Table of Contents

ACKNOWLEDGMENT

First I would like to thank Dr. Ali Serener to be my advisor. He encouraged me to search more to understand and overcome the difficulties I faced in learning Error Correction Codes. His supporting was continuous all over the way.

I also want to thank the management of Near East University for supplying the university library with precious books in all fields that helped me to find materials related to my project.

Special thanks to Dr. Özgür Özerdem to be my academic advisor, also Dr. Adnan Al-Khasman whom I learned from him how to hold responsibilities, and how to plan for my future life.

I want to thank my friends: Munir, Ghyath, Ghazwan, Ahmad, Zaher, Somer and Basel to be very supportive in hard and easy times.

Finally, I want to thank my family, specially my parents. Without their support and love I wouldn't reach this position. Also, I want to thank my sister, brother and brother in law for encouraging me. I am trying to let them be proud of me and my achievements.

ABSTRACT

Because of the importance of digital communication in our life, and in order to get the best performance of it, error control coding helps technicians meet there goals which is developing error free channels to increase the capacity and transmission rate.

Mathematicians and technicians related to the communication field tried to find the codes that helps them to achieve the aimed results. They discovered various error correcting codes that have the ability of detecting and/or correcting the errors that occurred in the information received from the channel by the receiver.

In this report we first defined different types of codes that are used in different fields according to there application. Then we concentrated later on linear block codes that are the simplest in construction and usage, but that does not mean it is less efficient.

The aim of all the error correcting codes is to meet the Shannon's limit which is to use the full capacity to get the maximum rate in transmitting and receiving data (information).

INTRODUCTION

A profusion and variety of communication systems, which carry massive amounts of digital data between terminal and data users of many kinds, exist today. The data (message) that entered to the communication channel may expose to different types of noise that will effect on the transmitted data and increase the bit error rate. The communication system must transmit its data with very high reliability and keep the bit error rate as small as possible.

Scientists found out that in order to decrease the amount of error rate into a negligible rate they came out with codes (Error Control Codes). Error control codes with its several types have the ability of detecting and/or correcting errors happened in the received message.

This report aims to give explanation about the most popular types of codes and concentrate on linear block codes because of its generating simplicity and its efficiency in correcting and detecting errors.

Chapter 1 starts by telling about the importance of error controlling codes. After that we went back in time to find out historical information about how the idea started and what obstacles faced the scientists and people related to the field. Then we give information about some applications that error control codes serves. At the end of the first chapter we include different error control techniques.

Chapter 2 is related to the types of codes which have two main types: Block codes and Convolutional codes. Block codes have several types like Cyclic, Linear and Hamming codes. Low Density Parity Check (LDPC) code is a convolutional. We will talk about Turbo codes. There are two types of turbo codes, Block Turbo Code (BTC) and Convolutional Turbo Code (CTC).we will explain basic properties of each type.

Chapter 3 will go in more details about Linear Block codes and Hamming codes. These details are concerned of how to generate the generator and parity check matrices of the both codes. Also some basic properties that are important for constructing Linear and Hamming codes are given.

Chapter 4 is the results section. At the beginning of this chapter we will explain in more details how detecting and correcting errors work by using an example of brute force look-up method. Then we will show the way of designing a [7, 4] linear block code and some possible code words that [7, 4] code contain.

Conclusion presents the newly applications in newly invented technologies that uses error control codes.

CHAPTER 1

ERROR CONTROL CODING

1.1 Introduction to Error Control Coding (ECC)

Many of us have heard of the term *error control coding* communication systems, but few of us understand what it is, what it does, and why we should be as technology seekers concern about it. The field is growing rapidly and knowledge of the topic will be very important and asked to be earned by anyone involved in the design of modern communication systems.

1.2 History of Error Control Coding

In 1948 Claude Shannon [1] showed that every communication channel has a capacity, C (measured in bits/sec), and as long as the transmission rate, R (bits/sec), is less than C it is possible to design a virtually error free communication system using error control codes. His discover showed that channel noise limits the transmission rate, not the error probability. Prior to this discovery it was thought that channel noise prevented error free communications.

After the publication of Shannon's paper [1], which didn't included how to find the error control codes, researchers scrambled to find codes that would produce the very small probability of error that he predicted. In the 1950s the progress of finding good performing codes didn't succeeded in finding one. In the 1960s, the field split between the algebraists who concentrated on a class of codes called *block codes* and the probabilists who were concerned with understanding encoding and decoding as a random process. Probabilists eventually discovered a second class of codes, called *convolutional codes,* and designed powerful decoders for them. In the 1970s, the two research paths merged and several efficient decoding algorithms were developed. Finally in 1981, decoders became practical;

the entertainment industry adopted a very powerful error control scheme for the new compact disc (CD) players [2].

1.3 Digital Communication System Elements

Digital communication systems are often partitioned as in Figure 1.

- *Encoder and Decoder:* the encoder adds redundant (extra) bits to the sender's bit stream to create a *codeword.* The decoder uses the redundant bits to detect and/or correct as many errors as the particular error control code will allow.
- *Modulator & Demodulator:* the modulator transforms the output of the encoder, which is digital, into a format suitable for the channel, which is usually analog. The demodulator attempts to recover data that passed the channel and effected by noise.

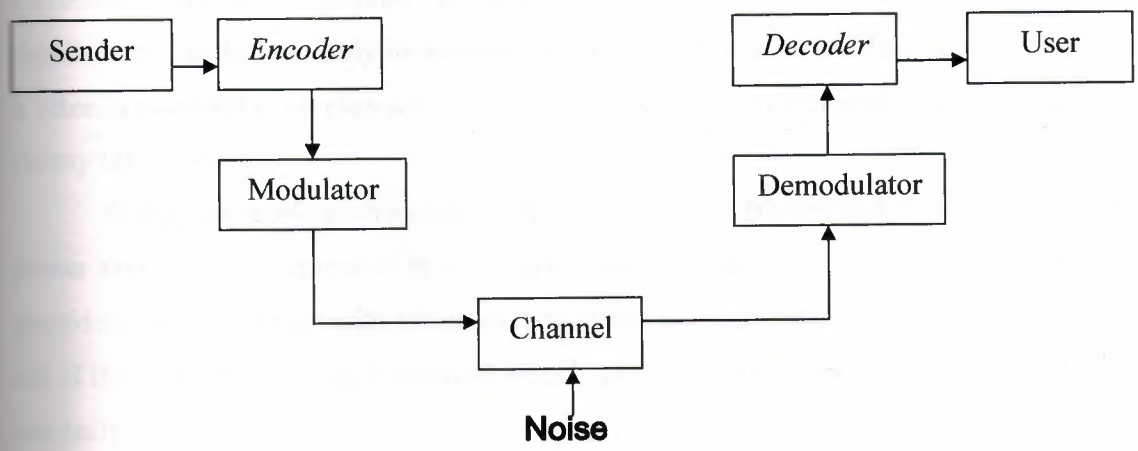

Figure 1 - The Digital Communication System

- *Communication channel:* is the medium the data travel through between the transmitter and receiver, where errors is introduced.

- *Error control code:* a set of codewords used with an encoder to detect errors, correct errors, or both.

1.4 Applications

Because the development of data-transmission codes was motivated primarily by problems in communications, much of the terminology of the subject has been drawn from the subject of communication theory. These codes, however, have many other applications. Codes are used to protect data in computer memories and on digital tapes and disks, and to protect against circuit malfunction or noise in digital logic circuits.

Applications to communication problems are diversified. Binary messages are commonly transmitted between computer terminals, in communication networks, between aircraft, and from spacecraft. Codes can be used to achieve reliable communication even when the received signal power is close to the thermal noise power. And, as the electromagnetic spectrum becomes ever more crowded with man-made signals, data-transmission codes will become even more important because they permit communication links to function reliably in the presence of interference. In military applications, it often is essential to employ a data-transmission code to protect against intentional enemy interference.

Many communication systems have limitations on transmitted power. For example, power may be very expensive in communication relay satellites. Data-transmission codes provide an excellent tool with which to reduce power needs because, with the aid of the code, the messages received weakly at their destinations can be recovered correctly.

Transmissions within computer systems usually are intolerant of even very low error rates because a single error can destroy the validity of a computer program. Error-control coding is important in these applications. Bits can be packed more tightly into some kinds of computer memories (magnetic or optical disks, for example) by using a data transmission code.

Another kind of communication system structure is a multi-access system, in which each of a number of users is preassigned an access slot for the channel. This access

may be a time slot or frequency slot, consisting of a time interval or frequency interval during which transmission is permitted, or it may be a predetermined coded sequence representing a particular symbol that the user is permitted to transmit. A long binary message may be divided into packets with one packet transmitted within an assigned access slot. Occasionally packets become lost because of collisions, synchronization failure, or routing problems. A suitable data-transmission code protects against these losses because missing packets can be deduced from known packets.

Communication is also important within a large system. In complex digital systems, a large data flow may exist between subsystems. Digital autopilots, digital process control systems, digital switching systems, and digital radar signal processing all are systems that involve large amounts of digital data which must be shared by multiple interconnected subsystems. This data transfer might be either by dedicated lines or by a more sophisticated, time-shared data-bus system. In either case, error-control techniques are important to ensure proper performance.

Eventually, data-transmission codes and the circuits for encoding and decoding will reach the point where they can handle massive amounts of data. One may anticipate that such techniques will play a central role in all communication systems of the future. Phonograph records, tapes, and television waveforms of the near future will employ digital messages protected by error-control codes. Scratches in a record, or interference in a received signal, will be completely suppressed by the coding as long as the errors are less serious than the capability designed into the error-control code. [5]

1.5 Briefly What Coding Can Do

The traditional role for error control coding was to make a troublesome channel acceptable by lowering the frequency of error events. Coding's role has expanded tremendously and today coding can do the following:

- Reduce the occurrence of undetected errors.
- Reduce the cost of communication systems.

- Overcome jamming.
- Eliminate interference.

1.6 How Error Control Codes Work

A foundation in modern algebra and probability theory is required to have a full understanding of the structure and performance of error control codes. As we mentioned before that error control codes have two classes, *Block codes and Convolutional codes.*

Block codes has several schemes: *linear block code, Hamming code and cyclic code*. The block encoder takes a block of k bits and replace it with a n-bit codeword ($n>k$). For a binary code, there are 2^k possible codewords. The channel introduces errors and the received word can be anyone of 2^n n-bit words of which only 2^k are valid codewords. The job of the decoder is to find the codeword that is closest to the received n-bit word.

Convolutional codes: differ from block codes in that there are no independent codewords. The encoding process can be envisioned as a sliding window, M block wide, which moves over the sequence of information symbols in steps of K symbols. M is called the *constraint length* of the code. With each step of the sliding window, the encoding process generates N symbols based on the MK symbols visible in the window.
A convolutional code so constructed is called an (N, K, M) code. Convolutional codes are commonly used in applications that require relatively good performance with low implementation cost.

1.7 Popular Coding Techniques

In this section we will give brief explanation about four of most popular error control coding techniques, which are:

- Automatic Repeat Request (ARQ)
- Hybrid ARQ
- Forward Error Correction (FEC)

1.7.1 Automatic Repeat Request

An error detection code by itself does not control errors, but it can be used to request repeated transmission of errored codewords until they are received error free. In terms of error performance, ARQ outperforms forward error correction because codewords always delivered error free. The chief advantage of ARQ is that error detection requires much simpler decoding equipment than error correction.

1.7.2 Hybrid ARQ

Hybrid ARQ schemes combine error detection and forward error correction to make more efficient use of the channel. At the receiver, the decoder first attempts to correct any errors present in the received codeword. If it cannot correct all the errors, it requests retransmission of the message again. Using one of these techniques. *Type 1 Hybrid ARQ* sends all the necessary parity bits for error detection and error correction with each codeword. *Type 2 hybrid ARQ,* on the other hand, sends only the error detection parity bits and reserves the correction parity bits. If the decoder detects errors, the receiver requests the error correction parity bits and attempts to correct the errors with these parity bits before requesting retransmission of the entire codeword.

1.7.3 Forward Error Correction (FEC)

Forward error correction is appropriate for applications where the user must get the message right the first time. The one-way or broadcast channel is one example.

CHAPTER 2

CODES

2.1 Introduction

Error-control coding techniques are used to detect and/or correct errors that occur in the message transmission in a digital communication system. The transmitting side of the error-control coding adds redundant bits or symbols to the original information signal sequence. The receiving side of the error-control coding uses these redundant bits of symbols to detect and/or correct the errors that occurred during transmission. The transmission coding process is known as encoding, and the receiving coding process is known as decoding.

2.2 Block Codes

In *block coding*[4], successive blocks of k information (message) symbols are formed. The coding algorithm then transforms each block into a codeword consisting of n symbols where $n>k$. This structure is called an (n, k) code. The ratio k/n is called the *code rate*. A key point is that each codeword is formed independently from other codewords.

Figure.2.1 shows the corrupted signal via an AWGN with variance 0.02 without error-control coding. The bit error-rate is 0.015.

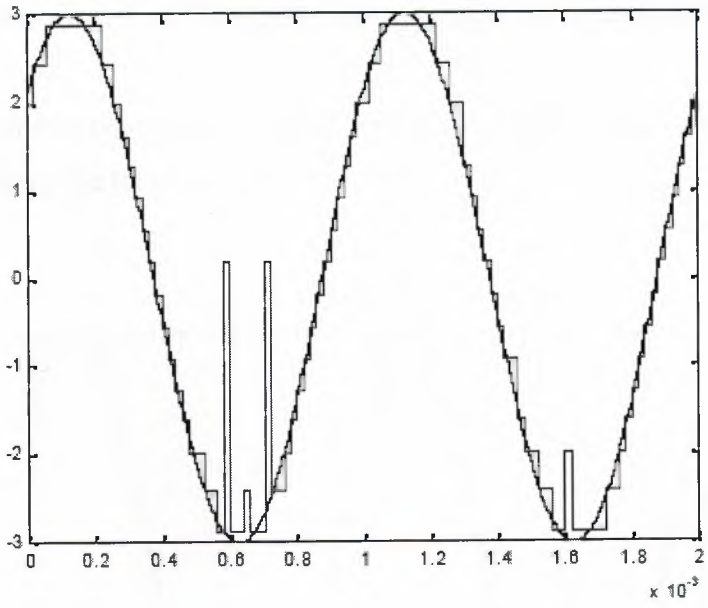

Figure 2.1 without error control coding

But when using different block code schemes the bit error- rate will be 0 *(n=7, k=4),* such like: *Linear block code, Hamming code and Cyclic code.*

2.3 Convolutional Codes

Convolutional codes [4] differ from block codes in that there are no independent codewords. The encoding process can be envisioned as a sliding window, *M* block wide, which moves over the sequence of information symbols in steps of *K* symbols. *M* is called the *constraint length* of the code. With each step of the sliding window, the encoding process generates *N* symbols based on the *M*×*K* symbols visible in the window. A convolutional code so constructed is called an (*N*, *K*, *M*) code. Convolutional codes are commonly used in applications that require relatively good performance with low implementation cost.

The *Viterbi method* is used for decoding the convolutional codes. The Viterbi algorithm is a *maximum likelihood* (ML) decoding procedure that takes advantage of the fact that a convolutional encoder is a *finite state machine*. The criterion used for decision

-making is the *metric* for *soft decision decoding* and the *Hamming distance* for *hard decision coding*.

Figure.2.2 shows the recovered signal via the same channel as Figure.1 with convolutional coding. The bit error rate is 0.

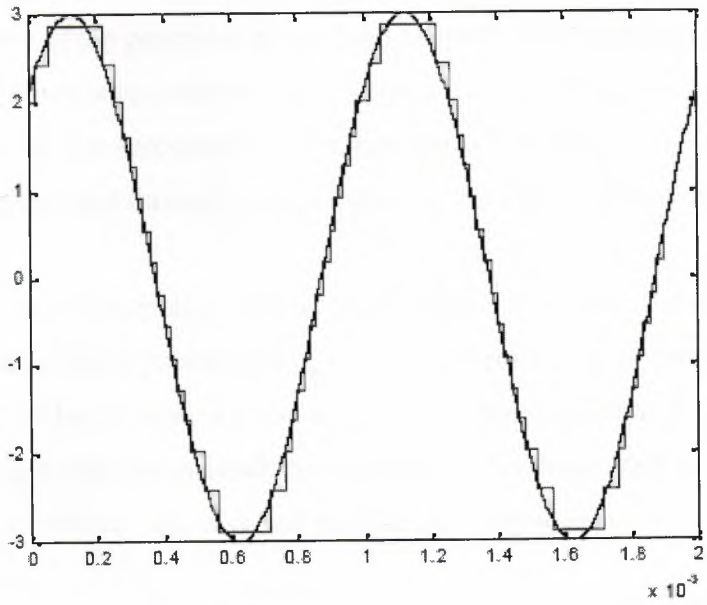

Figure 2.2 Convolutional coding

2.4 Linear Cyclic code

Cyclic code [3] is a subset of linear code which further has the cyclic property.

(Linear Cyclic Code) A code is linear cyclic if

1. The linear combination of any two codewords is also a codeword;
2. Any cyclic shift of a codeword is also a codeword.

In the definition, the first property is linearity and second cyclic.

2.4.1 Generator Polynomial and Parity Check Polynomial

Within the set of code polynomials of a linear cyclic code C, there is a unique polynomial1 g(x) with minimal degree r < n such that every code polynomial c(x) can be expressed as c(x) = m(x)g(x), where m(x) is a polynomial of degree less than (n-r). The order-r polynomial g(x) is called the generator polynomial of code C.

The existence of the generator polynomial suggests a convenient method for mapping message words onto codewords of a linear cyclic code. For a k = n-r long message word $m = m_0 \ldots m_{k-1}$, we can associate with it a message polynomial $m(x) = m_0 + \ldots m_{k-1} x^{k-1}$. Then, the $\underline{m}$ can be encoded through multiplication of $m(x)$ by the generator polynomial $g(x)$.

The generator polynomial g(x) of an (n, k) linear cyclic code divides $x^n - 1$. Reversely, any order-r factor polynomial of $x^n - 1$ can generate a linear cyclic code (n, n-r).

According to the theorem, there exist an order k polynomial $h(x)$ such that $g(x)h(x) = x^n - 1$. For a code polynomial, c(x) = m(x)h(x), we have $c(x)h(x) = m(x)g(x)h(x) = 0$ modulo $x^n - 1$. Therefore, $h(x)$ is called parity check polynomial.

2.4.2 Generator and Parity-Check Matrices

Cyclic codes also have generator and parity-check matrices, and these matrices are related to the generator and parity check polynomials of the codes. Consider the encoding procedure.

$$c(x) = m(x)g(x) = (m_0 + \ldots + m_{k-1})\, g(x) = [\, m_0, m_1 \ldots m_{k-1} \,] \begin{bmatrix} g(x) \\ xg(x) \\ \vdots \\ x^{n-k-1}g(x) \end{bmatrix} \qquad (2.1)$$

It provides a convenient matrix relation.

$$\underline{c} = [c_0, \ldots, c_{n-1}] = [m_0, \ldots, m_{k-1}] \begin{bmatrix} g_0 & g_1 & \cdots & g_r & & & 0 \\ & g_0 & g_1 & \cdots & g_r & & \\ & & g_0 & g_1 & \cdots & g_r & \\ 0 & & & g_0 & g_1 & \cdots & g_r \end{bmatrix} = \underline{m}G \qquad (2.2)$$

The generator matrix G is a Toeplitz matrix composed of the coefficients of the generator polynomial g(x). Similarly, we can obtain the parity check matrix.

$$H = \begin{bmatrix} h_k & h_{k-1} & \cdots & h_0 & & & 0 \\ & h_k & h_{k-1} & \cdots & h_0 & & \\ & & h_k & h_{k-1} & \cdots & h_0 & \\ 0 & & & h_k & h_{k-1} & \cdots & h_0 \end{bmatrix} \qquad (2.3)$$

It is easy to verify that $\underline{c}H^T = 0$. Similar to the discussion for linear code, if we use H as generator matrix and G parity check matrix, we obtain an $(n\text{-}k, k)$ code which is the dual code of the original one.

2.4.3 Systematic Linear Cyclic Code

Recall that a systematic code has the message word as part of the corresponding codeword, i.e., $c = b_0 \ldots b_{n-k-1}\ m_0 \ldots m_{k-1}$. Therefore, the code polynomial can be expressed as

$$c(x) = b(x) + x^{n-k} m(x) \qquad (2.4)$$

Since $c(x)$ must be a multiple of the generator matrix $g(x)$,

$$c(x) = a(x)g(x) = b(x) + x^{n-k} m(x) \tag{2.5}$$

This gives us the following systematic encoding algorithm for an (n, k) linear cyclic code.

Step 1 Multiply the message polynomial $m(x)$ by x^{n-k}.

Step 2 Divide the result from Step 1 by the generator polynomial $g(x)$; let $b(x)$ be the remainder.

Step 3 Set the code polynomial $c(x) = x^{n-k} m(x) + b(x)$.

2.4.4 Syndrome Decoder for Linear Cyclic Code

In the discussion of linear block codes, the syndrome vector was used to implement error correction. The basic idea is that every syndrome value corresponds to a coset of C in the binary word space. The error pattern with the lowest weight within a given coset is the most likely to occur, and is thus selected as the coset leader. Maximum likelihood error correction is performed by computing the syndrome for a received binary word, looking up the corresponding by coset leader, and subtracts the coset leader from the received word. This procedure also applies the linear cyclic code. The received binary word can be represented as a order-n polynomial.

$$r(x) = c(x) + e(x) \tag{2.6}$$

where $c(x)$ is the transmitted code polynomial and $e(x)$ is the error pattern polynomial. The syndrome polynomial s(x) is the remainder of dividing $r(x)$ by $g(x)$ (or the modulo x^{n-1} multiplication of r(x) with h(x)). So, we can write

$$r(x) = q(x)g(x) + s(x) \tag{2.7}$$

where $q(x)$ is the quotient polynomial of $r(x)$ divided by $g(x)$. Since $c(x)$ must be a multiple of $g(x)$: $c(x) = a(x)g(x)$,

$$s(x) = [q(x) + a(x)]g(x) + e(x) \tag{2.8}$$

The syndrome decoder for linear cyclic block codes includes the following steps:

Step 1 Express the received binary word in polynomial form, $r(x)$.

Step 2 Divide $r(x)$ by $g(x)$ and find the remainder $s(x)$

Step 3 If $s(x) = 0$, there is no error; if $s(x) \neq 0$, find the corresponding coset leader $e(x)$ to correct the error according $c(x) = r(x) + e(x)$.

Using the syndrome decoder, the receiver needs to store the syndrome table which contains 2^{n-k} entries. For linear cyclic codes however, the size the syndrome table can be reduced to 1/n of its original size thanks to the cyclic structure. Figure.2.3 shows the performance of cyclic codes on Additive White Gaussian Noise (AWGN) channel.

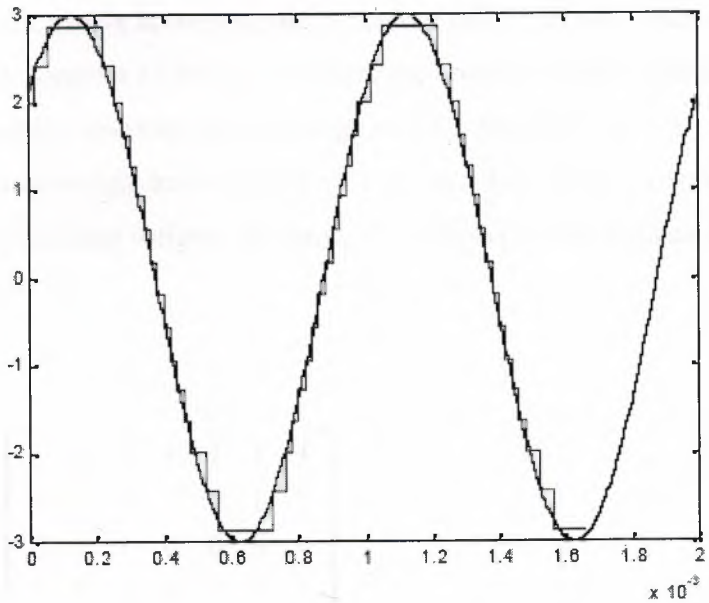

Figure.2.3 Cyclic code n= 7, k= 4

2.5 Low-Density Parity Check Codes (LDPC)

LDPC codes [6] are conceptually very simple, and are defined in terms of a sparse parity check matrix. The decoder uses the message passing algorithm. The message passing algorithm is highly parallel and is ideally suited for high data rate applications. Careful programming of the decoder results a surprisingly simple decoder.

An (n, k) block code C is a mapping between a k-bit message (row) vector m, and an n length codeword vector c. The code C is linear if a k-dimensional subspace of an n_dimensional binary vector space Vn. The code can also be viewed as a mapping of k-space to n-space by a k x n generator matrix G, where c = mG. The rows of G constitute a basis of the code subspace. The dual space, C^T consists of all those vectors in Vn orthogonal to C, namely for all $c \in C$ and all d $\in C^T$, < c, d >= 0. The rows of an (n-k x n) parity check matrix H constitute a basis for C^T. It follows that for all c $\in$ C, $cH^T = 0$. A code is completely specified by either G or H, but neither are unique.

A low density parity check code is one where the parity check matrix is binary and sparse, where most of the entries are zero and only a small fraction are 1's. In its simplest form the parity check matrix is constructed at random subject to some rather weak constraints on H. A t-regular LDPC is one where the column weight (number of ones) for each column is exactly t resulting in an average row weight of $[n\ t\ /(n-k)]$.
One might fix the row weight to be exactly s = $[\ n\ t\ /(n\ \_\ k)]$. An (s, t)-regular LDPC is one where both row and column weights are fixed. The following parity check matrix H is an LDPC matrix with t = 2

$$H=\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

and for any valid codeword c

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_5 \\ c_6 \end{bmatrix} \tag{2.9}$$

This expression serves as the starting point for constructing the decoder. The matrix/vector multiplication in H defines a set of parity checks, which for the specific example are

$$p_0 = c_0 \oplus c_3 \oplus c_4 \oplus c_5 \oplus c_6 \tag{2.10}$$

$$p_1 = c_1 \oplus c_3 \oplus c_5 \tag{2.11}$$

$$p_2 = c_1 \oplus c_2 \tag{2.12}$$

$$p_3 = c_0 \oplus c_2 \oplus c_4 \oplus c_6 \tag{2.13}$$

A complete discussion of the message passing decoder is given in the next section. Next, we address encoding. By defining an LDPC code in terms of H alone it is not obvious what constitutes the set C of valid codewords. Furthermore we need to specify the generator matrix G for the encoder. A straightforward way of doing this is to first reduce H to systematic form $H_{sys} = [I_{n-k} \mid P]$: In principle this is simple using Gaussian elimination and some column reordering. As long as H is full (row) rank, H_{sys} will have n _ k rows. There is some probability, however, that some of the rows of H are linearly dependent. In

that case H is not full rank and the resulting H will be in systematic form, albeit with fewer rows. Once H is in systematic form, it is easy to confirm that a valid (systematic) generator matrix is $G_{sys} = [\, P^T,\ I_k \,]$ since $GH^T = 0$, it is interesting to note that the H_{sys} no longer has fixed column or row weight, and with high probability P is dense. The denseness of P can make the encoder quite complex.

As an example the parity check matrix in (2.9) can be reduced to systematic form as

$$H_{systematic} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{2.14}$$

2.6 Turbo Codes

The main goal of coding theory has always been to produce error-correcting codes that come close to the Shannon limit performance. Aiming to achieve near the Shannon limit performance, the research in coding theory has seen many powerful codes with large codeword lengths (for block codes) or constraint lengths (for convolutional codes). However, the decoding algorithms for many of these codes are complex or sometime physically unrealizable due to the lengths of the codes. As a result, the complexity in decoding powerful error-correcting codes has always been thought of as the real difficulty in the field of channel coding.

One possible solution to this problem is to construct powerful codes with large block or constraint lengths structured so as to permit the breaking of the decoding into simpler partial decoding steps. Iterated codes, product codes, concatenated codes, and large constraint length convolutional codes with suboptimal decoding strategies are some examples of these attempts. The most recent successful attempt consists of the so-called turbo codes, whose amazing performance has given rise to a large interest in the coding community.

Turbo codes were introduced by Berrou [4] in 1993. Using turbo codes, Berrou showed that it is possible to transmit data with a code rate above the channel cutoff rate. He achieved an exceptional low BER with a signal to noise ratio (Eb/N0) close to the Shannon's theoretical limit on a Gaussian channel. The turbo coding scheme consists of two recursive systematic convolutional codes concatenated in parallel. The codewords are decoded using iterative maximum-likelihood (ML) decoding (soft decoding) of the component codes. *Maximum a posteriori* (MAP) algorithm is used to perform maximum space likelihood bit estimation and thus yields reliability information (soft-output) for each bit. This decoding algorithm is implemented using soft-input/soft-output decoders. By cascading several of these decoders, an iterative ML decoding can be performed on the component codes which are optimal at each decoding step.

Turbo codes have received much attention since 1993, and many papers related to turbo codes have been published. The turbo code proposed by these papers can be broadly divided into two major types: block turbo codes (BTC) and convolutional turbo codes (CTC).

In BTC, the encoder is formed by concatenating two or more linear block encoders to generate the codewords. In most cases, two dimensional product codes, which can be thought of as serially concatenated block codes, are used, instead of codes generated by concatenating linear block codes. To decode the product codes, iterative soft-input/soft-output (SISO) decoding, which is also called turbo decoding, is used, in place of the conventional hard decision decoding.

In CTC, the encoder is formed by concatenating two or more convolutional encoders in parallel through the use of an interleaver. The input information bits enter the first encoder and after having been scrambled by the interleaver, enter the second encoder.

The codeword of the CTC consists of the information bits followed by the parity checkbits of both convolutional encoders. As was in the BTC, turbo decoding is used to decodethe CTC codewords.

2.6.1 Block Turbo Codes (BTC)

In BTC, the encoder is formed by concatenating two or more block encoders to

generate the codewords. Most of the research works on BTC use product codes, which can be thought of as serially concatenated block codes, to encode the information data and very few have considered codes generated by concatenating linear block codes such as Hamming code. The next section will present the BTC which uses product codes to encode the information data.

2.6.1.1 Product codes

Product codes are serially concatenated codes which are widely used in practice due to the simplicity of their implementation and their capability in fighting against bursts of errors. They are generated by arranging the message bits in an array of k_1 rows and k_2 columns and then appending horizontal parity check bits to each row and vertical parity check bits to each column (as shown in Figure. 2.4)

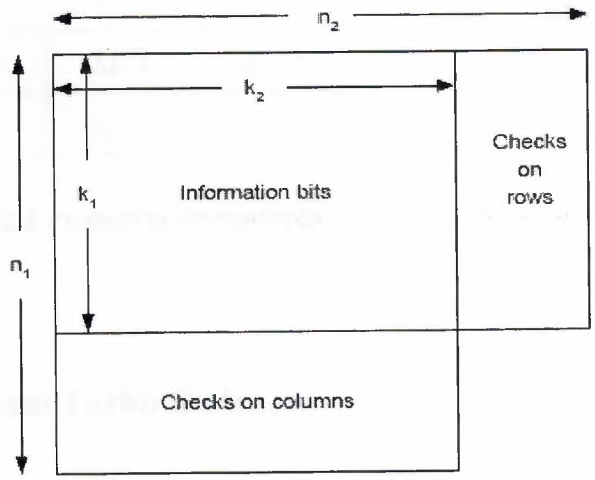

Figure 2.3 construction of product code

An example of single-parity product codes is shown in Figure 2. The relationship between the data and parity bits is as follows:

$p_{ij} = d_i \oplus d_j$

where ⊕ denotes exclusive or addition. As shown in Figure 2.4, the data sequence $d_1\ d_2\ d_3\ d_4$ is made up of the binary digits 1 0 0 1. Using Equation 2.11, the parity sequence $p_{12}\ p_{34}\ p_{13}\ p_{14}$ is found to be 1 1 1 1. Thus, the transmitted sequence is

$$\{d_1\ d_2\ d_3\ d_4\ p_{12}\ p_{34}\ p_{13}\ p_{24}\} = 1\ 0\ 0\ 1\ 1\ 1\ 1\ 1 \tag{2.15}$$

the transmitted sequence is

+1 -1 -1 +1 +1 +1 +1 +1

This example will be used in the subsequent sections to show the principles of block turbo coding.

d_1=1	d_2=0	p_{12}=1
d_3=0	d_4=1	p_{34}=1
p_{13}=1	p_{24}=1	

Figure 2.4 Product code example

2.6.2 Convolutional Turbo Codes

In CTC, the encoder is formed by concatenating two or more convolutional encoders in parallel through the use of an interleaver. The input information bits enter the first encoder and after having been scrambled by the interleaver, enter the second encoder. The codewords of the CTC consist of the information bits followed by the parity check bits of both convolutional encoders.

2.6.2.1 Construction of CTC

Figure.2.5 shows the concatenated encoders proposed by Berrou , where recursive systematic encoders (RSC) are used. The information data, d_k, goes directly to the first elementary RSC encoder C_1 and after interleaving, feeds the second elementary RSC encoder C_2. The information data, d_k, is systematically transmitted as symbol X_k and redundancies Y_{1k} and Y_{2k} produced by C_1 and C_2 respectively. In general, the two component encoders need not be identical with regard to constraint length and rate. Therefore, the two elementary coding rates R_1 and R_2 associated with C_1 and C_2 may be different. For best decoding performance, the two elementary coding rates should satisfy $R_1 \leq R_2$. The global rate R of the composite code is given by the following equation

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1 \tag{2.16}$$

In designing turbo codes, the main goal is to select the best component codes by maximizing the effective free distance of the code. At large values of E_b/N_0, this is equivalent to maximizing the minimum weight codeword. However, at low values of E_b/N_0, optimizing the weight distribution of the codewords is more important than maximizing the minimum weight.

Additional component codes can be added by parallel concatenating the component encoder. The parallel concatenation enables the elementary encoders, and therefore the associated elementary decoders, to run with the same clock. This provides an important simplification for the design of the associated circuits in a concatenated scheme.

2.6.3. Performance of BTC and CTC

Most of the research works on turbo codes are focused on the CTC and very few have considered the BTC. However, it has been shown in that BTC performs better than CTC for high-code-rate applications. The main reason is that the minimum distance of a

turbo code becomes critical when the interleaver size is small. While the minimum distance of a CTC can be relatively small, a product code can provide a minimum distance of 16, 36 (or more).

BTC also performs better than CTC for high-data-rate systems. In high-data-rate systems, the decoding speed of a BTC can be increased by using several elementary decoders for the parallel decoding of the rows (or columns) of a product code since they are independent.

The turbo codes have been shown to achieve exceptionally low BER with a signal to noise ratio close to the Shannon theoretical limit.

The choice of BTC and CTC depends on the code rate of the system. The simulation results from different authors show that for a given BER, it is possible to define a "threshold rate". For rates smaller than the threshold value, the CTC should be used, and for rates greater than this rate it is better to use BTC.

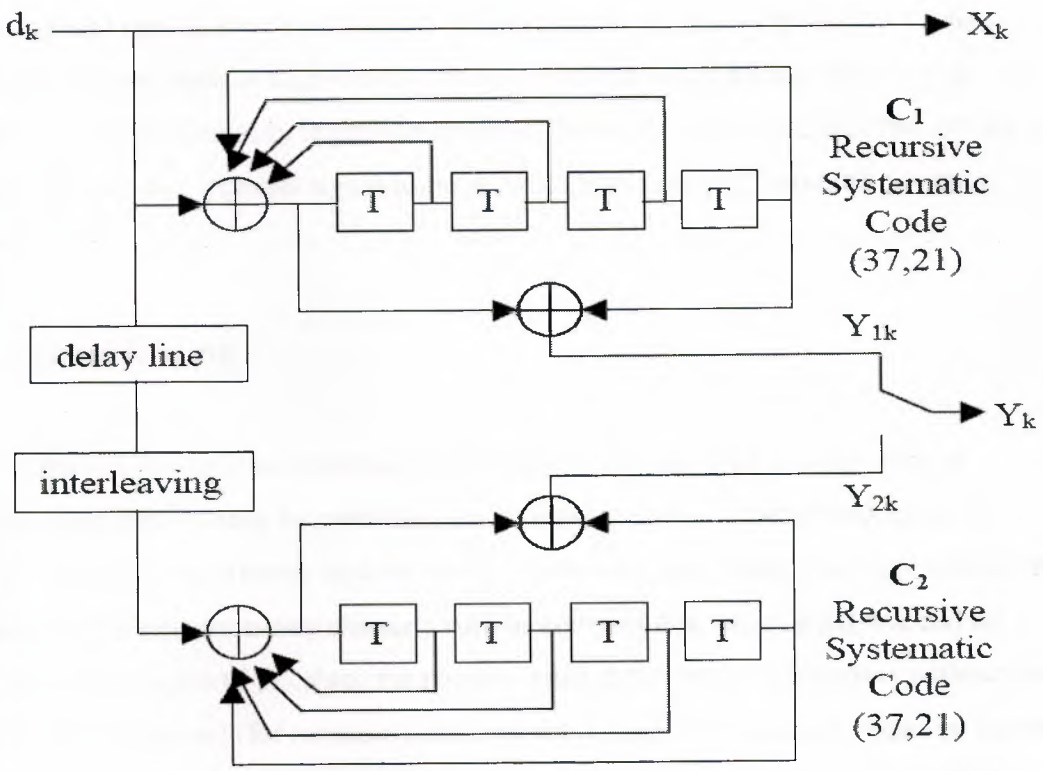

Figure 2.5. Berrou's Encoder for CTC

CHAPTER 3

LINEAR BLOCK CODES

3.1 Introduction to Linear Block Codes

Channel coding is an error-control technique used for providing robust data transmission through imperfect channels by adding redundancy to the data. There are two important classes of such coding methods: block and convolutional. Forward error correction (FEC) is the name used when the receiving equipment does most of the work. In the case of block codes, the decoder looks for errors and, once detected, corrects them (according to the capability of the code). The technique has become an important signal-processing tool used in modern communication systems and in a wide variety of other digital applications such as high-density memory and recording media. Such coding provides system performance improvements at significantly lower cost than through the use of other methods that increase signal-to-noise ratio (SNR) such as increased power or antenna gain.

3.2 Channel Coding

Channel coding involves data transformations that are used for improving a system's error performance by enabling a transmitted message to better withstand the effects of channel impairments such as noise, interference, and fading. For applications that use simplex channels (one-way channels such as compact disk recordings), the coding techniques must support FEC since the receiver must detect and correct errors without the use of a reverse channel (for retransmission requests). Such FEC techniques can be thought of as vehicles for accomplishing desirable tradeoffs that can reduce bit error rate (BER) at a fixed power level or allow a specified error rate at a reduced power level at the cost of increased bandwidth (or transmission delay) and a processing burden. A data or message

vector $\mathbf{m} = m_1, m_2, \ldots, m_k$ containing k message elements from an alphabet is transformed by the block code into a longer code vector or code word $\mathbf{U} = u_1, u_2, \ldots, u_n$ containing n code elements constructed from the same alphabet. The elements in the alphabet have a one to one correspondence with elements drawn from a finite field.

Finite fields are referred to as Galois fields, after the French mathematician Evariste Galois (1811–1832). A Galois field containing q elements is denoted GF(q), with the simplest such finite field being GF(2), the binary field with elements (1, 0), which have the obvious connection to the logical symbols (1, 0) called bits. When we deal with fields that contain more than two elements, these nonbinary elements are encoded as binary m-tuples (m-bit sequences). Then the elements are processed as binary words according to the rules of the field in much the same way that decimal integers were encoded as binary-coded decimal (BCD) symbols in early computers. and in contemporary calculators.

The number of output elements n (code bits) and input elements k (data bits) characterizing a block code are denoted by the ordered pair (n, k). Often, the designation (n, k, t) is used to indicate that the code is capable of correcting t-errors in the n-element code word. For transmitting the code bits (comprising **U**) with waveforms, a common practice is to use bipolar pulses with values (+1,−1) to represent the binary logic levels (1, 0), respectively. For a radio system, such pulses are modulated on to a carrier wave, typically denoted $s(t)$.

Channel impairments are responsible for transforming a transmitted waveform $s(t)$ into a corrupted waveform $r(t) = s(t) + n(t)$, which is received and processed by a demodulator/ detector. The demodulator recovers samples of the corrupted waveform, and the detector interprets the digital meaning of that waveform. A commonly used model for $n(t)$ is that of an additive white Gaussian noise (AWGN) process. Noise, interference, and channel distortion mechanisms account for the detector making errors. Consequently, instead of accurately reproducing the bipolar pulse values or logic levels (representing **U**), the detector might instead output a corrupted version **r**, written as

$$\mathbf{r} = \mathbf{U} + \mathbf{e} \quad (1) \qquad (3.1)$$

where $\mathbf{r} = r_1, r_2, \ldots, r_n$ represents a received block of n elements, and $\mathbf{e} = e_1, e_2, \ldots, e_n$ represents the corruption, referred to as the *error sequence* or *error pattern*.

3.2.1 Hard Decisions and Soft Decisions

In Figure.3.1, each detected element r_i of the received vector **r** can be described as a quantized-amplitude decision.

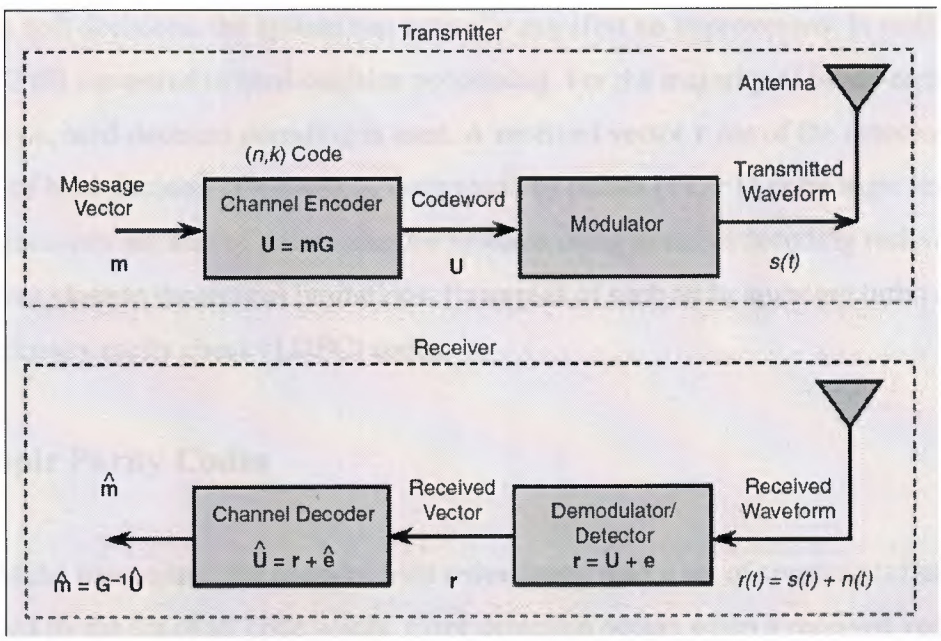

Figure.3.1 channel encoding/decoding

The decision may simply answer the question "Is the amplitude greater or less than zero?" yielding a binary decision of 1 or 0. Such a decision is called a hard decision because the detector firmly selects one of two levels. Sometimes the detector's decision may answer multiple questions such as "Is the amplitude greater or less than zero, *and* is it greater or less than some reference level?" For binary signaling, such multipart decisions are called *soft decisions*; they offer the decoder side information about the SNR of the corrupted analog waveform.

A soft decision might tell the decoder, "this signal has a positive amplitude, but it is not very far from the zero amplitude" or "this signal has a positive amplitude, and it is quite far from the zero amplitude." The most popular soft-decision format entails eight-level signal quantization, which can be interpreted as a hard decision plus a measure of confidence. The figure-of-merit for the error performance of a digital communication system is usually expressed as a normalized SNR, known as the ratio of bit energy to noise power spectral density, E_b/N_o. The coding gain or benefit provided by an error-correcting code to a system can be defined as the "relief" or reduction in required E_b/N_o that can be realized due to the code. For an AWGN channel, when the detector presents the decoder with such soft decisions, the system can typically manifest an improvement in coding gain of about 2 dB compared to hard-decision processing. For the majority of block-code applications, hard-decision decoding is used. A received vector **r** out of the detector is made up of hard-decision components, designated by pulses (+1, −1) or by logic levels (1, 0). Soft decisions are also of great value for systems using iterative decoding techniques that operate close to theoretical limitations. Examples of such techniques are turbo codes and low density parity check (LDPC) codes.

3.3 Simple Parity Codes

At the transmitter, the encoder adds redundancy with a set of constraints that must be satisfied by the set of all code words. Error detection occurs when a received vector does not satisfy the constraints. The simplest approach to error detection modifies a binary data sequence into a code word by appending an extra bit called a *parity bit*. When using the constraint that a code word must contain an even number of ones, the scheme is referred to as *even parity* (the constraint of an odd number of ones is called *odd parity*). To establish the even-parity condition, the parity bit p is formed, as the modulo-2 sum of the message bits, as

$$p = m_1 \oplus m_2 \oplus m_1 \oplus \cdots \oplus m_k \tag{3.2}$$

where the symbol $\oplus$ indicates modulo-2 addition. The test (even-parity check) conducted by the receiver verifies that the modulo-2 sum of the parity plus message bits in the received sequence **r** is zero. If the sequence fails the test, an error has been detected. We refer to the test result as the syndrome *S*, written as

$$S = r_1 \oplus r_2 \oplus \cdots \oplus r_k \oplus r_{k+1} \tag{3.3}$$

The syndrome in (3.3) can be modeled as the modulo-2 sum of the transmitted sequence and the error sequence, as

$$S = (m_1 \oplus e_1) \oplus (m_2 \oplus e_2) \oplus \cdots \oplus (m_k \oplus e_k) \oplus (p \oplus e_{k+1}) \tag{3.4}$$

$$\begin{aligned} S &= (m_1 \oplus m_2 \oplus \cdots \oplus m_k \oplus p) \oplus (e_1 \oplus e_2 \oplus \cdots \oplus e_k \oplus e_{k+1}) \\ &= 0 \oplus (e_1 \oplus e_2 \oplus \cdots \oplus e_k \oplus e_{k+1}). \end{aligned} \tag{3.5}$$

When factored into separate message and error sequences as seen in (3.5), we recognize that the syndrome tests both the transmitted sequence and the error sequence, but since the syndrome of the transmitted sequence is zero, the syndrome is only responding to the error sequence. For the case of a single parity bit, as in (3.5), only an odd number of errors can be detected, since an even number of errors will yield the syndrome $S = 0$.

A single parity bit can only be used for error detection. To perform error correction, we require additional information to locate the error positions; the code word needs to be embedded with more than a single parity bit. A simple example of a code that appends additional parity bits to the message sequence is shown in (3.6. Here, a set of eight message elements is packed into a two-dimensional array from which we form parity for each row and parity for each column.

The appended array can be rearranged into a code word sequence **U** as

$$\mathbf{U} = m1\ m2\ m3\ m4\ m5\ m6\ m7\ m8\ p1\ p2\ p3\ p4\ p5\ p6. \tag{3.6}$$

When **U** is received, it can be mapped back to the same two-dimensional array, and a set of syndromes can be calculated corresponding to each row and each column. A single error located anywhere in the message positions will cause a nonzero syndrome in a row and in a column, and thus the intersection of the row and column corresponding to the parity failure contains the single error. One should conclude that a block code capable of detecting and correcting error sequences needs to have multiple parity symbols appended to the data message and multiple syndromes generated during the parity checks at the receiver.

3.4 The Generator Matrix and Systematic Codes

The most general form of the parity generation process, in which each code element u_i of the code word **U** is a weighted sum of message elements, can be written in the form of a vector matrix equation as

$$\mathbf{U} = \mathbf{mG} \tag{3.7}$$

$$[u_1\ u_2\ u_3\ \cdots\ u_n] = [m_1\ m_2\ m_3 \cdots m_k] \times \begin{bmatrix} \mathcal{G}1,1 & \mathcal{G}1,2 & \mathcal{G}1,3 & \cdots & \mathcal{G}1,n \\ \mathcal{G}2,1 & \mathcal{G}2,2 & \mathcal{G}2,3 & \cdots & \mathcal{G}2,n \\ \mathcal{G}3,1 & \mathcal{G}3,2 & \mathcal{G}3,3 & \cdots & \mathcal{G}3,n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{G}k,1 & \mathcal{G}k,2 & \mathcal{G}k,3 & \cdots & \mathcal{G}k,n \end{bmatrix}$$

where the entries of the matrix **G**, called the *generator matrix*, represent weights (field-element coefficients), and the multiplication operation follows the usual rules of matrix multiplication. The product of a message row-vector **m** with the *i*th column-vector of **G** forms *ui* a weighted sum of message elements representing the *i*th element of the code word row-vector **U**. For a binary code, the data elements as well as the matrix weights are 1s and 0s.

A useful variant of the code word **U** is one in which the vector of message elements is embedded, without change, in the code word along with an appended vector of parity

elements. When the code word is constrained in this manner, the code is called a *systematic code*. To form a systematic code the generator matrix **G** can be modified in terms of submatrices **P** and $\mathbf{I}_k$ as follows:

$$\mathbf{U} = \mathbf{mG} = \mathbf{m}[\mathbf{P}|\mathbf{I}_k] \tag{3.8}$$

$$\mathbf{U} = [u_1\ u_2\ u_3\ \cdots\ u_n] = [m_1\ m_2\ m_3\ \cdots\ m_k]$$
$$\times \underbrace{\begin{bmatrix} \mathcal{G}_{1,k+1} & \cdots & \mathcal{G}_{1,n} \\ \mathcal{G}_{2,k+1} & \cdots & \mathcal{G}_{2,n} \\ \mathcal{G}_{3,k+1} & \cdots & \mathcal{G}_{3,n} \\ \vdots & \ddots & \vdots \\ \mathcal{G}_{k,k+1} & \cdots & \mathcal{G}_{k,n} \end{bmatrix}}_{\mathbf{P}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}}_{\mathbf{I}_k}$$

where **P** is the parity portion of **G**, and $\mathbf{I}_k$ is a *k*-by-*k* identity submatrix (ones on the main diagonal, and zeros elsewhere).

3.5 Weight and Distance Properties

The *Hamming weight* $w(\mathbf{U})$ of a code word **U** is defined as the number of nonzero elements in **U**. For a binary vector (or a nonbinary vector with field elements represented in binary form), this is equivalent to the number of ones in the vector. For example, if **U** = 1 0 0 1 0 1 1 0 1, then $w(\mathbf{U}) = 5$. The *Hamming distance* $d(\mathbf{U}, \mathbf{V})$ between two binary code words **U** and **V** is defined as the number of bit positions in which they differ. For example

if U = 1 0 0 1 0 1 1 0 1

and V = 0 1 1 1 1 0 1 0 0

then *d*(U, V) = 6.

By the properties of linear block codes, we say that the sum of two different codeword is a third codeword.

$$\mathbf{W = U + V = 1\,1\,1\,0\,1\,1\,0\,0\,1}$$

Thus, we observe that the Hamming distance between two code words is equal to the Hamming weight of the summed vectors: that is, $d(\mathbf{U},\mathbf{V}) = w(\mathbf{U}+\mathbf{V})$. Also, note that the Hamming weight of a code word is equal to its Hamming distance from the all-zeros vector.

3.6 Decoding Task

The decoding task can be stated as follows: Having received the vector **r**, find the best estimate of the particular code word **U***i* that was transmitted. The optimal decoder strategy is to minimize the decoder error probability, which is the same as maximizing the probability $P(\hat{\mathbf{U}} = \mathbf{U}i\,|r)$. If all code words are equally likely and the channel is memoryless, this is equivalent to maximizing $P(\mathbf{r}|\mathbf{U}i)$, the conditional probability density function (pdf) of **r**, expressed as

$$p(\mathbf{r}|\mathbf{U}_i) = \max_{\text{over all } \mathbf{U}_j} p(\mathbf{r}|\mathbf{U}_j) \tag{3.9}$$

where the pdf, conditioned on having sent **U** , is called the likelihood of $\mathbf{U}_j$. Equation (3.9), known as the maximum likelihood (ML) criterion , can be used for finding the "most likely" $\mathbf{U}_j$ that was transmitted. For algorithms using Hamming distances, the likelihood of $\mathbf{U}_j$ with respect to **r** is inversely proportional to the distance between **r** and $\mathbf{U}_j$, denoted $d(\mathbf{r}, \mathbf{U}_i)$. Therefore, we can express the decoder decision rule as: Decide in favor of $\mathbf{U}_j$ if

$$d(\mathbf{r}, \mathbf{U}_i) = \min d(\mathbf{r}, \mathbf{U}_j) \quad \text{over all } \mathbf{U}_j. \tag{3.10}$$

3.7 Error-Detecting and Error-Correcting Capability

The smallest member of the set is called the minimum distance of the code and is denoted d_{min}. To find d_{min}, we need not search the set of code words in a pairwise fashion. Because of the closure property, we need only find the nonzero code word having the minimum weight. The minimum distance, like the weakest link in a chain, gives us a measure of the code's capability (indicates the smallest number of channel errors that can lead to decoding errors). Figure 3.2 illustrates the distance between two code words **U** and **V** using a number line calibrated in Hamming distance, where each black dot represents a corrupted code word. In this example, let the distance $d(\mathbf{U}, \mathbf{V})$ be the minimum distance $d_{min} = 5$. Figure 3.2(a) illustrates the reception of a vector $\mathbf{r}_1$, which is distance 1 from **U** and distance 4 from **V**. An error-correcting decoder, following the ML strategy, will select **U** upon receiving $\mathbf{r}_1$. If $\mathbf{r}_1$ had been the result of a 1-b corruption to the transmitted code word **U**, the decoder has successfully corrected the error. But if $\mathbf{r}_1$ had been the result of a 4-b corruption to the transmitted code word **V**, the result is a decoding error. Similarly a double error in transmission of **U** might result in the received vector **r**2, which is distance 2 from **U** and distance 3 from **V**, as shown in Figure 3.2(b). Here too, the decoder will select **U** upon receiving **r**2. A triple error in transmission of **U** might result in a received vector **r**3 that is distance 3 from **U** and distance 2 from **V**, as shown in Figure 3.2(c). Here the decoder will select **V** upon receiving $\mathbf{r}_3$ and, given that **U** was transmitted, will have made a decoding error. From Figure 3.2, one can see that if the task is error detection (and not correction), then as many as 4-b errors can be detected. But, if the task is error-correction, the decision to choose **U** if **r** falls in region 1, and **V** if **r** falls in region 2, illustrates that this code (with $d_{min} = 5$) can correct as many as 2-b errors. We can generalize a linear block code's error-detection capability ε and error-correction capability t as

$$\varepsilon = d_{\min} - 1$$
$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \tag{3.11}$$

where the notation $\_x\_$, called the floor of x, means the largest integer not to exceed x (in other words, round down if not an integer).

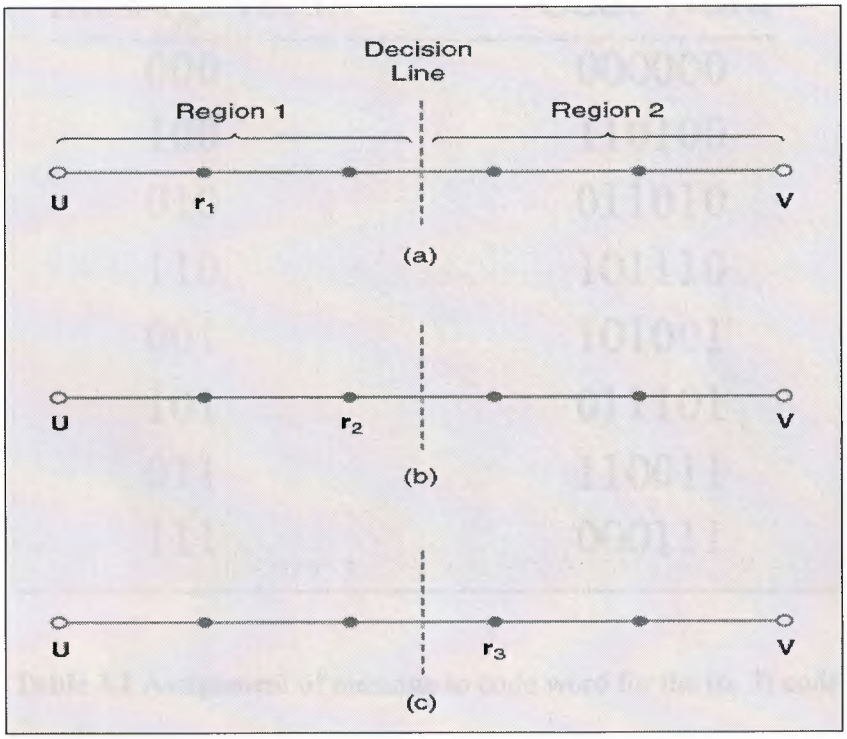

Figure 3.2. Error correction and detection capability

3.8 A (6, 3) Linear Block Code Example

Table.3.1 describes a code word-to-message assignment for a (6, 3) code, where the rightmost bit represents the earliest (and most-significant) bit. For each code word, the rightmost $k = 3$ bits represent the message (hence, the code is in systematic form).

Since $k = 3$, there are $2k = 23 = 8$ message vectors, and therefore there are eight code words. Since $n = 6$, then within the vector space $Vn = V6$ there are a total of $2n = 26 = 64$ 6-tuples. It is easy to verify that the eight code words shown in Table.3.1 form a subspace of $V6$ (the all-zeros vector is one of the code words, and the sum of any two code words is also a code word). Note that for a particular (n, k) code, a unique assignment does not exist; however, neither is there complete freedom of choice.

Message vector	Code Word
000	000000
100	110100
010	011010
110	101110
001	101001
101	011101
011	110011
111	000111

Table 3.1 Assignment of message to code word for the (6, 3) code

3.8.1 A Generator Matrix for the (6, 3) Code

For short codes, the message-to-code-word mapping in Table 3.2 can be accomplished via a lookup table, but if k is large, such an implementation would require a prohibitive amount of memory. Fortunately, by using a generator matrix **G.** It is possible to reduce complexity by generating the required code words as needed instead of storing them. Since the set of code words is a k-dimensional subspace of the n-dimensional vector space, it is always possible to find a set of n-tuples (row-vectors of the matrix **G**), fewer than 2^k that can generate all the 2^k code words of the subspace. The generating set of vectors is said to span

the subspace. The smallest linearly independent set that spans the subspace is called a basis of the subspace, and the number of vectors in this basis set is the dimension of the subspace. Any basis set of k linearly independent n-tuples $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_k$ (that spans the subspace) can be used to form a generator matrix **G**. This matrix can then be used to generate the required code words, since each code word is a linear combination of $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_k$. That is, each code word **U** within the set of 2^k code words can be described by

$$\mathbf{U} = m_1\mathbf{V}_1 + m_2\mathbf{V}_2 + \cdots + m_k\mathbf{V}_k \tag{3.12}$$

where each m_i = (1 or 0) is a message bit and the index $i = 1, \ldots, k$ represents its position. In general, we describe this code generation in terms of multiplying a message vector **m** by a generator matrix **G**. For the (6, 3) code introduced earlier, we can fashion a generator matrix **G** in systematic form, as

$$\mathbf{G} = \begin{bmatrix} \mathbf{V}_1 \\ \mathbf{V}_2 \\ \mathbf{V}_3 \end{bmatrix} = \left[\begin{array}{ccc|ccc} 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right] = \left[\underbrace{\begin{array}{ccc} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{array}}_{\mathbf{P}} \quad \underbrace{\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}}_{\mathbf{I}_k}\right] \tag{3.13}$$

where **P** and $\mathbf{I}_k$ represent the parity and identity sub-matrices, respectively, and $\mathbf{V}_1, \mathbf{V}_2$, and $\mathbf{V}_3$ are three linearly independent vectors (a subset of the eight code vectors) that can generate all the code words, made up of the weights $\{g_{ij}\}$. Note also that the sum of any two generating vectors does not yield any of the other generating vectors since linear independence is, in effect, the opposite of closure. The generator matrix **G** completely defines the code and represents a compact way of describing a block code. If the encoding operation utilizes storage, then the encoder only needs to store the k rows of **G** instead of

all 2^k code words of the code. For systematic codes, the encoder only stores the **P** sub-matrix; it doesn't need to store the identity portion of **G**.

3.8.2 Error Detection and the Parity-Check Matrix

At the decoder, a method of verifying the correctness of a received vector is needed. Let us define a matrix **H**, called the *parity-check matrix*, that will help us decode the received vectors. For each $(k \times n)$ generator matrix **G**, one can construct an $(n - k) \times n$ matrix **H**, such that the rows of **G** are orthogonal to the rows of **H**. Another way to express this orthogonality is to say that $\mathbf{GH}^{\mathbf{T}} = \mathbf{0}$, where $\mathbf{H}^{\mathbf{T}}$ is the transpose of **H**, and **0** is a $k \times (n - k)$ all-zeros matrix. $\mathbf{H}^{\mathbf{T}}$ is an $n \times (n - k)$ matrix (whose rows are the columns of **H**). To fulfill the orthogonality requirements of a systematic code, the **H** matrix can be written as $\mathbf{H} = [\mathbf{I}_{n-k} | \mathbf{P}^{\mathbf{T}}]$, where $\mathbf{I}_{n-k}$ represents an $(n - k) \times (n - k)$ identity submatrix and **P** represents the parity submatrix defined in (3.13). Since by this definition of **H**, we see that $\mathbf{GH}^{\mathbf{T}} = \mathbf{0}$, and since each **U** is a linear combination of the rows of **G**, then any vector **r** is a code word generated by the matrix **G**, if and only if

$$\mathrm{r}\mathrm{H}^{T} = 0. \tag{3.14}$$

Equation (3.14) is the basis for verifying whether a received vector **r** is a valid code word.

3.9 Towards Error Correction: Syndrome Testing

We can model the received word r as a summation r = c + e, where c is the transmitted codeword and e is the error pattern induced by the channel noise. If we could know e then the codeword is given as c = r + e. But, how can we get e from r? Consider

$$\underline{r}\boldsymbol{H}^{T} = \underline{c}\boldsymbol{H}^{T} + \underline{e}\boldsymbol{H}^{T} = \underline{e}\boldsymbol{H}^{T} = \underline{s}. \tag{3.15}$$

The length-(n - k) word ***s*** is called the syndrome [4] of the received word ***r***. Though the syndrome s depends only on the error patter e, there are many error patterns which can yield the same syndrome. Indeed, for any codeword c_0 other than c, c_0 + e is an error pattern having the same syndrome s. The collection of all the error patterns that give the same syndrome is called a coset of the code. Any two error patterns in a coset are different by a codeword, so each coset contains 2^k different error patterns. Since there are 2^n possible error patterns, there must be $2^n=2^k = 2^{n-k}$ different cosets corresponding to the 2n_k different syndromes. In other word, for each syndrome, there is a coset of 2k error patterns which can generate it. Then, which one is the right one?

The minimum Hamming distance decoder picks up a codeword c0 such that r = c0 + e0 where e0 has the smallest possible weight. Therefore, for a given syndrome, the decoder should choose among the corresponding coset the lowest-weight error patter. The error pattern with lowest weight in a coset is called the coset leader.

As a summary, the minimum Hamming distance decoder for linear block codes works this way:

When receiving a word r,

1. Compute the syndrome $s = rH^T$;
2. If s = 0, choose codeword c_0 = r and go to step 4; if s_6= 0, find the coset leader e' corresponding to s;
3. Choose the code c' = r + e;
4. Mapping c' back to the message word.

At step 2, if there are more than one candidate coset leader, the decoding fails. This decoding procedure is called syndrome decoding. Syndrome decoder only needs to store the parity check matrix H and the 2^{n-k} coset leaders, requiring less memory than simple minimum Hamming distance decoder which needs to store all 2^k codewords. In addition, the syndrome decoder only involves simple matrix computation and Figure(3.3) looking up which can be implemented easily using digital processors or circuits.

syndrome	0000000	1000111	0101011	0011101	1101100	1011010	0110110	1110001
0001	0000001	1000110	0101010	0011100	1101101	1011011	0110111	1110000
0010	0000010	1000101	0101001	0011111	1101110	1011000	0110100	1110011
0100	0000100	1000011	0101111	0011001	1101000	1011110	0110010	1110101
1000	0001000	1001111	0100011	0010101	1100100	1010010	0111110	1111001
1101	0010000	1010111	0111011	0001101	1111100	1001010	0100110	1100001
1011	0100000	1100111	0001011	0111101	1001100	1111010	0010110	1100001
0111	1000000	0000111	1101011	1011101	0101100	0011010	1110110	0110001
0011	0000011	1000100	0101000	0011110	1101111	1011001	0110101	1110010
0110	0000110	1000001	0101101	0011011	1101010	1011100	0110000	1110111
1100	0001100	1001011	0100111	0010001	1100000	1010110	0111010	1111101
0101	0011000	1011111	0110011	0000101	1110100	1000010	0101110	1101001
1010	0001010	1001101	0100001	0010111	1100110	1010000	0111100	1111011
1001	0010100	1010011	0111111	0001001	1111000	1001110	0100010	1100101
1111	0010010	1010101	0111001	0001111	1111110	1001000	0100100	1100011
1110	0111000	1111111	0010011	0100101	1010100	1100010	0001110	1001001

Figure 3.3 The syndrome table

3.9.1 The Standard Array and Error Correction

The syndrome test gives us the ability to detect errors and to correct some of them. Let us arrange the 2^n n-tuples that represent possible received vectors in an array, called the *standard array*. This array can be thought of as an organizational tool or a filing cabinet that contains all of the possible vectors in the space, nothing missing, and nothing replicated. The first row contains the set of all the $2k$ code words $\mathbf{U}_1, \mathbf{U}_2, \ldots, \mathbf{U}_{2k}$ starting with the all-zeros code word designated $\mathbf{U}_1$. In this array, each row, called a *coset*, consists of an error pattern in the leftmost position, called a *coset leader*, followed by corrupted code words (corrupted by that error pattern). Thus the first column, made up of coset leaders, displays all of the correctable error patterns. The structure of the standard array for an (n, k) code, is

$$
\begin{array}{ccccccc}
U_1 & U_2 & \cdots & U_i & \cdots & U_{2^k} \\
e_2 & U_2 + e_2 & \cdots & U_i + e_2 & \cdots & U_{2^k} + e_2 \\
e_3 & U_2 + e_3 & \cdots & U_i + e_3 & \cdots & U_{2^k} + e_3 \\
\vdots & \vdots & & \vdots & & \vdots \\
e_j & U_2 + e_j & \cdots & U_i + e_j & \cdots & U_{2^k} + e_j \\
\vdots & \vdots & & \vdots & & \vdots \\
e_{2^{n-k}} & U_2 + e_{2^{n-k}} & \cdots & U_i + e_{2^{n-k}} & \cdots & U_{2^k} + e_{2^{n-k}}
\end{array}
$$

Note that code word $\mathbf{U}_1$ plays two roles. It is one of the code words (the all-zeros code word), as well as the error pattern $\mathbf{e}_1$, that is the pattern that introduces no errors so that $\mathbf{r} = \mathbf{U} + \mathbf{e}_1 = \mathbf{U}$. Since the array contains all the 2^n *n*-tuples in the space, each *n*-tuple appearing only once, and each coset or row contains 2^n *n*-tuples, we can compute the number of rows in the array by dividing the total number of entries by the number of columns. Thus, in any standard array, there are $2^n/2^k = 2^{n-k}$ cosets. At first glance, the benefits of this tool seem limited to *small* block codes, because for code lengths beyond n = 20 there are millions of *n*tuples in *Vn*. Even for large codes, however, the standard array concept allows visualization of important performance issues, such as bounds on error-correction capability, as well as possible tradeoffs between error correction and detection. In the sections that follow, we show how the decoding algorithm replaces a received corrupted code word $\mathbf{r} = \mathbf{U} + \mathbf{e}$ with an estimate _U of the valid code word **U**. If code word $\mathbf{U}_i$ is transmitted over a noisy channel, and the corrupting error pattern is a coset leader, then the received vector will be decoded correctly into the transmitted code word $\mathbf{U}_i$. If the error pattern is not a coset leader, an erroneous decoding will result.

3.9.2 The Syndrome of a Coset

The name coset is short for "a set of numbers having a common feature." What do the members of a coset have in common? Each member has the same syndrome. We confirm this as follows: If $\mathbf{e}_j$ is the coset leader or error pattern of the j^{th} coset, then $\mathbf{U}_i + \mathbf{e}_j$ is an *n*-tuple in this coset. From (3.15), the syndrome of this *n*-tuple can be written as

$$S = rH^T = (U_i + e_j)H^T = U_iH^T + e_jH^T. \tag{3.16}$$

Since $\mathbf{U}_i$ is a valid transmitted code word, then $\mathbf{U}_i\mathbf{H}^T = \mathbf{0}$, since the parity check matrix $\mathbf{H}$ was constructed with this feature in mind. We can therefore express (3.16) as

$$S = rH^T = e_jH^T. \tag{3.17}$$

Thus, the syndrome test, performed on either a corrupted code vector or on the error pattern that caused it, yields the same syndrome. Equation (3.17) establishes that the syndrome is in fact only responding to the error pattern, which was similarly shown in (3.5) for a simple parity code. An important property of linear block codes, fundamental to the decoding process, is that the mapping between correctable error patterns and syndromes is one to one. The syndrome for each coset is different from that of any other coset in the code; it is the syndrome that is used to estimate the error pattern, which then allows for the errors to be corrected.

3.9.3 Locating the Error Pattern

Returning to the (6, 3) code example, we arrange the 26 = 64 6-tuples in a standard array. The valid code words are the eight vectors in the first row, and the correctable error patterns are the seven nonzero coset leaders in the first column. Note that all 1-b error patterns are correctable. Also note that after exhausting all 1-b error patterns, there remains some error-correcting capability since we have not yet accounted for all 64 6-tuples. There is still one unassigned coset leader; therefore, there remains the capability of correcting one additional error pattern. We have the flexibility of choosing this error pattern to be any of the *n*-tuples in the remaining coset. This final correctable error pattern was chosen, somewhat arbitrarily, to be the 2-b error pattern 010001. The error-correcting task performed by the decoder can be implemented to yield correct messages if, and only if, the error pattern caused by the channel is one of the coset leaders. For the (6, 3) code example,

we now use Equation 3.17 to determine the syndrome (symptom) corresponding to each correctable error pattern (ailment), by computing $\mathbf{e}_j \mathbf{H}^T$ for each coset leader, as follows:

$$S = e_j \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix} \tag{3.18}$$

The results are listed in Figure 3.3. Since each syndrome in the table has a one-to-one relationship with the listed error patterns, solving for a syndrome earmarks the particular error pattern corresponding to that syndrome .

3.9.4 Error Correction Decoding

Given a received vector **r** at the input of the decoder, we summarize the procedure for deciding on **U** and finally on _**m** as follows: 1) calculate the syndrome of **r** using $\mathbf{S} = \mathbf{rH}^T$ and 2) use Table 3 to locate the coset leader (error pattern) $\mathbf{e}_j$, whose syndrome equals $\mathbf{rH}^T$. This error pattern is assumed to be the corruption caused by the channel and will be our estimate ^**e** of the error, estimate of the code word _**U** is identified as _**U** = **r** + ^**e**. We can say that the decoder obtains an estimate of the transmitted code word by removing an estimate of the error ^**e** (in modulo-2 arithmetic, the act of removal is effected via addition). This step can be written as

$$\hat{\mathbf{U}} = \mathbf{r} + \hat{\mathbf{e}} = (\mathbf{U} + \mathbf{e}) + \hat{\mathbf{e}} = \mathbf{U} + (\mathbf{e} + \hat{\mathbf{e}}) \tag{3.19}$$

If the estimated error pattern is the same as the actual error pattern, that is, if ^**e** = **e**, then the estimate _**U** is equal to the transmitted code word **U**. However, if the error estimate is incorrect, the decoder will choose a code word that was not transmitted, resulting in a

decoding error. As an example from the (6, 3) code, assume that code word **U** = 101110 corresponding to **m** = 110 (see Table 2) is transmitted and that the vector **r** = 001110 is received. From 3.17 we compute the syndrome as

$$S = [0\,0\,1\,1\,1\,0]\,H^T = 1\,0\,0 \tag{3.20}$$

From Table 3.2, we can verify that the error pattern is **e** = 100000. Then, using (3.19), the corrected vector is estimated as

$$\hat{U} = r + \hat{e} = 0\,0\,1\,1\,1\,0 + 1\,0\,0\,0\,0\,0 = 1\,0\,1\,1\,1\,0 \tag{3.21}$$

Error pattern e	Syndrome S
000000	0 0 0
000001	1 0 1
000010	0 1 1
000100	1 1 0
001000	0 0 1
010000	0 1 0
100000	1 0 0
010001	1 1 1

Table 3.2 Syndrome lookup table

Since in this example, the estimated error pattern is the actual error pattern, the error correction procedure yields _**U** = **U**, which means that the output _**m** will correspond to the actual message 110. Note that the process of decoding a corrupted code word by first detecting and then correcting an error can be compared to a familiar medical analogy. A patient **r** (potentially corrupted code word) enters a medical facility (decoder). The examining physician performs a diagnostic test (multiplies **r** by $\mathbf{H}^T$) to find a symptom

(syndrome). Imagine that the physician finds characteristic spots on the patient's X rays. An experienced physician would immediately recognize the correspondence between the symptom and the ailment, for example tuberculosis. A novice physician might have to refer to a medical handbook (Table 3.2) to associate the symptom (syndrome) with the ailment (error pattern). The final step is to provide medication **_e**. If **_e** is the proper medication (if **_e** = **e**), then the ailment is removed, as seen in Equation 3.19. In the context of binary codes and the medical analogy, (3.19) and (3.21) reveal an unusual type of medicine practiced here. The patient is cured by reapplying the original ailment, a process that works because in the binary field 1 + 1 = 0.

000000	110100	011010	101110	101001	011101	110011	000111
000001	110101	011011	101111	101000	011100	110010	000110
000010	110110	011000	101100	101011	011111	110001	000101
000100	110000	011110	101010	101101	011001	110111	000011
001000	111100	010010	100110	100001	010101	111011	001111
010000	100100	001010	111110	111001	001101	100011	010111
100000	010100	111010	001110	001001	111101	010011	100111
010001	100101	001011	111111	111000	001100	100010	010110

Table 3.3 Standard array for a (6, 3) code

3.9.5 Decoder Implementation

When the code is short as in the case of the (6, 3) code, the decoder can be implemented with simple circuitry. The steps that such a circuit must take are: 1) calculate the syndrome, 2) locate the error pattern corresponding to that syndrome, and 3) modulo-2 add the estimated error pattern to the received vector to yield an estimate of the corrected vector. Consider the circuit in Figure 3.4 , made up of exclusive-OR gates and AND gates that can accomplish these decoder steps for any single-error pattern in the (6, 3) code. From Table 3.2 and (3.17), we can write an expression for the syndrome bits s_1, s_2, s_3 in terms of the received code word bits $r_1, \ldots, r_6$ as

$$S = rH^T$$

$$S = [s_1\ s_2\ s_3] = [r_1\ r_2\ r_3\ r_4\ r_5\ r_6] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

and

$$\begin{aligned} s_1 &= r_1 + r_4 + r_6 \\ s_2 &= r_2 + r_4 + r_5 \\ s_3 &= r_3 + r_5 + r_6. \end{aligned} \tag{3.22}$$

We use these syndrome expressions for wiring up the circuit in Figure 3.4. The exclusive-OR gate provides the same operation as modulo-2 arithmetic and hence uses the same symbol. A small circle at the termination of any line entering the AND gate indicates the logic complement of the signal. The corrupted signal **r** enters the decoder at two places simultaneously. At the upper part of the circuit, the syndrome **S** is computed, and at the lower part that syndrome is transformed to its corresponding error pattern **e**. The error is removed by adding it back to the received vector yielding the corrected code word **U**. Note that, Figure 3.4 has been drawn to emphasize the algebraic decoding steps, calculation of syndrome, error pattern, and corrected output. For real circuitry, the decoder would not need to deliver the entire code word; its output would consist of the message bits only. Hence, the Figure 6 circuitry becomes simplified by eliminating the gates that are shown with shading. For longer codes such an implementation is very complex, and the preferred decoding techniques conserve circuitry by using a sequential approach instead of this parallel method . It is important to emphasize that Figure 3.4 has been configured to detect and correct only single-error patterns for the (6, 3) code. Error control for a double-error pattern would require additional circuitry.

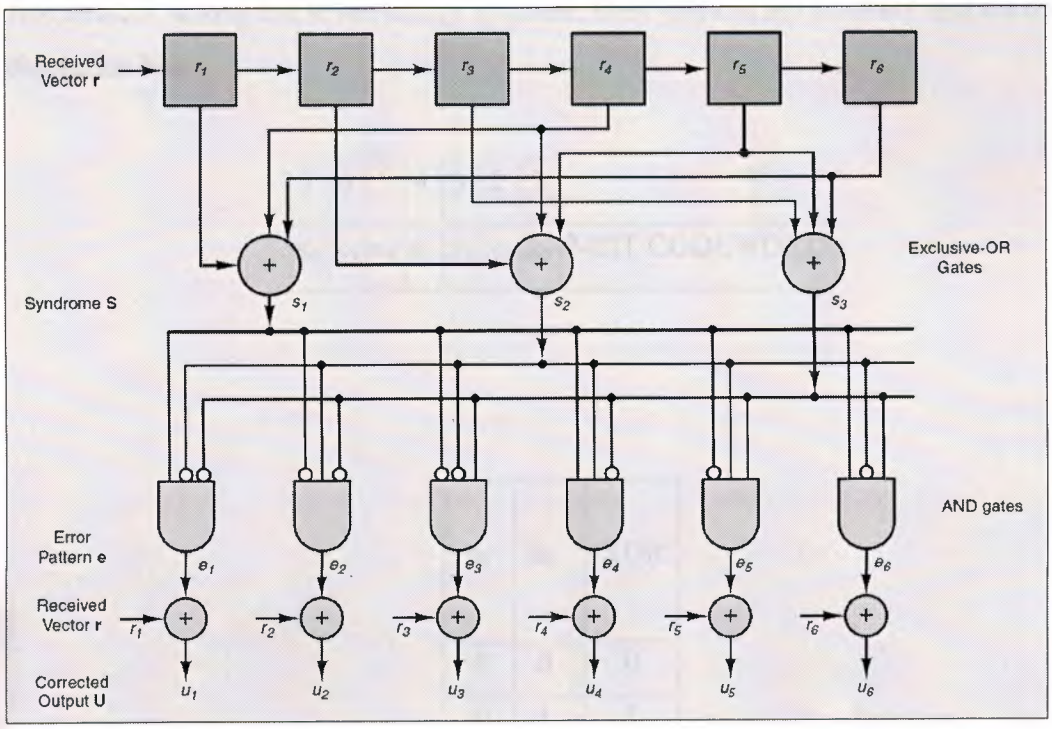

Figure.3.4 Implementation of (6, 3) code

3.10 Hamming Codes

Hamming found the first error control code, now known as [7, 4] hamming code, trying to extend the concept of parity checks to correcting errors. The [7, 4] hamming code is composed of the binary codewords of length 7, fulfilling the following parity check equations

$$\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5, x_6, x_7]$$

$$\begin{aligned} x_4 + x_5 + x_6 + x_7 &= 0 \\ x_2 + x_3 + x_6 + x_7 &= 0 \\ x_1 + x_3 + x_5 + x_7 &= 0 \end{aligned} \tag{3.22}$$

Where the (+) sign relates to XOR arithmetic relation shown in Table 3.4. This leaves only 4 free choices among the seven binary symbols, these choices are arbitrary and are the information bits.

7	6	5	4	3	2	1	
x_6	x_5	x_4	x_3	x_2	x_1	x_0	7-BIT CODEWORD

b_1	b_2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.4 modulo-2 adder(XOR)

The three parity bits (**1,2,4**) which are the extra information (redundant) are related to the data bits (**3,5,6,7**) as show in Figure.3.5 below. In this diagram, each overlapping circle corresponds to one parity bit and defines the four bits contributing to that parity computation. For example, data bit **3** contributes to parity bits **1** and **2**. Each circle (parity bit) encompasses a total of four bits, and each circle must have EVEN parity. Given four data bits, the three parity bits can easily be chosen to ensure this condition. The 3 redundant bits (parity bits) are decided by using mathematical equations related to the information data bits using the modulo-2 (XOR) arithmetic relation.

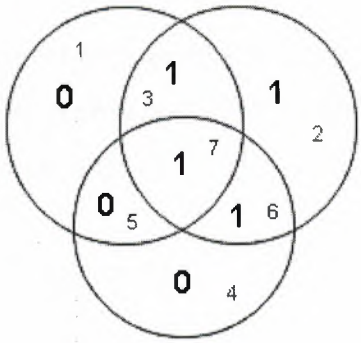

Figure 3.5 relation between bits in a codeword

3.10.1 How to get the Parity Bits

Just in this section we will say that the Hamming code 7-bits are

$$x = [\, p_0, p_1, x_0, p_2, x_1, x_2, x_3 \,]$$

$$\begin{aligned} p_0 &= x_0 + x_1 + x_2 \\ p_1 &= x_1 + x_2 + x_3 \\ p_2 &= x_0 + x_1 + x_3 \end{aligned} \tag{3.23}$$

This operation happen in the encoder of the hamming code and (see Figure 3.6)

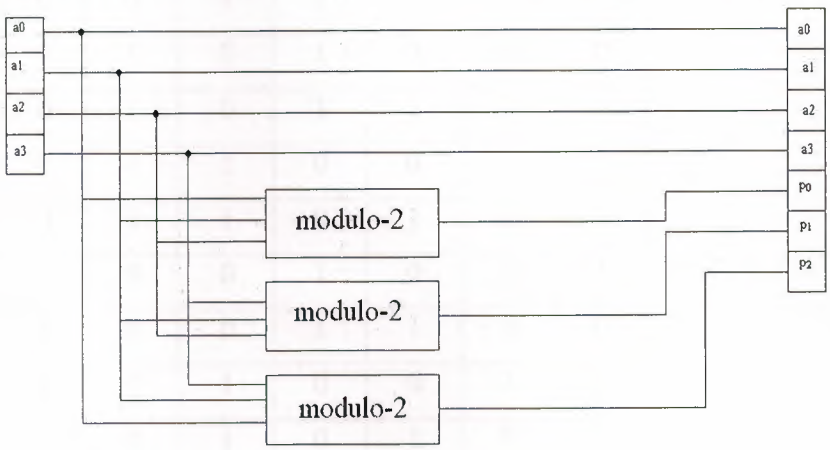

Figure 3.6 Hamming code encoder

We choose x_3, x_5, x_6 *and* x_7 as the information bits. If x is transmitted and received as y, the code can tolerate a single bit to be in error and still retrieve the information bits x_3, x_5, x_6,x_7. This is done in the following simple way: let

$$\begin{aligned} y_4 + y_5 + y_6 + y_7 &= \alpha \\ y_2 + y_3 + y_6 + y_7 &= \beta \\ y_1 + y_3 + y_5 + y_7 &= \gamma \end{aligned} \tag{3.24}$$

The vector $(\alpha, \beta, \gamma)^T$ gives the location of the error in binary forms, i.e., (1, 0, 1) means the fifth symbol is in error. We can notice that hamming codes has 2^n information messages, but that is not true. There are specific number of information messages, lets take the [7, 4] code as an example.

	7	6	5	4	3	2	1
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0
A	1	0	1	0	0	1	0
B	1	0	1	0	1	0	1
C	1	1	0	0	0	0	1
D	1	1	0	0	1	1	0

E	1	1	1	1	0	0	0
F	1	1	1	1	1	1	1

Table 3.5 Hamming [7, 4] , $d_{min} = 3$

3.10.2 Parity Check Equations

The parity check equations are conveniently expressed in the linear algebraic equation

$$\boldsymbol{Hx} = \mathbf{0} \tag{3.25}$$

where $\boldsymbol{H}$ is parity check matrix containing a one in position $\boldsymbol{(i, j)}$ if $\mathbf{x_j}$ is check in equation i.

$$\boldsymbol{H}_{[7,4]} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} = [\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3, \boldsymbol{h}_4, \boldsymbol{h}_5, \boldsymbol{h}_6, \boldsymbol{h}_7] \tag{3.26}$$

This linear algebraic immediately formulation immediately reveals some fundamental principles:

- The code is linear, i.e., $\boldsymbol{Hx_1 = 0}$, $\boldsymbol{Hx_2 = 0}$ which gives $\boldsymbol{H(x_1, x_2) = 0}$, and $\boldsymbol{x_3 = (x_1, x_2)}$ is also a codeword.

Theorem: the hamming codes are single error correcting.

Proof: A single error is identified by $\boldsymbol{h_j}$, where j is the location of error and all $\boldsymbol{h_j}$ are unique. Conversely, double errors in positions $\boldsymbol{i}$ and $\boldsymbol{j}$ are not identifiable since $\boldsymbol{h_i + h_j = h_k}$ look like a single error in position $\boldsymbol{k}$.

Hamming codes have the ability of detecting two errors but it can only correct one.

Hamming Bound: A *t*-error correcting code with M codewords must fulfill the inequality

$$M\left(1+\binom{n}{1}+\cdots+\binom{n}{t}\right)\leq 2^n \tag{3.27}$$

Where ***n*** is the code length and ***M*** is the number of codewords.

Proof: there are 2^n possible binary vectors of length ***n***. each of the ***M*** codewords needs a sphere of distance ***t*** around itself which cannot contain another codeword in order to tolerate t errors and still be uniquely identifiable.

A code which fulfills the **Hamming bound** with equality is called a **perfect code.**

3.10.3 Code Generator Matrix

We begin by rearranging the columns of $\mathbf{H}_{[7,4]}$ into

$$\boldsymbol{H}'_{[7,4]}=\begin{bmatrix}1&1&1&0&1&0&0\\1&1&0&1&0&1&0\\1&0&1&1&0&0&1\end{bmatrix}$$

By replacing each column by some linear combination of columns of $H_{[7,4]}$. This does not change the code, since **Hx** = 0 is unaffected.
This new arrangement has the form

$$\boldsymbol{H}=\left[\boldsymbol{A}\middle|\boldsymbol{I}_{n-k}\right] \tag{3.28}$$

Rearranging as follows and calling $[x_1, \ldots, x_k]^T = [u_1, \ldots, u_k]^T$ the information bits, we obtain

$$\boldsymbol{A}\begin{bmatrix} x_1 \\ \vdots \\ x_k \end{bmatrix} + \begin{bmatrix} x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = \mathbf{0}$$

$$\begin{bmatrix} x_{k+1} \\ \vdots \\ x_n \end{bmatrix} = -\boldsymbol{A}\begin{bmatrix} u_1 \\ \vdots \\ u_k \end{bmatrix} \Rightarrow \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_k \\ -\boldsymbol{A} \end{bmatrix}\begin{bmatrix} u_1 \\ \vdots \\ u_k \end{bmatrix} \tag{3.29}$$

The Code Generator Matrix is now given by:

$$\boldsymbol{x}^T = \boldsymbol{u}^T\boldsymbol{G} \tag{3.30}$$

The matrix G = [$\boldsymbol{I_k}$, $-\boldsymbol{A}^T$] is the Generator Matrix for the [7, 4]-Hamming code

$$\boldsymbol{G}_{[7,4]} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{3.31}$$

CHAPTER 4

RESULTS

4.1. Error Detection and/or Correction

We have mentioned before briefly how error-control codes work, but now after explaining the types of codes and how they are generated, we can go deeper in telling the manners and different uses of error-control codes. We will take the linear block code as an example, and we will show how its encoder and decoder work.

The block encoder takes a block of k bits and replaces it with n-bit codeword. For a binary code, there are 2^k possible codewords in the *codebook*. The channel introduces errors and the received word can be any one of 2^n n-bit words of which only 2^k are valid codewords. The job of the decoder is to find the codeword that is closest to the received n-bit word.

The examples will use a brute force look-up method. The decoding spheres of Figure 4.1 will be used to illustrate the decoding progress. In Figure 4.1, each valid codeword is represented by a point surrounded by a sphere of radius t, where t is the number of errors that the code can correct. Note that codewords **A** and **B** of Figure.1 are separated by a distance d *min*, called the *minimum distance* of the code. Usually, codes with large minimum distance are preferred because they can detect and correct more codes. First lets consider a decoder that can only detect errors, not correct them.

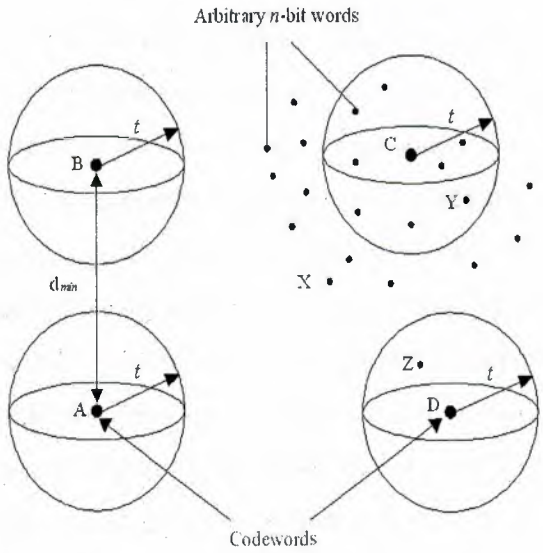

Figure 4.1 Decoding Spheres of Radius *t*

4.1.1 Error Detection Only

The minimum distance of a code gives a measure of its error detection capability. An error control code can be used to detect all patterns of u errors in any codeword as long as d $min = u + 1$. the code may also detect many error patterns with more than u errors, but it is guaranteed to detect all patterns of u errors or less. We'll assume that the error detection decoder comprises a look-up table with all 2^k valid codewords stored. When an n-bit word is received by the decoder, it checks the look-up table and if this word is one of the allowable codewords, it flags the n-bit word as error free and sends the corresponding information bits to the user. We'll use Figure 1 to illustrate three cases: no errors, a detectable error pattern, and an undetectable error pattern.

Case #1: *No errors*, Lets assume that the encoder sends codeword **C** and the channel introduces no errors. Then codeword **C** will also be received, the decoder will find it in the look-up table, and decoding will be successful.

Case #2: *Detectable error pattern.* This time we send codeword **C** and the channel introduces errors such that the *n*-bit word **Y** is received. Because **Y** is not a valid codeword, the decoder will not find it in the look-up table and will therefore flag the receiver *n*-bit word as an errored codeword. The decoder does not necessarily know the number or location of the errors, but that's acceptable because we only asked the decoder to detect errors. Since the decoder properly detected an errored codeword, decoding is successful.

Case #3: *Undetectable error pattern.* We send a codeword **C** for the third time and this time the introduces the unlikely (but certainly possible) error pattern that converts codeword **C** into codeword **D**. the decoder can't know that codeword **C** was sent and must assume that codeword **D** was sent instead. Because codeword **D** is a valid codeword, the decoder declares the received *n*-bit word error-free and passes the corresponding information bits to the user. This is an example of decoder failure.

Naturally, we want the decoder to fail rarely, so we choose codes that have a small probability of undetected error.

4.1.2 Forward Error Correction (*FEC*)

Comparing the spheres surrounding codewords **A** and **B** in Figure 4.1, we see that the error correcting capability of a code is given by d $min = 2t + 1$ (this is the minimum separation that prevents overlapping spheres). Or in other words, a code with d $min = 3$ can correct all patterns of 1 error, one with d $min = 5$ can correct all patterns of 2 errors, and so on. A code can correct t errors and detect v additional errors as long as (d $min \geq 2t + v + 1$). Now refer to Figure.1 and consider three error decoding cases for the error correction decoder: correct decoding, decoding failure, and error detection without correction.

Case #1: *Correct decoding.* Assume that codeword **C** is sent and the *n*-bit word **Y** is received. Because **Y** is inside **C**'s sphere, the decoder will correct all errors and error correction decoding will be successful.

Case #2: *Decoding failure*: this time we send the codeword **C** and the channel will give us *n*-bit word **Z**. the decoder has no way of knowing that the codeword **C** was sent and must decode to **D** since **Z** is in **D**'s sphere. This is an example of error correction decoder failure.

Case #3: *Error detection without correction.* This case shows one way that an error correction can be used to also to detect errors. We send the codeword **C** and receive *n*-bit word **X**. Since **X** is not inside any sphere, we won't try to correct it. We do, however, recognize that it is an errored codeword and report this information to the user.
We could try to correct *n*-bit word **X** to the nearest valid codeword, even though **X** was not inside any codeword's sphere. A decoder that attempts to correct all received *n*-bit words whether they are in a decoding sphere or not is called *complete decoder*. On the other hand, a decoder that attempts to correct only *n*-bit words that lie inside a decoding sphere is called *incomplete* or *bounded distance decoder*.

4.2. Designing the [7, 4] Code

A natural question to ask is, "For a linear code, how does one select codewords out of the space of 2^7 7-tuples?" There is no single solution, but there are constraints in how choices are made. Here are the elements that help point to a solution.

1- The number of codewords is $2^k = 2^4 = 16$.
2- The property of linearity must apply. This property dictates that the sum of any two codewords must yield a valid codeword.
3- The all-zeros vector must be one of the codewords. Since any codeword that is added (modulo-2) to itself yield an all-zeros vector.
4- Each codeword is 7 bits long.
5- Since *d*min = 3, the weight of each codeword (except for the all-zeros codeword) must also be at least 3. (this code has the ability of correcting 1 errors t=1).
6- Assume that the code is systematic, so the rightmost 4 bits of each codeword are the corresponding message bits.

	7	6	5	m	m	m	m
0	0	0	0	**0**	**0**	**0**	**0**
1	0	0	0	**0**	**1**	**1**	**1**
2	0	0	1	**1**	**0**	**0**	**1**
3	0	0	1	**1**	**1**	**1**	**0**
4	0	1	0	**1**	**0**	**1**	**0**
5	0	1	0	**1**	**1**	**0**	**1**
6	0	1	1	**0**	**0**	**1**	**1**
7	0	1	1	**0**	**1**	**0**	**0**
8	1	0	0	**1**	**0**	**1**	**1**
9	1	0	0	**1**	**1**	**0**	**0**
A	1	0	1	**0**	**0**	**1**	**0**
B	1	0	1	**0**	**1**	**0**	**1**
C	1	1	0	**0**	**0**	**0**	**1**
D	1	1	0	**0**	**1**	**1**	**0**
E	1	1	1	**1**	**0**	**0**	**0**
F	1	1	1	**1**	**1**	**1**	**1**

Figure 4.2 Possible codewords for [7, 4] code

4.3. Error Detection Versus Error Correction Tradeoffs

Using the codeword set in Figure.4.2, Error-detection and error-correction capabilities can be traded, provided that the following distance relationship prevails

$$d_{\min} \geq \alpha + \beta + 1 \tag{4.1}$$

where α represents the number of bit errors to be corrected, β represents the number of bit errors to be detected, and $\beta \geq \alpha$. The tradeoff choices available for the (7, 4) code example are as follows:

Detection β	Correction α
1	1
2	0

Figure.4.3 β and α tradeoffs

4.4. Performance of [7, 4] code

We found that $d_{min} = 3$ for the [7, 4] code, so if the errors that occurred in the received codeword was more than 2, the code will only detect a single error and correct one, or it will detect 2 errors and correct none. Figure 4.4 will show the performance of different types of linear block codes.

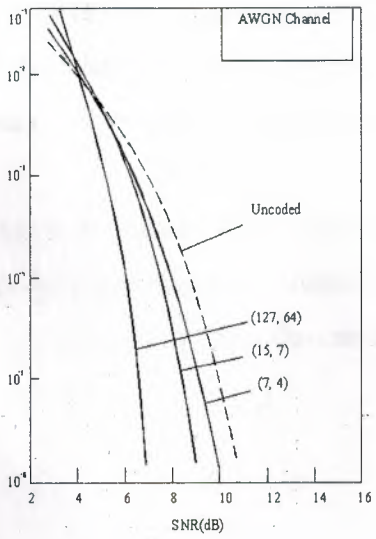

Figure 4.4 Performance of linear block codes

CONCLUSION

In the present scenario of telecommunication applications, wireless systems play a prominent role. Compared to the competing wired systems, they offer a number of practical advantages, like easy installation and maintenance, flexibility and reconfigure ability, mobility, low cost components, and others. On the other hand, the performance required to new wireless systems are very high, they must guarantee extremely low error rates, to face the poor error resilience of commercial source coding schemes, and large spectral efficiencies, to allow transmission rates as great as possible. The achievement of these objectives implies the adoption of very efficient error correcting schemes. Following the invention of the "turbo principle" in 1993, a number of turbo-like codes have been proposed, able to approach the theoretical Shannon limit. These codes include block and convolutional codes.

Low Density Parity Check (LDPC) codes also belong to this class: even if their origin is different, their decoding is based on "message passing" methods, whose rationale is identical to that of turbo codes. All these codes are potentially suitable for application in the wireless framework.

Block codes still having its position in error controlling. That is because of its simplicity of designing. Hamming codes which are another type of block codes are classified under the perfect codes group. Perfect codes can detect and correct all error bits that occurs in the codeword transmitted.

References

[1] C. E. Shannon, "A mathematical Theory Of Communication," *Bell System technical journal,*vol. 27, pp.379-423, 1948.

[2] F. Guterl, "Compact Disc," *IEEE Spectrum,* vol. 25, No. 11, pp. 102-108, 1988.

[3] Stephen G. Wilson, *Digital Modulation and Coding*, Prentice Hall, Inc. 1996.

[4] Simon Haykin, *Communication Systems*, John Wiley & Sons, Inc. 1994

[5] Robert E. Peile " The Application of Error Control to Communications", *IEEE Communication Magazine,* vol.25, No.4, April 1987.

[6] Tom Richardson. " Low Density Parity Check Codes ". *IEEE Communication Magazine.* August 2003