present. In both cases, the introduction of the TPE algorithm has not affected at all the decoding performance of the 'parent' codes.

5.4.1 Decoding bounded algorithm

An alternative implementation of the TPE algorithm is the *decoding bounded* (DB) algorithm. In this implementation an upper limit on the number of allowable decodings is placed. The decoder will then generate an appropriate number of TPs until the pre-set limit is reached. The difference with the algorithms presented in the previous sections is that no set number for N (the number of examined PLCs) exists. Therefore, the algorithm will use as many TPs as required to achieve the number of decodings specified.

5.5 Summary

In this chapter, the shortcomings of the Chase algorithms were identified. These consisted of a relatively poor decoding performance compared to the ML limit and a large number of TPs required to achieve that performance. Both shortcomings were addressed in three new algorithms.

The first of these algorithms used an extended number of least confident positions in order to generate a larger set of test patterns. This resulted in enhanced decoding capability but at the same time increased the necessary decoding time per word. Since this algorithm was an extension of the Chase algorithm it was termed the *generalised Chase* (GC).

The second new algorithm introduced a threshold for immediately selecting the appropriate TP. The use of the threshold technique implies that the complete set of TPs does not need to be generated before one is selected. This enables the average decoding speed to be greatly increased without affecting decoding performance. The resultant algorithm is termed *adaptive immediate decoding* (AID) algorithm. The latter can be applied to either the Chase or the generalised Chase algorithms and offers significant decoding speed improvements.

The final algorithm reduces the number of times the EC decoder must be used by eliminating all TPs that generate the same EEP. This algorithm is termed TPE and can be used in conjunction with either the Chase or the AID algorithms. It offers the same decoding performance as that of the 'parent' algorithm while significantly reducing the decoding time required.

These improvements in decoding speed and performance make it practical to consider the application of forward error control based on BCH codes and SDD, to high bit-rate optical fiber transmission systems. Such systems though also require the use of a line code. Previous work has shown that both line coding and error control functions can be merged to realise *error correcting line codes* (ECLCs).

To date, this type of work has generally utilised hard decision decoding. It is appropriate therefore to explore the extent to which the new SDD developments may be combined with line coding as discussed in Chapter 3, to achieve enhanced ECLC performance appropriate to high bit-rate systems. We turn our attention to this in the next chapter.

Bibliography

[1] D. Chase: "Class of Algorithms for Decoding Block Codes", *IEEE Transactions on Information Theory, Vol IT-18*, pp. 170-179, January 1972.

[2] G. D. Forney,"Generalised Minimum Distance Decoding", *IEEE Transactions on Information Theory, Vol IT-12* pp. 125-131, April 1966.

[3] S. Fragiacomo, Y. Bian, A. Popplewell, J. O'Reilly: "A New Low Complexity Near ML Soft Decision Decoding Algorithm for Linear Block Codes", *IEEE International Conference on Communication Systems*, IEEE ICCS/ISPACS '96, Singapore, 25-29 November, 1996.

[4] Y. Bian, A. Popplewell, J. O'Reilly, S. Fragiacomo, R. Blake: "FEC for Future Trans-Oceanic Optical Systems", *Fifth IEE Conference on Telecommunications*, Brighton UK, March 1995.

[5] Y. Bian, J. O'Reilly, A. Popplewell, S. Fragiacomo: "New Simulation Techniques for Evaluation Telecommunications Transmission Systems with FEC", *IEE Conference in Telecommunications*, Brighton, March 1995.

[6] Y. Bian, J. O'Reilly, A. Popplewell: "Novel Simulation Techniques for Assessing Coding System Performance", *Electronics Letters*, Vol 30, No. 23, pp. 1920-21, November 1994.

Chapter 6

Combining Error Control and Line Coding with Soft Decision Decoding

6.1 Introduction

In the previous chapters, the basic principles of *error correcting codes* (ECCs), *line coding* (LC) and *soft decision decoding* (SDD) were presented. It was also shown that each individual technique can offer significant improvements in increasing the performance of a communications link.

However, if a high degree of reliability is required it is not uncommon to utilise both the ECC and LC schemes. This is usually achieved by effecting a cascaded implementation with the LC being the inner-most code on both sides of the link. This arrangement has the advantage of simplicity and flexibility of choice of the

ECC and LC but it also has significant disadvantages. An alternative technique is to combine the two operations into a single code, an *error correcting line code* (ECLC). In this chapter, both the cascaded approach and ECLCs will be examined in some detail. Some existing codes will be reviewed and new codes introduced.

ECLCs can be modified so that with the use of SDD techniques, significantly better performance can be achieved. All three algorithms introduced in the previous chapter (GC, AID and TPE) can be used for decoding ECLCs, as will be shown in the following sections.

6.2 The Need for Error Correcting Line Codes

Error correction and line coding are usually regarded as two distinct operations. However, some circumstances require the simultaneous application of both; a particularly widespread example being the compact disk system. The conventional approach has been to apply the two operations in cascade with the line code being the innermost code, as shown in figure 6.1. The information bits (i) are initially taken through the error correcting encoder and parity bits (p_1) are appended. The resultant word is then taken through the line encoder and have further parity bits (p_2) appended, before the complete word is transmitted. Such a concatenated scheme was presented by e.g. Lin [1] whereby a number of error correcting/runlength limited codes were created using trellis encoding and decoding.

The aim of the LC is to match the transmitted signal to the transmission medium. It therefore has to be the inner-most code. If this were not the case, the transmitted word would be modified by the ECC, thus eliminating the benefits of

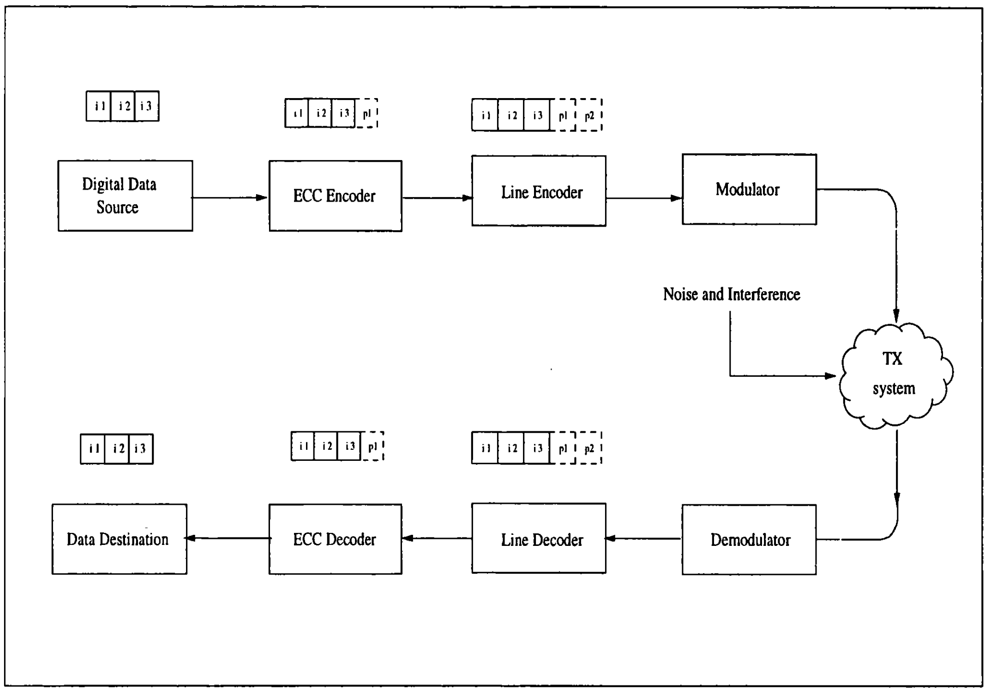


Figure 6.1: Conventional cascaded error correcting line code implementation.

the line code.

This arrangement however, may have several disadvantages: As was noted in chapter 3, line decoding can introduce error extension which reduces the overall decoding performance of the outer error correcting code since some of its power has to be used to combat the extra errors.

In addition, with the cascaded scheme each coding operation requires the introduction of redundancy. The redundant bits are introduced independently at each stage and can not be used for both EC and LC. This results in a decreased overall code rate R. Finally, the output of the line decoder is a binary waveform which prevents soft decision decoding techniques being used at the ECC decoder. This further limits the performance capabilities of such a cascaded scheme.

To summarise, the cascaded approach has the following performance characteristics, when compared to conventional ECC:

- Lower overall rate, due to two sets of independent parity bits.
- Error extension is present.
- The use of SDD may not be possible.

Most of these problems can be alleviated by using combined *error correcting line codes.* These codes combine the normally separate functions of line signal conditioning and error protection into a single operation. By adopting this approach error extension is minimised and higher overall code rates can be achieved.

Since the output of an ECLC encoder is a sequence of error control words, the full spectrum of error detection and correction can be applied directly to the received line signal. In particular, it will be demonstrated that it is possible to apply SDD to achieve further overall performance enhancement compared with conventional cascaded arrangements.

Figure 6.2 outlines such a system with the option of SDD included. Once more, i information bits are taken through an ECLC encoder and have a number of parity bits ($p_{1,2}$) added to them, before transmission takes place. At the receiver, the waveform goes through the error correcting decoder first so that any errors can be corrected before any line decoding is effected. This arrangement offers two significant benefits; SDD can now be used and the occurrence of error extension is minimised.

A number of such combined schemes already exist. Brooks [2], for example, has suggested a simple error detecting line code suitable for implementation in a high

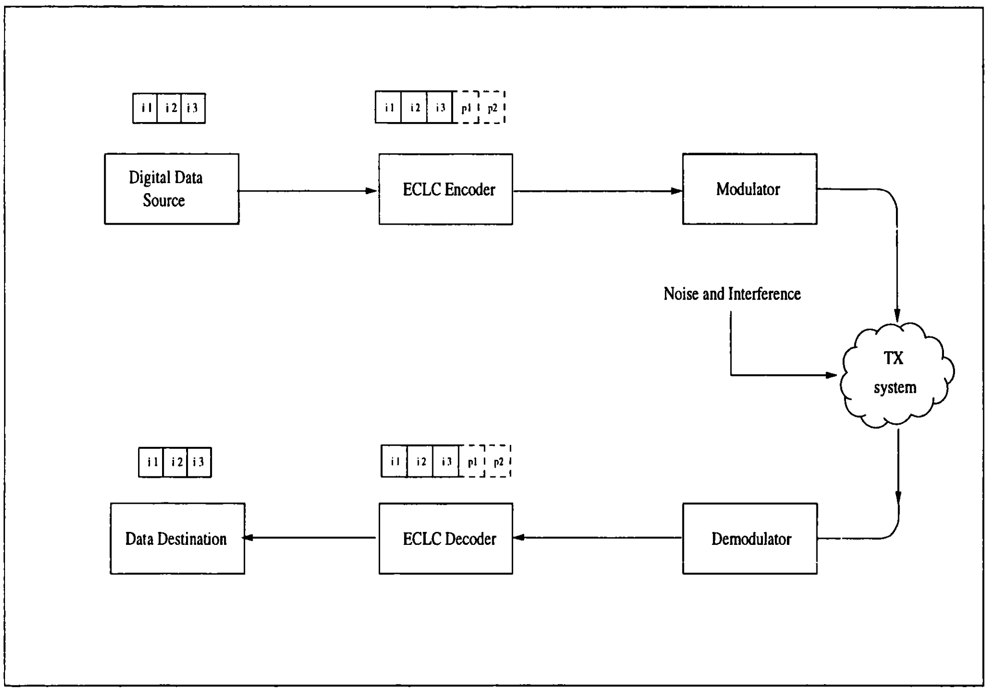


Figure 6.2: Error correcting line code block diagram.

speed system while Helberg *et al.* [3] has suggested a code that can either perform line coding or error control but not both simultaneously.

Some of the codes presented in this chapter exhibit concatenated code characteristics, such as separate redundancy for line and error correction coding, and error extension. However, in all cases, error correction is effected before line decoding takes place and SDD is always possible. For these reasons, all of the following codes are classified as ECLCs.

6.3 Generating ECLCs

We can identify three broad approaches to the description and identification of an ECLC:

1. The first approach exploits the redundancy of a line code to achieve some degree of error control. As a simple example consider an *alternate mark inversion* (AMI) code in which a binary waveform is given a ternary representation with a binary zero encoded as a ternary zero and a binary one is alternately encoded as a ± 1.

   If, at the decoder, two consecutive ones of equal polarity are received then an error has been detected. This allows isolated errors to be detected but the structure and degree of redundancy is insufficient to allow for correction if hard decision decoding is employed.

   In such a code, the principal reason for the introduction of redundancy is to achieve line code characteristics; this technique does not produce powerful or versatile error correcting codes although SDD can provide a degree of error performance improvement [4]. Similar drawbacks exist for most such schemes based on line codes and for these reasons they will not be considered in this thesis.

2. The second approach is to use a 'parent' error correcting code, modified in such a way so that line coding characteristics are present without the loss of the algebraic structure or the decoding power of the code. This can imply balancing the disparity of the transmitted code words and/or limiting the maximum runlength (RL_{max}). Deng's [5] and Popplewell's [6] codes belong to this category.

   In order to achieve these characteristics the code rate will usually have to be reduced to a certain extent but this is still a powerful technique. It will therefore be examined here in more detail.

3. Design totally new ECLC codes. A number of examples of this type of coding exist but they are generally rather complex, use look-up tables to encode and decode, and lack algebraic structure. Most such codes begin with zero disparity words which also have some minimum Hamming distance. Error correcting properties are then introduced, so that an ECLC is formed [7, 8]. The lack of structure and poor overall rate limit their usefulness, especially for more complex and longer codes. For these reasons they will not be considered here.

6.4 Proposed Codes

A number of codes presented in the following sections are based on those introduced by Popplewell [6]. These are further modified in order to exploit the advantages of soft decision decoding in combination with ECLC objectives. Such codes are relatively simple to implement, have an algebraic structure thus being applicable to any code length and are systematic.

In most cases a 'parent' ECC will be used to generate an ECLC. In our case, this will be a cyclic linear block (n, k) BCH code with n code word bits and k information bits, with $n > k$. The minimum Hamming distance between any two words is d_{min} and $t = \lfloor \frac{d_{min}}{2} \rfloor$ is the number of errors which can be corrected in any one word.

The parent ECC is then modified by the introduction of further redundancy, to achieve line coding characteristics, thus becoming an ECLC. As noted previously, a line code aims to improve the reliability of a link. Cattermole [9] indicates that a baseband channel with a low frequency cut-off point cannot transmit arbitrary bit sequences unless there is some constraint on the line characteristics. If no such constraints exist, the performance of the communication link may eventually degrade to an unacceptable level.

The line coding characteristics of interest to this study, are the *running digital sum* (RDS) and the maximum runlength, RL_{max}. As a reminder, the RDS for a number x of transmitted words is defined as

$$RDS = \sum_{i=1}^{x} d_i,$$

where d_i is the disparity of each individual word. The *digital sum variation* (DSV) is defined as the difference between the largest and smallest values of RDS. Thus a good disparity limiting line code must place tight RDS bounds which will in turn reduce the DSV value. This is necessary if the low frequency content of the code is to be suppressed.

Another factor is the need to limit the maximum allowable number of continuous ones or zeros (the maximum runlength) since these can cause synchronisation problems at the receiver. Both the RDS and the runlength are important factors which may need to be bounded by an ECLC.

In addition, the overall rate reduction and decoding complexity must be kept as small as possible without significantly compromising the decoding performance. All of the above factors place tight constraints on the code design. The ECLCs presented here attempt to satisfy as many of these constraints as possible.

Figure 6.3 presents a tree diagram of the codes that will be introduced in this chapter. These can be divided into two main categories: the $N = 2$ codes whereby two EC code words are coupled and transmitted as one, and the $nB1X$ family of ECLCs. The latter is sub-divided in to the bi-modal ($nB1I$ and $nB1DR$) and non bi-modal codes. The non bi-modal codes do not require the flag bit for decoding, which can therefore be ignored. The $nB1DR$ can not be easily modified to become an ECLC without the use of a look-up table. For this reason, it is not examined any further. Finally the $nB1I$ code can become an ECLC by the use of the *enhanced flag protection* (EFP) code or the *cascaded added bit* (CAB) family of codes. All of the above ECLCs will be presented in more detail in the following sections.

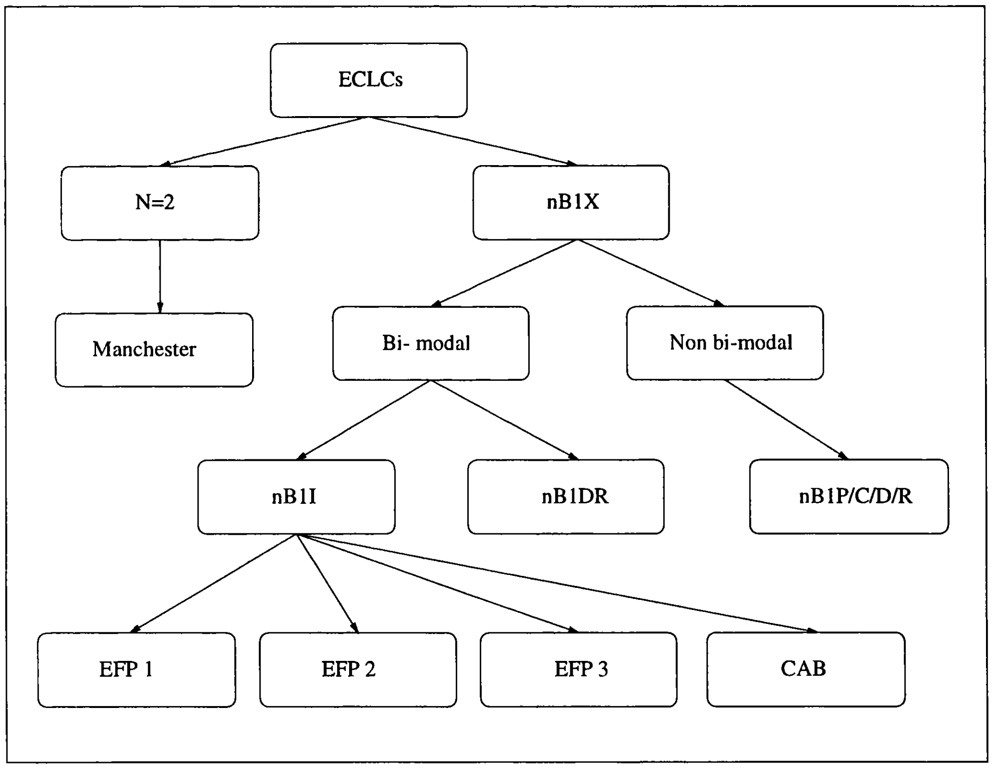


Figure 6.3: Tree diagram of the presented ECLCs.

6.5 The $nB1X$ Family of ECLCs

A well-known class of block line codes are the $nBmB$ as described by Smith [10] which convert a block of n binary bits into m binary bits. If m is chosen to equal $n+1$ then these codes are termed $nB1X$ and they were presented in chapter 3. In this section the $nB1X$ family of line codes will be re-examined and modified so that they exhibit error correcting characteristics. This is achieved by considering the n information bits of the $nB1X$ code as being error correcting code words in their own right. Therefore, k information bits are initially encoded into an n-bit word, using a suitable 'parent' ECC. In this study, BCH codes will be used as 'parent' codes and therefore n will be assumed as being odd.

The non bi-modal members of the $nB1X$ family do not exhibit error extension. This is because the line coding redundant bits are not required at the receiver and are thus discarded, without affecting the word. The bi-modal codes, i.e. the $nB1I$ and $nB1DR$, are affected by error extension. Each member of the $nB1X$ family of ECLCs will now be examined in more detail.

6.5.1 The $nB1P$ ECLC

The $nB1P$ line code is an added bit code whereby the extra bit is a parity bit. In order to offer error correcting properties a parent (n, k) BCH code is initially used. k information bits are therefore encoded into n code word bits which are then encapsulated in the $nB1P$ line code generating an $(n+1, k)$ code. As explained in chapter 3, if runlength limiting properties are required, then odd parity must be used. In such a case the maximum runlength will be equal to $RL_{max} = \frac{3n+1}{2}$.

The use of odd parity will not place bounds on the RDS. The $nB1P$ ECLC can be modified so that both the RDS and RL_{max} are bounded, but this can only be applied to specific codes such as the Hamming and BCH $(7,4)$ codes. This is achieved by the application of even parity and the elimination of the all-one and all-zero code words. For example, if the BCH $(7,4)$ code is used, the code word disparity will be equal to zero in all cases, except for the all-one and all-zero words. The alphabet for such a code, using a parent $(7,4)$ ECC which results in a $(8,4)$ code, is shown in table 6.1.

The major disadvantage of this code is that the all-zero and one words must be removed, otherwise the disparity and runlength will not be bounded. The remaining 14 possible messages can not be formed from all possible combinations of 4 bits. In order to avoid the need for complicated look-up tables only 3 information bits can be used, thus resulting in a $(8,3)$ code of rate $R = (\frac{3}{8}) = 0.37$. Such a code will have zero DSV and bounded maximum runlength.

Computer simulation of the $(8,4)$ code has proven that it has similar error correcting capabilities as the parent $(7,4)$ BCH code. This is because the extra parity bit can only be used to validate the accepted word after hard or soft decision decoding has taken place. This feature alone would offer a slight increase in the decoding performance of the code. The residual error however, might be on the parity bit itself which could result in invalidating a correct word, thus reducing the performance. A slight improvement in performance can be achieved if soft decision decoding is utilised.

The above algorithm can also be applied using any parent ECC. However, figure

Information word	$(8,4)$ ECLC code words	Disparity
0000	0000 000 0	-8
0001	0001 011 1	0
0010	0010 110 1	0
0011	0011 101 0	0
0100	0100 111 0	0
0101	0101 100 1	0
0110	0110 001 1	0
0111	0111 010 0	0
1000	1000 101 1	0
1001	1001 110 0	0
1010	1010 011 0	0
1011	1011 000 1	0
1100	1100 010 1	0
1101	1101 001 0	0
1110	1110 100 0	0
1111	1111 111 1	+8

Table 6.1: $(8,4)$ Parity code words, together with their corresponding disparities.

6.4 indicates that with increasing values of n, the percentage of code words with RDS values within ± 1 are significantly reduced. This implies that the significance of any single flag, non-bimodal code is also reduced, due to the inability of a single bit to affect the disparity. A significant amount of zero disparity words will only occur if the $(7,4)$ BCH or Hamming codes are used, because their RDS values range between ± 1 and can thus be reduced to zero by a single bit. In view of the inability of this technique to be applied to any code, i.e. the lack of generality, it is not investigated any further here.

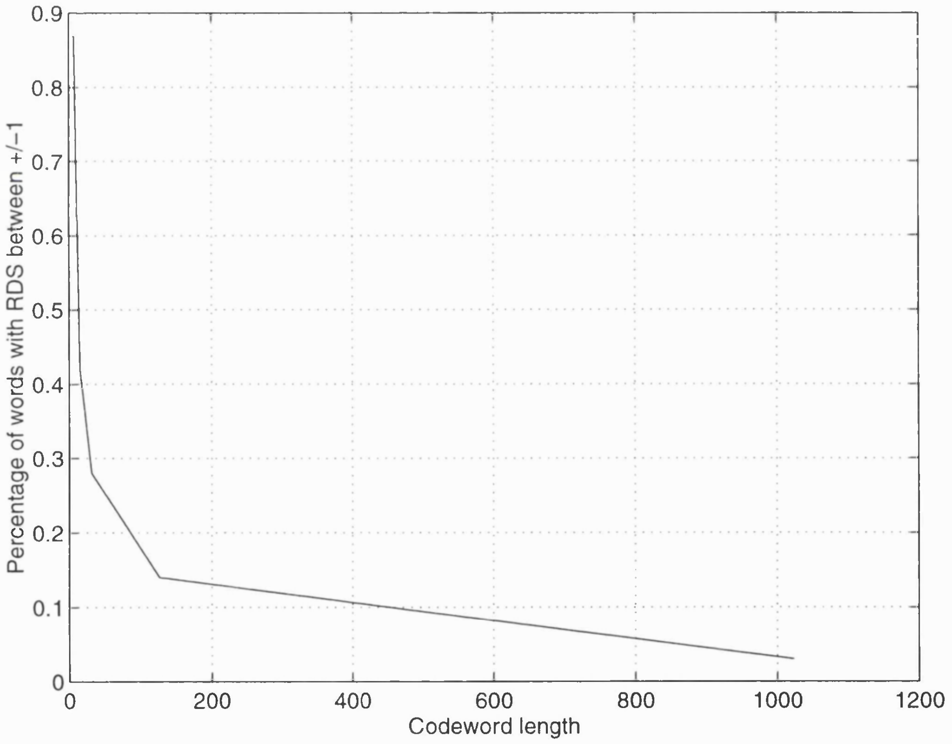


Figure 6.4: Percentage of code words with RDS values within ± 1 versus code word length n.

6.5.2 $nB1C$, $nB1D$ and $nB1R$ codes

The $nB1C$ is a very simple code that introduces runlength bounds to an ECC. Once more, the information bits are initially EC encoded using a parent ECC. They are then line encoded by adding an extra bit at the end of the word. The appended bit ($v_{(n+1)}$) has the inverse value of the $n - th$ bit, i.e. $v_{(n+1)} = \overline{v_n}$. The inclusion of this extra bit has the effect of reducing the runlength to $RL_{max} = (n + 1)$. The disparity however, is not bounded.

The $nB1D$ and $nB1R$ codes are very similar in operation. In the $nB1D$ code, the added bit value is determined by the disparity of the code word, while in the $nB1R$ code the appended bit is determined by the running digital sum of the code words transmitted so far. In both cases the added bits aim to reduce the disparity or the RDS respectively.

Computer simulation has proved the expected theoretical results. Specifically, in terms of decoding power, all three codes have similar performance to the conventional BCH code. This is because the extra bit is only used to introduce LC properties without otherwise affecting the ECC. In terms of line coding characteristics, all of the above codes have exactly the same performance as their corresponding line codes. These can be found in tables 3.3 and 3.4 of chapter 3.

As before, a slight reduction in the code rate is present, reducing R to $\frac{k}{n+2}$ which is insignificant, especially for long code lengths. In addition, SDD can be used since the only function of the line decoder is the removal of the appended bit.

6.6 Bi-modal ECLCs

In this section, the bi-modal members of the $nB1X$ family of ECLCs will be introduced. These use a 'split dictionary' where each code word has two alternative mappings each of opposing disparity. These offer tight runlength and disparity bounds but suffer from error extension. The first ECLC using this technique is the $nB1I$ ECLC, introduced by Popplewell in [6, 11] and presented here as an example. The basic concept behind this code is then expanded and the *enhanced flag protection* and *concatenated added bit* (CAB) algorithms are introduced. These offer significant decoding performance improvements without increasing complexity.

The $nB1I$ ECLC algorithm can be briefly described as follows:

1. A 'parent' ECC is initially selected and the code words are constructed. In this case, a BCH error correcting code is used.

2. The disparity of each code word is calculated.

3. The code words are then arranged into dictionaries in such a way that the RDS is bounded.

4. The first bit of the code word is used as a 'flag' to indicate which dictionary has been used.

5. At the receiver, conventional error correction decoding takes place using either hard or soft decision decoding techniques.

6. The flag bit is then used to determine which dictionary will be used for line decoding.

The reason behind the above procedure is that, generally speaking, EC code words are not of zero disparity. A bi-modal code is therefore required which means that each information word must have alternate mappings of opposite disparity, if the DSV is to be kept as low as possible. The placement of alternative code words into dictionaries is effected in such a way that the first information bit of the transmitted code word serves as a flag indicating which of the two mappings is utilised. The word is then transmitted and conventionally decoded for error correction. It is then passed through the line decoder which uses the 'flag' bit to determine the appropriate dictionary which is to be used for line decoding. Finally, the flag and parity bits are discarded while the remaining bits form the received information word.

The $nB1I$ ECLC has bounded disparity, runlength and a rate of $R = (\frac{k-1}{n})$, while retaining most of the error correcting power of the 'parent' code. The power spectrum of this ECLC is presented in figure 6.5, obtained using the Cariolaro and Tronca algorithm. [12]. As expected, the bounded RDS implies a zero DC content.

A simple (7,3) ECLC

As an example consider a single error correcting $(7, 4)$ BCH 'parent' code of rate $R = \frac{4}{7} = 0.57$. In order to place bounds on the RDS the available code words are divided into two groups. Each contains opposite disparity words, as shown in Table 6.2, where the numbers in the brackets indicate the disparity of each individual word. The first bit in each code word indicates which dictionary is used; if it is a zero then no inversion has taken place, otherwise the word is inverted. This results in a $(8, 3)$ ECLC of rate $R = \frac{3}{7} = 0.43$.

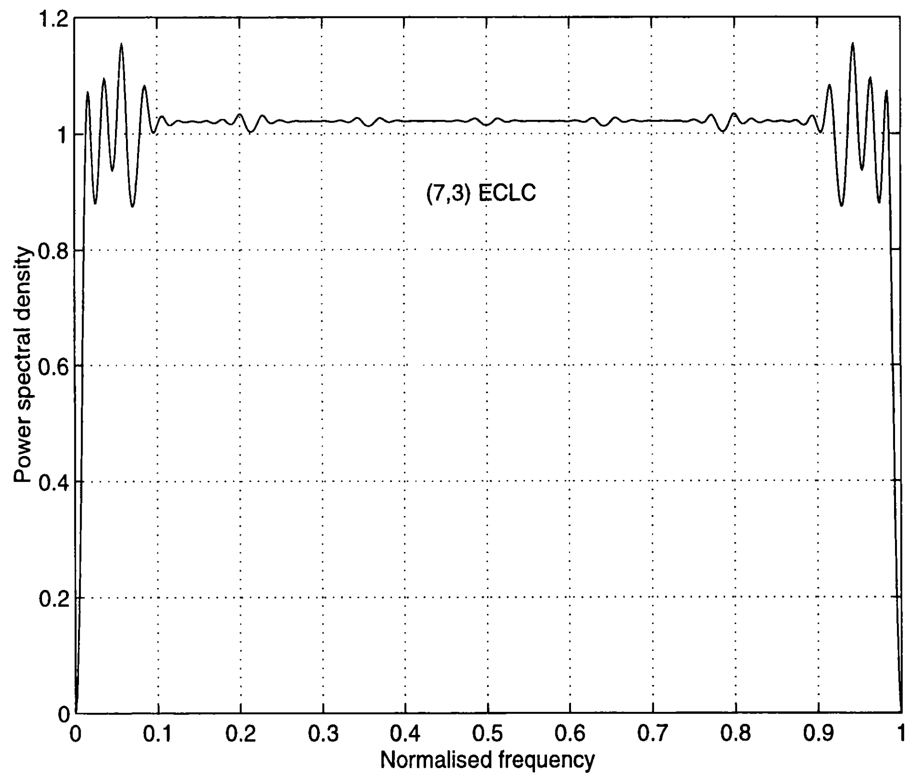


Figure 6.5: Power spectrum of a $(7, 3)$ ECLC.

Information	Code words	
words	$-7 \leq Disparity \leq -1$	$0 \leq Disparity \leq 6$
000	0000 000 (-7)	1111 111 (+7)
001	0001 011 (-1)	1110 100 (+1)
010	0010 110 (-1)	1101 001 (+1)
011	0011 101 (+1)	1100 010 (-1)
100	0100 111 (+1)	1011 000 (-1)
101	0101 100 (-1)	1010 011 (+1)
110	0110 001 (-1)	1001 110 (+1)
111	0111 010 (+1)	1000 101 (-1)

Table 6.2: Dictionary arrangement for a (7,3) BCH ECLC. The numbers in the brackets indicate the disparity of each code word.

If, for example, the all-zero information vector 0000 was to be continuously transmitted without any concern about disparity or runlength, it would be encoded as 0000000 using a standard BCH $(7,4)$ encoder. This would result in a continuous stream of zeros at the receiver (assuming no errors have occurred) which would cause timing recovery problems and unbounded disparity. The code rate in this case would be $R = \frac{4}{7} = 0.57$.

A $(7,3)$ ECLC is now created by dividing the complete set of code words into two halves of opposing disparities and using the first information bit as a flag to indicate which part of the table is selected. The resultant rate is equal to $R = 0.43$.

Using the above example, if a stream of zeros is to be transmitted, the first three information bits (as opposed to four information bits used in the 'parent' code) will still be encoded as an all-zero vector causing the RDS to become equal to -7. Since $RDS < 0$, the second set of three zeros will use the alternate mapping and will be encoded as the all-one vector. The disparity of this second word will therefore equal $+7$ and the RDS will now become $(-7 + 7) = 0$. The third set of information bits will be encoded as the all-zero word and the whole process will be repeated. Such a code offers bounded disparity and limited runlength.

Note that the first bit (v_0) indicates whether the rest of the code word bits are inverted or not. If this is in error then all the other information bits will be in error as well, i.e. error extension is now present. A modified algorithm is therefore introduced which offers enhanced protection to the sensitive flag bit.

6.7 Enhanced Flag Protection

The main drawback of the previous algorithm is the presence of error extension. This can be reduced by offering extra protection to the flag bit, given its critical role in correctly decoding a code word. In this section, a novel algorithm is presented, termed *enhanced flag protection* (EFP) [13, 14] which when applied to the $nB1I$ ECLC significantly reduces the effects of error extension. The EFP algorithm offers maximum benefit when used in conjunction with SDD.

The EFP algorithm is a generalisation of the ECLC presented in the previous section and increases the reliability of the flag of an $nB1I$ code. This is achieved by sending the flag bit v_0 twice. The added bit (v_{-1}) can either be made equal to the first bit, i.e. $v_{-1} = v_0$, or alternatively, equal to the complement of the first bit, i.e. $v_{-1} = \overline{v_0}$. The advantage of the latter implementation is that the flag and its copy will always be of opposite values, so their combined effect on the RDS will be equal to zero. In addition, the maximum runlength will be limited to a maximum of $RL_{max} = (n+1)$. Depending on the error rate required and the quality of the link, there are three possible implementations of such a code:

1. In the first implementation, EFP 1, both the flag and its copy are placed 'outside' the code word. This will create a $(n+2, k)$ code of rate $R = (\frac{k}{n+2})$. Protection to the flag bits is offered by repetition and at the receiver the two analogue values for the flags are averaged before the result is accepted as the correct flag. Such a code would be suitable for high SNR communication channels.

2. An alternative implementation, EFP 2, consists of increasing the code length from n to $(n+1)$ so that only the flag itself is accommodated within the code word. In such a case, protection to the flag bit is offered partly by the ECC and partly by repetition. The rate is now decreased to $R = (\frac{k-1}{n+1})$, but the effects of error extension are suppressed more, compared to EFP 1. The maximum runlength is equal to $RL_{max} = n$. At the receiver, the analogue values of both bits will be averaged and used to replace the analogue value of the flag bit within the code word.

3. Finally, both flag bits can be placed 'inside' the code word and thus replace two information bits. This is termed EFP 3. It will result in a $(n, k-2)$ code of rate $R = (\frac{k-2}{n})$. This code is suitable for poor quality channels as the flag bits are protected both by repetition and by the ECC. The maximum runlength

is reduced to $RL_{max} = n - 1$. The $nB1I$ ECLC presented in the previous section belongs to this category, but uses a single flag. At the receiver, the analogue values of both flag bits will be averaged and the resultant value used to replace the analogue values of both flags. SDD will then take place.

A detailed step-by-step presentation of the EFP algorithm using an example parent code is now presented.

6.7.1 An example EFP code

Once more, a $(7,4)$ parent BCH code is used as an example. This is modified so that it becomes an EFP ECLC which offers improved decoding performance and line characteristics. The modified encoding algorithm is as follows:

1. Use a 'parent' ECC to construct the code words.

2. Calculate the disparity of each code word.

3. Arrange the code words into dictionaries in such a way that the RDS is bounded.

4. Select a code word according to the information word and the RDS state. Depending on which of the three schemes is selected, the flag bits are calculated and applied. These will indicate which dictionary has been used.

5. At the receiver, the average analogue values of the two flag bits are calculated and used to determine the new most likely flag value, replacing the values of any flags within a code word. The code word is then EC decoded. Optimal

protection for the flag bits is therefore achieved regardless which of the three implementation schemes is used.

6. Finally, the line decoder selects the appropriate code word before the parity and flag bits are discarded.

Therefore by using the EFP algorithm the flag is in effect sent twice. At the receiver before any ECC decoding takes place, the two received analogue values representing the flag will be averaged. The resultant analogue flag value will thus be equal to $r_1 = (\frac{r_1 - \overline{r_1}}{2})$, which reduces the error extension, because the noise content of the flag bit is halved. The code word can then be decoded using any suitable algorithm. Finally, the line decoder will select the most likely code word from the dictionary based on the average value of the flag.

As a further minor modification, the two complementary flag bits can be distributed within the code word. This is achieved by placing the second (inverted) copy of the flag at a distance of $(\frac{n}{2})$ from the beginning of each word. The all-zero and all-one code word strings are now interrupted by the placement of the second flag in the middle. By time spreading the flag bits the probability of both of them being affected by a burst of interference is greatly reduced.

In the following section, a novel family of ECLCs will be introduced, termed *concatenated added bit* (CAB).

6.8 CAB Algorithm Description

The *concatenated added bit* ECLC is based on the $nB1I$ added bit ECLC. The latter utilises a 'parent' ECC to encode k information bits into n code word bits, with a t error correcting capability. The parent code can be any linear transparent code, to ensure that every code word has its inverse which is also a valid code word. Line coding properties are then achieved by inverting code words according to the RDS and code word disparity. Inversion is indicated by using a flag bit. If the flag equals zero no inversion has taken place, otherwise the code word has been inverted. However, if at the receiver the flag bit is erroneously decoded then all the code word bits will be in error.

In the CAB algorithm, an (n, k) error correcting code is used to form a $n \times (k+1)$ matrix, see figure 6.6. The initial k rows of this matrix contain k conventional EC code words. The k information bits (Info 1 to Info 4) of each code word are EC encoded into n code word bits, the flag bit is set to zero, and the disparity (d_i) of each code word and its associated flag is calculated. This is compared to the *running digital sum intermediate* (RDSI) variable which is the current value of the RDS. RDSI is updated every time a complete code word and its associated flag are generated, as opposed to the RDS which is updated once the complete matrix is transmitted.

If $d_i \times RDSI > 0$ then all the code word bits are inverted and the flag bit is set to one. If $d_i \times RDSI \leq 0$ then the flag bit remains a zero and no inversion takes place. This process is repeated k times which results in the generation of k flag bits (Flag 1 to Flag 4). These are then conventionally encoded into n bits thus forming the $(k + 1)$ row.

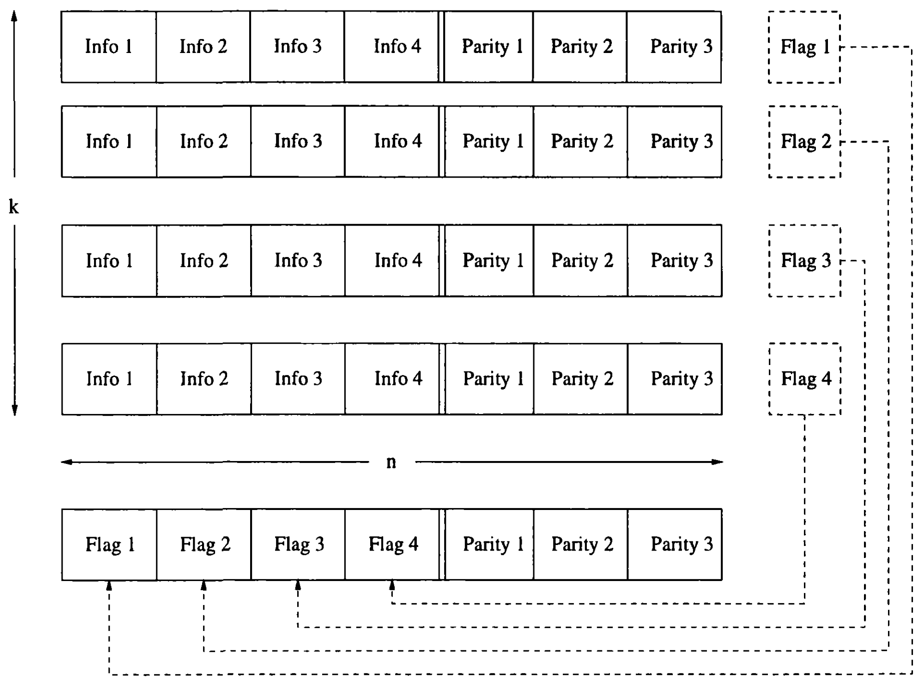


Figure 6.6: Cascaded added bit ECLC matrix.

EC encoding the flag bits ensures that some degree of protection exists, thus minimising the presence of error extension. The resultant matrix can then be transmitted either column by column (interleaved) so that the flag bits are separated in the time domain, or non-interleaved if tight runlength and disparity bounds are necessary. Interleaving is performed so that a possible burst of noise or interference will not affect all the flag bits, thus minimising the effects of error extension but at the expense of tight line coding bounds.

In our example, we have used an (n, k) BCH ECC where n is an odd number. With the introduction of the flag bit it is possible to generate zero disparity words. These can cause the code to exceed its disparity bounds since their inversion does not affect the disparity. If such words have occurred, an extra step is added to the algorithm, whereby the overall disparity (od_i) is calculated at the completion of the matrix. If $od_i \times RDS > 0$ then the complete matrix is inverted.

In order to place tighter disparity bounds, if a code word and its associated flag have zero overall disparity and the RDSI is positive then inversion of the code word will take place. The flow diagram of the CAB algorithm is shown in figure 6.7.

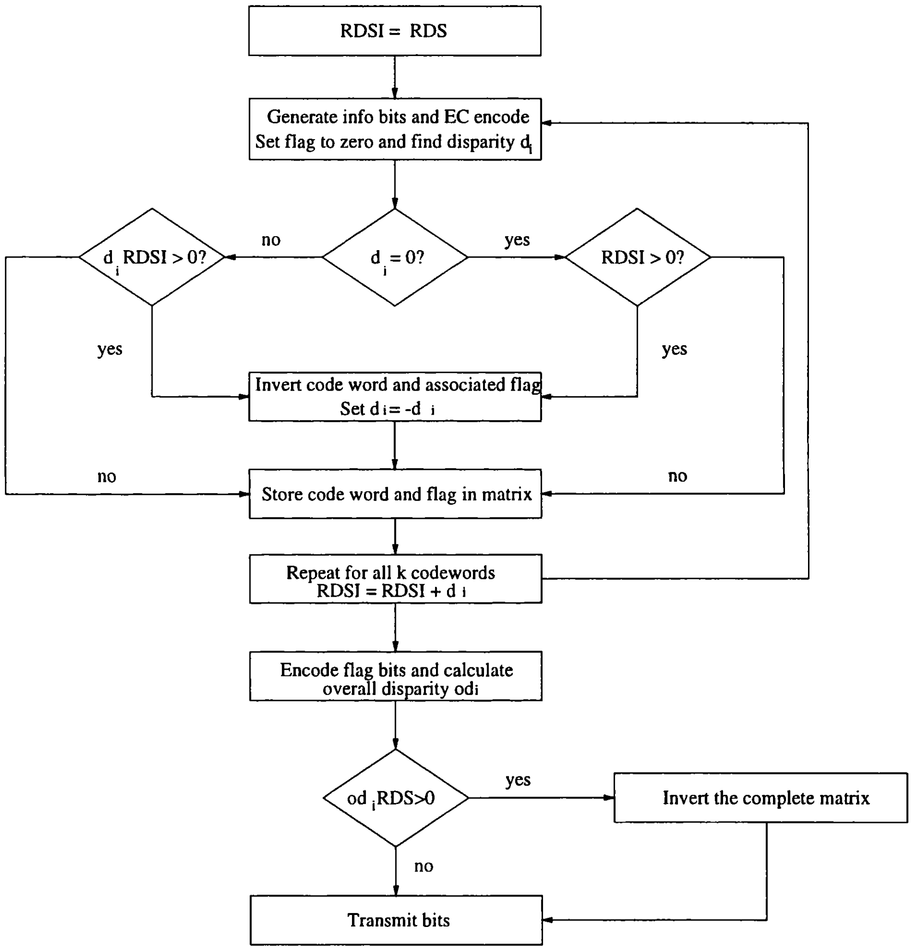

RDSI = RDS
Generate info bits and EC encode
Set flag to zero and find disparity d_i
d_i RDSI > 0?
no
d_i = 0?
yes
RDSI > 0?
yes
yes
Invert code word and associated flag
Set d_i = -d_i
no
no
Store code word and flag in matrix
Repeat for all k codewords
RDSI = RDSI + d_i
Encode flag bits and calculate overall disparity odi
od_i RDS>0
yes
Invert the complete matrix
no
Transmit bits

Figure 6.7: CAB flow diagram.

At the receiver, once the matrix is complete, each row of code words is conventionally decoded, using SDD if required. Then line decoding takes place, whereby each code word is inverted according to its corresponding flag. The parity and flag bits are then discarded, and the resultant information bits are accepted.

The CAB algorithm consists of k^2 information bits and $n \times (k+1)$ code word bits. The code rate is thus equal to $R = \frac{k^2}{n \times (k+1)}$. The disparity bounds are $(-2n - 1 + k) \leq RDS \leq (2n - 1 - k)$. The latter ensures a zero DC, a suppressed low frequency content and a bounded maximum runlength.

Depending on the quality of the communications link, the CAB algorithm can be modified to offer a variable decoding performance and rate. This is achieved by using a less powerful ECC (of higher rate)for high SNR applications, and vice-versa.

Finally, the CAB algorithm allows the use of SDD at the receiver. This offers improved decoding performance with a minimal increase in complexity.

6.8.1 CAB N algorithm

The CAB algorithm can be very easily modified so that the rate is increased without significantly affecting the runlength and disparity bounds or decreasing decoding performance. This is achieved by considering groups of N error control words as line code words and using a single flag to indicate inversion. Such a code is termed 'CAB N' ECLC and can be considered as the generalised case of the CAB code.

For example, if $N = 2$ then a group of two conventionally encoded n bit code words will use a single flag to indicate whether both are inverted or not, see figure 6.8. Thus, the k initial information bits (Info 1 to Info 4) are conventionally encoded, producing three parity bits (Parity 1 to Parity 3). The second set of information bits (Info 5 to Info 8) is also conventionally encoded, producing three further parity bits (Parity 4 to Parity 6). The disparity of both code words (and their associated flag) is then evaluated and depending on the current value of the

RDSI, inversion may then take place.

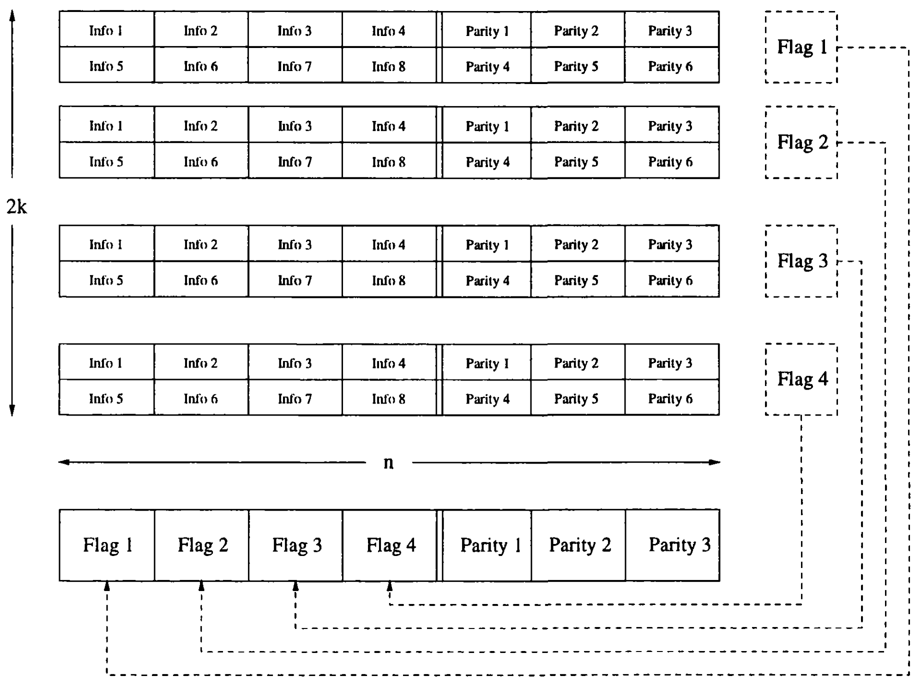


Figure 6.8: Cascaded added bit ECLC with grouped (N=2) code words.

Grouping excessive numbers of error correcting words together ($N > 3$) does not offer significant rate improvements while it adversely affects the maximum run-length and disparity bounds. Figure 6.9 presents the rate gain versus the group size N, for codes of different error correcting capability. The rate gain is defined as the fraction of the CAB N ECLC rate over the conventional CAB rate. From figure 6.9 it is clear that increasing the value of N over 2, does not offer a significant advantage in rate, to justify the reduced performance in terms of line coding characteristics. For this reason only $N = 2$ ECLCs will be considered here.

The CAB N rate is equal to $R = \frac{k^2 \times N}{n \times (N \times k + 1)}$ while it can be mathematically proven that the decoding performance of the CAB N code is exactly equal to that of the conventional CAB code (shown in the following section). Finally, the RDS can obtain any value between $(-(N+1)n - 1 + k) \leq RDS \leq ((N+1)n - 1 - k)$.

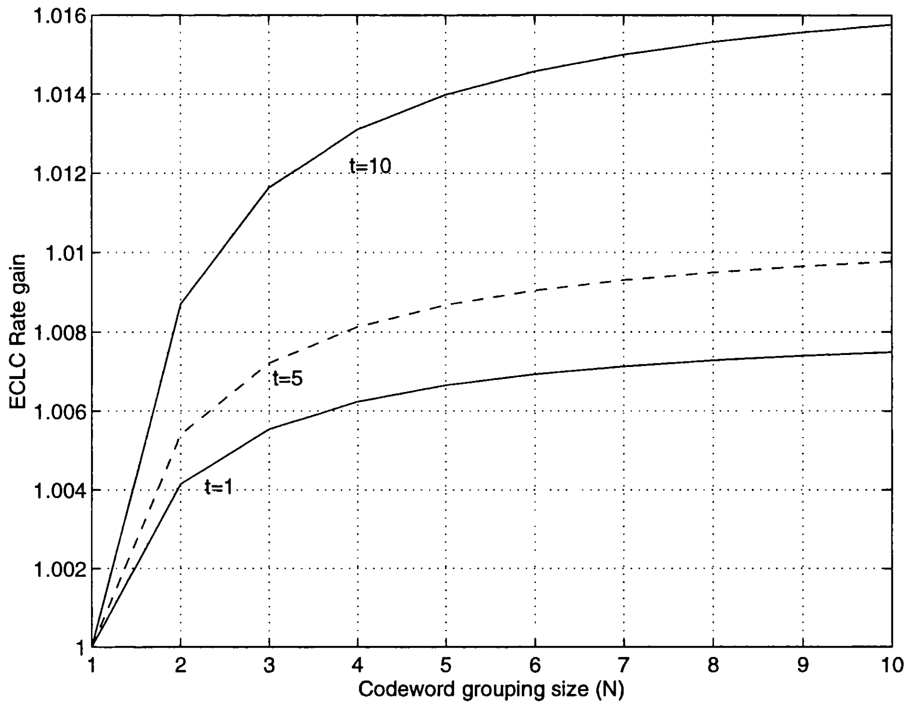


Figure 6.9: CAB ECLC rate gain for various group sizes, using a $n = 127$ BCH parent code with varying values of t.

The main advantages and disadvantages of the CAB family of ECLCs can be summarised as follows:

- SDD can be used at the receiver.
- Rate can be very good, especially for larger values of N.
- Decoding performance is the same as that of the $nB1I$ code, regardless of the value of N used.
- The CAB ECLCs are easy to implement.
- All of the above are achieved at the expense of very tight runlength and disparity bounds.

6.9 Simulation Results of the $nB1X$ ECLCs

Each ECLC presented in this chapter is assessed by using computer simulations. Three such programs will be discussed here:

1. The first program presents the decoding performance of the code in terms of error ratio versus signal to noise ratio [15, 16]. This was presented in detail in chapter 4. It should be noted that in the previous section, three alternative schemes were suggested for placing the added flag bit. Since all three will give similar decoding results, only one such scheme is simulated, i.e. the EFP 2 $(n+1, k-1)$ code.

2. The second program determines the line coding characteristics, such as maximum runlength and running digital sum, as presented in chapter 4.

3. Finally, the third one uses the Cariolaro and Tronca [12] algorithm to determine the power spectrum of each code, as presented in chapter 4.

The BCH (7,4) and BCH $(31, 26)$ single error correcting codes with rates $R = 0.57$ and $R = 0.84$ respectively, are used as examples.

It should be noted here that the simulation results on decoding power are not fully representative of a realistic situation. This is because the communications channel used is not limited in any way by effects such as low frequency cut-off and runlength. For this reason, the introduction of line coding only creates error extension without offering any performance benefit to the simulation.

Figure 6.10, demonstrates the decoding performance of a $(7, 4)$ $nB1I$ ECLC with HDD while the SD decoded version is presented in figure 6.11. The $(31, 26)$

code with SDD is presented in figure 6.12.

In the first two figures the lower line indicates the decoding performance of an $(8,3)$ EFP BCH code of rate $R = 0.37$, with double flag if HDD or SDD are respectively used. The top line shows the performance of a $(7,3)$ $nB1I$ ECLC of rate $R = 0.43$, where the first information bit is used as a flag. The middle line shows the performance of a conventional $(7,4)$ ECC of rate $R = 0.57$. The improved decoding performance of the soft decision decoded algorithm can also be observed by comparing the two figures.

Since the first bit indicates whether the rest of the information bits are inverted or not, the added protection for this sensitive data becomes more important for longer code lengths. This is reflected in the decreased performance of the $(31,26)$ code used in the second example, shown in figure 6.12.

The lower line indicates the decoding performance of a conventional BCH $(31,26)$ ECC, of rate $R = 0.84$ with no line coding properties. The middle line indicates the performance of a $(31,24)$ EFP ECLC of rate $R = 0.77$ which utilises a double flag, while the top line indicates the performance of a $(31,25)$ $nB1I$ ECLC of rate $R = 0.81$ which utilises a single flag.

In this example, the double flag code gives slightly degraded decoding performance results compared to the conventional code (which has no LC characteristics) due to the larger value of k. However, the reduction in rate due to the second flag is not as significant as it is for the smaller code lengths.

Generally speaking, the decoding performance of the double flag ECLC will be very similar to that of the parent ECC. Although the decoding performance of two

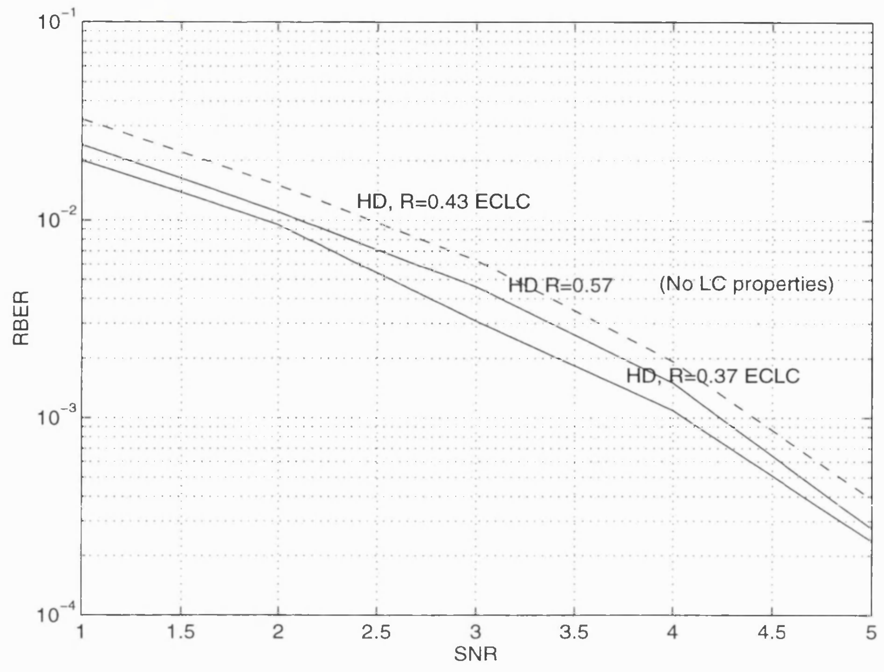


Figure 6.10: HDD performance using a parent BCH $(7, 4)$ ECC with and without ECLC properties.

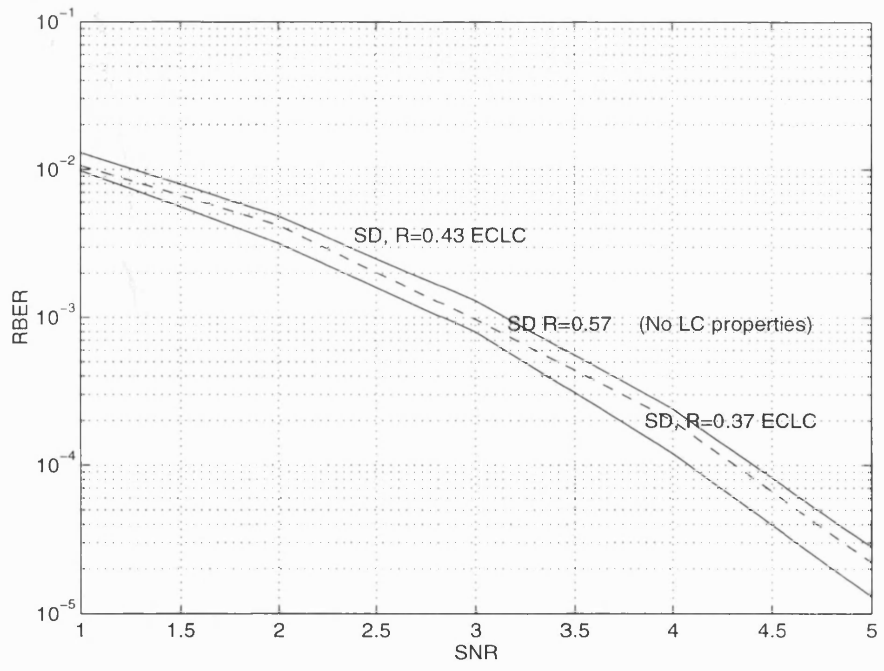


Figure 6.11: SDD performance using a parent BCH $(7, 4)$ ECC with and without ECLC properties.

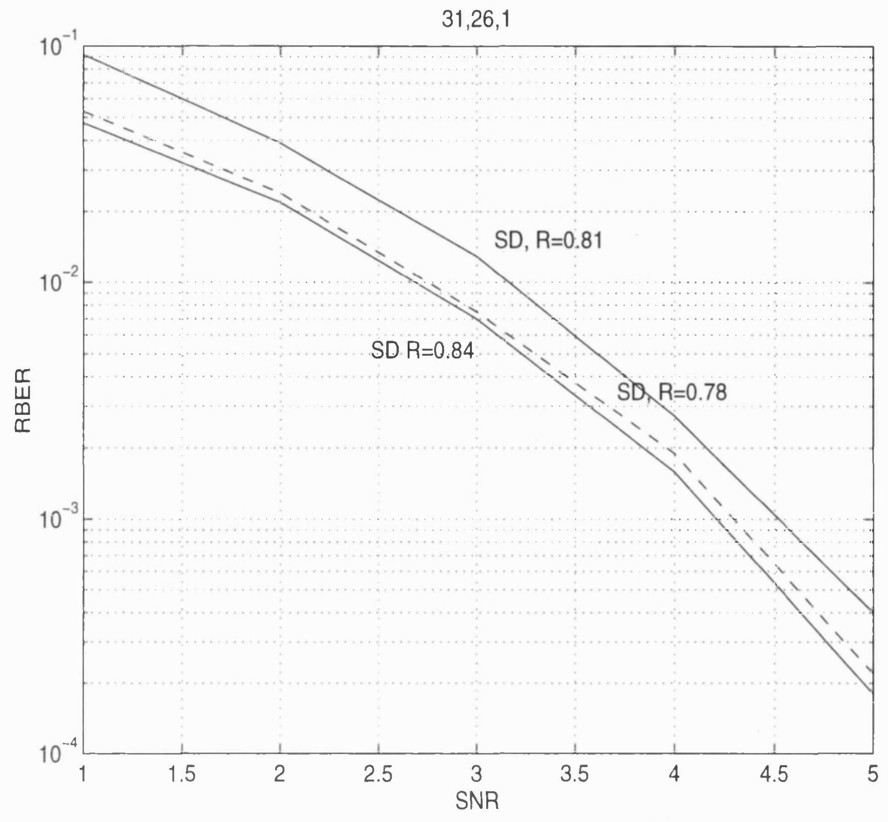


Figure 6.12: SDD performance using a parent BCH (31, 26) ECC with and without ECLC properties.

example BCH codes has been presented here, computer simulations indicate that similar results occur regardless of the code length used.

However, the line characteristics of the proposed code must also be examined: Since the disparity is bounded, a null in the power spectrum at DC is generated, see figure 6.13. In addition, the runlength characteristics are as shown in table 6.3. The EFP algorithm offers better performance that any of the other codes at the expense of rate, as can be seen from the third column of table 6.3.

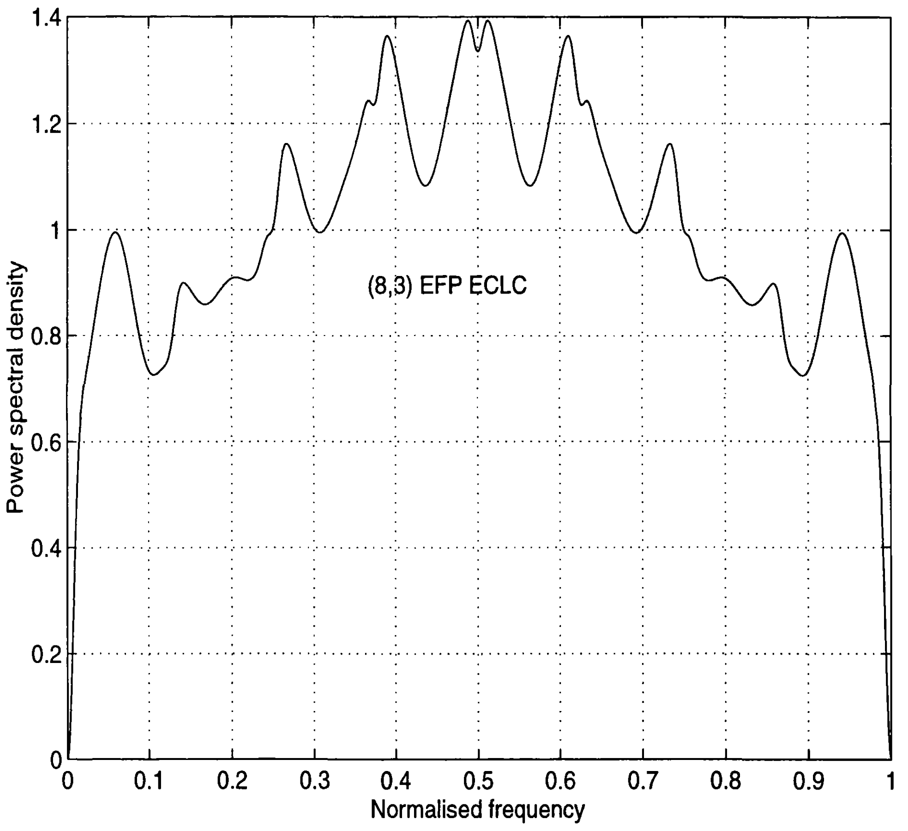


Figure 6.13: Power spectrum of a $(8, 3)$ ECLC.

Code Used:	Maximum Runlength	Rate
Parent ECC	Unbounded	$\frac{k}{n}$
Single flag ECLC	$2 \cdot n$	$\frac{k-1}{n}$
EFP algorithm 1	$n + 1$	$\frac{k}{n+2}$
EFP algorithm 2	n	$\frac{k-1}{n+1}$
EFP algorithm 3	$n - 1$	$\frac{k-2}{n}$

Table 6.3: Runlength performance and rates of various ECLC codes.

Finally, it should be noted that since the overhead of the extra bit remains constant regardless of code length, the EFP code is more efficient for longer code words. In such cases the reduction in rate is less significant which makes the ECLC more attractive. The decoding performance of the CAB family of ECLCs is presented in section 6.11.

6.10 N=2 ECLCs

In this section, the second main set of ECLCs is presented. These codes are termed $N = 2$ ECLCs, since two code words are coupled together and transmitted as a single word.

A problem in constructing ECLCs from odd code word length codes is that, by definition, no zero disparity code words exist. If BCH 'parent' codes are used then the code word length will be odd. Popplewell [17] has addressed this problem by transmitting groups of N code words together so that the overall disparity is bounded. It was also proved that using groups of more than two code words together does not significantly increase the rate, so only pairs of code words ($N = 2$) will be examined.

As an example, consider a BCH (n, k) parent code which will then be used to generate a $(2n, 2k-1)$ $N = 2$ code. The latter will have a large percentage of zero disparity code words and a reasonable rate $R = (\frac{(2k-1)}{(2\times n)})$. The algorithm for producing such a code can be divided in the following steps:

1. Generate k information bits and conventionally encode into n bits. This word will have a disparity of d_1.

2. Generate $k-1$ information bits, set the $k-th$ bit equal to 0 and conventionally encode to n bits. This word will have a disparity of d_2. The $k-th$ bit is used as a flag to indicate if both the first and second words are inverted or not.

3. Calculate the disparity (d) of both words, i.e. $d = d_1 + d_2$, transmit both words together and update the value of the RDS.

4. Repeat step 1 and 2 using the next set of information bits so that the third and fourth code words are generated, of disparity d_3 and d_4, respectively. If the sum of $d_3 + d_4$ has the same sign as the RDS then both words are inverted, otherwise they are transmitted. The process is repeated from step 1 for the next information bits.

At the decoder, the $k-th$ information bit of the second code word of each group is examined. If it is equal to 0 then the code words are normally decoded, otherwise they are inverted before decoding is effected. Thus the disparity and the runlength are both bounded while the reduction in rate is insignificant.

A variation of this algorithm was also introduced whereby the all-zero and all-one code words were not conventionally encoded but look-up tables were used instead. Thus tighter disparity and runlength bounds were achieved but the generality of the code was lost. For this reason, such codes were not considered any further.

A simple example of a $N = 2$ code

The $N = 2$ ECLC is made easier to understand if an example is presented. For this purpose, a $(7,4)$ BCH code is used as a parent code. It is assumed that the information bits consist of a row of identical symbols (e.g. a string of ones) which is a worst case situation in terms of line performance criteria (RDS and RL_{max}). This is because if a normal ECC were to be used, it would produce a continuous stream of ones which would cause both the RDS and RL_{max} to tend to infinity. A $N = 2$ code will thus create a $(14,7)$ code of rate $R = 0.5$ using the following encoding algorithm:

1. Encode the initial four information bits. The complete code word is now 1111 111 with disparity $d_1 = +7$.

2. Encode the next three information bits. The fourth information bit is used as a flag and is initially set to zero. The code word is then conventionally encoded, producing the code word 1110 010 with a disparity $d_2 = +1$.

3. The running digital sum is now equal to $RDS = (+7 + 1) = +8$ and both words are transmitted.

4. Steps 1 and 2 are repeated and since the information bits are a continuous stream of ones, the same two code words are generated. However, since the disparity of this second group of code words is once more equal to +8 and $RDS = +8$ both words are inverted. Thus the transmitted code words are now 0000000 and 0001101 with a disparity of −8 and the RDS is now equal to $RDS = (+8 - 8) = 0$. The $k - th$ bit of the second word is used as a flag and for this reason is always equal to zero unless both words are inverted. The runlength for the worst case (the all-zero information stream) is also limited to $RL_{max} = 14$.

At the decoder, if the $k - th$ bit of the second word is equal to one then both words are inverted and then decoded, otherwise they are conventionally decoded. The advantage of this algorithm is that line coding characteristics are achieved without requiring alternative mappings for each word and therefore without significantly reducing the rate of the code.

6.10.1 Improvements on the N=2 codes

The N=2 code offers good line characteristics together with a reasonable rate and is simple to implement. However similarly to the codes presented in the previous section, it also suffers from reduced decoding performance due to error extension.

Specifically, if the flag bit is in error then the information bits of *both* code words will be erroneously decoded. The use of SDD can offer a slight improvement in performance, but as before, it is much more practical if a second information bit is sacrificed to increase the resilience of the flag bit [18]. This is achieved by replicating and inverting it. With the use of SDD at the decoder the average values of both flag bits can be used, thus significantly reducing the possibility of erroneously decoding the flag bit and further reducing the runlength bounds. In addition each copy of the flag is protected by the corresponding EC word to which it belongs.

The improved $N = 2$ algorithm is once more demonstrated using the $(7, 4)$ BCH code as an example. A continuous stream of ones will be chosen as the information vector once more, since it highlights the differences with the previous code:

1. Encode the initial three information bits. The fourth information bit is used as a flag and is initially set to one. The complete code word is now 1111111 with disparity $d_1 = +7$.
2. Encode the next three information bits. The fourth information bit is again used as a flag and is initially set to zero. The code word is then conventionally encoded, i.e. 1110010 with a disparity $d_2 = +1$.
3. The resultant disparity is now equal to $RDS = (+7 + 1) = +8$. Both words are transmitted.
4. Steps 1 and 2 are repeated and the same two code words are generated. However, since the total disparity of this group of code words is once more equal to $+8$ and $RDS = +8$ then they are both inverted. The transmitted code words are now 0000000 and 0001101 with the total disparity equal to

-8 and $RDS = 0$.

Therefore by replicating the flag a significantly improved decoding performance is achieved. The penalty is a further reduction in rate which now equals $R = (\frac{2k-2}{2n})$ or $R = (\frac{6}{14}) = 0.43$, if the example code is used.

6.10.2 Simulation results for the N=2 codes

The simulation used in the previous section was modified to produce the results shown in this section. Specifically the $(14, 7)$ and $(14, 6)$ codes were simulated with both HDD and SDD and the results are shown in figures 6.14 and 6.15. The lower line in each figure indicates the decoding performance of a $(7, 4)$ parent BCH code if HDD or SDD are used. The top line shows the performance of a $(14, 7)$ code with a single flag while the middle line shows the performance of the $(14, 6)$ code with a double flag.

As expected, the decoding performance of the single flag N=2 code is worse than that of the parent code due to error extension. If however, the flag bit is replicated then the decoding performance is improved and almost matches the performance of the ECC. Similar results are present for various other code lengths.

6.10.3 Manchester ECLC

The Manchester ECLC presented in this section is a specific case of the $N = 2$ codes presented in the previous section. It is based on the idea that by transmitting two EC code words together, an even length code is generated which enables zero disparity words to be created. These do not require alternate mappings and thus

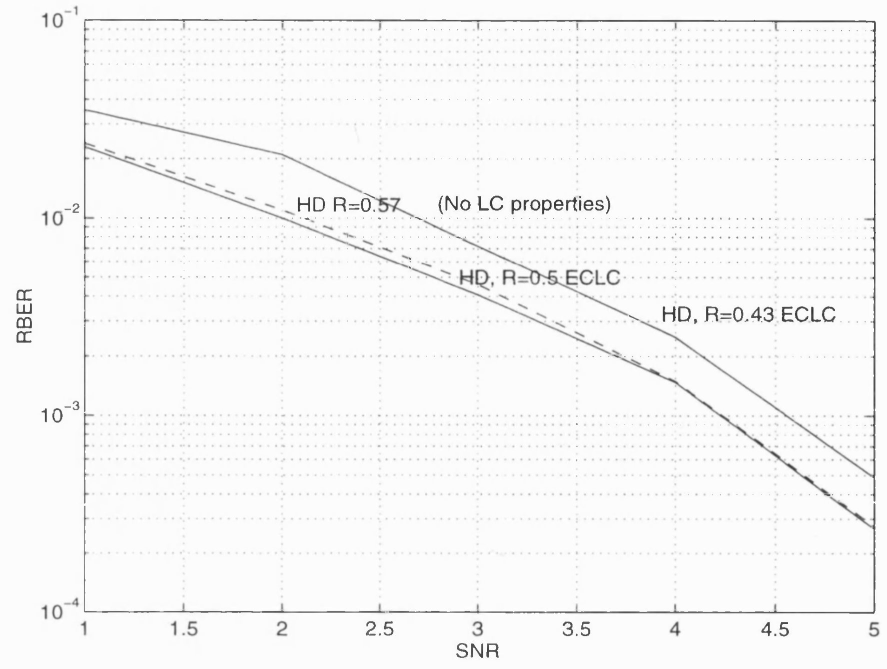


Figure 6.14: HDD performance of N=2 error correcting line coding code, using a (7, 4) parent ECC.

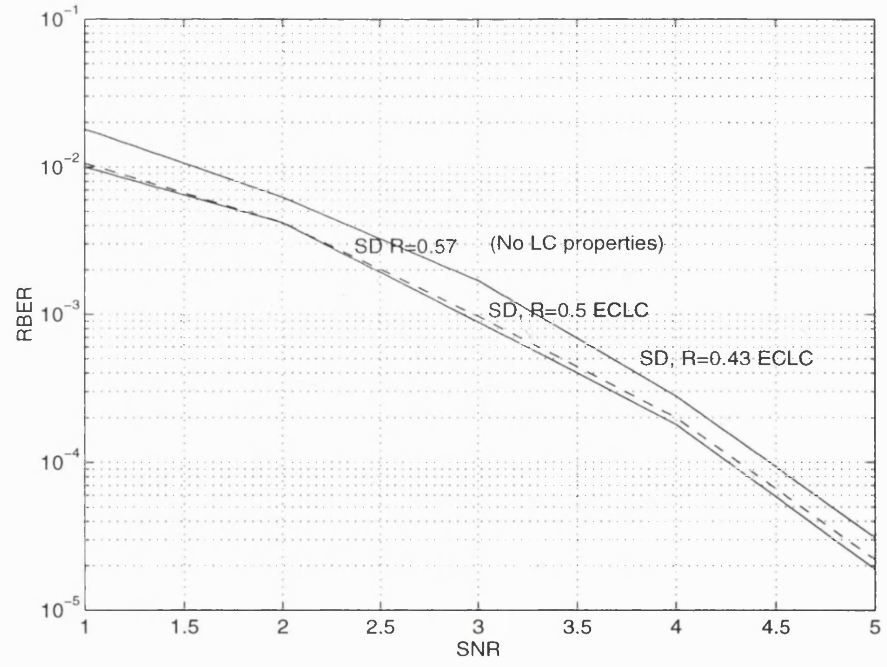


Figure 6.15: SDD performance of N=2 error correcting line coding code, using a (7, 4) parent ECC.

significantly simplify the coding operation. A simple and yet very powerful solution would be to transmit a code word together with its inverse.

There are two ways in which this could be done, either by appending the inverted bits at the end of each word or by interleaving them in the word itself, so that each bit is followed by its inverse (as performed in Manchester encoding). The advantages of this method are the very tight bounds of both the running digital sum and the maximum runlength achieved which in the case of the Manchester encoded word are $RDS = \pm 1$ and $RL_{max} = 2$. The disadvantage is the large reduction of rate; this is now halved since the 'parent' (n, k) BCH code will become a $(2 \times n, k)$ code. However, if SDD is introduced, the fact that each bit is transmitted twice can be used to achieve significant decoding performance gains. This is because at the receiver the two analogue values for each bit can be averaged and the resultant value taken as the accepted one. This technique will be termed 'Manchester SDD' [19]. Conventional SDD can then take place (if required) using the resultant analogue values to determine the least confident bits [20]. The new algorithm is presented in block diagram form in figure 6.16.

In order to theoretically explain why this decoding performance improvement is possible by replicating the code word bits and inverting them, a simple noise analysis is performed:

Assume that a bit v_i (where $1 \leq i \leq n$) and its complement $\overline{v_i}$ are transmitted through a communications channel affected by Gaussian noise. At the receiver $r_i = v_i + n$ and $\overline{r_i} = \overline{v_i} + n'$ are obtained, where n and n' are the independent Gaussian noise samples with σ being the standard deviation.

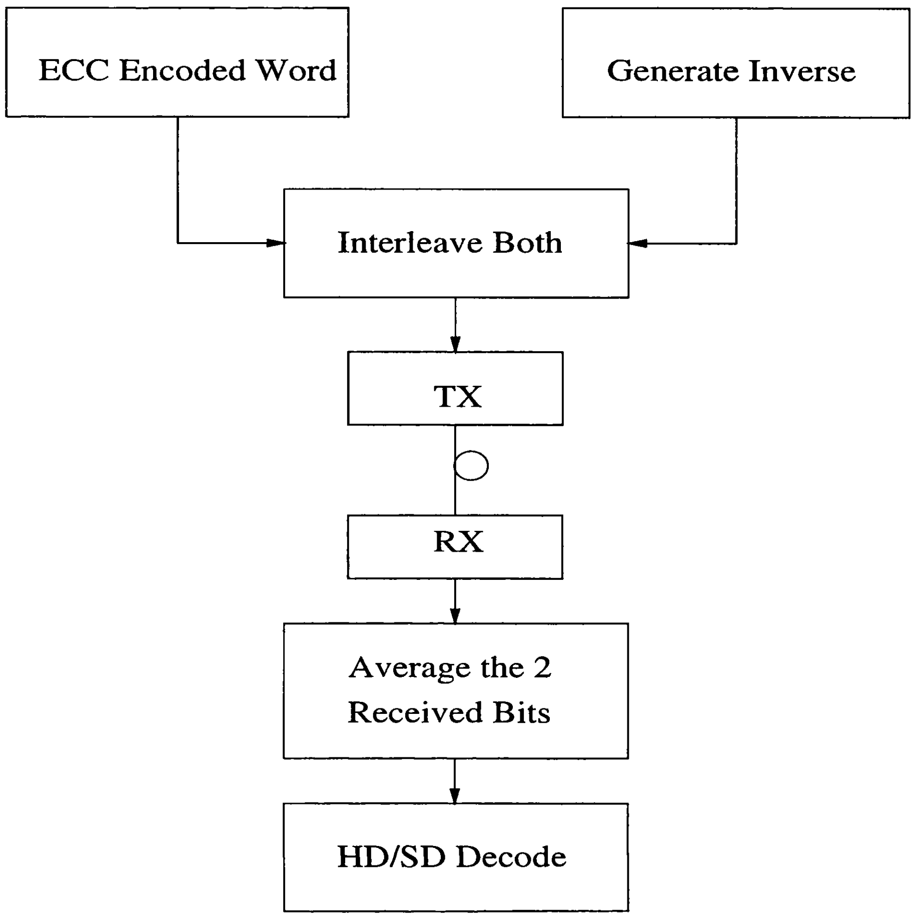


Figure 6.16: Block diagram of third algorithm.

At the receiver the two bits v_i and $\overline{v_i}$ are averaged thus providing us with an analogue value equal to

$$\frac{(v_i + n) - (\overline{v_i} + n')}{2} = \frac{2 \times v_i + (n - n')}{2}. \tag{6.1}$$

Concentrating on the noise vector only, $\frac{(n-n')}{2}$ has a standard deviation (SD) of

$$SD = \frac{\sqrt{2 \times \sigma^2}}{2} = \frac{\sigma}{\sqrt{2}} = \sigma'. \tag{6.2}$$

Now the ratio of $\frac{\sigma}{\sigma'}$ equals $\sqrt{2}$ therefore the noise reduction will be equal to

$$20 \times log\frac{\sigma}{\sigma'} = 10 \times log2 = 3dBs. \qquad (6.3)$$

It therefore has been proven that by averaging the analogue values of the received bits a maximum 3dB noise improvement is achieved. It should be noted here that as a general rule, SDD can achieve about 2dBs of coding gain. However the 2 dB improvement presented here and obtained through Manchester SDD is generated by replicating the code word bits before transmission. This allows us to use conventional SDD as well as Manchester encoding and have about 4 dBs of overall coding gain.

Simulation results of the Manchester codes

The computer simulator was further modified to generate both versions of the codes mentioned in the previous section, i.e. the Manchester code with the extra bits interleaved in the word and the variant with the second set of bits placed at the end of each word. A number of different code lengths were simulated and one is used here as an illustrative example.

In figure 6.17 the top line indicates the computed decoding performance of a BCH $(31, 26)$ single error correcting code of rate $R = 0.84$ with no LC properties. The middle line represents the Manchester encoded version of the same code which forms a $(2 \times 31, 26)$ code of rate $R = 0.42$ with very tight RDS and RL_{max} bounds. The significantly enhanced decoding performance of the latter code is also evident. Finally for comparison purposes, the lower line indicates the performance of a $(63, 30)$ 6 error correcting code. This has a similar rate $(R = 0.48)$ to the Manchester encoded one but offers no LC properties. In all three cases hard decision decoding was used, but similar relative results occurred if SDD was used.

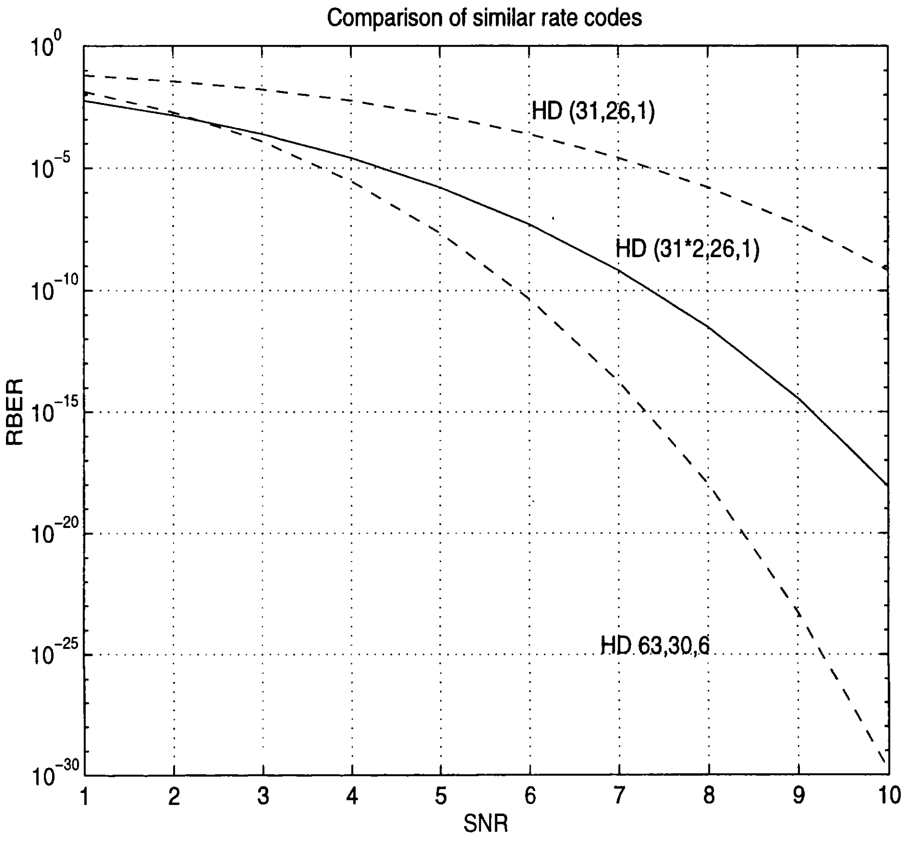


Figure 6.17: Decoding performance of the third algorithm.

The same soft decision decoding process can be used in the case where the complementary bits are all appended at the end of the word. In this case the bounds for both disparity and runlength are expanded. Specifically, the RDS values increase from $RDS = \pm 1$ to $RDS = \pm n$ while the maximum runlength from $RL_{max} = 2$ to $RL_{max} = 2 \cdot n$. The advantage of this algorithm is that for bursty noise channels greater separation between copies of the same bit in the time domain exists. It therefore becomes unlikely that a burst of noise will affect both copies, especially for long code word lengths.

Another possibility would be to transmit the 'raw' information data without using a 'parent' ECC. Even in such a case, most of the theoretical 2dB gain could still be recovered thus offering limited error correction capabilities and maintaining full line coding properties. Such a scheme would be very simple to implement and under suitable circumstances could offer acceptable decoding performance.

The power spectrum of the Manchester ECLC is shown in figure 6.18. The latter indicates no DC component and very good suppression of low frequencies, both of which arise from the very tight disparity bounds.

Using the third proposed ECLC algorithm it was demonstrated that significant line and error control coding properties can be gained, at the expense of rate. It therefore becomes clear that decoding performance can be effectively 'exchanged' for LC properties. The main advantages of this algorithm are the very simple nature of the encoding and decoding algorithm which can be used in either uncoded or hard/soft decision coded words, and the very good decoding and line coding characteristics present.

Up to this point, a number of existing and novel algorithms have been presented. In the following sections several aspects of the performance of these ECLCs, such as the calculated decoding performance and rate, will be examined in detail.

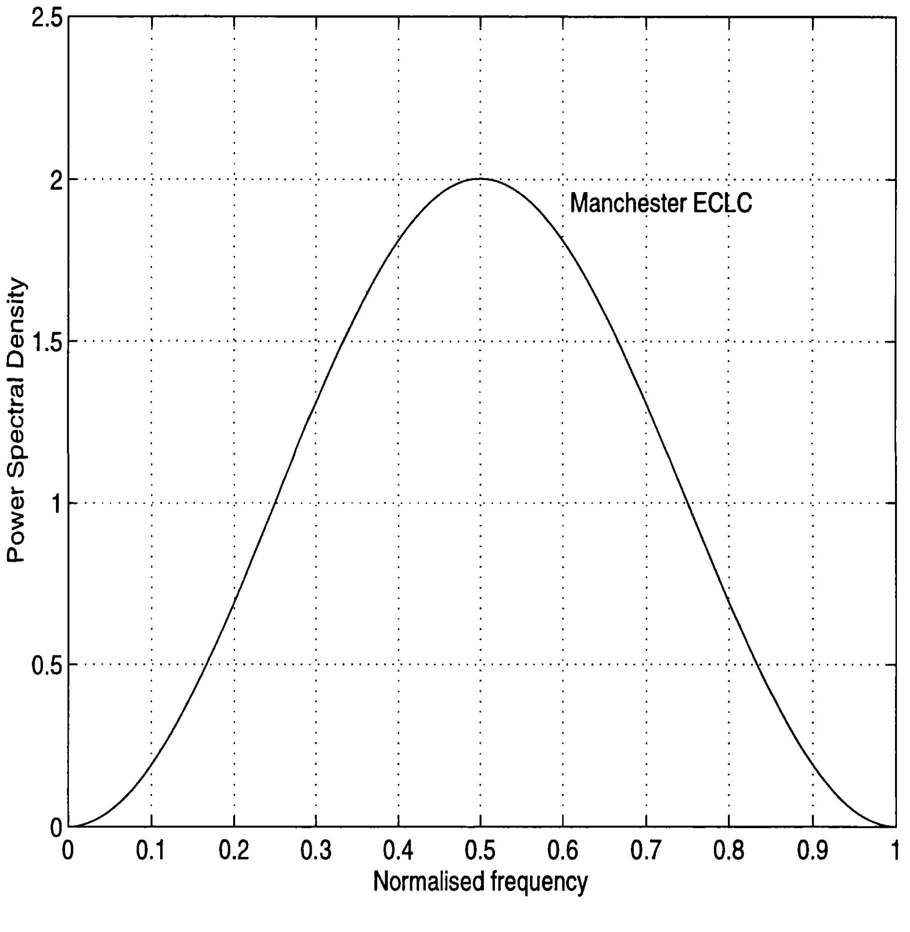


Figure 6.18: Spectral densities of the Manchester ECLC.

6.11 Calculating the Decoding Performance of the Single Flag Codes

The decoding performance of all the single flag, bi-modal codes can be very easily calculated. This applies to the $nB1I$, $nB1DR$, CAB, CAB N, and $N = 2$ ECLCs all of which have exactly the same decoding performance. In this section, the CAB code will be used as an illustrative example:

Assume that the parent (n, k) ECC is used to create a CAB ECLC consisting of $(k + 1)$ rows of n bits each. Since only the information bits of each code are of interest, all the parity bits can be ignored. The *residual bit error rate* of the CAB code ($RBER'$) can be expressed in terms of the *residual bit error rate* (RBER) of the parent code. By definition,

$$RBER' = \frac{Total\ number\ of\ info\ bits\ in\ error}{Total\ number\ of\ info\ bits}$$

Assume that the total number of information bit errors in the CAB code are e_t while the total number of information bits is equal to k^2. Therefore,

$$RBER' = \frac{e_t}{k^2} = \frac{e_1 + e_2}{k^2} \tag{6.4}$$

where e_1 indicates errors caused by inverting code words due to errors in their corresponding flags, and e_2 indicates conventional errors. By definition $e_t = e_1 + e_2$.

In order to calculate e_1, we assume that a number x of errors occur in the last $(k + 1)$ word which contains the flag bits. In such a case, $x = RBER \times k$, since the code words are encoded using the parent code. Therefore, x code words will be wrongly inverted, causing $e_i = x \times k$ bit errors in total. From this figure however, a number e_o of original errors (errors that pre-existed the inversion) must be subtracted, because they will now be corrected. Therefore, $e_1 = e_i - e_o$ and

$$e_i = k \times x = k \times (RBER \times k) = k^2 \times RBER \tag{6.5}$$

and

$$e_o = k \times x \times RBER = k \times (RBER \times k) \times RBER = k^2 \times RBER^2 \tag{6.6}$$

Subtracting equation 6.6 from equation 6.5, e_1 is determined to be

$$e_1 = e_i - e_o = (k^2 \times RBER) - (k^2 \times RBER^2) \tag{6.7}$$

The number of conventional errors in the remaining code words is equal to:

$$e_2 = (k - x) \times RBER \times k = k^2 \times RBER \times (1 - RBER) \tag{6.8}$$

Therefore the total number of errors can be determined by substituting equation 6.7 for e_1 and equation 6.8 for e_2 into equation 6.4,

$$e_t = e_1 + e_2 = (k^2 \times RBER) - (k^2 \times RBER^2) +$$

$$+(k^2 \times RBER \times (1 - RBER)) \tag{6.9}$$

Thus $RBER'$can now be obtained:

$$RBER' = \frac{e_t}{k^2} = \frac{2 \times k^2 \times RBER \times (1 - RBER)}{k^2} \Rightarrow$$

$$RBER' = 2 \times RBER \times (1 - RBER) \tag{6.10}$$

The $RBER$ of the parent code can be very easily simulated or calculated. Using equation 6.10 it is now possible to calculate the decoding performance of all the single flag bi-modal codes. Simulation results confirm the validity of the calculated results.

As an illustrative example the (127, 106) BCH parent code is used to form a CAB ECLC. The decoding performance of the CAB code shown in figure 6.19 with dashed lines, is slightly degraded compared to that of the parent code (shown in solid lines) due to error extension. The decoding performance of hard decision decoding is shown on top (HD), followed by the decoding performance using the Chase 2 algorithm and the maximum likelihood limit (ideal SD).

The decoding performance of all of the single flag added bit codes is exactly the same as that of the conventional CAB algorithm.

6.12 Calculating the Decoding Performance of the EFP Codes

The decoding performance of the EFP ECLC can be also very easily calculated. Once more we assume that the *residual bit error rate* (RBER) of the EFP code

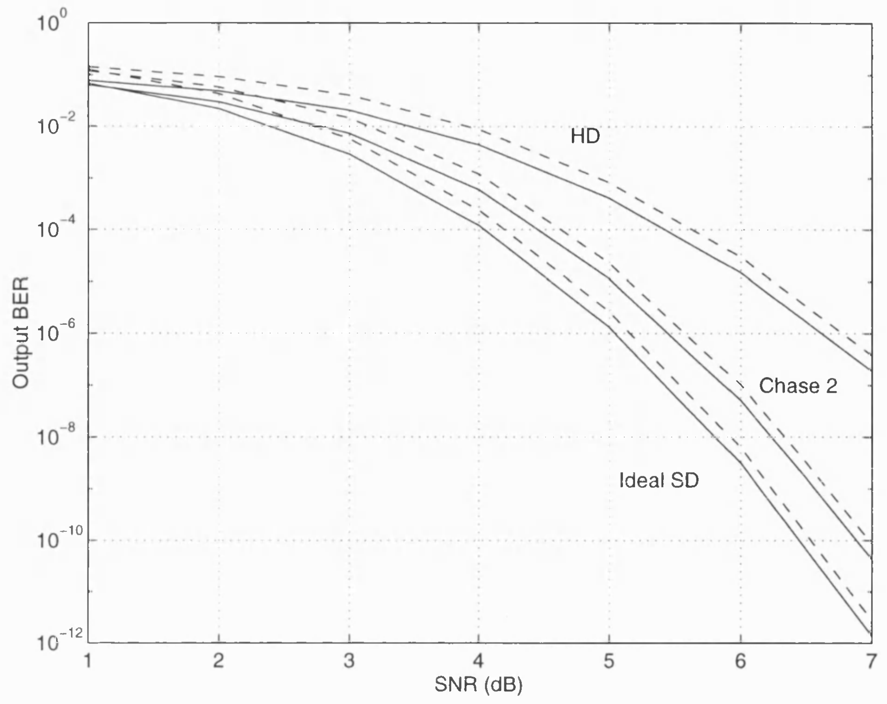


Figure 6.19: CAB decoding performance for a (127, 106) BCH code with SDD.

is termed $RBER'$, and that a total number x of codewords are transmitted. Of these x words, x_1 will have conventional errors and x_2 will have errors due to an incorrectly decoded flag bit. Once more, by definition

$$x = x_1 + x_2 \tag{6.11}$$

$$x_2 = \frac{RBER \cdot x}{2} \tag{6.12}$$

$$x_1 = x - x_2 = x - \frac{RBER \cdot x}{2} \tag{6.13}$$

The total number of errors e_t can be sub-divided into two parts, e_1 and e_2 which are the errors in x_1 and x_2, respectively. Therefore, substituting equations 6.12 and 6.13 in equations 6.14 and 6.16 we obtain equations 6.15 and 6.17.

$$e_1 = x_1 \cdot k \cdot RBER \Rightarrow \tag{6.14}$$

$$e_1 = k \cdot x \cdot RBER \cdot (1 - \frac{RBER}{2}) \tag{6.15}$$

$$e_2 = x_2 \cdot k - x_2 \cdot k \cdot RBER \Rightarrow \tag{6.16}$$

$$e_2 = k \cdot x \cdot RBER \cdot (\frac{1 - RBER}{2}) \tag{6.17}$$

The total number of errors is now equal to:

$$e_t = e_1 + e_2 = kx \cdot RBER \cdot (\frac{3 - 2RBER}{2}) \tag{6.18}$$

Finally, using equation 6.18, $RBER'$ becomes equal to:

$$RBER' = \frac{e_t}{k \cdot x} = RBER \cdot \frac{(3 - 2RBER)}{2} \tag{6.19}$$

We have therefore managed to calculate the performance of the EFP ECLC based on the RBER of the ECC. As an example, we apply formula 6.19 to a $(7, 3)$

ECLC and figure 6.20 is obtained. Clearly the calculated decoding performance equals the simulated decoding performance of the same code, shown in figures 6.10 and 6.11.

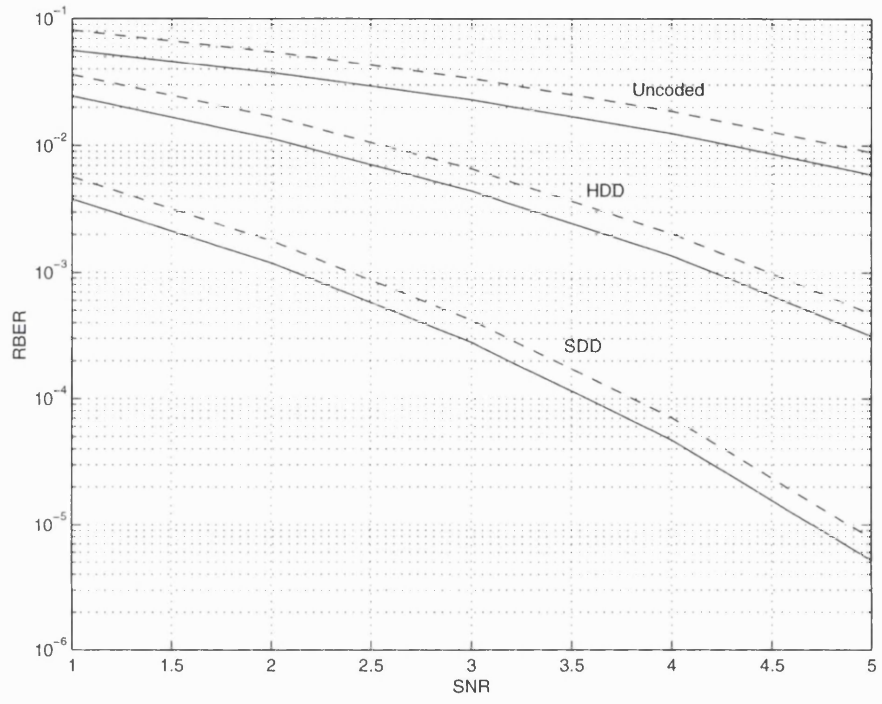


Figure 6.20: Decoding performance for a $(7, 3)$ EFP ECLC.

Similar results are present for other code lengths.

6.13 Rate Considerations

Any realistic system requiring the use of FEC or ECLCs is likely to have a defined error performance target. This would mean that maintaining a given decoding performance level (or residual bit error rate) would be paramount and the designer would have to work around it, should any other features be required. Applied to ECLCs this would be translated as the rate reduction required for introducing line characteristics while maintaining a given RBER. This is in contrast to the results presented so far, where the RBER was allowed to vary in value with each successive ECLC.

Figure 6.21 demonstrates two codes having on average similar decoding power but each with different properties. The dashed line indicates the performance of a $(127, 92)$ 5 error correcting BCH code of rate $R = 0.72$ which has no line coding characteristics. The continuous line indicates the performance of a $(63 \times 2, 57)$ single error correcting code which forms a $(126, 57)$ code of rate $R = 0.45$. This uses the Manchester encoding principle presented in section 6.10.3 and has very tight runlength bounds ($RL_{max} = 2$) and disparity bounds ($-1 \leq RDS \leq +1$). This figure demonstrates that for a given RBER there will be a reduction in rate from 0.72 to 0.45, i.e. a reduction of 62.5%.

If using figure 6.21 an arbitrary target RBER of $1.0e - 7$ is selected (which occurs at a SNR of 6dBs) a rate comparison can be effected for the other ECLCs.

Specifically, if a single flag bi-modal code is used, then a parent $(31, 11)$ seven error correcting code with a resultant rate $R = (\frac{10}{31}) = 0.32$ is required to achieve the target RBER. Compared to the Manchester code (which also offers improved line coding characteristics) this encoding scheme has an unacceptably low rate while offering similar decoding performance.

If a bi-modal code with a double flag is used, then in order to achieve the target RBER a parent $(31, 16)$ triple error correcting code is required with a resultant rate $R = (\frac{14}{31}) = 0.45$. The performance of this coding scheme compares favourably in terms of decoding performance and rate to the Manchester ECLCs, but does not

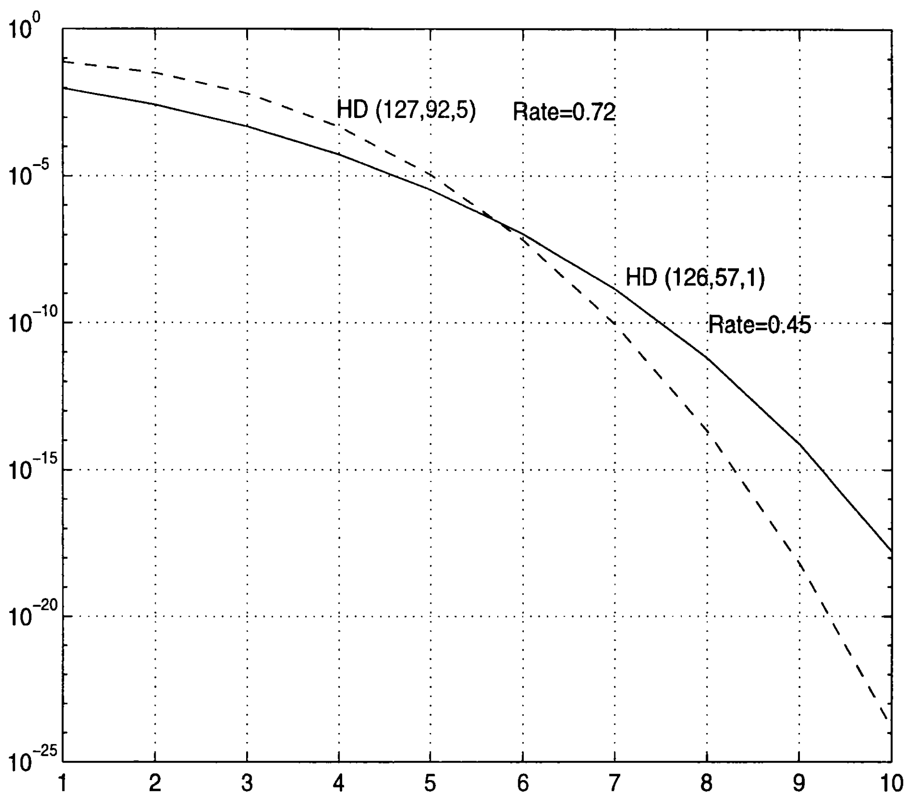


Figure 6.21: Rate comparison of similar codes.

offer tight runlength and disparity bounds.

These results are presented in table 6.4 assuming that a target RBER of 1×10^{-7} is required.

It should be noted that the relationship between the above line coding parameters remains constant regardless of the code used. Thus from the above table the Manchester code seems to have the best line code characteristics and is the simplest one to implement.

Target RBER=1×10^{-7}			
Code Used:	(RL_{max})	(RDS)	Rate reduction (%)
(127,99) BCH ECC code	Unbounded	Unbounded	0
$(63 \times 2, 57)$ ECLC	2	± 1	37.5%
(31,11) Single flag ECLC	17	± 8	55.6%
(31,14) Double flag ECLC	17	± 8	37.5%

Table 6.4: Comparison between rate and LC characteristics of various ECLC codes.

6.13.1 Line code rate

An alternative method of quantifying the rate of each of the proposed and existing ECLCs can be described using the concept of concatenated codes. Specifically, the ECLCs can be considered to consist of two distinct parts: the error correcting code of rate $R_1 = \frac{k}{n}$ and the line code of rate R_2. The overall rate will be $R_o = R_1 \times R_2 = \frac{k}{n} \times R_2$. Using the known value of R_o we can describe most of the presented ECLCs as functions of k (the information bit length) versus R_2, as follows:

- The $nB1I$ ECLC has $R_o = \frac{k-1}{n} = \frac{k}{n}\frac{k-1}{k}$. Therefore, $R_2 = \frac{k-1}{k}$.
- The $nB1I$ EFP 1 has $R_o = \frac{k}{n+2} \Rightarrow R_2 = \frac{n}{n+2}$
- The $nB1I$ EFP 2 has $R_o = \frac{k-1}{n+1} \Rightarrow R_2 = \frac{k+1}{k}\frac{n}{n+1}$.
- The $nB1I$ EFP 3 has $R_o = \frac{k-2}{n} \Rightarrow R_2 = \frac{k-2}{k}$.
- The CAB ECLC has rate $R_o = \frac{k^2}{n(k+1)} \Rightarrow R_2 = \frac{k}{k+1}$
- The CAB N ECLC has rate $R_o = \frac{Nk^2}{n(Nk+1)} \Rightarrow R_2 = \frac{Nk}{Nk+1}$

- The $N = 2$ ECLC has $R_2 = \frac{2k-1}{2k}$, while the improved $N = 2$ has a rate $R_2 = \frac{k-1}{k}$ equal to that of the $nB1I$ ECLC.

- Finally, the 'Manchester' ECLC has a constant LC rate $R_2 = \frac{1}{2}$.

All of the above values are used to derive figure 6.22, where a $n = 127$ BCH code is used with various values of k to present the differences in rate R_2. Increasing values of k indicate that a progressively less powerful EC decoding code is used.

For example, a $(127, 120)$ BCH code (upper end of the x-axis) can correct a single error while a $(127, 8)$ BCH code (lower end of the x-axis) can correct upto 32 errors. The rate of the 'Manchester' code is constant at $R_2 = \frac{1}{2}$ and is therefore not shown. The x-axis displays $log_2(k)$ for clarity.

Figure 6.22 indicates that all ECLCs asymptotically approach unity as the values of k increase. This means that they are becoming progressively more efficient, with the EFP 1 being the most efficient ECLC for low values of k and the $CABN$ with $N = 2$ code for higher values of k. These observations are valid when $n = 127$.

If other values of n are used, then the performance of the codes will also vary. Figure 6.23 presents an indicative example where the ECLC rate performance with $n = 31$ is shown. As expected, for lower values of n the rate performance of the EFP 1 code decreases and the cross-over point with the CAB $N = 2$ ECLC occurs for lower values of k.

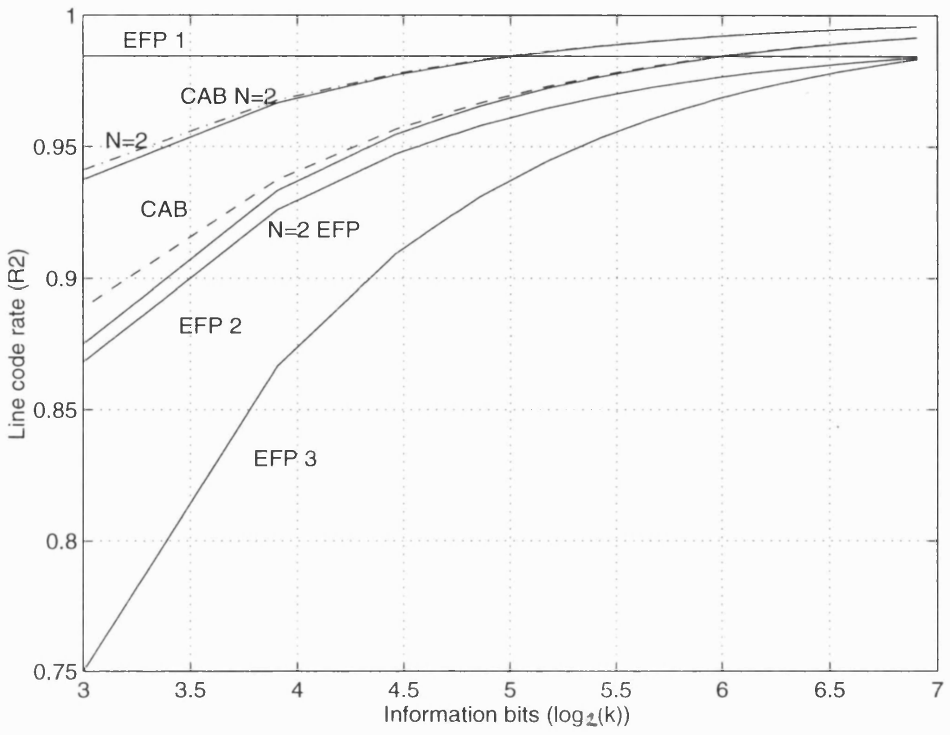


Figure 6.22: ECLC rate comparison with $n = 127$.

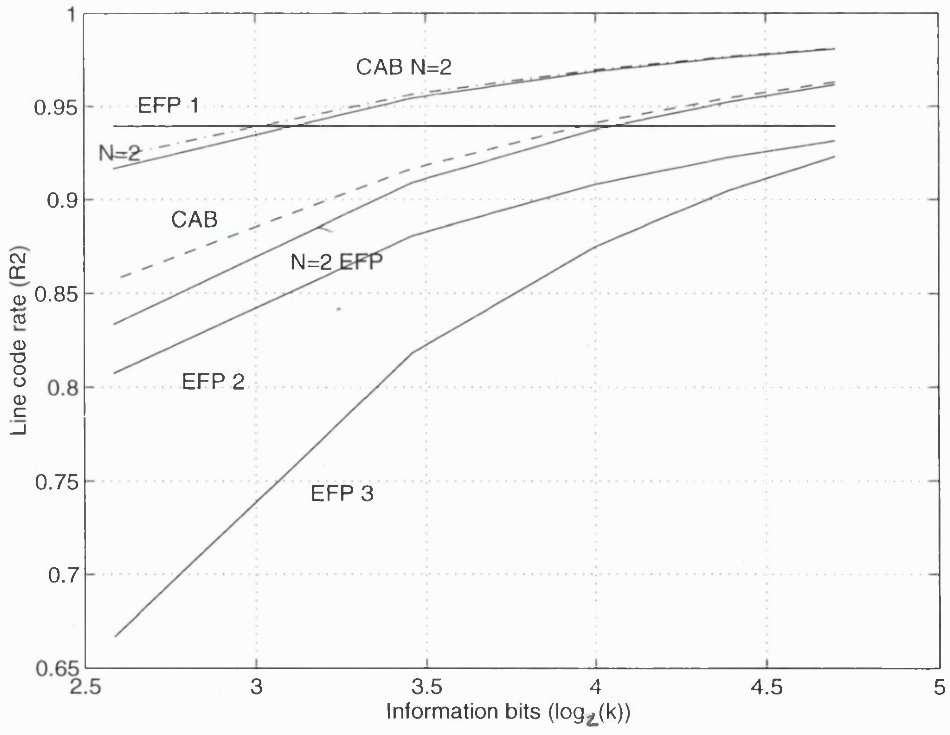


Figure 6.23: ECLC rate comparison with $n = 31$.

6.14 Summary

In this chapter, a number of new concatenated and combined error correcting and line codes were presented. The $nB1X$ family of ECLCs demonstrated that elementary LC properties can be easily achieved with a very small reduction in rate and without significantly affecting decoding performance or increasing circuit complexity.

The EFP and 'Manchester' ECLC combined soft decision decoding techniques together with modified existing ECLC algorithms and produced tight bounds for both runlengths and disparity. At the same time, they achieved improved decoding performance and are relatively simple to implement. Finally the CAB family of ECLCs was introduced which can offer very good overall code rates and decoding performance at the expense of tight line coding bounds. Simulations of both the power spectrum and the decoding power indicate the expected performance improvements for all ECLCs.

Bibliography

[1] Y. Lin, J. Wolf: "Combined ECC/RLL Codes",*IEEE Transactions on Magnetics, Vol 24, No 6*, November 1988.

[2] R. Brooks: "$7B8B$ Balanced Code with Simple Error Detecting Capability", *Electronics Letters*, Vol 16, No. 12, pp.458–459, June 1980.

[3] A. S. Helberg, W. Clarke, H. Ferreira, A. Han Vinck: "A Class of DC-Free Synchronisation Error Correcting Codes", *IEEE Transactions on Magnetics, Vol 29, No 6*, pp. 4048-4049, November 1993.

[4] M. Hodges, E. Jones: "Soft Decision Decoding of Transmission Codes", *Electronics Letters*, Vol 14, No. 18, pp. 352-353, August 1978.

[5] R. Deng, M. Herro: "DC-Free Coset Codes", *IEEE Transactions on Information Theory, Vol 34, No 4*, pp. 786–792, July 1988.

[6] A. Popplewell: "Combined Line and Error Control Coding", *PhD Thesis, University of Wales*, Bangor, 1990.

[7] H. Ferreira: "Lower Bounds on d_{min} Achievable with RLL or DC-Free Block", *IEEE Transactions, Mag 20, No 5*,pp 881–883, 1984.

[8] K. W. Cattermole: "Principles of Pulse Code Modulation", *Ilffe Books*, London 1969.

[9] D. R. Smith: "Digital Transmission Systems", *Van Nostrad Reinhold, Second Edition*, London 1993.

[10] A. Popplewell, J. O'Reilly: " Error Performance Evaluation of a Class of ECLCs", *Symposium on Information Theory and its Applications, SITA 90*, pp. 89–90, Japan 1990.

[11] G. L. Cariolaro, G. P. Tronca: "Spectra of Block Coded Digital Signals", *IEEE Transaction in Communications, COM-22*, No 10, pp 1555–1564, Oct. 1974.

[12] S. Fragiacomo, C. Matrakidis, J. O'Reilly: "Exploiting Soft Decision Decoding for Error Correcting Line Codes", *IEEE International Conference on Communication Systems*, IEEE ICCS/ISPACS '96, Singapore, 25-29 November, 1996.

[13] S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A Novel Error Correcting Line Code", *Third Communication Networks Symposium*, 8-9 July 1996, Manchester, UK.

[14] Y. Bian, J. O'Reilly, A. Popplewell, S. Fragiacomo: "New Simulation Techniques for Evaluation of Telecommunications Transmission Systems with FEC", *IEE Conference in Telecommunications*, Brighton, March 1995.

[15] Y. Bian, J. O'Reilly, A. Popplewell: "Novel Simulation Techniques for Assessing Coding System Performance", *Electronics Letters*, Vol 30, No. 23, pp. 1920–21, November 1994.

[16] A. Popplewell, J. O'Reilly: "Class of Disparity Reducing Transmission Codes with Embedded Error Protection", IEE Proceedings, Vol 137, Pt 1, No 2, pp. 73–77 April 1990.

[17] S. Fragiacomo , C. Matrakidis, J. O'Reilly: "A New Error Correcting Line Code", *ITS/IEEE ROC&C '96, International Telecommunications Symposium*, October 28-31 1996, Acapulco, Mexico.

[18] S. Fragiacomo, C. Matrakidis, J. O'Reilly: "A Novel Error Correcting Line Code", *Networks and Optical Communications (NOC) - Post-Deadline Session*, June 25-28 1996, Heidelberg, Germany.

[19] S. Fragiacomo, C. Matrakidis, J. O'Reilly: "Soft Decision Error Correcting Line Code for Optical Data Storage", *9th Annual Meeting, LEOS 96*, 18-21 November 1996, Boston, USA.

Chapter 7

Conclusions

In this thesis, a number of existing and novel error correcting, line, and error correcting line codes were presented. Particular emphasis has been placed on developing codes which were efficient, general in nature and suitable for high bit rates.

The fundamental principles of error correcting and line coding were initially presented. Existing coding schemes were then examined and new ones developed, which addressed their shortcomings. Computer simulation was then used to validate the theoretical results. The latter required the development of novel techniques which offered a significant reduction in the amount of computational time required.

Line and error control coding were then combined together to create an error correcting line code. This manages to avoid most of the problems created by the use of concatenated codes while at the same time offers improved performance.

Finally the concept of soft decision decoding was also introduced. This can be used on all three types of code presented so far, i.e line, error control and error correcting line codes. It offers significant gains in terms of improved decoding performance but at the same time requires more complex receivers. Various SDD algorithms were examined and the Chase algorithm number 2 was deemed to be the most useful. The generalised Chase 2 together with the AID and TPE algorithms also proved to offer an attractive solution, because of their near ML performance and low complexity.

7.1 Contributions

The main points and conclusions derived from the present work can be summarised as follows:

- Chapter 3 presents a number of novel line codes. These added bit codes are very simple to implement while offering tight runlength and disparity bounds. Because the resultant rate reduction is very small, and the associated encoding and decoding processes are simple, such codes are suitable for high bit-rate applications.

- The need for simulation at the error floors of interest required the development of a novel simulation technique. This was presented in chapter 4. Significant gains in terms of computational time were achieved by only implementing code words which contributed to the RBER.

- Soft decision decoding is an appropriate method for increasing decoding performance without significantly increasing complexity.

- In chapter 5, a novel SDD algorithm is introduced, termed *generalised Chase 2* (GC-2). It achieves near ML performance but is rather complex and computationally intensive for implementation in a practical high speed system. This problem is alleviated by the introduction of the AID and TPE algorithms.

- The AID algorithm reduces the number of test patterns required by the GC-2, without any reduction in decoding power. In addition, the TPE algorithm reduces the number of decodings without affecting performance. These improvements make feasible the use of a GC-2 code for practical system applications.

- Finally, novel error correcting line codes were introduced in chapter 6. These achieved very good line coding characteristics without significantly affecting the decoding performance of the parent EC codes. Additionally most are very simple to implement.

The proposals presented in this thesis were developed in the abstract, i.e. they were not linked to some specific application. Therefore a suggestion for possible future work would be to explore line coding, FEC codes and ECLCs with respect to potential applications. These may include the following:

- Magnetic recording

- Long-haul optical systems

- High speed digital communications links using copper as a transmission medium

7.2 Concluding Remarks

In this thesis, a number of novel line codes were presented followed by a the introduction of a very efficient SDD algorithm suitable for error correcting codes. These were 'combined' in chapter 6 to generate a number of novel ECLCs which utilised SDD to achieve improved performance. The results were validated down to the low bit error levels of interest using a novel simulation acceleration technique.

Appendix A

Calculating the Power Spectrum

A.1 Introduction

In this section, a more detailed explanation of the analytical technique for determining the power spectrum of the $nB1X$ codes will be presented. This was briefly described in chapter 3.

As has been mentioned before, the aim of a line code is to eliminate the DC component and suppress the low frequency content of a transmitted signal as much as possible. The power spectrum is a measure of success of the line code, since it displays the power spectral density versus the normalised frequency.

A.2 Calculating the Power Spectral Density

In order to determine the *power spectral density* (PSD) of a signal ($S_{yy}(f)$) we must first determine the auto-correlation function $R_{yy}(j)$. This is performed at two distinct time intervals, one at $j = 0$ and the other at $j \neq 0$. Let n be the information bit length of a $nB1X$ code, $P[X(k) = \pm 1]$ is the probability that any bit k will have a value of a logic one or zero and j is a variable that can take any value between $0 \leq j \leq n$.

A.2.1 Determining $R_{yy}(j)$ at $j = 0$

The value of $R_{yy}(j)$ at $j = 0$ is determined by the following equation:

$$R_{yy}(0) = \frac{1}{(n+1)} \sum_{k=1}^{n+1} (P[X(k) = 1])(1)^2 + (P[X(k) = -1])(-1)^2$$

Since equal probabilities of having a logic one or zero exist within any code word, $P[X(k) = 1] = P[X(k) = -1] = \frac{1}{2}$, thus:

$$R_{yy}(0) = \frac{1}{(n+1)} \sum_{k=1}^{n+1} 1 = \frac{n+1}{n+1} = 1$$

This equation holds true for all codes of interest, so $R_{yy}(0) = 1$ always.

A.2.2 Determining $R_{yy}(j)$ at $j \neq 0$

The value of $R_{yy}(j)$ at $j \neq 0$ is determined by the following equation:

$$R_{yy}(j) = \frac{1}{(n+1)} \sum_{k=1}^{n+1} (P[X(k-j) = 1])(P[X(k) = 1|X(k-j) = 1]) -$$

$$(P[X(k-j) = 1])(P[X(k) = -1|X(k-j) = 1]) -$$

$$(P[X(k-j) = -1])(P[X(k) = 1|X(k-j) = -1]) +$$

$$(P[X(k-j)=-1])(P[X(k)=-1|X(k-j)=-1])$$

Since the codes under consideration are symmetric towards the number of bits having a logic value of one or zero, $P[X(k-j)=1]=P[X(k-j)=-1]=\frac{1}{2}$ and the above equation now becomes:

$$R_{yy}(j)=\frac{1}{(n+1)}\sum_{k=1}^{n+1}\frac{1}{2}(P[X(k)=1|X(k-j)=1]-$$

$$\frac{1}{2}(P[X(k)=-1|X(k-j)=1]-\frac{1}{2}(P[X(k)=1|X(k-j)=-1]+$$

$$\frac{1}{2}(P[X(k)=-1|X(k-j)=-1]$$

Simplifying the above we get that:

$$R_{yy}(j)=\frac{1}{(n+1)}\sum_{k=1}^{n+1}(P[X(k)=1|X(k-j)=1])$$

$$-(P[X(k)=1|X(k-j)=-1]) \qquad \text{(A.1)}$$

We have therefore managed to determine all the necessary parameters which characterise the auto-correlation function $Ryy(\tau)$. In order to obtain the power spectrum $S_{yy}(f)$ we must now perform a Fourier transformation to the auto-correlation function, i.e.

$$S_{yy}(f)=\mathcal{F}(R_{yy}(\tau))=\mathcal{F}(R_{yy}(0)\delta(\tau)+\sum_{k=1}^{n}(\delta(\tau\pm k)+R_{yy}(k)) \qquad \text{(A.2)}$$

Equation A.2 holds true for all codes of interest. In order to determine the power spectrum of a given code we must calculate the specific values of $Ryy(j)$. These calculations are shown in the following sections for the Manchester and $nB1D$ codes.

A.3 Power spectrum of the Manchester Code

A Manchester code is a code where each bit is always followed by its inverse. Therefore for $j \neq 0$:

$$P[X(1) = 1|X(0) = 1] = P[X(1) = 1|X(0) = -1] = \frac{1}{2}$$

This is true, since there is no correlation between individual code words. In addition

$$P[X(2) = 1|X(1) = 1] = 0$$

and

$$P[X(2) = 1|X(1) = -1] = 1$$

since each bit within a code word is followed by its inverse. Substituting these values in equation A.1 we obtain:

$$R_{yy}(1) = \frac{1}{2}(\frac{1}{2} - \frac{1}{2} - 0 - 1) = -\frac{1}{2}$$

We have therefore evaluated R_{yy} at all points of interest. Therefore,

$$R_{yy}(\tau) = \delta(\tau) - \frac{1}{2}\delta(\tau \pm 1)$$

If we apply the Fourier transform on $R_{yy}(\tau)$ we obtain the equation describing the power spectrum $S_{yy}(f)$ which for this example is equal to:

$$\mathcal{F}(R_{yy}(\tau)) = S_{yy}(f) = 1 - cos(2\pi f)$$

A.4 Power Spectrum of the $nB1D$ Code

The $nB1D$ code is an added bit code whereby the flag is determined by the disparity of the information bits. As before, $R_{yy}(0) = 1$ and if $1 \leq j \leq n$ then

$$(P[X(k) = 1|X(k-j) = 1]) = (P[X(j) = 1|X(n-j) = -1]) = \frac{1}{2}$$

because the information bits are random.

However, the parity bit $(k = (n+1))$ is dependent on the information bits and its value is therefore not random. Assume without loss of generality that $X(0) = 1$, $X(n+1) = 1$ and the disparity of the in-between information bits (excluding the first one) is less than zero. Let y equal the number of $(n-1)$ bits with disparity less than zero and z equal the number of $(n-1)$ bits with disparity equal to zero. Due to symmetry, y is also equal to the number of $(n-1)$ bits with disparity larger than zero. Thus, $2y + z = 2^{n-1}$ and

$$(P[X(n+1) = 1|X(n+1-j) = 1]) = \frac{y}{2^{n-1}}$$

and

$$(P[X(n+1) = 1|X(n+1-j) = -1]) = \frac{z+y}{2^{n-1}}$$

therefore

$$R_{yy}(j) = \frac{1}{(n+1)}(\frac{y}{2^{n-1}} - \frac{z+y}{2^{n-1}}) = \frac{-1}{(n+1)}\frac{z}{(2^{n-1})}$$

By definition z is equal to $\begin{pmatrix} n-1 \\ \frac{n-1}{2} \end{pmatrix}$ thus $R_{yy}(j)$ now becomes equal to:

$$R_{yy}(j) = \frac{1}{(n+1)}\frac{\begin{pmatrix} n-1 \\ \frac{n-1}{2} \end{pmatrix}}{2^{n-1}}$$

Thus

$$R_{yy}(\tau) = \delta(\tau) - \frac{1}{n+1}\frac{\begin{pmatrix} n-1 \\ \frac{n-1}{2} \end{pmatrix}}{2^{n-1}}\sum_{k=1}^{n}\delta(\tau \pm k)$$

If we one more apply the Fourier transform on $R_{yy}(\tau)$ we obtain the power spectrum $S_{yy}(f)$ which is equal to:

$$S_{yy}(f) = 1 - \frac{1}{n+1}\frac{\begin{pmatrix} n-1 \\ \frac{n-1}{2} \end{pmatrix}}{2^{n-1}}cos(2\pi f)$$

We have therefore succeeded in obtaining an analytical formula for the power spectrum on an $nB1D$ code.