MULTIPLE LAYERED REED-SOLOMON CODES IN AUDIO DISKS WITH A FEEDBACK SYSTEM

BY

SREENIVAS RAO BELAVADI SATISH

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Professor Richard E. Blahut

ABSTRACT

In this thesis, we study the performance of powerful multilayered Reed-Solomon error-correcting coding applied to audio disks to protect data from errors. A detailed analysis of the performance of four layers of Reed-Solomon coding is addressed in presence of burst errors. The application of Reed-Solomon coding in an audio disk is explored from a mathematical viewpoint. We propose to have a feedback system between layers of Reed-Solomon coding to facilitate the error correction. A successful simulation of the four layers of Reed-Solomon coding applied to audio disks is performed. Our simulation results combined with a U.S. patent by Shinya Ozaki indicate that there is an equivalent or an improvement in the performance of the system when the error correction system is assisted with feedback. We conclude with a discussion of the advantages and limitations of including such a feedback system.

To my Guru, parents and brother

ACKNOWLEDGEMENTS

I would like to thank Prof. Blahut for giving me an opportunity to do research and learn the beautiful areas of coding theory and information theory. It is his advice and support that gave me confidence to face challenges during the course of my study. I thank all my teachers, gurus, who have educated me and whose blessings are always with me. I thank my loving parents and brother who have given me strength, patience and belief to proceed. Their blessings and love helped me to keep going and made me what I am today. I thank God for guiding me along the path and blessing me with the above.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction and Scope of Thesis

The introduction of information theory and coding theory during late 1940's was a remarkable period for digital communications. Claude Shannon reshaped the development of communications theory and established a remarkable foundation. The development of coding theory was a consequence of information theory, and later various accomplishments were made by Hamming and other mathematicians**.**

In the area of coding theory, there has always been an interest to achieve error-correcting codes. The designs of such codes allow the system to receive erroneous messages and correct the errors induced. The intelligence in such codes lies in the redundant bits added, which is a consequence of the underlying math. Reed-Solomon codes are one such class of powerful error-correcting code used in a wide range of systems today. Some of these applications are audio disks, wireless systems, and space telecommunications.

This thesis focuses on the application of Reed-Solomon codes in audio disks and solutions to improve the performance. Audio disks were introduced jointly by Sony and Philips in 1982, and since then have played a prominent role as mediums for mp3 and audio files [1]. Audio disks are used to store mp3 music and recordings. Since the audio disks are prone to scratches and fingerprints due to mishandling, the data written on the disk may get corrupted. Multiple layers of Reed-Solomon coding accompanied with strategic interleaving methods establish a strong error-correcting coding scheme.

Though the thesis emphasizes the application of Reed-Solomon coding in audio disks, the usage of this profound coding technique can be generalized to other applications as well. For example, the Hubble space telescope sends data from deep space which when received on earth has a signal level barely above the noise floor. The receiver is sophisticated with temperature of the receiving system kept at low temperatures. However, even with such care taken, it is difficult to decode the codeword and retrieve the message signal due to errors present. In such applications, re-transmission is not an alternative and priority in retrieving the data from received signal is high. Reed-Solomon codes offer a powerful error correction capability for such applications.

The performance of the Reed-Solomon coding is remarkable and is worth appreciation. However, there are situations when the multiple layered Reed-Solomon coding scheme in the audio disk fails. There is a scope for improvement and it is discussed in this thesis. An improvement in performance is suggested by utilizing a feedback system to the existing multiple-layered Reed-Solomon codes. The feedback promises an enhanced data protection at the cost of increased complexity during implementation. Hence, there exists a trade-off between performance and system complexity.

This thesis discusses the mathematics involved in such multiple-layered coding systems and an effort is made to explore the application of Reed-Solomon coding on audio disks in detail. The involvement of finite field arithmetic and polynomial representation of codes requires a deeper foundation of mathematical descriptions. To understand the Reed-Solomon coding scheme it is necessary to adopt a mathematical point of view. An effort is made to present the material with the required mathematical maturity.

1.2 Motivation

The digital audio disk was implemented in 1982 together by Sony and Philips [1]. Today, the technology is used in BLU-RAY, DVDs, CD-ROMs, etc. The error correction coding mechanism is crucial in these disks due to their vulnerability to scratches, fingerprints and rough use. To understand how delicate a common audio disk is, and to appreciate the need for error control coding, it is necessary to study its design.

An audio disk is 12 cm in diameter with a center hole of 15 mm. The digital audio data is written in spiral strips from the inner to the outer region of the disk. The digitizing rate of the recorded audio music is at 44.1 kHz. The distance between successive stripes where digital data is recorded is 1.6 μm [2]. The data when written into the audio disk is actually written onto the surface aluminum layer of the disk causing pits and bumps. There are additional layers below the aluminum layer which act as layers for protection. However, the aluminum layer provides high reflectivity to the laser and directs it to the lens if there is a bump. While reading data off the disk, the optical signal reflected from the disk is read into an objective lens. The transition from a pit to a bump is decoded as a 1 and no transition for time period T is considered a 0. The distance between a bump and height is about 125 nm. Considering the numbers and the dimensions, one can imagine how a fingerprint or even a scratch could damage the data stored. Several bytes of data could be affected, resulting in loss of momentary music and this is often observed as a click or a jump during the playback of the audio file.

Such handling and rough usage is unavoidable and it is desired that the audio disk system handles them. Thus, there is a necessity of error-correcting codes in audio disks. To design error-correcting codes, it is necessary to treat an audio disk as a communication system.

1.3 Audio Disks as a Communication System

A communication system models the data transfer that takes place from source to destination through a channel medium. The source and destination may be separated in space or, equivalently, may be separated in time. In an audio disk system, we consider the data written on the disk as the source, and the receiver or destination as the reading system attempting to read the data back. The discrepancy in the channel is due to the fingerprints and scratches on the disk and noise. The data is not actually transferred or sent from a channel, and the data to be read is already available on the disk. Hence, the necessity of modeling the channel is avoided since error correction is based on correcting the corrupted data available. However, one can imagine that the data has already been passed through an imaginary channel, which represents fingerprints, scratches, etc. It is this corrupted data that we are trying to make the most of by decoding and correcting errors.

Although we will not be requiring the modeling of the channel, there is a necessity to treat the system as a communication system. The reason being that the process of data being encoded and retrieving back is similar to that of a communication system. It is convenient to treat the audio disk as a communication system where the data is read after being exposed to mishandling for a long period of time. Hence, it is necessary to understand the basics of a communication system and its structure.

1.4 Basics of Communication Systems

The transmission of data from source to destination is modeled into a series of communication blocks that when integrated act together for successful transmission. Figure 1.1 gives a block diagram view of a communication system. Notice that each block has a corresponding functionality which plays a vital role in affecting the performance of the entire system.

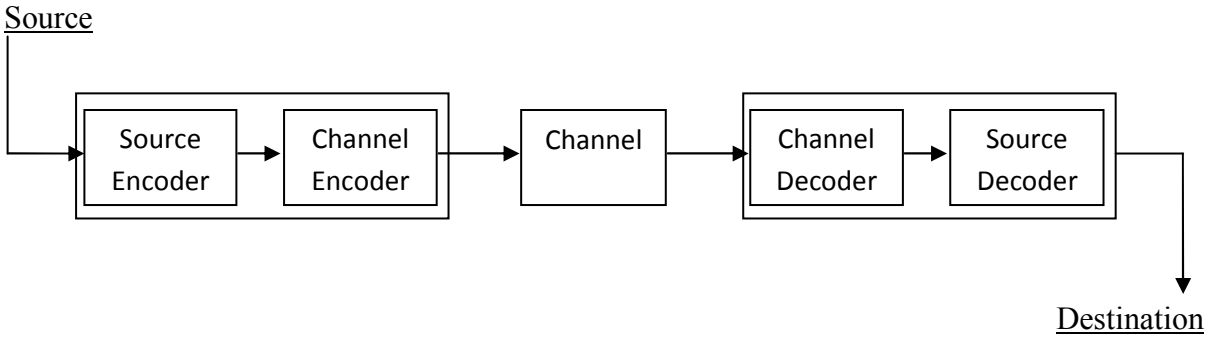

FIG. 1.1 Typical Communication System.

1.4.1 Source

The generation of bits as a sequence originates here. The source is equipped with a statistically strategic way of producing bits. This helps in the estimation of bits at the receiver end as there is a prior probability defined in the process of generation of bits.

1.4.2 Source Encoder

The source encoder is used to compress the data in order to avoid redundant information. The benefit of doing so is that fewer data need to be sent during every channel use. The data compression process is done prior to the Reed-Solomon code application.

1.4.3 Channel Encoder

This is the unit where the Reed-Solomon code is applied on the data available. The channel encoder is an intelligent way to add redundancy to the information, thereby causing inefficiency in data; however, it helps in data correction at the receiver. The redundant bits are referred to as parity bits or check bits, and as shall be seen later, the error correction capability of Reed-Solomon code depends on how many of these parity bits are added. The audio disk is equipped with complex multiple layers of Reed-Solomon codes with interleaving to enhance data correction. Note that unlike the source encoder, this unit introduces redundancy.

1.4.4 Channel

The data is then passed through the channel which may corrupt the data at the receiver. There are various ways data could be corrupted in the channel, such as the effects of instrumental inefficiency, attenuation of the signal, interference, and so forth. However, the channel is modeled to a known mathematical function which closely resembles its behavior for the simplicity of understanding the effects. In this thesis, study is on data which is available from the disks and which can be thought of as data that has already been sent on a channel which has the effects of fingerprints, scratches, etc.

1.4.5 Channel Decoder

The function of the channel decoder is the opposite of the channel encoder. The decoder tries to find a close match of the data word that corresponds to this codeword. If successful, the data is pushed to the next step and, if not successful, then several attempts are made in order

to decode. If the codeword is accompanied with continuous unsuccessful decoding events, then the receiver asks for re-transmission of the signal.

1.4.6 Source Decoder

At this stage the data is assured to be correctly received or corrected by the channel decoder. The data is passed through the decoder and performs the reverse process of source encoding. The data is hence progressed through to the destination.

1.4.7 Errors in an Audio Disk

In an audio disk there are various types of errors that affect the data. The two main types are burst errors and other general / random errors. The reason burst errors are given attention is because of their ability to affect the data. Burst errors affect multiple bits together as a sequence of errors and hence they affect multiple codewords.

Multiple layers of Reed-Solomon coding in an audio disk protect the data against burst errors. Random errors are usually handled by a single layer, but occasionally they might propagate through multiple layers.

1.5 Origin of Random Errors in an Audio Disk

Random errors are common in a communication system and in an audio disk. They usually affect single bit or multiple bits, but are very sparse. Error-correcting codes can handle such sparse codes, and are not a major concern. However, it is necessary to incorporate these errors during performance evaluation.

Following are the origins of such errors in an audio disk:

1) Jitter in system: Jitter originates mainly due to cross talk between signals or interference between signals in a system. In an audio disk, there exists no emphasis on frame address, frames being collections of codewords, and hence, while seeking the frame to be played, delays may be induced.

2) Interference of signals in the system.

1.6 Origin of Burst Errors in an Audio Disk

Burst errors affect multiple bits as a sequence of errors. The result is losing almost entire packets of data and leading to a necessity of re-transmission. In audio disks, burst errors are commonly observed and need to be treated. Scratches and fingerprints are enough to corrupt data which are at a microscopic level. Burst errors affect a series of data bits hence causing the conventional decoding techniques to fail since the true codeword is lost. Burst errors may range from bits in a codeword to multiple codewords being entirely corrupted.

In an audio disk there are various reasons as to why burst errors occur. Some are listed as follows:

1) Disk errors at the time of production. These include errors induced during the industry production of the audio disks. They range from the quality of reflective index to the poor film of the audio disk onto which the data is written.

2) Incorrect handling by user: Fingerprints and scratches by the user due to mishandling the audio disk. These are one of the main reasons producing burst errors. Burst errors need to be

carefully treated with the error correction coding techniques (Reed-Solomon coding) so that the data stored are not affected.

3) Playback system faults: The playback system may have faults of its own when dispatched from the industry. As the playback faults are macroscopic and affect multiple seconds during playback, they are capable of inducing burst errors.

CHAPTER 2

CODING THEORY

2.1 Introduction

Given an erroneous channel with certain characteristics, and set of symbols to be transmitted, we can code the symbols in such a way that a certain number of errors can be detected or even corrected depending on the coding scheme. The cost paid is the redundancy added to the symbols that we transmit. With an acceptable amount of redundancy bits, the error detection and correction achieved plays a vital role in successful message transmission.

This chapter introduces various fundamentals that help assist in the understanding of the applied coding theory aspects in systems. These concepts act as preliminaries in the understanding of future introduced concepts. Later in the chapter different strategies are discussed that are generally used on coding schemes for data correction. These strategies are applied on audio disks and provide a look at the coding scheme which will be discussed in further detail later.

2.2 Codeword

Data is usually not transmitted in its pure form over any channel because valuable information could be lost due to unavoidable noisy errors in the channel. Hence, an intelligent way is to implement coding such that the errors are detected while receiving the data. The symbol sent on a channel is termed a codeword, and the set of such codewords forms the code.

The message and codeword pair can be understood as a mapping scheme. This mapping scheme should be a one-to-one mapping to avoid discrepancy at the receiver. Set *A* and *B* are one-to-one if for every element in set *A* there exists only one unique element in set *B*.

2.3 Block Code

A codeword is called a block code when the message symbols are encoded as blocks or groups of size *k* into codewords of size *n*. As an example, the Reed-Solomon code is a nonbinary block code, which means it encodes on multiple message symbols to produce a codeword.

2.4 Linear Block Code

A block code is said to be linear if the arithmetic addition of two codewords also results in a codeword belonging to the code. If the code is a block code, then it is termed a linear block code.

Linearity plays an important property in the arithmetic nature of the codewords. Reed-Solomon code is a linear block error-correcting code. This property is extensively used during the encoding and decoding process of Reed-Solomon coding where the codes are represented by polynomial equivalents and arithmetic is performed on them.

2.5 Code Rate

Code rate is defined as the ratio of the message length, *k*, encoded to the resulting block length of size *n*.

$$\text{Code rate} = k / n \tag{2.1}$$

As shall be seen later, as the message content is reduced, or as the redundancy is increased, the error correction capability of Reed-Solomon code increases.

For example, if message of 3 bits (010) is coded into 7 bits (0101101) then the resulting code rate = $k/n = 3/7$.

Intuitively, there seems to be a trade-off between information content transmitted per use of the channel and the error correction capability, and shall be visited in the later chapters.

2.6 Hamming Weight

Hamming weight is defined as the number of nonzero elements in a codeword *C*. It is often represented as $w_H(C)$.

As an example, consider the codeword (0101101) that is used to encode message bit (010) as shown above. The Hamming weight of this codeword is:

$$w_H(C) = w_H(0101101) = 4.$$

because the number of nonzero elements in (0101101) is 4.

2.7 Hamming Distance

The Hamming distance between two codewords is defined by the minimum number of positions by which the two codewords in a coding system differ. Hamming distance between codewords *C1* and *C2* is represented as $d_H(C1, C2)$.

As an example, consider the Hamming distance between the codeword introduced above:

C1 = (0101101) and the all-zero codeword, *C2* = (0000000).

The Hamming distance, $d_H(C1, C2) = 4$ because the number of bits that *C1* and *C2* differ by is 4 (in positions 2, 4, 5 and 7).

Since in mod 2 arithmetic, the sum of 1+1 results in 0, the Hamming distance can be thought as the sum of codewords *C1* and *C2*. This arithmetic point of view of understanding the codewords is vital in coding theory and will be explored in the following chapters. This mathematical understanding assists in appreciating the arithmetic involved in Reed-Solomon coding.

The Hamming weight of a codeword *C* introduced in the previous section can also be defined as the Hamming distance between *C* and the all-zero codeword in the code.

2.8 Minimum Distance in a Code

The minimum distance in a code is the minimum Hamming distance that is present between any set of two codewords. This provides a way of seeing how far apart the codewords are in a coding scheme. Not all sequences need to be a codeword, only some selected ones are. Minimum distance is a measure of how diverse the selected codewords are.

The error correction and detection capability of a code is directly related to the minimum distance of the code. A code can detect up to *t* errors if and only if:

$$\text{Minimum distance, } d_{min} \geq t + 1 \tag{2.2}$$

This equation is only for detecting *t* errors. However, this criterion does not hold for error correction. For a code to correct up to *t* errors the following must hold:

$$\text{Minimum distance, } d_{min} \geq 2t + 1 \tag{2.3}$$

It is to be noted that for correcting t errors, the minimum distance is almost twice as much it was for error detection. This fact will be explained later treating codewords as spheres.

Also, not all sequences in a code are codewords. Only a selected group of sequences form the code. This is the foundation of coding theory, and the addition of parity bits assures that we enlarge the content of such non-codeword sequences.

Consider each codeword chosen in a code as a center of an imaginary sphere. Then each codeword will be separated from each other by the minimum distance since they differ by that many bits. Now each codeword is surrounded by non-codeword sequences which have a closer distance than the minimum distance of the code.

Now if a codeword satisfies Equation (2.2), then the codewords are far apart such that they are recognizable when received. In other words, if any bit is corrupt the receiver will recognize the difference and hence the error is detectable. Since, the sphere centers are t +1 distance apart, two neighboring codewords can be distinguished as long as they have up to t different bits. Hence, t changes in the bit stream of the codeword are detectable. Hence, the coding scheme can detect t errors.

If a codeword satisfies Equation (2.3), then the sphere centers are $2t + 1$ distance apart. This reveals two conclusions. The first is that $2t$ errors are now detectable which is intuitive. However, the second conclusion is more informative. Since the spheres are $2t + 1$ distance apart, the radius of each of them is t. Hence, a sphere with a selected codeword as center has those sequences whose minimum distance is up to t from the center of the sphere. This explains why the code now can realize the t different errors and correct them to the

codeword. In other words, if a non-codeword sequence is received, it is mapped to the center. This explains that the code has t error correction capability.

From the above discussion it is desirable to have a large minimum distance. Having a large minimum distance will help in detecting and correcting errors. However, having a code with a large minimum distance would result in two problems. The first is that the codeword length should be large enough to accommodate the minimum distance. And the second is that only a few sequences can be used as actual codewords in order to get a larger sphere radius. Hence, there is need for an upper bound. This upper bound is known as the Singleton bound and is represented as:

$$2^{dmin} \leq 2^{n+1} / |C| \tag{2.4}$$

Where, d_{min} is the minimum distance, n is the length of the codeword, and $|C|$ being the cardinality of the code or the number of codewords used. Note that 2 can be replaced by q to obtain the Singleton bound for q-ary code.

2.9 Strategies Used on Existing Codes

There are many existing codes to choose from, for example, the Hamming code, Reed-Solomon codes, LDPC codes, etc.; however, depending on the application, these codes can be modified and grouped to provide an overall improvement in the error correction performance.

Existing codes can be grouped or modified to obtain better error correction capabilities. Some of the strategies are discussed in this section. The strategies mentioned here are the ones employed in audio disks and will be discussed in the following sections.

2.9.1 Interleaving

Interleaving is the process of spreading out the codeword symbols over the entire set of codewords. This process is a spreading technique and requires reverse interleaving which is also referred to as de-interleaving at the receiver.

The significance of interleaving can be explained with the following example. Consider an error burst that affects 3 or 4 codeword symbols completely. Since they are corrupted in all bits including parity bits, it is not possible to recover the data since the error surpasses the error correction capability. Instead of re-transmission, interleaving can be employed, which spreads out the codeword over all codewords used. This assures that the error is also spread across many codewords. During the de-interleave phase, these errors are spread out and each codeword is allowed to correct certain number of errors. Hence, data which was lost in the previous case is recovered based on the error correction capability of the code used. Interleaving needs to be applied during the encoding process.

2.9.2 Product Codes

Product coding is a technique of generating codes with increased minimum distance. The rows and columns are coded individually with coding schemes *C1* and *C2*. If the minimum distance of *C1* is *d1*, and the minimum distance of *C2* is *d2*, then the resulting product code will have minimum distance, d_{prod}, as:

$$d_{prod} = (d1)(d2) \tag{2.5}$$

This shows that product codes can be used to generate a better code by coding the given data set in a special way. One has to code the rows followed by coding the columns separately or vice versa.

Reed-Solomon code (45, 43) and Reed-Solomon code (26, 24) are applied as product codes in an audio disk. This will be visited in the later chapters.

2.9.3 Multiple Layers of Coding

Multiple layers of coding employed with interleaving helps in improving the error correction capability. In audio disks, multiple layers of Reed-Solomon coding are employed with interleaving and are termed as CIRC or cross-interleaved Reed-Solomon codes. The advantage of having many layers of Reed-Solomon coding helps in finding the erasures in the following coding layers.

The price paid is the more complex coding system that needs to be implemented. However, in audio disks since the data recovery is emphasized, multilayered coding and interleaving is employed. As we shall see, product codes are also applied using two Reed-Solomon codes.

The complexity added due to the above techniques is the trade-off and needs consideration depending on the application.

CHAPTER 3

FINITE FIELD ARITHMETIC

3.1 Introduction

In order to understand the Reed-Solomon codes, it is necessary to look at the codes from a mathematical perspective. Certain abstract mathematical concepts are introduced in this chapter to help understand the Reed-Solomon codes. Reed-Solomon codes are constructed from finite field extensions, and it is important to understand the mathematical prerequisite to proceed.

3.2 Fields

A good way to define fields is that they are algebraic systems of numbers in which we can add, subtract, divide and multiply the numbers such that the standard properties hold.

For a field F, the standard properties include:

Distributive law:

$$(a1+a2) +a3 = a1+ (a2 + a3), \forall\ a1, a2, a3 \in F \tag{3.1}$$

The additive identity (0 in ***R***, real number system) :

$$a1 + 0 = \text{a1}, \forall\ a1 \in F \tag{3.2}$$

The multiplicative identity (1 in ***R***, real number system):

$$(a1)\ (\text{J}) = a1, \forall\ a1, J \in F \tag{3.3}$$

The additive inverse:

$$a + a^{-1} = 0, \ \forall\ a, a^{-1} \in F \tag{3.4}$$

The multiplicative inverse:

$$(a)(a^{-1}) = J, \ \forall\ a, a^{-1}, J \in F \tag{3.5}$$

Examples of fields include ***R***, the real number system; ***C***, the complex number system; ***Q***, the set of rational numbers; and so on.

3.3 Finite Fields or Galois Fields

A field that has finitely many elements is called a finite field. Clearly, ***R***, ***C*** and ***Q*** as defined above are not finite. However, there exist other fields which have finitely many elements.

Since finite fields are fields with special properties, the general properties that apply to a field apply to finite fields. However, in addition to the properties mentioned above, there are additional properties mentioned as follows.

The basic operations of the finite field are addition and multiplication. The basic operations always result in an element in the finite field. Division and subtraction, however, can be thought of as the inverse functions of addition and multiplication.

Addition:

$$a + b = c1, \quad \forall\ a, b, c1 \in F \tag{3.6}$$

Multiplication:

$$(a)(b) = c2, \quad \forall\ a, b, c2 \in F \tag{3.7}$$

Subtraction:

$$a + (-b) = c3, \forall\, a, -b, c3 \in F \tag{3.8}$$

Division:

$$a / b = (a)(b^{-1}) = c4, \forall\, a, b^{-1}, c4 \in F \tag{3.9}$$

The modular arithmetic satisfies the condition that all the elements are contained in the field. Any overflow is folded back into the field because of the mod p arithmetic where p is the number of elements in the field.

3.4 Existence of Prime or Power of Prime-sized Finite Fields

For any prime number q, there exists a finite field or Galois field, termed as GF(q) which has q elements contained in it. The order of a finite field is the number of elements that are contained in the finite field. The order of GF(q) is q since there are q unique elements comprising this field. It is to be noted that the field contains q elements, and hence to avoid overflow during arithmetic operations, modular arithmetic is implemented. The arithmetic operations of addition and multiplication are made possible by having a modular arithmetic of mod q, where q is the prime number of elements in GF(q).

For every prime number-sized field there exists at least one element, called the primitive element, such that it can uniquely define all other elements of the field through its powers.

3.5 Primitive Element

Every finite field with a prime number of elements has at least one element that can list out all nonzero elements of the field through its powers. An element with such a characteristic

property is termed the primitive element of the field. As an example, consider GF(7) which is a finite field with a prime number of elements. It is to be noted that mod 7 arithmetic is to be applied in this field to ensure the outcome of all arithmetic operations will be contained in the field. The element 3 is the primitive element for this field since:

$3^0 = 1,\quad 3^1 = 3,\ 3^2 = 9 \bmod 7 = 2,\ 3^3 = 27 \bmod 7 = 6,\ 3^4 = 81 \bmod 7 = 4,\ 3^5 = 243 \bmod 7 = 5.$

1 is not primitive since the resulting values are only 1. It is interesting that there exist elements in the finite field which are not primitive and have an interesting property of cycling through only certain elements as their powers are evaluated. Such elements form a cyclic group in the finite field.

3.6 Extension of Galois Fields

Finite fields can also be used to construct larger fields called extension fields. The process of constructing higher fields from smaller fields is called field extension.

The previously mentioned behavior for prime number-sized finite fields, GF(q), can be extended for the powers of prime number-sized fields, GF(q^m) for m >1. The fields that are sized with elements with powers of prime numbers are often termed extension fields and can be constructed from GF(q), a finite field having q elements. Moreover, for every prime power of prime number q, there exists only one unique field GF(q^m) for a given value of m. A good analogy to extension fields is the real and complex number systems. Complex number fields can be considered as the field extensions of real number fields. The real number field is a subset of the complex number field, just like the general case where the GF(q) field is a subset of GF(q^m).

In Reed-Solomon coding, we will be considering GF(2^m) fields, and hence the discussion here is concentrated on GF(2) and its extension fields. Clearly, GF(2) has 2 elements, namely 1 and 0. GF(2^m) has 2^m elements since it is a finite field with $q = 2^m$ elements. As GF(2^m) is an extension field of GF(2), elements 0 and 1 belong to the extension field GF(2^m) as well. Since GF(2^n) is a finite field, it has a primitive element. Let the primitive element in GF(2^m) be α, and as discussed earlier, every nonzero element of the field can be constructed from the powers of α. Note that there are $\alpha^{(2\hat{}m)-1}$ nonzero unique consecutive powers of α that will result in all the nonzero elements in the field.

There needs to be a limitation introduced to identify $\alpha^{(2\hat{}m)-1}$ unique elements since the higher orders are irrelevant and give repeats of the existing elements. For this purpose, the concept of irreducible primitive polynomials is introduced. Similar to the prime number sized-finite fields, where it was comfortable to use the mod p arithmetic, extension fields use the irreducible primitive polynomial. It is necessary to understand the polynomial representation of the elements of extension fields before discussing the relevance of irreducible primitive polynomials.

3.7 Polynomial Representations of Field Elements

The introduction of polynomial representation into field arithmetic gives the necessary understanding of the concept since it helps in appreciating the algebra, and in understanding the implementation process.

Any element of the field GF(2^m), α^i where i takes values from 0 to $2^m - 1$, can be represented as a polynomial of degree $m - 1$ as follows:

$$P_{ext}(X) = p_{0,i} + p_{1,i}(X) + p_{2,i}(X^2) + p_{3,i}(X^3) + \ldots + p_{m-1,i}(X^{m-1}) \text{ for } \alpha^{i\,th} \text{ element} \quad (3.10)$$

The first subscript represents the coefficients and the second subscript denotes the i^{th} element in the GF(2^m) field. Note that the coefficients are all from GF(2) and hence during addition and multiplication, mod 2 arithmetic is applied to the coefficients to prevent overflow.

An aid to understand the coefficients of the polynomial for extension fields is to consider the elements in extension fields as words of bits. For example, elements from GF(2^3) can be thought of word of 3 bits with the aid of polynomial representation. The construction of these 3 bit words is based upon the primitive polynomial that defines a field.

3.8 Primitive Polynomial

Once there is a polynomial-based representation, there is a need for an irreducible polynomial to define a field and assist arithmetic operations. Such an irreducible polynomial used to define a field is called the primitive polynomial. The necessary condition for a polynomial to be primitive in GF(2^m) is that it should be irreducible, polynomial of degree m, and the quotient when it divides $X^{2\wedge n}+1$ should be less than $2m - 1$. If α is the root of the primitive polynomial then it is called the primitive element of the field. α is used in construction of fields, as seen in the later sections.

A primitive polynomial is an irreducible polynomial and assures that the α^i nonzero unique elements are well represented with the polynomial representation. This ensures that the higher powers of α are wrapped into the existing field elements. Moreover, it needs to be of degree $m - 1$, since the polynomial representations of all 2^{m-1} elements need a degree of $m - 1$. For the consecutive nonzero α^i elements to exist:

$$\alpha^{(2^{\wedge}m)-1} + 1 = 0 \qquad (3.11)$$

This ensures that higher powers of α are folded back into the field, for example:

$$\alpha^{(2^{\wedge}m)+10} = \alpha^{((2^{\wedge}m)-1)+11} = (\alpha^{(2^{\wedge}m)-1})\,(\alpha^{11}) = (1)(\alpha^{11}) = \alpha^{11} \text{ when } 2^m - 1 > 11.$$

Primitive polynomials can be represented using a linear feedback shift register (LFSR) as shown in Figure 3.1. The circuit shown is an LFSR representation of polynomial $1 + X + X^4$. It is to be noted that only the non- zero coefficients have taps or feedback.

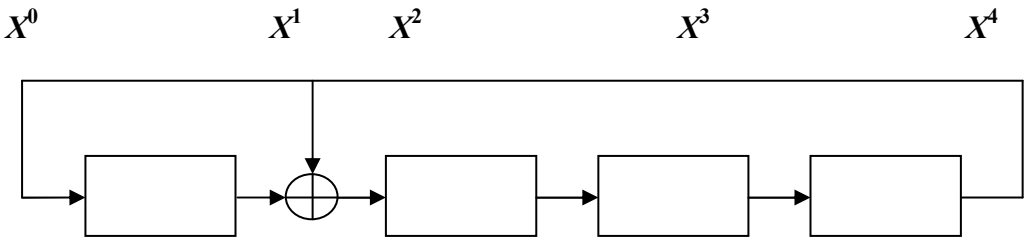

FIG. 3.1 Example of an LFSR Circuit

3.9 Addition in Extension Fields

By representing the codewords as polynomials, addition in GF(2^n) field can be defined as the mod 2 sum of the polynomials. For each k^{th} power of X, the coefficients corresponding to the same power are added up and the mod 2 arithmetic is applied to the result. The following represents the polynomial addition:

$$\alpha^a + \alpha^b = \{p_{0,a} + (p_{1,a})X + (p_{2,a})X^2 + .. + (p_{m-1,a})X^{m-1}\} + \{p_{0,b} + (p_{1,b})X + (p_{2,b})X^2 + .. + (p_{m-1,b})X^{m-1}\}$$

$$= (p_{0,a} + p_{0,b}) \bmod 2 + \{(p_{1,a} + p_{1,b}) \bmod 2\}(X) + .. + \{(p_{m-1,a} + p_{m-1,b}) \bmod 2\}(X^{m-1})$$

3.10 Multiplication and Exponential Representation

Elements of the finite field can also be represented as exponential representation because they assist in the multiplication arithmetic. For example consider GF(8) with a primitive polynomial: X^3+X+1. Assuming α as the root to the primitive polynomial, we have $\alpha^3 + \alpha + 1 = 0$ which gives $\alpha^3 = \alpha + 1$. This holds since the + and – operations in mod 2 arithmetic can be interchanged.

The primitive element is represented as α. The nonzero elements of the field GF(8) can be constructed as follows:

$$\boldsymbol{\alpha^0} = 1;\ \boldsymbol{\alpha^1} = \alpha;\ \boldsymbol{\alpha^2} = \alpha^2;\ \boldsymbol{\alpha^3} = \alpha + 1;\ \boldsymbol{\alpha^4} = \alpha(\alpha^3) = \alpha(\alpha+1) = \alpha^2 + \alpha;$$

$$\boldsymbol{\alpha^5} = \alpha^2(\alpha+1) = \alpha^3 + \alpha^2 = \alpha^2 + \alpha + 1;\ \boldsymbol{\alpha^6} = \alpha(\alpha^2+\alpha+1) = \alpha^3 + \alpha^2 + \alpha = (\alpha+1) + \alpha^2 + \alpha = \alpha^2 + 1;$$

$$\boldsymbol{\alpha^7} = \alpha(\alpha^2+1) = \alpha^3 + \alpha = (\alpha+1) + \alpha = 1 = \boldsymbol{\alpha^0}$$

The α^7 folds back to α^0, and hence the above are the nonzero elements of the field. Since 0 is also an element in the field, there are a total of 7+1 = 8 elements in the field as expected.

3.11 The Use of Finite Field Arithmetic in Reed-Solomon Coding

The finite field arithmetic introduced serves as the mathematical background in the construction of Reed-Solomon codes. The finite field selected in construction of Reed-Solomon codes plays a role in the error correction capability. The need for arithmetic in finite fields is obvious since the codeword is assumed to have errors before it is sent to the decoder.

The finite field arithmetic and the construction of the Reed-Solomon codes will be introduced in the following chapter.

CHAPTER 4

REED-SOLOMON CODING

4.1 Introduction

The original paper about polynomial codes over certain finite fields was a 5-page paper by Reed and Solomon [3] that introduced the concept of Reed-Solomon coding and also a method of encoding and decoding it. This was a breakthrough in coding theory which was associated with a practical implementation idea.

Various encoding techniques for Reed-Solomon coding are discussed, followed by the hardware implementation and complexity trade-offs. Among the different encoding techniques introduced, the systematic generator polynomial approach is given more attention as it is implemented in audio disks.

4.2 Reed-Solomon Coding Setup

Reed-Solomon coding is a block coding technique where the message symbols are coded as message symbol blocks. Each message symbol may be comprised of several bits. The number of bits involved is related to the finite field used for coding. For example, in an audio disk, 8 bits or 1 byte is considered one message symbol. This interprets that the finite field used for the coding in an audio disk is GF(2^8) or GF(256).

A Reed-Solomon code is also represented as RS(n, k), where n and k are two numbers that define the coding scheme. The number k represents the number of data symbols encoded in each codeword, and the number n denotes the resulting codeword length after parity bits have

been added. If RS(n, k) code is applied to code on m-bit long symbols, then the following relation holds:

$$0 < k < n < 2^m + 2 \tag{4.1}$$

Note that the field used for m-bit sequences is GF(2^m) if m is in bits.

The difference between n and k represents the parity bits that have been added,

$$\text{Parity bits added} = 2t = n - k \tag{4.2}$$

In the above equation, t represents the number of symbols that can be corrected by RS(n, k) code. In other words, with $2t$ parity bits added, an RS(n, k) code is capable of correcting t-symbol errors. Note that the error-correcting capability is in symbols, and hence if a symbol is 1 byte long, as in the case of GF(256), then the error-correcting capability is t bytes [4].

Apart from the t-symbol error-correcting capability, the RS(n, k) code can also correct up to 2t erasures. Erasures are errors with locations known. For example, if the 2^{nd} bit is an erasure, we know that it is the 2^{nd} bit that is corrupted from a trusted source; however, we do not know which the correct bit is.

The errors and erasures can be corrected together. This is true as long as the following equation holds:

$$(2 \times \text{number of errors}) + (\text{number of erasures}) \leq 2t \tag{4.3}$$

4.3 Code Rate of RS(*n, k*) Code

As defined earlier the code rate is k / n, where k is data bits encoded and n is the length of the coded word or codeword. Since an RS(n, k) code codes on a k-sized m-bit sequence which results in an n-sized m-bit sequence, the code rate for RS(n, k) is:

$$\text{Code rate} = n / k \tag{4.4}$$

The lower the code rate the more the redundant bits are being added and higher is the error correction capability of the code. As mentioned before, this introduces a trade-off between error correction capability and the code rate.

4.4 RS(*n, k*) Coding and Burst Errors

In Reed-Solomon coding, error correction is based on the symbol of the bits of the codeword. If the errors are contained in one symbol, then the entire symbol containing corrupted bits is replaced with the right symbol. This implies that if there is a burst of errors affecting a symbol of m bits, all the bits are replaced. In other words, if a symbol is corrupted by just one bit, it may as well be wrong in all the bits since a fresh corrected symbol is introduced. This shows the advantage of using RS(n, k) coding to overcome burst errors [5].

For example, consider RS(32, 28) code from GF(256) applied on 8-bit sequences. This is actually a coding scheme used in one of the coding stages in audio disks. Now, consider there is a burst error from first bit in symbol k to last bit in symbol $k + 2$. This means that the burst error lasts for 16 bits. The RS(32, 28) code is capable of correcting 2-byte words since parity bytes added are 4 bytes. Hence, all the corrupted bits are replaced with the correct bits. This

shows the power of RS(*n, k*) codes. This emphasizes the point that RS(*n*, *k*) codes can handle burst errors of such kind.

4.5 RS(*n*, *k*) Coding and Random Errors

Random errors occasionally occur and affect a single bit, or several bits, and hence are usually corrected by a single layer of RS(*n*, *k*) codes.

If the errors are sparse and high in number, e.g. affecting one bit in each and every m symbols, then the Reed-Solomon coding will incorrectly decode the codeword. This is due to the fact that the number of symbol errors crosses the t-symbol error-correcting capability of the decoder.

In an audio disk, the decoder has a protected four levels of RS(*n, k*) codes. An interleave mechanism exists between the layers to use the flags generated by previous RS(*n, k*) decoder as a parity for the next layer. Moreover, the function of the interleave mechanism is to spread out multiple errors in one codeword over many codewords, so that there are fewer symbols in each codeword which are affected by it. This strategy is used to overcome errors which are of burst nature and also are spread over many codewords.

4.6 The Encoding Technique Introduced in the Original Paper

Reed and Solomon introduced the original approach to coding and decoding by treating the message as a polynomial and evaluating the polynomial at each of the points of the field, thus generating the code. If the message is represented as (m_0, m_1, m_2, m_3... m_{n-1}) then the equivalent polynomial representation of the message will be:

$$P_m(X) = m_{0+} (m_1)X + (m_2)X^2 + (m_3)X^3 + \ldots + (m_{n-1})X^{n-1} \quad (4.5)$$

Note that the message symbols are contained in the finite field, GF(q), used for construction of the code. The message symbols will be 1 byte long, 8 bits, if the field GF(256) is considered as in the audio disk.

To encode the data, the message polynomial is evaluated at all the points in the field. In other words, codeword C is formed by the following construction:

$$C = [P_m(0) \quad P_m(\alpha) \quad P_m(\alpha^2) \quad P_m(\alpha^3) \quad \ldots \quad P_m(\alpha^{q-1})], \quad (4.6)$$

where α denotes the primitive element of the field GF(q) [6].

4.7 Encoding Technique Using the GFFT Approach

The Galois field Fourier transform approach (GFFT) is similar to the original approach, however it uses the concept of Fourier transforms. The GFFT applied on the codeword C= ($c_0, c_1, c_2 \ldots c_{n-1}$) is given by:

$$\text{GFFT}(C) = (\text{GFFT}(c_0), \text{GFFT}(c_1), \text{GFFT}(c_2) \ldots \text{GFFT}(c_{n-1})) \quad (4.7)$$

where,

$$\text{GFFT}(c_i) = \sum_{(\text{from } j = 0 \text{ to } n-1)} (c_i)^{ij}, \text{ where } i, j = 0, 1, 2 \ldots, n-1 \quad (4.8)$$

The frequency and time domain concept for the GFFT is unclear as Wicker and Bhargava discuss [6]. The GFFT of a code with 2t consecutive powers of α results in 2t consecutive zeroes, and is hence sometimes considered the dual of the generator polynomial approach.

The generator polynomial approach or the systematic approach is introduced in the following section.

4.8 Encoding Using Systematic Generator Polynomial Approach

RS(n, k) are treated as cyclic codes in this technique. A cyclic code is defined as a code C that produces a codeword, $C_{shifted}$, after arbitrary shifts have been applied to the original codeword, C_{orig}. An alternate way to look at RS(n, k) codes as cyclic codes is to consider all the polynomial codewords as a multiple of a predefined polynomial which is referred to as the generator polynomial.

The predefined polynomial is called the generator polynomial and is often represented as $g(X)$. The generator polynomial also decides the number of parity bits, say $2t$, the newly constructed RS(n, k) code would have. The degree of this polynomial equals the number of parity bits, or $2t$. This generator polynomial is constructed from the roots of the finite field GF(2^m) where the RS(n, k) code is defined. The roots of the polynomial are hence $2t$ consecutive powers of the primitive element α. The generator polynomial can be represented as:

$$g(X) = (X + \alpha)\,(X + \alpha^2)\,(X + \alpha^3)\,\ldots..\,(X + \alpha^{2t}) \qquad (4.9)$$

$$g(X) = g_0 + g_1X + g_2X^2 + \ldots.. + g_{2t-1}X^{2t-1} + X^{2t} \qquad (4.10)$$

where g_i, $0 < i < 2t$, are obtained from the finite field GF(2^m).

4.8.1 The Encoding Procedure:

The encoding process consists of several major steps. The first step is left shifting the data polynomial by some degree and then doing modular division by the generator polynomial. This encodes the data as a multiple of the generator polynomial. To understand the procedure, the encoding process is broken into two steps.

4.8.1.1 Left Shift of the Data Polynomial

If the data polynomial is represented as m(X) has degree k, the message polynomial m(X) has to be left shifted by $2t$ places or $n - k$ places where n denotes the length of the generated codeword. This can be represented by:

$$m_{new}(X) = X^{2t}\ m(X) \tag{4.11}$$

4.8.1.2 Modular Division by Generator Polynomial

After the polynomial $m_{new}(X)$ is obtained, the modular division on the generator polynomial, or $g(X)$, is performed. This ensures that the residue polynomial represents the parity polynomial or the redundant polynomial. This action is represented as:

$$C(X) = m_{new}(X) \bmod g(X) + p(X) \tag{4.12}$$

The $C(X)$ generated is a codeword of the cyclic RS(n, k) code which is a multiple of the generator polynomial. The message embedded in the multiple of $g(X)$. The set of all such $C(X)$ is the Reed-Solomon code.

The hardware implementation of the generator polynomial approach is accomplished by using an LFSR circuit. Note that the division applied is more complicated since it needs to be

performed according to the finite field arithmetic. Hence, the LFSR circuit operators carry out the arithmetic based on the finite field chosen for coding.

4.9 Decoding Using Systematic Generator Polynomial Approach

Since the major focus of the thesis is the decoding process, it will be discussed in greater detail. The decoding process is a little complicated since we receive a corrupted codeword. Several algorithms are involved to identify error locations and error values.

If the codeword when received, or when reading off the audio disk, does not contain any error, then the decoding procedure is straightforward. The decoder looks up the codeword dictionary and matches the values and decodes it to the original message polynomial. However, the transmission need not be error free, and the effect of errors must be considered regardless. Consider the received corrupted codeword as:

$$R_x C(X) = C(X) + errors(X) \tag{4.13}$$

Note that the $errors(X)$ denote the polynomial representation of the errors introduced in the codeword. Hence, $errors(X)$ can be of any degree up to $n-1$. This polynomial $errors(X)$ is unknown to the decoder since it only sees $R_x C(X)$. The solution to finding $C(X)$ is equivalent to finding the error polynomial $errors(X)$, since it can be subtracted from $R_x C(X)$ to obtain $C(X)$.

The decoding procedure is listed in the following sections.

4.9.1 Syndrome Check

In order to see if the received codeword is error free or not, it is necessary to carry out a simple test. Note that the codeword is a multiple of the generator polynomial. Hence, the 2t consecutive powers of α should be solutions to the polynomial representing the codeword. This property is used and the 2t evaluated computations are called syndromes.

$$S_{yndrome,\ k} = R_x C\ (\alpha^k), \tag{4.14}$$

where k = 1, 2, 3.., 2t for all 2t consecutive powers of α.

If the received codeword $R_x C(X)$ is error free then it equals any one codeword $C(X)$. Since $C(X)$ is a multiple of $g(X)$, one can conclude that all the 2t powers of α are roots to $C(X)$. Hence, for a valid codeword, or an error free codeword, the following will hold:

$$\text{Syndrome values, } S_{yndrome,k} = 0,\ \forall k \text{ in Code } C \tag{4.15}$$

4.9.2 Computation of Error Locations

If the syndrome check process results in nonzero outputs for at least one α^i, then there exist errors. Consider that the number of errors is less than or equal to t, since the RS(n, k) code has t-error correction capability.

The syndromes evaluated earlier $S_{yndrome,\ k}$ will be used to evaluate the error locations. The 2t $S_{yndrome,\ k}$ values can be represented as system of equations as shown:

$$S_{yndrome,\ 1} = R_x C\ (\alpha^1) = e_{l1}\ \alpha^{l1} + e_{l2}\ \alpha^{l2} + \ldots. + e_{lz}\ \alpha^{lz} \tag{4.16}$$

In the above equation, e_{li}, represents the error values, and l_i represents the actual locations of the errors.

The same representation is used to describe the other $2t - 1$ equations. It is to be noted that the RS(n, k) error correction code is assumed to be t-symbol error-correcting, and hence we limit to $2t$ parity bits.

Similarly, we can represent the other $2t - 1$ equations:

$$S_{yndrome,\ 2} = R_x C\ (\alpha^2) = e_{l1}\ (\alpha^{l1})^2 + e_{l2}\ (\alpha^{l2})^2 + \ldots + e_{lz}\ (\alpha^{lz})^2$$

$$S_{yndrome,\ 3} = R_x C\ (\alpha^3) = e_{l1}\ (\alpha^{l1})^3 + e_{l2}\ (\alpha^{l2})^3 + \ldots + e_{lz}\ (\alpha^{lz})^3$$

$$S_{yndrome,\ 2t} = R_x C\ (\alpha^{2t}) = e_{l1}\ (\alpha^{l1})^{2t} + e_{l2}\ (\alpha^{l2})^{2t} + \ldots + e_{lz}\ (\alpha^{lz})^{2t} \qquad (4.17)$$

Note, that in the above system of equations, the powers of α are unknown; hence, this system of equations is non-linear. There are several available algorithms for solving these equations. The two most prominent ones are:

1) Euclidean algorithm
2) Berlekamp-Massey algorithm

Each of these methods is discussed in the following sections.

4.9.2.1 Euclidean Algorithm

The Euclidean algorithm is a powerful algorithm in evaluating the polynomial representing the error locations. It is based on finding the greatest common divisor of two polynomials.

The greatest common divisor, GCD(*a, b*), of two numbers *a* and *b* is the largest number that divides both *a* and *b*.

The Euclidean algorithm is based on the Bezouts identity which says that the GCD(*a, b*) between two numbers is unchanged if the smaller number is subtracted from the larger and GCD(*a, b*) is computed on them. Since there are usually multiple subtraction steps involved, the Euclidean algorithm uses the fact that if the larger number, say *b*, were to be divided by the smaller, say *a*, then the GCD(*a, b*) can be obtained by simply computing GCD(*a* mod *b*, *b*). This can be repeated iteratively until one of the numbers is reduced to 0; then we obtain the GCD(*a*, *b*) which is the other nonzero number.

To appreciate the Euclidean algorithm applied in the decoding procedure, it is important to understand the logic of the algorithm.

Consider two numbers, say 42 and 144, and we wish to determine their GCD(42,144) by the Euclidean algorithm. Applying the Euclidean algorithm:

$$\text{GCD}\ (144, 42) = \text{GCD}(144 \bmod 42, 42) = \text{GCD}(18, 42) = \text{GCD}(42, 18)$$

On the next iteration:

$$\text{GCD}(42, 18) = \text{GCD}(42 \bmod 18, 18) = \text{GCD}(6, 18) = \text{GCD}(18 \bmod 6, 6) = 0$$

Hence, the GCD(144, 42) = 6.

The Euclidean algorithm for polynomials is represented as following:

$$G(X) = a(X)\, g_a(X) + b(X)\, g_b(X) \qquad (4.18)$$

The polynomial $G(X)$ does not necessarily represent the GCD($a(X)$, $b(X)$). The Euclidean algorithm terminates when the degree of $G(X)$ is less than t. Note that the Euclidean algorithm also gives the values $g_a(X)$ and $g_b(X)$; hence the algorithm is also termed as the extended Euclidean algorithm.

The Euclidean algorithm is applied in the decoding procedure not to find the GCD($a(X)$,$b(X)$) but to evaluate the common factor polynomial which represents the error locations. The Euclidean algorithm is applied on X^{2t} and polynomial $S(X)$, where $S(X)$ is defined as:

$$S(X) = (S_{yndrome,\ 1}) + (S_{yndrome,\ 2})\, X + \ldots + (S_{yndrome,\ 2t})\, X^{2\text{-}1} \qquad (4.19)$$

We wish to compute the Euclidean algorithm on (X^{2t}, $S(X)$), and this can be represented as:

$$\Lambda\ (X) = S(X)\ \Delta(X) + X^{2\text{t}}\ \partial(X) \qquad (4.20)$$

In the above equation, $\Delta(X)$ will result in the polynomial which represents the error locations. The Euclidean algorithm terminates when the polynomial $\Delta(X)$ obtained in the current iteration reaches a degree less than t.

First iteration ($a(X) = S(X)$, $b(X) = X^{2t}$):

$$X^{2\text{t}} = k_1(X)\ S(X) + \Delta_1(X) \qquad (4.21)$$

If degree ($\Delta_1(X)$) less than t, then the algorithm terminates with $\Delta(X) = \Delta_1(X)$, otherwise we need more divisions.

Second iteration ($a(X) = \Delta_1(X)$, $b(X) = S(X)$):

$$S(X) = k_1(X)\ \Delta_1(X) + \Delta_2(X) \qquad (4.22)$$

Now, if the degree of ($\Delta_2(X)$) less than t, then the algorithm terminates with $\Delta(X) = \Delta_1(X)\Delta_2(X) + 1$. And the iteration continues.

Terminating condition: The algorithm terminates when $\Delta_i(X)$, for i^{th} step, has degree less than t. As an alternative, the algorithm terminates when $\Delta_i(X) = 0$ or $k_{i-1}(X) = 0$, for i^{th} step.

Final Return result:

$$\Delta_i(X) = \Delta_i(X)\ \Delta_{i-1}(X) + \Delta_{i-2}(X) \qquad (4.23)$$

Thus, the previous two iteration results must be stored in the available memory space.

It is to be noted that the degree t significance in deciding the termination of the Euclidean algorithm comes from the fact that the number of errors affecting the codeword is assumed to be less than t. It should be noted that if there were to be more than t errors, the algorithm could result in the incorrect $\Delta(X)$ polynomial and hence incorrect decoding.

Pseudocode or Outline of the Euclidean Algorithm:

The Euclidean algorithm with inputs $S(X)$ and X^{2t} can be written as:

ECD(X^{2t}, $S(X)$) {

The first step involves the long division of X^{2t} by $S(X)$, which is represented as:

$X^{2t} = S(X)\ q_1(X) + r_1(X)$

If (degree of $r_1(X) \leq t$) {

then the quotient polynomial $q_1(X)$ will be our required error location polynomial.

Return answer ($q_1(X)$) }

Else {

Compute $\Delta(X)$ and $\Delta_i(X)$

Store the result value in allotted heap space.

Evaluate ECD ($q_1(X)$, $r_1(X)$), or the recursive call.}

} // closure parenthesis for the function.

Final value = $\Delta(X) = \Delta_i(X)\ \Delta_{i-1}(X) + \Delta_{i-2}(X)$, and hence consider the last three memory allocations only.

Computation of $\Delta(X)$ requires memory storage for previous two $\Delta_i(X)$ computations.

For example:

In the first call: $\Delta_1(X) = q_1(X)$.

In the second recursive call: $\Delta_2(X) = q_2(X)\ \Delta_1(X) + 1$,

In the third recursive call: $\Delta_3(X) = q_3(X)\ \Delta_2(X) + \Delta_1(X)$ and so on.

Note that $\Delta(X)$ contains error locations as its solutions. Hence, $\Delta(X)$ is the intended result and not GCD(X^{2t}, $S(X)$).

4.9.2.2 Berlekamp-MasseyAlgorithm

As an alternative to the Euclidean algorithm for finding the error locations, the Berlekamp-Massey algorithm is used. The Berlekamp-Massey algorithm has lower time complexity because it does not involve polynomial divisions like the Euclidean algorithm; however, the trade-off is the mathematical complexity involved. This makes Berlekamp-Massey algorithm efficient to implement, and therefore it is used in the audio disks. The simulation results obtained in this thesis are based on the Berlekamp-Massey algorithm.

Following are the steps involved in the Berlekamp-Massey algorithm:

1) Transformation to linear equations:

The set of equations for syndromes (4.17) can be represented as a single equation given as:

$$\mathrm{BM}(X) = (1 + k_1X)\,(1 + k_2X)\,(1 + k_3X)\ldots\,(1+k_lX) \tag{4.24}$$

where each k_i denotes the inverse of the error locations represented in the set of syndrome equations. In other words, the roots of the equation BM(X) represent the inverse of the locations of the errors.

2) Finding the minimal degree polynomial satisfying the Newton identities:

The Newton identities are a set of linear equations which give the relation between the syndromes and the roots of the BM(X) polynomial. The set of equations which represents the Newton identities are:

$$S_{syndrome,\,1} + k_1 = 0$$

$$S_{syndrome,\ 2} + k_2(S_{syndrome,\ 1}) = 0$$

$$S_{syndrome,\ l+1} + k_1(S_{syndrome,\ l}) + \ldots + k_l(S_{syndrome,\ 1}) = 0 \qquad (4.25)$$

The minimal degree polynomial is the polynomial with the smallest degree that satisfies this set of equations. This polynomial when obtained will give the inverse of the locations of the errors [7].

4.9.3 Chien Search

The Euclidean algorithm and the Berlekamp-Massey algorithm converge at this point in the process of decoding. Each of them results in a polynomial which has roots as the inverse of the locations of errors.

The Chien search is an effective way to find the roots of this polynomial. Note that the roots must belong to the field, since the residue error polynomial is a reduced form of the syndrome polynomial.

In equation (4,16), the $e_{i,\ j}$ for the i^{th} syndrome is detached using the Berlekamp-Massey or the Euclidean algorithms. In other words, the error polynomial has information only about the error locations; however, the characteristic of field elements as roots is not lost. Hence, an exhaustive search of all the elements of the finite field must be made. Chien search efficiently searches through the different polynomials of the field and results in the roots [7].

The error polynomial, as described before, can be of any degree up to $2t$, and the roots give information about the error locations. The roots do not give us the error locations in direct form, and there is some evaluation involved in computing them.

The k^{th} element which satisfies the error polynomial, or $\Delta(\alpha^k) = 0$, results in the information of error location. The exponent of the inverse of α^k results in the location of the error.

4.9.4 Forney Algorithm

The Forney algorithm involves finding the error values and hence concluding the decoding process. The effort of correction is complete only after this algorithm is complete. The Forney algorithm computes the error values by computing the ratio of the $\Lambda(X)$ to derivative of the $\Delta(X)$, where each of the polynomials is obtained during the Berlekamp-Massey algorithm, or alternatively the Euclidean algorithm. After calculating the ratio, we obtain the polynomial $F_\Delta(X)$. Now, $F_\Delta(X)$ is evaluated at certain points. These points represent the solutions to the error location polynomial obtained during the Chien search stage [8].

4.9.5 Error Correction Stage

Once the error locations and error values are obtained, these are added to the existing polynomial. Note that $R_xC(X) = C(X) + errors(X)$, and hence the errors can be eradicated. The addition and XOR are identical in mod 2 arithmetic, and hence a simple XOR with the $R_xC(X)$ polynomial will result in the correct codeword, $C(X)$.

Once the correct codeword is obtained, a mapping lookup will result in the corresponding message symbol.

CHAPTER 5

REED-SOLOMON CODING IN AUDIO DISKS

5.1 Introduction

To understand the various levels of RS(*n, k*) coding applied to the audio disk, it is first necessary to look at the structure of the audio disk. The fundamental unit of an audio disk is a sector. A single sector consists of 98 frames. A single frame holds the data values and progressively grows bigger as the RS(*n, k*) code is applied to it. In other words, the frame is initially 24 bytes long and as the RS(*n, k*) coded layers are added, it increases to 28 due to RS(28, 24) and further to 32 due to RS(32, 28).

The coding process also involves coding at the sector layer and at a lower layer among frames. If a sector has a high number of corrupted byte words such that the RS(*n, k*) coded layers together cannot resolve them, then the sector is banned. When a sector is banned, all the data written in the sector is lost even if some of the frames were properly decoded. This is a major drawback in the audio disk codec systems. The feedback system introduced later helps in avoiding such a banning of the sector.

5.2 Incorrect Decoding in Audio Disks

One of the major concerns of RS(*n, k*) coding is incorrect decoding. If the codeword contains more errors than the correction capability of the RS(*n, k*) code, then the code incorrectly decodes them rather than alerting that there are many uncorrectable errors.

This behavior of RS(*n, k*) coding of incorrectly decoding corrupted codewords introduces inefficiency. The performance degrades and the reliability of the systems is also reduced. Hence, it is desirable to have some kind of indication of when the RS(*n, k*) code fails to decode and when it doesn't fail to decode.

In an audio disk, there exist flag values dedicated to indicate if errors are corrected or if uncorrected. Each level of RS(*n, k*) code is equipped with flag values. In addition to giving information about the decoding process, these flags also provide assistance for further layers of RS(*n, k*) coding by indicating erasures.

5.3 RS(*n, k*) Encoding Procedure in Audio Disks

Data in audio disks is protected by two layers, while in CD-ROMs by four layers of RS(*n, k*) coding. In CD-ROMs, the first two layers are applied as product codes in the sector encoding layer. The next two layers are added during the CIRC (cross interleaved Reed-Solomon coding) encoding stage. Figure 5.1 shows the various layers of coding applied as a block diagram. The CIRC coding is a powerful coding strategy and is highly applicable in communication systems. The Reed-Solomon coding layers accompanied with the interleaving make the error-correcting codes stronger. The CIRC encoding layer is focused upon in this paper for the feedback strategy, which is later discussed. The encoding procedure consists of the following stages mentioned in the sections that follow.

5.3.1 Sector Encoding Stage

The Sector encoding includes the Reed-Solomon product codes. Initially CRC and control bits are added to keep a check of the data bits. The sector is divided into two subsectors,

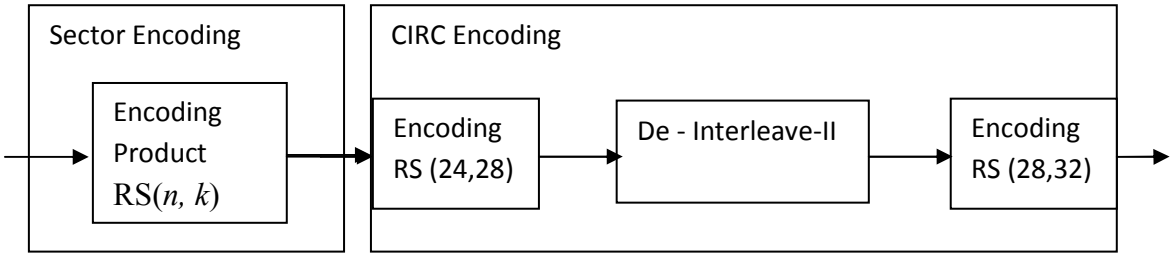

FIG. 5.1 Encoding Procedure in Audio Disks.

which are coded using the Reed-Solomon product codes. The Reed-Solomon product codes applied are RS(45, 43) and RS(26, 24) which are applied rowwise and columnwise respectively. The product codes are applied at a higher sector level, and the focus is more on frames rather than on the actual data, which are stored in their respective codeword versions.

After the sector encoding, the sector level coded data is passed through an interleaver, Interleaver I, which interleaves alternate frames of sectors. The interleaving involved in CIRC, as seen later, is involved at a frame level.

5.3.2 CIRC Encoding Stage

The CIRC encoding stage is at the frame level. There are 98 frames in each sector and each of them is coded with the RS(n, k) codes that are involved in CIRC.

There are two RS(n, k) with an interleaver applied between them. The first Reed-Solomon code, *C1*, is RS(28, 24) applied to the 24 bytes in each frame. Since the words are at a byte level, the value of m = 8. The length of datawords, k, is 24 and the length of the codeword generated, n, is 28. *C1* can correct up to 2 corrupted words, which are each 1 byte long. Since after coding n = 28, the resulting codeword is 28 words in length. This implies that each frame after *C1* encoding contains 28 byte-words.

The next stage in CIRC is the Interleaver II stage, which unlike Interleaver I, is applied over all the 98 frames. This Interleaver II delays each word of each frame by $4(n - 1)$ frames. In other words, the first word in the first frame is delayed by zero frames, but the second word is delayed by four frames. Hence, after passing through this interleaver the number of words in each frame is still 28, but the words contained in each frame are shuffled with other frames.

The second Reed-Solomon code, *C2*, is RS(32, 28) applied to the 28 bytes in each frame. This coding stage is similar to *C1*, but the datawords here are $k = 28$ in length, and the resulting codeword length is $n = 32$. *C2* encoding and *C1* encoding each add four parity byte-words during their encoding stages. Figure 5.2 shows the CIRC block diagram as applied in audio disks.

The last stage of the CIRC is the Interleaver III that is applied on each of the 98 frames. This interleaver delays each word in each frame by one frame. The Interleaver III marks the end of the CIRC encoding stage [9].

There are extra control, parity and synchronization bytes added during the encoding stages. However, the focus of the feedback system is mainly on the CIRC encoding stage.

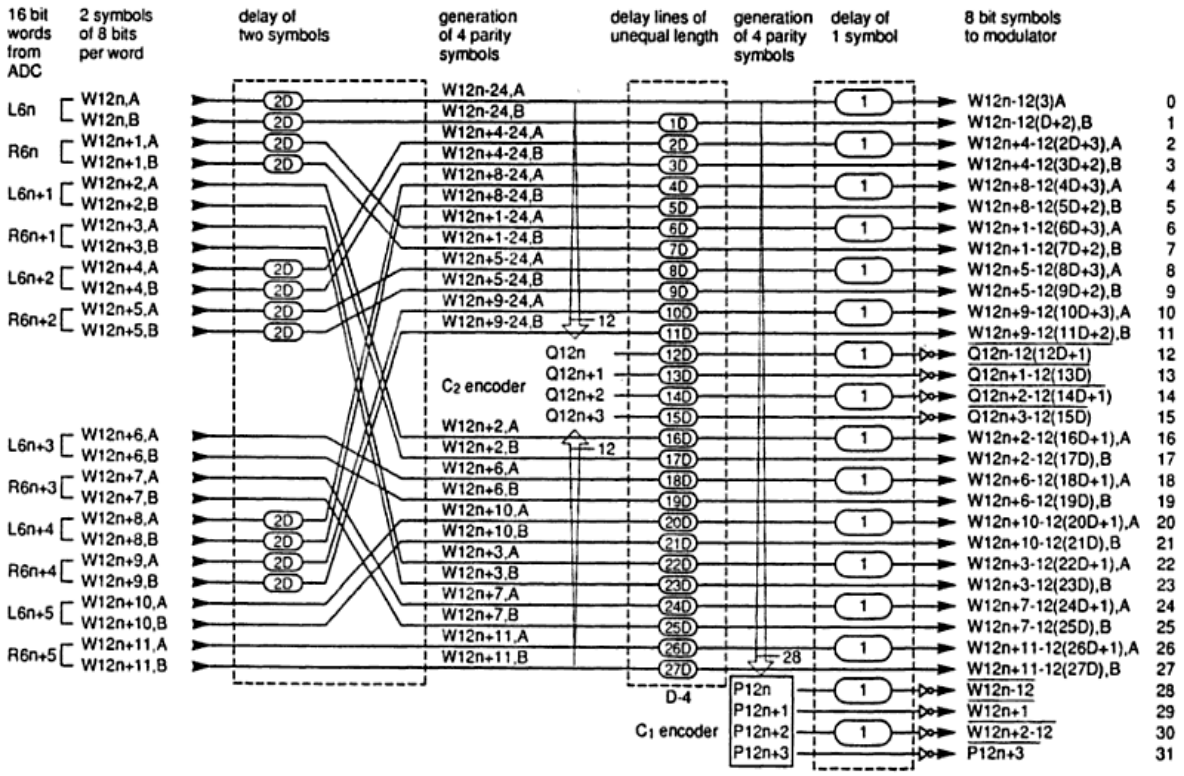

FIG. 5.2 CIRC Block Diagram by K.S. Immink [2].

5.4 RS(*n, k*) Decoding Procedure in Audio Disks

The decoding stage is the reverse of the encoding stage. The data that is coded and interleaved must be retrieved only by sequential reverse process. Hence, first the CIRC encoding stage is completed followed by the sector encoding stage.

In the CIRC stage, the *C2* decoding stage conveys additional information to the *C1* decoding stage. It is to be noted that since the decoding process is the sequential reverse of encoding, the *C2* Reed-Solomon decoding is performed first prior to *C1*. It is in the process of decoding that the functionalities of the two layers of coding and the interleaver are observed.

When the *C2* decoding process is complete, the errors corrected are stored in a flag register, called *C2-flag*. The *C2-flag* conveys information about the number of errors that were corrected, or if the codeword had errors in more than four words. This flag is associated with each of the bytewords in each frame, even though the decoding process is applied to the entire frame. After the De-interleaver II is applied, the words are shuffled back and are ready for *C1* decoding. However, since the *C2-flag* is associated with each byteword, this additional information is available before *C1* decoding begins. This *C2-flag* serves as the additional information in the decoding stage of CIRC. It specifies the errors in locations and hence provides the information on erasures. Any Reed-Solomon code can correct *t* erasures and *t*/2 errors where *t* denotes the parity words added during encoding stage. Since the *C1* is a RS(28, 24) code, there are four parity bytewords added; hence, *t* is four bytes. Since the *C2-flag* is available during the *C1* decoding stage, it is possible to correct four words of erasures and two word errors. This correction boosts the error correction capability of *C1* due to both *C2* and the interleaver.

However, it is important to note that erasure information is unavailable to the *C2* Reed-Solomon code. *C2* can correct up to only *t*/2 errors where *t* denotes the parity bytes added. In the *C1* code, $t = 4$, and hence can correct two error words.

This thesis focuses on improving the *C2* code and thereby improving the overall performance of the system. After the *C1* code has been decoded, it could serve as extra information by providing the flag value back to *C2*. If the flag value is provided back to *C2,* the erasures will also be corrected in the *C2* decoding stage. This feedback information, which involves a couple of encoding steps during the feedback, will boost the performance further.

5.5 Importance of Interleaving

It is important to mention the role of interleaving in CIRC encoding. Though the Reed-Solomon codes correct errors, the interleaver plays a lead role in assisting them. It spreads the errors out, thus converting a burst error into errors spread across all frames. As an example, consider a situation where two to three frames are completely corrupted due to an error burst. Instead of losing the entire data in those frames, the interleaver spreads the errors into other uncorrupted frames. As a result, the errors are shared among several frames. Now, the Reed-Solomon code corrects these errors by visiting each frame.

As mentioned before, CIRC consists of two Reed-Solomon codes separated by an interleaver. The interleaver spreads the error over different frames. A good analogy is that of incorrectly spelled words in a book where each page of the book represents the frames. Now, since the words on a page are incorrectly spelled, the Reed-Solomon tries to correct them based on its capability. It is known that if the number of corrupted words are higher in number than (parity bits)/2, then the Reed-Solomon code incorrectly decodes the message. Hence, it is important to spread out this error over multiple pages such that on each page, there are fewer incorrectly spelt words. After the words are spread out over multiple pages, since on each page the incorrect words are now less than (parity bits)/2, the Reed-Solomon code successfully corrects. This analogy shows the advantage of the interleaver in spreading out the error and hence reducing the average error on each frame. This reduction in average error is so designed that it falls well below the Reed-Solomon error-correcting capability.

5.6 Simulation of Encoding and Decoding Processes

The simulation (see Figure 5.3) of the CIRC encoding and decoding process was successfully performed in MATLAB. MATLAB has a built-in Reed-Solomon encoder and decoder and is used in the simulation. The Interleaver II involved more work since it spreads out the code through a standard procedure.

Initially the encoder and decoder functionalities were checked in order to see if the CIRC is performing right. For this, a random generated message was written and encoded with the MATLAB-coded CIRC encoder. This was later retrieved from decoding which resulted in the same message. Hence, the correct functionality of the CIRC was verified.

The next step was to introduce burst errors. Burst errors were introduced from frame 29 and progressively increased in size. The plot of the number of errors versus the errors that were uncorrected was observed. Note that the plot shown described the uncorrected errors after passing through the CIRC.

The burst errors were introduced and incremented in each loop. As soon as the number of bytes reaches a multiple of 28, the entire frame is corrupted and the iteration is progressed to the next frame. In this way, multiple frames are affected by the burst errors. The simulation results can be observed in Figure 5.3.

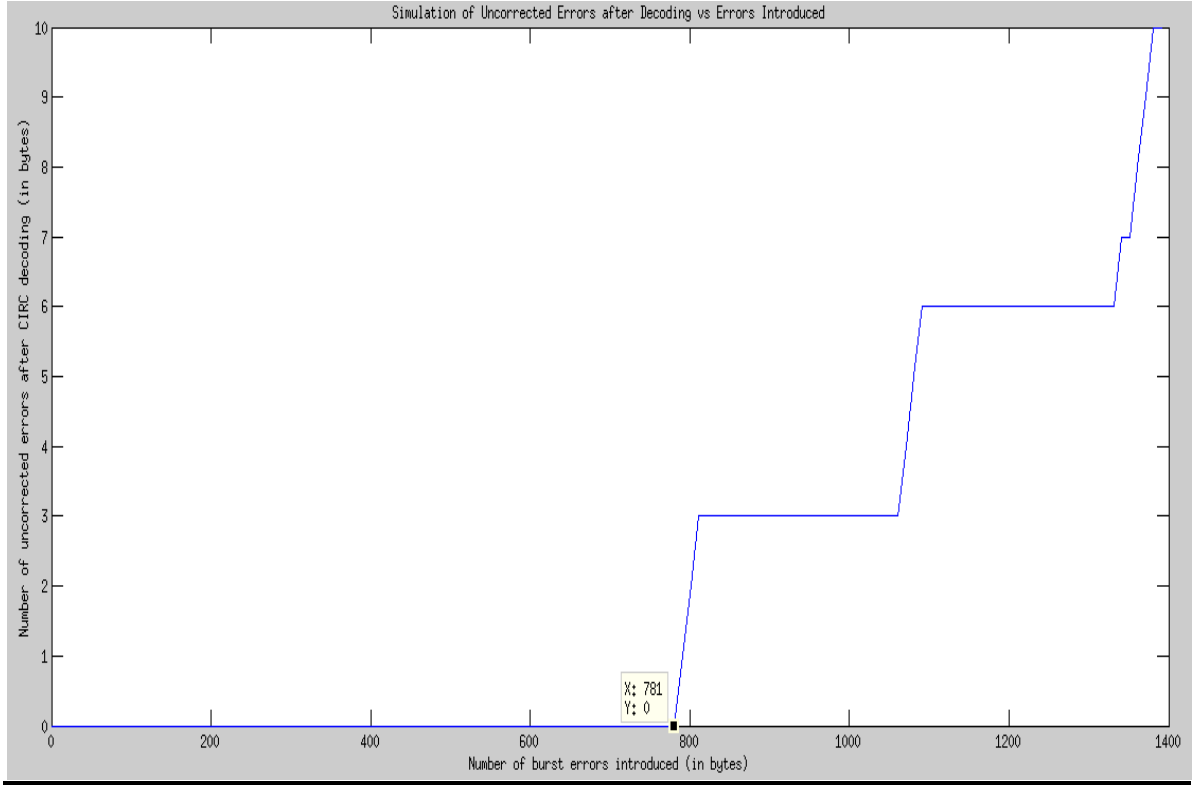

FIG. 5.3 Simulation of the CIRC Used in Audio Disks.

5.7 Observations from Simulation

Observation 1: All the errors are corrected when the error burst affects up to 781 bytes. This shows the advantage of having multiple layers of Reed-Solomon coding equipped with interleavers. Figure 5.4 gives a visual representation of frames affected by burst errors. The dark shaded region represents the corrupted bytewords. Figure 5.5 represents the affect of the Interleaver II applied on the set of frames.

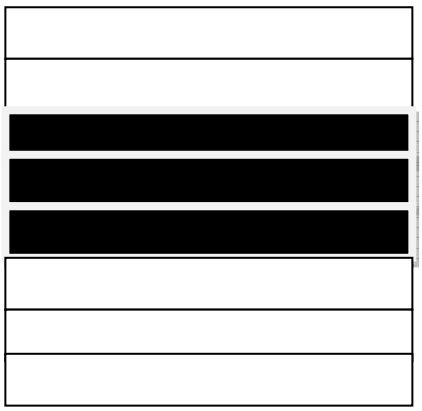

FIG. 5.4 Burst Errors Introduced before Decoding Begins.

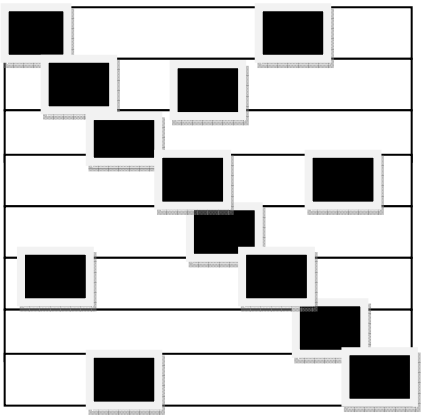

FIG. 5.5 Burst Errors after Interleaver II is Applied

In Figure 5.4, the burst errors are introduced into the frames just as they are introduced during simulation. After one layer of Reed-Solomon coding, Interleaver II spreads out all the bytes across frames. This ensures that the errors are not coagulated together, and each frame is corrected independently with fewer errors. This is a big contributor for boosting the error correction performance in audio disks.

Observation 2: Beyond 800 bytes of burst errors introduced, the CIRC fails to correct all errors. This usually occurs when the CD is affected by deep scratches concentrated in an area. However, it is to be observed that the uncorrected errors are not high. The uncorrected errors range from 1-10 bytes for 800-1400 bytes of burst errors introduced. This shows that a significant amount of errors are corrected; however, the audio disk system bans the sector due to uncorrected errors. Observation 2 suggests that the corrected errors go wasted if the sector is banned. The error correction capability of the CIRC system is not made complete use of. This is one of the motivations in having a feedback system for the audio disk system.

The feedback system makes an effort to enhance the error correction performance which will be discussed in further detail in the following chapter.

CHAPTER 6

THE FEEDBACK SYSTEM FOR DECODING

6.1 Introduction and Motivation

After successive decoding stages, if the errors are such that they persist, then the data cannot be retrieved. In such cases, the sector is banned since the data is corrupted and cannot be recovered. If the data represents a music file, this missing data can be noticed when the music is being played because of a click or a jump during the playback of the song. If the data represents a part of the file, then the entire file may be lost since some of the data is corrupted.

If a sector is banned, a significant amount of data might have been corrected during different decoding stages. Though there was successful decoding on some of the frames during each stage, the overall outcome neglects them. This results in inefficient use of the performance of decoders from each stage.

Secondly, there is always some inefficiency in the system since the final coding layers cannot communicate with the previous layers. There is one-sided assistance from previous to next layers which is represented by the check flags and the interleaving.

6.2 The Feedback System - Process

The feedback system introduced [10] uses the decoded information in the later stages of the coding as extra valuable information to the first layer. The flag values, which carry information of codewords correctly decoded or not, can represent erasures. The importance

of the flag values helps in the erasure detection and correction, and this is exploited in the feedback design. The feedback system process begins after the decoded data and the flag values are obtained or after the first cycle of decoding. It is important to note that the input to the later decoding stage, Decoding RS(28, 24), is saved in memory as explained later.

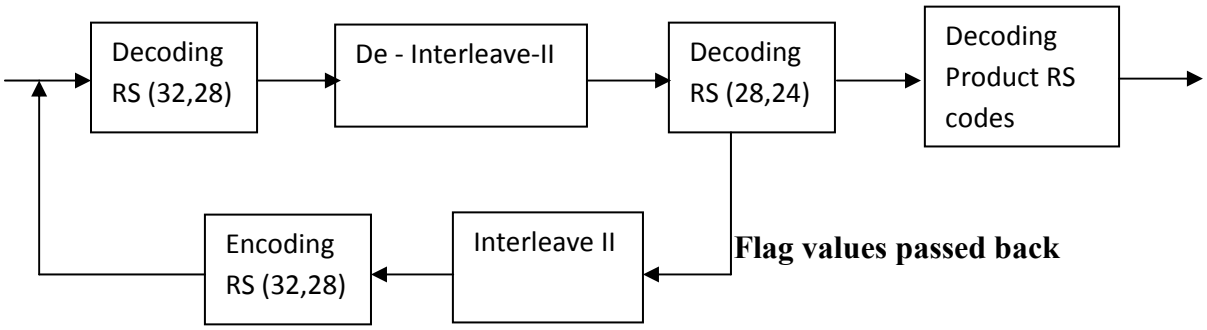

FIG. 6.1 Feedback during CIRC Decoding in Audio Disks

Figure 6.1 shows the feedback as applied to the CIRC. The following steps explain the process:

1) The flag values which were obtained during the last stage of decoding have a one-to-one mapping with the codeword. These flag values are saved in available memory space and can be represented as a 98 length vector as follows:

$$\boldsymbol{F1} = \{f_i\}, \text{ where } i \in \{1, 98\}, \tag{6.1}$$

where f_i gives the number of corrected symbols or is –1 if the symbol was incorrectly decoded.

Also, the codewords used for the final stage decoding RS(28, 24) are saved in memory as well to avoid re-encoding them during the feedback stage. These codewords which are inputs to the final RS(28, 24) decoder are represented as :

$$\boldsymbol{C_{final}} = \{C_{i,1}, C_{i,2,}C_{i,3},C_{i,4},\ldots,C_{i,28}\}, \quad (6.2)$$

where $i \in \{1,98\}$ and are each 28 byte-length for 98 frames. Flag values from ***F1*** represent each 28 byte long C_i codeword. $C_{i,j}$, where j ranges from 1 to 28, represents each byte of the codeword. This notation is used since the Interleaver II spreads the codeword at a byte-level and is necessary to keep track of the flags corresponding to each of these C.

2) Interleave II or reverse de-interleave is applied on the $\boldsymbol{C_{final}}$**.** This process plays a role to provide acceptable inputs to the RS(32, 28) encoder which is later employed. ***F1*** vector is also passed through the Interleaver II by first constructing $\boldsymbol{F1_{matrix}}$ as described below:

$$\boldsymbol{F1_{matrix}} = [f_i \quad f_i \quad f_i \quad \ldots\ldots \quad f_i], \quad (6.3)$$

which is a 98 × 28 sized matrix. This makes sure that each byte $C_{i,j}$ has a corresponding flag value f_i associated with it even after the Interleaver II is applied. The spread also affects the $\boldsymbol{F1_{matrix}}$ and each entry is shifted by equal amounts corresponding to the $\boldsymbol{C_{final.}}$

The $\boldsymbol{C_{final}}$ is now freed from the memory since it is no longer required in the feedback process. However, the interleaved $\boldsymbol{C_{final}}$ is saved in memory. Similarly, $\boldsymbol{F1_{matrix}}$ is saved in memory replacing the ***F1*** vector.

3) Now, the $\boldsymbol{C_{final}}$ has to be passed through the RS(32, 28) encoder. Note that this process adds to the constraints of the feedback system as it introduces further delays. After the

process is complete, the obtained set of codewords represented as $\boldsymbol{C_{enc,\,final}}$ should be saved in memory, and the $\boldsymbol{C_{final}}$ is freed. $\boldsymbol{C_{final}}$ is freed since it is no longer required in further feedback process.

The primary purpose of $\boldsymbol{C_{enc,\,final}}$ is to provide acceptable inputs to the first layer of the CIRC decoding stage which is the RS(32, 28) decoder. The encoding RS(32, 28) marks the end of the feedback process. However, the second round decoding takes place which improves the error correction capability of the system.

6.3 Decoding Process after Feedback

The decoding process begins once again after the feedback. The $\boldsymbol{C_{enc,\,final}}$ is retrieved from the memory and is passed into the RS(32, 28) decoder. However, there are flag values available which enlist the erasures and assist the decoding process. Since, an RS(n, k) code can correct up to $2t$ erasures where t represents the number of parity bits, the RS(32, 28) can now correct eight bytes of corrupted symbols instead of the four byte symbols that it did earlier.

This assisted decoding stage facilitates the future decoding stages as well and the performance improvement is rolled over to the following stages. Hence, the overall system performance is the same or better than before.

6.4 Simulation of the Feedback System

A successful simulation of the feedback system was implemented in MATLAB. Figure 6.2 shows the simulation results.

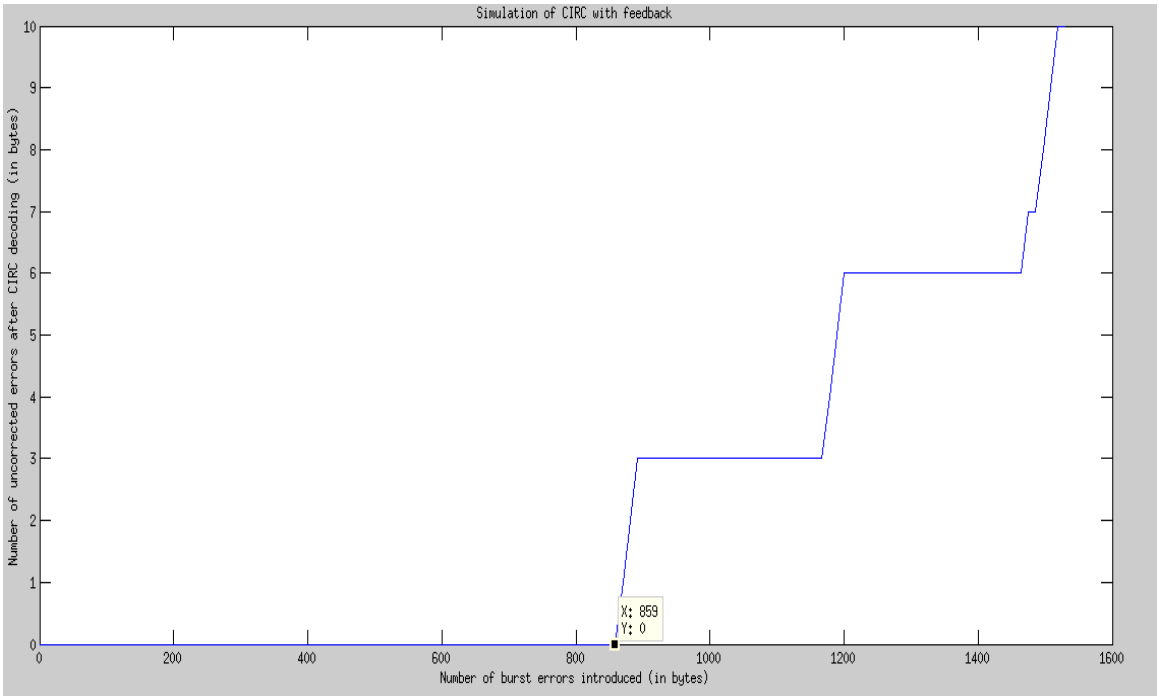

FIG. 6.2 Simulation of the CIRC with Feedback.

Observation: The error correction performance has improved after applying the feedback. Without feedback, all errors were corrected as long as the number of burst errors introduced was less than 781 bytes. There were uncorrected errors present beyond this margin of 781 bytes. The simulation results of the CIRC with feedback show an improvement. This margin of 781 bytes is shifted to 859 bytes. The uncorrected errors are also reduced for higher number of burst errors.

6.5 The Feedback System and Constraints

While designing the feedback, there are a couple of constraints in the design which are mentioned as follows:

1) Encoding requirement in feedback during the decoding stage:

Note that the codeword used in the first layer is an encoded version of the codeword in the second layer. Thus, the check flag values fed back will not be accepted by the first layer unless there is encoding facilitated. In other words, the input to the decoder should be from the same finite field to be consistent with the decoding procedure. Thus, the feedback requires a partial encoding procedure in the decoding stage. The encoding RS(32, 28) needs to be applied in order to obtain back the codewords accepted by the first CIRC decoding stage.

2) Memory requirement:

The flag values need to be stored for the feedback process. Earlier these flag values were used and freed during the process of decoding. However, since the feedback process involves multiple steps these flag values must be stored in memory. Secondly, during the decoding process the codewords which are input to RS(28, 24) need to be saved since they are utilized in the feedback. This memory requirement serves the purpose of avoiding the re-encoding of an additional layer. Encoding requirement for RS(32, 28) is unavoidable since the flag values are not yet obtained.

3) Reverse de-interleave (or interleave) in feedback during decoding stage:

Since the CIRC uses an interleave stage between the two layers of RS(n, k) coding, a reverse interleave stage is required for the feedback. For the decoding stage, the reverse de-interleaver (or the interleaver) needs to be implemented to negate the effect of spreading. The first layer will then recognize the codewords as acceptable versions of input codewords.

Once these constraints are considered in the design, there is a consistency introduced in the feedback such that the inputs provided to the first layer are acceptable. It should be noted that during the feedback stage, it is assumed that the decoding is complete since the feedback will not assist ongoing decoding, but will improve the decoded data with better error correction.

6.6 Design Trade-off in Feedback System

The system performance improves by introducing this feedback. The error correction capability is improved and could result in saving corrupted data than losing it forever. This could serve its purpose when the data storage disks are old and some technique needs to be used to save the data.

Though there is tremendous benefit in using the feedback, there are constraints that lead to a trade-off in the design. Following are the trade-offs in the feedback design:

1) Additional memory requirement:

The memory requirement to save flag values and the set of codewords asks for additional memory space. The matrix of data is sector-wise, and if parallel processed, memory requirement may be reduced.

2) Time delay in the overall system:

Due to an interleave stage and the RS(32, 28) encoder, there is a time delay introduced in the system. If τ_{rs} represents time required to process the RS(32, 28) encoder, τ_{Int} represents the time required to process the Interleaver II, and τ_{mem} represents the net time required to retrieve data from memory during feedback, then the total time delay:

$$\tau_{net\text{-}delay} = \tau_{rs} + \tau_{Int} + \tau_{mem} \quad (6.4)$$

3) Additional hardware to incorporate the RS(32, 28) encoder and Interleaver II:

Since the RS(32, 28) and Interleaver II need to be employed during the decoding process, the hardware must be incorporated in the overall system design. However, the error correction hardware involves both encoder and decoder units. This means that the requirement of additional hardware is avoided by intelligently using the encoding set of circuits during the feedback stages.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 Summary

In this thesis, the application of multi layered RS(*n, k*) coding in an audio disk was introduced from a mathematical viewpoint. The introduction to coding theory aspects followed by finite field arithmetic was made to give a mathematical background before introducing the concrete Reed-Solomon code setup. Performance evaluation through simulation gave scope to improve the error correction performance in the system. A feedback system along with its trade-off was discussed.

Reed-Solomon coding is a powerful error correction coding scheme. It can correct up to t errors and $2t$ erasures where $2t$ are the parity bits added during coding. When equipped with multiple layers and interleaving, the data is protected securely. This concept is applied in audio disks which have the capability of correcting errors.

An audio disk may have several scratches or damages induced due to rough usage; however, the RS(*n, k*) coding system can protect data most of the time. It is worth appreciating that there could be frames affected with burst errors; however, they can be corrected completely as seen in the simulations.

However, there are occasions when the multiple layered RS(*n, k*) coding scheme fails to correct codes. During these scenarios, the sector is banned and the entire data is lost. Some of

the data in the sector which was properly decoded goes away unused, which is a tremendous loss. In these cases, the thesis emphasizes using the feedback system.

The feedback system is accompanied with a reverse decoding process, or part of encoding process. The benefit is that the first layer is also capable of correcting $2t$ erasures, which is twice the amount of data it can correct than what it was correcting earlier. This performance is applied to the entire disk, and hence the gain is appreciable.

However, the feedback system introduces greater complexity in the system, thereby introducing a trade-off between the higher error correction performance versus the higher complexity due to part of the encoding procedure induced.

7.2 Recommendation and Future Work

The feedback system discussed in this thesis can be applied to a general multiple layered RS(n, k) coding scheme. The number of protected layers can vary and feedback can still be applicable. However, the feedback recommended is unit wise, as shown in Figure 7.1.

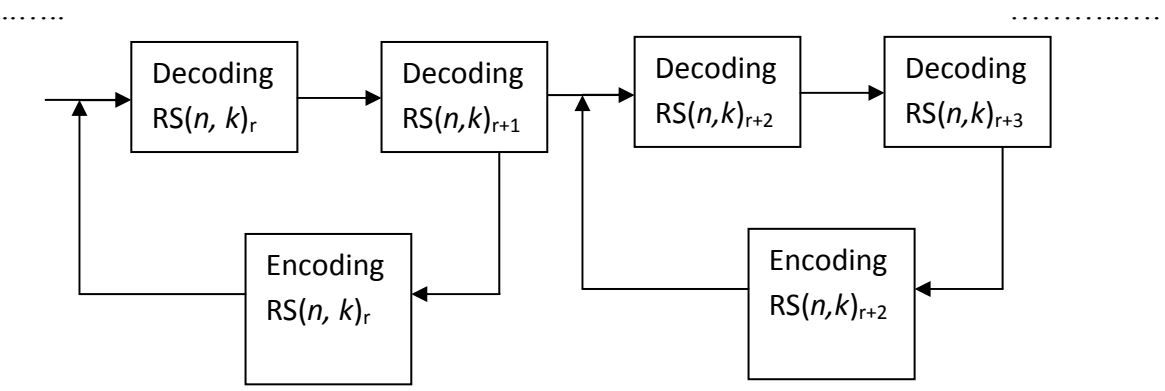

FIG 7.1 Feedback for General Multi Layered RS(n, k) Code.

Note that the RS(n, k)$_{r+1}$ denotes the r +1th stage of decoding, and it is assumed that the r +1th stage input is saved in memory, which needs to be sent back as feedback. The application of feedback can be summarized in the following:

Step 1: First round of decoding: Performing the decoding process for the entire multi layered structure. This ensures that the flag values are available at every RS(n, k) decoder unit. As mentioned before, the flag values of the second layer need to be saved along with the codeword input to the decoders.

Step 2: Breaking into pairs: The first step involves breaking the complicated multi layered structure into pairs of two. These pairs now act as units and are assisted with feedback. The pairs are treated individually with the feedback assisting them.

Step 3: Feedback system on each pair: Each pair is associated with its feedback and the performance improvement is later summed up over all pairs. The feedback requires the input to the second decoder to be saved in memory since it needs to be sent back. The flag values also need to be saved for each pair.

Step 4: Interaction with the next pair unit: After the feedback has been passed through each layer, each unit can now pass on the updated decoded word to the following unit. The updated decoded word will now have the same or better error correction performed on it. This ensures an improvement in performance.

These steps involve a higher memory requirement and also more computation time. This is an interesting problem and can be explored for improvement in performance. Since the time

and memory requirement is demanding, efforts can be made in order to reduce them. A better solution for such cases is an opportunity for further work.

APPENDIX A

SOURCE CODE FOR SIMULATION OF CIRC CODEC

```
clc;
Errors = 1:1: 28*5;
klast = 30 ;
for reset_counter = 1: length(Errors)
   Errors(reset_counter)=0;
end
main =0;
Number_of_Err =1:10:280*5;
temp_main = 0;
while main <= 28*5
%------ REED-SOLOMON CODING LEVEL 1 : C1 coding (28,24)--------
mC2 = 8; % Number of bits per symbol
nC2 = 28; % n= 28 for C1
kC2 = 24; % k= 24 for C1
% Creates a matrix of size 98 × 24 from Galois Field(2^m)
m = randi([0,(2^mC2)-1],98,kC2);
msgC2 = gf(m,mC2);
%  Since the original message encoded is first encoded here, the final decode message should
%  be compared to this.
msg = msgC2;
codeC2 = rsenc(msgC2,nC2,kC2);
%------- C1 coding finish ------------------------------------
```

```
%------- Interleaving II -------------------------------------
% Every word is delayed by 4n(n-1)frame mod 98
copyCodeC2= codeC2;
for i=1:98
   for j=1:28
        copyCodeC2(i,j)= codeC2(1+mod(i+4*(j-1),98),j);
  end
end
%------- Finished Interleaving II ----------------------------
% ------ REED-SOLOMON CODING LEVEL 2 : C1 coding (32,28)--------
mC1 = 8; % Number of bits per symbol (words are still bytes).
nC1 = 32; % Now n = 32 in C2.
kC1 = 28; % The coded C1 is data for C2. Hence data, k = 28.
% Now the gf function is unnecessary since we already have our message
% Just encode the previous codeC2 into RS(32, 28).
codeC1 = rsenc(copyCodeC2,nC1,kC1);
%------- C1 coding finish -----------------------------------
%---------- ENCODING is DONE --------------------------------
% Errors are introduced from frame 29 and affect the
% Propagation over channel :  add errors
% klast denotes the number of frames affected by errors.
% Burst error for klast continuous frames.
errors = codeC1-codeC1;
if(temp_main==28)
   klast = klast +1;
   temp_main = 0;
```

```
end
% Systematic error introduction.
% Bytes affected by errors shown.
% j denotes the columns in frames affected by burst
jlast = 1;
jlast = jlast + temp_main;
if(klast > 29)
   for j = 1:jlast
   errors(klast,j)= 3;
   end
end
for k = 28:klast-1
   for j = 1:28
   errors(k,j)= 3;
   end
end
% Errors have been added.
% These errors represent fingerprints, scratches etc.
codeC1 = codeC1 + errors;
%---------- BEGIN DECODING ----------------------------------
% Reverse process first decode C1.
% C1 is first decoded since it was last encoded.
[C1dec,C1cnumerr] = rsdec(codeC1,nC1,kC1);
%---------- Deinterleave II ----------------------------------
copyCodeC1 = C1dec;
for i=1:98
```

```
    for j=1:28
      copyCodeC1(i,j)= C1dec(1+mod(-2+i-4*(j-1),98),j);
    end
end
codeC2;
copyCodeC1;
check = codeC2 - copyCodeC1;
[C2dec,C2cnumerr] = rsdec(copyCodeC1,nC2,kC2);
%The final version of the decoded message is C2dec
% No further decoding is required.
dec = C2dec;
for o=1:length(C2cnumerr)
    if(C2cnumerr(o)==-1 || C2cnumerr(o)>2)
       Errors(main)=Errors(main)+1;
    end
end
main = main +1;
temp_main = temp_main +1;
end
plot(Number_of_Err, Errors,'-')
title('Simulation of Uncorrected Errors after decoding vs Errors Introduced');
xlabel('Number of Burst Errors Introduced (in bytes)');
ylabel('Number of Uncorrected Errors after CIRC Decoding (in bytes)');
```

REFERENCES

[1] Immink, K.S., "The CD story", *Journal of the AES*, vol. 46, pp 458-465, 2007.

[2] Immink, K.S., *Coding techniques for digital recorders*, New York: Prentice-Hall, 1991.

[3] Reed, I.S. and Solomon, G., "Polynomial codes over certain finite fields*", SIAM Journal of Applied Math.*, vol. 8, 1960, pp. 300-304.

[4] Blahut, R. E., *Theory and practice of error control codes*, Reading, MA: Addison-Wesley, 1983.

[5] Sklar, B., *Digital communication: Fundamentals and applications*, Second Edition, NJ: Prentice-Hall, 2001.

[6] Wicker, S.B. and Bhargava, V. K., *Reed-Solomon codes and their applications*, Piscataway, NJ: IEEE Press, 1983.

[7] Arai, T., et al., "High capability error correction LSI for CD players and CD-ROM", *IEEE Transactions on Consumer Electronics*, vol. CE-30, no. 3, pp 353-359, 1984.

[8] Lin, S., and Costello, D., *Error control coding: Fundamentals and applications*. NJ: Prentice-Hall, 1983.

[9] Roberts, J.D. et al., "Analysis of error correction constraints in an optical disk", *Applied Optics*, vol. 35 no. 20, pp. 3915- 3924, 1996.

[10] Shinya, O., "Error Correction method using Reed-Solomon code," US Patent 4852099, July 25, 1989.