

University of South Wales



2064783

Bound by **Abbey**
Bookbinding Co.,
Cardiff, South Wales
Tel: (01222) 395882

THE LIMITING ERROR CORRECTION
CAPABILITIES OF THE CDROM

JONATHAN D. ROBERTS

A thesis submitted in partial fulfilment of the
requirements of the University Of Glamorgan
for the degree of Doctor of Philosophy

November 1995

Acknowledgments

I would like to express my thanks to all those who have given their support and time in the duration of this research programme. This research was funded by EPSRC (formerly known as SERC) in the form of a CASE Award with sponsorship from British Gas PLC. I would like to convey my thanks to all those at the British Gas Engineering Research Station in Killingworth with whom I have worked. With particular thanks to David Burke and Martin Morey.

I would also like to thank Professor Alan Ryley and Dr. David Jones for their continued support, interest and enthusiasm. Additional thanks to Dr. Warwick Clegg of Manchester University for reading my PhD Transfer Report and his continued interest in my work.

Special thanks to my parents who always believed in my abilities when I and others did not. Finally, deep thanks to my wife Anne for her continued support.

Declaration

This is to certify that neither this thesis or any part of it has been presented or is being currently submitted in candidature for any degree other than the degree of Doctor of Philosophy of the University of Glamorgan.

Candidate.....

Abstract

The purpose of this work was to explore the error correction performance of the CDROM data storage medium in both a standard and hostile environment. A detailed simulation of the channel has been written in Pascal. Using this the performance of the CDROM correction strategies against errors may be analysed.

Modulated data was corrupted with both burst and random errors. At each stage of the decoding process the remaining errors are both illustrated and discussed. Results are given for a number of varying burst lengths each at different points within the data structure. It is shown that the maximum correctable burst error is approximately 7000 modulated data bytes.

The effect of both transient and permanent errors on the performance of a CDROM was also investigated. Here software was written which allows both block access times and retries to be obtained from a PC connected to a Hitachi drive unit via a SCSI bus. A number of sequential logical data blocks are read from test discs and access times and retry counts are recorded for each.

Results are presented for two classes of disc, one which is clean and one with a surface blemish. Both are exposed to both standard and hostile vibration environments. Three classes of vibration are considered: isolated shock, fixed state sinusoidal and swept sinusoidal. The critical band of frequencies are demonstrated for each level of vibration. The effect of surface errors on the resistance to vibration is investigated.

Contents

	page
Chapter One : Introduction and Outline	
1.1 Introduction	1
1.2 Outline	2
Chapter Two : Mass Storage Devices	
2.1 Various Mass Storage Devices	4
2.2 The Winchester Disk Drive	4
2.3 High Speed Tape Drives	5
2.4 The Optical Disc	7
Chapter Three : A Mathematical Basis For Error Control In Optical Recording	
3.1 The Need For Error Control	11
3.2 Reed Solomon Codes	11
3.3 Enhancing The Effectiveness Of Error Correcting Codes	20
Chapter Four : The Production Of Channel Code By Modulation	
4.1 The Need For Modulation	23
4.2 Simple Channel Codes	23
4.3 The Block Codes	25
Chapter Five : The Encoding Processes Of The CDROM	
5.1 Introduction	28
5.2 Sector Encoding	28
5.3 The CIRC (Cross Interleaved Reed Solomon Codes) Scheme	40
5.4 The Eight Fouteen Modulation (EFM) Code	46
Chapter Six : The Decoding Processes Of The CDROM	
6.1 Introduction	50
6.2 Error Detection and Correction Using CIRC	51
6.3 Sector Decoding	59
Chapter Seven : Illustrating the Effects Of Errors Upon The CDROM By Use Of The Simulation Model	
7.1 Introduction	62
7.2 The Effect Of A Burst Error On the Channel Data	63
7.3 The Effect Of Random Errors On the Channel Data	76
7.4 Conclusions	85

Chapter Eight : Performance Measurement and Inferencing Using The Simulation Model

8.1	Outline	85
8.2	Burst Correction Performance Analysis Of The CDROM	86
8.3	Inferencing Burst Errors	92

Chapter Nine : Obtaining A Measure For CDROM Performance

9.1	The Need To Measure Performance	96
9.2	The Hardware and Software Requirements	97
9.3	The Low Level Communications Objectives Of ASPI	100
9.4	Transferring Data Using The SCSI Bus	101
9.5	The Access Times and Retry Counts From The Drive	107

Chapter Ten : Measured Performance Of The CDROM Against Transient Errors

10.1	Introduction	109
10.1	Test Disc Details	109
10.3	Outline Of Experiments	112
10.4	Experimental Results	115

Chapter Eleven : Conclusions and Future Work

11.1	Conclusions	135
11.2	Future Work	136

Appendices

Appendix A : Using The Simulation Model		A-1
A.1	Encoding Programs	A-2
A.2	Decoding Programs	A-4
A.3	Error Incorporation Programs	A-5
A.4	Illustration Programs	A-8
A.5	Inferencing Software and Database	A-10
Appendix B : Drive Communication Issues		
B.1	Assemble Module 1 : OPEN.ASM	B-1
B.2	Assembly Module 2 : SEND_DATA.ASM	B-10
B.3	Main C Program : COMMUN.C	B-14
Appendix C : The Effects Of Multiple Burst Errors		C-1
C.1	Illustration Of Two Bursts Within A Sector	C-2

References

Bibliography

Glossary

List Of Figures

<u>Number</u>	<u>Heading</u>	<u>page</u>
2.1	The Layout of Tracks On RDAT	6
3.1	Generation Of Galois Field Elements	13
3.2	Parity Production In Product Codes	20
4.1	Illustration Of FM and Miller ²	25
4.2	Illustration Of DSV Control	27
5.1	The Compact Disc Encoding Processes	29
5.2	The Three Sector Configurations	30
5.3	The Sector Encoding Processes	31
5.4	Matrix For P Parity Calculation	33
5.5	V_p Vector	34
5.6	H_p Matrix	35
5.7	Matrix For Q and P Parity Calculation	36
5.8	Q Matrix Parity Calculation	37
5.9	H_Q Matrix	37
5.10	The Scrambler Circuit	39
5.11	Example Of The Circuit On Real Data	39
5.12	The CIRC Encoding Processes	40
5.13	An Example Of Delay	41
5.14	V_Q Vector	42
5.15	H_Q Matrix	42
5.16	H_p Vector	44
5.17	H_p Matrix	44
5.18	Control Of DSV By Use Of Merging Bits	47
5.19	Minimisation Using Multiple Look-ahead	48
6.1	Flow Diagram Of C1 Error Correction	52
6.2	Flow Diagram Of C2 Error Correction	56
7.1	Error Incorporation Into The Channel	63
2900 bit burst		
7.2	Effect Upon The Channel	64
7.3	Synchronisation Loss	64

7.4	Deinterleave Strategy III	64
7.5	C1 Decoding	66
7.6	Deinterleave Strategy II	66
7.7	Effect Of C2 Decoding	66
3000 bit error		
7.8	Effect Upon The Channel	67
7.9	Synchronisation Loss	67
7.10	Deinterleave Strategy III	67
7.11	C1 Decoding	68
7.12	Deinterleave Strategy II	68
7.13	C2 Decoding	68
7.14	Deinterleave Strategy II	70
7.15	Byte Interchanging	70
7.16	Position Of Errors In Product Matrices	70
7.17	Q Decoding	71
7.18	Decoding On Raw Data	71
3600 bit error		
7.19	Effect Upon The Channel	72
7.20	Synchronisation Loss	72
7.21	Deinterleave Strategy III	72
7.22	C1 Decoding	73
7.23	Deinterleave Strategy II	73
7.24	C2 Decoding	73
7.25	Deinterleave Strategy II	74
7.26	Byte Interchanging	74
7.27	Position Of Errors In Product Matrices	74
7.28	Q Decoding	75
7.29	P Decoding On Raw Data	75
Random Error with Probability 0.0018		
7.30	Effect Upon The Channel	77
7.31	Synchronisation Loss	77

7.32	Deinterleave Strategy III	77
7.33	C1 Decoding	78
7.34	Deinterleave Strategy II	78
7.24	C2 Decoding	78
7.36	Position Of Errors In Product Matrices	80
7.37	Q Decoding	80
Random Error with Probability 0.0019		
7.38	Effect Upon The Channel	81
7.39	Synchronisation Loss	81
7.40	Deinterleave Strategy III	81
7.41	C1 Decoding	82
7.42	Deinterleave Strategy II	82
7.43	C2 Decoding	82
7.44	Position Of Errors In Product Matrices	83
7.45	Q Decoding	83
7.46	P Decoding On Raw Data	83
8.1	Error Statistics Produced By Bursts At Frame 40	87
8.2	Error Statistics Produced By A 4000 Bit Burst	88
8.3	Error Statistics Produced By A 7000 Bit Burst	90
8.4	Maximum Burst Correction Along A Sector	91
8.5	Inferencing Package	93
9.1	Hardware Set-up	97
9.2	Simplistic Communication	98
9.3	Relationship Between SRB and CDB	99
9.4	Program Structure	II
10.1	CDROM With Error	II
10.2	Output From Performance Software	III
10.3	Illustration Of The Effects Of Frequencies On Drive At 0.5g-2g	
10.22		122-133

CHAPTER ONE

Introduction and Outline

1.1 Introduction

In recent years the development of computers with greater processing power and greatly increased storage requirements has led to an inexorable demand for storage devices which are physically smaller, have an increased capacity, but are cheaper in real terms. The increases in storage capacity occur at such a rate such that many of these figures mentioned here will have been surpassed.

Mass storage devices can be divided into 3 main categories. These are thin film magnetic, thick film magnetic and optical storage. Thin film magnetic devices are used for fast random access and relatively low storage applications, particularly for rapid file handling in computers. For example the Winchester hard disk now has a capacity of 550 MByte. Thick film magnetic tape devices are used for applications with sequential access, for example data backup and archiving. A Digital Data Storage (DDS) Tape currently has a 1.4 Gbyte storage capacity. Optical disk devices are used for relatively slow random access and high storage applications, for example large commercial databases. A Compact Disc Read Only Memory (CDROM) has a storage capacity of 553 Mbytes *Sponheimer[1,pp 39]*.

The CDROM data structure has been developed from the CD (Compact Disc) audio. The Compact Disc interpolation procedure while appropriate for audio is clearly unsuitable for data storage. To compensate for this additional error control strategies are incorporated.

Each of the three classes of application is served by proprietary devices. All must therefore conform to International Standards, e.g. ECMA (European Computer Manufacturers Association). The standards for the CDROM have been devised in the context of a controlled environment. One purpose of the present work is to explore the performance of the CDROM media and drive operations in environments which may be hostile.

The aims of the work were :

- to understand the full encoding and decoding error correction of the CDROM;
- to assess the limits of the error correcting performance of the CDROM operating against permanent error mechanisms;
- to conduct detailed investigations into the retry strategy against transient error mechanisms;
- to measure the effect of imperfect disk surfaces on error control performance;
- to investigate the effect of an adverse vibrating environment on the CDROM;
- to produce software in procedural and user language which will :
 - (1) simulate the response of the CDROM to a permanent error;
 - (2) allow access individual Sectors of the CDROM and to monitor the access times and retries of the blocks involved.

1.2 Outline

In **Chapter Two** the three main classes of Mass Storage Devices are reviewed, with particular emphasis upon the CDROM.

The CDROM error correction strategy is based upon Reed Solomon Codes. In Chapter Three the theoretical basis and treatment of Reed Solomon Codes is discussed. In addition a practical computer simulation implementation of these strategies is considered. This Chapter also gives an account of product codes and interleaving, both of which are major features of the CDROM.

The CDROM uses EFM (Eight Fourteen Modulation code) which is a Block Modulation Code. In **Chapter Four**, Block modulation coding is introduced and briefly discussed.

Chapter Five gives a detailed account of the specific strategies for encoding data of the CDROM and how they are simulated. **Chapter Six** describes the inverse operations of **Chapter Five**. It explains how the modulated data together with its errors is decoded and how the signal

processing uses the error syndromes which are introduced by corrupted data.

Chapter Seven gives examples of the use of a simulation model to illustrate and correct both burst and random errors. In each example, data is generated, encoded, modulated and corrupted. The corrupting errors which have been introduced are addressed by each stage of the strategy in turn. At each stage the remaining errors are illustrated with a commentary. Examples are used to illustrate the full scope of the error control of the CDROM.

In **Chapter Eight** the performance of the CDROM for a large systematic range of burst errors is illustrated and discussed. Here the simulation model is exposed to bursts which range between 100 and 8000 bits in length, with varying starting position through the Sector. In every case interim results are produced for residual errors within the sector at each stage of the decoding. Using this data, it is possible to illustrate the performance of the correction strategies.

In addition **Chapter Eight** discusses the uses of the data for inferencing errors on the CDROM. Here the intermediary error results produced by the model may be used as a database which may be compared to empirical results. The intermediary decoding errors from a practical CDROM with a blemished surface are measured. The most likely form of the blemish is then inferred by interrogating the database.

Some error mechanisms produce permanent errors: surface scratches for example. In this case, if the data corruption defeats the error control procedures the Sector decoding fails and that Sector is lost. Further attempts to access the Sector will fail in exactly the same manner. By contrast in **Chapter Nine** the effect of transient and semi-transient errors are considered. Here a sector of data may be successfully accessed after several attempts. Data on Sector access times and retries is obtained using a Hitachi CDROM drive connected via a SCSI bus to a PC which monitors CDROM error performance. Both assembler and 'C' software are used to access the CDROM control information. The results of the experiments are described in **Chapter Ten**. Conclusions and proposals for future work are offered in **Chapter Eleven**.

CHAPTER TWO

Mass Storage Devices

2.1 Various Mass Storage Devices

Since digital computers established a widespread industrial and commercial application some three decades ago, the demands made on digital storage media by digital technology have increased inexorably and exponentially. Interestingly throughout this period the two principal classes of storage media have remained unchanged. In general storage devices continue to employ either disk or tape for the storage media.

The demands for data storage are addressed by storage media and the associated enabling technologies. Very high areal densities have been achieved using Write Once Read Many (WORM) optical media, or by writing and reading to magnetic media using rotatory heads. The advances in storage media technology have been accompanied by sophisticated signal processing procedures *Bell[2], Bell[3], Wood[4] & Laub[5]*. Taken together these developments necessitate enhanced error protection, even as areal densities increase and costs drop.

This chapter considers the common examples of each of the three principal classes of data storage. The Winchester disk as an example of thin film magnetic storage. The RDAT (Rotary Digital Audio Tape) and associated and RDAT-DDS (Digital Data Storage) as an example of thick film magnetic storage. Lastly the CD and CDROM as an example of optical storage *Alford[6]*.

2.2 The Winchester Disk Drive

The development of storage disk technology has been driven by computer technology. Random access computer storage is prohibitively expensive for general data storage. The exploitation of computers would have been

impossible without a cheap mass storage medium which nevertheless offered acceptable access times *Zoellick[7,pp 177]*.

The Winchester disk (or Hard Disk) is a thin film device with a higher areal density than the thick film floppy disk. For Hard Disks the head positioning equipment together with the disk itself are sealed as a single unit. This permits track width reduction and hence greater capacity. The single unit inhibits disk exchange, however, and the use of the floppy disks remains widespread for data exchange and back up. More recently high speed tape drives such as DDS have been developed for this purpose.

A disk consists of a number of platters, the more platters the greater the storage capacity. Each platter consists of many concentric rings from the hub to the rim, each ring is called a track. Each platter is subdivided into sectors. A disk is formatted so that data is stored in blocks which contain a fixed number of bytes. Each sector of a track holds one data block. Thus a block of data occupies more space at the rim than at the hub due to the constant angular velocity. Since data is accessed at the same rate since the Constant Angular Velocity of the disk produces linear disk/head speeds across the surface *Christodoulakis[8,pp 152]*. Data is written to these sectors by magnetising particles in medium.

Hard disks with capacities in excess of 100 Megabytes are of particular importance as fixed computer disks or file servers.

2.3 High Speed Tape Drives

Two high speed tape devices of particular interest are RDAT and RDAT-DDS. Both devices use an advanced form of helical scan technology which was originally developed for VHS video. Fixed head tape magnetic recording devices pass thick film magnetic tape across fixed, mounted recording heads usually with a velocity that greatly exceeds 5 cm/s. Recording tracks are therefore laid down along the axis of the tape. Although recording densities using fixed head recording continue to improve, developments are hampered by such error mechanisms as crosstalk and mechanical tolerance. The guard

bands of un-magnetised tape between adjacent tracks are required to overcome these errors.

In RDAT the tracks are recording diagonally across the tape *Dare*[9]. This is achieved by using two heads mounted on a rotating drum at 180 degrees *Baugh*[10]. The drum rotates at a speed of 200 rpm whilst the tape moves at 8 mm/sec in the same direction. A helical pattern is described over the tape by each head. The two heads are offset by equal and opposite azimuth angles. By this method alternative tracks are skewed. This reduces crosstalk, since the heads will pick up stronger signals from data written in the same azimuth angle as itself. The heads are wider than the tracks, so avoiding the need for guard bands. Hence very high areal densities are obtained *Watkinson*[11,pp 280]. RDAT data integrity is further reinforced by complex signal processing procedure incorporating multiple interleaving and Reed-Solomon coding. In RDAT digital signals which are being recorded are interleaved between adjacent tracks, so dispersing burst errors due to dropouts and abrasions of the tape. In addition RDAT also employs interpolation which is acceptable for audio application. There are two levels of error correction coding called C1 and C2 codes respectively. Each is supported by a Reed Solomon Code error control strategy.

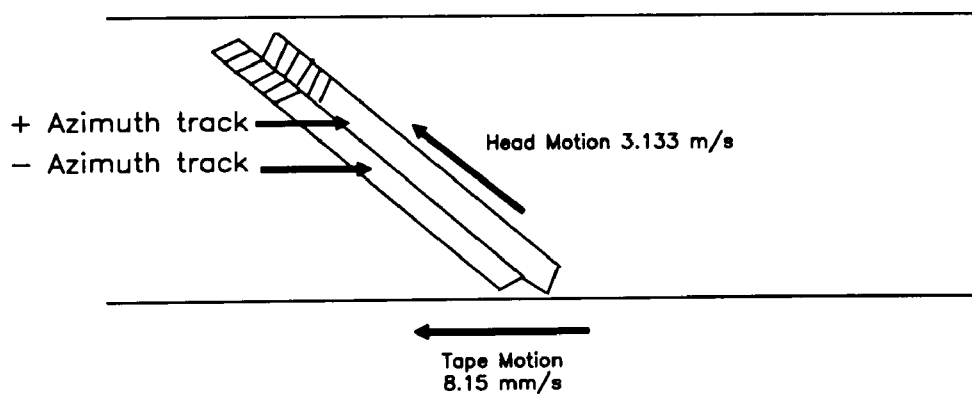


Figure 2.1 : The Layout of Tracks On RDAT

DDS is a recording format developed by Hewlett-Packard and Sony, which develops RDAT for computer applications. In RDAT continuous tracks of data are used, but in DDS frames are organised into groups in which each contains 22 frames. A third error correcting code, C3 is introduced at group

level. By contrast with C1 and C2 codes all error correction occurs within a single track *Odaka[12]*. In addition to an extra level of error correction the DDS format incorporates Read after Write and Multiple group writing.

High speed tapes have the advantage of many write - many read media. However the sequential mean access times can be unsatisfactory, particularly when accessed data is widely dispersed along the tape.

2.4 The Optical Disc

The present work is concerned with the CD and associated CDROM *Lambert[13] & Poel[14]*. The CD offers enormous advantages as a storage medium which were originally exploited for the digital storage of speech and music *Carasso[15], Goedhart[16]*. Complex strategies for data integrity such as interleaving, second order Reed Solomon coding and an extended block modulation code were developed to ensure that advances in areal density were allied to suitably low recording error rates for this purpose *Vries[17]*. The compact disc has been specified in a number of standards *ECMA[18], BSI[19] & ECMA[20]*.

Compact Discs are produced as a physically protected medium from which data can be repeatedly read without degradation of the medium. The channel bits are incorporated into an aluminium layer which is physically protected on the read side by plastic and on the other side by both a specially designed lacquer and a chemically inert paint *Verkaik[21], Watkinson[22,pp 70]*. In order to destroy data on the aluminium one side must be penetrated and the metal thereby rendered ineffective. In addition the channel data is protected by the sophisticated algorithms to correct those errors which are sustained by the disc.

The data on a Compact Disc is stored in a spiral from the centre to the rim, however initial data is stored at the centre *Watkinson[23, pp 1046]*. The head-disc speed is significant. The CD is rotated with a Constant Linear

Velocity 1.3 m/s so that the relative angular head-disc speed varies depending upon which part of the disc is being read, varying from the highest speed (458 revs/min) at the centre to the lowest at the rim (197 revs/min) *Christodoulakis*[24,pp 152], *Barbosa*[25,pp 189-191], *Miyaoka*[26,pp 37]. The CD was been designed as a compromise between access time and capacity *Christodoulakis*[27], *Davies*[28,pp 38]. The quantity of data stored is determined by two parameters, track pitch and linear information density *Immink*[29,pp 410]. The use of CLV has facilitated the data capacity of the Compact Disc to be double that which would have existed if it were a CAV application. The linear data density along the spiral also varies inversely with the head-disk speed, thus ensuring that data is read at a constant rate of 176.4 Kbytes/s over the whole of the disc *Watkinson*[11,pp 463].

The data is laid out on the disc in sectors within tracks *Zoellick*[7,pp 177]. The distance between each track is known as the track pitch and this is 1.6 micrometers *Miyaoka*[26,pp 35]. This is similar to the format of a Winchester Disk except that with a CD sectors are laid along a spiral instead of lying in concentric rings. The data nearest the hub is the Table Of Contents *Peek*[30,p 8]. This is the index of the disk which conveys control data to communicating hardware: how many tracks are present, where the starting sector of each track is or whether the disc is a CD or a CDROM. A CD can in be played in a CDROM player but not vice versa.

When a CD is placed in the player the initial concern is to accelerate the disc to reading speed *Sponheimer*[1,pp 41]. The read laser is pushed to the centre of the disc where it reads the Table Of Contents. On looking for a designated sector or block (they are the same) of data the laser head will be moved further out until it finds a sector which is near to that of the target. Finding the designated sector is dependant upon both the tracking and focusing abilities of the laser *Watkinson*[31] & *Miyaoka*[26,pp 36]. Each sector has associated control data which aids addressing, the address being the time displaced from the start of the track.

When the target Sector is accessed the channel bits are decoded and raw data retrieved. By the time that decoding of a CD sector has been completed the disc will have rotated sufficiently to read the next logical sector. Access time is crucial and sophisticated algorithms have been developed which predict the best arrangement of logically sequential blocks on the medium; i.e. minimising average access times. Furthermore this procedure enhances the protection of data against physical damage by distributing associated data around the disc. Due to the refractive index properties of the transparent layer are also used to reduce the effects of scratches. The 0.8 mm diameter read laser spot on the surface is reduced via a convex lens and diffraction to a 1.7 μm spot upon the channel data surface *Watkinson[22,pp 70]*. Any obstruction or scratch smaller than 0.5 mm will not effect the laser spot at the pit and land of the channel data level *Hoever[32,pp 70]* & *Miyaoka[26,pp 35]*. The depth of each pit is approximately 0.11 micrometers.

External vibration to the CD will degrade its performance, since the read laser may be propelled in any direction. This may cause the hardware to fail to read the channel data accurately and the attempt to access the Sector to fail. In this case the hardware will reseek to the appropriate sector. The choice of sector to be resought is specified by the CD drive and associated hardware. For example, in an audio CD player any sector in the current track can be resought, although the quality of the sound would be degraded. The two error correcting codes of the CD are able to cope with specific levels of byte errors. However if errors do remain after such correction schemes have been applied then labelling occurs and interpolation is used, where corrupted data is estimated using associated data *Vries[33,pp 2]*.

More recently the CDRom has been developed as a major mass data storage medium for computing applications *Cardinali[34]*. The CDRom contains an extra layer of error correction and detection. Two extra layers of protection are required to attain the error performance necessary for mass data storage *Sako[35,pp 3996]*. After these stages a Cyclic Redundancy Check

is applied to the data. If this fails then the whole sample of data is considered erroneous and discarded. Note that a verified Sector of CDROM data is absolutely correct, no interpolation is involved *Chen[36]*. Clearly latency cannot be exploited when reseeking the same sector. Reseeks will consequently increase a Sector access time.

Each of the three storage media discussed in this chapter has a specific computing application. The Winchester Disk is used for fixed disk computer storage, the CDROM where mass storage with random access is required and DDS, which has sequential access is used for data backup. The storage device of greatest interest is the CDROM.

CHAPTER THREE

A Mathematical Basis For Error Control In Optical Recording

3.1 The Need For Error Control

Any digital data encoding channel is liable to produce recording errors in which there are inconsistencies between the data which is written onto the medium and the data which is read from it *Forney[37], Shannon[38] & Shannon[39]*. Even in controlled environments like computer storage recording errors will occur due to such error mechanisms as additive noise or abrasions on the surface of the medium.

In environments which are less well controlled the error rate will be correspondingly higher. The need therefore is for an error coding strategy which will identify that a recording error has occurred and will then correct the error to re-establish the input data *Berlekamp[40], Watkinson[41]*.

3.2 Reed Solomon Codes

Every bit of a binary data stream is drawn from a two state alphabet {0,1}; thus if a single bit is known to be in error it can be corrected by simply reversing its state. When data is considered as a sequence of code vectors, each containing multiple bits, more complicated correction strategies must be employed *Doi[42]*. Such strategies must detect and then correct all bits which are in error.

The most simple strategy of error detection and correction is the single error correcting Hamming code, which is of limited practical value *Hamming[43]*. The more complex Reed Solomon Codes (RSC) are designed to deal with error bursts, where several contiguous bits are in error. These codes are able to achieve multiple error correction, these are now investigated *Berlekamp[44], Golomb[45,pp 204] & Reed[46]*. In contrast to Hamming codes,

in which codewords of bits are considered separately, Reed Solomon codes group codeword bits into sub-codewords, or symbols. For the CDROM each symbol has eight bits and referred to as a code byte.

Reed Solomon Codes are Linear Block Codes *Vries[17,pp 3], Hoeve[32,pp 167]*. Such codes are called systematic (n, k) codes where k is number of symbols or bytes entering the encoding process; n is the number of symbols being output. Hence the codes introduce (n-k) parity check symbols.

3.2.1 Galois Fields

Galois Fields are the mathematical basis upon which much complex error coding is based *Golomb[45,pp 208]*. Galois fields are finite fields and consist of a finite set of elements and two defined binary operations. In finite field arithmetic, any operation carried out on two elements from the field results in a member of that field *Sweeny[47,pp 73]*.

A Galois field in which there are n elements is referred to as GF(q), where q must be prime. The Galois Field GF(q) is defined over the set {0, ... , q-1}.

3.2.2 Binary Extensions To The Galois Field

The Galois Field GF(q^m) is known as an extended field. If $q=2$ then the Galois Field is called an extended binary field.

For the field GF(2^m), it is known that:

$$\begin{aligned} \alpha^z &= 1 \\ \alpha^z + 1 &= 0 \end{aligned} \quad \text{where } z = (2^m - 1).$$

A polynomial $g(\alpha)$ is irreducible in GF(2) if it has no roots in GF(2^m); but any irreducible polynomial does have roots in the extended field GF(2^m), where m is the degree of the polynomial. In GF(2^3) the polynomial is $\alpha^7 + 1$ and it can be reduced to three constituent polynomials:

$$\alpha^7 + 1 = (\alpha + 1)(\alpha^3 + \alpha^2 + 1)(\alpha^3 + \alpha + 1).$$

The three constituent polynomials cannot be reduced any further, i.e. they are irreducible. The generation of all the elements in the field is illustrated in **Figure 3.1**. The generator polynomial equation being :

$$\alpha^3 = \alpha + 1.$$

The polynomial $\alpha^3 + \alpha + 1 = 0$ ($\alpha^3 = \alpha + 1$) is used to generate the field elements, e.g. α^3, α^4 etc. If the code generator polynomial is prime then it generates a cyclic code. A table of such polynomials is available in most related texts *Sweeny[47,pp 52]*. For simplicity this is illustrated using a small field.

Figure 3.1 : Generation Of Galois Field Elements

The first few elements are simple to produce *Doi[42,pp 176]*:

$$\begin{aligned} 0 &= 0 &= &000 \\ \alpha^0 &= 1 &= &001 \\ \alpha^1 &= \alpha &= &010 \\ \alpha^2 &= \alpha^2 &= &100 \end{aligned}$$

The irreducible generator equation can be used to find the remaining elements.

$$\begin{aligned} \alpha^3 &= \alpha + 1 &= &011 \\ \alpha^4 &= \alpha(\alpha^3) \\ &= \alpha(\alpha + 1) &= &\alpha^2 + \alpha &= &110 \\ \text{Similarly: } \alpha^5 &= \alpha + 1 + \alpha^2 &= &111 \\ \alpha^6 &= \alpha^2 + 1 &= &101 \\ \alpha^7 &= \alpha + \alpha + 1 &= &001 \\ \text{hence, i.e. } \alpha^7 &= \alpha^0 && \text{ which illustrates the cyclic properties.} \end{aligned}$$

After α^6 the GF elements will repeat because the code is cyclic. The operations work as shown.

$$\begin{array}{lclclcl}
\alpha^4 \times \alpha^5 & = & \alpha^{9(\bmod 7)} & = & \alpha^2 \\
\alpha^3 \times \alpha^1 & = & \alpha^{4(\bmod 7)} & = & \alpha^4 \\
\alpha^4 + \alpha^5 & = & 110 + 111 = & 001 & = & \alpha^0 \\
\alpha^3 + \alpha^1 & = & 011 + 010 = & 001 & = & \alpha^0
\end{array}$$

3.2.3 Direct Production Of The Galois Fields

In the previous section elements of the finite fields have been generated by algebraic means. However so as to generate elements of larger fields it is appropriate to use logical operations. In this case higher powers of the field element ' α ' may be generated simply by a sequence of bit shift and exclusive 'OR' (XOR) operations. The algorithms is as follows;

To generate α^{t+1} from α^t :

- (i) Left shift the binary representation of α^t , introducing a zero to the least significant end. The shift causes overflow at the most significant end.
- (ii) If the overflowing bit is a '0' then the shifted binary pattern represents α^{t+1} .
- (iii) If the overflowing bit is a '1' then the shifted pattern is Xor'd (logical exclusive OR operation) with the binary representation of the generator equation $g(\alpha)$. This gives new pattern represents α^{t+1} .

As an example consider the generation of $\alpha, \alpha^2, \alpha^3$, where α^0 has the bit pattern of 001 and the generator equation $g(\alpha)$ is $\alpha^3 = \alpha + 1$. (\rightarrow signifies the binary shift).

$$\alpha^0 \quad : \quad 001 \rightarrow (0)010 = \alpha.$$

$$\alpha^1 : 010 \rightarrow (0)100 = \alpha^2.$$

$$\alpha^2 : 100 \rightarrow (1)000 = 011 = \alpha^3.$$

xor 011

By the use of this logic the field elements can be generated for any sized field for any polynomial. This is how such elements can be generated for use in the simulation model which will be discussed later.

3.2.4 Encoding Using Reed Solomon Codes

Reed Solomon Codes are designed to correct bursts, i.e. the number of consecutive bits in error *Hoeve[32,pp 166], Peek[31,pp 11]*. The number of correctable symbols is governed by the amount of redundancy added to the data. The number of correctable symbols is half the number of check symbols added. Reed Solomon Codes are effective as burst correcting code, however random errors spread evenly through the codewords can frustrate their effectiveness *McEliece[48]*.

The data bits are assembled into symbols which are elements of the extended Galois Field. An illustration of how parity equations are produced from data follows. To ensure ease of illustration the three bit GF discussed previously is used. For three bit Reed Solomon Codes over $GF(2^3)$ there will be seven three bit symbols, this is a (7,5) code. The chosen generator polynomial is $\alpha^3 + \alpha + 1$.

The equations which describe the relationship between data and parity symbols are produced using the following matrices. V is the matrix holding the data and parity symbols. In addition H describes the GF coefficients which will be combined with the seven symbols to produce the desired equations, with result zero. A-E are data symbols and P & Q are parity symbols.

and these are combined thus

$$\mathbf{Y} = [A \ B \ C \ D \ E \ P \ Q] \quad (3.1)$$

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha^0 & \alpha^1 & \alpha^2 & \alpha^3 & \alpha^4 & \alpha^5 & \alpha^6 \end{bmatrix} \quad (3.2)$$

$$\mathbf{H} \mathbf{Y}^T = \mathbf{Q} \quad (3.3)$$

The matrix equations are rearranged in terms of P and Q using Galois Field operations. From the equations it can be seen that the result of adding the data to the parity bytes should be zero. The matrix describing the mathematical equation linking data and parity is as follows:

$$\begin{bmatrix} P \\ Q \end{bmatrix} = \begin{bmatrix} \alpha^6 & \alpha^1 & \alpha^2 & \alpha^5 & \alpha^3 \\ \alpha^2 & \alpha^3 & \alpha^6 & \alpha^4 & \alpha^1 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} \quad (3.4)$$

As an example of parity byte production consider the 15 bit data word {101 100 010 100 111}, where n=3. The parity symbols P and Q are found to be 100 and 100, respectively. A-E, P and Q are collectively known as the codeword.

3.2.5 Decoding Using Single Error Detection and Correction

The codewords previously produced can be used to assess whether corruption has occurred, this shall now be investigated. At the destination the codeword is tested in order to ascertain whether an error has occurred, V_e denotes the received codeword. Decoding is carried out using the identical equations described in equations (3.1), (3.2) and (3.3). Thus by definition if there is no error present then the result of adding V^T to H is zero. If any contamination is present then the result will be non-zero. The linear difference will produce unique error syndromes. By working back from these syndromes both the location and value of the contamination may be found.

$$\mathbf{H} \mathbf{V}'_e = \mathbf{H} (\mathbf{V}^T + \mathbf{e}) = \begin{bmatrix} S_0 \\ S_1 \end{bmatrix} \quad (3.5)$$

The outputs S_0 and S_1 are the syndromes, if both are zero then no error has occurred *Vries[17,pp 6]*.

If a one symbol error does occur then the syndromes will be non-zero, these syndromes can be used to locate the error *Doi[42,pp 152]*. It should be stressed that errors referred to are symbol errors and that a single or multiple bit error in a symbol will produce a symbol error. The codeword has been corrupted from {101 100 010 100 111 100 100} to {111 100 010 100 111 100 100}.

$$\begin{bmatrix} S_0 \\ S_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \alpha^0 \\ \alpha^6 \end{bmatrix} \quad (3.6)$$

In this example the syndromes are non-zero indicating that at least one symbol error is present in the codeword. The syndrome S_0 gives the error bit pattern within the symbol whereas S_1 identifies the symbol which is in error. S_1 has been calculated by multiplying symbols with different power of α depending upon the position of the symbol within the codeword. An error in A will be multiplied by α^6 whereas an error in Q would be multiplied by α^0 . The erroneous symbol can be located as follows:

$$\frac{S_1}{S_0} = \frac{\alpha^6}{\alpha^0} = \alpha^6 \quad (3.7)$$

Symbol in error = Symbol (7 - power of alpha) = symbol 1

This indicates that symbol one is in error, i.e. symbol A is in error. This is now represented by A^* , (Symbol A is the first symbol and symbol Q is seventh). The correct bit pattern of symbol A can now be determined by adding the erroneous symbol A^* to the S_0 syndrome.

$$\text{Hence } \mathbf{A} = \mathbf{A}^* + \mathbf{S}_0 = 111 + 010 = 101 \quad (3.8)$$

as required.

This method only works if one symbol is in error, for multiple errors the same logic can be used. However the application is slightly different.

3.2.6 Decoding Using Multiple Error Detection and Correction

The power of a Reed Solomon Code is decided upon at the design stage, the number of correctable bits being half the redundancy *Doi[42,pp 153]*. If however the location of the symbols in error has been identified by other means, the number of symbols that can be corrected is equal to the redundancy. As an example consider the same (7,5) code where two known symbols have been corrupted. The codeword has been corrupted from {101 100 010 100 111 100 100} to {111 100 010 000 111 100 100}, where $A^*=111$ and $D^*=000$. The erroneous codeword gives syndromes of value $S_0=110$ and $S_1=110$ *Hoeve[32,pp 168]*. Since the positions of the corrupted symbols are known the following equations are true.

$$\begin{bmatrix} S_0 \\ S_1 \end{bmatrix} = \begin{bmatrix} (A^* + A) + (B^* + B) \\ \alpha^6(A^* + A) + \alpha^3(B^* + B) \end{bmatrix} = \begin{bmatrix} S_A + S_B \\ \alpha^6 S_A + \alpha^3 S_B \end{bmatrix} \quad (3.8)$$

By solving these equations the correct bit values of the two erroneous symbols can be found. The first row of the matrix described as **Equation 3.8** can be rearranged such that:

$$S_A = S_D + S_0 \quad (3.9)$$

Using the second row of **Equation 3.8** and substituting **Equation 3.9** where possible **Equation 3.10** is produced as shown:

$$\begin{aligned} S_1 &= \alpha^6 S_A + \alpha^3 S_D &= \alpha^6(S_D + S_0) + \alpha^3 S_D \\ &= \alpha^6 S_D + \alpha^6 S_0 + \alpha^3 S_D &= \alpha^6 S_0 + S_D(\alpha^3 + \alpha^6) \\ &= \alpha^6 S_0 + S_D(\alpha^4) & \end{aligned} \quad (3.10)$$

Rearranging and using the known syndromes :

$$\begin{aligned}
 S_D(\alpha^4) &= \alpha^6 S_0 + S_1 \\
 S_D &= \alpha^{(6-4)} S_0 + \alpha^{(0-4)} S_1 &= \alpha^2 S_0 + \alpha^3 S_1 \\
 &= (100)(110) + (011)(110) &= 100
 \end{aligned}$$

Using **Equation 3.9** and the value of S_D :

$$\begin{aligned}
 S_A &= S_D + S_0 \\
 &= 010.
 \end{aligned}$$

Applying the given correction:

$$A = A^* + S_A = 101.$$

$$D = D^* + S_D = 100.$$

In practical applications of Reed Solomon Coding in data storage there are far more than three bits per symbol. A good example is that of CDROM and RDAT, where 8 bit symbols are used. Eight bits fit in conveniently with both sixteen bit audio samples and byte orientated computer chips.

3.3 Enhancing The Effectiveness Of Error Correcting Codes

The effectiveness of Reed Solomon error correction can be significantly enhanced by using those codes in combination.

There are two methods. The first involves Product Codes *Sweeny[47,pp 144]*. Here a sequence of data codes is fed into an array. Reed Solomon parity are computed for both the data rows and columns of the array, so enhancing burst error capability. This is illustrated in **Figure 3.2**.

Figure 3.2 : Parity Production In Product Codes

	A	B	C	D	E	row checks	
						P	Q
F	X	X	X	X	√		
G	√	√	√	√	√		
H	√	√	√	√	√		
R							
S							
	column checks					check on checks	

If all the bytes of one codeword were in error denoted by X, then the row checks would be unable to correct the errors. However the column checks use the bytes from the other correct codewords denoted by √. Thus the whole codeword can be re-established byte by byte.

The second combination involves multiple order Reed Solomon encoding which exploits the ability of Reed Solomon codes to either detect and correct symbols in error, or to correct error symbols at known locations. As an example consider the (7,5) Reed Solomon code above. Five data symbols are encoded into a seven symbol codeword. Using CIRC encoding these 7 symbols are treated as data and are now encoded into a nine symbol codeword using a (9,7) code.

The first Reed Solomon code may locate two symbols in error and the second may correct two error symbols of known location. Thus the combination may find and correct up to two data symbols in error. By using an intermediary strategy known as interleaving between the calculation of parity bytes in the (7,5) and (9,7) codes greater error correction is possible *Doi[42,pp 154] & Ramsey[49]*. Interleaving ensures that the error bytes are deposited over a wide number of codewords so enabling greater total error

correction to be applied *Vries*[17,pp 2], *Verterbi*[50,pp 144].

For Reed Solomon codes corresponding symbols of successive codewords are interleaved. As an example, consider the three, 7 - symbol Reed Solomon codewords.

Codeword A : A1 A2 A3 A4 A5 A6 A7

Codeword B : B1 B2 B3 B4 B5 B6 B7

Codeword C : C1 C2 C3 C4 C5 C6 C7

These could be interleaved to produce the symbol sequence

A1 B1 C1 A2 B2 C2 A3 B3 C3 C7

on the medium.

Here a three byte burst error on the medium which effects bytes B1, C1 and A2 will only cause one byte error in each of the three codewords after interleaving. In CDROM applications the more powerful Cross Interleaved techniques are applied. Hence Reed Solomon codewords are produced both before and after interleaving. In the CDROM all symbols used in Reed Solomon codes are eight bit bytes. As an example of Cross Interleaving consider the same seven byte (symbol) Reed Solomon codewords.

Codeword A : A1 A2 A3 A4 A5 A6 A7

Codeword B : B1 B2 B3 B4 B5 B6 B7

Codeword C : C1 C2 C3 C4 C5 C6 C7

Interleaving in a predefined manner the resultant codewords are:

Codeword 1 : A1 B2 C3 A4 B5 C6 A7

Codeword 2 : B1 C2 A3 B4 C5 A6 B7

Codeword 3 : C1 A2 B3 C4 A5 B6 C7

These codewords are now encoded and then recorded on the medium. Any burst errors now effect the all encoded codewords 1-3. The power of the parity symbols of all these codewords is combined to correct some of the errors. The three codewords are then deinterleaved into A,B and C. These are decoded and further error correction may take place. This process is used by the CDROM though in a very much more complex application of the technology. This is discussed in **Chapter Five**.

CHAPTER FOUR

The Production Of Channel Code By Modulation

4.1 The Need For Modulation

In the previous chapter it was seen that errors can cause data loss, but also that sophisticated error correction strategies exist which can result in data recovery. However, errors will not only corrupt the data but will also affect clocking or sampling of the signal. Channel codes, also called modulation codes or recording codes, aim to overcome the problem. The Channel codes may be regarded as the code which is actually written onto the disk.

The modulation code must introduce a further layer of redundancy. For example an eight bit data byte may take any of 256 bit patterns. Without the presence of modulation these patterns would be laid down directly as channel bits on the medium. This would lead to significant problems and introduces major recording errors.

A continuous sequence of identical symbols will disrupt timing recovery, as reclocking occurs at state changes. Repetitions of the same channel bit will also generate DC content (or low frequency content) in the recording signal. The DC content of such a signal is required to be as small as possible in both magnetic and optical recording. In magnetic recording the channel cannot reproduce the low frequencies with sufficient SNR. To minimise distortions on the reproduced data the DC content should be removed by a channel code *Immink[51,pp 99]*. In optical recording the servo systems controlling the laser are sensitive to low-frequency signals. The servo systems for track following and focusing are controlled by low frequency signals, thus low frequency components in the code could interfere with the servo systems *Immink[52,pp 587] Immink[29,pp 410-31] & Immink[51,pp 27]*.

Recording codes must be designed to:

- minimise DC content;
- match the Power Density Spectrum of the encoded data to the frequency response of the channel;
- provide a data clock to facilitate reclocking.

Additional functions include those which are device specific. Thus the eight-fourteen (EFM) code of the CDROM and CD:

- provides positional information for the servo systems;
- enables the system to resynchronise automatically;
- provides additional error detection by detecting channel code violations.

In general such codes attempt to smooth deterioration in signals due to the channel imperfections.

4.2 Simple Channel Codes

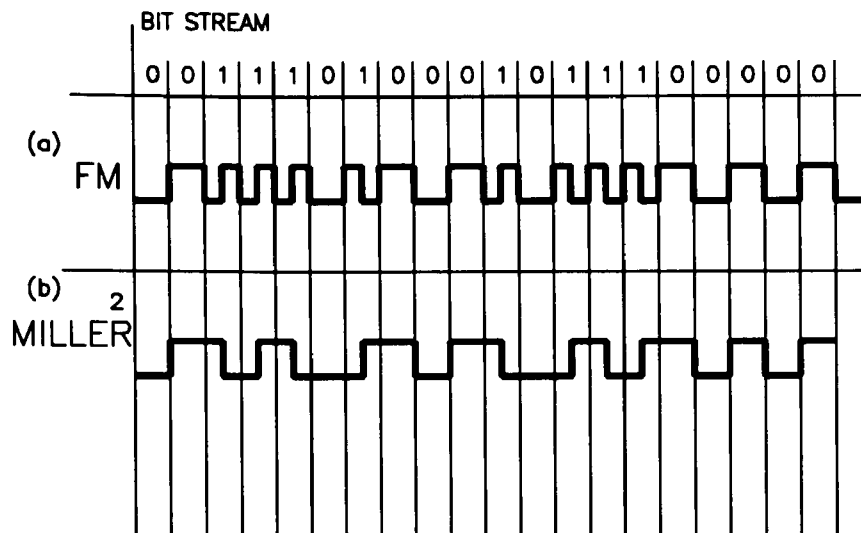
Examples of simple codes are Manchester Encoding and Miller² encoding. Such codes impose their own rules to the data sequence *Mallinson[53]*, *Mackintosh[53]*.

4.2.1 Manchester Encoding

Manchester Encoding manipulates the data code by ensuring that there is always a transition at the bit cell boundary, thus ensuring self clocking. For a data 'one' there is an additional transition at the bit cell centre. Each data bit is thus represented by two recording bits. Although this is not an efficient use of bandwidth, it is highly effective.

Manchester Encoding is efficient since one data bit may be represented by two channel bits. Data recovery is possible for a wide range of speeds. An example of this code is illustrated in **Figure 4.1(a)**. Note that although Manchester Encoding is effective as a code, it is not DC free *Watkinson[11,pp 176]*.

Figure 4.1 : Illustration Of FM and Miller²



4.2.2 Miller² Encoding

The Miller² code minimises the DC content of a data code. It is a modification of Miller. Miller is an extension of Manchester Encoding where bit cell boundary conditions only occur between successive 'zeros'. In Miller² when an even number of 'ones' occur between 'zeros' the transition at the last 'one' is omitted *Watkinson[11,pp 178]*. This code is illustrated in **Figure 4.1(b)**.

4.3 The Block codes

Both the RDAT-DDS and CDROM employ Block Recording Codes. This relies on a codebook method, in which the modulation encoding of a data byte is looked up directly. An m-bit data symbol can be mapped via the codebook to an n-bit channel symbol. In Eight Fourteen Modulation (EFM) an 8-bit data byte is associated with a 14 bit block of recording code *Ogawa[55]*.

4.3.1 Run Length Limited Codes

The number of modulation bit cells between channel transitions is known as the run length and in most recording codes is constrained to lie between fixed maximum and minimum values. This class of codes are known as the

Run-Length-Limited (RLL) Codes. Such codes are introduced to satisfy channel constraints and assist with channel clocking.

EFM is a (2,10)RLL code for which the maximum number of zeros between two channel 1's is 10, the minimum number is 2 *Heemskerk[56] & Immink[52]*. These constraints are introduced so as to aid the sampling *Tang[57]*. If clocking is lost then the a state change will occur after not more than 10, when reclocking is possible. The minimum distance of two exists so as to reduce the Inter Symbol Interference between two channel state changes.

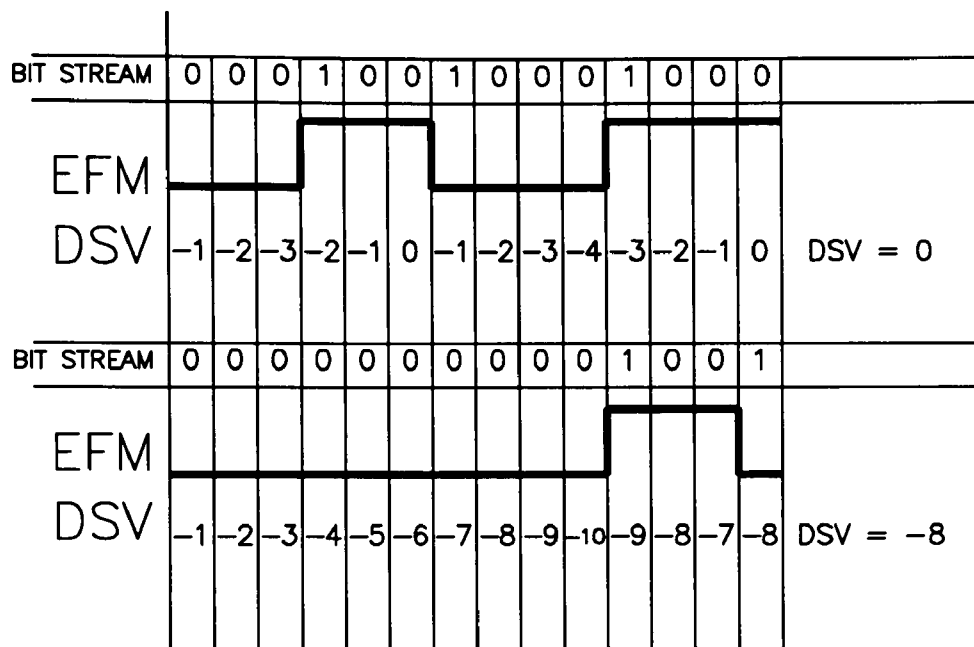
Channel codes are produced from data bytes using a codebook. For EFM there are 2^{14} possible 14 bit codes of which only 267 of these satisfy the desired criteria *Watkinson[58,pp 27]*. For each 8 bit code there is an associated 14 bit channel code, thus only 256 are necessary. In fact 258 are used since two of the unused codes are employed as unique synchronizing codewords. The remaining codes are not used. If they do occur they represent a code violation and indicate an error.

4.3.2 The Digital Sum Variation (DSV)

The DSV of the channel bits is used as a measure of DC suppression and RLL use. The channel codes were designed with this concept in mind, the RLL is also chosen so as to minimise DC *Patel[59]*. The DSV of the channel symbols are determined by adding one for every high channel bit period and removing one for each low channel bit *Watkinson[11,pp 183]*. This is illustrated in **Figure 4.2**. In EFM successive channel symbols are separated by three extra bits, the binary pattern of these being chose to minimise the resultant DSV. In the 8/10 Block code which is used for RDAT each eight data symbols has two associated ten bit channel symbols, one with positive and one with a negative DSV *Fukuda[60]*. Which code is chosen depends on the current DSV total. If the current DSV is positive then the channel symbol with negative DSV content is chosen and vice versa. This is in direct comparison to EFM which has one associated channel symbol but uses merge

bits to select whether it will be negative or positive.

Figure 4.2 : Illustration of DSV Control



An extended discussion of the EFM code of the CDROM is given in **Chapter Five**.

CHAPTER FIVE

The Encoding Processes Of The CDROM

5.1 Introduction

The Compact Disc is an example of the application of the error control strategy discussed in **Chapter Three**. Multiple Reed Solomon Codes are used extensively for error protection together with interleaving strategies and as product codes *Vries[17,pp 8]*.

In this Chapter the Error Detection Code (EDC) and the scrambler are introduced. The EDC is a Cyclic Redundancy Check (CRC) and is the final stage of the error protection strategy. The scrambler produces a pseudo-randomisation of the data in a logical sector, which has the effect of whitening the Power Density Spectrum.

The three stages of the encoding of raw data into channel bits are *Watkinson[61]* :

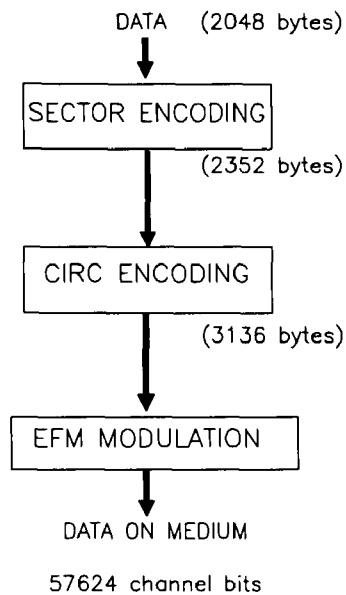
- Sector Encoding.
- CIRC Encoding.
- EFM Encoding.

These stages are defined in specified standards and are discussed in turn *ECMA[18,pp 18-22]*. **Figure 5.1** gives a diagrammatic representation of these processes.

5.2 Sector Encoding

Throughout the encoding process data bits are processed as eight bit bytes which form the eight bit subcode of the Reed solomon codes using GF(28). The digital data to be recorded is thus represented by eight bit bytes which are grouped into logical Sectors. This comprises of the raw data, synchronisation bytes and header and as shown in **Figure 5.2**.

Figure 5.1 : The Compact Disc Encoding Processes



The area on the disc where data is stored is called the physical Sector and is the smallest part of the Information Area that can be independently addressed. The sectors are encoded, modulated and laid down on the medium as Sections. A Section is thus the physical representation of the data which originated from a logical Sector. The size in bytes of each component of a logical Sector is shown in **Figure 5.2**.

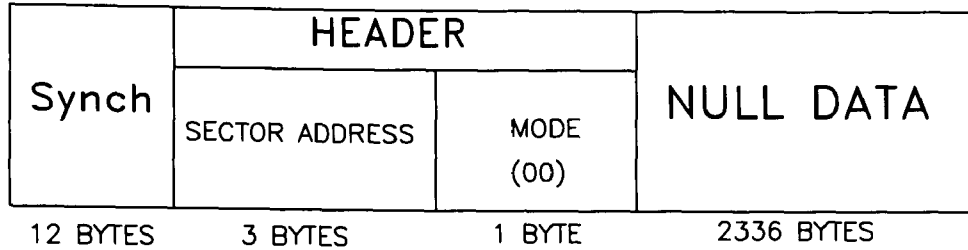
5.2.1 The Difference Between The CD and the CDROM

In the CD-ROM there is a need for greater data protection, thus two further Reed Solomon Codes are used, each adding two bytes of redundancy *Sako[35,pp 3997]*. An EDC is also used in the CD-ROM. Both these additional strategies are applied when data is placed into a logical sector. Hence the sector configuration of the CDROM will differ from that of the CD.

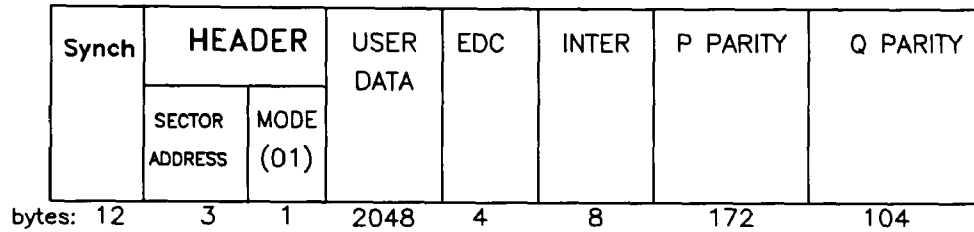
There are three possible layouts depending upon the setting of the Sector Mode byte as shown in **Figure 5.2**. **Figure 5.2(a)** depicts Mode 0 where no data is stored. Such sectors are found in the lead in and lead out areas of the disc. The lead in and lead out areas of the disc are regions where no data is stored. **Figure 5.2(b)** portrays Mode 1 which

Figure 5.2 : The Three Sector Configurations

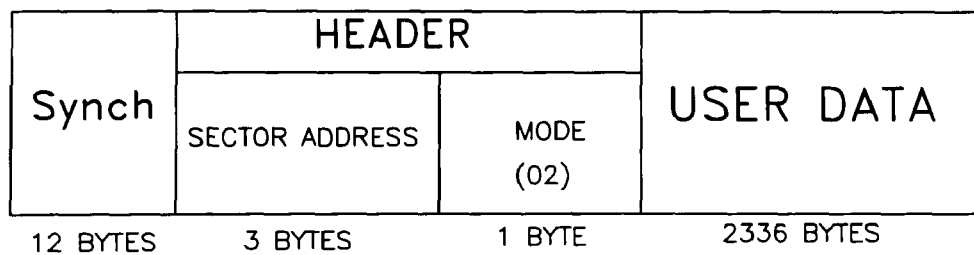
(a) SECTOR MODE 00 : NO DATA PRESENT
SECTOR 2352 BYTES



(b) SECTOR MODE 01 : CDROM DATA PRESENT
SECTOR 2352 BYTES

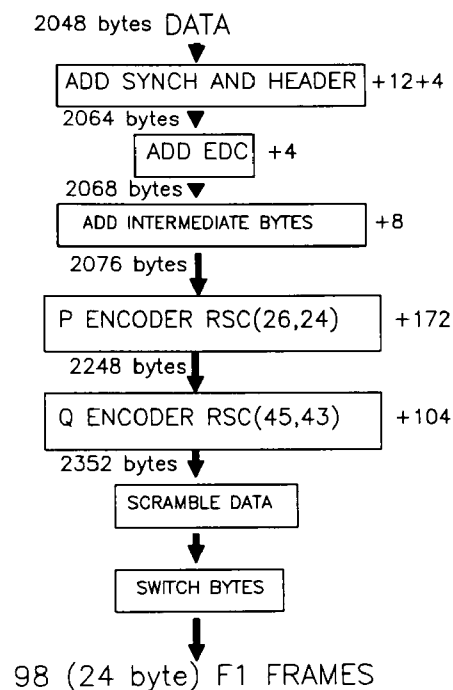


(c) SECTOR MODE 02 : CD DATA PRESENT
SECTOR 2352 BYTES



indicates that CDROM data is present and that there are three extra levels of error protection. Finally **Figure 5.2(c)** illustrates Mode 2. This is the mode used by the CD where only the CIRC protects the data. In this case there is less protection, hence less redundancy and more data can be stored.

Figure 5.3 : The Sector Encoding Processes



Although the section layout differs between CD and CDROM the sector sizes are the same. Hence hardware associated with the CDROM can communicate with the CD format, but not vice versa.

The extra error protection is the only structural difference between the CDROM and the CD. The CDROM requires the extra layers of protection to ensure better data retrieval *Sako[35,pp 3996]*. Whereas the CD has applications when it is possible to interpolate data, this is not the case with

the CDROM where the application is digital data storage *Hoeve[32,pp 171]*. The various stages of encoding the 2048 bytes of data in a CDROM sector are as illustrated in **Figure 5.3**. All these stages shall be reviewed in turn.

5.2.2 The Synchronisation Field

The Synchronisation field is a twelve byte block that acts as an identifier and signifies that the beginning of a sector has been located. The synchronisation field is identical for all sectors as it is unrelated to the data present. It informs the hardware that important information is following, also providing a fixed time for the hardware to synchronise to the signal.

5.2.3 The Header Field

The header field comprises of a three byte Sector Address and the single Mode byte which indicates the extent of the error protection which is to be used and also the quantity of data.

The Sector Address contains the Physical Address of the sector. This is represented by the elapsed time from the beginning of the User Data Area, in minutes, seconds and fractions. The User Data Area is the area on the disk with physical tracks containing data as opposed to bytes for format overheads.

5.2.4 The Error Detection Code (EDC)

This four byte code is a 32 bit CRC (Cyclic Redundancy Check), where the EDC codeword must be divisible by the check polynomial $P(x)$ *ECMA[18,pp 20]*:

$$P(x) = (x^{16} + x^{15} + x^2 + 1) \cdot (x^{16} + x^2 + x + 1) \quad (5.1)$$

Each byte of data is applied to the polynomial such that two values are found, the remainder and the quotient. These are stored as the CRC *Doi[42,pp 149]*. If on decoding the calculated values do not agree with the

CRC either the data is erroneous or the CRC is in error.

The intermediate field adds eight bytes of null data which are not used at present. It is there for any future requirements.

5.2.5 The P-Parity Field

The P-Parity field consists of 172 bytes of redundancy, these are calculated using a (26,24) Reed Solomon Code on bytes 12-2075 *Sako[35,p 3999]*. Using the 2064 remaining bytes, there are eighty-six 24-byte frames. Two parity bytes are calculated for each of these thus 172 parity bytes are produced. The bytes are ordered into 1032 words each of two 8-bit bytes, each consisting of a Most Significant Byte (MSB) and a Least Significant Byte (LSB). For example byte 0 and byte 1 are the constituent bytes of word one, byte 0 being the MSB and byte 1 being the LSB.

Figure 5.4 : Matrix For P Parity Calculation

	0	1	2	41	42		
0	0000	0001	0002		0041	0042	DATA input into matrix	
1	0043	0044	0045		0084	0085		
2	0086	0087						
3	0129							
:								
:								
:	axis of matrix							
:	P Parity is							
:	calculated along							
:	(column)							
:	▼							
23	0989	0990			1030	1031		
24	1032				1073	1074		P Parity
25	1075				1116	1117		

Two equally sized matrices are constructed with 43 columns and 28 rows, into which the bytes are fed row by row as shown in **Figure 5.4**. One matrix is filled with the MSB whereas the other is arranged with the LSB,

both have the same pattern as illustrated.

The redundancy can now be added using a Reed Solomon Code (26,24) applied as a Product code, the GF(2⁸) field is generated by the primitive polynomial where :

$$P(x) = (x^8 + x^4 + x^3 + x^2 + 1) \quad (5.2)$$

The primitive element α of GF(2⁸) = (00000010), in which the right most bit is the least significant bit. The GF and polynomial generator are the same for all the RSC used by the CD and CDROM.

The parity bytes are produced using **Equation 5.3**.

$$H_p * V_p = 0 \quad (5.3)$$

Here:

V_p is the vector containing the position in the matrix of each of the 26 bytes which is used. Each byte is obtained from subsequent rows of the matrix. For example the first element of the vector is the first byte of a column and so on. It's structure is displayed in **Figure 5.5**.

Figure 5.5 : V_p Vector

$$V_p^T = [A_1 \ B_1 \ C_1 \ D_1 \ . \ . \ . \ . \ . \ T_1 \ U_1 \ V_1 \ W_1 \ X_1]$$

H_p is the parity check matrix, it contains the powers of α by which each of the 26 bytes must be multiplied. It's structured is displayed in **Figure 5.6 ECMA[18,pp 32]**.

Figure 5.6 : H_p Matrix

$$H_P = \begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \dots & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^{25} & \alpha^{24} & \alpha^{23} & \dots & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 & \alpha^0 \end{bmatrix}$$

By rearranging the equations in terms of the parity bytes, the following equations are produced.

$$\begin{aligned} P1 = & \alpha^{231}A_1 + \alpha^{229}B_1 + \alpha^{171}C_1 + \alpha^{210}D_1 + \alpha^{240}E_1 + \alpha^{17}F_1 + \alpha^{67}G_1 + \alpha^{215}H_1 \\ & + \alpha^{43}I_1 + \alpha^{120}J_1 + \alpha^8K_1 + \alpha^{199}L_1 + \alpha^{74}M_1 + \alpha^{102}N_1 + \alpha^{220}O_1 + \alpha^{251}P_1 \\ & + \alpha^{95}Q_1 + \alpha^{175}R_1 + \alpha^{87}S_1 + \alpha^{166}T_1 + \alpha^{113}U_1 + \alpha^{75}V_1 + \alpha^{198}W_1 \\ & + \alpha^{25}X_1. \end{aligned} \quad (5.4)$$

$$\begin{aligned} P2 = & \alpha^{230}A_1 + \alpha^{172}B_1 + \alpha^{211}C_1 + \alpha^{241}D_1 + \alpha^{18}E_1 + \alpha^{68}F_1 + \alpha^{216}G_1 + \alpha^{44}H_1 \\ & + \alpha^{121}I_1 + \alpha^9J_1 + \alpha^{200}K_1 + \alpha^{75}L_1 + \alpha^{103}M_1 + \alpha^{221}N_1 + \alpha^{252}O_1 + \alpha^{96}P_1 \\ & + \alpha^{176}Q_1 + \alpha^{88}R_1 + \alpha^{167}S_1 + \alpha^{114}T_1 + \alpha^{76}U_1 + \alpha^{199}V_1 + \alpha^{26}W_1 \\ & + \alpha^1X_1. \end{aligned} \quad (5.5)$$

In these equations the letters represent the 24 bytes originating from any given column of the matrix of Figure 5.4, A_1 is the first byte and X_1 is the 24th byte.

Thus the two parity bytes for a column are found. This is repeated for all 43 columns. The parity bytes are inserted into the matrix in the positions as illustrated in Figure 5.4. For example the two parity bytes associated with the first column are labelled 1032 and 1075, these are placed into the matrix in this order, as illustrated.

5.2.6 The Q-Parity Field

The Q-Parity field consists of 104 bytes of redundancy being added using a RSC(45,43), on bytes 12-2247 *Sako*[35,p 3999]. Using the 2236 remaining bytes, there are fifty-two 43-byte codewords, two parity bytes are calculated for each of these thus 104 parity bytes are produced.

Figure 5.7 : Matrix For Q and P Parity Calculation

	0	1	2	41	42		
0	0000	0001	0002			0041 0042	DATA input into matrix	
1	0043	0044	0045			0084 0085		
2	0086	0087						
3	0129							
:								
:								
:								
:								
:								
:								
:								
:								
:								
23	0989	0990				1030 1031		
24	1032					1073 1074		P Parity
25	1075					1116 1117		
26	1118				1143	Q Parity		
27	1144				1169			
	0	1	2	25			

axis of matrix P Parity is calculated along (column)

axis of matrix Q Parity is calculated along (diagonal)

The same process occurs as with P-Parity. This time, however there will be 43 columns and 26 rows *ECMA[18,pp 33]* as show in **Figure 5.7**. Previously the bytes were fed into vector V column by column. However as with all product codes the redundancy is determined along a different axis than previously used.

In the Q-Parity process the group of bytes operated upon are on the axis parallel to the diagonals of the matrix, this is identical to that used in the P-Parity plus the added parity. The matrix illustrated in **Figure 5.8** is obtained when this process occurs.

The redundancy equations are produced in the usual manner using **Equation 5.6**.

$$H_Q * V_Q = 0 \tag{5.6}$$

V_Q is the vector containing the position in the matrix of each of the 43 bytes which is used.

H_Q is the parity check matrix. It contains the powers of α with which to multiply the 45 bytes as displayed in Figure 5.9 ECMA[18,pp 34].

Figure 5.8 : Q matrix Parity Calculation

	0	1	2	40	41	42	Q0	Q1
0	0000	0044	0088			0686	0730	1118	1144
1	0043	0087	0131			0729	0773	1119	1145
2									
.....									
23									
24									
25	1075	0001	0045			0643	0687	1143	1169

The Q parity bytes are calculated using the LSB and MSB bytes along the diagonals of the matrix

Figure 5.9 : H_Q Matrix

$$H_Q = \begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \dots & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^{44} & \alpha^{43} & \alpha^{42} & \dots & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 & \alpha^0 \end{bmatrix}$$

The parity generating equations were found to be as illustrated in Equation 5.7 and 5.8 ECMA[18,pp 33].

$$\begin{aligned}
 Q1 = & \alpha^{215}A_2 + \alpha^{121}B_2 + \alpha^{20}C_2 + \alpha^{157}D_2 + \alpha^{84}E_2 + \alpha^{106}F_2 + \alpha^{184}G_2 + \\
 & \alpha^{179}H_2 + \alpha^{225}I_2 + \alpha^{32}J_2 + \alpha^{136}K_2 + \alpha^{15}L_2 + \alpha^{35}M_2 + \alpha^{45}N_2 + \\
 & \alpha^{66}O_2 + \alpha^{181}P_2 + \alpha^{193}Q_2 + \alpha^{104}R_2 + \alpha^{198}S_2 + \alpha^{231}A_1 + \alpha^{229}B_1 + \\
 & \alpha^{171}C_1 + \alpha^{210}D_1 + \alpha^{240}E_1 + \alpha^{17}F_1 + \alpha^{67}G_1 + \alpha^{215}H_1 + \alpha^{43}I_1 + \\
 & \alpha^{120}J_1 + \alpha^8K_1 + \alpha^{199}L_1 + \alpha^{74}M_1 + \alpha^{102}N_1 + \alpha^{220}O_1 + \alpha^{251}P_1 + \\
 & \alpha^{95}Q_1 + \alpha^{175}R_1 + \alpha^{87}S_1 + \alpha^{166}T_1 + \alpha^{113}U_1 + \alpha^{75}V_1 + \alpha^{198}W_1 + \\
 & \alpha^{25}X_1.
 \end{aligned} \tag{5.7}$$

$$\begin{aligned}
Q2 = & \alpha^{97}A_2 + \alpha^{251}B_2 + \alpha^{133}C_2 + \alpha^{60}D_2 + \alpha^{82}E_2 + \alpha^{160}F_2 + \alpha^{155}G_2 + \\
& \alpha^{201}H_2 + \alpha^8I_2 + \alpha^{112}J_2 + \alpha^{246}K_2 + \alpha^{11}L_2 + \alpha^{21}M_2 + \alpha^{42}N_2 + \\
& \alpha^{157}O_2 + \alpha^{169}P_2 + \alpha^{80}Q_2 + \alpha^{174}R_2 + \alpha^{232}S_2 + \alpha^{230}A_1 + \alpha^{172}B_1 + \\
& \alpha^{211}C_1 + \alpha^{241}D_1 + \alpha^{18}E_1 + \alpha^{68}F_1 + \alpha^{216}G_1 + \alpha^{44}H_1 + \alpha^{121}I_1 + \\
& \alpha^9J_1 + \alpha^{200}K_1 + \alpha^{75}L_1 + \alpha^{103}M_1 + \alpha^{221}N_1 + \alpha^{252}O_1 + \alpha^{96}P_1 + \\
& \alpha^{176}Q_1 + \alpha^{88}R_1 + \alpha^{167}S_1 + \alpha^{114}T_1 + \alpha^{76}U_1 + \alpha^{199}V_1 + \alpha^{26}W_1 + \\
& \alpha^1X_1.
\end{aligned} \tag{5.8}$$

The power of the RSC and the way in which they have been applied will prove highly significant in the decoding processes. The parity is fed into the matrix in the manner described in **Figure 5.7**.

This process is repeated for all axis parallel to the diagonal of each matrix in turn. The parity bytes are inserted into the matrix in the positions as illustrated in **Figure 5.8**. For example the two parity bytes associated with the first diagonal are labelled 1118 and 1144. The bytes are placed back into the sector by reading each MSB and LSB byte from the matrices row by row.

5.2.7 Scrambling The Data

A regular bit pattern fed into the EFM encoder can cause large values of the DSV which cannot be reduced by the merging bit strategy. The scrambler reduces the risk by converting bytes 12-2351 of a Sector in a predefined manner, such that the original data can be reclaimed. Each bit of the input stream of the scrambler is added modulo 2 to the least significant bit of a maximum length register.

The 15-bit register is illustrated in **Figure 5.10** is of the parallel block synchronized type, and fed back according to the polynomial ($x^{15} + x + 1$) *ECMA[18,pp 35]*.

As each bit, least significant bit (lsb) first is passed through the scrambler it is added, modulo 2 to the contents of the lsb of the register. The lsb is altered in accordance with the polynomial.

Figure 5.10 : The Scrambler Circuit

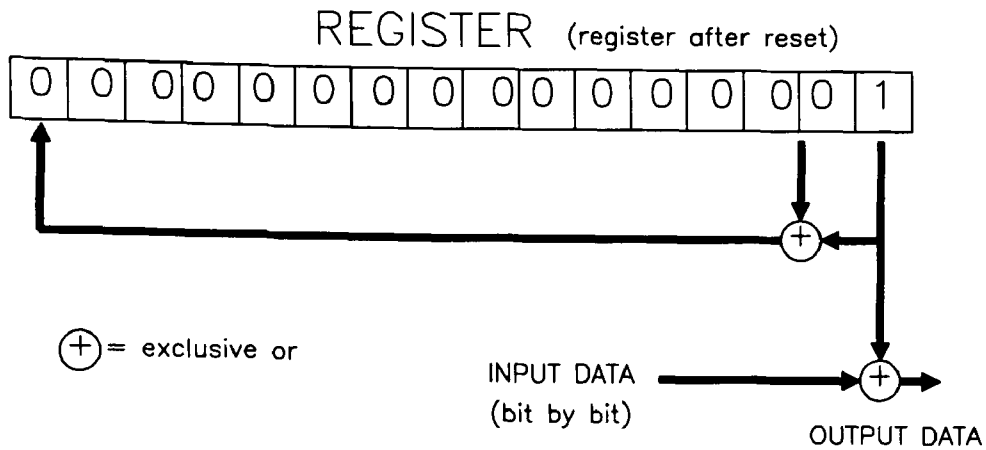


Figure 5.11 : Example Of The Circuit On Real Data

DATA IN	REGISTER														⊕	DATA OUT		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

This process continues until the register is reset by a synch field

Figure 5.11 illustrates how this would work in practice. Before being fed into the CIRC encoder each scrambled sector is mapped onto a series (98) of consecutive 24-byte frames. The consecutive bytes of each frame are switched as follows *ECMA[18,pp 20]*:

For example:

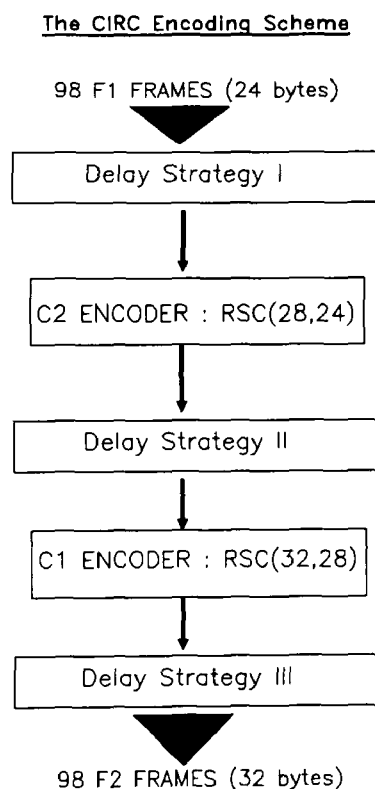
(Frame) 1 2 3 4 5 6.....23 24 → (F1 Frame) 2 1 4 3 6 5.....24 23

This byte interchanged frame is known as the F1 Frame and it is these frames which are the input to the CIRC encoder.

5.3 The CIRC (Cross Interleaved Reed-Solomon Code) Scheme

CIRC consists of two encoding processes which use RSC not dissimilar to those seen in **Chapter Three**. However much greater redundancy is introduced *Hoeve[32]*, *Vries[62]*, *Doi[42,pp 170]*, *Driessen[63,pp 386]* & *Doi[64]*. Instead of the RSC being used as product codes three interleaving stages are employed. **Figure 5.12** illustrates the various stages which occur in the CIRC encoding scheme.

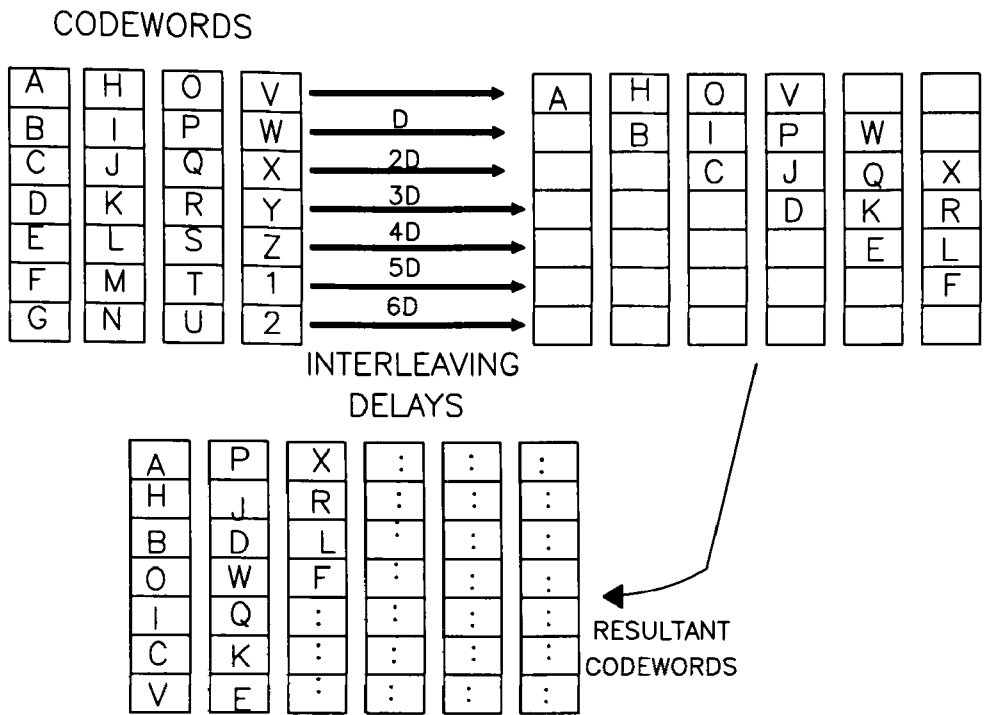
Figure 5.12 : The CIRC Encoding Processes



5.3.1 Interleaving Delay Strategy I

All interleaving strategies in CIRC are based upon delaying schemes much like that illustrated in **Figure 5.13** *Watkinson[61,pp 80]*, *Doi[42,pp 155]*.

Figure 5.13 : An Example of Delay



In the example codewords consisting of seven bytes are input into a delaying circuit. The delay experienced by each byte will be determined by its position in the codeword. Byte one will have no delay, byte two experiences a one codeword delay and so on. In the example byte B is observed to be delayed by one codeword and byte E to be delayed by four codeword times.

The first stage interleave introduces a two 24-byte frame delay between odd and even samples. At the end of this stage the even bytes have all been displaced by two 24 byte frames *ECMA[18,pp 37]*.

5.3.2 The C2 Encoder

The delayed bytes are fed into the first of the two Reed Solomon Encoders. This encoder uses a (28,24)RSC, generating four parity bytes referred to as the Q Parity Bytes (Q1-Q4). This labelling is unfortunate as it is similar to the

identification of the Q-Parity Code in the logical sector coding; it is completely unrelated. As was seen with the previous Reed Solomon Codes the parity equations satisfy **Equation 5.9**:

$$H_Q * V_Q = 0 \quad (5.9)$$

Where:

V_Q is the vector containing the bytes from the same frame which are to be used in the operation. For example the first element of vector V_Q is the first byte of the codeword, the last element being the last byte. The structure is displayed in **Figure 5.14**.

Figure 5.14 : V_Q Vector

$$V_P^T = [A_1 \ B_1 \ C_1 \ D_1 \ . \ . \ . \ . \ . \ T_1 \ U_1 \ V_1 \ W_1 \ X_1]$$

H_Q is the parity check matrix, the structured of which is displayed in **Figure 5.15 ECMA[18,pp 38]**.

Figure 5.15 : H_Q Matrix

$$H_Q = \begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & . & . & . & . & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^{27} & \alpha^{26} & \alpha^{25} & . & . & . & . & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 & \alpha^0 \\ \alpha^{54} & \alpha^{52} & \alpha^{50} & . & . & . & . & \alpha^8 & \alpha^6 & \alpha^4 & \alpha^2 & \alpha^0 \\ \alpha^{81} & \alpha^{78} & \alpha^{75} & . & . & . & . & \alpha^{12} & \alpha^9 & \alpha^6 & \alpha^3 & \alpha^0 \end{bmatrix}$$

By use of this information the following parity equations can be determined. The parity equations were found to be :

$$\begin{aligned} Q1 = & \alpha^{58}A_1 + \alpha^{152}B_1 + \alpha^{173}C_1 + \alpha^{95}D_1 + \alpha^{88}E_1 + \alpha^{43}F_1 + \alpha^{134}G_1 + \\ & \alpha^{205}H_1 + \alpha^{143}I_1 + \alpha^{131}J_1 + \alpha^{163}K_1 + \alpha^{75}L_1 + \alpha^{249}M_1 + \alpha^{66}N_1 + \\ & \alpha^{151}O_1 + \alpha^{116}P_1 + \alpha^{125}Q_1 + \alpha^{184}R_1 + \alpha^{110}S_1 + \alpha^{16}T_1 + \alpha^{58}U_1 + \\ & \alpha^{62}V_1 + \alpha^{137}W_1 + \alpha^{113}X_1. \end{aligned} \quad (5.10)$$

$$\begin{aligned}
Q2 = & \alpha^{30}A_1 + \alpha^{214}B_1 + \alpha^{148}C_1 + \alpha^{138}D_1 + \alpha^{112}E_1 + \alpha^{154}F_1 + \alpha^{157}G_1 + \\
& \alpha^{96}H_1 + \alpha^{49}I_1 + \alpha^{198}J_1 + \alpha^{189}K_1 + \alpha^{249}L_1 + \alpha^{69}M_1 + \alpha^{47}N_1 + \\
& \alpha^{147}O_1 + \alpha^{235}P_1 + \alpha^{156}Q_1 + \alpha^{47}R_1 + \alpha^{209}S_1 + \alpha^{183}T_1 + \alpha^{138}U_1 + \\
& \alpha^{232}V_1 + \alpha^{205}W_1 + \alpha^{120}X_1.
\end{aligned} \tag{5.11}$$

$$\begin{aligned}
Q3 = & \alpha^{162}A_1 + \alpha^{244}B_1 + \alpha^{13}C_1 + \alpha^{171}D_1 + \alpha^{213}E_1 + \alpha^{236}F_1 + \alpha^{71}G_1 + \\
& \alpha^{177}H_1 + \alpha^{253}I_1 + \alpha^{162}J_1 + \alpha^{59}K_1 + \alpha^{78}L_1 + \alpha^{243}M_1 + \alpha^{180}N_1 + \\
& \alpha^{186}O_1 + \alpha^{34}P_1 + \alpha^{78}Q_1 + \alpha^{136}R_1 + \alpha^{130}S_1 + \alpha^{85}T_1 + \alpha^{108}U_1 + \\
& \alpha^{115}V_1 + \alpha^{178}W_1 + \alpha^{246}X_1.
\end{aligned} \tag{5.12}$$

$$\begin{aligned}
Q4 = & \alpha^{158}A_1 + \alpha^{179}B_1 + \alpha^{101}C_1 + \alpha^{94}D_1 + \alpha^{49}E_1 + \alpha^{140}F_1 + \alpha^{211}G_1 + \\
& \alpha^{149}H_1 + \alpha^{137}I_1 + \alpha^{169}J_1 + \alpha^{81}K_1 + \alpha^6L_1 + \alpha^{72}M_1 + \alpha^{157}N_1 + \\
& \alpha^{122}O_1 + \alpha^{131}P_1 + \alpha^{190}Q_1 + \alpha^{116}R_1 + \alpha^{22}S_1 + \alpha^{64}T_1 + \alpha^{68}U_1 + \\
& \alpha^{143}V_1 + \alpha^{119}W_1 + \alpha^{22}X_1.
\end{aligned} \tag{5.13}$$

For each 24-byte frame entering the encoder there now exists a 28-byte Frame. The parity bytes are inserted into the centre of the Frame with twelve data bytes going to either side. This is a characteristic of audio recording, by doing this the odd/even delay of two blocks permits greater interpolation.

5.3.3 Interleaving Strategy II

This scheme subjects each byte of the codeword to a differing measure of delay. The delay algorithm being:

The Delay of a Byte = (Byte number) * 4 Frames ECMA[18,pp 37].

Thus byte 0 experiences no delay and byte 27 experiences a (27*4) 108 28-byte Frame delay.

5.3.4 The C1 Encoder

This encoder generates a (32,28)RSC, creating four parity bytes referred to as the P Parity Bytes P1-P4. Again the labelling is unfortunate ECMA[18,pp 37].

The parity byte equations must conform to equation 5.14 :

$$H_p * V_p = 0 \quad (5.14)$$

Where:

H_p is the parity check matrix, the structured of which is displayed in **Figure 5.16 ECMA[18,pp 38]**.

Figure 5.16 : H_p Matrix

$$H_p = \begin{bmatrix} \alpha^0 & \alpha^0 & \alpha^0 & \dots & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 & \alpha^0 \\ \alpha^{31} & \alpha^{30} & \alpha^{29} & \dots & \alpha^4 & \alpha^3 & \alpha^2 & \alpha^1 & \alpha^0 \\ \alpha^{62} & \alpha^{60} & \alpha^{58} & \dots & \alpha^8 & \alpha^6 & \alpha^4 & \alpha^2 & \alpha^0 \\ \alpha^{93} & \alpha^{90} & \alpha^{87} & \dots & \alpha^{12} & \alpha^9 & \alpha^6 & \alpha^3 & \alpha^0 \end{bmatrix}$$

V_p is the vector containing the bytes from the same frame, used in the operation. This vector is organised in the manner illustrated in **Figure 5.17**.

Figure 5.17 : V_p Vector

$$V_p^T = [A_1 \ B_1 \ C_1 \ D_1 \ \dots \ W_1 \ X_1 \ Y_1 \ Z_1 \ A_2 \ B_2]$$

By use of this information the following parity equations are found to be:

$$\begin{aligned} P1 = & \alpha^{249}A_1 + \alpha^{142}B_1 + \alpha^{180}C_1 + \alpha^{197}D_1 + \alpha^5E_1 + \alpha^{155}F_1 + \alpha^{153}G_1 + \\ & \alpha^{132}H_1 + \alpha^{143}I_1 + \alpha^{244}J_1 + \alpha^{101}K_1 + \alpha^{76}L_1 + \alpha^{102}M_1 + \alpha^{155}N_1 + \\ & \alpha^{203}O_1 + \alpha^{104}P_1 + \alpha^{58}Q_1 + \alpha^{152}R_1 + \alpha^{173}S_1 + \alpha^{95}T_1 + \alpha^{88}U_1 + \\ & \alpha^{43}V_1 + \alpha^{134}W_1 + \alpha^{205}X_1 + \alpha^{143}Y_1 + \alpha^{131}Z_1 + \alpha^{163}A_2 + \alpha^{75}B_2. \end{aligned} \quad (5.15)$$

$$\begin{aligned} P2 = & \alpha^{205}A_1 + \alpha^{252}B_1 + \alpha^{218}C_1 + \alpha^{199}D_1 + \alpha^{202}E_1 + \alpha^{41}F_1 + \alpha^{136}G_1 + \\ & \alpha^{106}H_1 + \alpha^{119}I_1 + \alpha^{238}J_1 + \alpha^{193}K_1 + \alpha^{103}L_1 + \alpha^{123}M_1 + \alpha^{242}N_1 + \\ & \alpha^{83}O_1 + \alpha^{178}P_1 + \alpha^{30}Q_1 + \alpha^{214}R_1 + \alpha^{148}S_1 + \alpha^{138}T_1 + \alpha^{112}U_1 + \end{aligned}$$

$$\alpha^{154}V_1 + \alpha^{157}W_1 + \alpha^{96}X_1 + \alpha^{49}Y_1 + \alpha^{198}Z_1 + \alpha^{189}A_2 + \alpha^{249}B_2(5.16)$$

$$\begin{aligned} P3 = & \alpha^{67}A_1 + \alpha^{11}B_1 + \alpha^{131}C_1 + \alpha^{40}D_1 + \alpha^7E_1 + \alpha^{41}F_1 + \alpha^{80}G_1 + \\ & \alpha^{147}H_1 + \alpha^{151}I_1 + \alpha^{17}J_1 + \alpha^{245}K_1 + \alpha^{253}L_1 + \alpha^{208}M_1 + \alpha^{66}N_1 + \\ & \alpha^{228}O_1 + \alpha^{116}P_1 + \alpha^{162}Q_1 + \alpha^{244}R_1 + \alpha^{13}S_1 + \alpha^{171}T_1 + \alpha^{213}U_1 + \\ & \alpha^{236}V_1 + \alpha^{71}W_1 + \alpha^{177}X_1 + \alpha^{253}Y_1 + \alpha^{162}Z_1 + \alpha^{59}A_2 + \alpha^{78}B_2. (5.17) \end{aligned}$$

$$\begin{aligned} P4 = & \alpha^{148}A_1 + \alpha^{186}B_1 + \alpha^{203}C_1 + \alpha^{11}D_1 + \alpha^{161}E_1 + \alpha^{159}F_1 + \alpha^{138}G_1 + \\ & \alpha^{149}H_1 + \alpha^{250}I_1 + \alpha^{107}J_1 + \alpha^{82}K_1 + \alpha^{108}L_1 + \alpha^{161}M_1 + \alpha^{209}N_1 + \\ & \alpha^{110}O_1 + \alpha^{64}P_1 + \alpha^{158}Q_1 + \alpha^{179}R_1 + \alpha^{101}S_1 + \alpha^{94}T_1 + \alpha^{49}U_1 + \\ & \alpha^{140}V_1 + \alpha^{211}W_1 + \alpha^{149}X_1 + \alpha^{137}Y_1 + \alpha^{169}Z_1 + \alpha^{81}A_2 + \alpha^6B_2. (5.18) \end{aligned}$$

Where each letter indicates a byte from the vector V. Each 28-byte frame is thus augmented to a 32-byte frame, the four parity bytes having been appended to each frame.

5.3.5 Interleaving Strategy III

The third and last delay strategy delays alternate bytes of the 32-byte frames by one frame *ECMA[18,pp 38]*. The result of CIRC encoding/interleaving is that the ninety eight 24-byte Frames are augmented to 32-byte F2-Frames *ECMA[18,pp 21]*.

5.3.6 The Control Byte

An extra byte is added to the beginning of each F2-Frame thus yielding a 33-byte F3 Frame. This is the Control byte and is added for addressing purposes *Watkinson[65], ECMA[18,pp 21]*. A group of 98 F3 Frames are collectively known as a Section. The control bytes from each of the 98 frames have an associated table for the purposes of addressing.

5.4 The Eight Fourteen Modulation (EFM) Code

As discussed in Chapter Four the Modulation code of the Compact Disc must operate under a number of restrictions *Vries*[33,pp 2], *Watkinson*[58,pp 27]. The DC content of the code is required to be as small as possible for a number of reasons associated with disc technology *Ogawa*[55,pp 118]. DC content in the code will appear as noise in the tracking. The optical servo systems that position the laser spot are also sensitive to low frequency content *Watkinson*[22,pp 70-71] & *Immink*[52].

The EFM scheme records the F3 Frames on the disc with each 8-bit byte being represented by a 14-bit channel byte *Doi*[66,pp 235], *ECMA*[18,pp 43]. Each F3 Frame is thus represented by a Channel Frame. To enable self clocking the data remains as a Frame. This consists of a synchronisation header, Merging bits and thirty three 14-bit channel bytes. This process can be separated into three stages *ECMA*[18,pp 21-2]:

- The 8-bit byte mapping to a 14-bit channel Byte.
- Selection of Merging Bits.
- Coding onto The Medium.

5.4.1 Eight To Fourteen Encoding

EFM is a self clocking Run Length Limited Block Modulation Code which conforming to the rules of such codes *Immink*[67] ,*Vries*[33,pp 2&10]. A lookup table is used which holds the 256 combination of eight bits. Each 8-bit byte is thus mapped to a 14-bit channel byte. Contiguous channel bytes comply with the (2,10)RLL block code constraints. Between two channel 'ones' there is at least two channel zeros and at most ten. This is in order to retain a reasonable clock content in the signal whilst providing acceptable immunity to jitter *Peek*[30,pp 8].

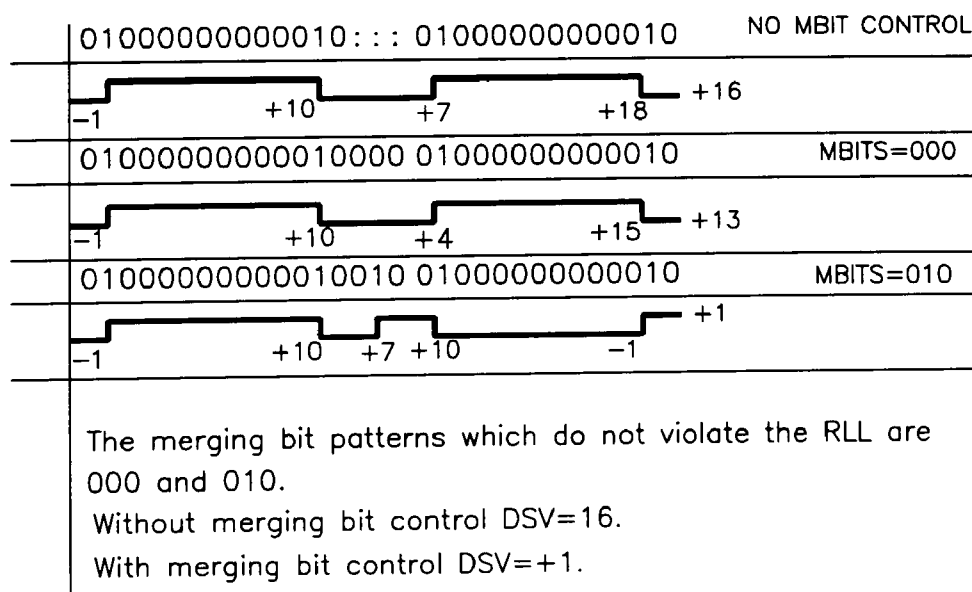
5.4.2 Selection Of The Merging Bits

If successive channel bytes are written onto the medium the Digital Sum Variation (DSV) will vary greatly. The DSV is required to be as close to zero as possible. In order to achieve a minimal DSV three merging bits are inserted between successive channel bytes *Immink[67,pp 64]*. The value of each bit is altered so as to minimise the DSV.

Merging bits and their adjacent channel bytes must also conform to the RLL constraints of the code. Thus if ten 'zeros' have preceded the merging bits then the first merge bit must be a 'one'. Equally, if the last bit before the merging bits is a 'one' then merging bits one and two must be 'zeros'. Although there are three merging bits there are only four possible combinations which obey the RLL guidelines.

Figure 5.18 shows examples of merging bits where two successive channel bytes are merged (a) without DSV control; (b) with DSV control. With the control strategy in place there are at most eight combinations of the merging bits, though of them some will violate the RLL constraints of the code. If this process were occurring in the middle of the code then the initial DSV would arise from the previous minimisation.

Figure 5.18 : Control Of DSV By Use Of Merging Bits

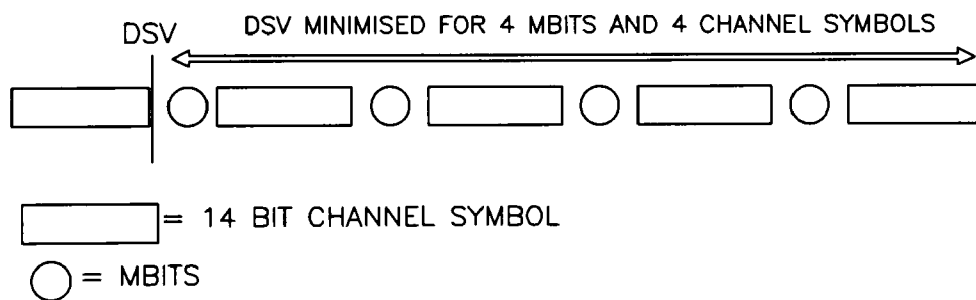


Using the same system the next minimisation which occurs after that illustrated would begin with an initial DSV value of +1.

The benefits of the use of Merging bits can be developed further by using a look ahead method. Here the DSV is averaged over several successive channel bytes *Immink[67]*. At present EFM uses a look-a-head method of four bytes . This is illustrated in **Figure 5.19**.

Each time a channel symbol and associated merging bits are selected the resulting DSV is noted. This DSV value is employed as the starting value for the DSV calculations when minimisation occurs. If the noted DSV value is -4 then minimisation will proceed starting from this value. If the first bit is a 'zero' the resulting DSV for this bit is -5; if it is a 'one' then the resulting DSV is -3. The four channel symbols are combined with all possible viable merging bit combinations.

Figure 5.19 : Minimisation Using Multiple Look-ahead



With a maximum of 2^3 combinations for each four Merging bit sets there are $(2^3)^4$, i.e. 2^{12} , possible combinations of channel symbols and merge bits. The combination which gives the minimum DSV will be chosen. The first merge bit and byte set are written and the resulting DSV noted. The three remaining bytes are shifted forward such that second becomes first, third becomes second and fourth becomes third. The next sequential 8-bit byte is encoded into a 14-bit channel symbol. This becomes the fourth symbol. The process is repeated using the four new channel bytes and the

new DSV. This occurs for each four byte set in each frame.

Clearly by extending the number of bytes considered at any one time to a greater number a greater system averaging could be achieved, but at the cost of increasing the processing time of the encoding process.

5.4.3 Storing the Channel Frames On The Medium

The three stages, 8-to-14 encoding, Merge bits selection by minimisation and storage organisation, occur in conjunction with one another. Each of the ninety eight F3 Frame is converted to a Channel Frame with the following configuration *ECMA[18,pp 22], Vries[33,pp 11] & Doi[66]* :

1 Synchronisation Header :	24 Channel Bits	<i>Watkinson[58,pp 28]</i>
3 Merging Bits :	3	.
1 Control Byte :	14	.
3 Merging Bits :	3	.
	44 channel bits	
32 Data Bytes :	14	.
32 Merging sets :	3	.
	32*(14+3) = 544	

A 24-bit synchronisation is added to enable recovery from loss of clocking due to channel errors *Golomb[68], Gilbert[69] & Ullman[70,pp 95], Levenshtein[71,pp 707]*.

For a CDROM the medium surface consists of a sequence of depressions or pits and mounds or lands *ECMA[18,pp 22]*. Each channel frame of 588 bits, each of which is recorded along the physical track of the CDROM, i.e. along the spiral from hub to rim which is traversed by the optical system. In the recording process a channel 'one' is represented by a transformation between pit and land, a 'zero' is signified by no change *Davies[28,pp 35]*. The account of the encoding process has covered the stages by which raw data input to an encoder can be transformed to channel code by coding, scrambling and modulation. The decoding process is considered in the next Chapter.

CHAPTER SIX

The Decoding Processes Of The CDROM

6.1 Introduction

This Chapter is concerned with decoding the data which is held on the CDROM in the form of the EFM recording code. In the absence of errors the decoding procedure will be the inverse of the operations of the encoding: i.e. channel decoding, CIRC decoding and de-interleaving; descrambling and double RSC decoding. In addition, the decoding algorithm must detect and correct recording errors. Hence a component of the decoding procedure is the identification and resolution of error syndromes.

All optical data is recorded directly onto the medium at the mastering stage. Hence recording channel errors which occur are read errors and occur when a channel bit is missed or mis-interpreted. It is the function of the decoding strategy to correct all such errors.

Reed Solomon Codes have been designed for the correction of burst errors since they deal with symbols (e.g. bytes) of data rather than individual bits, as discussed in **Chapter Three** *Forney*[37], *Preparata*[72] & *Doi*[42,pp 148]. Nevertheless Reed Solomon Codes have a limited capacity for dealing with low levels of random errors or with both burst and random errors together. However the performance falls below those of codes designed for random errors alone *Helgerson*[73,pp 406].

The fourteen bit modulation codes offer a potential 2^{14} distinct codewords. However, due to the RLL constraints only 267 bit patterns produce valid codewords. Fourteen bit codewords are mapped to the corresponding data byte from a look-up table. Those patterns falling outside the table may either be interpreted as the nearest legal 14 bit pattern (interpolation), or treated as a codeword violation. This will produce a data byte in error, which is the starting point for the CDROM error control

strategy. The 33 byte F3 Frame is obtained by demodulation. The first byte of the F3 Frame is the control byte, this is used by the drive hardware and then discarded, so producing a 32 byte Frame.

6.2 Error Detection and Correction Using CIRC

The decoding strategies must resolve the full diet of error syndromes *Johnson[74]*. Hence decoding algorithms are far more complex and more time consuming than their encoding counterparts, Special LSI (Large Scale Integration) Circuits have been designed to accommodate this additional complexity *Arai[75,pp 356]*.

All three interleaving stages of the CIRC encoding process were discussed and illustrated in **Chapter Five**. Clearly the de-interleaving must always be the exact reverse of the deinterleaving strategies. Various strategies concerning CIRC have been investigated *Ko[76] & Vries[62]*. In the following work the C1 decoder locates and corrects two errors in each frame, the C2 uses the C1 flags to correct a maximum of four erasures.

6.2.1 C1 Decoding

The 32-byte frames produced after demodulation and control byte removal are fed into the CIRC Decoder. The first stage is to reverse Interleaving Strategy III such that the frames acted upon by the original C1 Encoder are recovered. The C1 decoder is intended to both detect burst errors and correct random errors *Vries[62,pp 184]*. The purpose of detecting burst errors is to provide flags for the C2 decoder *Watkinson[61,pp 82]*.

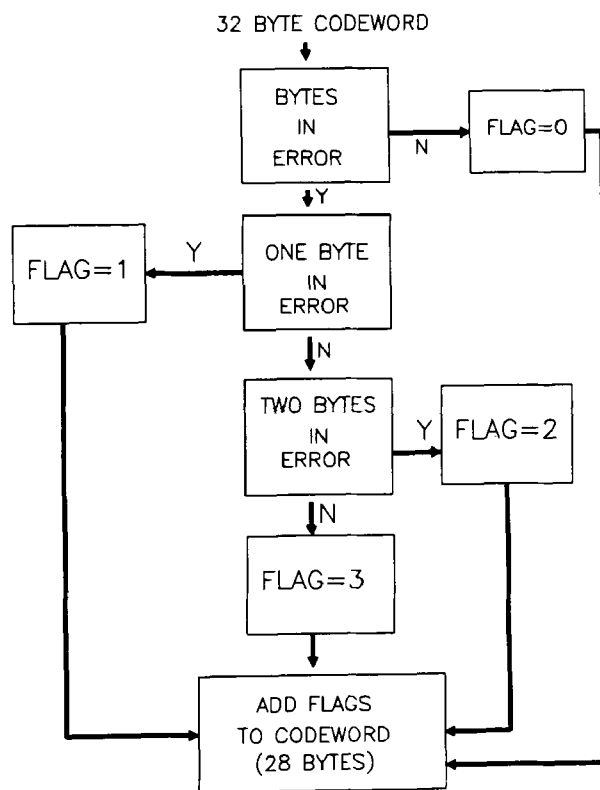
Each 32-byte C1 Frame contains 4 parity bytes and thus has the capacity to correct a maximum of two erroneous bytes per frame *Hoevel[32,pp 168] & Ko[77]*. The parity bytes were originally obtained by the use of equations from **equation 6.1** (see **Section 5.3.4**). For Reed Solomon codes the decoding matrix is identical to the encoding matrix. Hence the C1 decoding is given by:

$$H_p * V_p = S \quad (6.1)$$

where H_p and V_p are specified in the C1 Encoder of **Chapter Five** and right hand side is now the error syndrome. Clearly with no error in the data the syndrome S equals zero, consistent with **Section 5.3.4**.

Each syndrome is the result of the row multiplication matrix and vector, and subsequent addition of rows. The C1 Error Correction algorithm is illustrated in **Figure 6.1 Arai[75,pp 355], Vries[62,pp 184]**.

Figure 6.1 : Flow Diagram Of C1 Error Correction



Using Syndromes For Error Correction and Byte Location

The syndromes are calculated in the same manner as for the three bit symbol case which was discussed in **Chapter Three**, however with far greater complexity. If the syndromes are non-zero then the frame is erroneous. The syndromes are exploited so as to find the byte(s) in error. With $S = (S_0, S_1, S_2, S_3)$. The four error syndromes (S_0, S_1, S_2, S_3) are calculated as in **Chapter Three**, noting that here there are more symbols in each codeword.

If all four syndromes are zero then no bytes are in error. If this is not so then the Frame is checked for a single byte error. The necessary condition for this is $S_1/S_0 = S_2/S_1 = S_3/S_2 = \alpha^k$, where k is the number of the byte in error. Note that k can be at most 32, any value returned greater than this maximum is incorrect. If the above condition is not valid then the Frame is checked for two byte errors.

Two Byte Error Location

If one byte is not in error then there are two or more bytes in error in the codeword. The use of syndromes in the location of a two byte error is possible by using them in an error location equation *Peterson*[78,pp 114].

Following *Ko*[77]:

$$S_i = \alpha^{ai}E_a + \alpha^{bi}E_b \quad (6.2)$$

Where the two byte errors are E_a and E_b at byte locations a and b respectively then $i = 0, 1, 2$ and 3 .

It has been shown that the error locations satisfy the following error location quadratic *Patel*[79] :

$$(S_1^2 + S_0S_2)x^2 + (S_1S_2 + S_0S_3)x + (S_2^2 + S_1S_3) = 0 \quad (6.3)$$

Patel showed that the error locations α^a and α^b are the roots of the quadratic *Patel*[79]. *Heise* has shown that **Equation 6.3** can be solved by substitution *Heise*[80]:

$$x = \beta y \quad (6.4)$$

where :

$$\beta = \frac{S1S2 + S0S3}{S1^2 + S0S2} \quad (6.5)$$

by applying this **Equation 6.3** becomes:

$$y^2 + y + \gamma = 0 \quad (6.6)$$

where:

$$\gamma = (S2^2 + S1S3) \left\{ \frac{S1S2 + S0S3}{S1^2 + S0S2} \right\}^2 \quad (6.7)$$

that is:

$$\gamma = \frac{S2^2 + S1S3}{S1^2 + S0S2} * \beta \quad (6.8)$$

Equation 6.6 is a quadratic based upon the constant γ and this can be solved by using a look up table. The two roots of **equation 6.6** are $f_a(\gamma)$ and $f_b(\gamma)$ and thus the byte locations are given by:

$$\alpha^{-a} = f_a(\gamma) \quad (6.9)$$

and

$$\alpha^{-b} = f_b(\gamma) \quad (6.10)$$

A table lookup method is used in order to determine the byte locations from the values generated by the above equations. The bytes in error can be

corrected using the algorithms illustrated in **Section 3.2.6**.

If the location of the bytes which are in error exceeds 32, they will lie outside the frame boundary and thus are wrong. If the positions could not be found the frame is deemed to contain more than two bytes in error. So as to facilitate greater error correction power in the C2 Decoder each byte of every frame is flagged *Hoeve[32,pp 169]*. This flag indicates the error status of the Frame from which the byte originated. This strategy is clearly efficient for burst errors but not random *Ko[77,pp 17]*. There are four types of flags *Vries[33,pp 8]*:

Flag A : no errors	Flag B : 1 error
Flag C : 2 errors	Flag D : >2 errors (uncorrectable)

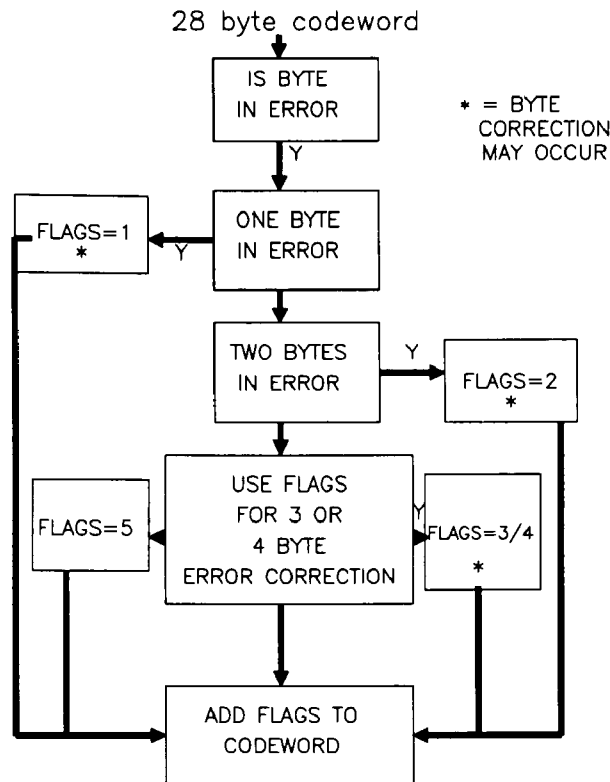
The purpose of the flag will be discussed in the description of the C2 Decoder. Now that the parity bytes have fulfilled their function they are removed to produce a flagged 28-byte frame. At present CDROM error correction does not differentiate between flags B,C and D. Valuated flagging does enable further complexity to be incorporated in the decoding procedures to improve correction.

6.2.2 C2 Decoding

The data is manipulated so as to reverse the effect of Interleaving Strategy II. Any faulty bytes which may exist are dispersed amongst the 98 28-byte Frames. The purpose of the C2 decoder is to correct burst errors and those random errors that the C1 could not *Vries[62,pp 184]*.

Each 28-byte C2 Frame has 4 bytes of parity, so that two erroneous bytes can be again corrected per frame. Unlike the 32-byte C1 Frame, the C2 Frame has a flag associated with each byte. The identity of bytes which have been flagged as possible errors are known and 4 bytes per frame can be corrected. This is known as erasure correction *Peek[30,pp 11]*. An erasure is a byte in a known position where the integrity of that byte is in question. The C2 Error Correction is carried out as illustrated in **Figure 6.2** *Arai[75,pp 354-5] & Vries[62,pp 185]*.

Figure 6.2 : Flow Diagram Of C2 Error Correction



The parity bytes and their associated syndromes are obtained from **Equation 6.11**.

$$H_Q * V_Q = S \quad (6.11)$$

Where H_Q and V_Q are as specified in **Section 5.3.2** and again $S=0$ if no bytes are in error.

Using Syndromes in Conjunction With Flags

The four syndromes S_0 - S_3 , can be used to find any two bytes in error in the same way as seen in the C1 Decoding. If the error bytes fall outside the 28-byte Frame boundary then more than two bytes are in error within that frame.

If more than two bytes are in error then the syndromes cannot be used to both locate and correct the bytes in error. Instead the flagged bytes are considered. Each flag illustrates that the byte has been deinterleaved from a C1 Frame where error correction has been attempted. The bytes originating from such Frames are all considered to be potentially in error. If more than four bytes have been flagged as potentially in error then no error correction is applied for fear of possible mis-correction. Otherwise these four bytes are assumed to be in error and are corrected.

The Correction Of Four Byte Errors

The correction of up to four flagged bytes follows the approach of *Ko*[77].

Let there be four suspect bytes written A,B,C and D respectively. These are processed to produce the equivalent Galois Field elements α^A , α^B , α^C & α^D . The corresponding errors are E^A , E^B , E^C & E^D . As before the syndromes are denoted by S_0 - S_3 , which are the associated Galois Field elements. Three bytes in error can also be corrected by using the following methodology.

By using **Equation 6.12** which illustrates the relationships between the syndromes and associated errors, the errors may be found.

Following *Ko*[77,pp 19]:

$$\begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \alpha^A & \alpha^B & \alpha^C & \alpha^D \\ \alpha^{2A} & \alpha^{2B} & \alpha^{2C} & \alpha^{2D} \\ \alpha^{3A} & \alpha^{3B} & \alpha^{3C} & \alpha^{3D} \end{bmatrix} * \begin{bmatrix} E^A \\ E^B \\ E^C \\ E^D \end{bmatrix} \quad (6.12)$$

By Gaussian Elimination **equation 6.13** is produced:

$$\begin{bmatrix} Q_0 \\ Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & (\alpha^A + \alpha^A) & (\alpha^C + \alpha^A) & (\alpha^D + \alpha^A) \\ 0 & 0 & (\alpha^C + \alpha^A)(\alpha^C + \alpha^B) & (\alpha^D + \alpha^A)(\alpha^D + \alpha^B) \\ 0 & 0 & 0 & (\alpha^D + \alpha^A)(\alpha^D + \alpha^B)(\alpha^D + \alpha^C) \end{bmatrix} \begin{bmatrix} E_A \\ E_B \\ E_C \\ E_D \end{bmatrix} \quad (6.13)$$

where:

$$\begin{bmatrix} Q_0 \\ Q_1 \\ Q_2 \\ Q_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \alpha^C & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \alpha^B & 1 & 0 \\ 0 & 0 & \alpha^B & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ \alpha^A & 1 & 0 & 0 \\ 0 & \alpha^A & 1 & 0 \\ 0 & 0 & \alpha^A & 1 \end{bmatrix} \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \quad (6.14)$$

The errors correction symbols E^A , E^B , E^C and E^D , may be produced by the back substitution of **equation 6.13**:

$$E_D = \frac{Q_3}{(\alpha^D + \alpha^A)(\alpha^D + \alpha^B)(\alpha^D + \alpha^C)} \quad (6.15)$$

$$E_C = \frac{Q_2 + \frac{Q_3}{(\alpha^D + \alpha^C)}}{(\alpha^C + \alpha^A)(\alpha^C + \alpha^B)} \quad (6.16)$$

$$E_B = \frac{Q_1 + \frac{Q_2}{(\alpha^C + \alpha^B)} + \frac{Q_3}{(\alpha^D + \alpha^B)(\alpha^B + \alpha^C)}}{(\alpha^A + \alpha^B)} \quad (6.17)$$

$$E_A = Q_0 + E_B + E_C + E_D \quad (6.18)$$

Using this method the four bytes in error may be corrected, however there is a chance of miscorrection *McEliece[48,pp 701-703]*. The processing time taken to implement two error location and correction, or four error correction in the CDROM is less than 4 μ m *Ko[77,pp 24]*.

The reversal of Interleaving Strategy I will further disperse errors across the Frames of the Sector, thus aiding error correction and error interpolation.

This concludes Compact Disc error correction. The stages which remain are unscrambling and addressing the logical sector. All 24-byte F1 frames are flagged in the same manner as discussed previously. In the Compact Disc these flags are used for byte interpolation before digital to analogue conversion *Hoeve[32,pp 172]*, *Goedhart[16,pp 174]*. In the CD-ROM however these flags are passed to the sector decoding codes which employ two Reed Solomon Codes. Instead of using complex interleaving strategies these codes are treated as product codes. These are discussed in the next section.

6.3 Sector Decoding

The configuration of the sector is shown in **Figure 5.2** and the encoding processes in **Figure 5.3**. A number of decoding processes are carried out, most are the merely the inverse of the coding process.

The synchronisation field (bytes 0-11) is used to identify the beginning of a sector to the CD hardware. The synchronisation field also prompts the scrambling register to be reset to the initial settings. Hence the scrambler is reset at the beginning of each new block. Before the bytes were fed into the CIRC encoder the consecutive bytes of each frame were interchanged. This process is reversed before unscrambling of the data. The unscrambling of the data is a repetition of the scrambling procedure and not the inverse; the same circuit, register and polynomial are used. Since modulo two arithmetic is employed.

The Header Field holds addressing information for the sector,

establishing for example that the CD is addressing to the correct sector. The Header Field also holds details of the sector type, whether CDRom, CD or empty of data.

6.3.1 Decoding Using Q-Parity

There remain 2340 bytes which are read into two 43 by 26 matrices in the same manner as described in **Section 5.2.6** and illustrated in **Figures 5.4** and **5.7?**. The remaining 104 bytes are known to be the parity bytes and are read into two associated 2 by 26 parity matrices. Each matrix's 26 byte axis parallel to the diagonal and associated two parity bytes are read from the matrix.

$$2352 - 12 = 2340$$

$$2340 = 2236 \text{ (data)} + 104 \text{ (parity)}$$

$$2 \times 43 \times 26 = 2236 \text{ bytes}$$

What follows is another standard Reed Solomon Decoding strategy. The parity bytes and hence the syndromes are generated using **Equation 6.19**:

$$H_Q * V_Q = S \quad (6.19)$$

Any byte in error which is left uncorrected by the C2 decoder will be dispersed by the reversal of Interleaving Strategy I, again spreading erroneous bytes over a number of codewords.

The two syndromes can be used to locate and correct one erroneous byte per 26 byte codeword. This is done in the same manner as discussed in **Section 6.2.1**. Again C2 flags are used to correct a maximum of two bytes in error.

Once the two parity bytes have fulfilled their usefulness they are discarded. Again frames which held erroneous bytes are flagged non-zero for use by the P-Parity check.

6.3.2 Decoding Using The P-Parity

The identical 43 x 24 matrices are again used for this error correction strategy. **Equation 6.20** is used to generate error Syndromes.

$$H_P * V_P = S \quad (6.20)$$

H_p and V_p are specified in the Encoding Process. The code can be employed in the same manner as seen previously. Two parity bytes are present in the codeword and one byte may be corrected, whereas the flags from the Q-Parity can be used to correct two. Since the bytes are sampled from the matrix along a different axis from that of the Q-Parity, greater error correction is possible. This is an application of product codes. In the same way that interleaving disperses errors over a number of codewords, so product codes aid error correction by calculating parity along different axis.

6.3.3 Using The EDC Check

The error correction of the CD-ROM also contains an error detection code as the final stage of the strategy.

A CRC code is used at the encoding stage to add parity bytes to the data. Since this parity is determined bit by bit from each symbol, any remaining errors will be detected at this point. The two checks associated with the CRC are calculated from the data in the manner as discussed in same way as **Section 5.2.4**. If there are any residual discrepancies from the encoded data the CRC check fails the whole Sector.

In this case the block of data must then be re-sought and accessed afresh from the disc. Data can only be resought up to a fixed maximum number of times from an area of the disc; continual failure of the EDC will constitute a hard error when the disc is considered corrupt.

CHAPTER SEVEN

Illustrating The Effects Of Errors Upon The CDROM By Use Of The Simulation Model

7.1 Introduction

A deterministic model of the full signal processing of the CDROM has been built. This enables a channel burst error to be precisely prescribed within a Sector. Both the bursts duration in modulated bits, and its starting position in the Sector can be varied. The ability of the CDROM to contend with this particular error may be analysed completely.

The simulation model covers a full CDROM Sector on the surface of the disc. Its operation is shown diagrammatically in **Figure 7.1**. Data is generated and encoded using the full sequence of CRC, Sector, CIRC and EFM encoding. The error normally in the form of a burst is imposed on the modulated data. The decoding is applied, with the associated error syndromes being generated for each of the steps. In this way the correction of the burst error, as well as its residual disposition across the Sector can be traced and output at each level of the decoding, from EFM to the final CRC.

At present the use of this model is the only method in which to investigate the effect of varying sizes of burst at varying positions along a Sector. No other method is thought to exist in Academia or in Industry.

By this method the main error types, both random and burst may be incorporated. The effect of these errors upon the code and the subsequent dispersal and correction may be investigated.

Due to the complexity of the decoding process it was decided that the software should identify the errors remaining after each stage of the process.

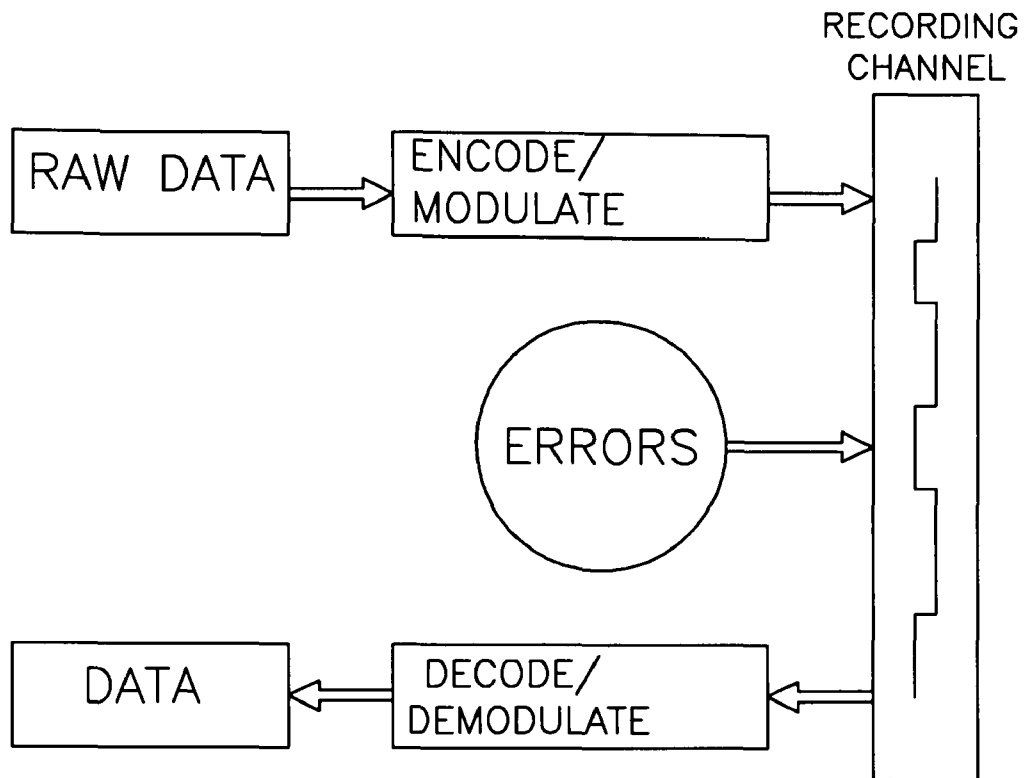


Figure 7.1 : Error Incorporation Into The Channel Data

7.2 The Effect Of A Burst Error On The Channel Data

The following examples illustrate situations where : CIRC corrects the burst errors, Sector correction corrects the errors, and where the error control strategy fails.

7.2.1 Burst Length Of 2900 Bits

As an example a burst error of length 2,900 bits is used, this is equivalent approximately 1.8 mm scratch upon the medium *Vries[33,pp 8]*. The effect upon the modulated bits is shown in **Figure 7.2**. This burst begins at bit 580 of Frame 10. The Figure displays all 98 Frames of the Sector by rows with the darker shading indicating the bits in error. With 588 modulated bits in each Frame, the burst extends almost to the end of Frame 15.

2900 Bit Burst

Figure 7.2
Effect Upon The Channel

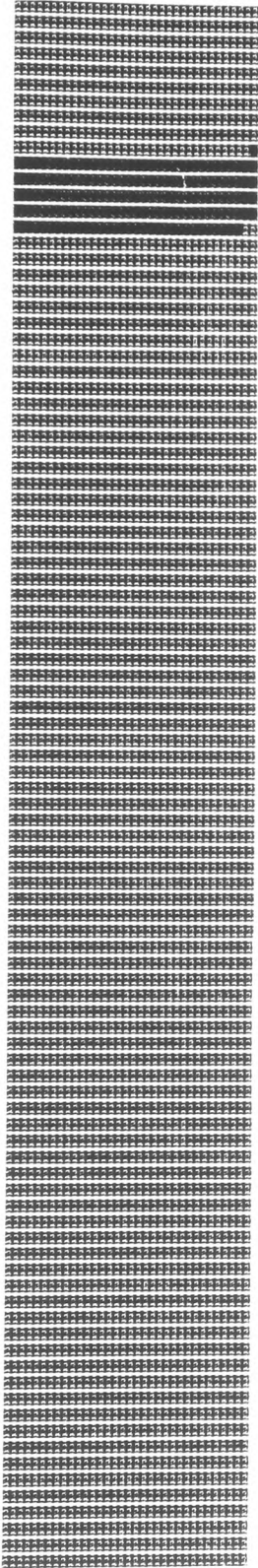


Figure 7.3
Synchronisation Loss

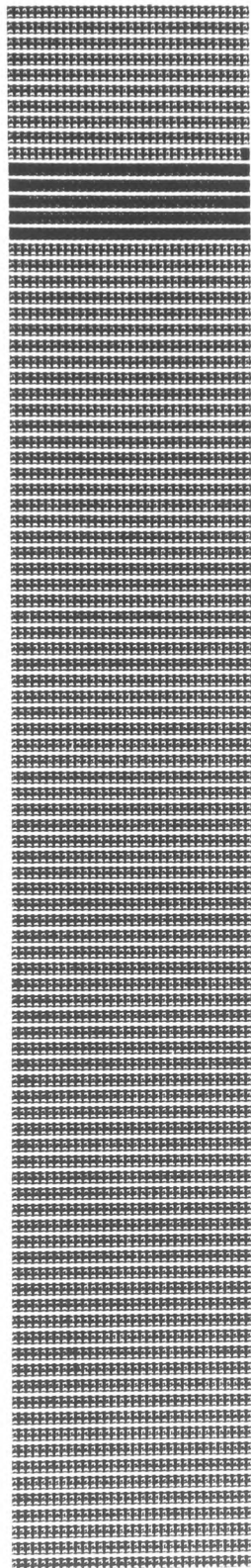
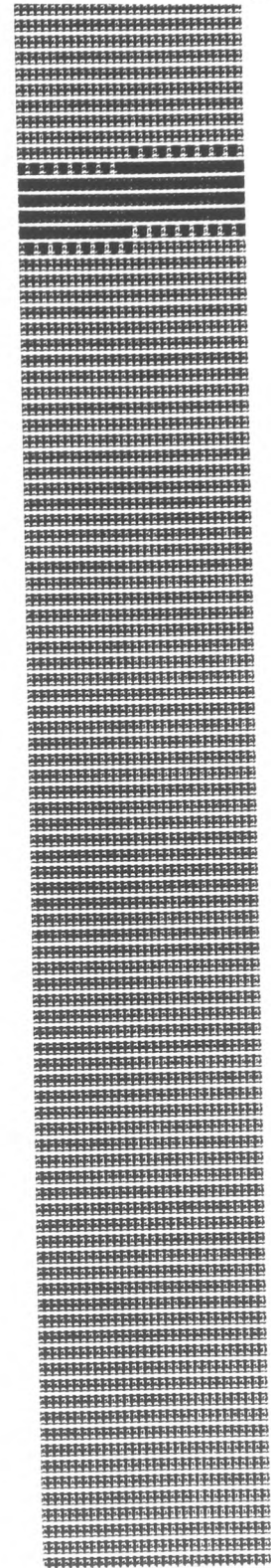


Figure 7.4
Deinterleave Strategy III



In **Figure 7.3** the effect of synchronisation loss is illustrated. Since the synchronisation bits of Frame 15 are lost, no part of this Frame can be read and all its bits are held to be in error. Thus there are six Frames in error at this stage.

Deinterleave Strategy III disperses bytes between adjacent Frames, each of 32 bytes, as shown in **Figure 7.4**. There are now seven Frames containing errors. The C1 decoding can resolve a maximum of two byte errors per Frame. Since all seven corrupted Frames have multiple errors no correction is possible, as shown in **Figure 7.5**. Note that the four C1 parity bytes have been removed giving 28 bytes per Frame.

Deinterleave Strategy II spreads the error bytes over most of the 98 Frames, as shown in **Figure 7.6**. By referring to the C1 error flags the C2 decoding can correct up to four suspect bytes per Frame, and in **Figure 7.7** it is shown that there is complete error correction. Again the C2 parity bytes have been removed giving 24 bytes per Frame.

Since all errors have been resolved by CIRC, further the error correction and dispersal schemes are discussed and illustrated with a larger burst. The CD audio error correction codes were able to resolve the bytes in error, the extra CDRM schemes were not utilised. In this case the errors would be corrected by either the CD or the CDRM.

7.2.2 Burst Length Of 3000 Bits

Here, a burst initiating at the same position but of larger magnitude is applied. The progression of the errors may be followed in the same manner as discussed for the 2,900 bit burst. A burst of 3000 bits is equivalent to a approximately 1.9 mm scratch upon the medium.

In **Figure 7.8** the effect of the error upon the medium is illustrated. Here, the increase in magnitude of 100 bits is shown to cause overlap into Frame 16. The effect of the subsequent synchronisation loss is illustrated in **Figure 7.9**. Here, error propagation occurs, to the end of Frame 16. Thus the

Figure 7.5
C1 Decoding

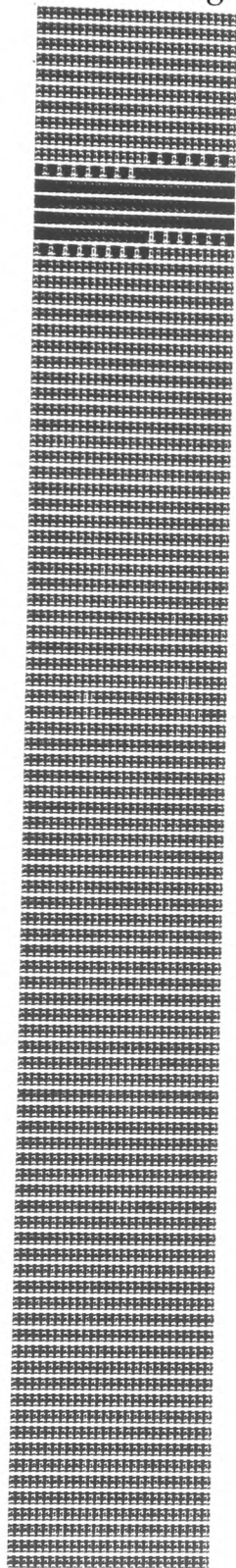


Figure 7.6
Deinterleave Strategy II

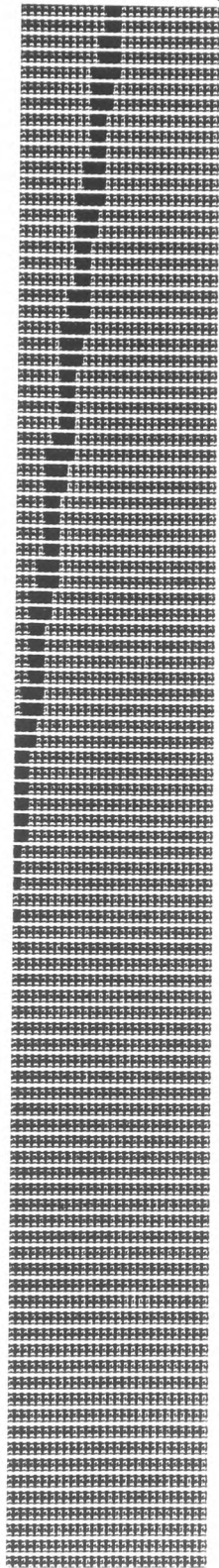


Figure 7.7
C2 Decoding



3000 Bit Burst

Figure 7.8
Effect Upon The Channel

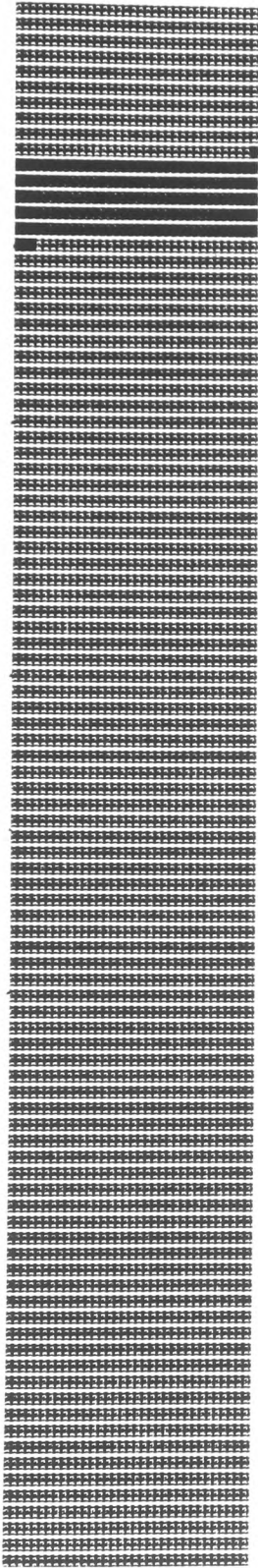


Figure 7.9
Synchronisation Loss

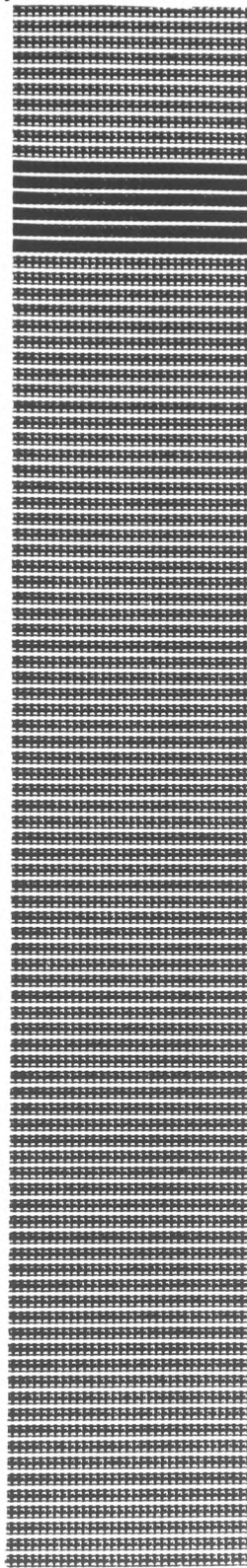


Figure 7.10
Deinterleave Strategy III

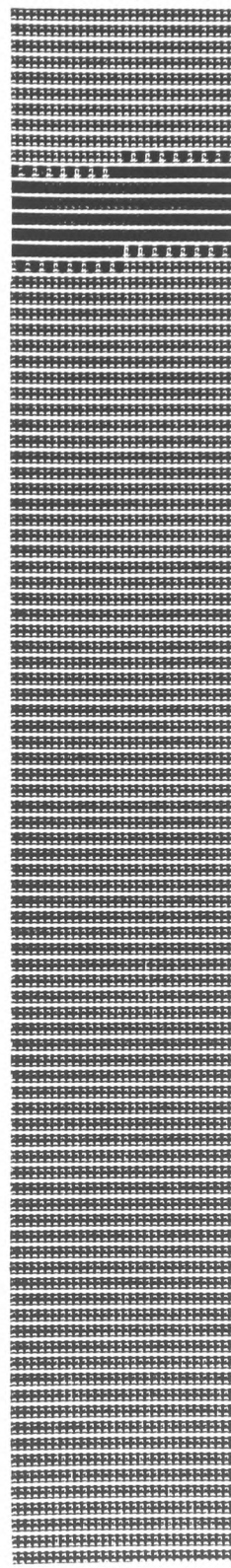


Figure 7.11
C1 Decoding

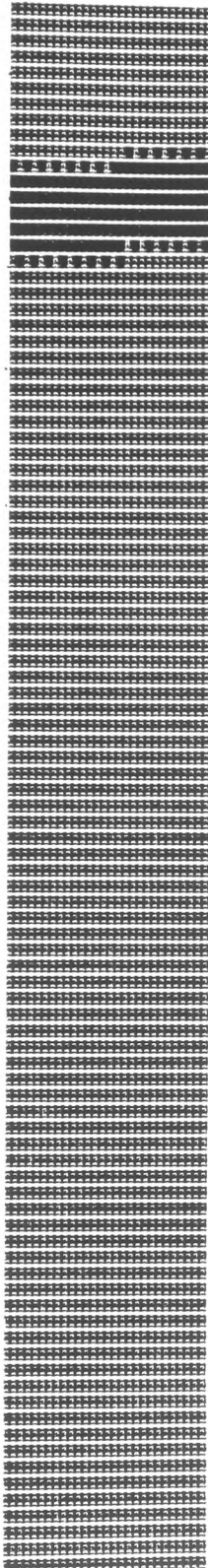


Figure 7.12
Deinterleave Strategy II

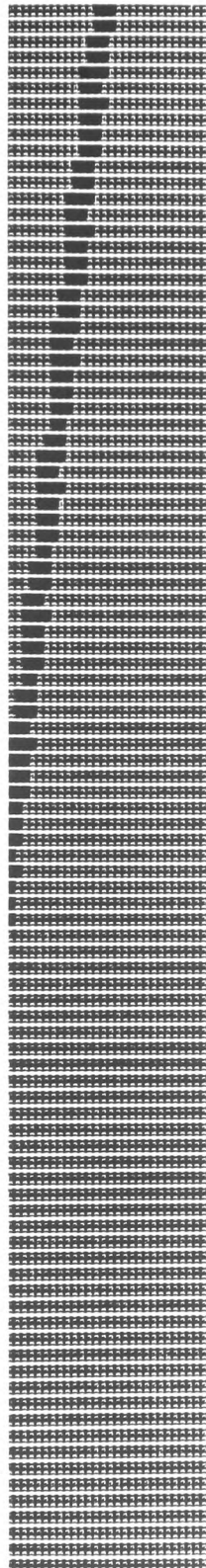


Figure 7.13
C2 Decoding



burst is perceived as approximately 3,500 bits. **Figures 7.10, 7.11 and 7.12** shown the effect of Deinterleave Strategy III, C1 Decoding and Deinterleave Strategy II, respectively.

In the 2900 bit case, the effect of C2 Decoding resolved all the errors. In this case errors persist. **Figure 7.13** illustrates the errors remaining after C2 Decoding. Here eight Frames are affected.

Deinterleave Strategy I now spreads the bytes over adjacent Frames as shown in **Figure 7.14**. This completes those deinterleaving and decoding procedures which are common with the Compact Disc.

Figure 7.15 shows the position of the errors after successive bytes of each Frame have been interchanged. **Figure 7.16** illustrates the position of the errors within the two matrices, as explained in **Chapter Five**, to enable product decoding by a pair of orthogonal Reed Solomon codes.

In Q decoding, where reference is made to the C2 flags, a maximum of two bytes in each of the 26 rows can be corrected. The results are shown in **Figure 7.17**, where only two errors remain. In P decoding, in association with Q flags, up to two bytes may be corrected in each of the 43 columns. As can be seen in **Figure 7.18**, no errors now remain and the integrity of the Sector data has been restored.

In this case CIRC has been unable to resolve the burst. The CD has no further error correction schemes, although interpolation is used. However in the CDROM the two extra levels of error correction have corrected the remaining errors.

7.2.3 Burst Length Of 3600 Bits

By applying bursts of greater magnitude at the same position it is possible to ascertain the size at which error correction fails. The progression of the errors arising from a 3600 bits can be observed in **Figures 7.19 - 7.29**. A burst of 3600 bits is equivalent to approximately 2.3 mm scratch upon the medium. In this example, the CDROM error correction scheme cannot cope with the quantity of errors. Errors persist after P correction. These are detected by the EDC which signals that the data is in error.

Figure 7.14
Deinterleave Strategy I

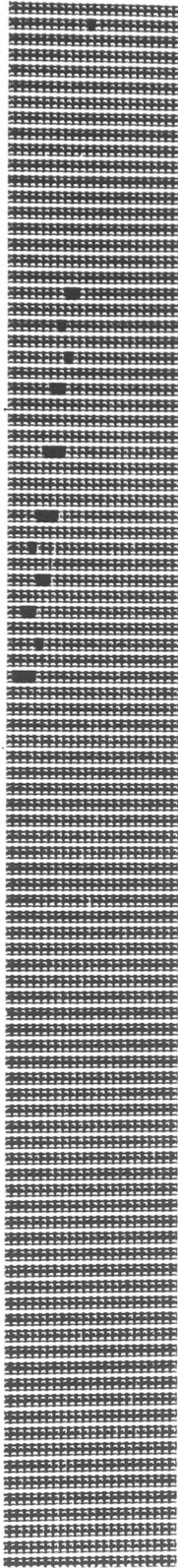


Figure 7.15
Byte Interchanging

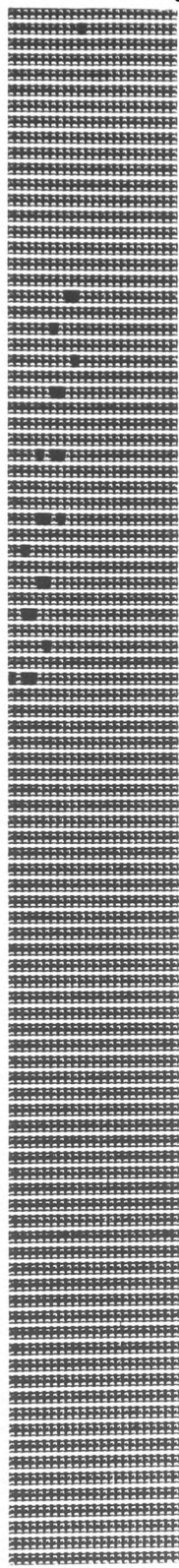


Figure 7.16
Position Of Errors In Product Matrices

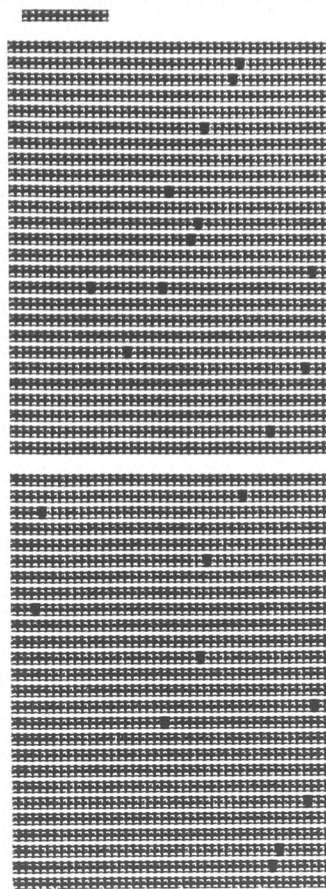


Figure 7.17
Q Decoding

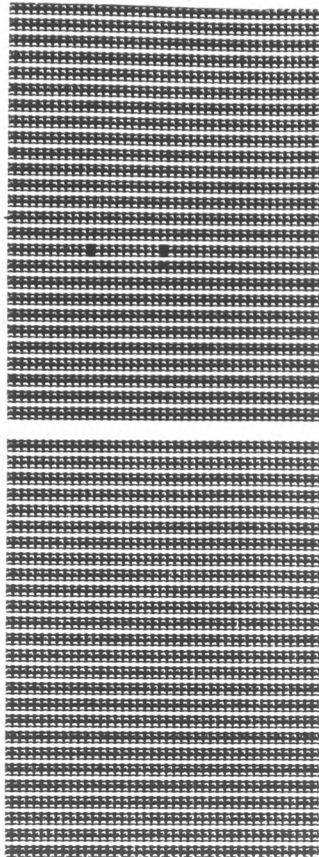
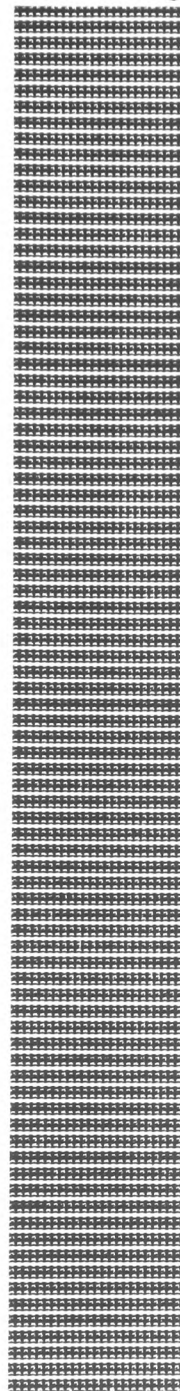


Figure 7.18
P Decoding On Raw Data



3600 Bit Burst

Figure 7.19
Effect Upon The Channel

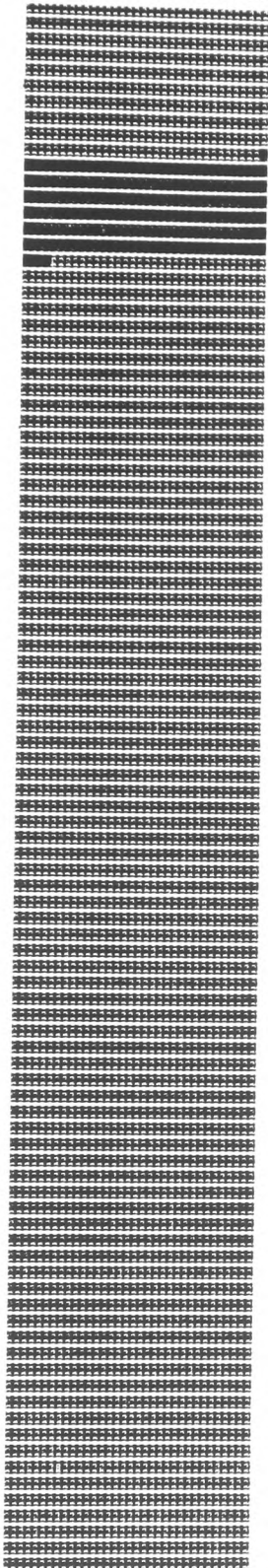


Figure 7.20
Synchronisation Loss

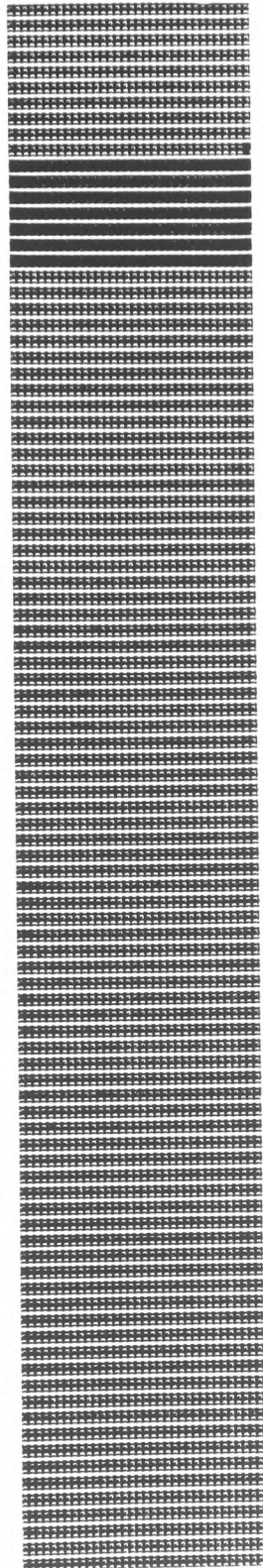


Figure 7.21
Deinterleave Strategy III

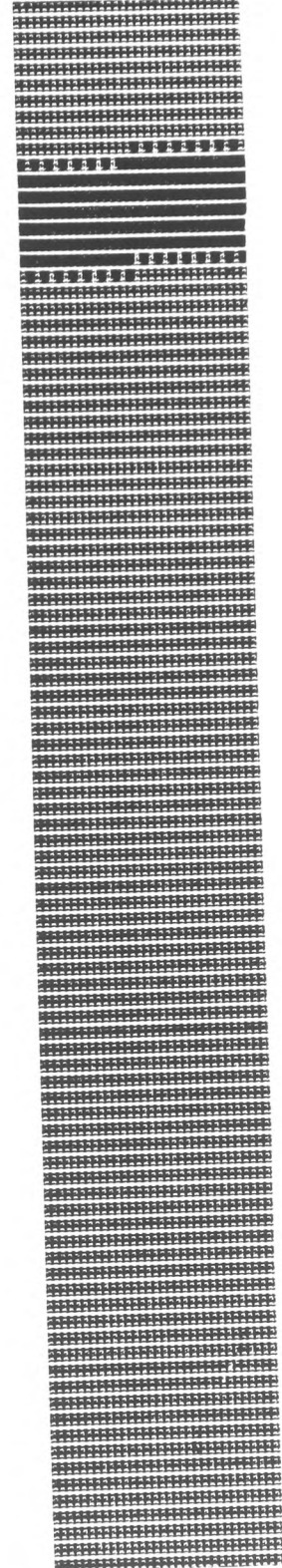


Figure 7.22
C1 Decoding

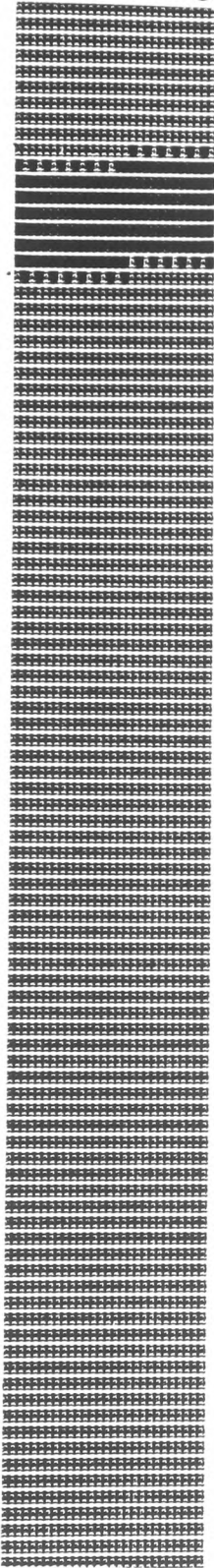


Figure 7.23
Deinterleave Strategy II



Figure 7.24
C2 Decoding

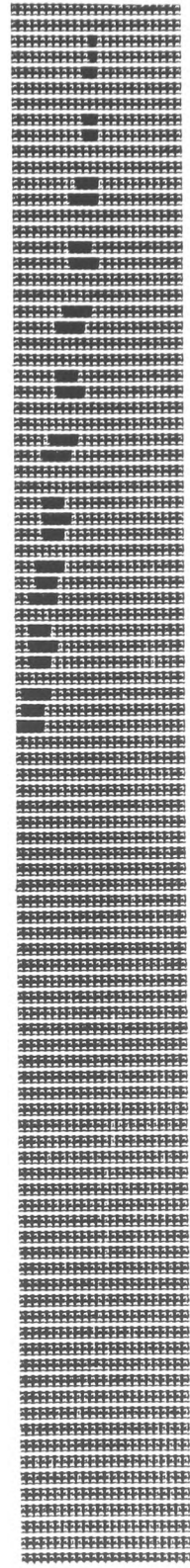


Figure 7.25
Deinterleave Strategy I

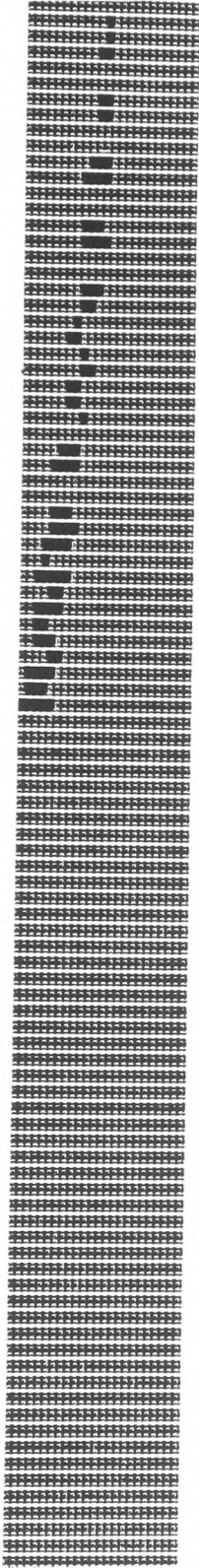


Figure 7.26
Byte Interchanging

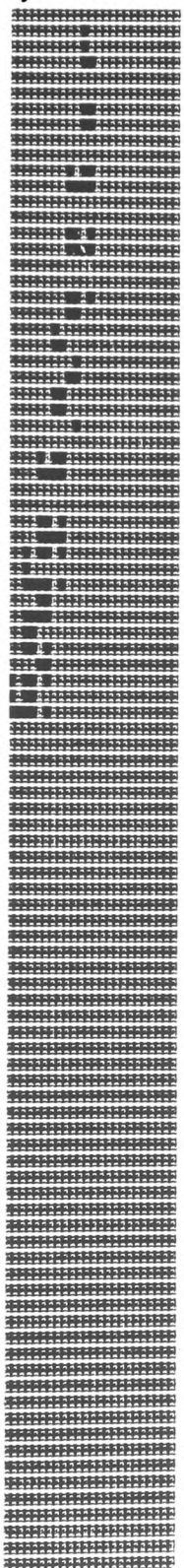


Figure 7.27
Position Of Errors In Product Matrices

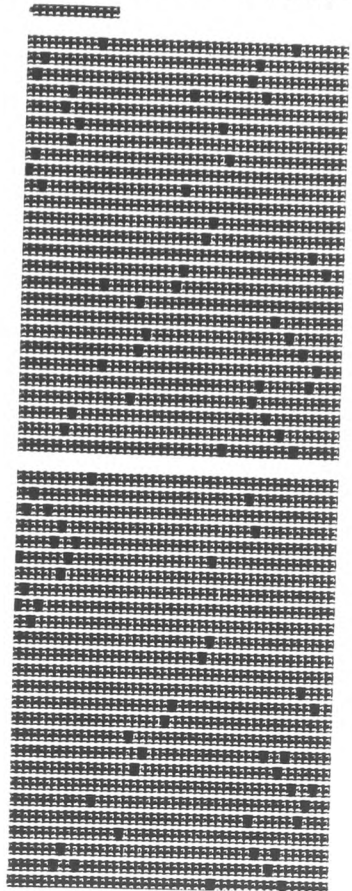


Figure 7.28
Q Decoding

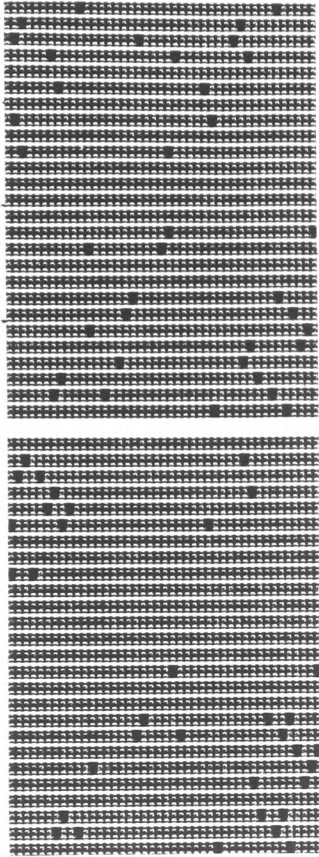
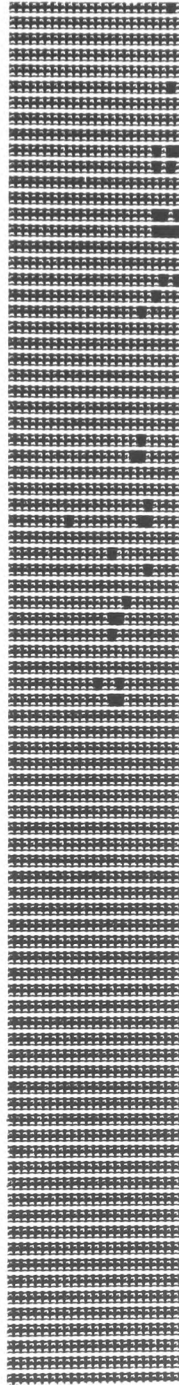


Figure 7.29
P Decoding On Raw Data



7.3 The Effect Of Random Errors On The Channel Data

The CDROM was not designed to resolve high proportions of random errors it is nevertheless appropriate to investigate the effect of such errors on the correction scheme. A random error is defined as an error affecting a single bit of the channel code regardless of the mechanism behind that error. If a bit error has no correlation with other bit errors it may be called a random error *Doi[42,pp 148]*. In the CDROM an asperity must be of size 0.8 mm so as to obstruct channel data from the lens, this is due the refractive index of the plastic. From this it is evident that any asperity affecting channel data will always cause a burst error. Surface debris small enough to cause a random error would not obstruct the read lens sufficiently to cause an error.

Random errors will only exist in low quantities, much less than those investigated here. Examples of both successful and unsuccessful error correction are given

7.3.1 Probability 0.0018 Error Rate

In a physical sector of data there are 98 Frames, each composed of 588 channel bits,i.e. 57624 bits. The expected number of bits in error per Sector will therefore be 104 with a standard deviation of 10.

In **Figure 7.30** the effect of the random errors upon the channel data is illustrated. Of considerable significance are Frames 18, 21, 54 and 74 where the synchronisation bits are effected. This can be seen to have striking effects upon the related Frames in **Figure 7.31**. Here, synchronisation errors have led to Frame errors, where all 588 bits are considered lost. In addition all random bit errors will cause a symbol or byte error. It is clear from this that occurring random errors will give rise to a mixture of symbol and burst (or Frame) errors.

Deinterleave Strategy III is illustrated in **Figure 7.32**, here again the errors are dispersed between adjacent Frames. This has little or no consequence for the symbol errors, however as usual the bursts are successfully dispersed. The effect of the C1 Decoding scheme is shown in

0.0018

Figure 7.30
Effect Upon the Channel

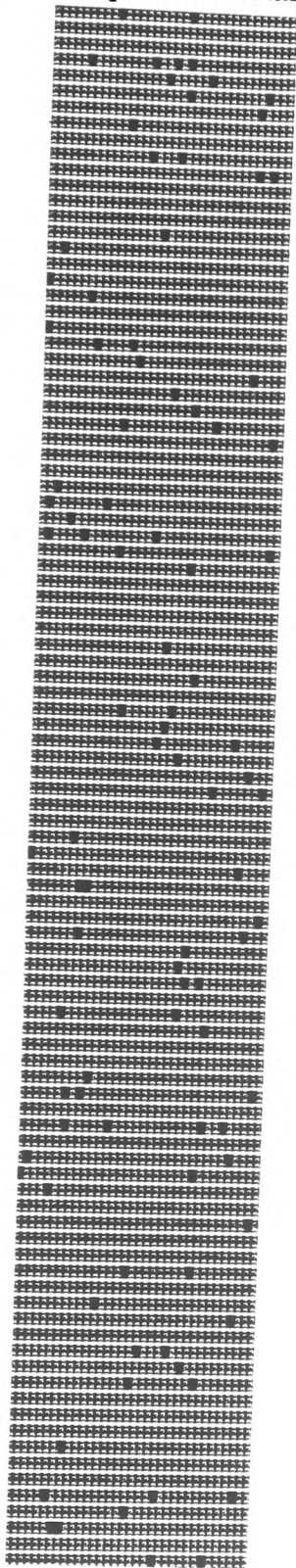


Figure 7.31
Synchronisation Loss

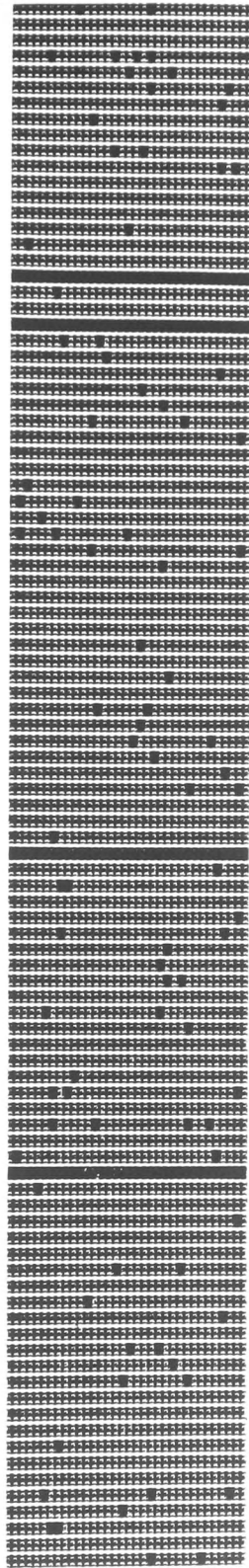


Figure 7.32
Deinterleave Strategy III

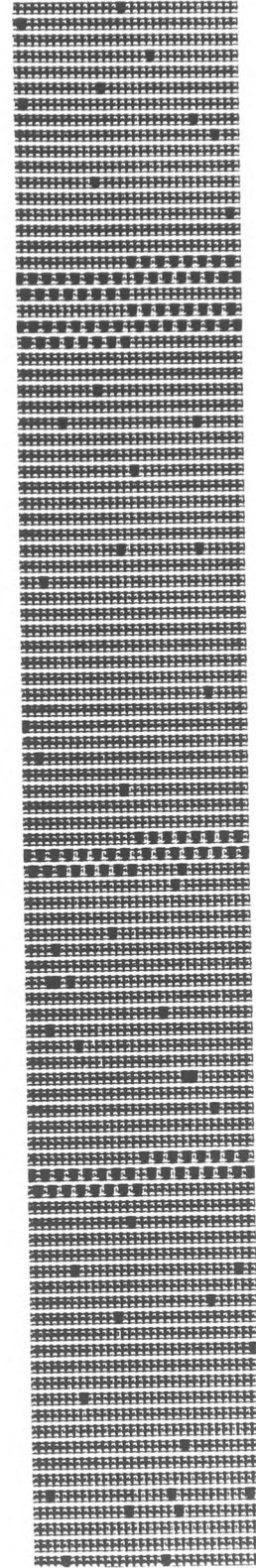


Figure 7.33
C1 Decoding

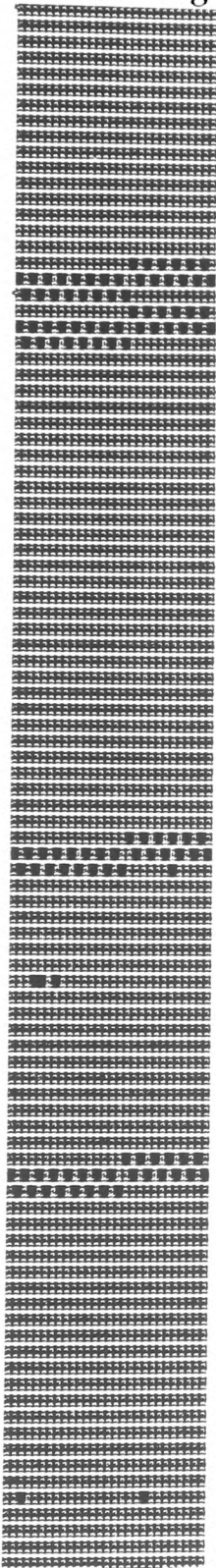


Figure 7.34
Deinterleave Strategy II

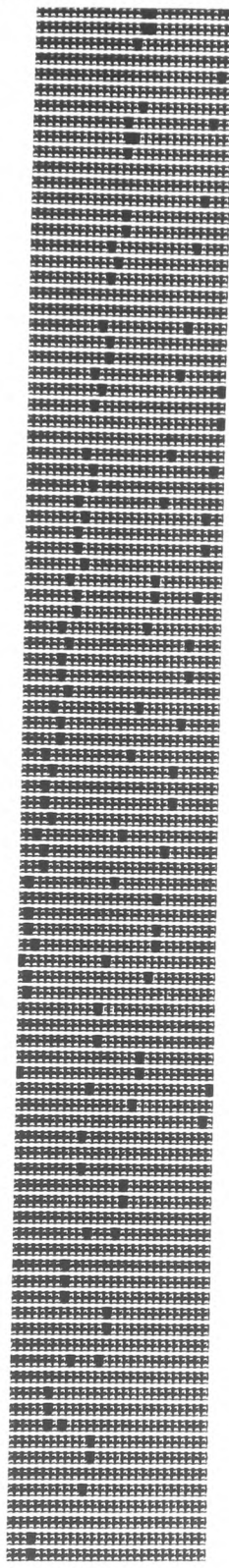


Figure 7.35
C2 Decoding

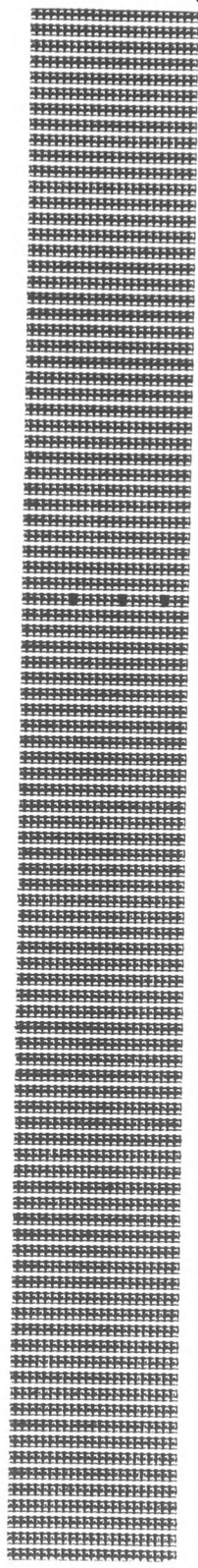


Figure 7.33. Here, the majority of the symbol errors have been corrected, although the burst effected Frames remain. After correction fifteen Frames remain in error.

The dispersal of the remaining errors by Deinterleave Strategy II is shown in **Figure 7.34**. Upon C2 error correction being applied to this data, one Frame remains in error. This is depicted in **Figure 7.35**. In **Figure 7.36** the remaining errors before Q correction scheme are shown. **Figure 7.37** illustrates that Q correction has corrected these errors.

7.3.2 Probability 0.0019 Error Rate

The expected number of bits in error per Sector is 110 with a standard deviation of 11. In **Figure 7.38** the effect of the random errors upon the channel data is illustrated. Frames 18, 21, 54, 74 and 81 have synchronisation bits which are effected. This can be seen to have the typical effect upon the related Frames in **Figure 7.39**. Deinterleave Strategy III is illustrated in **Figure 7.40**.

The result of the C1 Decoding scheme is shown in **Figure 7.41**. After correction seventeen Frames remain in error. The dispersal of the remaining errors by Deinterleave Strategy II is shown in **Figure 7.42**. When C2 error correction is applied to the data, thirteen Frames remain in error. This is depicted in **Figure 7.43**.

In **Figure 7.44** the distribution of errors in the product matrices is depicted. In **Figure 7.45** the effect of Q correction upon these matrices is shown, here nine of the matrix rows remain effected by errors. After P correction, **Figure 7.46** illustrates that only one matrix column remains effected by error, however this is enough to fail the data integrity check.

Figure 7.36
Position Of Errors In Product Matrices

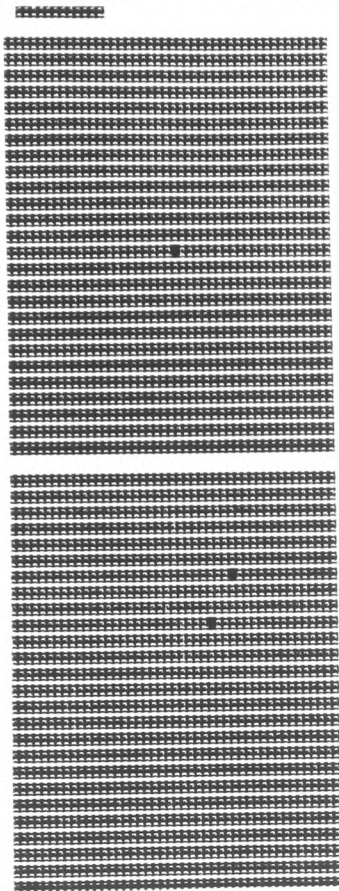


Figure 7.37
Q Decoding

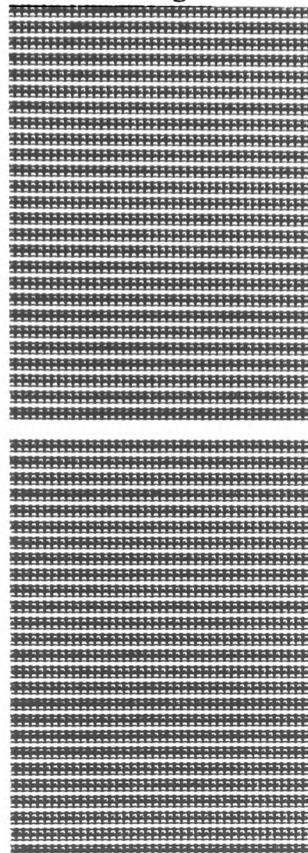


Figure 7.38
Effect Upon the Channel

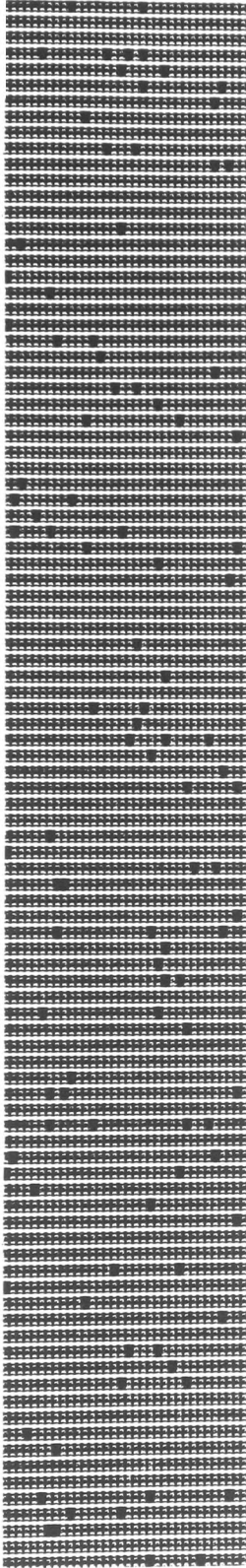


Figure 7.39
Synchronisation Loss

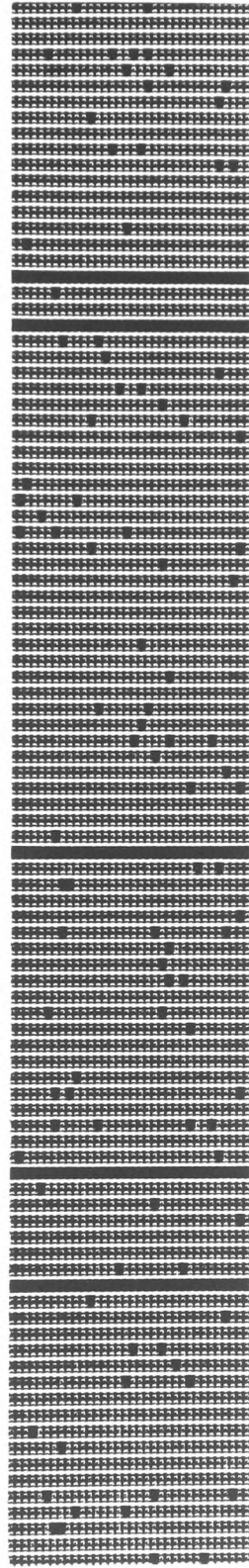


Figure 7.40
Deinterleave Strategy III

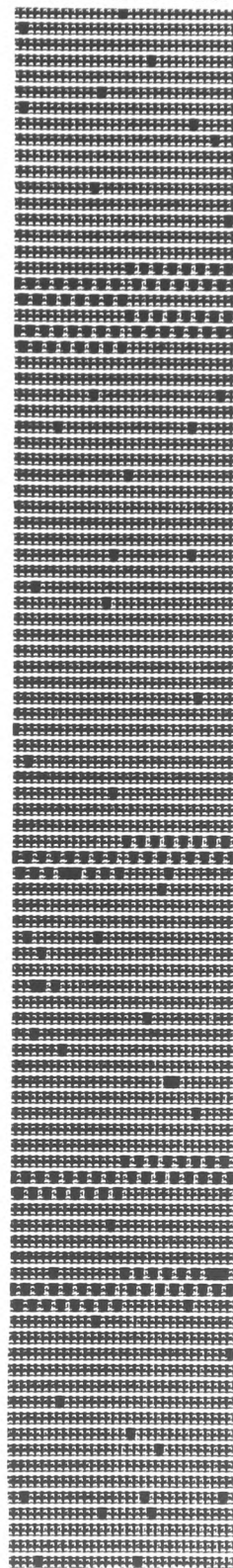


Figure 7.41
C1 Decoding

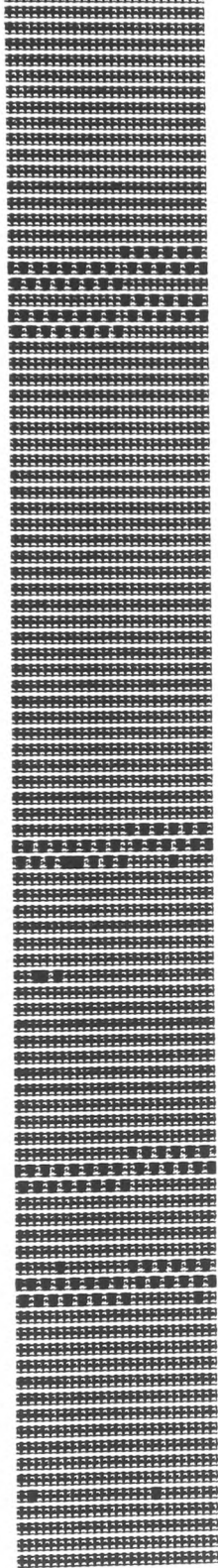


Figure 7.42
Deinterleave Strategy II

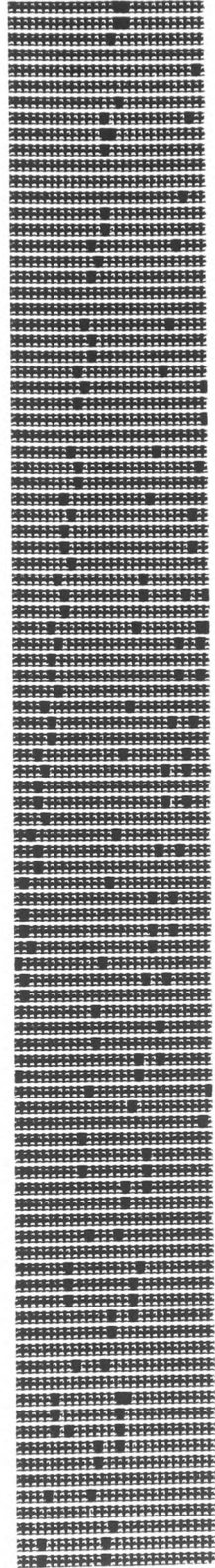


Figure 7.43
C2 Decoding

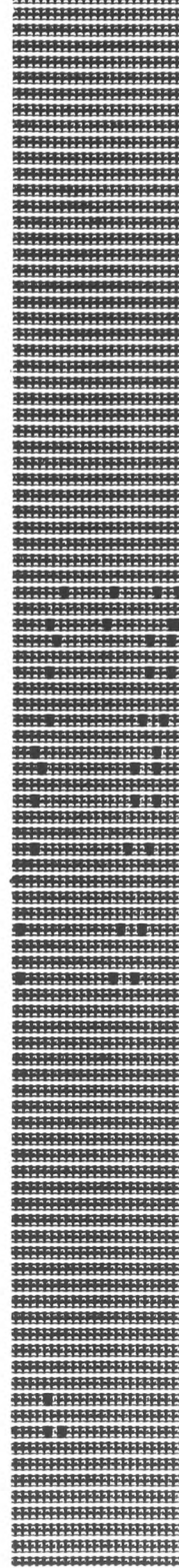


Figure 7.44
Position Of Errors In Product Matrices

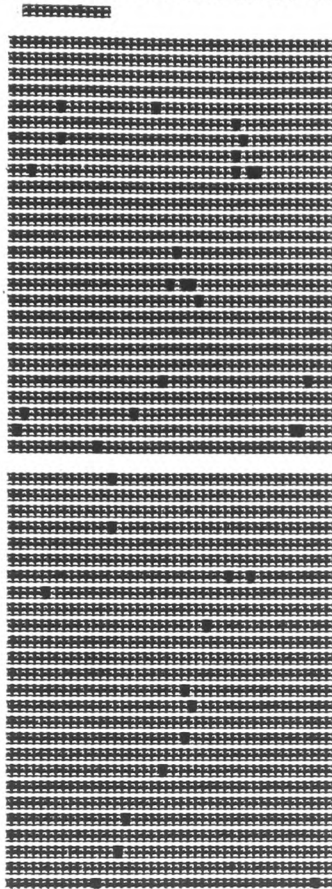


Figure 7.45
Q Decoding

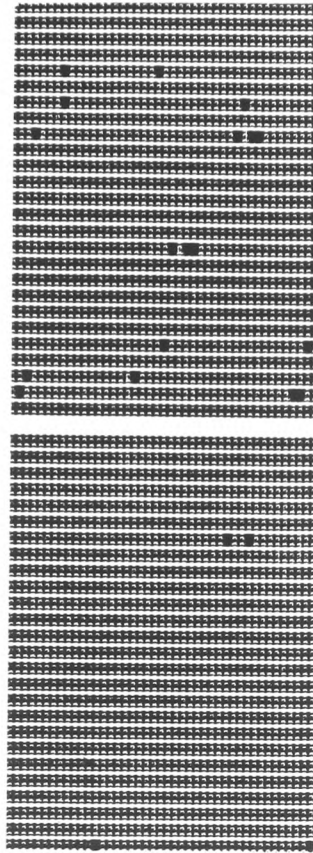


Figure 7.46
P Decoding On Raw Data



7.4 Conclusion

In the case of both random and burst errors it is shown how synchronisation loss can be of considerable significance. With burst errors, synchronisation loss leads to burst propagation to the end of that Frame. This can be of profound importance in situations where the error correction is at its limit.

When random bit errors are applied to channel data and synchronisation is lost the result is to produce both symbol and burst errors. This occurrence complicates the issue and lowers the tolerance of the system to random errors. Synchronisation loss is further discussed in **Chapter Eight**, where its effect upon the maximum error correction performance is illustrated.

CHAPTER EIGHT

Performance Measurement and Inferencing Using The Simulation Model

8.1 Outline

One of the objectives of this research was to identify the sources of errors effecting the CDROM and to assess their effects. For any Mass Storage Device, such as the CDROM, it is possible to obtain error counts at each stage of the decoding process. For the CDROM this will be after the C1 and C2 Decoding of the CIRC and the decoding of the Sector. Also the status of the final CRC check will be reported.

The model described in **Chapter Seven** can easily generate the equivalent statistics for simulated errors. The examples in **Chapter Seven** show that varying the position and length of a burst error will affect the error counts. A systematic programme of such simulations will generate a large set of such error counts. One can then use this information to infer the position and length of an error from empirical error counts. Error statistics for a DDS drive have been produced *Odaka[12], Woolley[81] & Woolley[82]*.

8.1.1 Data Production

To obtain a comprehensive set of error counts the simulation model was used for burst errors of length 100 bits to 10,000 bits (in steps of 100) and starting positions at 100 bit intervals in each Frame. Each simulation took approximately 1.5 minutes due principally to the file handling involved. Consequently the model was run on ten PC's using a supporting platform, so enabling execution. This still takes one week to complete. Here each PC is responsible for running bursts between specified sizes. There are 6 positions per Frame, 80 Frames to which each burst is applied and 100 bursts (10,000/100). There is therefore $(6 \times 80 \times 100)$ 48,000 jobs to carry out each taking 1.5 minutes.

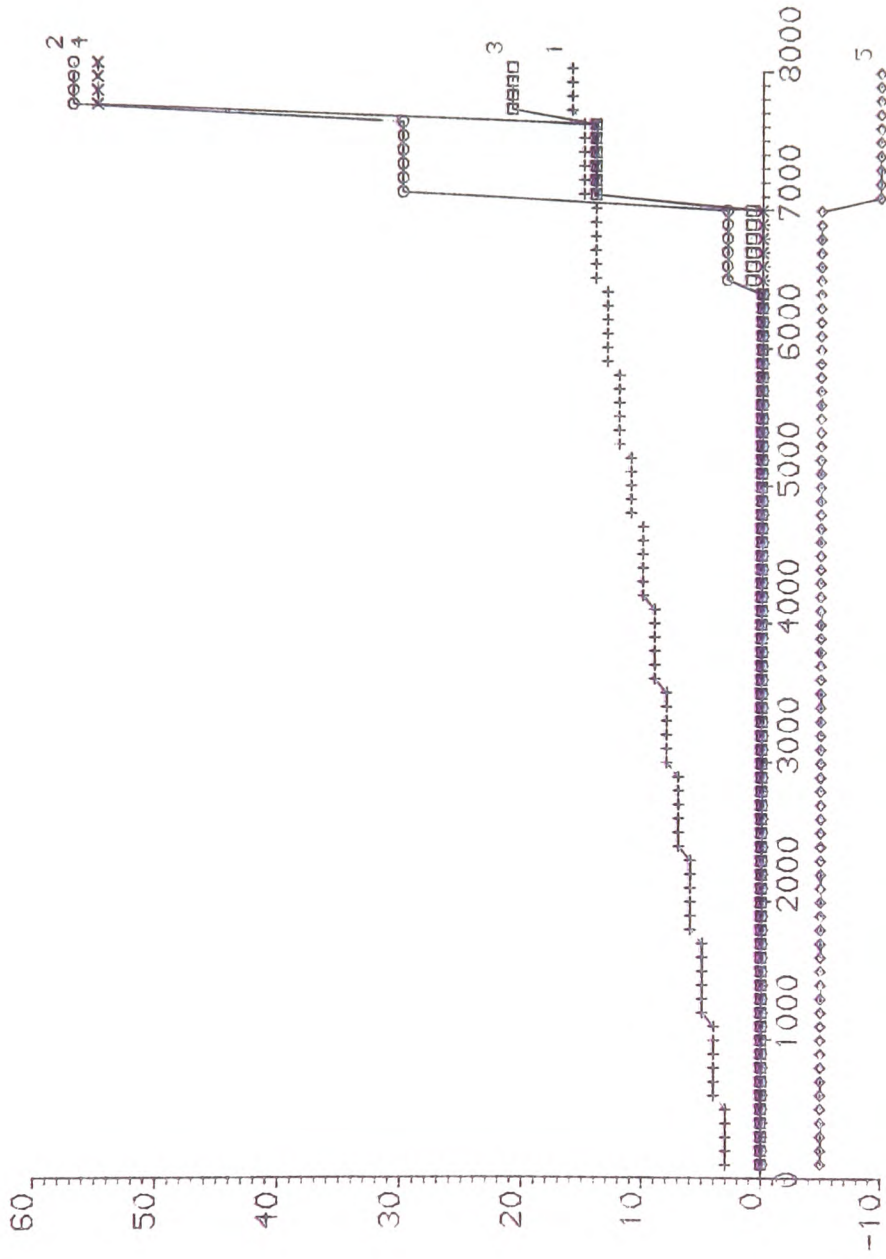
8.2 Burst Correction Performance Analysis of the CDROM

Recall that the statistics produced for each simulation run were: C1 and C2 error counts, the P and Q error counts and the status of the final CRC check. The C1 and C2 counts (which derive from Sector Frames) will lie in the range 0-98, whereas the P and Q counts which come from the product codes of the sector encoding have maximum values of 86 and 52 respectively. Each error count points to the number of Frames (codewords), which remain in error after one of the four decoders has been applied to the data. It has been found that the C1 count increases with burst length whereas the C2 count reflects the position of the burst within the Sector. The CRC status is simply PASS/FAIL.

Figure 8.1 shows the response of the CD-ROM to bursts of varying length with a common starting point within the Sector. It shows the effect of bursts of 100 - 8000 bits all beginning at bit one of Frame 40 and gives the C1, C2, P and Q counts and the CRC status in each case. As expected, the C1 count rises steadily with burst length and reflects the adjacent Frame deinterleaving strategy of Deinterleave III. As observed in **Chapter Seven** a burst affecting a number of Frames will be spread marginally between adjacent Frames. The burst length is therefore related to the number of Frames in error at C1 Decoding. The C2 count is largely static at zero, reflecting the fixed burst position. However, there is a threshold at a burst length of approximately 6,500 when the C2 count rises suddenly to 52. This reflects the Deinterleave II strategy which spreads the error bytes evenly across most of the Frames of the Sector. The C2 decoding fails (and the C2 count is augmented) at all 52 Frames on or near to the threshold value of the burst length. The P and Q counts follow the behaviour of C2. Both are zero below the threshold burst length: thereafter P rises sharply to 20 and Q to 55.

By contrast **Figure 8.2** shows the effect of a fixed burst length with different starting positions within the Sector. In this case the Figure shows the C1 and C2 counts and the CRC status for a burst of length 4000 bits. The results confirm that a burst of this size is readily correctable when it occurs near the middle of a Sector: ie between Frames 18 and 74. The results of

Figure 3.1 : Error Statistics Produced By Bursts At Frame 40

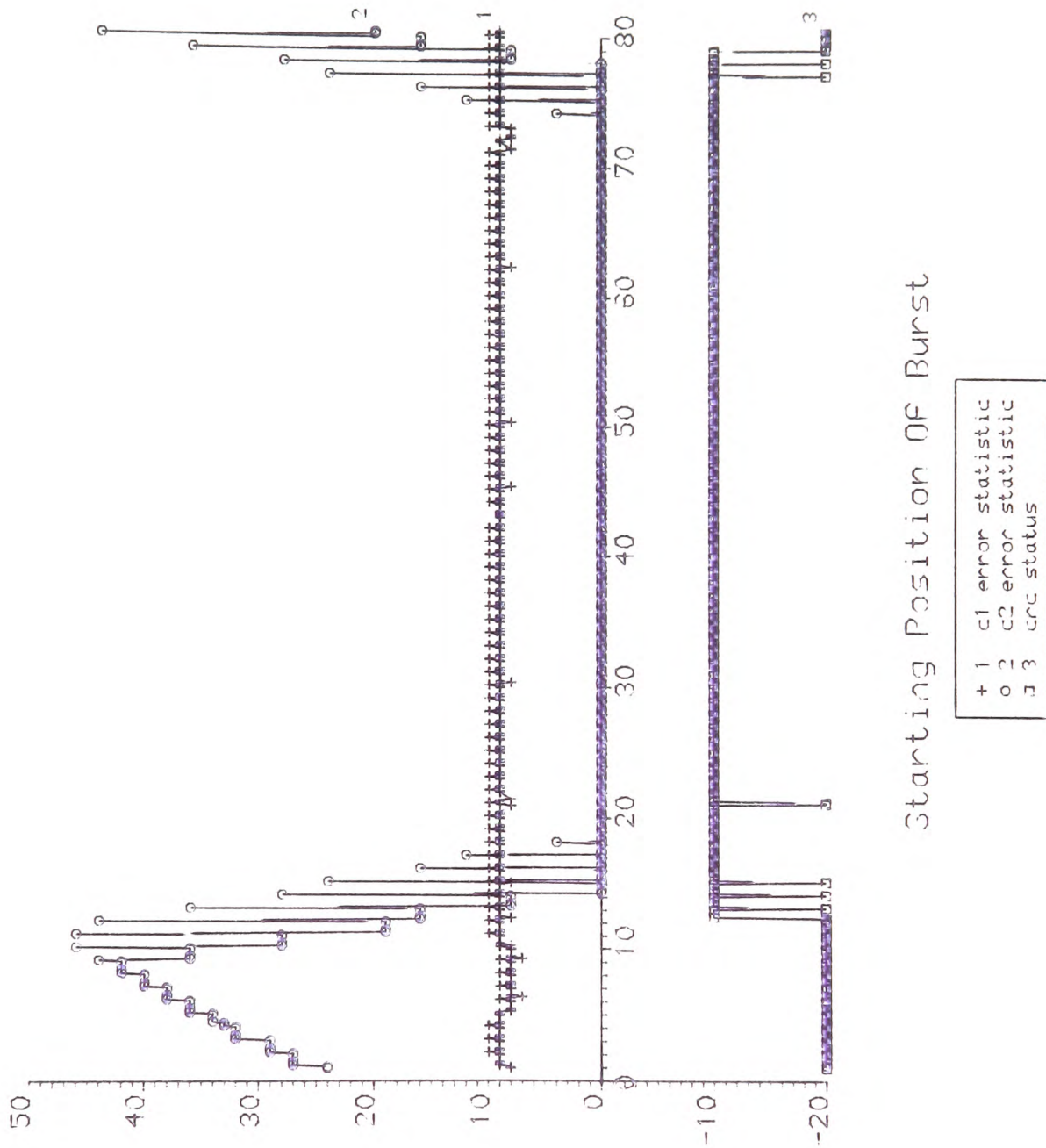


Burst Size (Bits)

+	c1 error statistic
o	c2 error statistic
□	error statistic
x	P error statistic
◇	CRC status

Error Statistic Count and Status

Figure 8.2 : Error Statistics Produced By A 4000 Bit Burst



Error Statistics Count and Status

Figure 8.2 confirm that C1 counts reflect burst length and C2 counts show burst position within a Sector. In this case the C1 count is approximately constant at 10 as the burst moves across the Sector. Note that the C2 count changes in an approximately symmetric fashion across the Sector. Initially the C2 count rises. This is because Deinterleave II has spread the burst over a larger number of Frames. However the C2 decoding remains unable to cope and the number of error Frames increases appropriately. As the starting point of the burst progresses across the Sector the burst error bits are dispersed over still more Frames. The number of error bytes within each Frame falls and C2 error correction is successful in each of them. The C2 count drops to zero and the CRC status moves from FAIL to PASS. The reverse process occurs towards the end of the Sector. Near the transition at Frames 12 and 78, where correction is marginal, the CRC status fluctuates between PASS and FAIL due to synchronization loss.

If the burst contaminates the first 24 bits of a Frame, synchronisation is lost and the entire Frame is treated as being in error. This increases the effective burst length with a corresponding reduction in the error correction. **Figure 8.2** also shows an example of CIRC miscorrection. In Frame 21 aliasing has occurred which has allowed the C2 decoding to apply ostensibly successful correction to the bytes of this Frame and to produce a valid element of the C2 data set. Hence no C2 count is recorded for the Frame. However the incorrect element has been generated and this has been identified at the CRC check.

Figure 8.3 shows the results for a burst error of 7000 bits duration and shows similar correction characteristics. In this case the burst length is close to the limit of CD-ROM error correction capacity. Successful correction is confined to bursts near the centre of the Sector, starting in Frames 33 to 54. In this case, near to the limits of correction performance, the effects of synchronisation loss are still more marked. Limiting performance is again considered in **Figure 8.4** which shows the maximum correctible burst length as the burst moves across the Sector derived from the CRC PASS/FAIL status. Again Deinterleave Strategy III is shown to be the most effective

Figure 8.3 : Error Statistics Produced By A 7000 Bit Burst

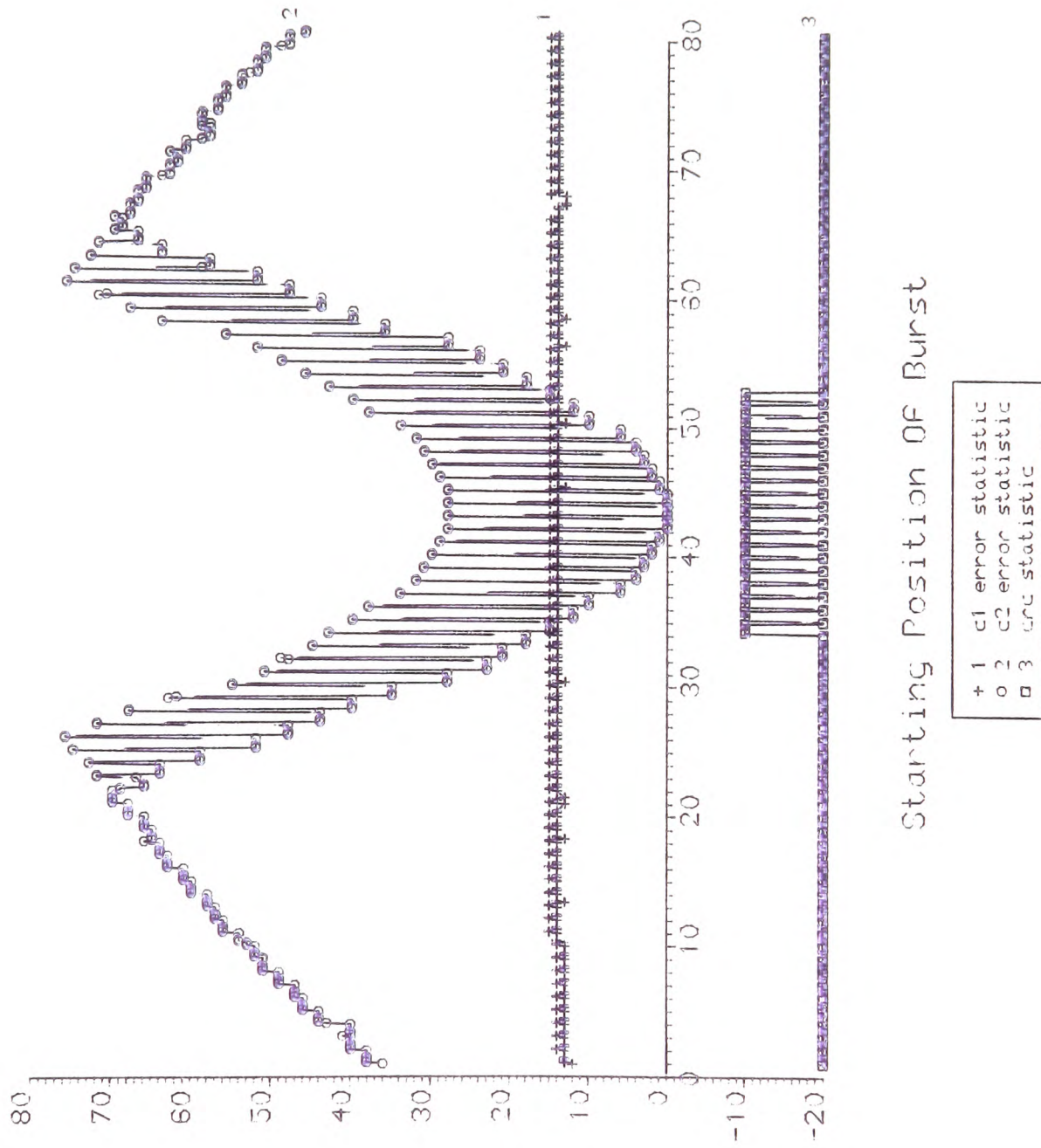
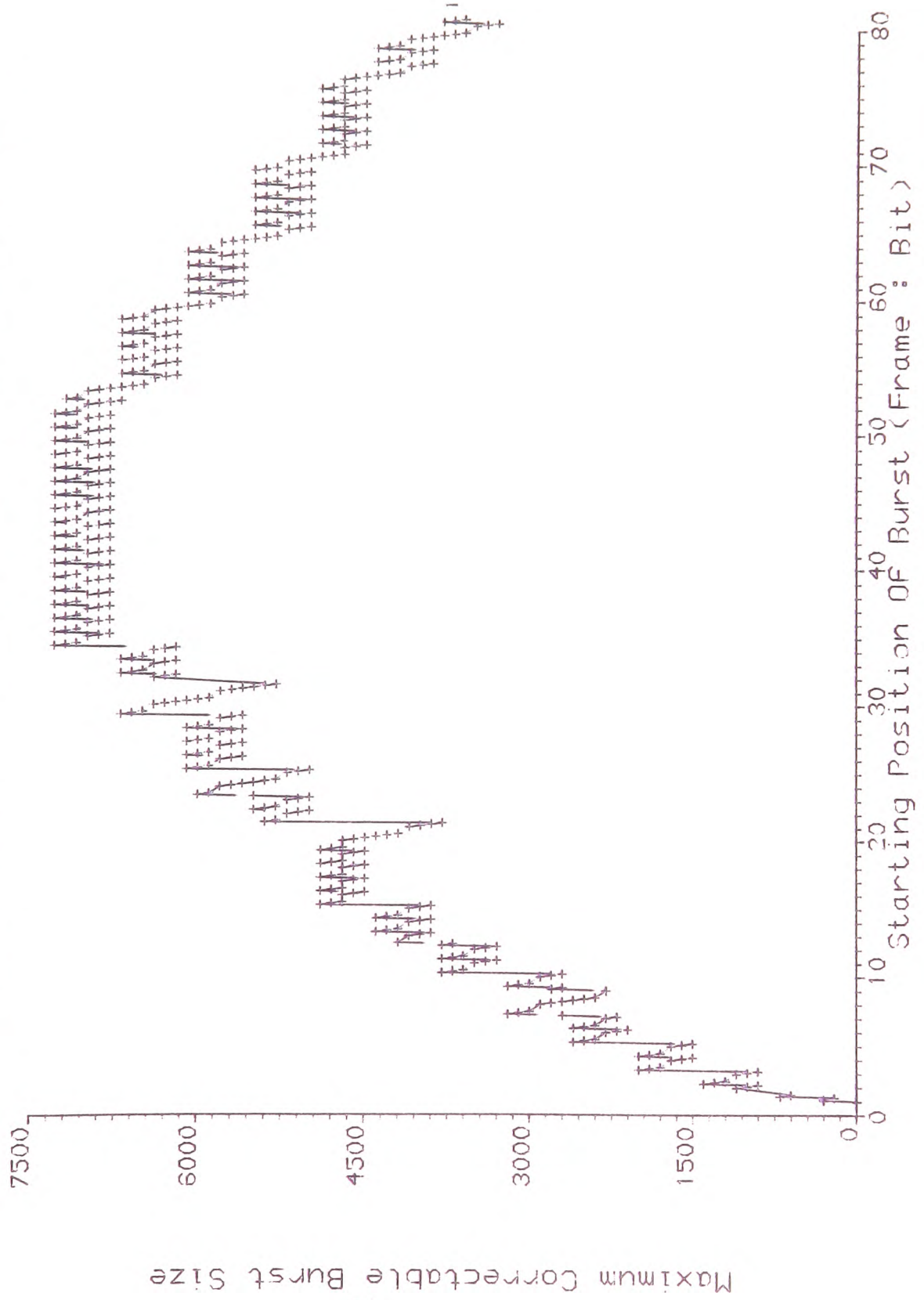


Figure 8.4 : Maximum Burst Correction Along A Sector



dispersal scheme near the centre of the Sector.

These results suggest that the maximum correctable burst is 7,100 bits long. Also, as expected when operating near its limit the CDROM performance is critically affected by synchronisation loss.

8.3 Inferencing Burst Errors

Simple inferencing may be done by observing trends in the data which occur with bursts of specific sizes. This is discussed in **Section 8.3.1**. For maximum likelihood decision making it is necessary to use a dedicated software package.

8.3.1 Inferencing By Observation

In **Section 8.3.2** the Inferencing Package will be discussed in full and some results reviewed. However it is possible to make some conclusions about a burst from the error statistics without reverting to the software package. By observing the magnitude of the C1 statistic it is possible to determine the probable burst sizes affecting the data channel. The C1 error statistic is a reflection of the number of Frames which are in error after De-interleave III and C1 error correction have been carried out. De-interleave III is a mild byte disperser. The bytes in error will only be spread between adjacent Frames.

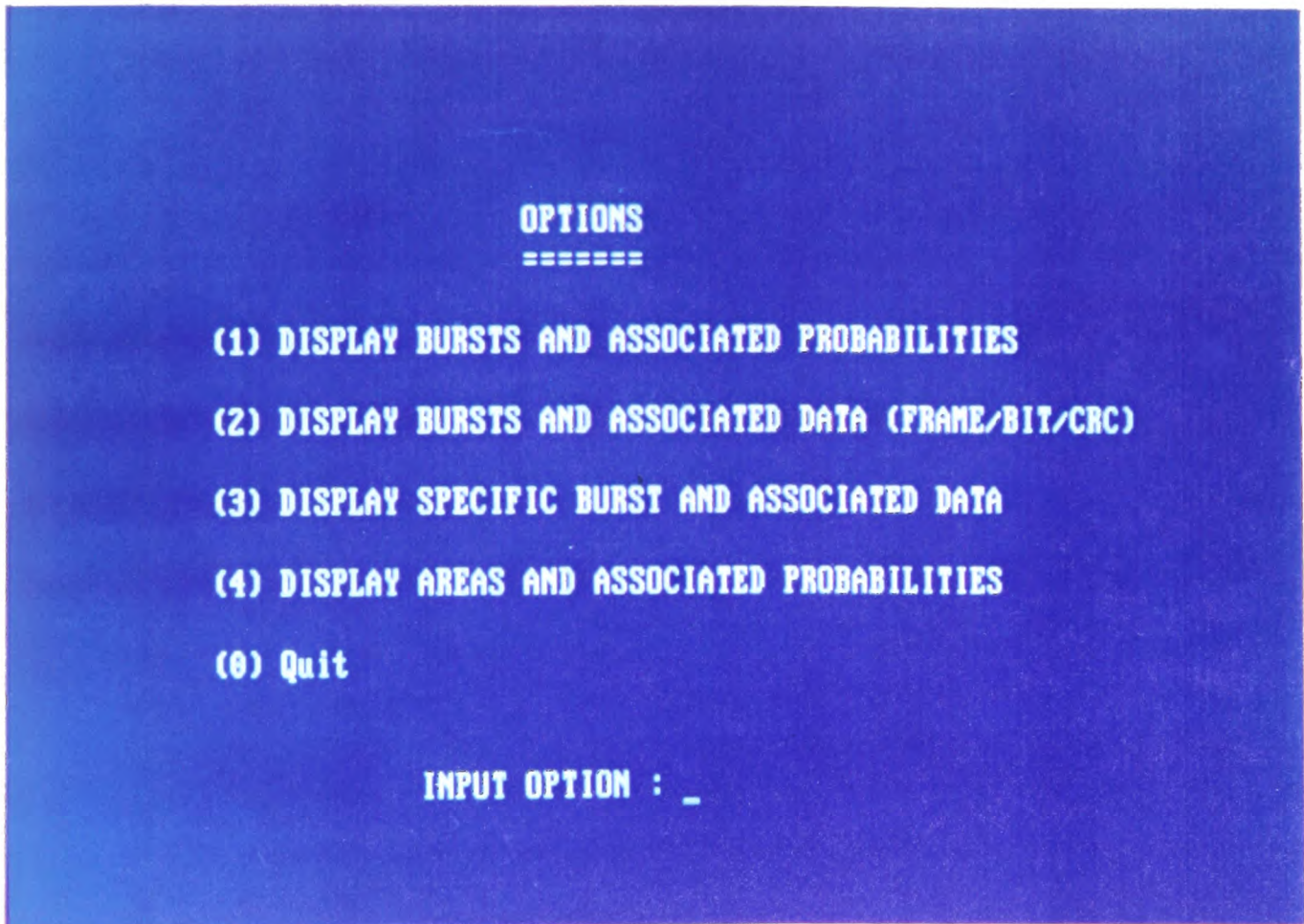
Due to synchronisation loss, burst results only occur in certain ranges. The C1 count attributed to a burst can also vary. For example a burst of 4000 may result in a C1 count between 7 and 10. This is dependent upon its position in both the Frame and the Sector.

8.3.2 Inferencing The Burst

A software package has been produced which carries out the required inferencing. The C1 and C2 values are input into the package. Using these variables the data sets searched and all the possible bursts logged in a linked list. **Figure 8.5** summarises the functions of the package. Since the P and Q values follow the C2 and CRC, they have been excluded

On the search concluding the user may interrogate the package about the bursts which fit the specified values.

Figure 8.5 : Inferencing Package



Consider C1 and C2 counts of eight and 36, respectively. From these variables it is possible to obtain the bursts which generate such outcomes.

8.3.3 Example

The output from these stipulated tests is as follows:

(i) Associated Bursts

The bursts and crc status associated with $c1=8$ and $c2=36$:

<u>burst</u>	<u>frame</u>	<u>bit</u>	<u>crc</u>	<u>burst</u>	<u>frame</u>	<u>bit</u>	<u>crc</u>
3400	6	201	f	3400	10	201	f
3500	6	101	f	3500	6	201	f
3500	10	101	f	3500	10	201	f
3600	6	1	f	3600	6	101	f
3600	6	201	f	3600	10	1	f
3600	10	101	f	3600	10	201	f
3700	5	501	f	3700	6	1	f
3700	6	101	f	3700	9	501	f
3700	10	1	f	3700	10	101	f
3800	5	401	f	3800	5	501	f
3800	6	1	f	3800	9	401	f
3800	9	501	f	3800	10	1	f
3900	5	301	f	3900	5	401	f
3900	5	501	f	3900	9	301	f
3900	9	401	f	3900	9	501	f
4000	5	301	f	4000	5	401	f
4000	5	501	f	4000	9	301	f
4000	9	401	f	4000	9	501	f
4100	5	301	f	4100	5	401	f
4100	5	501	f	4100	9	301	f
4100	9	401	f	4100	9	501	f
4200	5	301	f	4200	5	401	f
4200	5	501	f	4200	9	301	f
4200	9	401	f	4200	9	501	f
4300	5	301	f	4300	5	401	f
4300	9	301	f	4300	9	401	f
4400	5	301	f	4400	9	301	f

(ii) Bursts and Associated Probabilities

The bursts associated with $c_1=8$ and $c_2=36$ and their probabilities are:

<u>burst</u>	<u>probability</u>
3400	0.04
3500	0.07
3600	0.11
3700	0.11
3800	0.11
3900	0.11
4000	0.11
4100	0.11
4200	0.11
4300	0.07
4400	0.04

From these results it can be seen that the most likely burst size ranges from 3600 to 4200.

(iii) Most Likely Areas

The burst occurring with the designated error statistics occur in specific areas. There may be a number of different area, each has an associated probability. In this example there is only one area of interest, however in other cases there will be multiple areas. The areas of bursts associated with $c_1=8$ and $c_2=36$ and their associated probabilities are:

<u>area</u>	<u>probability</u>
3400-4400	1.00

CHAPTER NINE

Obtaining A Measure For CDROM Performance

9.1 The Need To Measure Performance

It has been shown how errors are dispersed and corrected by the CDROM error correction scheme. However, if the final CRC fails then a reseek (or retry) is attempted. Here a second attempt to find and decode the Sector is undertaken. In this case the full error correction scheme is reapplied.

The burst errors considered in Chapters Seven and Eight may be regarded as permanent errors (for example a scratch). Here reseeking will encounter exactly the same error *Woolley[82]*. In contrast, if the errors are transient (electrical interference or a vibration) or semi-permanent (a moveable hair, dust fragment) then a reseek may well result in the data may be corrected at the second attempt *Watkinson[11, pp 202]*.

Clearly the presence of reseekes will affect the access times for a given block¹. of data. An access time is composed of rotational latency, data transfer time and seek times *Barbosa[25,pp 195]*. Whilst the times to access multiple blocks can be measured. Here attention is concentrated on the time to access a single specified block of data. The time taken to access several blocks, some of which require reseekes, can be measured. This only indicates that the reseekes have taken place. It will not show which blocks required retries. To allow a more direct analysis of the performance, attention has been restricted to inter-block access times: i.e. the time to access a single block.

In Chapter Ten it will be seen how both access times and the number of retries can be used to measure the performance of a CDROM system, whilst subject to vibration. The remainder of this Chapter explains how these two measures of performance are obtained from a drive unit.

In order to obtain information regarding access times and reseekes,

¹ In order to avoid confusion a sector is referred to as a homogeneous data block.

dedicated software has been written to facilitate the communication between the CDROM drive unit and the host PC. The role of the SCSI (Small Computer System Interface) card is described here together with ASPI (the Advanced SCSI Programming Interface), the protocol on which the communication is based. The communication commands, both device specific and non-device specific commands. These commands are incorporated into a piece of dedicated software. This facilitates controlled communication between the host (PC) and the target (CDROM).

9.2 The Hardware and Software Requirements

The list of the hardware used is as follows:

- 486 Personal Computer.
- SCSI card which is ASPI programable.
- CDR-1950S CDROM drive.
- Interconnecting cable.

These devices are connected as shown in **Figure 9.1**. The software is written in both 'C' and Assembly language.

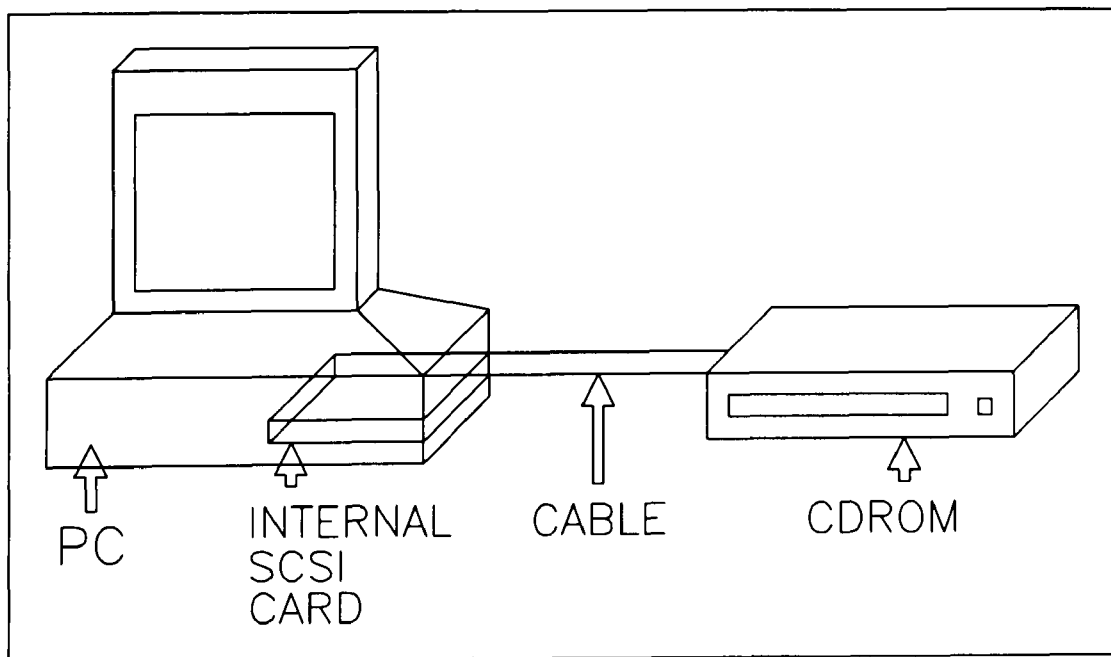


Figure 9.1 : Hardware Set-up

9.2.1 The SCSI

The successive introduction of new computer peripherals has led to a widely recognised need for a common interface between host computers and these peripheral devices *NCR[83]*. The SCSI was introduced to fulfil this need. The SCSI bus has a standard interface with the computer and each new peripheral is viewed as another device connected to the SCSI bus. There is no longer any need for specialised hardware to accommodate each new device to the computer itself.

SCSI specifies two communication protocols between the target and host. These are asynchronous and synchronous. Asynchronous communication requires handshaking for each byte transferred, whereas synchronous communication transfers a series of bytes before further handshaking is necessary. Here only the synchronous mode is used. An example of the communication is given in **Figure 9.2**. The initiator decides on an operation. It sends a command code specifying that operation to the target. The target then executes the specified operation.

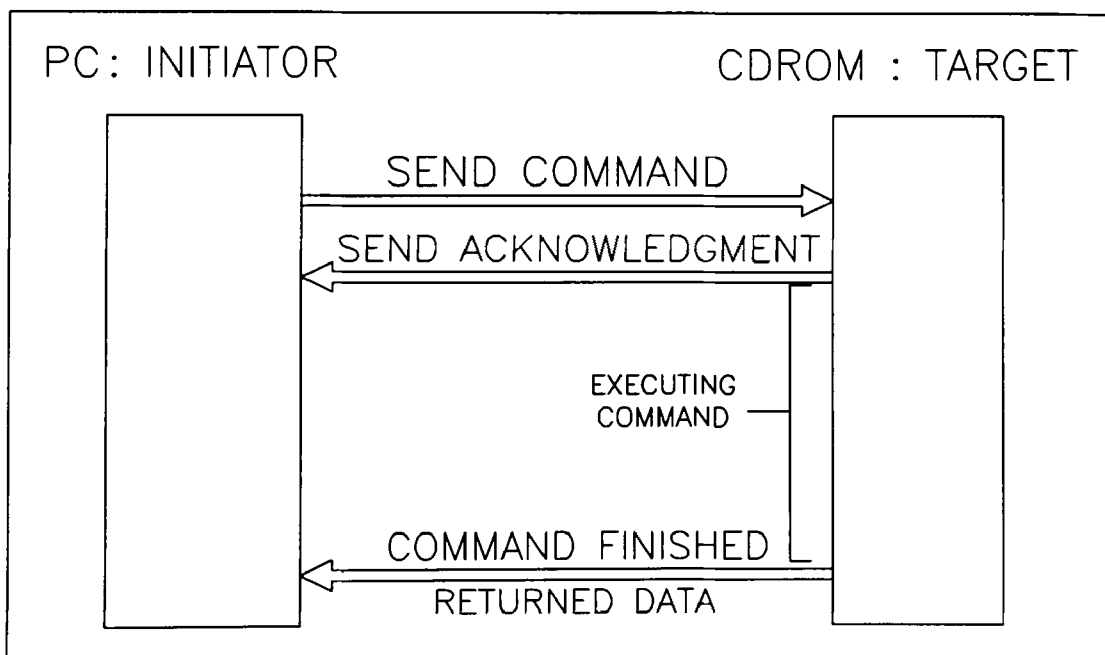


Figure 9.2 : Simplistic Communications

The figure provides a simplified view of SCSI communication. In practice data transfer is more involved than simply sending a control code. A command is sent in the form of a control contained within a data block called

a SCSI Request Block (SRB). The command code and additional information necessary to execute the operation are known collectively the Command Data Block (CDB). This forms part of the SRB. These data structures are crucial to the use of ASPI.

9.2.2 Using ASPI To Communicate With A CDROM Drive

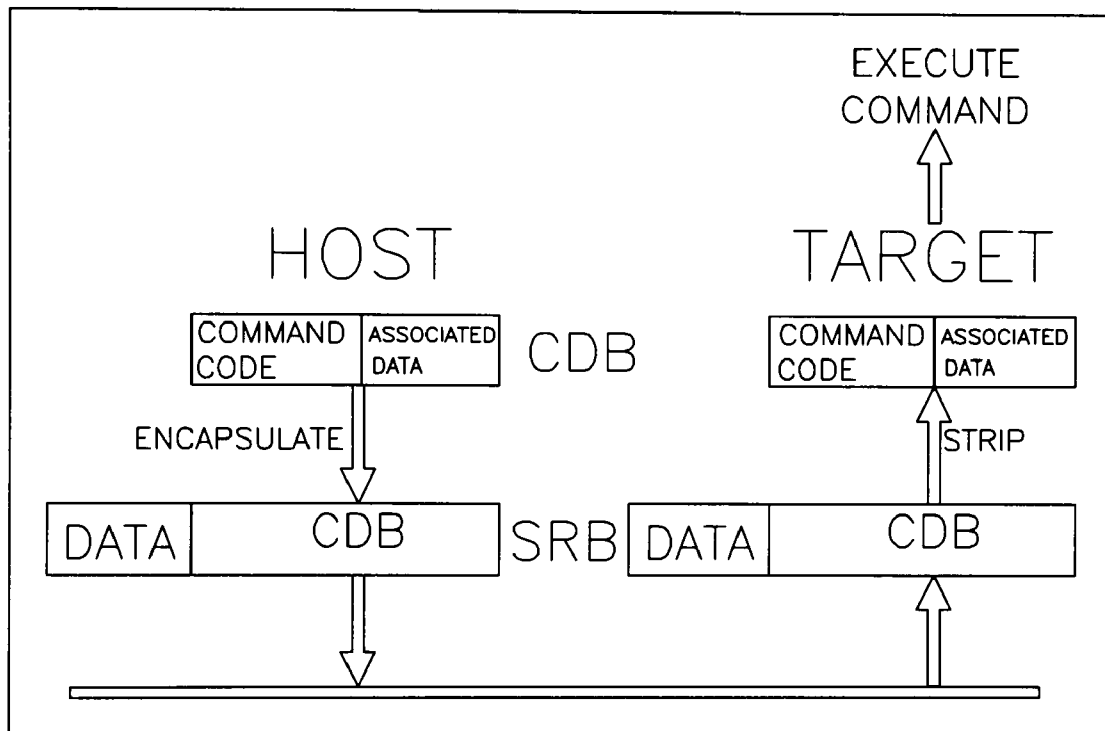


Figure 9.3 : Relationship Between SRB and CDB

Figure 9.3 illustrates the relationship between the SRB and the CDB and how the data is used. To enable full communication ASPI must :

- obtain the address of the target device must be resolved in order to locate the point to which data shall be sent;
- establish a communications route between the two devices, so that data can be sent and received with relative ease;
- establish a procedure for obtaining the information specified in **Section 9.1** (access times and retry counts).

A dedicated piece of software has been built so that these functions are transparent to the user. This is discussed in **Appendix B**.

9.3 The Low Level Communications Objectives Of ASPI

The ASPI Manager manages the PC and provides the hardware independent ASPI for SCSI applications. This manages the communications channel and provides the SCSI commands. It is implemented transparently using MS-DOS. This SCSI card is manufactured by the ADAPTEC company who originally produced the ASPI protocols. User information on ASPI has not yet appeared as yet in the public domain. It is believed that this is the first academic application of this protocol.

Furthermore the documentation that does exist appears only to exist in synoptic form and outlines assembly code which may be used *Adaptec*[84]. For the present work C was used. A High Level Language was considered easier to manipulate and update. Assembly language is used in order to carry out the low level requirements. These are incorporated as functions which are called by the C main program.

9.3.1 Getting The ASPI Entry Point

The ASPI entry point is the address from which the ASPI protocol is called prior to communication via the SCSI bus. It must be obtained from the ASPI manager. Details are given in **Appendix B**.

9.3.2 Sending and Receiving Data

The main program accesses two assembly modules. Both modules contain low level functions which may be called from the C program.

The first module contains the function responsible for obtaining the ASPI entry point. Once this address is obtained communication can be started. The second assembly module contains the function which facilitates communication. This module is continually called in order to send the SRB to the target. This may be considered as the send/receive module.

The main program is also responsible for file handling, message interpretation and message handling. Full listings are given in **Appendix B**.

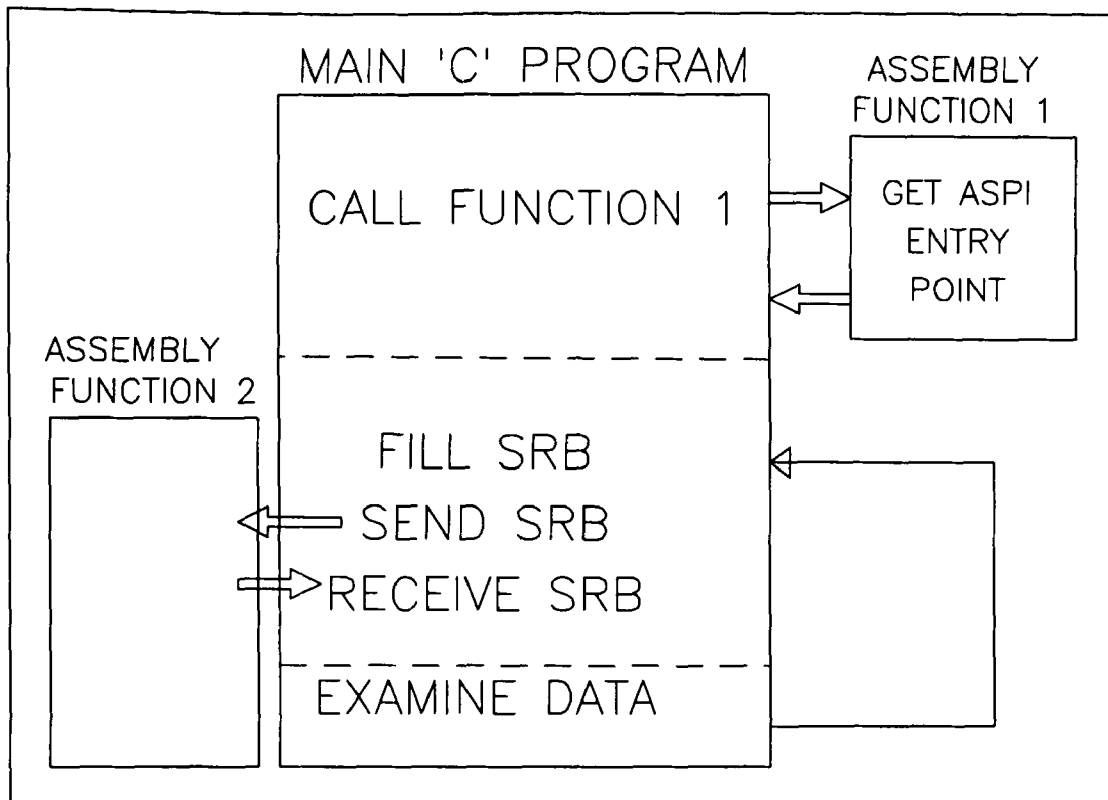


Figure 9.4 : Program Structure

Figure 9.4 illustrates how the C program initially calls the first assembly function and so obtains the ASPI entry point. The SRB, including the CDB is now filled with command data by the main C program.

The contents of the SRB are sent to the CDROM device. The C program waits until the target sends an updated SRB to the host. Its contents are examined. It can then be reset so that another command can be sent to the target device for execution.

9.3 Transferring Data Using The SCSI Bus

Establishing a communication channel is non-device specific. However obtaining information from the CDROM is device specific and requires device specific commands. The two significant elements of each SRB are the **command code** and the **status field**.

The command code specifies the non-device specific command which is to be performed. For example, one command is **Get Device Type**. This

indicates the type of devices which are installed on the SCSI bus. However, the most important command code is the **Execute SCSI Input/Output** command. This facilitates data transfer.

The status field refers to the progress of a command: whether or not it has been completed and if errors have arisen. The status values and their descriptions are as follows:

<u>Status value</u>	<u>Description</u>
00h ²	SCSI request in progress.
01h	SCSI request completed without error.
02h	SCSI request aborted by host.
04h	SCSI request completed with error.
80h	Invalid SCSI request.
81h	Invalid Host Adapter Number.
82h	SCSI Device Not Installed.

A zero status is reported when a SRB has been sent but not returned. Since the software must pause at this point, until the SRB is returned, the status field is constantly polled for a non-zero value.

A command request has a number of associated fields. Important fields are : the Data Allocation Area Length and Address, the Sense Allocation Area Length and Address , the CDB Length and Address, the Host Status and the Target Status.

The CDB defines the area in which the device dependant commands will be placed *Hitachi[85]*. These commands describe the I/O function to be performed. The address of the data area and its length are placed in the SRB.

Similarly for the Sense and Data Allocation Area. The Sense Allocation Area is filled with data when an error status is returned from a command. This sense data may be used to deduce where and why an error occurred. It will be shown in **Section 9.3.2** that this is significant when diagnosing errors.

The Data Allocation Area may be filled with data by either target or host as specified in the CDB. For example, if a **Read** operation is specified

² All values are in hexadecimal

then data will be read from the medium and placed in the Data Allocation Area.

The SRB may be considered as a packet of data which may be accessed by both the host and the target via the SCSI bus. This is done by passing the address of the SRB structure to the Send Data Module.

9.3.1 Error Status Flagging Of Host and Target Devices

Initially the length and contents of the CDB are set to zero, so that no device operation is selected. This enables the communication channel to be tested in isolation.

The initial aim is to test whether the Send Assembly module was correctly transmitting the contents of the SRB to the drive unit. Problems which could arise include: incorrect addressing, the difficulties associated with imbedding assembly language in C, and sharing variables between languages.

The **Send Data** Module uses the SRB pointer to send the address of the SRB to the CDROM hardware. The SCSI bus is used to access the SRB dynamic stack. The status field of the SRB is initialised to a value which would not be produced by a SCSI request. The SRB is now sent.

There are a number of possible responses. If information is not sent correctly the status field is not altered. The Send module must then be examined for erroneous communication. If an error status is returned the communication channel is functioning correctly. However one of the devices is causing an error. Two further status fields are examined to determine which device is at fault.

Host States *Adaptec*[84]:

- 00h Host Adapter did not detect any error.
- 11h Selection Timeout.
- 12h Data overrun
- 13h Unexpected Bus Free
- 14h Target bus phase sequence failure

Target States :

- 00h No Target Status
- 02h Check status (i.e.Sense Data in Sense Allocation Area)
- 08h Specified Target/LUN busy
- 18h reservation conflict

The most likely error attributed to the host is a timeout, in this instance the Target device has taken too long to respond to the SRB. This may be due to either incorrect addressing or a busy device. The following example illustrates the three status fields associated with sending an SRB:

```
SRB Status : 04h
Host Status : 11h
Target Status : 0h
```

The SRB Status:04h states that the SCSI request was completed with error and the host status reports a timeout. The target status reports no problem. This occurs when no target devices is installed on the SCSI bus. If an error status is reported then data is stored in the Sense Allocation Area.

9.3.2 The Sense Allocation Area

The Sense Allocation Area (S.A.A) may be regarded as an array, it is a data area within the SRB. In this example the SRB status indicates that an error has occurred. The Target status reports that sense data is in the Sense Allocation Area and that the Host status did not detect any error. In the following example, the sense data is the information which aids error diagnosis.

	Sense Allocation Area For Absent Medium			
SRB status : 4	70	00	02	00
Host Adapter Status : 0	00	00	00	06
Target Status : 2	00	00	00	00
	3a	00	00	00

A diagnosis of the error is possible by cross referencing this information with the error reports in the CDROM device specifications *Hitachi[85]*. The three important items are the Sense Key, the Additional Sense Code and the Additional Sense Length, all of which lie within the Sense Allocation Area. Important codes for each of these are:

Sense Key codes (lower 4 bits of byte 2) :

02h	Logical unit not ready.
04h	Hardware error.
05h	Illegal Request (illegal parameter in CDB)

Additional Sense codes (byte 12) :

04h	Logical unit not ready.
11h	Unrecovered read error.
24h	illegal field in CDB.
3ah	Medium not present.

Error code (byte 0):

70h	Current errors.
-----	-----------------

Additional Sense Length (byte 7):

0nh	There are n additional bytes of data following.
-----	---

Whenever data is placed in the S.A.A the error code of 70h will be present in byte zero. In the above example the sense and additional sense are 02h and 3ah respectively.

The Sense Key indicates that the device is Not Ready. The logical unit addressed cannot be accessed. The additional sense key, reports that the medium is not present, i.e the compact disc was absent.

This is a simple example. The two sense keys enable more complex situations to be diagnosed. Use of the S.A.A and sense key descriptions allow diagnosis of problems.

With successful transmission of the SRB, the contents of the CDB are filled and the device specific commands may now be used.

9.3.3 Specifying Operations Using The CDB

In the same way that the SRB is used to facilitate SCSI commands, the CDB is used to perform specified input/output operations. In the example above no data has been sent in the CDB. This was to simplify SCSI communication and limit the factors which cause error. The CDB is a small packet of information which is contained in the SRB. As well as specifying the required operations the CDB also contains data associated with that operation. A good example is the **Read** command. Here the starting address and number of blocks to read must also be given. In the example below failure to insert correct information in both position and value has led to a status check, implying failure.

Status Values	S.A.A For An Incorrect CDB			
SRB Status : 04h	70	00	05	00
Host Status : 00h	00	00	00	06
Target Status : 02h	00	00	00	00
	24	00	00	00

The Sense Key indicates that an illegal request has been made and furthermore the Additional Sense Key indicates that an illegal field is present in the CDB.

9.3.4 Obtaining and Sending CDB Data

Some CDB commands require data to be sent back to the host. This can be either the contents of a data Sector, or details of the performance statistics, e.g. the retries. To keep the SRB at a manageable size, only the address of the Data Area is stored in the SRB.

The device specific commands are used to obtain a block of data from the medium. Commands were designed using the Hitachi product manual *Hitachi*[85]. With devices from other manufacturers their product manuals must be used *Sony*[86] & *Philips*[87]. The **Verify** command is used to check the integrity of a block on the CDROM medium using CIRC and the CRC. This command was used in preference to the **Seek** or **Read** command. The **Seek** command seeks a block of data without verifying it. The **Read** command does verify the data, but it also returns the decoded raw data to the host. Since this is unnecessary and requires a large data area to be considerably large (2048 bytes), **Verify** is the preferred command.

When data is read from the CDROM, log statistics are updated within the drive unit. These statistics include the number of retries. Here the total number of retries, the number per command and the address of the last retry is given. There is also a field which identifies the cause, however this is not defined in the current release of information from Hitachi. These statistics are obtained by the PC, using the **LogSense** and the **LogSelect** CDB commands.

LogSelect is used to specify which statistics are to be recorded. After each block is accessed, **LogSense** obtains the error statistics. In addition, **ModeSense** and **ModSelect** may be used together in order to change the CDROM mode. For example, the maximum number of retries may be altered. A default maximum of ten retries was used throughout.

9.4 The Access Times and Retry Counts From The Drive

The communications link has been established and the device specific CDBs have been constructed. These are used to obtain performance measures for the transient and semi-permanent errors.

As discussed in **Section 9.1**, the inter-block access time is recorded, i.e. the time taken to read an individual block of data, including retries. The access time is taken to be the interval between Time A and Time B.

Time A is taken before a block is requested by and SRB being sent. It is recorded immediately before the Send assembly module is accessed.

Time B is the time recorded upon the SRB being returned successfully. This is recorded when a successful status is reported from the Send Module.

This interval includes the access time plus an additional command overhead, which is constant for each access.

CHAPTER TEN

Measured Performance Of The CD-ROM Against Transient Errors

10.1 Introduction

In this Chapter the data generated by the methods outlined in Chapter Nine is presented and analysed. The performance measures are obtained both in a standard working environment and when the CDROM drive is subject to vibrations. The results are outlined in **Section 10.3** and presented in **Section 10.4**. The results for the vibrated system were obtained using dedicated equipment at the British Gas Engineering Research Station in Killingworth.

Blocks are read from the CDROM disc in sequential logical order. This is not necessarily the order in which they are physically recorded, which is dependant upon both the manufacturer and the mastering software. In the results below blocks are presented in two ways. Either individual blocks or a number of sequential blocks are used. A number of sequential blocks used are referred to as a block sample. The experiments are carried out by verifying one thousand logically sequential blocks in each case.

10.2 Test Disc Details

Two discs are used in the following investigations. Disc A is as clean as possible, having being stored and handled with care. Consequently it contains no errors. Disc B has been partially covered by a piece of non-transparent adhesive tape, as illustrated in **Figure 10.1**. The effect of the tape will be similar to that of a scratch or obstruction in that the underlying data cannot be read by the optical hardware.

Figure 10.1 : CDROM With Error



Note that the adhesive tape cannot be positioned to cover given blocks of data since the disc does not contain a common reference point from which to measure any angular displacement. So, in contrast to **Chapter Seven** a burst error of known length and position cannot be investigated.

Measurements were taken for Disc A both near the centre of the disc and near the rim. The inner blocks area were read from block zero to 1000. Block zero is the first block of the data area. For the outer area the same number of blocks are read, however access begins from block 200,000. The error statistics for the first block access (0 and 200,000) are discarded for reasons which will become apparent later in this chapter. The access times will increase for blocks further from the centre of the disc and it is also possible that the physical faults, such as disc wobble, will affect the results.

For each run of the software the inter-block access time and the number of reseekes are recorded. The access times are recorded to the nearest

hundredth of a second. After ten reseek a block is deemed to contain permanent errors that cannot be corrected and is abandoned. In such a case the access time does not represent the time taken to access a block, but the time taken to carry out the ten reseek, use both CIRC and the CRC.

10.2.1 Typical Error statistics

Figure 10.2 illustrates typical error statistics associated with a run of six blocks. To limit the quantity of data the retries are only recorded when non-zero.

Figure 10.2 : Output From Performance Software

<u>Inter Block Access Times</u>	<u>Read Retry Count</u>
1 : 22	
2 : 17	
3 : 22	
4 : 15	
5 : 46	5 : 2
6 : 17	

In the instance of a retry occurring for a given block, the access time of that block should increase markedly from the normal range. This can be observed in the figure where block five has two associated retries, this is referred to as a soft retry. A hard retry is described as a block where the maximum number (ten) of retries have been applied, without success. Here the access time is also seen to rise. Despite this apparent relationship between these two statistics it will be shown later that this is not always the case. For this reason it is important to use both statistics.

10.3 Outline Of Experiment

All experiments were carried out in both standard and hostile environments. The standard environment is defined as the normal working conditions of the CDROM drive and medium.

Unlike humidity and temperature there is no vibration specification in ECMA and other standards. For this reason vibration was chosen *ECMA[18,pp 4], BSI[19,pp 6] & Hitachi[88,pp 1193]*. This is applied in three forms: a shock test; fixed frequency vibration; and swept sine vibrations.

Note that it is not possible to run repeated runs of the software for swept frequency and sudden shock tests, for this reason the results rely on a single run. In most results an average is taken over a number of blocks.

10.3.1 Standard Environment With Disc A

In this environment the performance of this disc in the drive unit should be at an optimum level, i.e. low access times and zero retry counts should prevail. For both data areas the block access times will vary. These variations should be small in comparison to those generated by reading problems.

It also likely that the average access time observed in each areas will differ. This is due to increased latency *Barbosa[25,pp 31]* and the Constant Linear Velocity. For this reason it is necessary to only compare like areas of a disc.

10.3.2 Standard Environment With Disc B

Information is accessed from the area of Disc B near the centre. This is the area affected by the incorporated error. It is expected that the existence of errors will be highlighted by the presence of both high access times and non-zero retry counts. These retries are likely to be hard errors, i.e. the block in question is affected by a permanent error.

10.3.3 Hostile Environment With Disc A

(i) Shock Test

Only the inner data area of Disc A is used for this test since these results are almost identical. The drive is affected by an impulse which is unmeasured.

This is

in contrast to (ii) and (iii) where the applied vibration is measured. The physical displacement of the laser will prevent access of the current block. Its subsequent performance is of significance.

(ii) Constant Fixed Vibration

The frequencies used in these tests range from 10 to 100 Hz, in steps of 10 Hz. In addition the maximum acceleration levels applied range from 0.5 to 2 g, in steps of 0.5 g.

In order to establish the effect of a given frequency and acceleration level, the drive is exposed to each combination in turn. For example, the first experiment is 10Hz at 0.5g, the second is 20Hz at 0.5g, and so on.

(iii) Swept Sine Vibration

Identical ranges of frequency and acceleration to those in (ii), apply to these experiments. In contrast to the previous experiments the frequency increases from 10 to 100Hz as the experiment progresses. To maintain a constant maximum acceleration the amplitude decreases as shown below.

The equation of motion of the vibration is:

$$x'' + \omega^2 x(t) = 0 \quad (10.1)$$

with solution:

$$x(t) = A\cos(\omega t + \phi) \quad (10.2)$$

The acceleration is:

$$x'' = -A\omega^2\cos(\omega t + \phi) \quad (10.3)$$

with peak value: $A\omega^2$.

Hence for constant peak acceleration:

$$A \propto \frac{1}{\omega^2} \quad (10.4)$$

10.3.4 Hostile Environment With Disc B

The hostile environment discussed in Section 10.3.3 are identical to those applied to Disc B. However, in this case two error mechanisms will affect performance: both surface errors and vibration. By comparison of the data produced in both standard and hostile environments the effect of the latter may be established.

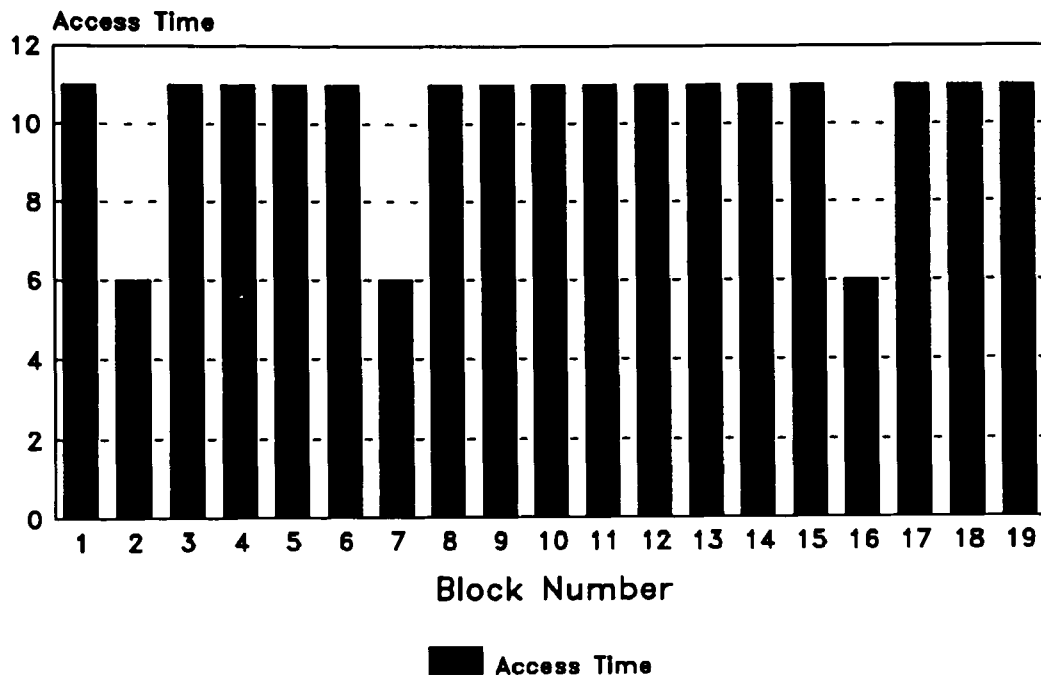
10.4 Experimental Results

The experiments follow the order outlined in Section 10.3. The relevant graphs are illustrated for each experiment.

10.4.1 Standard Environment With Disc A

(i) Inner Area

Graph 10.1 depicts the typical access times associated with the inner data area.

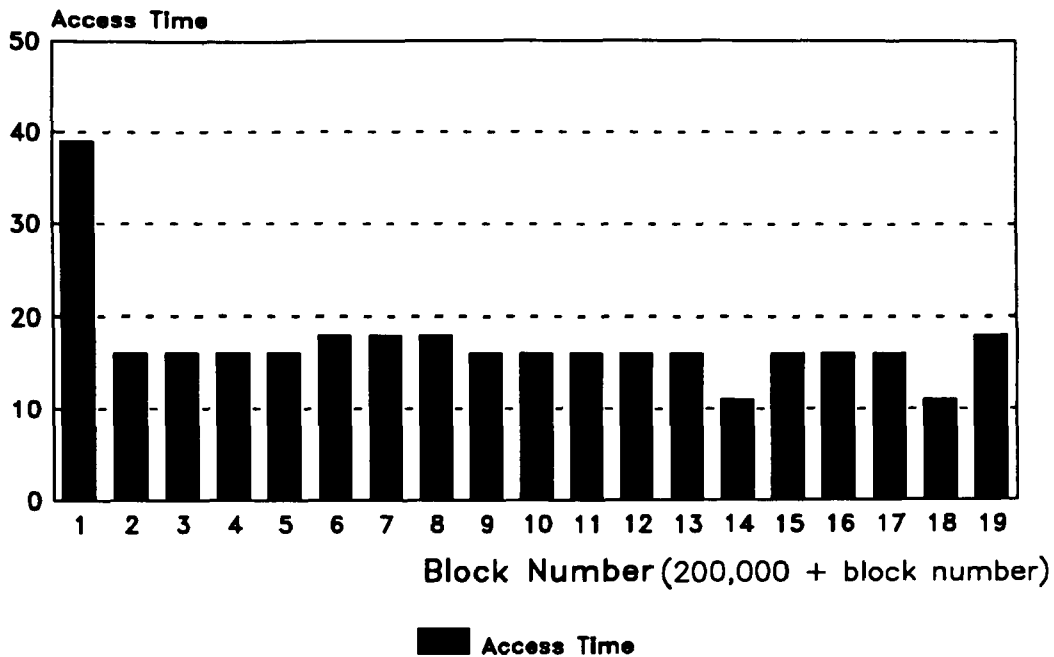


Graph 10.1 : Access Times For Inner Data Area

The access times are not constant. They vary between 6 Hsec (Hundredths of a second) and 11 Hsec. As would be expected for a clean disc no retries were observed.

(ii) Outer Area

In Graph 10.2 typical access times for this area are illustrated.



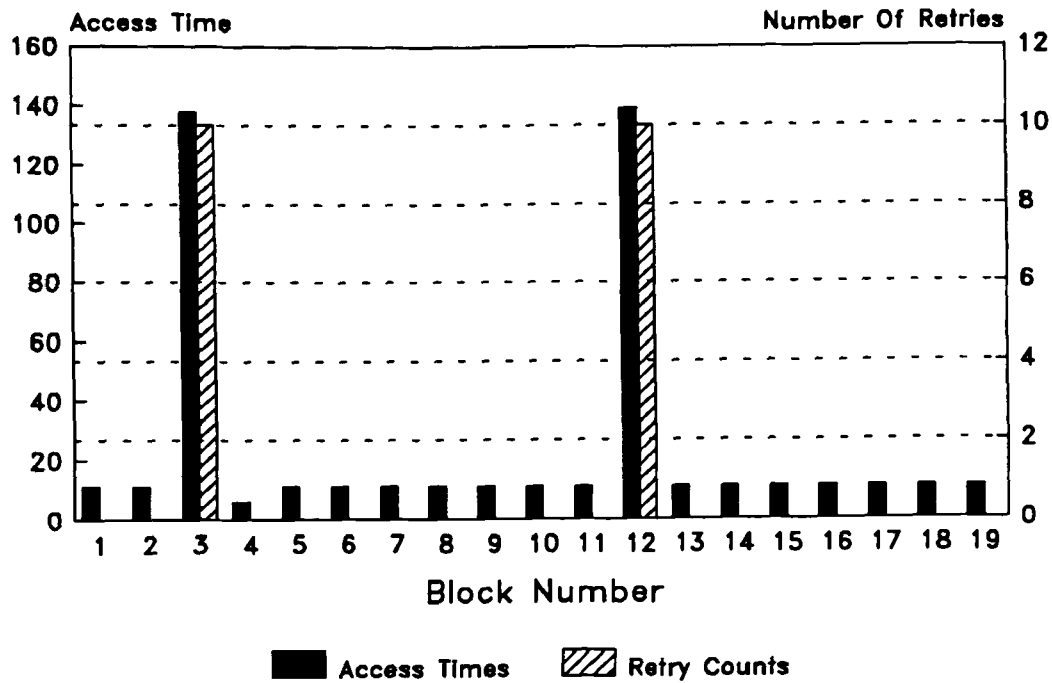
Graph 10.2 : Access Times For The Outer Data Area

Excluding the first, these vary between 11 Hsec and 18 Hsec. The access times are larger than those shown in Graph 10.1. This difference is due to the increased latency experienced by the laser as it accesses data nearer the rim. Again no retries are observed.

The access time for the first block is atypical. This is due to the system being at rest when the block access was requested. If a drive is unused for a period of time the unit enters a waiting state. Upon a data block being requested the read laser must move from rest and find that block's position on the disc by reading the Table of Contents. The Table Of Contents is positioned at the centre of the disc. A large initial access time always occurs when a block is accessed from an inactive drive. Hence the initial block of data is always discounted.

10.4.2 Standard Environment With Disc B

Graph 10.3 illustrates the statistics associated with Disc B.



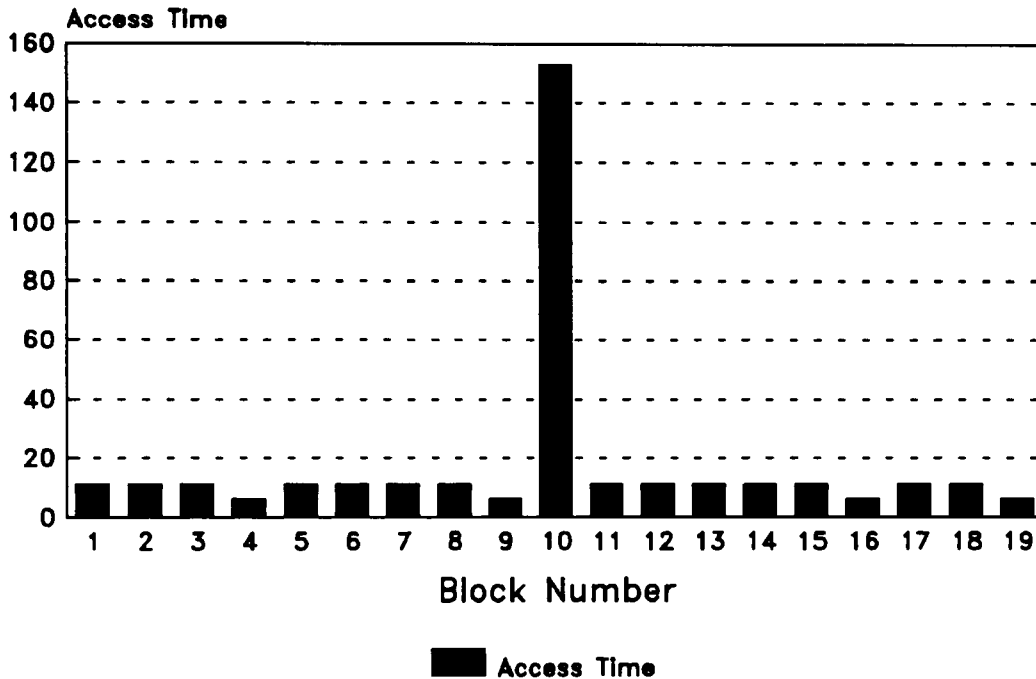
Graph 10.3 : Error Statistics Generated By Disc B

Here blocks 3 and 12 are observed to be affected by the surface error, with access times of 132 and 137 Hsec respectively. The blocks in error experience maximum retries, due to the permanent error upon the disc medium. Here the access time represents the time taken to reseek the block ten times and attempt decoding.

10.4.3 Hostile Environment With Disc A

(i) Shock Test

Graph 10.4 illustrates the affect of a shock test on the drive. This test can be seen to affect block eleven significantly. Here the block access time can be seen to rise to 153 Hsec. Of further significance is the lack of any retries.



Graph 10.4 : A Shock Test On The Inner Data Area Of Disc A

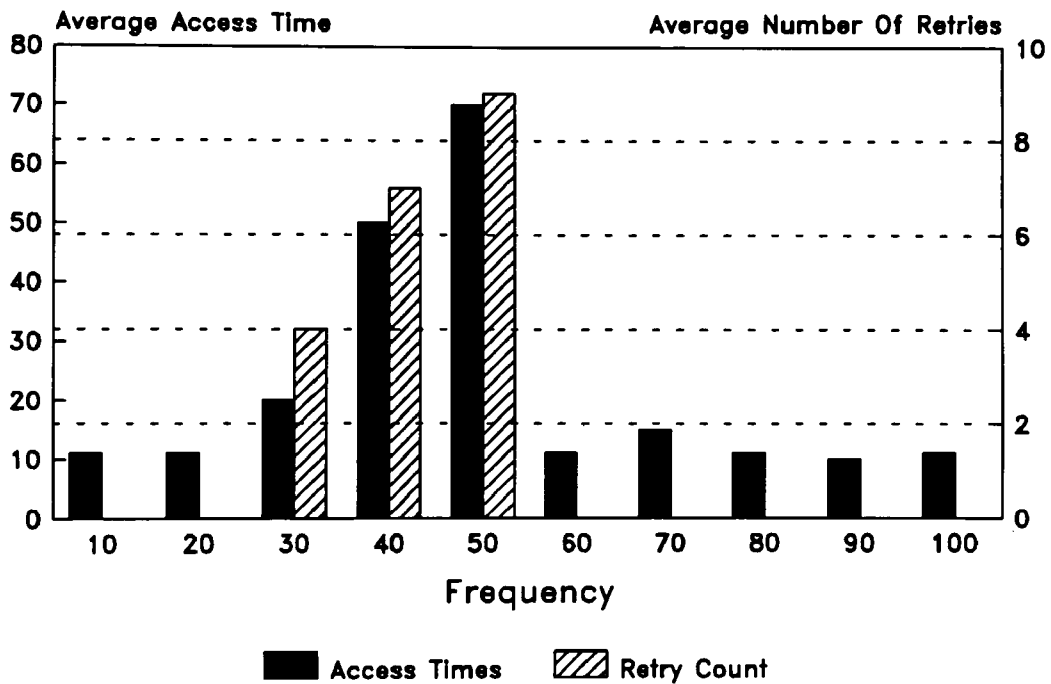
The applied force affects the read laser such that it cannot find the next sequential block, in this case block eleven. However since the block was not found, the data is not read. For this reason no retries are present, since the block is retrieved successfully upon the block being found. Further tests did cause retries to occur. The effect of a shock is therefore dependant upon the reading stage, whether finding (seeking to) or reading (accessing) the block.

(ii) Constant Vibration

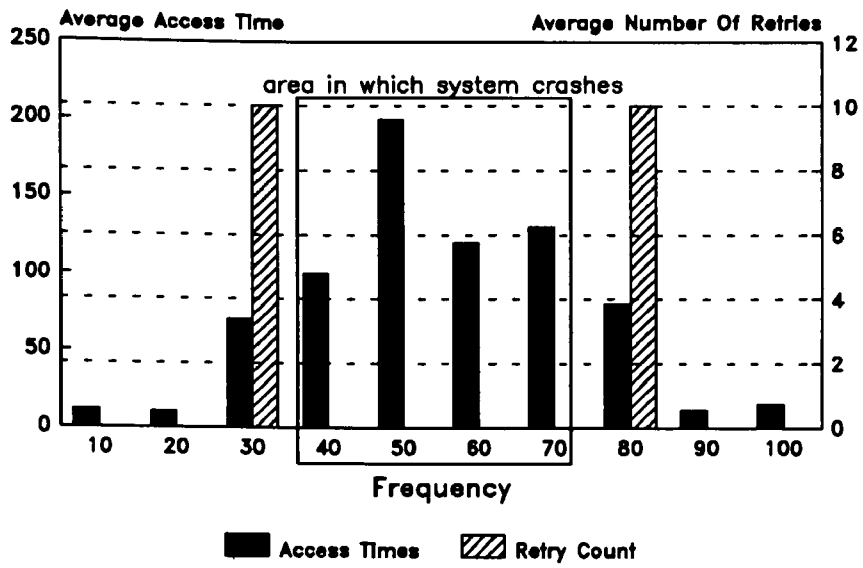
Inner Data Area

Graphs 10.5 to 10.8 depict the average access times and retry counts for the inner data area which are plotted against frequency for acceleration levels 0.5, 1, 1.5 and 2g respectively.

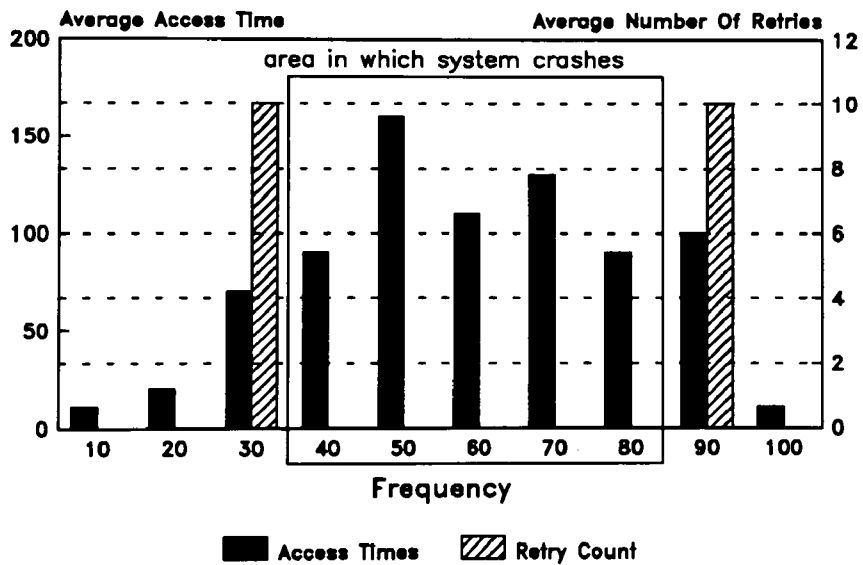
Here the adverse frequency regions for each acceleration level can be identified by the rise in both statistics. As the acceleration level increases the region of 'none functioning' broadens. At some frequency values the system is unable to retry or access due to the inability to find the specified block. In this case the system is observed to crash without completing the specified block accesses. This is denoted on each graph by a box, within this the system crashes.



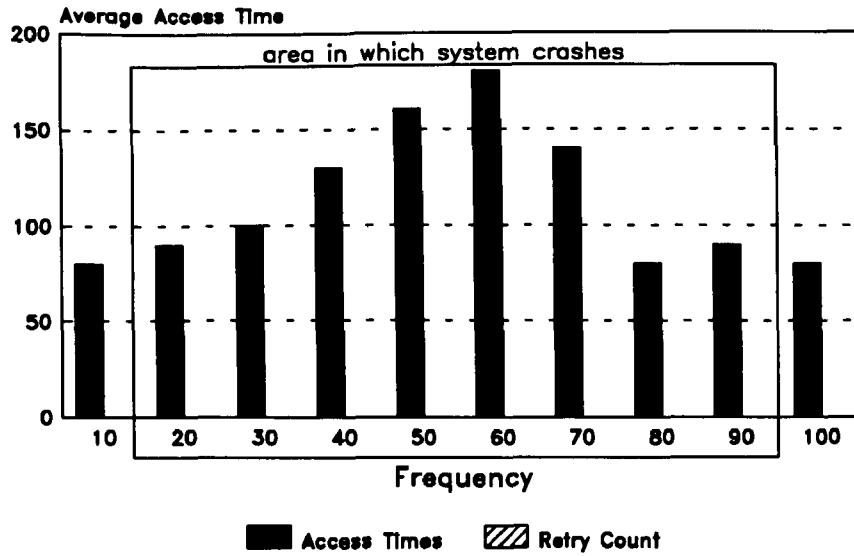
Graph 10.5 : Error Statistics Arising From Fixed Frequency With 0.5g Tests



Graph 10.6 : Error Statistics Arising From Fixed Frequencies With 1g Tests



Graph 10.7 : Error Statistics Arising From Fixed Frequency Tests With 1.5g



Graph 10.8 : Error Statistics Arising From Fixed Frequency With 2g Tests

Figures 10.3 to 10.6 depict the area of vibration for each acceleration level and the effect upon the performance of the drive with the inner data area. In each figure the white area depicts the region where the drive behaves normally. These and all such figures are produced by the same data used to produce the associated graphs.

KEY

- normal functioning
- ▨ area of retries
- ▩ high access times
- ▧ drive unable to function (crashes)

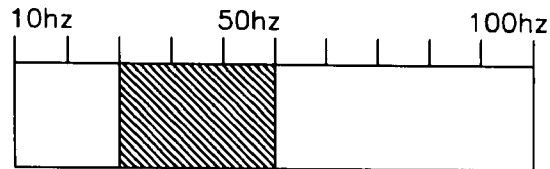


Figure 10.3 : Illustration Of The Effects Of Frequency On Drive At 0.5g

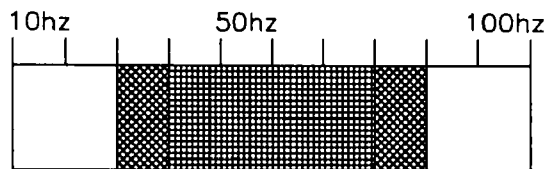


Figure 10.4 : Illustration Of The Effects Of Frequency On Drive At 1g

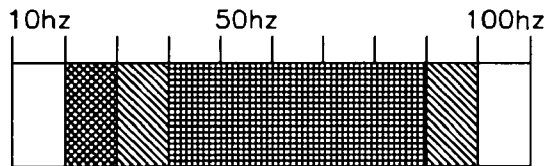


Figure 10.5 : Illustration Of The Effects Of Frequency On Drive At 1.5g

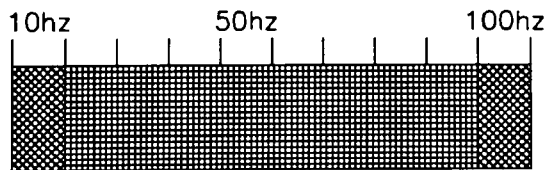
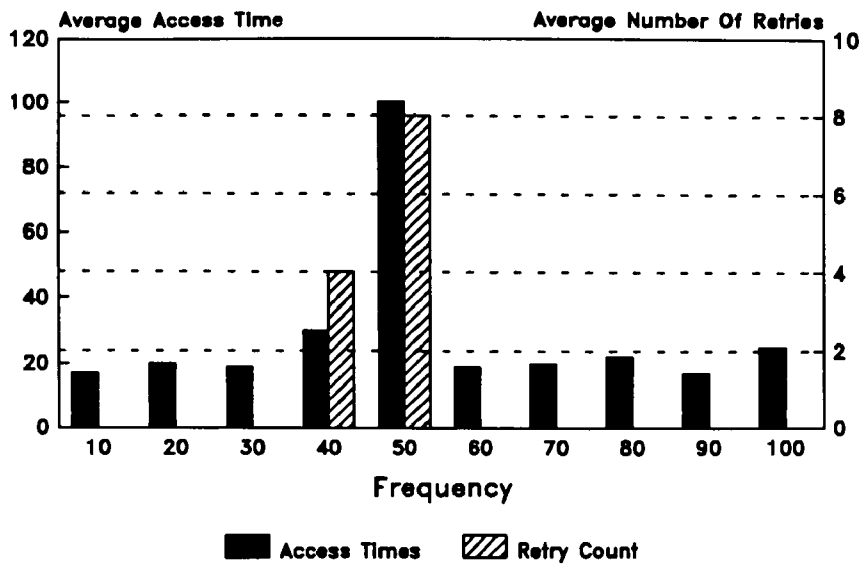


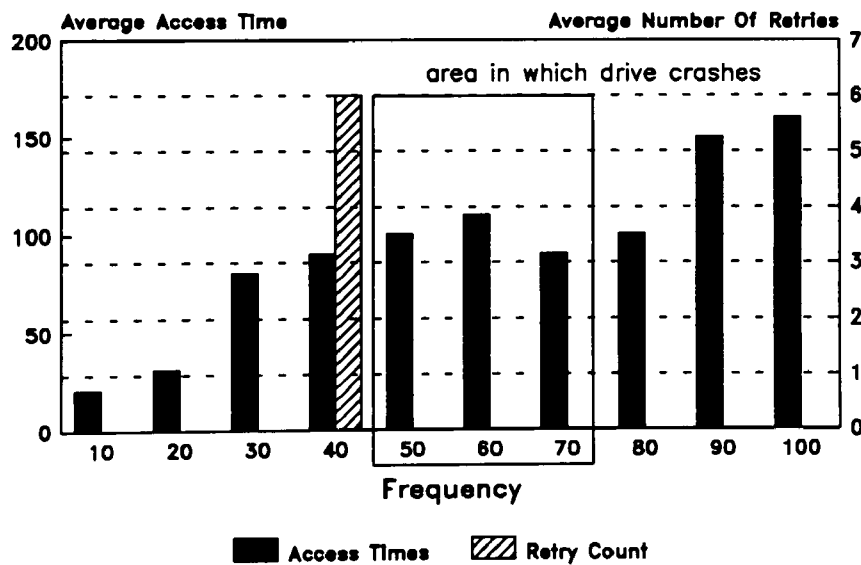
Figure 10.6 : Illustration Of The Effects Of Frequency On Drive At 2g

Outer Data Area

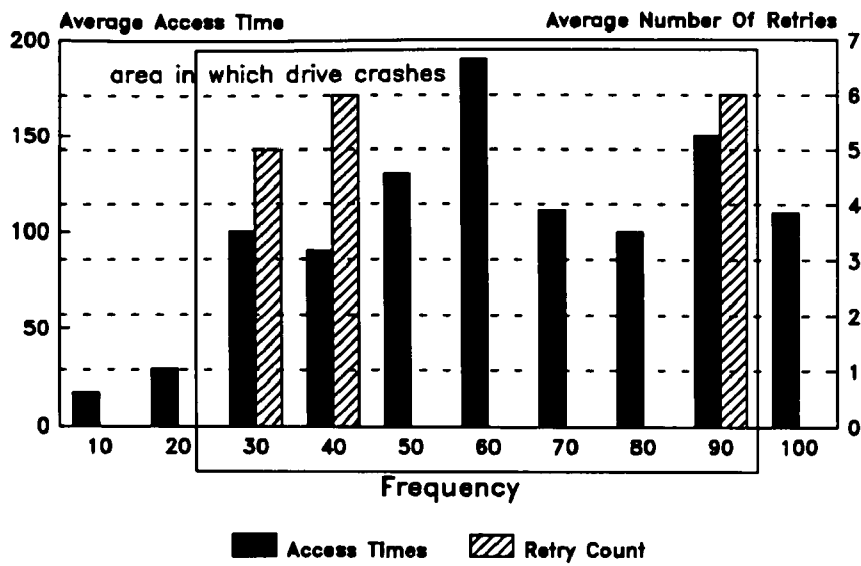
Graphs 10.9 to 10.12 depict the average access times and retry counts for the outer data area.



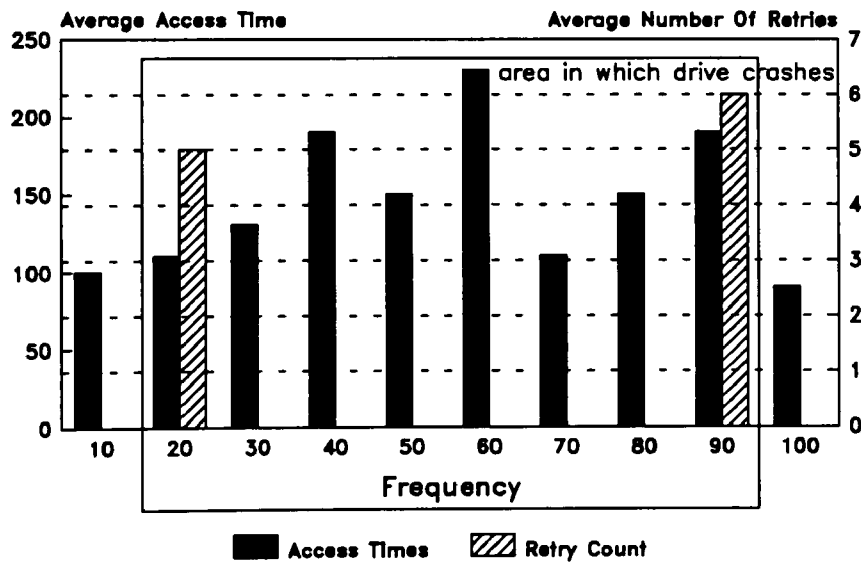
Graph 10.9 : Error Statistics Arising From Fixed Frequency With 0.5g Tests



Graph 10.10 : Error Statistics Arising From Fixed Frequency With 1g Tests






Graph 10.11 : Error Statistics Arising From Fixed Frequency With 1.5g Tests



Graph 10.12 : Error Statistics Arising From Fixed Frequency With 2g Tests

Figures 10.7 to 10.10 illustrate the effect of each area of vibration for each acceleration level upon the drives operation at the outer data area. The same key applies as used in the inner area.

KEY

- normal functioning
-  area of retries
-  high access times
-  drive unable to function (crashes)

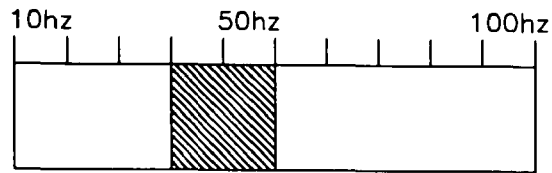


Figure 10.7 : Illustration Of The Effects Of Frequency On Drive At 0.5g

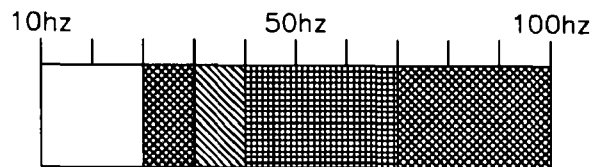


Figure 10.8 : Illustration Of The Effects Of Frequency On Drive At 1g

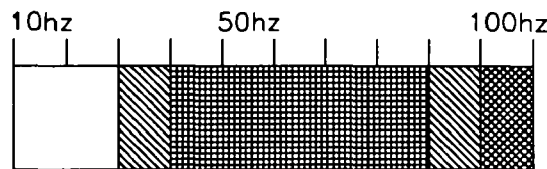


Figure 10.9 : Illustration Of The Effects Of Frequency On Drive At 1.5g

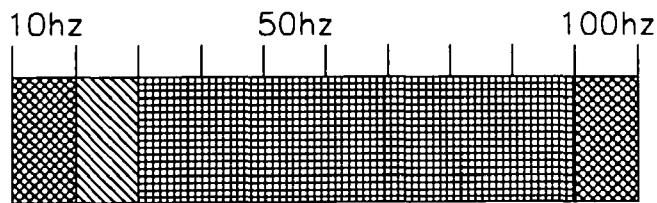


Figure 10.10 : Illustration Of The Effects Of Frequency On Drive At 2g

In the 0.5g test, **Graph 10.5** shows that the drive is adversely affected over the frequency range of 30 to 50 Hz. By comparison **Graph 10.9** shows difficulties occur only between 40 and 50 Hz. At 0.5g the outer area is observed to perform marginally better than the inner.

At the 1g acceleration level both areas experience serious problems which lead to a system crash. However again the outer area can be seen to be marginally more resilient than the inner. In **Graph 10.6** the inner area is observed to fail between the frequencies of 40 and 70 Hz. In **Graph 10.10** this range only occurs between 50 and 70 Hz.

In comparison, the inner area can be seen to perform better at 1.5g. Here the inner areas problem frequencies range from 40 to 80 Hz, whereas the outer area frequencies range from 30 to 90 Hz. At 2g both areas react in a similar manner. Both experience problems in the frequencies ranging from 20 to 90 Hz.

It is apparent from the results that some difference exists between the two areas. However these differences are only marginal. The range of frequencies which are seriously effecting the drive must be attributed to an area of resonance of one or more of the drive components.

(iii) Swept Sine Vibration

The frequency of vibration is constantly increasing, even at the period of seeking. It is therefore not possible to determine the exact frequency of each block. In the event of a crash, the present frequency may be recorded. However this may not be the frequency at which access problems began. An overall picture can be produced for the various maximum acceleration levels. These are illustrated in **Figures 10.11 to 10.14**. Due to the similarity between the inner and outer data area, the latter is not illustrated.

KEY

- normal functioning
- area of retries
- high access times
- drive unable to function (crashes)

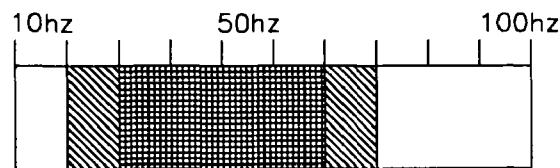


Figure 10.11 : Illustration Of The Effects Of Frequency On Drive At 0.5g

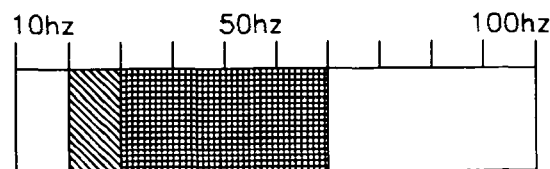


Figure 10.12 : Illustration Of The Effects Of Frequency On Drive At 1g

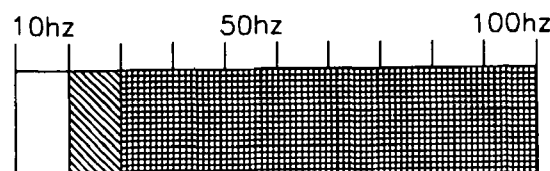


Figure 10.13 : Illustration Of The Effects Of Frequency On Drive At 1.5g

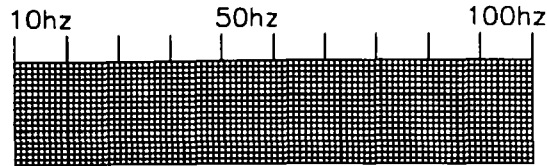


Figure 10.14 : Illustration Of The Effects Of Frequency On Drive At 2g

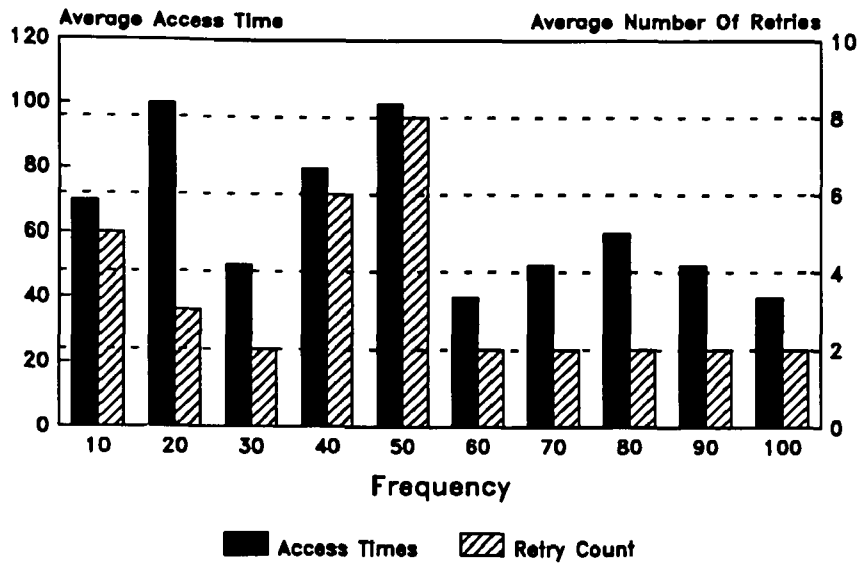
These ranges of frequencies which cause the CDROM system to crash can be compared with those produced by those at the fixed frequency.

In all cases it is apparent that the CDROM system functions far worse under this type of vibration than that of fixed frequency. This is probably due to the greater susceptibility to resonance. Whilst some frequencies only cause accessing problems others cause the system to crash. Those frequencies which only cause access problems will effect sequential blocks increasing those blocks access times. If during the period that a block is being accessed the frequency changes from an 'access problem' frequency to a 'system crash' frequency then that block will not be accessed and lost. However the frequency will be perceived as causing the block loss. It is this effect which causes the apparent widening of the hostile frequency range.

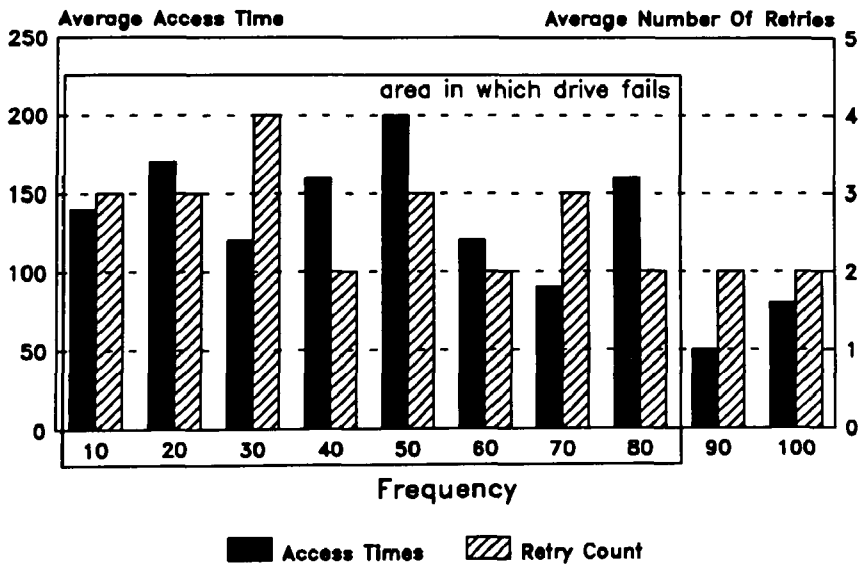
10.4.4 Hostile Environment With Disc B

(i) Constant Vibration

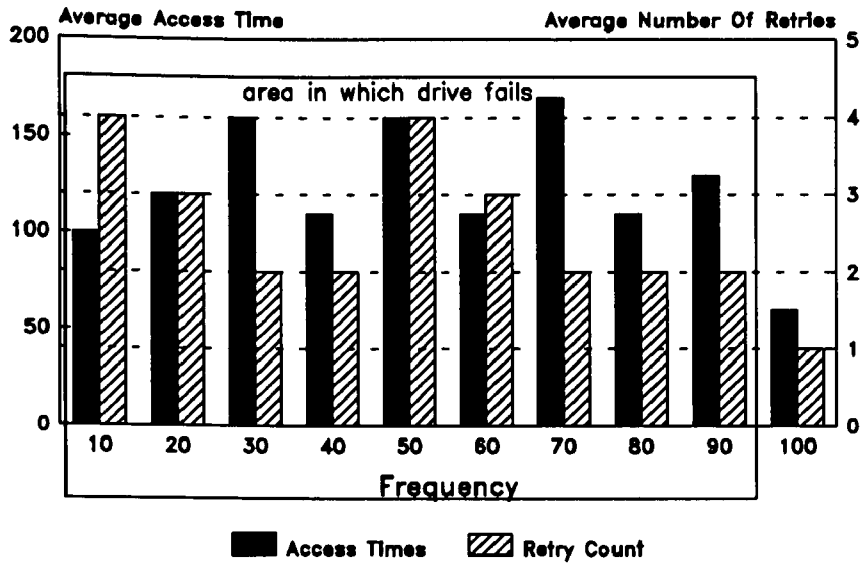
Graphs 10.13 to 10.16 depict the average access times and retry counts which are plotted against frequency for acceleration levels 0.5, 1, 1.5 and 2g respectively.



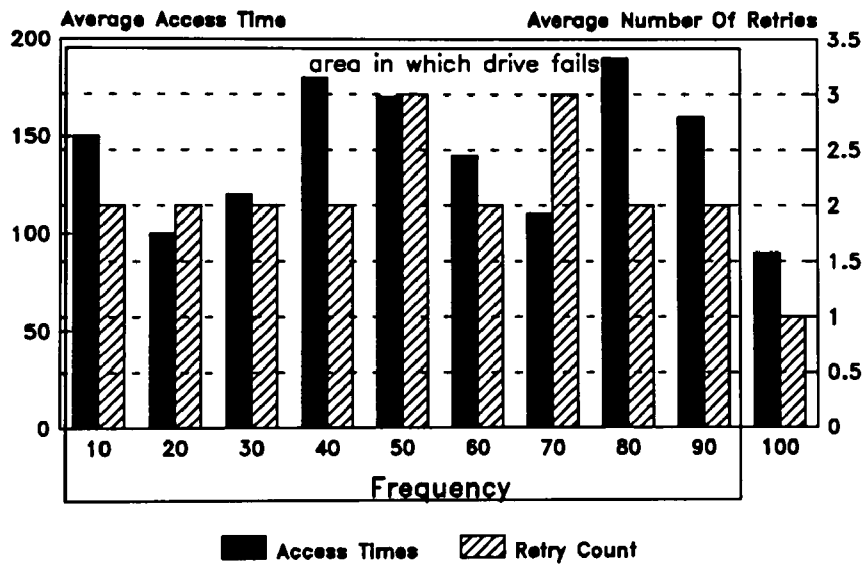
Graph 10.13 : Error Statistics Arising From Fixed Frequency Tests With 0.5g



Graph 10.14 : Error Statistics Arising From Fixed Frequency Tests With 1g



Graph 10.15: Error Statistics Arising From Fixed Frequency Tests With 1.5g



Graph 10.16: Error Statistics Arising From Fixed Frequency Tests With 2g

The region of non-functioning is depicted in the following figures.

KEY

- normal functioning
- ▨ area of retries
- ▩ high access times
- ▧ drive unable to function (crashes)

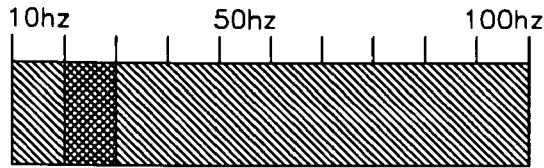


Figure 10.15 : Illustration Of The Effects Of Frequency On Drive At 0.5g

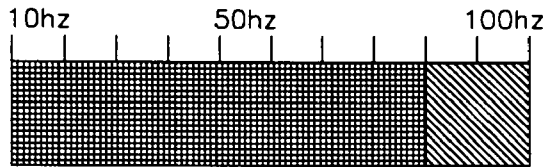


Figure 10.16 : Illustration Of The Effects Of Frequency On Drive At 1g

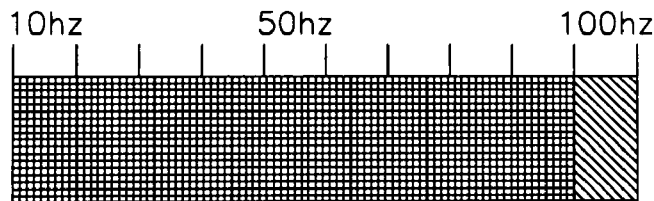


Figure 10.17 : Illustration Of The Effects Of Frequency On Drive At 1.5g

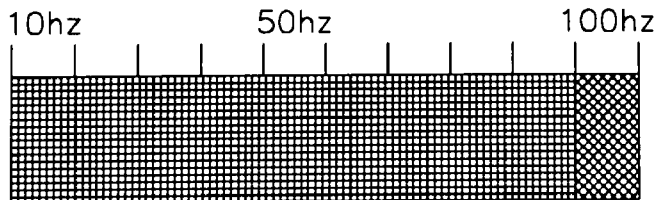


Figure 10.18 : Illustration Of The Effects Of Frequency On Drive At 2g

Here both retries and high access times are far more numerous in comparison to the inner and outer areas of Disk A. Without vibration a low level of retries are expected, due to the surface errors.

The results are similar to those illustrated in **Figures 10.3 to 10.6** and **Figures 10.7 to 10.10**, for the inner and outer area of Disk A. The statistics due to the surface errors are compounded by those caused by the adverse vibrations. The retries and high access times cause the drive system to completely crash within certain frequencies.

(ii) Swept Sine Vibration

KEY

- normal functioning
- ▨ area of retries
- ▩ high access times
- ▧ drive unable to function (crashes)

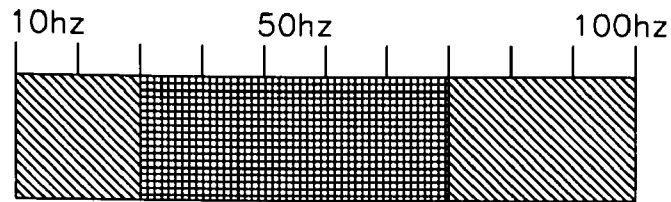


Figure 10.19 : Illustration Of The Effects Of Frequency On Drive At 0.5g

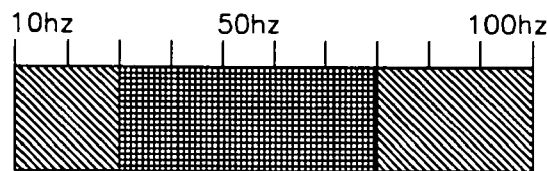


Figure 10.20 : Illustration Of The Effects Of Frequency On Drive At 1g

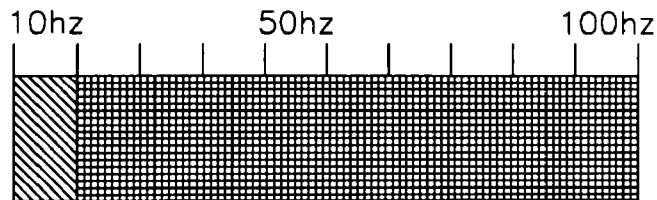


Figure 10.21 : Illustration Of The Effects Of Frequency On Drive At 1.5g

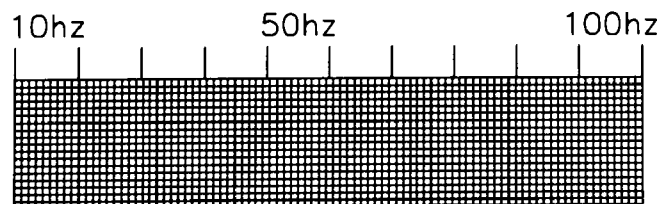


Figure 10.22 : Illustration Of The Effects Of Frequency On Drive At 2g

The Effect of the Swept Sine Frequency on Disc B is illustrated in **Figures 10.19-10.22**. A comparison with **Figures 10.11-10.14** illustrates the differences between the effects on Disc A and B. The results again show how the operational frequency of the drive works decreases in the presence of vibration.

CHAPTER ELEVEN

Conclusions and Future Work

11.1 Conclusions

The CDROM has been discussed and its error control strategies reviewed. Each stage of the encoding/decoding schemes detailed in the Standards (ECMA, BSI, etc) has been investigated. This was necessary to fully develop the simulation model produced at the University of Glamorgan. **Chapter Five** discussed the encoding and **Chapter Six** the decoding. The simulation was described in **Chapter Seven**, an account of which is given in **Appendix A**. Each stage of the decoding is simulated to enable error control to be traced from start to finish.

Note that within existing technology it is impossible to place an error in a fixed prescribed position on the media. This could be achieved only if the error were introduced by the write laser itself, which is clearly impractical. It was shown that the maximum correctable burst size is dependant on the burst position in a Sector. When the burst is at the middle of the Sector, the burst correction limit is approximately 7000. This figure decreases towards the ends. At the limits of performance the synchronisation of a channel Frame is critical.

The simulation model is therefore believed to be the only means by which error on CDROM media may be incorporated and investigated. It is further believed that this novel solution to the problem is of both academic and commercial importance.

In **Chapter Nine** novel software is described which obtained the access times and retries for a given Sector of data. These values are important when determining the performance of the CDROM to adverse environments caused by vibration.

In **Chapter Ten** the software is used to identify the ranges of hostile vibrations within which the CDROM fails. It was established that both error statistics were necessary to describe the systems response to its environment. The ranges of frequency which cause the CDROM to malfunction were established for a given range of maximum acceleration values. At accelerations of 2g there was almost complete system failure, i.e. 2g is the maximum force which the CDROM can sustain. It was established that surface blemishes make the discs less resilient to vibration. It was also shown that the CDROM was not particularly resistant to impulse shock, which displaces the reading laser.

The error control strategy is designed to combat both permanent and transient error mechanisms. The work addresses these two classes of error in complimentary styles. Permanent error mechanisms will produce deterministic errors. It is therefore best investigated by a detailed deterministic simulation of every stage of the decoding. Practical data is not feasible since true error (e.g. a scratch) may not be attributed to any particular Sector.

By their very nature transient error mechanisms are non-deterministic and must be investigated experimentally. Hence the user code software was written to monitor and access retry data of individual blocks. Together the two methods provide a comprehensive analysis of the error performance of the CDROM.

11.2 Future Work

11.2.1 The Simulation Model

(i) Extending The Existing Model

The simulation model has been used to model the effect of errors on the CDROM. However model results were restricted to an independent sector of data. This was due to the run time and computing requirements which would be made by a more general model. Future work could include extending the model to incorporate burst errors across adjacent sectors.

(ii) Multiple Error Application

To date only single burst errors within a Sector have been investigated. It is reasonable to suppose that multiple errors within a Sector will occur. It is of interest to investigate the effect of these errors on the decoding scheme. Preliminary work suggests that a systematic investigation into multiple bursts would be very long and complicated, and would require very extensive computing support. To produce a guide for multiple bursts input would include length and position of all the bursts. This would be a huge database, several times more complex than the single burst database discussed in **Chapter Eight**.

(iii) Compressed Areal Densities

Derivatives of the CDROM will have higher areal densities and greater disc-read head speeds *Nadeau[89], Fox[90]*. It may be presumed that the error correction strategies will largely follow those of the CDROM. A variant of the model may be used to investigate the tighter specifications likely to be encountered.

11.2.2 Future Uses Of Performance Measures

(i) Additional Hostile Environments

In **Chapter Ten**, the performance measures were used in order to investigate the effects of vibrations on the CDROM. This software may also be used for testing the drive with other hostile conditions, such as temperature and humidity. It could be used to measure drive operations near the boundaries of the values outlined in the standards *ECMA[18]*.

The platform vibration is monitored independent of retries and access time data. More information could be obtained if the vibration information was integrated with the access time and retry data. In this way the ambient conditions for each block can be associated precisely for the performance data.

(ii) Conformance To Standards

It was shown in **Chapter Ten** that disc blemishes reduce the robustness against vibration. Although discs are manufactured to a standard, the conformance to these standards may vary from manufacturer to manufacturer. At the margins of the conformity one may suppose that the error correction is degraded. It would be most useful to explore and quantify the relationship between the error correction capability and conformance with standards.

11.2.3 Using The ASPI Program

The ASPI template used for the CDROM may be adapted for RDAT-DDS. Equally it could be applied to the CDROM derivatives.

References

- [1] E.W.Sponheimer and J.C.Santon.
"A CD-ROM Drive For HP 3000 and HP 9000 Computer Systems".
Hewlett Packard Journal. December 1990. pages 38-41.
- [2] A.E.Bell and V.Marrello.
"Magnetic and Optical Data Storage : A Comparison Of The Technological Limits". IBM San Jose Research Lab. Proceedings IEEE COMPCON. 1984. pages 512-517.
- [3] A.E.Bell.
"Critical Issues In High Density Magnetic and Optical Storage". Proceedings of SPIE. 1983. Volume 382. pages 2-15.
- [4] R.Wood.
"Magnetic and Optical Storage Systems : Opportunities For Communications Technology". IEEE International Conference On Communications. 1989. Vol 3. Pages 1605-1612.
- [5] L.Laub.
"The Evolution Of Mass Storage". BYTE Magazine. May 86. pages 161-175.
- [6] B.Zoellick.
"CDROM Software Development". BYTE Magazine. May 86. pages 177-188.
- [7] R.C.Alford.
"CD-ROM Inside and Out". BYTE. March 1993. pages 197-206.
- [8] S.Christodoulakis and D.A.Ford.
"File Organisation and Access Methods For CLV Optical Disks". ACM SIGIR Forum 23 (1+2). 1989. pages 152-159.
- [9] P.A.Dare and T.Katsumi.
"Rotating Digital Audio Tape (RDAT) : A Format Overview". SMPTE (Society Of Motion Picture and Television Engineers, Incorp.) Journal. October 1987. pages 943-948.
- [10] R.A.Baugh, D.J.Bromley and B.F.Spencer.
"Extremely Low Error Rate Digital Recording With A Helical Scan Recorder". Colloquium On Mass Storage Devices For Computers. IEE. October 1986. pages 1-3.

- [11] J.Watkinson.
"The Art Of Digital Audio". Focal Press. First Edition. ISBN 0-240-51270-7.
1988.
- [12] K.Odaka, T.Tan and B.Vermeulen.
"Designing A Data Storage Format For Digital Audio Tape (DAT)". DDS
Manufacturing Group. October 1988.
- [13] S.Lambert and S.Ropiequet.
"CDROM - The New Papyrus". Microsoft Press. First Edition. ISBN 0-
914845-74-8. 1988.
- [14] W.Poel and N.Barton.
"The CDROM Story". Computer Shopper. October 1989. pages 85-88.
- [15] M.G.Carasso, J.B.H.PEEK and J.P.Sinjou.
"The Compact Disc Digital Audio System". Philips Technical Review.
Volume 40. Number 6. 1982. pages 151-156.
- [16] D.Goedhart, R.Van de Plassche and E.Stikvoort.
"Digital-To-Analog Conversion In Playing A Compact Disc". Philips
Technical Review. Volume 40. Number 6. 1982. pages 174-179.
- [17] L.B.Vries.
"The Error Control System Of The Philips Compact Disc". Presented at the
64th AES Convention. November 2-5, 1979. AES. 1548 (G-8). pages 1-9.
- [18] ECMA (European Company Manufacturers Association).
"ECMA-130 : Data Interchange On 120mm Optical Data Disks (CD-ROM)".
July. 1988.
- [19] British Standards Institution.
"British Standard Specifications For The Compact Disc Digital Audio
System". BS 7064 : 1989. IEC 908 : 1987.
- [20] ECMA (European Computer Manufacturers Association).
"ECMA-119 : Volume and File Structure Of CDROM For Information
Interchange". Second Edition. December 1987.
- [21] W.Verkaik.
"Compact Disc Mastering - An Industrial Process". The AES conference -
The New World of Digital Audio. New York. June. 1982. part 3-6.

- [22] J.R.Watkinson.
"Principles Of Optical Storage". Electronics and Wireless World. March 1986. pages 70-72.
- [23] J.R.Watkinson.
"CD : The 600 Megabyte ROM". Electronics and Wireless World. June 1987. pages 1046-1049.
- [24] S.Christodoulakis.
"Analysis Of Retrieval Performance For Records and Objects Using Optical Disk Technology". ACM Transactions On Database Systems. Volume 12. Number 2. June 1987. pages 137-169.
- [25] E.F.Barbosa and N.Ziviani.
"Data Structures and Access Methods For Read-Only Optical Disks". 11th International Conference Of Chilean Science Society. 15-18 October 1992. pages 189-207.
- [26] S.Miyaoka.
"Digital Audio is Compact and Rugged". IEEE Spectrum. March 1984. pages 35-39.
- [27] S.Christodoulakis and D.A.Ford.
"Performance Analysis and Fundamental Performance Trade Offs For CLV Optical Disks". ACM SIGMOD Proceedings. June 1988. pages 286-294.
- [28] D.H.Davies.
"The CD-ROM Medium". Journal Of The American Society For Information Science. Volume 39. Number 1. January 1988. ISSN:0002-8231. pages 34-42.
- [29] K.A.S.Immink.
"Coding Methods For High Density Optical Recording". Philips J. Re. Volume 41. September 1986. pages 410-431.
- [30] J.B.H.Peek.
"Communications Aspects Of The Compact Disc Digital Audio System". IEEE Communications Magazine. Volume 23. Number 2. February 1985. pages 7-15.
- [31] J.R.Watkinson.
"Principles Of Optical Storage - 2". Electronics and Wireless World. April 1985. pages 43-46.

- [32] H.Hoeve, J.Timmermans and L.B.Vries.
"Error Correction and Concealment In The Compact Disc System". Philips Technical Review. 40. 1982. Number 6. pages 166-172.
- [33] L.B.Vries, K.A.S.Immink, J.G.Nijboer, H.Hoeve, T.T.Doi, K.Odaka and H.Ogawa.
"The Compact Disc Digital Audio System : Modulation and Error Correction". 67th AES Convention. 1980. October. (H-8). pages 1-14.
- [34] R.Cardinali.
"Optical Storage: Future Educational Impact". Journal of Educational Technology Systems. Volume 19. Part 3. 1991. pages 181-190.
- [35] Y.Sako and T.Suzuki.
"Data Structure Of The Compact Disk - Read - Only Memory System". Applied Optics. Volume 25. Number 22. November 1986. pages 3996-4000.
- [36] P.P.Chen.
"The Compact Disc ROM : How It Works". IEEE Spectrum. April 1986. pages 44-49.
- [37] G.D.Forney.
"Burst Correcting Codes For The Classic Bursty Channel". IEEE Transactions On Communications. Volume COM-19. Number 15. October 1971. pages 772-781.
- [38] C.E.Shannon.
"A Mathematical Theory of Communication". Bell Systems Technical Journal. Volume 27. Number 3. July 1948. pages 379-423.
- [39] C.E.Shannon.
"A Mathematical Theory of Communication (Conclusions Of July 1948 Issue)". Bell Systems Technical Journal. Volume 27. Number 4. 1948. pages 623-656.
- [40] E.R.Berlekamp.
"The Technology Of Error Correcting Codes". Proceedings Of The IEEE. Volume 68. Number 5. May 1980. pages 564-593.
- [41] J.R.Watkinson.
"Data Error Detection and Correction". Wireless World. February 1983. pages 44-48.

- [42] T.T Doi.
"Error Correction For Digital Audio Recordings". Digital Audio. AES Conference. 1982. June. pages 147-177.
- [43] R.W.Hamming.
"Error Detecting and Error Correcting Codes". Bell System Technical Journal. Volume 26. Number 2. April 1950. pages 147-160.
- [44] E.R.Berlekamp.
"Error Correcting Codes For Digital Audio". Digital Audio. AES Conference. June. 1982. pages 127-138.
- [45] S.W.Golomb.
"Optical Disk Error Correction". BYTE Magazine. May 86. pages 203-210.
- [46] I.S.Reed and G.S.Solomon.
"Polynomial Codes Over Certain Finite Fields". Journal Of The Society Of Industrial and Applied Mathematics. Volume 8. 1960. pages 300-304.
- [47] P.Sweeny.
"Error Control Coding : An Introduction". Prentice Hall. ISBN 0-13-284118-5.
- [48] R.J.McEliece and L.Swanson.
"On The Decoder Error Probability For Reed Solomon Codes". IEEE Transactions On Information Theory. Volume IT-32. Number. 5. September 1986. pages 701-703.
- [49] J.L.Ramsey
"Realisation Of Optimum Interleavers". IEEE Transactions On Information Theory. Volume IT-16. Number 3. May 1970. pages 338-345.
- [50] A.J.Verterbi.
"Coding and Interleaving For Correcting Burst and Random Errors In Recording Media". Digital Audio. AES Conference. June. 1982. pages 139-146.
- [51] K.A.S.Immink.
"Coding Techniques For Digital Recorders". 1991. ISBN 0-13-140047-9.

- [52] K.A.S.Immink.
"Modulation Systems For Digital Audio Discs With Optical Readout". IEEE International Conference On Acoustic Speech and Signal Processing. March. 1981. pages 587-589.
- [53] J.C.Mallinson and J.W.Miller.
"Optimal Codes For Digital Magnetic Recording". Radio and Electronic Engineer. Volume 47. April 1977. pages 172-176.
- [54] N.D.Mackintosh.
"The Choice Of A Recording Code". The Radio and Electronic Engineer. Volume 50. April. 1980. pp 177-193.
- [55] H.Ogawa and K.A.S.Immink.
"EFM - The Modulation Method For The Compact Disc Digital Audio System". Digital Audio. AES. Collected Papers from the AES Premiere Conference, New York. June 3-6. 1982. pages 117-124.
- [56] J.P.J.Heemskerk and K.A.S.Immink.
"Compact Disc : Systems Aspects and Modulation". Philips Tech Review. Volume 40. Number 6. 1982. pages 157-164.
- [57] D.T.Tang and L.K.Bahl.
"Block Codes For A Class Of Constrained Noiseless Channels". Information and Control. 17:436. 1970. pages 200-210.
- [58] J.R.Watkinson.
"Channel Code and Disc Format - 1 ". Electronics and Wireless World. May 1985. pages 27-28.
- [59] A.M.Patel.
"Zero-Modulation Encoding In Magnetic Recording". IBM J. Res. Dev. Volume 19. 1975. pages 366-78.
- [60] S.Fukuda, Y.Kojima, Y.Shimpuku, K.Odaka.
"8/10 Modulation Codes For Digital Magnetic Recording". IEEE Transactions On Magnetics. Volume MAG-22. Number 5. September 1986. pages 1194-1196.
- [61] J.R.Watkinson.
"Channel Code and Disc Format - 2 ". Electronics and Wireless World. June 1985. pages 80-82.

- [62] L.B.Vries and K.Odaka.
"CIRC - The Error Correcting Code For The Compact Disc Digital Audio System". Digital Audio. AES. Collected Papers from the AES Premiere Conference. New York. June 3-6. 1982. pages 178-186.
- [63] L.M.Driessen and L.B.Vries.
"Performance Calculations Of The Compact Disc Error Correcting Code On A Memoryless Channel". Proceedings Of 4th IERE Conference On Video and Data Recording. Part 54. 1982. pages 385-395.
- [64] T.T.Doi, K.Odaka, G.Fukuda and S.Furukawa.
"Cross Interleaving Code For Error Correction Of Digital Audio Systems". Presented At The 64th Convention Of The Audio Engineering Society. Journal Of the Audio Engineering Society. Volume 27. Part 1559 (H-4). November 2-5. 1979. pages 1-4.
- [65] J.R.Watkinson.
"Subcodes Explained". Electronics and Wireless World. September 1986. pages 26-30.
- [66] T.T.Doi.
"Channel Codings For Digital Audio Recording". Presented At The 70th Convention Of The Audio Engineering Society. Journal Of The Audio Engineering Society. Volume 31. Number 4. April 1983. pages 224-238.
- [67] K.A.S.Immink and U.Gross.
"Optimization Of Low Frequency Properties Of Eight-To-Fourteen Modulation". The Radio and Electronic Engineer. Volume 53. Number 2. February 1983. pages 63-66.
- [68] S.W.Golomb, J.R.Davey and I.S.Reed.
"Synchronisation". IEEE Transactions On Communication Systems. Volume CS-11. December 1963. pages 481-491.
- [69] E.N.Gilbert.
"Synchronisation Of Binary Messages". IRE Transactions On Information Theory. Volume IT-6. September 1960. pages 470-477.

- [70] J.D.Ullman.
"On The Capability Of Codes To Correct Synchronisation Errors". IEEE Transactions On Information Theory. Volume IT-13. Number 1. January 1967. pages 95-105.
- [71] V.I.Levenshtein.
"Binary Codes Capable Of Correcting Deletions, Insertions and Reversals". Soviet Physics.-Doklady. Volume 10. Number 8. February 1966. pages 707-710.
- [72] F.P.Preparata.
"Systematic Construction Of Optimal Recurrent Codes For Burst Error Correction". Calcolo (PISA). 1964. pages 147-153.
- [73] L.Helgerson.
"Detecting and Correcting Errors On CDROM". CD Data Report. April 1985. pages 405-409.
- [74] B.L.Johnson.
"Design and Hardware Implementation Of A Versatile Transform Decoder For Reed Solomon Codes". Impact Of Processing Techniques On Communications. 1985. pages 447-464.
- [75] T.Arai, H.Okamoto, K.Nishimura, M.Kobayashi and T.Takeuchi.
"High Capability Error Correction LSI For CD Player and CD-ROM". IEEE Transactions On Consumer Electronics. Volume CE-30. Number 3. August 1984. pages 353-359.
- [76] C.C.Ko and T.T.Tjhung.
"Comparison Of Simple Cross Interleaved Reed Solomon Decoding Strategies For Compact Disc Players". TENCON 1987. pages 378-382.
- [77] C.C.Ko and T.T.Tjhung.
"Simple Programmable Processor For Decoding Reed Solomon Codes In Compact Disc Devices at High Speed". Int Journal of Electronics. 1989. Volume 67. Number 1. pages 15-25.
- [78] W.Peterson and E.J.Weldon.
"Error Correcting Codes". MIT Press. 1972. ISBN 0-262-16-039-0. pages 283-299.

- [79] A.M.Patel.
"On-The-Fly-Decoder For Multiple Byte Errors". IBM Journal Of Research and Development. 30. 1986. pages 259-269.
- [80] N.N.Heise and T.J.Krzystan.
"Direct Hardware Solution To The Quadratic Equation $y^2+y+c=0$ in $GF(2^m)$ ". IBM Technical Disclosure Bulletin, 27. 1986. pages 4767-4771.
- [81] S.I.Woolley, B.K.Middlton, A.Ryley, M.E.Morey, W.N.Carrick and R.Saunders.
"Error Statistics of Magnetic Recording In Severe Environments". International Conference On Storage and Recording SysteMS 1994. IEE. 5-7 April 1994. pages 98-102.
- [82] S.I.Woolley.
"Error Statistics and Data Compression in Digital Instrumentation Recording Systems". PhD Thesis. 1994.
- [83] NCR Corporation.
"Understanding The Small Computer System Interface (SCSI)". Prentice Hall. 1988. ISBN 0-13-796855-8.
- [84] Adaptec.
"Advanced SCSI Programming Interface (ASPI), DOS Specifications". May. 1990.
- [85] Hitachi.
"Interface Specification For CD-ROM Drive : Model CDR - 6750". Hitachi Ltd. 1994. ISE-0250-00.
- [86] SONY.
"Theory Of Operations Of A CDROM Drive Unit". Revision 1.20. December 1989.
- [87] Philips Laser Magnetic Storage International Company : Optical Storage Division.
"Product Specifictation : CDROM Drive SCSI". Feb 1989.
- [88] Hitatchi.
"Device Specifications". Hitachi Drive Information. Product Release. NM-E128. 1995. page 1193.

- [89] M.Nadeau.
"Two Ways To Cram More Data On A CDROM". BYTE Magazine.
September. 1994. page 34.
- [90] B.Fox.
"CDs : The Next Generation". New Scientist. 10 September. 1994. pages 33-
35.

Bibliography

- [1] C.Delannoy.
"Turbo Pascal Programming". Macmillan Education Ltd. First Edition. ISBN 0-333-48426-6.
- [2] L.Miller, Q.Quilici.
"The Turbo C Survival Guide". Wiley. First Edition. ISBN 0-471-61708-3.
- [3] Borland.
"Turbo Assembler 3.0 : User Guide". Borland International. 1991
- [4] Borland.
"Turbo C++ 3.0 : User Guide". Borland International. 1992.
- [5] S.Smith.
"The Waite Group's MS-DOS Bible". H.W.Sams & Company. Second Edition. 1988. ISBN 0-672-22617-0.

Appendix A

Using The Simulation Model

The Simulation model is composed of both the encoding and decoding strategies of the CDROM. Both schemes are composed of a number of program modules. Each program Module uses the output of the previous module as its input, manipulates the data in a predefined manner and outputs the information to another data file.

The complexity of each process and storage constraints creates a need for individual program modules rather than one large complete program. This proves to be more useful when illustrating the progression of errors at decoding.

Appendix A.1 ; Encoding Programs

The programs associated with encoding are:

synch.pas	Add twelve byte synchronisation.
crc.pas	Calculate CRC and add to data.
intermed.pas	Add intermediary redundant bytes.
ecc_p.pas	Calculate and add P parity.
ecc_q.pas	Calculate and add Q parity.
scramble.pas	Scrambles the data.
switch.pas	Reorder Consecutive bytes.
delay1.pas	First stage interleave of CIRC.
rsc_q.pas	C2 RSC(28,24) Encoding.
delay2.pas	Second stage interleave of CIRC.
rsc_p.pas	C1 RSC(32,28) Encoding.
delay3.pas	Third stage interleave of CIRC.
control.pas	Add control data.
mod2.pas	Modulate Data. EFM conversion and DSV minimisation using merge bits.

Each programs is independant, however an inter-relationship is governed by the data file produced by each. These programs must therefore be executed in sequence. The software suite may be run transparently using the batch file **encode.bat**. Here it is only necessary to construct the desired raw data file, all encoding runs automatically. The raw data to be encoded is placed in **tdata.txt**, this consists of 2048 bytes.

The result of the whole encoding scheme is the channel data produced in **output2.txt**. Again an example file is included.

Data files are necessary to enable correct performance of the Reed Solomon parity calculations used in the four encoders.

Associated Files

P-Parity Equation Files:

P_1.DAT P_2.DAT

Q-Parity Equation Files:

Q_2.DAT Q_1.DAT

C1-Parity Equation Files:

P1.DAT P2.DAT P3.DAT P4.DAT

C2-Parity Equation Files:

Q1.DAT Q2.DAT Q3.DAT Q4.DAT

additional files required :

codes.dat (EFM conversion table)

table.dat (Galois Field Elements)

input file : TDATA.TXT

output file : OUTPUT2.TXT

execution file : ENCODE.BAT

Appendix A.2 : Decoding Programs

The following files are associated with the decoding simulation:

demodulate.pas	decontro.pas	undel_3.pas	p_decode.pas
undel_2.pas	q_decode.pas	undel_1.pas	switch.pas
descram.pas	q_dec.pas	p_dec.pas	rem_inte.pas
crc_check.pas	rem_synch.pas		

These files must also be used in sequence and may be run transparently using **decode.bat**. Error incorporation occur before decoding and will affect the data placed in **output2.txt**. This is discussed in **Appendix A.3**. The error affected data is placed in **output4.txt** and this is decoded.

Decoding is constrained as the inverse of the encoding process. The data is manipulated in order to reverse those schemes used in the encoding scheme. The result of the decoding scheme is the data file **xdata.dat**. The integrity of this data is checked using the CRC present in the code. For our purposes this data may be compared with the raw data used for encoding.

additional files required :

codes.dat (EFM conversion table)

table.dat (Galois Field Elements)

(identical to those used with the encoding files)

input file : OUTPUT4.TXT

output file : XDATA.TXT

execution file : DECODE.BAT

Appendix A.3 : Error Incorporation Programs

The following files are associated with the production of errors upon the channel:

burst_er.pas	bur_err2.pas	b2_errf.pas
b2_err.pas	rand_err.pas	

Each of these incorporates errors into the channel data in **output2.txt**, producing a hybrid data file **output4.txt**. The former four files are all associated with the incorporation of a burst into the data. The latter simulates random errors.

A number of burst program modules are necessary in order to simulate bursts which are both fixed and random and those causing Frame errors. A Frame error is caused by synchronisation loss due to one or more bits of the twelve bit channel synchronisation field. Here the burst propagates as illustrated in **Chapter Seven** to the end of that Frame.

Burst_er.pas simulates a randomly positioned burst of a specified magnitude in bits. Here a seed must be input to ensure the randomness of the burst position. The module does not cause Frame loss when synchronisation is lost. Frame overlap does not cause the discussed burst propagation.

Bur_err2.pas again does not simulate synchronisation loss. The module is not random, here the Frame and bit position are required as input in addition to burst size. Frame position can vary between 1 and 98, however it is recommended that care is taken not to overlap bursts with the end of the Sector. A burst of uncorrectable size would be perceived as correctable if it were to affect Frame 98 since it would be addressed as a one Frame error. Bit position within a Frame can vary between 1 and 588 bits.

B2_errf.pas simulates Frame loss by bursts overlapping the synchronisation bits. Burst position and length are required as an input.

B2_err.pas simulates Frame loss by bursts overlapping synchronisation. The burst size and random location seed are required as

input.

Rand_err.pas simulates random errors in the data channel. Here a probability of error is required plus a seed to ensure varying random bit patterns. Frame loss is produced by synchronisation bit loss.

The program modules used for the results in this thesis were **b2_errf.pas** and **rand_err.pas**.

input file : OUTPUT2.TXT

output file : OUTPUT4.TXT

execution file : BURST.BAT

In addition to this an interactive menu driven program has been produced. This facilitates fully transparent incorporation of burst and random errors into the channel data and runs the decoding simulation.

MENU : error incorporation programs

F_BUR1.PAS R_BUR1.PAS R_BUR2.PAS

RAND2_ER.PAS

F_BUR1.PAS produces a fixed burst at a fixed position.

R_BUR1.PAS produces a random burst at a random position.

R_BUR2.PAS produces a random burst at a random position.

However this is associated with the incorporation with multiple burst incorporation.

RAND2_ER.PAS produces a random error.

associated programs

COPY6_4.PAS COPYINIT.PAS

COPY4_2.PAS **FIX_SYNC.PAS**

all decoding program modules

COPY6_4.PAS and **COPY4_2.PAS** are associated with multiple error incorporation. These copy the error file **output4.txt** to **output2.txt** so that further errors can be incorporated.

COPYINIT.PAS initialises **output2.txt** to original data after multiple errors have been utilised.

FIX_SYNC.PAS causes Frame errors when synchronisation bits have been lost. All error incorporation programs exhibited here do not examine synchronisation loss.

Appendix A.4 : Illustration Programs

It is possible to check the integrity of final data produced by the decoding simulation with the original raw data. By this method it is possible to identify errors within the data. Since both the Decoding and Encoding simulations produce intermediary data files the same approach may be used to follow the progression and correction of errors through each decoding stage.

There are three types of schemes used in order to illustrate these errors. All programs locate errors by inconsistencies between corresponding encoding and decoding files. For example, the data file produced prior to C1 encoding and that produced after C1 decoding.

Since this software is dependant upon data file discrepancies both the encoding and decoding simulations must be run and the intermediary files produced.

A.4.1 Illustration Scheme 1

This illustratory software locates an error due to data inconsistencies and outputs the Frame and byte position. This allows close scrutiny of errors present in the data block. The performance of the error correcting codes can be demonstrated by the removal of errors within a Frame.

The programs are :

COMPMOD2.PAS	COMPCON.PAS	COMPD3DA.PAS
COMPC1.PAS	COMPDEL2.PAS	COMPD2DA.PAS
COMPD1.PAS	COMPSCR.PAS	COMPSW.PAS
COMPP_DE.PAS	COMPINTE.PAS	

associated data : all encode/decode data files

execution file : test.bat

A.4.2 Illustration Scheme 2

By the use of a bi-level grey scale it is possible to illustrate the actual position of persisting errors within the code block. The first scheme found errors and output the Frame and byte position of that error. This scheme illustrates the changing codeblock and maps the progressing errors. An ASCII text output is produced for each decoding stage. All output is all piped into a text file using a batch file. This system was used to produce the illustrations used in **Chapter Seven**.

The programs are :

CINTE.PAS	CMOD3.PAS	CPCON.PAS
CD3DA.PAS	CC1.PAS	CDEL2.PAS
CD2DA.PAS	CD1.PAS	CSW.PAS
CSCR.PAS	CQ_DEC.PAS	CAQ_DEC.PAS
CAQ2_DEC.PAS	CINTE.PAS	

associated data : all encode/decode dat files

execution file : ill.bat

output file : test.txt

A.4.3 Illustration Scheme 3

The progression of errors are followed in the same manner as Scheme 2. However a graphical programme is used to illustrate this progression upon the PC screen. The screen shots can be seen in the following pages.

associated data : all encode/decode files

execution file : illustra.exe

A.5 Inferencing Software and Database

The database was produced with a number of associated programs which run the decoding and burst incorporation software for a range of bursts at varying positions. An example of the type of platform used is **plat1.pas**.

A number of these programmes were run on a group of PC's over a number of weeks to provide the database **dataf.dat**. Data has been incorporated in a readable form for ease of use.

The database is used in conjunction with **infer.pas** which is used to inference bursts as shown in **Chapter Eight**.

Appendix B

Drive Communications Issues

In **Chapter Nine** the CDROM was used in order to obtain statistics which reflect the performance of a drive with a given disk in a specified environment. This was demonstrated to be of significant use in **Chapter Ten**. Here the drive is used with both hostile vibration environments and a disc with a surface error.

In this **Appendix**, complete listings are given of the main program and associated assembly modules used in order to facilitate communications. Each program module is fully commented. More information on DOS interrupts may be found in most related texts, e.g. **Waites MSDOS Bible** (bibliography).

B.1 Assembly Module 1 : OPEN.ASM

```
*****  
;  
;  
; Filename:   open.asm (version 6.c)  
;           (version 6.c replaces 6.b and 6.a which were beta test versions)  
;  
;  
; Description: Initialises ASPI  
;               open aspi;  
;               gets SCSI manager address  
;               closes aspi  
;               places SCSI manager address in globally uable  
variable  
;  
;  
; Date:       Final version : 25 September 1992  
;  
;  
*****  
;
```

.MODEL small

```
*****  
;  
; Specifies the archetechure model small, medium or large  
;  
;  
; This allows specification of the size of segmentation model  
; to be used for program  
;  
*****
```

.DATA

```
*****  
;  
;  
; sets up data area segment  
;  
*****
```

EXTRN aspi_entry_point:DWORD

```
*****  
;  
;  
; Declare aspi_entry_point as an external variable  
; external variables are defined outside a module  
; i.e aspi_entry_point is specified in the main C program.  
;  
;  
; Aspi_entry_point which is a double word i.e. 16 bits.;  
;  
*****
```

public scsimgr

```

,*****
;
;   scsimgr is a public variable
;   this is a variable which may be used by other modules
;
,*****

scsimgr      db 'SCSIMGR$',0
error        dw      ?
msg          db  "Working!", 13, 10, "$"

,*****
;
; scsimgr is a byte of characters 'SCSIMGR$' this string will be
; looked for in dos memory to find the entry point
;
; error is a word of data which will be returned by this program module
;
; error = 1 is an error caused by no 'SCSIMGR$' string being found in
memory
; error = 0 is no error caused by the 'SCSIMGR$' string being found in
memory
;
,*****

init_message db  13,10,'Initialising',13,10
no_aspi_message db  'ASPI device driver not loaded',13,10,13,10,0
aspi_message db  'ASPI device driver was loaded',13,10,13,10,0

,*****
;

```

```
; messages to be used if aspi_message found
;
,*****
```

```
.CODE
```

```
,*****
;
; start of code segment
;
,*****
```

```
PUBLIC _open_aspi
```

```
,*****
; open aspi FUNCTION
;     since it is defined as public this function
;     can be used by other programs or modules
;
; function call used by main program
; opens the aspi manager
;
,*****
```

```
_open_aspi proc near
```

```
    push    bp
    push    si
    push    di
    push    dx
    push    cx
    push    bx
```

```

,*****
;
; push current register values onto stacks
;
,*****

    mov    ax,03d00h

; place open file command in ax for interrupt call

    mov    dx,OFFSET scsimgr
    int    21h

; call interrupt with variable settings

    jnc    open_ok

,*****
;
; look for SCSI string
; if found jump to open_ok
; else continue
;
,*****

    mov    ax,0
    mov    bx,OFFSET no_aspi_message

; no ASPI found message

    call   display_string
    mov    error,1

```

```
; message not found
```

```
    jmp    open_exit
```

```
,*****
```

```
;
```

```
; jump to open exit if no message found
```

```
; leave assembler module
```

```
;
```

```
,*****
```

```
open_ok:
```

```
    mov    bx,OFFSET aspi_message
```

```
; load aspi found message
```

```
    mov    error,0
```

```
    call   display_string
```

```
,*****
```

```
;
```

```
; display string
```

```
; since aspi found continue along program path
```

```
,*****
```

```
    mov    bx,ax
```

```
    mov    ax,4402h
```

```
    lea   dx,aspi_entry_point
```

```
    mov    cx,4
```

```
    int   21h
```

```
; find aspi entry point and place in variable
```

```
    mov    ax,1
```

```

open_exit:
    mov    ax,error
; place error value into variable
    pop   bx
    pop   cx
    pop   dx
    pop   di
    pop   si
    pop   bp

;*****
;
; pop current register values off stacks
;
;*****
ret
_open_aspi endp

```

```

;*****
; end of function
;*****

```

```
display_string proc near
```

```

;*****
;
; display_string function
;         displays strings : outputs charactres to screen
;
;*****

```



```

    push    ax
    push    bx

;*****
;
; push current register values onto stacks
;
;*****

next_char:
    mov     al,byte ptr [bx]
    cmp     al,0
    jz      string_done
; if no char left then jump to string_done
    push    bx
    mov     ah,0eh
    mov     bx,0
    int     10h
    pop     bx
    inc     bx
    jmp     next_char
; output string character by character.

string_done:
    pop     bx
    pop     ax

;*****
;
; push current register values off stacks
;
;*****

```

```
ret
display_string endp
; end of function
END
```

```
.*****
,
;
; end of program module
;
.*****
,
```

B.2 Assembly Module 2 : SEND_DATA.ASM

```
*****  
;  
;  
; Filename : send_DATA.asm (version 5.c)  
;  
;  
; Description : uses aspi entry point produced in open.asm  
;               enables Pc <-> SCSI communication  
;  
;  
; Date : September 24 1995.  
;  
;  
*****  
  
    .MODEL small  
  
; define model type  
  
    .DATA  
;  
; start of data segment  
  
public    aspi_entry_point  
public    _SRB_ptr  
public    _data_ptr  
  
;  
; aspi_entry_point : aspi entry point received from open.asm via C main  
program  
;
```

```

; SRB_ptr : pointer to the SRB data structure .
;           this is the predefined data structure used for data transfer
;
; data_ptr : pointer to the data areain the SRB
;

```

```

aspi_entry_point    dd    0
_SRB_ptr            dw    0
_data_ptr           dw    0

```

```

;
; define variables
;

```

```

.CODE

```

```

;
; start of code segment
;

```

```

PUBLIC _send_data

```

```

,*****
,
;
; send_data program FUNCTION
;
,*****
,

```

```

_send_data PROC

```

```

; push contents of registers on stack
; registers are needed for communication
    push  ax
    push  cx
    push  dx
    push  bx
    push  bp
    push  si
    push  di
    push  ds
    push  es
    push  ds
    mov   bx,[_data_ptr]
    mov   bx,[_SRB_ptr]

;
; place pointers to data and SRB data structures in bx register
;

    push  bx

; push pointers (contents of bx) onto stack ready for ASPI call

    lea   bx,aspi_entry_point

;
; load effective address of ASPI into ax register
;

    call  dword ptr[bx]

```

```
;
; call ASPI : message on SCSI bus
;

    add     sp,4

; push contents of registers off stack

    pop    es
    pop    ds
    pop    di
    pop    si
    pop    bp
    pop    bx
    pop    dx
    pop    cx
    pop    ax
    ret

_send_data endp
;
; end of function
;
END
```

B.3 Main C Program : COMMUN.C

```
#include "stdio.h"
#include "stdlib.h"
#include <dos.h>
#include <string.h>

/*    include standard libraries */

union REGS Regs;
struct SREGS SRegs;

/*    library defined regisers : ax, bx etc */

extern void send_data(void);

/*****/
/* send_data is defined as an external function : this is in scsicomm.asm */
/* this function does not return any values */
/*****/

extern int open_aspi(void);

/*****/
/* open_aspi function is an external function : (in open.asm) */
/* returns an integer indicating error or success */
/*****/

void verify(long no, long ink);

/* verify function used for verifying a sector : menu option */
```

```

void read_toc(void);

/* read table of contents function to read t.o.c of a disc : menu option */

void read6(long no);

/* read function used to read a sector : not used */

void seek6(long no);

/* seek function used to seek a sector : not used */

void show_sense(void);

/* show_sense function used to show sense data : shows Sense allocation
area */
/* shows data !! : only used at end of program */

void init_SRB(void);

/*    function initialises SRB values to default */
/*    CDB area is filled up later */

void read_capacity(void);

/*    function to read capacity of disc : menu option */

int test_data(void);

/*    used to constantly poll SCSI bus for status received !*/

void sense_data(char a[]);

```



```

/*    as show_sense but no data is shown */

void mode_sense(void);

/*    carries out a mode_sense on the drive unit : menu option */

void log_sense(char a[], int answer);

/*    carries out a log_sense on the drive unit : menu option */

void read_header(long no);

/*    reads the header : menu option */

void log_select(char a[]);

/*    carries out a log_select on the drive unit : menu option */

void mode_select(char a[]);

/*    carries out a mode_select on the drive unit : menu option */

void menu(void);

/*    outputs menu to screen */

char mdl, bdl, code;

/* variables are specified as characters : i.e. bytes */

struct msb
    {

```

```

        char  command_code;

/*  SCSI command code - non device specific */
        char  device_status;

/*  device status : status value which may be polled for */

        char  host_adapter_number;

/*  addressing : the host adapter has an id number */
/*  in this case there is only one host - but an id is still needed */
/*  since ASPI requires such data */
        char  scsi_request_flag;
/*  no used */

        char  reserved1 [4];
/*  not used : nevertheless must be present in data structure */

        char  target_id;
/*  target id : since >1 target can be present */

        char  lun;
/*  logical unit number : each target is allocated a lun by the resident */
/*  hardware */

        char  data_allocation_length[4];
/*  length of data address pointer in bytes */

        char  sense_allocation_length;
/*  pointer to sense allocation area*/

        char  data_offset[2];

```

```

/* data area offset address */

        char data_segment[2];
/* data segment address */

        char link_offset[2];
/* not used */

        char link_segment[2];
/* not used */

        char cdb_length;
/* length of CDB : this can vary : 10 bytes in this case */

        char host_status;
/* status of host on response */

        char target_status;
/* status of target on response */

        char post_routine_offset[2];
/* not used */

        char post_routine_segment[2];
/* not used */

        char reserved2[34];
/* not used */

        char cdb_data[10];
/* CDB area */

```

```

        char  sense_allocation_area[20];
/*      sense allocation area : 20 bytes are set in this case */

        } SRB;

/* SRB message data structure as specified in ASPI guidelines */

struct msb far *SRB_ptr;

/* SRB_ptr is defined as a pointer to the SRB data structure */

main()
{

/* LIST OF VARIABLES USED */

FILE *fptr, *fptr2;
int hour1;
int Thresh;
int retry;
long blocks;
int val;
int min1;
long pres;
int sec1;
int h1;
char last;
int hour2;
long inc;
int min2;
int sec2;
int h2;

```

```

int extra;
long block;
long start;
int h,s,m,hour;
long errs;
int chrs;
int error;
int ready;
int i;
char filename[15], filename2[15];
char *f1, *f2;
int choice;
char Data[80];
int time[4];
char far *data_ptr;

/* START OF PROGRAM */

error = open_aspi();

/* call open_aspi function - this is in open.asm */
/* if no aspi is present then error = 1 */

SRB_ptr = &SRB;

/* SRB pointer points to the SRB i.e. it is the address of the SRB structure */

data_ptr = &Data[0];

/* data pointer points to the data array i.e. it is the address of the data array
*/

```

```

val = 0;
if (error == 0)

/* if aspi is present then enter loop*/

{
printf("Requesting Input/Output");
init_SRB();

/*    function initialises all fields in the SRB ready for use */

send_data();

/*    place SRB on the SCSI bus using external assembler function*/

ready = test_data();

/*    test_data constantly poll SRB status for successful transfer*/

if (ready == 0 )

/*    if successful communication then enter loop */
    {

        for(i=0;i<80;i++)Data[i] = 0x00;

        /*    initialise data area */

        for (i=0;i<20;i++) SRB.sense_allocation_area[i]=0;

        /*    initialise S.A.A */
    }
}

```

```

Regs.x.ax = FP_SEG(data_ptr);

/* find the data segment address of the data_ptr*/

SRB.data_segment[1] = Regs.h.ah;
SRB.data_segment[0] = Regs.h.al;

/* set address of data segment using low and high
byte*/

Regs.x.ax = FP_OFF(data_ptr);

/* find the data offset address of the data_ptr*/

SRB.data_offset[1] = Regs.h.ah;
SRB.data_offset[0] = Regs.h.al;

/* set address of data offset using low and high
byte*/

choice=5;
while (choice!=0)
{
    menu();

    /* setup menu */

    scanf("%d",&choice);

    /* get choice */

    if (choice==0)

```

```

        /*    exit program */

        printf("\nENDING PROGRAM\n");
    }
    if (choice==6)
        /*    read header from a specified block
of data */

        {
        for(i=0;i<30;i++)Data[i] = 0xff;
        printf("\ninput block number:");
        scanf("%d",&block);
        read_header(block);
        /* use function to setup CDB for chosen
operation*/

        send_data();
        /*    send data to drive unit */
        sense_data(Data);
        /*    poll until sucessful status received */
        }
    if (choice==7)
        {
        for(i=0;i<30;i++) Data[i] = 0xff;
        read_toc();
        send_data();
        sense_data(Data);
        }
    if (choice==8)
        {
        for(i=0;i<30;i++)Data[i] = 0x0f;

        read_capacity();

```



```

        send_data();
        sense_data(Data);
    }
if (choice==1)
    {
    mode_sense();
    sense_data(Data);
    /*    show data area before command */
    send_data();
    sense_data(Data);
    /*    show data area after command */
    }
if (choice==2)
    {
    mode_select(Data);
    send_data();
    sense_data(Data);
    }
if (choice==3)
    {
    log_select(Data);
    send_data();
    sense_data(Data);
    }
if (choice==4)
    {
    /*    report on retries */
    retry=0;
    for(i=0;i<30;i++)Data[i] = 0x00;
    log_sense(Data,retry);
    send_data();
    ready = test_data();
    }

```

```

sense_data(Data);
if (Data[0]==0x30)
    printf("\nRetries = %d\n",Data[5]);
}
if (choice==5)
{
printf("\ninput filename :");
scanf("%s",&filename);
f1 = ".ret";
f2 = ".act";
strcpy(filename2,filename);
strcat(filename,f1);
strcat(filename2,f2);
fptr2 = fopen(filename2,"wt");
fptr = fopen(filename,"wt");
/* files to place results in */
printf("\nInput number of blocks:");
scanf("%ld",&blocks);
/* number of blocks to gather */
printf("\nInput Starting point:");
scanf("%ld",&start);
block = start;
/* starting point from which to get N
blocks */

inc=1;
printf("\n increment : ");
scanf("%ld",&inc);
/* increment x blocks at a time */
printf("\n Threshold : ");
scanf("%d",&Thresh);
/* set threshold T for access times */
/* times less than T are not recorded */

```

```

printf("\n retries per block (y/n): ");
last = 0;
errs = 0;
pres = 0;
blocks = blocks+block;

for(block=start;block<blocks;block=block+inc)
    {
    if (block==blocks)
        exit(0);
    printf("\nblock %ld\n",block);
    verify(block,inc);
    /*****/
    /***  get time 1      ***/
    /*****/
    Regs.h.ah = 0x2c;
    intdos(&Regs,&Regs);
    hour1 = Regs.h.ch;
    min1 = Regs.h.cl;
    sec1 = Regs.h.dh;
    h1 = Regs.h.dl;
    time[0] = hour1;
    time[1] = min1;
    time[2] = sec1;
    time[3] = h1;
    /*  time 1  is  before  message
transferred */

    send_data();
    /*****/
    /***  get time 2      ***/
    /*****/
    ready = test_data();

```

```

Regs.h.ah = 0x2c;
intdos(&Regs,&Regs);
hour2 = Regs.h.ch;
min2 = Regs.h.cl;
sec2 = Regs.h.dh;
h2 = Regs.h.dl;
/* time 2 obtained when successful

*/

/* status is recorded */

if (time[3]<=h2 )
    {extra = 0;
      sec2 = sec2 - 0;
    }
    else {
          extra = 100;
          sec2 = sec2 - 1;
        }
h = h2 - time[3] + extra;
if (time[2]<=sec2 )
    {
      extra = 0;
      min2 = min2 - 0;
    }
    else {
          extra = 60;
          min2 = min2 - 1;
        }
s = sec2 - time[2] + extra;
if (time[1]<=min2 ) {
      extra = 0;
      hour2 = hour2 - 0;
    }

```

```

        else {
            extra = 60;
            hour2 = hour2 - 1;
        }
    m = min2 - time[1] + extra;
    hour = hour2 - time[0];
    /* calculates time taken */
    fprintf(fp2, "\n%ld : %d
HSECS",block,s*100+h);

    for(i=0;i<30;i++)Data[i] = 0x00;
    log_sense(Data,retry);
    /* specify retries to be */
    /* recorded for block accessed */
    send_data();
    ready = test_data();
    sense_data(Data);
    if ((Data[0]==0x30)&&(Data[7]>0))
        {

    fprintf(fp2, "\n%ld : %d :
%d",block,Data[7],Data[8]);

        pres = pres + Data[7];

        }
    for(i=0;i<30;i++)Data[i] = 0x00;
    log_sense(Data,retry);
    send_data();
    ready = test_data();
    sense_data(Data);
    getchar();

```

```

        getchar();
        if (Data[0]==0x30)
            {
                printf("\n\nTotal Retries = %ld\n",pres);
            }

        fclose(fptr);
        fclose(fptr2);
        printf("\nfinished\n");
        show_sense();
    } /* end of verify choice*/

    if (choice!=0)chrs=getchar();
    if (choice!=0)chrs=getchar();
    } /*end of while loop */

} /*end of ready loop */
}
else
{
    printf("no aspi manager");
}
return(0);
}

void read_capacity(void)
{
    char *cdb;
    char READ_CAPACITY;
    printf("\nREAD_CAPACITY");
    cdb = SRB.cdb_data;
    READ_CAPACITY = 0x25;

```

```

*cdb++ = READ_CAPACITY;
*cdb++ = 0x00;
*cdb++ = 0;
*cdb++ = 0;
*cdb++ = 0;
*cdb++ = 0;
*cdb++ = 0;
*cdb++ = 0;
*cdb++ = 0;
*cdb++ = 0;
}

```

```
void read_toc(void)
```

```

{
char *cdb;
char READ_TOC;
printf("\nREAD_TOC");
cdb = SRB.cdb_data;
READ_TOC = 0x43;
*cdb++ = READ_TOC;          /* 0 */
*cdb++ = 0x00;             /* 1 MSF */
*cdb++ = 0;                /* 2 res */
*cdb++ = 0;                /* 3 res */
*cdb++ = 0;                /* 4 res */
*cdb++ = 0;                /* 5 res */
*cdb++ = 0;                /* 6 starting track */
*cdb++ = 0;                /* 7 all */
*cdb++ = 40;               /* 8 all */
*cdb++ = 0;                /* 9 res */

}

```

```

void read_header(long no)
{
    int i;
    char *cdb;
    char READ_HEADER;
    char Log_Block_Addr[4];
    for (i=0;i<4;i++) Log_Block_Addr[i] = 0;
    Log_Block_Addr[0] = (char)no;
    no = no>>8;
    Log_Block_Addr[1] = (char)no;
    no = no>>8;
    Log_Block_Addr[2] = (char)no;
    no = no>>8;
    Log_Block_Addr[3] = (char)no;
    printf("\nREAD_HEADER");
    cdb = SRB.cdb_data;
    READ_HEADER = 0x44;
    *cdb++ = READ_HEADER;          /* 0 */
    *cdb++ = 0x2;                 /* 1 MSF */
    *cdb++ = Log_Block_Addr[3];   /* 2 LBA */
    *cdb++ = Log_Block_Addr[2];   /* 3 LBA */
    *cdb++ = Log_Block_Addr[1];   /* 4 LBA */
    *cdb++ = Log_Block_Addr[0];   /* 5 LBA */
    *cdb++ = 0;                   /* 6 res */
    *cdb++ = 8;                   /* 7 res */
    *cdb++ = 0;                   /* 8 res */
    *cdb++ = 0;                   /* 9 res */
}

```

```

void log_select(char a[])
{
    char *cdb;

```



```

char LOG_SELECT;
printf("\nLOG SELECT");
cdb = SRB.cdb_data;
LOG_SELECT = 0x4c;
*cdb++ = LOG_SELECT;          /* 0 */
*cdb++ = 0x02;                /* 1 lun & res & PCR & SP */
*cdb++ = 0x40;                /* 2 PC */
*cdb++ = 0;                   /* 3 res */
*cdb++ = 0;                   /* 4 res */
*cdb++ = 0;                   /* 5 res */
*cdb++ = 0;                   /* 6 res */
*cdb++ = 0;                   /* 7 PLL */
*cdb++ = 0x0;                 /* 8 PLL */
*cdb++ = 0;                   /* 9 control */
}

```

```

void log_sense(char a[], int answer)

```

```

{
char *cdb;
char v2,v1;
char LOG_SENSE;

printf("\nLOG SENSE");
cdb = SRB.cdb_data;
LOG_SENSE = 0x4d;
v1=0;
v2=0;
if (answer==0)
{
printf("\npage code ");
scanf("%x",&v1);

```

```

        printf("\n%x\n",v1);
        printf("\nlent ");
        scanf("%x",&v2);
    }
else
    {
        v1=0x30;
        v2=20;
    }
*cdb++ = LOG_SENSE;
*cdb++ = 0x00;          /* lun & res & PPC & SP */
*cdb++ = v1 | 0x40;    /* PC & Page code */
*cdb++ = 0;           /* res */
*cdb++ = 0;           /* res */
*cdb++ = 0;           /* PP */
*cdb++ = 0;           /* PP */
*cdb++ = 0;           /* All lent */
*cdb++ = v2;          /* All lent */
*cdb++ = 0;           /* control */
}

```

```

void verify(long no, long ink)
{
    char *cdb;
    int i;
    char VER;
    char Log_Block_Addr[4];
    char length[2];
    VER = 0x2f;
    for (i=0;i<2;i++) length[i] = 0;
    for (i=0;i<4;i++) Log_Block_Addr[i] = 0;
    length[0] = (char)ink;
}

```

```

ink = ink>>8;
length[1] = (char)ink;
Log_Block_Addr[0] = (char)no;
no = no>>8;
Log_Block_Addr[1] = (char)no;
no = no>>8;
Log_Block_Addr[2] = (char)no;
no = no>>8;
Log_Block_Addr[3] = (char)no;
cdb = SRB.cdb_data;
*cdb++ = VER;
*cdb++ = 0;
*cdb++ = Log_Block_Addr[3];
*cdb++ = Log_Block_Addr[2];
*cdb++ = Log_Block_Addr[1];
*cdb++ = Log_Block_Addr[0];
*cdb++ = 0x0;
*cdb++ = length[1];
*cdb++ = length[0];
*cdb++ = 0x0;
}

```

```

void read6(long no)
{
char *cdb;
int i;
char READ;
char Log_Block_Addr[3];
READ = 0x08;
for (i=0;i<3;i++) Log_Block_Addr[i] = 0;
Log_Block_Addr[0] = (char)no;
no = no>>8;

```

```

Log_Block_Addr[1] = (char)no;
no = no>>8;
Log_Block_Addr[2] = (char)no;
cdb = SRB.cdb_data;
*cdb++ = READ;
*cdb++ = Log_Block_Addr[2];
*cdb++ = Log_Block_Addr[1];
*cdb++ = Log_Block_Addr[0];
*cdb++ = 0x1;
*cdb++ = 0;
}

```

void seek6(long no)

```

{
char *cdb;
int i;
char SEEK;
char Log_Block_Addr[3];
SEEK = 0x0b;
for (i=0;i<3;i++) Log_Block_Addr[i] = 0;
Log_Block_Addr[0] = (char)no;
no = no>>8;
Log_Block_Addr[1] = (char)no;
no = no>>8;
Log_Block_Addr[2] = (char)no;
cdb = SRB.cdb_data;
*cdb++ = SEEK;
*cdb++ = Log_Block_Addr[2];
*cdb++ = Log_Block_Addr[1];
*cdb++ = Log_Block_Addr[0];
*cdb++ = 0;
*cdb++ = 0;
}

```

```

void mode_select(char a[])
{
    char *cdb;
    char MODE_SELECT;
    char val;
    char erp;
    printf("\nMODE SELECT");
    cdb = SRB.cdb_data;
    MODE_SELECT = 0x15;
    printf("\npage:");
    scanf("%d",&val);
    *cdb++ = MODE_SELECT;
    *cdb++ = 0x10;
    *cdb++ = 0;
    *cdb++ = 0;
    *cdb++ = 20;
    *cdb++ = 0;
    a[0] = 0;
    a[7] = 0;
    a[3] = 8;
    a[10]= 8;/*9;*/
    a[11]= 0;/*0x20; */
    if (val==0)
    {
        a[12]= 00;
        a[13]= 6;
        a[14]= 0x0;
        a[15]= 0;
        a[16]=0;
        a[17]=0;
    }
}

```

```

        a[18]=0;
        a[19]=0;
    }
    if (val==1)
    {
        printf("\n? (00 = good/ 04 = retry):");
        scanf("%x",&erp);
        a[12]= 1;
        a[13]= 6;
        a[14]=erp;
        a[15]=0xa;
        a[16]=0;
        a[17]=0;
        a[18]=0;
        a[19]=0;
    }
    if (val==0x0d)
    {
        a[12]= 0x0d;
        a[13]= 6;
        a[14]= 0x0;
    }
}

```

```

void mode_sense(void)
{
    char val;
    char *cdb;
    char PC;
    char MODE_SENSE;
    char len;
    char page_code;

```

```

printf("\nMODE SENSE");
cdb = SRB.cdb_data;
MODE_SENSE = 0x1a;
PC = (char)((0x0) << 6);
page_code = 0x00;
val = 0;
printf("\npage:");
scanf("%d",&val);
/* page to use */
page_code = val;
code = val;
len = 0x25;
*cdb++ = MODE_SENSE;
*cdb++ = 0x00;
*cdb++ = PC | page_code;
*cdb++ = 0;
*cdb++ = len;
*cdb++ = 0;
}

void sense_data(char a[])
{
int i;
char *cdb;
int ready;

cdb = SRB.cdb_data;
ready = 1;
printf("\n");
while (SRB.device_status==0)
{
printf("waiting for connection to..");

```

```

        for (i=1;i<28;i++) printf("\b");
    }

printf("\n");
if ((SRB.target_status!=0) || (SRB.host_status!=0))
    {
        printf("\nstatus:%x",SRB.device_status);
        printf("\nh.a.s status:%x",SRB.host_status);
        printf("\nt.s status:%x",SRB.target_status);
        show_sense();
    }

printf("\nDATA");
printf("\n====");
for (i=0;j<80;i++) {
    if ((i % 10) == 0) printf("\n");
        printf("%x:",a[i]);
    }
}

```

```

void menu(void)
{
    printf("\n\t MENU");
    printf("\n\t ==== \n");
    printf("\n\t (0) EXIT");
    printf("\n\t (1) MODESENSE");
    printf("\n\t (2) MODESELECT");
    printf("\n\t (3) LOGSELECT");
    printf("\n\t (4) LOGSENSE");
    printf("\n\t (5) READ N BLOCKS");
    printf("\n\t (6) READ HEADER");
    printf("\n\t (7) READ TOC");
}

```



```

printf("\n\t (8) READ CAPACITY\n");
printf("\nEnter option: ");
}

int test_data(void)
{
int i;
int ready;
ready = 1;
printf("\n");
while (SRB.device_status==0)
    {
        printf("waiting for connection to..");
        for (i=1;i<28;i++) printf("\b");
    }
printf("\n");
if ((SRB.target_status!=0) || (SRB.host_status!=0))
    {
        printf("\nstatus:%x",SRB.device_status);
        printf("\nh.a.s status:%x",SRB.host_status);
        printf("\nt.s status:%x",SRB.target_status);
        show_sense();
    }
    else {
        printf("\nDEVICE ok\n");
        ready = 0;
    }
return(ready);
}

```

```

void show_sense(void)
    {
int i;
    printf("\n\nMODE SENSE DATA");
    printf("\n=====");
    for (i=0;i<20;i++) {
        if ((i % 4) == 0) printf("\n");
            printf("%x:",SRB.sense_allocation_area[i]);
        }
    }

```

```

void init_SRB(void)
    {
int i;

    SRB.command_code = 2;
    SRB.device_status = 0xff;
    SRB.host_adapter_number = 0;
    SRB.scsi_request_flag = 0;
    for (i=0;i<4;i++) SRB.reserved1[i] = 0;
    SRB.target_id = 1;
    SRB.lun = 0;
    for(i=0;i<3;i++) SRB.data_allocation_length[i] = 0xff;
    SRB.data_allocation_length[3] = 0xff;
    SRB.sense_allocation_length = 20;
    SRB.cdb_length = 6;
    SRB.host_status = 4;
    SRB.target_status = 4;
    for(i=0;i<34;i++)SRB.reserved2[i] = 0;
    for(i=0;i<6;i++)SRB.cdb_data[i] = 0;
    for(i=0;i<20;i++)SRB.sense_allocation_area[i] = 0;

```

}

Appendix C

The Effect Of Mutiple Burst Errors

Multiple burst errors have not been examined in the course of this thesis due to the reasons outlined in **Chapter Eleven**. However the effect of such errors on the coding scheme can be illustrated using the simulation model and illustrations.

The effect is clear, when bursts occur in close proximity they effect the correction possible at the C2 Decoder. The position of errors before and after Deinterleave Strategy II is illustrated in **Figure C.1** and **Figure C.2** respectively. The dispersal of errors is successful, however due to the proximity of the errors, two parallel strips of errors occur along the Sector. Error correction is compromised in each Frame where both bursts occur.

Appendix C.1 : Illustration Of Two Bursts Within A Sector

Figure C.1

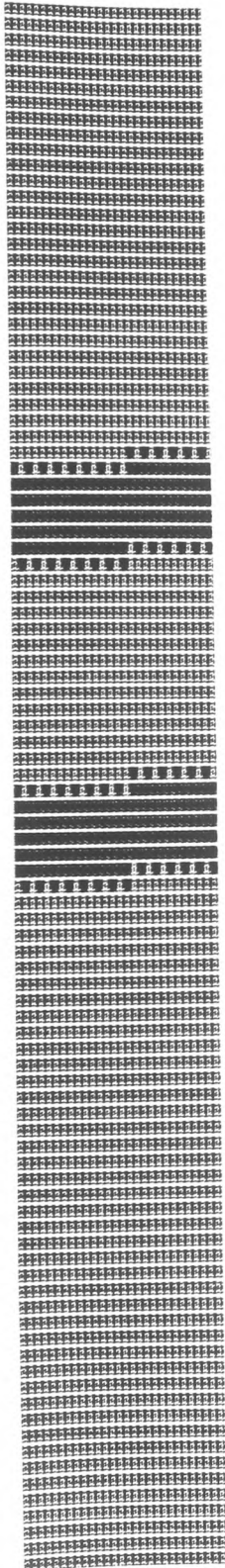
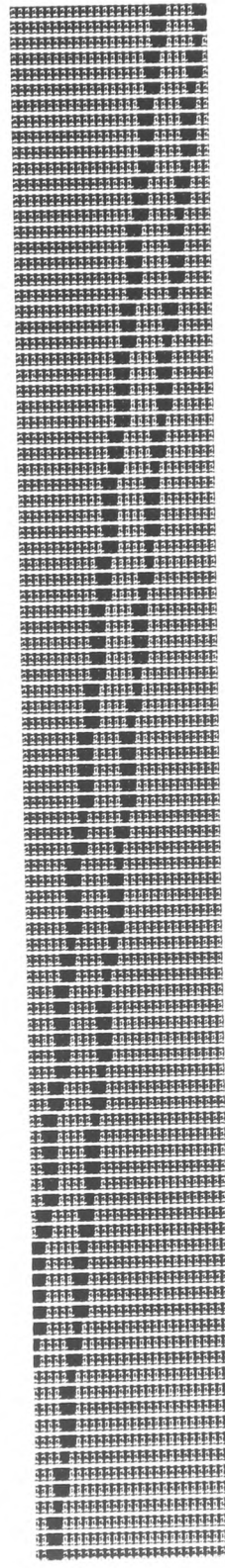


Figure C.2



Glossary

Access Time	Quantity of time taken to access a block of data. This includes the time taken to seek to, decode and check a given block of data.
ASPI	Advanced SCSI Protocol Interface. Used to communicate with the CDROM
Burst Error	A number of contiguous bit errors occurring in sequence.
Channel Code	See modulation code.
C1 Code	The first error correction code of CIRC.
C2 Code	The second error correction code of CIRC.
C1 Error Count	The error statistic used to indicate the perceived number of Frames in error after C1 error correction has been applied.
C2 Error Count	The error statistic used to indicate the perceived number of Frames in error after C2 error correction has been applied.
CIRC	Cross Interleaved Reed-Solomon Codes.
CD	Compact Disc.
CDROM	Compact Disc Read Only Memory. The mass data storage application upon the compact disc.
CAV	Constant Angular Velocity.
CLV	Constant Linear Velocity.
CRC	Cyclic Redundancy Check.
Data Block	A sector of information.
DDS	Digital Data Storage. This is the form of RDAT which is specific to digital data storage as a pose to analogue data storage. Also referred to as DDS-RDAT.

DDS-RDAT	see DDS.
DSV	Digital Sum Variation.
ECMA	European Computer Manufacturers Association.
EDC	Error Detection Code.
EFM	Eight Fourteen Modulation code. The modulation code of the CDROM and CD.
Flagging	An error location system used when more than one error correction code is used. Symbols are flagged such that they can be identified when the next decoding occurs. In the Reed Solomon Codes this facilitates greater error correction.
Galois Field	Finite field upon which Reed Solomon encoding, decoding and correction is based.
Inter-Block Access Time	The Access Time between successive logical blocks. This is used in the course of this thesis.
Latency	The period of time it takes for a disc to rotate fully.
Logical Sector	The 2352 bytes organisation of data before CIRC is applied.
Modulation Codes	The coding schemes applied to data so that they may be recorded upon a communications channel. Also known as channel or recording codes.
Recording Code	See modulation code.
RDAT	Rotatory Digital Audio Tape.
Reed Solomon Codes	A family of error codes used for the correction of symbols rather than bits. Codes with $2n$ parity can locate and correct n bytes or $2n$ if there locations are known.
RLL	Run Length Limited. Constraints applied to data when modulation occurs.
SCSI	Small Computer Systems Inteface.

Sector	The smallest addressable area upon the CDROM.
Section	The physical representation of a sector on the CDROM disc.
SNR	Signal To Noise Ratio.
Symbol	A number of associated bits. In the case of the CDROM eight bit byte symbols are used.
Syndrome	This is an indication of error. It can also be used for both error location and correction in the Reed Solomon codes used in the CDROM.
Table Of Contents	A data area situated at the centre of a CD/CDROM disc. The Table Of Contents holds addressing data for tracks and data blocks upon the disc. This is written many times since it is essential for data capture.
Track	A area on the disc where data is held.
WORM	Write One - Read Many.

