Syrian Private University
Faculty of Informatics & Computer Engineering
Communication and Networking Department

Multi-user Code Division Multiple Access Communication System

A Senior Project Presented to the Faculty of Computer and Informatics Engineering

In Partial Fulfillment of the Requirements for the Degree

Of Bachelor of Engineering in Communications and Networks

Under the supervision of

Dr. Eng.: Ali Awada

Project prepared by

Anas Al Abas

Hasan Hamad

Baraa Al Shammaa

Fares Al Zein

Academic Year: 2013/2014

ACKNOWLEDGEMENTS

بالأمس فتحت لنا الأبواب لننهل من علومها

واليوم وبعد مضي خمس سنوات

لايسعني هنا الا أن أتقدم بالشكر والعرفان بالجميل لكل من ساهم في تقديم

العلم والعطاء

وبذل جهوداً وجهوداً لنكون مهندسين اقوياء مستنيرين بعلمنا ومخلصين بعملنا

وأخص بالشكر :

الدكتور علي عواضه الذي اشرف على هذا البحث

وعميد كليتنا **الدكتور المهندس علي سكاف** الذي بوجوده تألقت الكلية

كما اشكر كافة الأساتذة والمهندسين والمشرفين في الجامعة

راجياً ربي ان يديم عليهم الخيروالتوفيق الدائم.

DEDICATION

من أوصاني ربي به احساناً.....
من كان لسانه لايفتر بالدعاء لي بالنجاح.....
من علمني معنى الخير والعطاء.....
وسقاني من ينابيع الحب والتقوى.....
.....والدي الحبيب

قلب رحيم أتعب نهاره وأضاء ليله من أجلنا.....
ينبوع الحب والحنان الذي لاينضب.....
من غسلت بدموعها أحزاننا.....
يامن حبك أحب إلي من نفسي.....
إليك..... إليك.....
.....والدتي الحنون

هم من شاركوني أفراحي وأحزاني.....
وهم من مسك بيدي لأسلك طريق المعرفة.....
رسمو لي دروباً بخطوط الأمل والمحبة.....
من فاح عطر ياسمينهم في حياتي.....
.....إخوتي وأخواتي

هم قلوب عامرة بالحب والصدق.....
رفاق درب أحببتهم وأحبوني.....
مشاعل مضيئة كانت على الطريق الذي أخى الله بيني وبينهم بلا نسب.....
.....الـى أصدقـائـــي

Abstract

This project titled "Multi-user **C**ode **D**ivision **M**ultiple **A**ccess(**CDMA**) Communication System " revolves around the study of various components of a communication system which is used in CDMA and their simulation of implementation in the real time system.

The objective of the project is to discusses different multiple access techniques in comparison with each other. Also explained Spread Spectrum technique (Direct Sequence and Frequency Hopping) and the general CDMA communication system and a description of its various components. And simulate the system in AWGN channel with BER between transmitter and receiver.

The bit error rate (BER) performance of the Code Division Multiple Access (CDMA) cellular system based on IS-95 standard in the presence of an additive white Gaussian noise (AWGN) and interference has been investigated in this paper. The performance is evaluated under noise and interference for the CDMA forward link. Rake receiver is used in CDMA-based (Code Division Multiple Access) systems and can combine multipath components, which are time-delayed versions of the original signal transmission. Combining is done in order to improve the signal to noise ratio at the receiver and so reduced the BER in practical simulation of system.

Introduction

As the world progresses forward in technology, with the data rate requirements increase in the different digital communication systems, there is a decrease in the available bandwidth. To meet for this problem, different multiple access techniques are used. By using these Code division multiple access technique the bandwidth utilization can be maximized. In these multiple access techniques multiple users can use the same bandwidth. Multiple access techniques have a much greater importance in radio communications in order to make use of the available spectrum.

CDMA is one method for implementing a multiple access communication system. MULTIPLE ACCESS is a technique where many subscribers or local stations can share a communication channel at the same time or nearly so despite the fact originate from widely different locations. A channel can be thought of as merely a portion of the limited radio resource, which is temporarily allocated for a specific purpose, such as someone's phone call. A multiple access method is a definition of how the radio spectrum is divided into channels and how the channels are allocated to the many users of the system.

The objective of the project is to implement a digital communication system using CDMA depending on direct sequence Spread Spectrum technique, source coding, and channel coding. The CDMA system simulated in MATLAB to calculate BER for N bit stream in AWGN channel.

The report is organized as follows:

- **Chapter 1**: Discusses different multiple access techniques in comparison with each other. Direct Sequence and Frequency Hopping CDMA is also explained in this chapter as this would be implemented during the project.

- **Chapter 2**: Focuses on the general CDMA communication system and a description of its various components.
- **Chapter 3**: Includes the simulations and results of the project.
- **Chapter 4**: Discusses some of the observations during the project and some future enhancements are also mentioned here.

List of Contents

List of Figures

List of Abbreviations

Abbreviations	Full name
CDMA	Code Division Multiple Access
BER	Bit Error Rate
TDMA	Time Division Multiple Access
FDMA	Frequency Division Multiple Access
SDMA	Space Division Multiple Access
SS	Spread Spectrum
FHSS	Frequency Hopping Spread Spectrum
DSSS	Direct Sequence Spread Spectrum
AWGN	Additive White Gaussian Noise
PN	Pseudo Noise
MAI	Multiple Access Interference
BPSK	Binary Phase Shift Keying
QPSK	Quadrature Phase Shift Keying
QAM	Quadrature Amplitude Modulation
PSD	Power spectral density
B	Bandwidth
PCM	Pulse code modulation
NRZ	Non Return to Zero

Chapter 1

Study of the Multiple Access Techniques

1-1 Background

Mobile communications are rapidly becoming more and more necessary for everyday activities. With so many more users to accommodate, more efficient use of bandwidth is a priority among cellular phone system operators. Equally important is the security and reliability of these calls. One solution that has been offered is a code division multiple access system.

1-2 Frequency Division Multiple Access:

One of the earliest multiple access techniques used for cellular transmission is FDMA. This transmission is useful when a continuous transmission is required. The total bandwidth assigned is divided among a number of users hence enabling them to use a specific portion of frequencies out of the whole bandwidth simultaneously. These channels are allotted to a user on request from the user. Therefore if there is no request for a certain channel, that portion of bandwidth is wasted. These channels have narrow bandwidth and are consequently implemented in narrow band systems. FDMA does not entail synchronization or timing control as a particular portion of the overall bandwidth is assigned to a specific user for all the time, this helps in decreasing the number of overhead bits. To prevent the interference between adjacent channels, they are separated by guard bands which results in a waste of bandwidth. In order to achieve simultaneous transmission and reception FDMA makes use of duplexer,(see Figure 1.1) as the base station has got a separate transmitter and receiver, duplexer enables the mobile to switch between the transmitter and receiver. [1]

FDMA systems commonly use one antenna for a number of channels. In order to achieve maximum power efficiency the power amplifier has to operate near the saturation point. At this operating point. This in turn results in the adjacent channel interference. [2], [3]

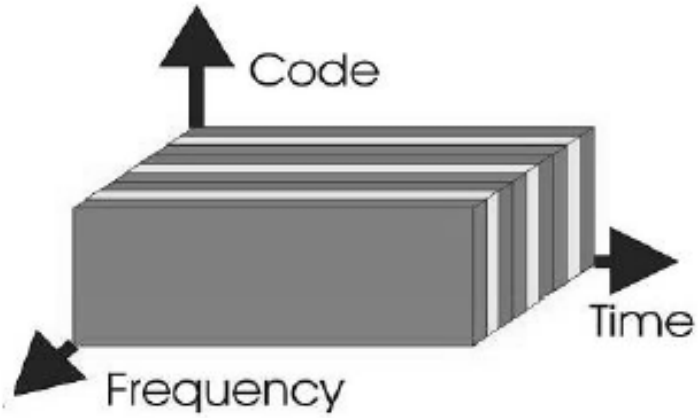

Figure 1.1: Channel handling by FDMA

1-3 Time Division Multiple Access:

TDMA enables a user to occupy the entire bandwidth, but for a finite period of time. Each and every user is allotted time slots during which he/she can use the entire bandwidth. Time synchronization is very important in this technique as the receiver should be able to know that a user has ended sending a data and now it is another user who is transmitting. If time synchronization has been achieved properly, there would be no chance of inter channel interference, still as a precautionary measure, two channels are separated by a small guard time. There are chances that a call is lost while using TDMA as when a user moves from one cell to another cell there is a probability that no free time slot is available. Two different time slots are used for the transmission and reception in TDMA. This is known as time division duplexing and does not require any duplexers. Moreover as in TDMA the transmission is not continuous, digital modulation techniques are to be used. Transmission from various users is fed into a frame. The preamble bits in the beginning of the frame contain synchronization and address information of the base station and subscribers. [1],[2]

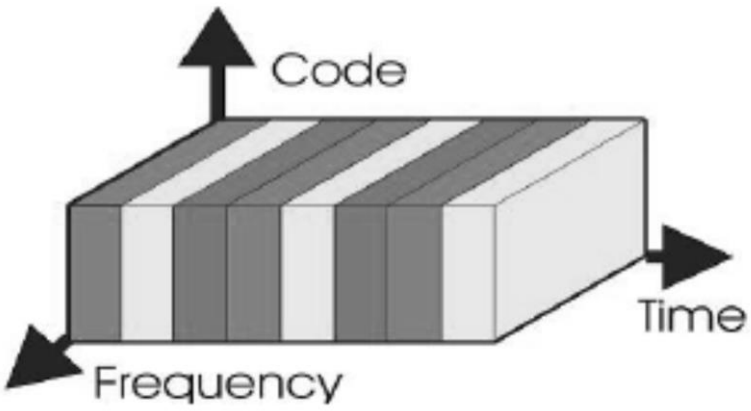

Figure 1.2: Channel handling by TDMA

1-4 Space Division Multiple Access:

In SDMA the users are separated spatially. In other words, same frequency can be used in different cells in a cellular network system. These cells should be at a safe distance from each other for the sake of preventing inter-channel interference. Therefore the number of cells in which a region can be divided is limited and consequently limits the frequency re-use factor. Smart antennas can be used to enhance the performance of these systems by steering the antenna beam in the direction of the user and placing nulls in the direction of interfering signals. [1]

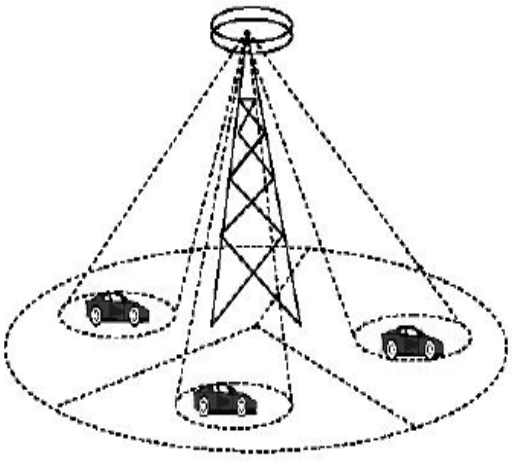

Figure 1.3: Intra-cell SDMA

1-5 Code Division Multiple Access:

Code Division multiple Access is a spread spectrum technique. It increases the throughput (bandwidth efficiency) of a digital communication system. All the spread spectrum techniques including CDMA have the following main components.

- The signal occupies a bandwidth which is greater than the minimum bandwidth required to send the information. [6]
- Spreading is accomplished by a spreading signal.[6]
- De-spreading is achieved at the receiver by the correlation of the received signal with a replica of the spreading signal used at the transmitter. [6]

1-5-1 General Principles of CDMA

Code division multiple access (CDMA) is a multiple access technique where different users share the same physical medium, that is, the same frequency band, at the same time. The main ingredient of CDMA is the spread spectrum technique, which uses high rate signature pulses to enhance the signal bandwidth far beyond what is necessary for a given data rate.

In a CDMA system, the different users can be identified and, hopefully, separated at the receiver by means of their characteristic individual signature pulses (sometimes called the signature waveforms), that is, by their individual codes.

Nowadays, the most prominent applications of CDMA are mobile communication systems like (IS-95) or UMTS, which are explained later.

To apply CDMA in a mobile radio environment, specific additional methods are required to be implemented in all these systems. Methods such as power control and soft handover have to be applied to control the interference by other users and to be able to separate the users by their respective codes.

1-5-2 The concept of spreading

Spread spectrum means enhancing the signal bandwidth far beyond what is necessary for a given data rate and thereby reducing the power spectral density (PSD) of the useful signal so that it may even sink below the noise level. One can imagine that

this is a desirable property for military communications because it helps to hide the signal and it makes the signal more robust against intended interference (jamming). Spreading is achieved by a multiplication of the data symbols by a spreading sequence of pseudorandom signs. These sequences are called pseudonoise (PN) sequences or code signals.

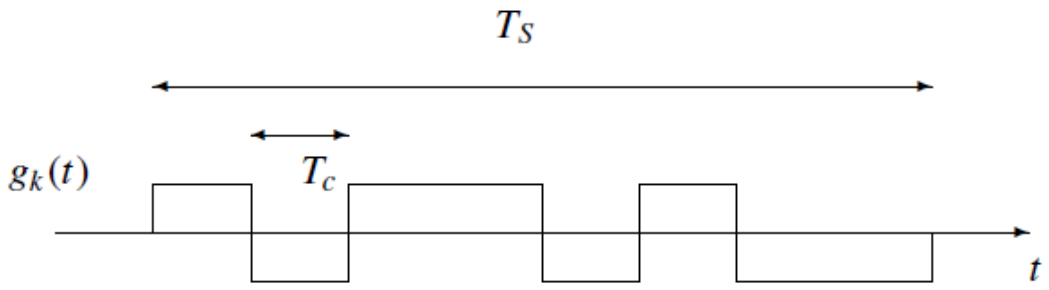

Figure 1.4 Signature pulse with N = 8 rectangular chips.

Consider a rectangular transmit pulse of length TS:

$$g(t) = \frac{1}{\sqrt{T_S}} \prod \left(\frac{t}{T_S} - \frac{1}{2} \right) \qquad (1-1)$$

The divided pulse into N sub rectangles, referred to as chips, of length $T_c = T_S/N$ and change the sign of the sub rectangles according to the sign of the pseudorandom spreading sequence. Figure 1.4 shows the resulting transmit pulse $gk\ (t)$ of user number k for N = 8. Here, the spreading sequence for user k is given by $(+,-,+,+,-,+,-,-)$. When it is convenient (e.g. for the performance analysis) the sign factors shall be appropriately normalized. We note that in practice smooth pulse shapes (e.g. raised cosine pulses) will be used rather than rectangular ones.

The increase of the signaling clock by a factor N from $T_s^{-1}$to T_c^{-1} leads to an increase of bandwidth by a factor of T_c/T_S (see Figure 1.5). For this reason, $N = T_S/T_c$ is called the spreading factor or, more precisely, the spreading factor of the signature pulse. This spreading is due to multiplication by the code sequence. While within the specification documents for CDMA mobile communication systems the spreading factor is often denoted by SF, formulas are kept simpler by using the symbol N. Hence, we use both notations.

Later we may have different spreading mechanisms that work together, especially in the context of channel coding. Therefore, we reserve the notion of the effective spreading factor. It is often not uniquely defined where channel coding ends and where modulation starts and thus it may be ambiguous to speak of a bit rate after channel coding. We regard it as convenient to define the effective spreading factor by

$$SF_{eff} = \frac{R_{chip}}{R_b} \qquad (1-2)$$

Where $R_b$is the useful bit rate and $R_{chip} = 1/T_c$ the chip rate. Obviously, this spreading factor is approximately the inverse of the spectral efficiency for a single user.

The objective of spreading is a waste of bandwidth for the single user to achieve more robustness against multiple access interference (MAI). It would thus be a contradiction to this objective to use bandwidth-efficient higher-level modulation schemes.

Any modulation scheme that is more efficient than BPSK would reduce the spreading factor. Therefore, BPSK and QPSK are used as the basic modulation schemes in most practical communication systems. Nevertheless, higher-order modulation techniques like 8-PSK and 16-QAM also are applied as additional transmission options to offer a high-speed packet transfer at good propagation conditions. Furthermore, we point out the special role of channel coding. Channel coding usually means that a higher power efficiency has to be paid by a lower spectral efficiency. Thus, channel coding can be interpreted as an additional spreading mechanism. In the extreme case, all the spreading can be done by channel coding, and the PN sequences serve only for user separation. Keeping this in mind, we can interpret the conventional spreading by a PN-sequence as a repetition code combined with the repeated transmit symbols multiplied by a pseudorandom sign. The symbol will be repeated N times at a clock rate increased by the factor $N = T_S/T_c$ and scrambled by a random sign. Equivalently, this is time delay diversity. Because of the time dispersion of the channel, we may get a multipath diversity gain. The appropriate diversity combiner is the RAKE receiver.

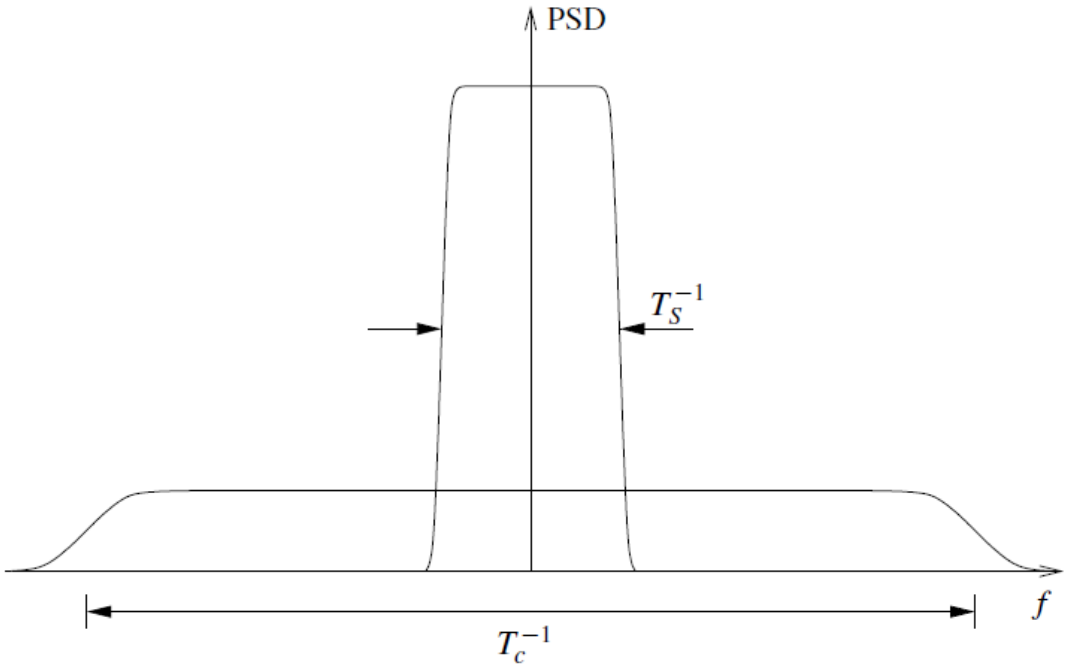

Figure 1.5 Power spectral density (PSD) for DS-CDMA.

1-5-3 RAKE receiver

The name RAKE receiver originates from the fact that there are some similarities to a garden rake. As illustrated in Figure 5.3, the receiver consists of a certain number of correlator (called RAKE fingers) correlating the received signal to the used code signal.

One of the correlator (the so-called search finger) has the task to determine the propagation delay values τi $(i = 1, 2, \ldots)$ of the most relevant propagation paths. These values are used within the other correlator (fingers) to adjust the exact timing for the respective multipath components. By this method, the multipath components can be detected separately (if the codes have a good autocorrelation property); subsequently, they can be combined by a maximum ratio combiner. It should be noted that multipath components can only be resolved if their delay difference is higher than about a quarter of the chip duration T_c. Furthermore, the number of RAKE fingers is usually restricted to 4–6 (including the search finger).

It must be emphasized that spreading by itself does not provide any performance gain in the AWGN channel1. As shown in Figure 1.6, for a single user, spreading means nothing but choosing a spectrally rather inefficient waveform that smears the spectral

power over an SF times higher bandwidth. Thus, for the same signal power, the SNR decreases by a factor of SF. The necessary power per bit rate, which equals the energy per bit, E_b, does not depend on the pulse shape. We therefore avoid the popular but misleading word processing gain for the factor SF. Originally, for a single user, it is nothing but a waste of bandwidth.

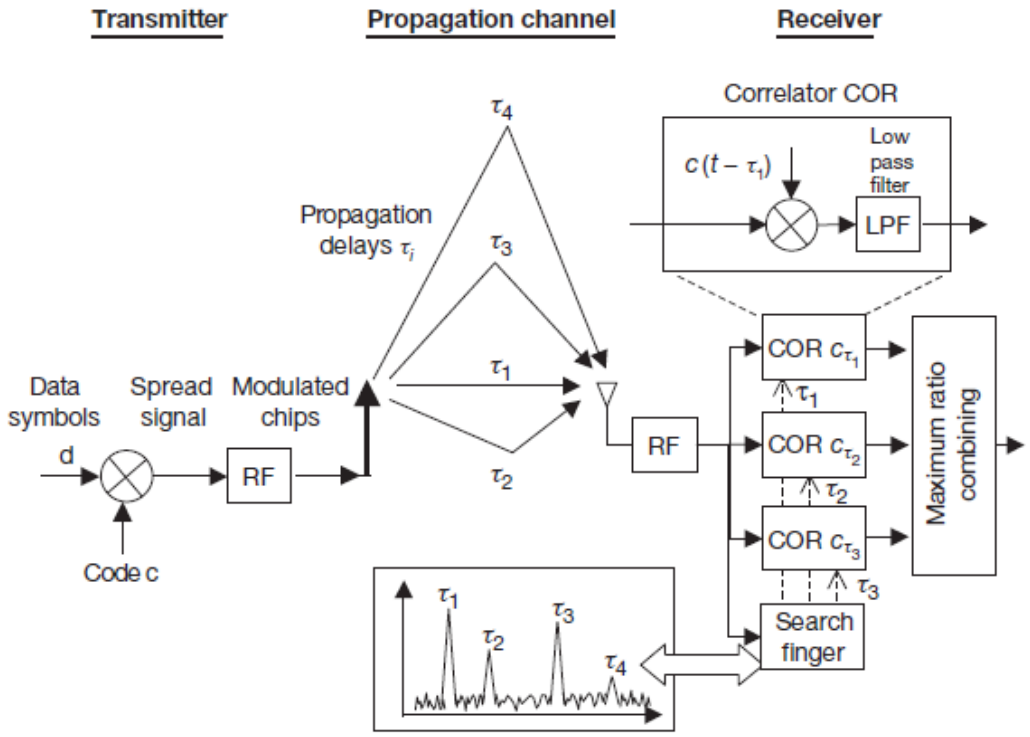

Figure 1.6 Illustration of the RAKE receiver.

The reason for using spreading is not this virtual processing gain. A real gain of spreading concerning the range of data transmission can be achieved in a frequency-selective fading environment. The increased bandwidth of a spread signal provides us with an increased frequency diversity as compared to a narrowband FDMA system. Such a frequency diversity can only be exploited if the signaling bandwidth significantly exceeds the correlation frequency (i.e. the coherency bandwidth) of the channel. In that case, we speak of a wideband CDMA system.

However, such diversity can also be achieved by increasing the carrier bandwidth by multiplexing different users to a frequency carrier using a time division multiplex scheme. Comparing the spread spectrum and the TDMA technique at the same signal bandwidth, at the same data rate and mean transmission power (energy per transmitted bit), roughly the same performance will result since the receive $E_b/N_o$is the same.

Nevertheless, having in mind the discussion on electromagnetic compatibility of mobile phones, spreading may have an advantage since it uses a continuous transmission while transmission in time multiplex systems is pulsed. For this reason, sometimes the peak transmission power of systems is limited by regulatory bodies. Obviously, at equal peak transmit power the performance of spread spectrum systems is higher than that of TDMA systems.

1-5-4 Direct sequence Spread Spectrums (DSSS)

Direct sequence spread spectrum systems are used in CDMA.

First of all, a PN-code is generated; PN-codes are different for different channels. The message signal modulates the PN-codes, as a result of which the message signal is spread over a greater band of frequencies(see Figure 1.7). Up-sampling is done for the signals to have a greater energy so that they can be received at the receiver correctly. Using an M-ary PSK modulation technique the phase of the signal is shifted pseudo-randomly. A short code system uses PN-sequences of the length equal to the data symbol length, where as a long code system uses PN-sequences of more than the data symbol length. [3]

When the signal reaches the receiver it is again multiplied by the same PN-code. The signal is retrieved and other processing is done on it before completely receiving it. [3]

The pulse width of a signal limits the bandwidth of a signal. Therefore a message signal with a pulse width 'T' would have a bandwidth $B = 1/T$. The Walsh code with a pulse width (chip rate) 'T_c 'would have a bandwidth of $B_w = 1/T_c$. The factor by which the increase bandwidth of the signal due to spreading can be represented is $N = T/T_S$. [4]

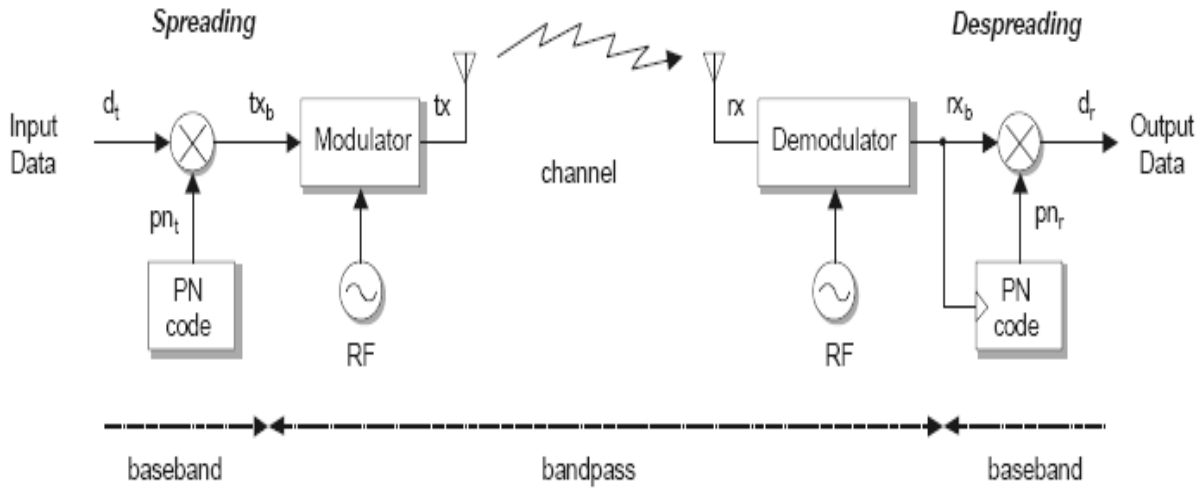

Figure 1.7: Building blocks of DS-SS

DS sequence allows each station to transmit over the entire frequency. Spectrum all the time. Multiple simultaneous transmissions are separated using some sort of coding technique that is each user is assigned a chip sequence. The sender and receiver synchronize by the receiver locking into the chip sequence and the sender and receiver locking into the chip sequence of the sender. All the other (unsynchronized) transmission is then seen as random noise. So with CDMA each user uses the full frequency spectrum.

They employ a high speed code sequence along with the basic information being sent, to modulate their RF carriers. The high speed code sequence is used directly setting the transmitted RF bandwidth.

Binary phase shift keying (BPSK) is the most common technique used in DS system. Direct sequence is, in essence, multiplication of a more conventional communication waveform by PN sequence in the transmitter.

1-5-6 Frequency Hopping Spread Spectrums (FHSS)

FH – CDMA is a kind of spread spectrum technology that enables many users to share the same channel by employing a unique hopping pattern to distinguish different users' transmission. The type of spread spectrum in which the carrier hops randomly from one frequency to another is called FH spread spectrum (see Figure 1.8). A

common modulation format for FH system is that of M-ary frequency shift keying (MFSK).the combination is referred to as FH/MFSK.

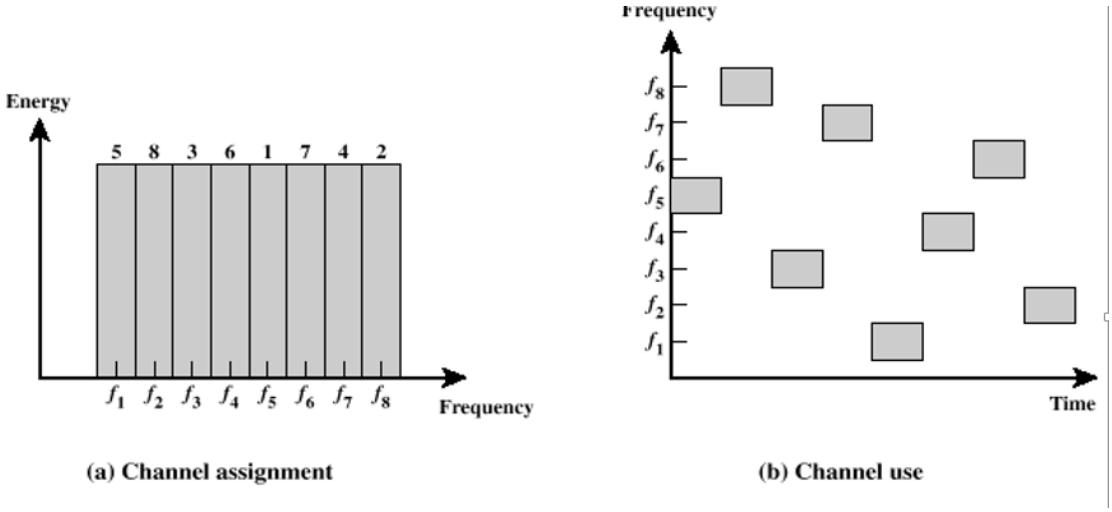

Figure 1.8: FHSS channels

A major advantage of frequency hopping is that it can be implemented over a much larger frequency band than it is possible to implement DS- spreading, and the band can be noncontiguous. Another major advantage is that frequency hopping provides resistance to multiple – access interference while not requiring power control to prevent near – far problems. In DS – systems , accurate power control is crucial but becomes less effective as the carrier frequency is increased.

Frequency hopping does not cover the entire spread spectrum, instantaneously, we are led to consider the rate at which the hops occur. So, we may identify two basic characterizations of frequency hopping.

1. *Slow frequency hopping*: in which the symbol rate Rs of MFSK signal is an integrator multiple of the hop rate Rh. That is, several symbols are transmitted on each frequency hop (see Figure 1.9).

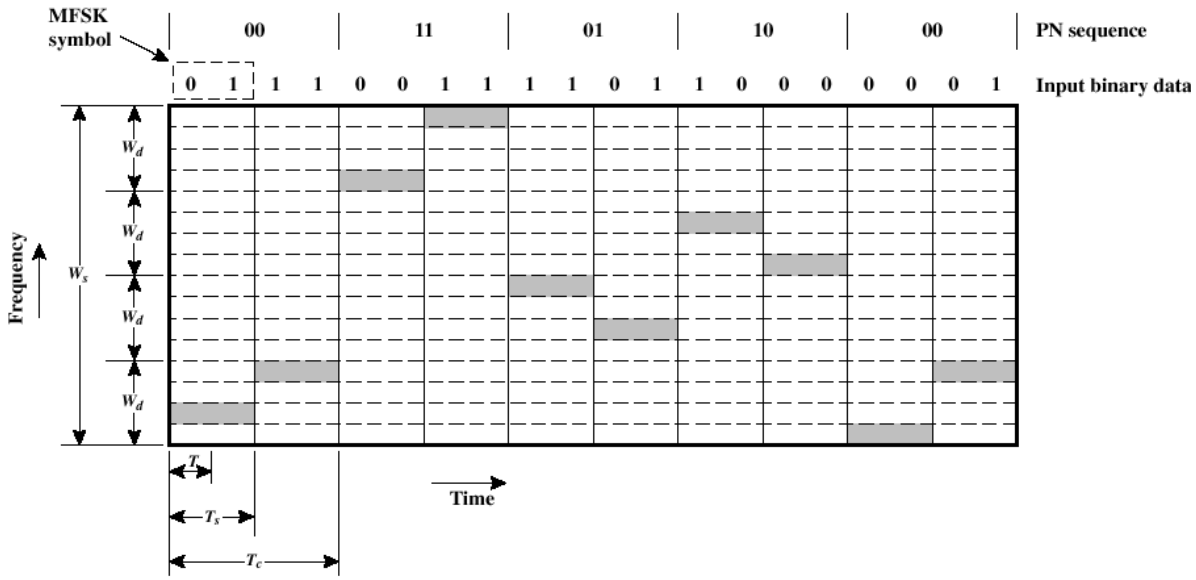

Figure 1.9: Slow FHSS

2. *Fast frequency hopping:* in which the hop rate Rh is an integrator multiple of the MFSK symbol rate Rs. that is, the carrier frequency will change or hop several times during the transmission of one symbol. (see Figure 1.10).

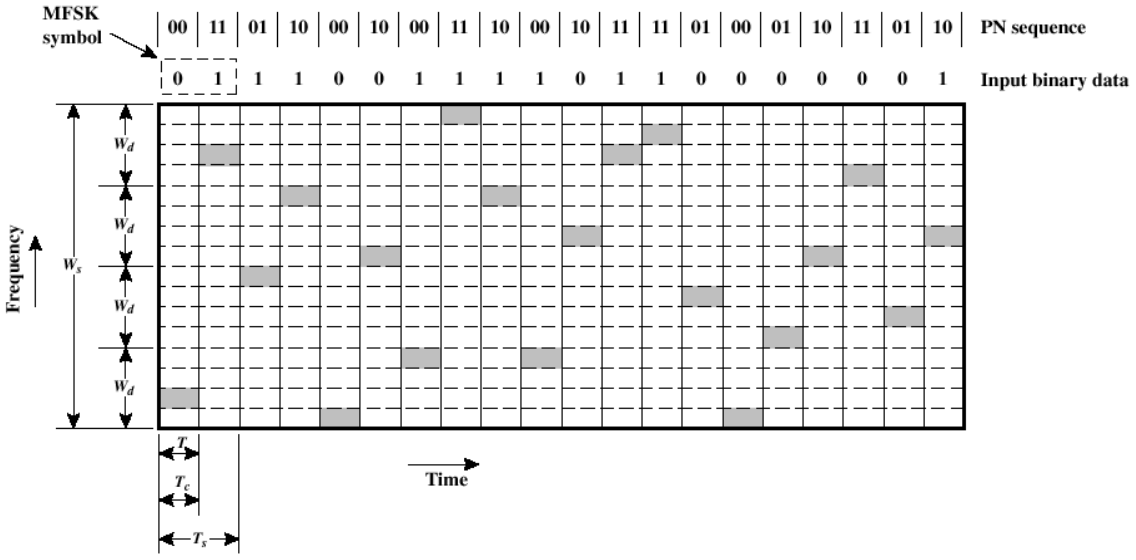

Figure 1.10: Fast FHSS

1-6 Conclusion

- In FDMA systems, users are allotted limited amount of bandwidth, whereas in TDMA & CDMA all the channel bandwidth is available to every user.
- Synchronization is an issue for the TDMA system, while FDMA and CDMA have no such problems.
- CDMA systems have the maximum capacity, whereas the FDMA systems have the minimum capacity.

Chapter 2

Communication System Fundamentals

2-1 System architecture

This section focuses on a general communication system that can be employed for CDMA, and different techniques which can be used in different components of the system. A general diagram for a CDMA based communication system is shown below.

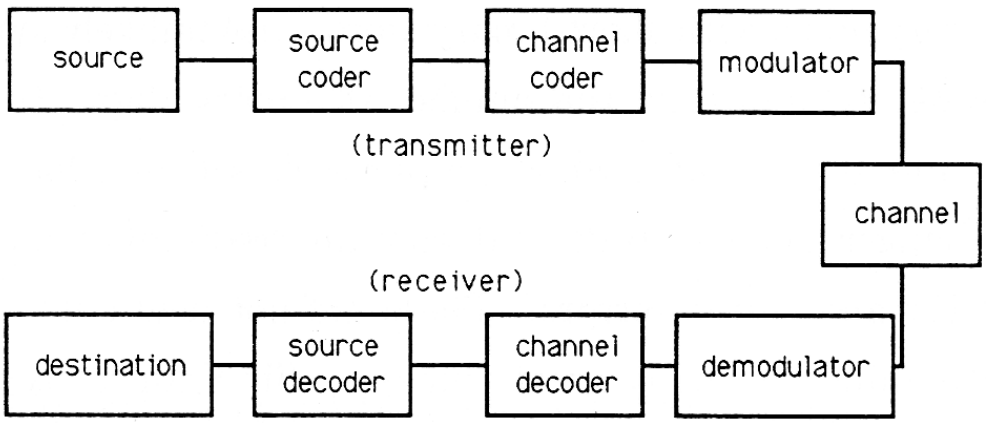

Figure 2.1: General CDMA Communication System Block Diagram

2-2 Source Coding

The first and foremost part of a digital communication system is source encoding or formatting. It ensures that the incoming signal is compatible with the digital processing. Whenever data compression is applied to formatting it is known as source coding.

The three basic steps involved in formatting are sampling, quantization and pulse code modulation (**PCM**).

In order to make the message signal compatible with the upcoming components of the communication system the message signal needs to be formatted.

The technique used in this project is **NRZ** (Non Return to Zero). This coding technique is used due to its simplicity and secondly it provides a phase difference of 1800 between two transmitted bits which was 900 when the message was transmitted in the form of 1s and 0s.

2-3 Channel Coding

Channel coding refers to a signal transformation whereby enabling the transmitted signals to better withstand the effect of channel non idealities, the communications performance can be increased. [5]

Channel coding follows source encoding in a general communication system. It caters for the channel's non-ideal behavior to the transmitted signal, i.e. to achieve a minimum bit energy to noise spectral density ratio given a specific probability of bit error, so that the communication system can operate at a low energy with the same probability of bit error or to decrease the probability of bit error for a specific energy to noise spectral density ratio. [5]

Block codes, code the incoming message sequence one block at a time. Convolution codes are different from them as they have a memory property, i.e. the n-tuple output of a convolution encoder is the function of the input and the previous K-1 states of the block, where K is the constraint length. [5].

2-3-1 Linear Block Codes

Linear block codes can be used to enable the error correcting and error detecting capability of a communication system. The general expression for a linear block code is (n,k). This notation represents that the linear block code converts the incoming block of 'k' information bits into a coded block of 'n' bits. These code bits have no memory, i.e. they only depend on the current information bits.

The functionality of these codes is also illustrated in the following example. Where the information bits are (i1, i2, i3, i4). The redundant bits r1, r2 and r3 that are to be added in the message are

$$r1 = i1 + i2 + i3 \qquad 2-1$$

$$r2 = i2 + i3 + i4 \qquad 2-2$$

$$r3 = i1 + i2 + i4 \qquad 2-3$$

These redundant bits are simply appended with the information bits and the coded bits are formed. The error performance of the communication system increases proportionally with respect to the increase in redundancy.

This code is also known as the hamming code and the Hamming Distance is defined as the number of disagreements between two codes. Minimum Distance is defined as the hamming distance of a pair of code words with minimum hamming distance. [5]

2-3-2 Convolution Block Codes

Three major integers used in these codes are n, k, K. n is the number of encoded bits, k is the number of input information bits and K is the constraint length that describes the number of stages on which the output would be dependant. k/n defines the information per coded bit. The codeword that is produced as a result of this encoding maybe denoted as $\mathrm{U} = (\mathrm{U1}, \mathrm{U2} \ldots \mathrm{Un})$

At the receiver end, the decoder receives a sequence $\mathrm{Z} = (\mathrm{Z1}, \mathrm{Z2} \ldots \mathrm{Zn})$. This happens due to the non-ideal behavior of the transmission channel, i.e. the noise present in the channel corrupts the incoming signal. The task of the detector would be to rectify the data by extracting the sequence U from the corrupted sequence Z. This would be possible if the receiver has a priori knowledge of how the information was encoded at the transmitter's end. [6]

2-3-2-1 Connection Representation

The connection representation of convolution encoders is shown in the Figure 2.2 below. This can be represented as

$$g1 = 111$$

$$g2 = 101$$

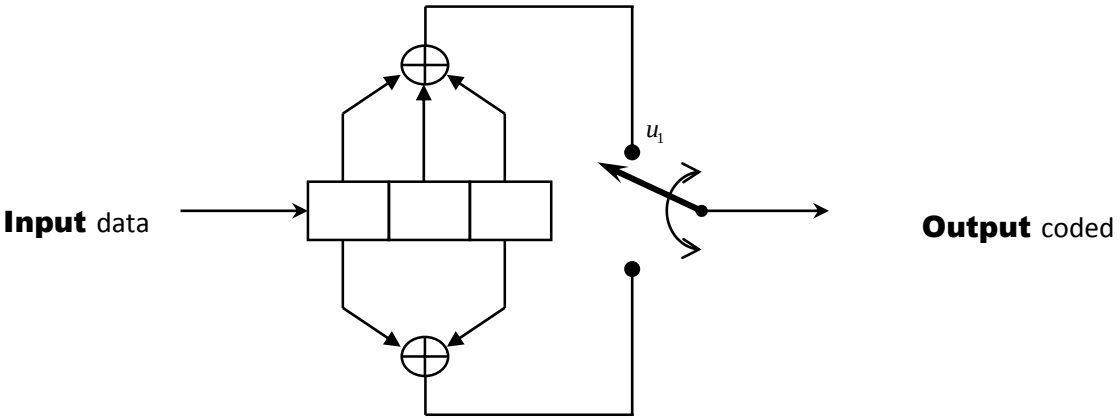

Figure 2.2: Connection Representation of Convolution Coding

2-3-2-2 Polynomial Representation

This representation is done with a point of view that that convolution coder can be treated as a set of cyclic code shift registers. Convolution encoder connections can also be represented by 'n' generator polynomials. The degree of the polynomials should be equal to or less than 'K'. The polynomials are used for the representation of modulo-2 adders in the encoder. For the figure given above the polynomials are as follows.

$$g_1(X) = 1 + X + X^2 \qquad (2-4)$$

$$g_2(X) = 1 + X^2 \qquad (2-5)$$

$U(X) = m(X)g_1(X)$ along with $m(X)g_2(X)$

Consider that m=101, therefore $m(X) = 1 + X^2$

$$\mathrm{m(X)}\ g_1(\mathrm{X}) = (1 + \mathrm{X}^2)(1 + \mathrm{X} + \mathrm{X}^2) = 1 + \mathrm{X} + \mathrm{X}^3 + \mathrm{X}^4 \qquad (2-6)$$

$$\mathrm{m(X)}\ g_2(\mathrm{X}) = (1 + \mathrm{X}^2)(1 + \mathrm{X}^2) = 1 + \mathrm{X}^4 \qquad (2-7)$$

$$\mathrm{m(X)}\ g_1(\mathrm{X}) = 1 + \mathrm{X} + 0.\mathrm{X}^2 + \mathrm{X}^3 + \mathrm{X}^4 \qquad (2-8)$$

$$g_1(\mathrm{X}) = 1 + \mathrm{X} + \mathrm{X}^2 \qquad (2-9)$$

$$g_2(\mathrm{X}) = 1 + \mathrm{X}^2 \qquad (2-10)$$

$$\mathrm{m(X)}\ g_1(\mathrm{X}) = 1 + 0.\mathrm{X} + 0.\mathrm{X}^2 + 0.\mathrm{X}^3 + \mathrm{X}^4$$

$$U(X) = (1,1) + (1,0)X + (0,0)X^2 + (1,0)X^3 + (1,1)X^4$$

$$U = 11 \quad 10 \quad 00 \quad 10 \quad 11$$

2-3-2-3 Functionality

In order to understand the functionality of the convolution encoder, suppose a message m= 1 0 1 has to be encoded. (see Figure 2.3).

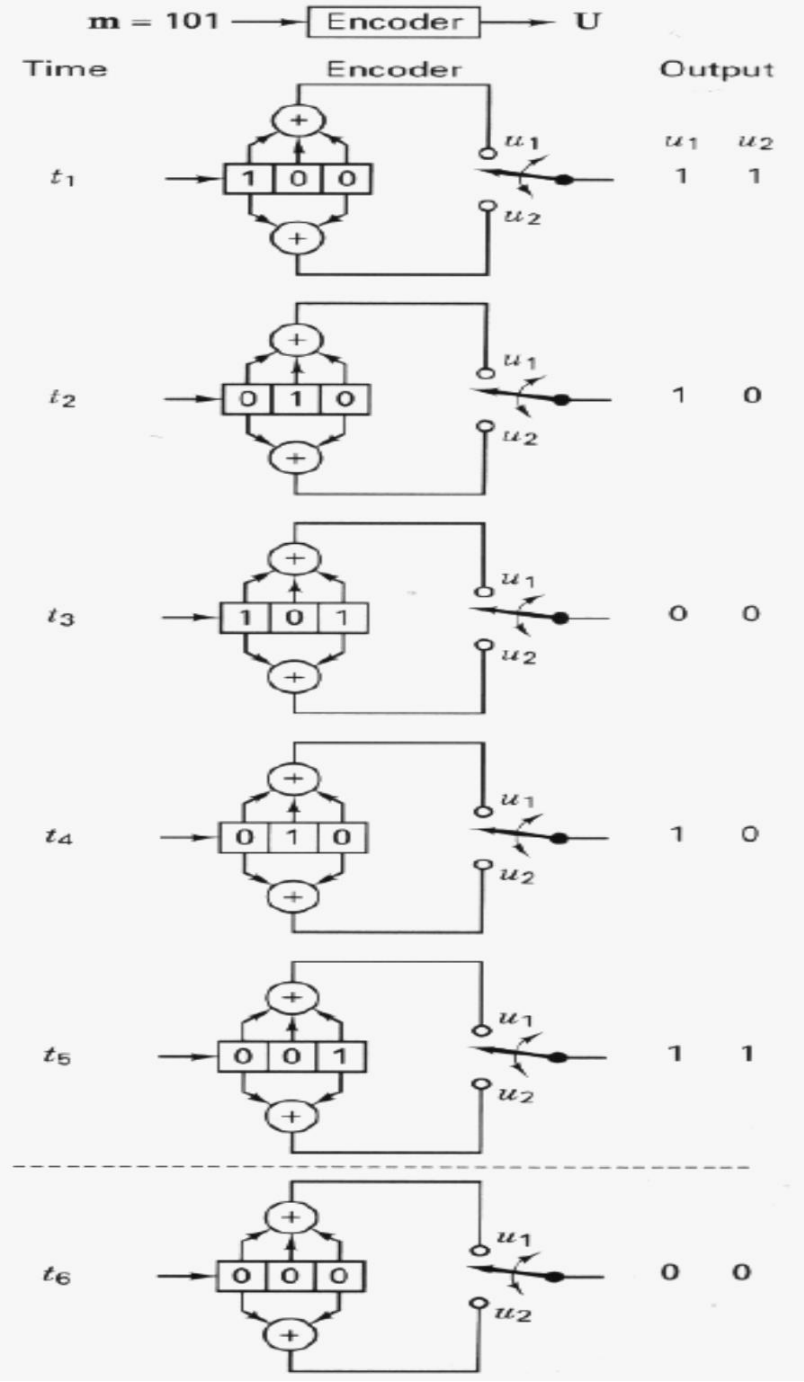

Figure 2.3: Convolution Encoding

U= 11 10 00 10 11

The important thing to note here is that at the time instant t6 the flushing bits were added in order to empty the register for the upcoming messages. Therefore the output of this was not added in the encoder. [6]

2-3-2-4 Viterbi Convolutional Decoding

The complexity of a viterbi decoder is not a function of the number of symbols in the codeword sequence. The essence of this algorithm is that it calculates a measure of similarity or difference between a received signal at a certain time instant and all the trellis paths entering each state at that instant. When two paths enter the same state, the one with the best metric is chosen, i.e. the one with the minimum path length.[6] This selection of surviving paths is performed for all the states. The decoder continues in this way to advance deeper into the trellis, making decisions by eliminating the least likely path.

2-4 Multiple Access Modulator

Multiple Access is a method by which the total throughput of a communications resource can be increased. These techniques are FDMA, TDMA, CDMA, SDMA. These have been discussed in detail in the previous chapter.

There are a number of spreading codes available distinguishable upon their cross and auto correlation properties. Some of these codes are elaborated below.

These spreading codes must have the following properties, if they are to be used in a direct sequence spread spectrum.

1. The cross-correlation should be near to zero.
2. Each sequence in the set has an equal number of 1s and –1s, or the number of 1s differs from the number of –1s by at most 1.
3. The scaled dot product of each code should be equal to 1.

2-4-1 Maximum length sequences

These sequences can be generated by using shift registers with feedback applied on them. These sequences meet all the conditions for spreading sequences very strictly. The cross correlation between this sequence and noise is very low, which helps in detecting signal out of noise in the receiver. These sequences are very useful for encryption as they also have a very low cross correlation with each other.

The randomness properties of maximal length sequences can be seen here.

2-4-2 Gold Sequence

In order to create two gold sequences as Figure 2.4, two maximum length sequences are to be combined. They have a very low auto-correlation which enables CDMA systems to transmit asynchronously. Gold sequences are constructed from pairs of preferred m-sequences by modulo-2 addition of two maximal Sequences of the same length [6]

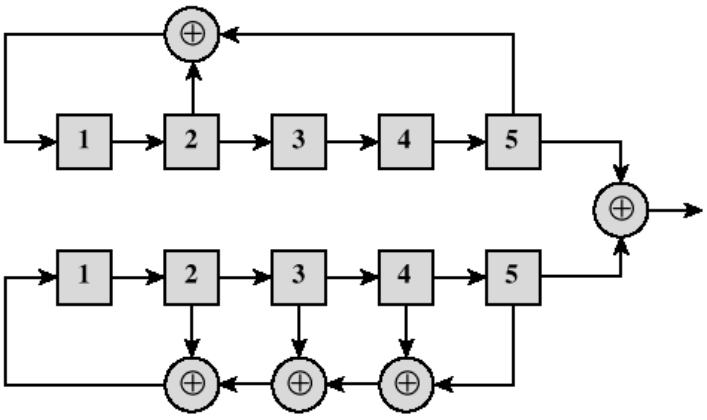

Figure 2.4: Gold sequence

2-4-3 Walsh Codes

Walsh codes have to be created from hadamard matrices. All generated Walsh codes would be orthogonal to each other. The basic hadamard matrix is shown below. These sequences provide low cross-correlation between each other. Secondly, the number of 1s is equal to number of 1s in every codeword.

$$H_{2N} = \begin{matrix} H_N & H_N \\ H_N & \overline{H_N} \end{matrix} \qquad (2-11)$$

By looking at the matrix above, Walsh codes with different lengths can be generated with the help of recursion. For a clear understanding Walsh codes with length equal to 4 are illustrated below.

$$H_4 = \begin{matrix} H_2 & H_2 \\ H_2 & \overline{H_2} \end{matrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \qquad (2-12)$$

2-5 Band-pass Signaling

In band-pass modulation, the shaped pulses from the baseband modulation are modulated with a carrier wave. Band-pass modulation can be used to separate different signals if a channel is utilized by more than one signals. [6]

Modulation techniques are used in order to make the signal compatible with the channel characteristics. For the transmission of signal in the form of electromagnetic waves, the signal must be set to a higher carrier frequency in order to limit the antenna size of the relation as both the frequency of the signal and the antenna size are inversely proportional to each other.

2-5-1 Binary Phase Shift Keying

A transmitted sequence uses M phases to represent different symbols in PSK. BPSK uses two phases to represent a 1 and -1. The two output signals due to this modulation would then be represented as follows.

$$+1: \qquad \sqrt{\frac{2E}{T}}\cos(2\pi f t) \qquad (2-13)$$

$$-1: \qquad -\sqrt{\frac{2E}{T}}\cos(2\pi f t) \qquad (2-14)$$

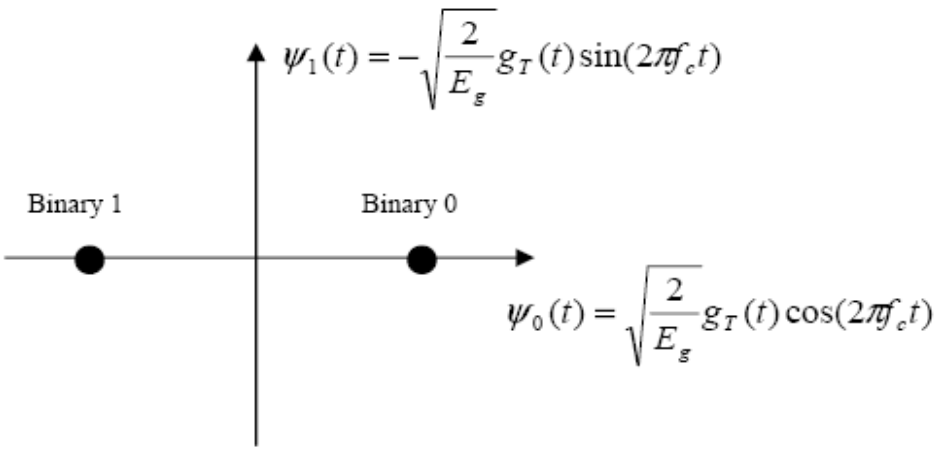

Figure 2.5: BPSK Constellation

The receiver receives a band-pass signal after the carrier multiplication. One implementation of the BPSK demodulator is the matched-filter approach. The received signal r (t) has two components: the originally transmitted signal si (t) where i could be either +1 or -1; and noise n(t), which has been introduced by the channel. The received signal r (t) is multiplied by the reference signals. The multiplied result is then integrated over one bit interval T. The resulting output would be as follows.

$$y = \frac{2E}{T}\int_0^T cos^2(2\pi ft)dt + \sqrt{\frac{2E}{T}}\int_0^T cos(2\pi ft)n(t)dt \qquad (2-15)$$

Where the first term is the result of the transmitted 1 and the second term the result of noise. Assuming that there is no noise present in the atmosphere, the output would be as follows.

$$\frac{2E}{T}\int_0^T cos^2(2\pi ft)dt = \frac{2E}{T}\left(\frac{1}{2}\right) = +\frac{E}{T} \qquad (2-16)$$

Similarly, assuming that a -1 was transmitted, the output in the absence of noise would then be as shown below. Therefore, the decision threshold decides that the transmitter sent a +1 if the integrated result is greater than 0 and -1 if the integrated result is less than 0. In this maximum likelihood detector implementation, it is assumed that the probability of sending a +1 is equal to the probability of sending -1.

Furthermore, it is assumed that the demodulator is coherent (i.e., the phase of its reference signal perfectly matches the phase of the transmitter). [5]

2-5-2 Quadrature phase-shift keying

BPSK is capable of transmitting one bit of information (+1 or -1) per symbol period T. On the other hand QPSK can transmit two bits of information per symbol period. It makes use of the quadrature component in addition to the in-phase component. The in-phase and quadrature components can be combined without interfering with each other because the two are orthogonal to each other; that is, [5]

$$\int_0^{\frac{k}{f}} cos\,(2\pi ft)\sin(2\pi ft)\,dt = 0 \qquad\qquad k = 0,1,2,\ldots\ldots\ldots\ldots$$

Therefore, a second BPSK signal in quadrature can be added to the first without introducing interference to either one. This technique, known as QPSK, effectively doubles the bandwidth efficiency of BPSK because it is able to transmit an additional bit during T. For this system to transmit four different symbols, it is necessary to have four different waveforms. These are shown below. [5]

$$Symbol\ 0:\ \ S_0(t) = \sqrt{\frac{2E}{T}}cos\left(2\pi ft + \frac{\pi}{4}\right) \qquad 0 < t < T$$

$$Symbol\ 1:\ \ S_1(t) = \sqrt{\frac{2E}{T}}cos\left(2\pi ft + \frac{3\pi}{4}\right)$$

$$= \sqrt{\frac{2E}{T}}cos\left(2\pi ft + \frac{\pi}{4}\right) \qquad 0 < t < T$$

$$Symbol\ 2:\ \ S_2(t) = \sqrt{\frac{2E}{T}}cos\left(2\pi ft + \frac{5\pi}{4}\right)$$

$$= -\sqrt{\frac{2E}{T}}cos\left(2\pi ft + \frac{\pi}{4}\right) \qquad 0 < t < T$$

$$Symbol\ 3:\ \ S_3(t) = \sqrt{\frac{2E}{T}}cos\left(2\pi ft + \frac{7\pi}{4}\right)$$

$$= -\sqrt{\frac{2E}{T}}sin\left(2\pi ft + \frac{\pi}{4}\right) \qquad 0 < t < T$$

The transmitter changes the phase of the signal depending upon which signal it has to send.

The transmitter consists of I and Q channel i.e. the two bits of the symbol are fed into the two of the channels simultaneously hence improving the data rate of the communication system.

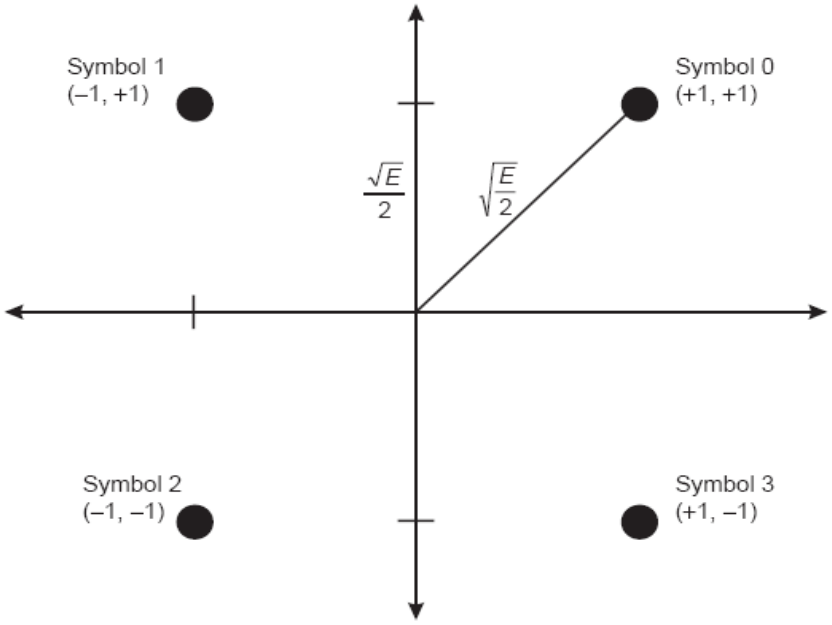

Figure 2.6: QPSK Modulator

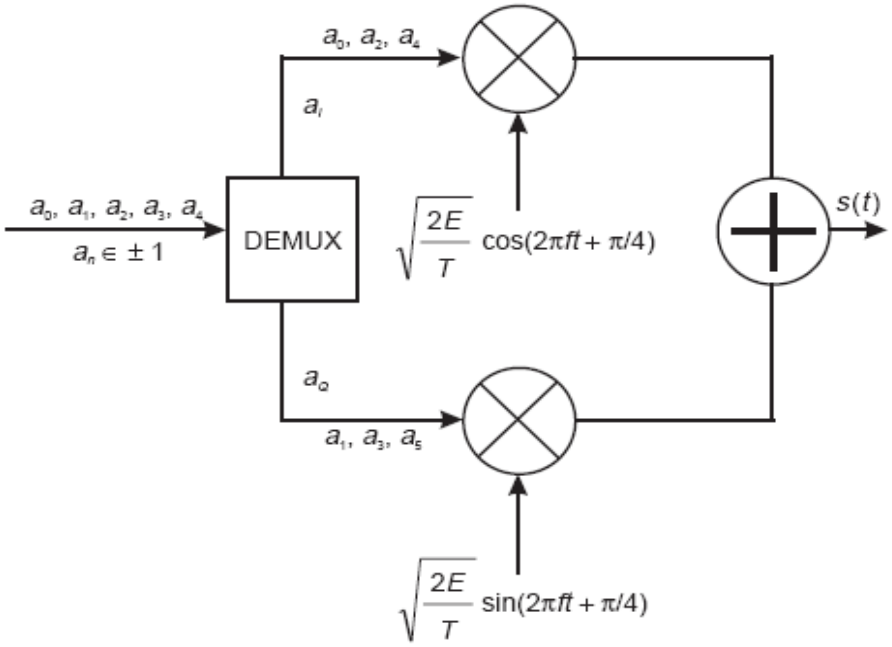

Figure 2.7: QPSK Modulator

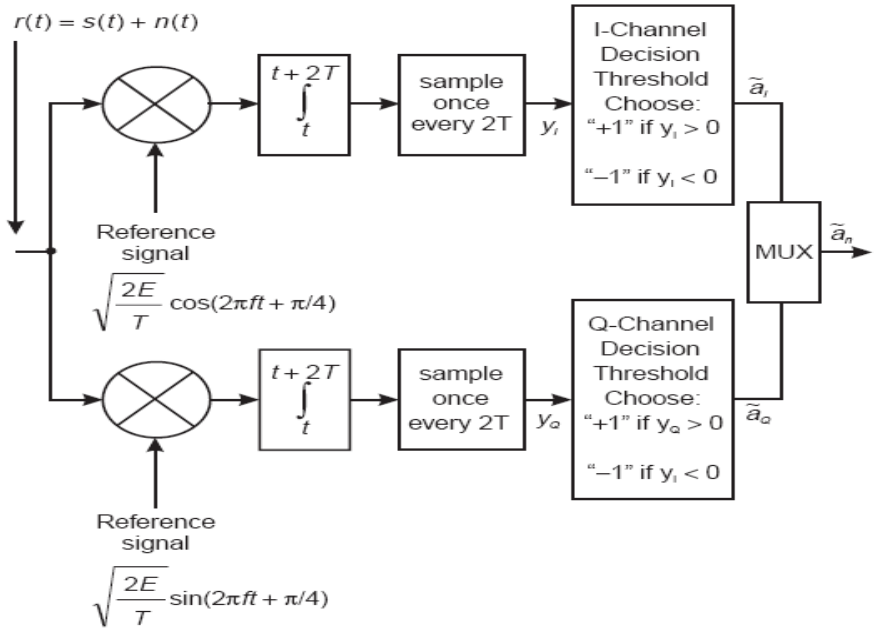

Figure 2.8: QPSK Demodulator

The detection procedure is same as that of BPSK. The received signal is multiplied by a reference signal and integrated over a time interval T. The decision is made by checking whether the received signal level is greater or less than 1.

Two bits at a time are transmitted in QPSK. Therefore, the bandwidth of the channel would be utilized to the maximum due to an increased data rate. On the other hand this technique will be inefficient in the systems where there is a limited bandwidth.

Chapter 3

Simulation of CDMA system

3-1 System description

This section describes how different baseband, band pass and multiple access techniques were used for the communication system of this project depend on Eb/No of transmitted bit.

3-2 Implementation

```
clear all
```
It removes all variables from the current workspace, releasing them from system memory. When called from within a function with persistent variables, clear reinitializes the persistent variables.

```
clc
```
It clears all input and output from the Command Window display, giving you a "clean screen."

```
streamLength = [10 100 1000 10000];
```
Create a Matrix contains the length of each stream which will be used in the following code to analyze the BER of the CDMA system with different Eb/No.

```
streamLength: 1x4 double =

          10         100        1000       10000
```

We start with a “for loop” that get each streamLength’s value and make the CDMA processing for that length as following.

```
for streamLengthN=1:length(streamLength)
    EbNo = 0.0001:1/100:10;
```

Create a Matrix called Eb/No that contains the values of Eb/No that will effect on the AWGN channel.

```
EbNo: 1x1000 double =

  Columns 1 through 9

    0.0001    0.0101    0.0201    0.0301    0.0401    0.0501    0.0601    0.0701    0.0801

  Columns 10 through 18

    0.0901    0.1001    0.1101    0.1201    0.1301    0.1401    0.1501    0.1601    0.1701

  Columns 19 through 27

    0.1801    0.1901    0.2001    0.2101    0.2201    0.2301    0.2401    0.2501    0.2601

  Columns 28 through 36

    0.2701    0.2801    0.2901    0.3001    0.3101    0.3201    0.3301    0.3401    0.3501
```

```
BER = [];
```

Create an empty matrix called BER in order to use later to fill the BER values in it.

```
BER: empty 0x0 double
```

3-3 Transmitter

```
rng('shuffle');
```

Seeds the random number generator based on the current time so that rand, randi, and randn produce a different sequence of numbers after each time you call rng.

```
spf = randi([streamLength(streamLengthN),...
streamLength(streamLengthN)],4,1);
```

Return an 4-by-1 matrix of numbers. This array containing integer values drawn from the discrete uniform distribution on the interval

[streamLength(streamLengthN), streamLength(streamLengthN)].

Note: The purpose of this command line is to generate random number of data length for each user.

```
spf: 4x1 double =

    10
    10
    10
    10
```

```
HPNSequence = comm.PNSequence('VariableSizeOutput',1,...
'MaximumOutputSize',[streamLength(streamLengthN),1]);
```

The `HPNSequence` object generates a sequence of pseudorandom binary numbers using a linear-feedback shift register (LFSR). This block implements LFSR using a simple shift register generator (SSRG, or Fibonacci) configuration. You can use a pseudonoise sequence in a pseudorandom scrambler and descrambler. You can also use one in a direct-sequence spread-spectrum system.

`'VariableSizeOutput'`: Set this property to true to enable an additional input to the step method. The default is false. When we set this property to true, the enabled input specifies the output size of the PN sequence used for the step. The input value must be less than or equal to the value of the `'MaximumOutputSize'` property.

`'MaximumOutputSize'`: Specify the maximum output size of the PN sequence as a positive integer 2-element row vector. The second element of the vector must be 1. The default is [10 1].

```
HPNSequence: 1x1 comm.PNSequence =

  System: comm.PNSequence

  Properties:
                Polynomial: [1 0 0 0 0 1 1]
    InitialConditionsSource: 'Property'
          InitialConditions: [0 0 0 0 0 1]
                 MaskSource: 'Property'
                       Mask: 0
         VariableSizeOutput: true
          MaximumOutputSize: [10 1]
             ResetInputPort: false
           BitPackedOutput: false
            OutputDataType: 'double'
```

Generates a random data stream for each user using “step” function with the required length in case of using different length of data set.

```
D1 = step(HPNSequence,spf(1))';
```

Outputs a frame of the PN sequence in column vector D1. Specify the frame length with the spf(1) property. The PN sequence has a period of N = 2^n-1, where n is the degree of the generator polynomial that you specify in the Polynomial property.

```
D1: 1x10 double =

     1     0     0     0     0     0     1     0     0     0
```

```
D2 = step(HPNSequence,spf(2))';
```

```
D2: 1x10 double =

     0     1     1     0     0     0     1     0     1     0
```

```
D3 = step(HPNSequence,spf(3))';
```

```
D3: 1x10 double =

     0     1     1     1     1     0     1     0     0     0
```

```
D4 = step(HPNSequence,spf(4))';
```

```
D4: 1x10 double =

     1     1     1     0     0     1     0     0     1     0
```

Concatenate all the data stream of each user in a single array called D.

```
D = [D1;D2;D3;D4];
```

```
D: 4x10 double =

     1     0     0     0     0     0     1     0     0     0
     0     1     1     0     0     0     1     0     1     0
     0     1     1     1     1     0     1     0     0     0
     1     1     1     0     0     1     0     0     1     0
```

```
[m,n] = size(D);
```

Returns the size of matrix D in separate variables m and n. we will use m and n in convert the unipolar data array D to bipolar data array as the “for loop” below.

```
m: 1x1 double =
     4

n: 1x1 double =
    10
```

```
% Convert to Bipolar Coding
for i=1:m
     for j=1:n
          if D(i,j)==0
               D(i,j) = -1;
          end
     end
 end
```

```
D: 4x10 double =

     1    -1    -1    -1    -1    -1     1    -1    -1    -1
    -1     1     1    -1    -1    -1     1    -1     1    -1
    -1     1     1     1     1    -1     1    -1    -1    -1
     1     1     1    -1    -1     1    -1    -1     1    -1
```

3-4 Channel Coding

```
hWalsh = comm.WalshCode('SamplesPerFrame', 4,'Length',4,'Index',0);
```

Creates a Walsh code generator object, `hWalsh`, with each specified property set to the specified value.
`'Length'`: Specify the length of the generated code as a numeric, integer scalar value that is a power of two. The default is 64.
`'SamplesPerFrame'`: Specify the number of Walsh code samples that the step method outputs as a numeric, positive, integer scalar value . The default is 1. If we set this property to a value of M, then the step method outputs M samples of a Walsh code of length N. N is the length of the code that you specify in the Length property.
`'Index'`: Specify the index of the desired code from the available set of codes as a numeric, integer scalar value in the range [0, 1, ... , N-1]. N is the value of the Length property. The default is 60. The number of zero crossings in the generated code equals the value of the specified index.

Walsh codes are defined as a set of N codes, denoted Wj, for j = 0, 1, ... , N - 1, which have the following properties:

- Wj takes on the values +1 and -1.
- Wj[0] = 1 for all j.
- Wj has exactly j zero crossings, for j = 0, 1, ... , N - 1.
- $W_j W_k^T = \begin{cases} 0 & j \neq k \\ N & j = k \end{cases}$
- Each code Wj is either even or odd with respect to its midpoint.

Walsh codes are defined using a Hadamard matrix of order N. The Walsh Code Generator block outputs a row of the Hadamard matrix specified by the Walsh code index, which must be an integer in the range [0, ..., N - 1]. If you set Walsh code index equal to an integer j, the output code has exactly j zero crossings, for j = 0, 1, ... , N - 1. Note, however, that the indexing in the Walsh Code Generator block is different than the indexing in the Hadamard Code Generator block. If you set the Walsh code index in the Walsh Code Generator block and the Code index parameter in the Hadamard Code Generator block, the two blocks output different codes.

```
hWalsh: 1x1 comm.WalshCode =

  System: comm.WalshCode

  Properties:
              Length: 4
               Index: 0
    SamplesPerFrame: 4
     OutputDataType: 'double'
```

```
C1=step(hWalsh)';
```

Outputs a frame of the Walsh code in column vector C1. Specify the frame length with the `'SamplesPerFrame'` property. The Walsh code corresponds to a row of an NxN

Hadamard matrix, where N is a nonnegative power of 2 that you specify in the 'Length' property. Use the 'Index' property to choose the row of the Hadamard matrix. The output code is in a bi-polar format with 0 and 1 mapped to 1 and -1 respectively.

```
C1: 1x4 double =

     1     1     1     1
```

```
hWalsh = comm.WalshCode('SamplesPerFrame', 4,'Length',4,'Index',1);
C2=step(hWalsh)';
```

```
C2: 1x4 double =

     1     1    -1    -1
```

```
hWalsh = comm.WalshCode('SamplesPerFrame', 4,'Length',4,'Index',2);
C3=step(hWalsh)';
```

```
C3: 1x4 double =

     1    -1    -1     1
```

```
hWalsh = comm.WalshCode('SamplesPerFrame', 4,'Length',4,'Index',3);
C4=step(hWalsh)';
```

```
C4: 1x4 double =

     1    -1     1    -1
```

Concatenate the orthogonal codes in one array called C.

```
C = [C1;C2;C3;C4];
```

```
C: 4x4 double =

     1     1     1     1
     1     1    -1    -1
     1    -1    -1     1
     1    -1     1    -1
```

```
M = length(C);      % length (number of bits) of code
```

Finds the number of elements along the largest dimension of an array. Array is an array of any MATLAB® data type and any valid dimensions. M is a whole number of the MATLAB double class.

```
M: 1x1 double =

     4
```

```
[N,I] = size(D);    %(N) number of unique senders / bit streams (I) ...
% number of bits per stream
```

N: 1x1 double =	I: 1x1 double =
4	10

```
T = [];             % sum of all transmitted and encoded data on
channel
```

T: empty 0x0 double

```
RECON = [];         % vector of reconstructed bits at receiver
```

RECON: empty 0x0 double

Create an empty matrices TX and RX in order to use it in the BER function

```
TX = [];
RX = [];
```

```
% show data bits and codes
% 'Vector of data bits to be transmitted:', D
% 'Vector of codes used for transmission:', C
```

```
%% Encode bits and transmit
G = zeros(I,M);
```

Creates an array G with I x M dimension. That contains zero values.

G: 10x4 double =

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

The next “for loop” make the CDMA coding for each data set with corresponding user code and concatenate them in G array according to the following Figure.

```
for n = 1:N
    Z = zeros(I,M);
    for i = 1:I
        for m = 1:M
            Z(i,m) = D(n,i)*C(n,m);
        end
    end
  G = G + Z;
End
```

```
G: 10x4 double =

     0     0     4     0
     2    -2    -2    -2
     2    -2    -2    -2
    -2    -2    -2     2
    -2    -2    -2     2
    -2    -2     2    -2
     2     2    -2     2
    -4     0     0     0
     0     0     0    -4
    -4     0     0     0
```

Simulate the T array as it is a stream in the channel.

```
% show channel traffic
 for i = 1:I
     T = [ T G(i,:) ];
 end
% 'Resulting traffic on the channel:', G
```

```
T: 1x40 double =

  Columns 1 through 16

    0     0     4     0     2    -2    -2    -2     2    -2    -2    -2    -2    -2    -2     2

  Columns 17 through 32

   -2    -2    -2     2    -2    -2     2    -2     2     2    -2     2    -4     0     0     0

  Columns 33 through 40

    0     0     0    -4    -4     0     0     0
```

3-5 The AWGN Channel

Create an AWGN channel with different value of Eb/No Value in each iteration.

```
%% Channel Configuration
 HawgnChannel = comm.AWGNChannel;
```

The AWGN Channel object adds white Gaussian noise to a real or complex input signal. When the input uses a real- valued signal, this object adds real Gaussian noise and produces a real output signal. When the input uses a complex signal, this object adds complex Gaussian noise and produces a complex output signal.

In reality, transmission is always corrupted by noise. The usual mathematical model is the AWGN (Additive White Gaussian Noise) channel. It is a very good model for the physical reality as long as the thermal noise at the receiver is the only source of disturbance. Nevertheless, because of its simplicity, it is often used to model man-made

noise or multiuser interference. The AWGN channel model can be characterized as follows: ·

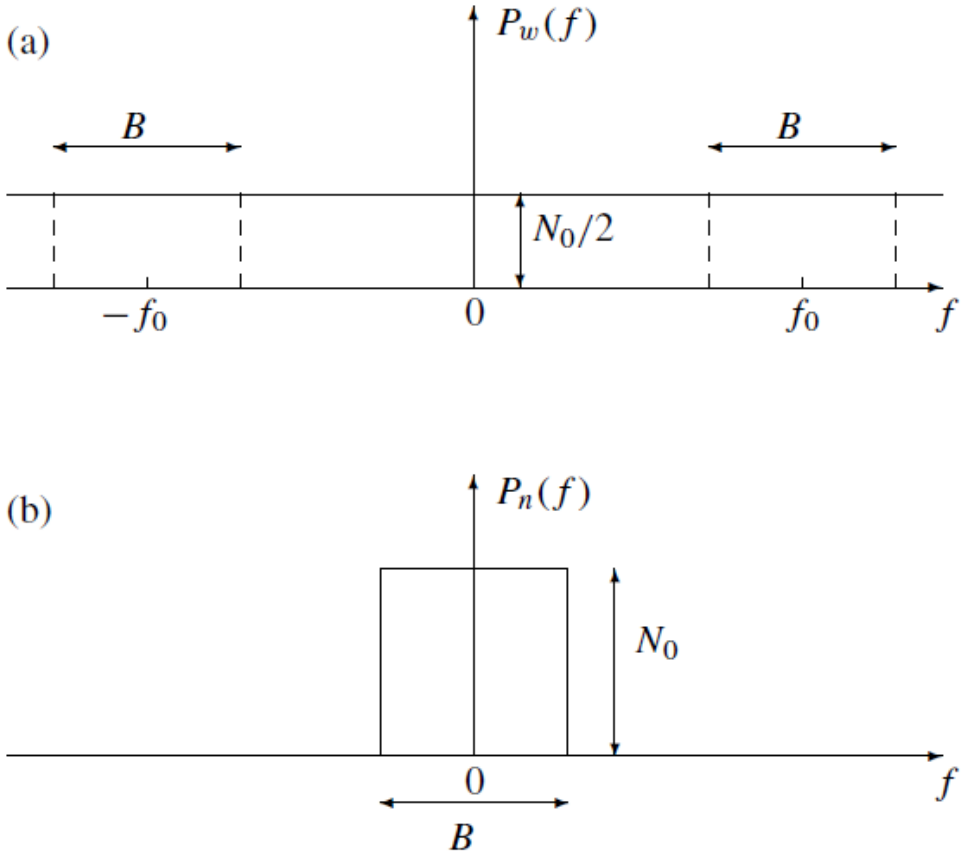

Figure 3.1 Equivalence wideband and baseband AWGN

As seen in Figure 3.1 Equivalence of (a) wideband and (b) complex baseband AWGN for band limited detection.

The noise w(t) is an additive random disturbance of the useful signal s(t), that is, the receive signal is given by r(t) = s(t) + w(t).

The noise is white, that is, it has a constant power spectral density (psd). The onesided psd is usually denoted by N0, so N0/2 is the two-sided psd, and BN0 is the noise inside the (noise) bandwidth B, see part (a) of Figure 1.13. For thermal resistor noise, N0 = kT0, where k is the Boltzmann constant and T0 is the absolute temperature. The unit of N0 is [W/Hz], which is the same as the unit [J] for the energy. Usually, N0 is written as dBm/Hz. For T0 = 290 K, N0 ≈ −174 dBm/Hz. However, this is only the ideal physical limit for an ideal receiver. In practice, some decibels according to the so-called

noise Figure have to be added. Typically, N0 will be a value slightly above −170 dBm/Hz.

The noise is a stationary and zero mean Gaussian random process. This means that the output of every (linear) noise measurement is a zero mean Gaussian random variable that does not depend on the time instant when the measurement is done.

One must keep in mind that the AWGN model is a mathematical fiction, because it implies that the total power (i.e. the psd integrated over all frequencies) is infinite. Thus, a time sample of the white noise has infinite average power, which is certainly not a physically reasonable property. It is known from statistical physics that the thermal noise density decreases exponentially at (very!) high frequencies. But to understand the physical situation in communications engineering it is better to keep in mind that every receiver limits the bandwidth as well as every physical noise measurement. So it makes sense to think of the noise process to be white, but it cannot be sampled directly without an input device. Each input device filters the noise and leads to a finite power.

3-6 Signal Processing and Input Dimensions

This block can process multichannel signals. When you set the Input Processing parameter to Columns as channels (frame based), the block accepts an M-by-N input signal. M specifies the number of samples per channel and N specifies the number of channels. Both M and N can be equal to 1. The block adds frames of length-M Gaussian noise to each of the N channels, using a distinct random distribution per channel.

3-7 Specifying the Variance Directly or Indirectly

You can specify the variance of the noise generated by the AWGN Channel block using one of these modes:

- Signal to noise ratio (Eb/No), where the block calculates the variance from these quantities that you specify in the dialog box:
    - **Eb/No**, the ratio of bit energy to noise power spectral density
    - **Number of bits per symbol**
    - **Input signal power**, the actual power of the symbols at the input of the block
    - **Symbol period**

- Signal to noise ratio (Es/No), where the block calculates the variance from these quantities that you specify in the dialog box:
    - **Es/No**, the ratio of signal energy to noise power spectral density
    - **Input signal power**, the actual power of the symbols at the input of the block
    - **Symbol period**
- Signal to noise ratio (SNR), where the block calculates the variance from these quantities that you specify in the dialog box:
    - **SNR**, the ratio of signal power to noise power
    - **Input signal power**, the actual power of the samples at the input of the block
- Variance from mask, where you specify the variance in the dialog box. The value must be positive.
- Variance from port, where you provide the variance as an input to the block. The variance input must be positive, and its sampling rate must equal that of the input signal.

Changing the symbol period in the AWGN Channel block affects the variance of the noise added per sample, which also causes a change in the final error rate.

$$NoiseVariance = \frac{SignalPower \times SymbolPeriod}{SampleTime \times 10^{\frac{Es/No}{10}}}$$

A good rule of thumb for selecting the Symbol period value is to set it to be what you model as the symbol period in the model. The value would depend upon what constitutes a symbol and what the oversampling applied to it is (e.g., a symbol could have 3 bits and be oversampled by 4).

In both Variance from mask mode and Variance from port mode, these rules describe how the block interprets the variance:

- If the variance is a scalar, then all signal channels are uncorrelated but share the same variance.
- If the variance is a vector whose length is the number of channels in the input signal, then each element represents the variance of the corresponding signal channel.

3-8 Relationship Among Eb/No, Es/No, and SNR Modes

For complex input signals, the AWGN Channel block relates Eb/N0, Es/N0, and SNR according to the following equations:

Es/N0 = (Tsym/Tsamp) · SNR

Es/N0 = Eb/N0 + 10log10(k) in dB

Where:

- E_s = Signal energy (Joules)
- E_b = Bit energy (Joules)
- N_0 = Noise power spectral density (Watts/Hz)
- T_{sym} is the **Symbol period** parameter of the block in `Es/No` mode
- k is the number of information bits per input symbol
- T_{samp} is the inherited sample time of the block, in seconds

For real signal inputs, the AWGN Channel block relates Es/N0 and SNR according to the following equation:

Es/N0 = 0.5 (Tsym/Tsamp) · SNR

Note that the equation for the real case differs from the corresponding equation for the complex case by a factor of 2. This is so because the block uses a noise power spectral density of N0/2 Watts/Hz for real input signals, versus N0 Watts/Hz for complex signals.

```
HawgnChannel: 1x1 comm.AWGNChannel =

  System: comm.AWGNChannel

  Properties:
        NoiseMethod: 'Signal to noise ratio (Eb/No)'
              EbNo: 10
      BitsPerSymbol: 1
        SignalPower: 1
    SamplesPerSymbol: 1
       RandomStream: 'Global stream'
```

The following “for loop” calculate the BER for each value of Eb/No and save the results in BER matrix to show the result at a Figure.

```
for EbNoValue=1:length(EbNo)
      TX = [];
      RX = [];
      RECON = [];
      HawgnChannel.EbNo = EbNo(EbNoValue);
```

Changes the AWGN channel's Eb/No value to the corresponding value in EbNo matrix indexed with Eb/No Value counter.

```
      channelOutput = step(HawgnChannel,G);
```

Adds white Gaussian noise to input G and returns the result in channelOutput. Depending on the value of the FrameBasedProcessing property, input G can be a double or single precision data type scalar, vector, or matrix with real or complex values.

```
channelOutput: 10x4 double =

    0.8069    0.6302    4.1602    0.2364
    3.1375   -1.8534   -1.1814   -1.5300
    2.1252   -1.8177   -1.9053   -3.2046
   -2.0100   -2.0079   -2.5115    3.3046
   -2.7352   -1.8944   -1.9856    1.9659
   -1.0154   -2.3278    2.6098   -2.8455
    1.5746    1.5227   -1.4506    2.3132
   -3.9876    0.5606   -0.7049    0.2432
   -0.1129    0.2149    0.0192   -2.8056
   -3.0827   -0.2698   -0.1104    0.0893
```

The following "for loop" decode and reconstruct the channelOutput Matrix that resulted from AWGN channel by multiply the received data with the corresponding code. After that we sum all the resulted matrix and divided on the number of user M to get the Reconstructed matrix RECON.

```
      % decode and reconstruct
      for n = 1:N
          TOT = zeros(1,I);
          R   = zeros(I,M);
          for i = 1:I
              for m = 1:M
                  R(i,m) = channelOutput(i,m) * C
(n,m);
                  TOT(i) = TOT(i) + R (i,m);
              end
          end
          RECON = [RECON ; TOT / M];
      end
      RECON = round(RECON);
```

```
%   'Reconstructed data at the receiver:'
%   RECON
```

The following “for loop” just prepare the TX and RX matrices to be suitable for the BER function object hError.

```
for i = 1:N
    TX = [TX D(i,:)];
    RX = [RX RECON(i,:)];
end
TX = TX';
RX = RX';

hError = comm.ErrorRate();
```

3-8-1 Input Data

This block has between two and four input ports, depending on how you set the dialog parameters. The inports marked Tx and Rx accept transmitted and received signals, respectively. The Tx and Rx signals must share the same sampling rate.

The Tx and Rx input ports accept scalar or column vector signals. For information about the data types each block port supports, see the Supported Data Types table on this page. If Tx isa scalar and Rx is a vector, or vice-versa, the nthe block compares the scalar with each element of the vector. (Overall, the block behaves as if you had preprocessed the scalar signal with the Communications System Toolbox™ Repeat block with the Rate options parameter set to Enforce single rate.)

- If you select Reset port, then an additional input port appears, labeled Rst. The Rst input accepts only a scalar signal (of type double or Boolean) and must have the same port sample time as the Tx and Rx ports. When the Rst input is nonzero, the block clear sand then recomputed the error statistics.
- If you set the Computation mode parameter to select samples from port, then an additional input port appears, labeled Sel. The Sel input indicates which elements of a frame are relevant for the computation. The Sel input can be a column vector of type double.
- The guidelines below indicate how you should configure the inputs and the dialog parameters depending on how you want this block to interpret your Tx and Rx data.

- If both data signals are scalar, then this block compares the Tx scalar signal with the Rx scalar signal. For this configuration, use the Computation mode parameter default value, Entire frame.
- If both data signals are vectors, then this block compares some or all of the Tx and Rx data:
- If you set the Computation mode parameter to Entire frame, then the block compares all of the Tx frame with all of the Rx frame.
- If you set the Computation mode parameter to Select samples from mask, then the Selected samples from frame field appears in the dialog. This parameter field accepts a vector that lists the indices of those elements of the Rx frame that you want the block to consider. For example, to consider only the first and last elements of a length-six receiver frame, set the Selected samples from frame parameter to [1 6]. If the Selected samples from frame vector includes zeros, then the block ignores them.
- If you set the Computation mode parameter to Select samples from port, then an additional input port, labeled Sel, appears on the block icon. The data at this input port must have the same format as that of the Selected samples from frame parameter described above.
- If one data signal is a scalar and the other is a vector, then this block compares the scalar with each entry of the vector. The three sub bullets above are still valid for this mode, except that if Rx is a scalar, then the phrase "Rx frame" above refers to the vector expansion of Rx.

3-8-2 Output Data

This block produces a vector of length three, whose entries correspond to:

A. The error rate

The total number of errors, that is, the number of instances that an Rx element does not match the corresponding Tx element.

The total number of comparisons that the block made

The block sends this output data to the base MATLAB® workspace or to an output port, depending on how you set the Output data parameter:

- If you set the Output data parameter to Workspace and fill in the Variable name parameter, then that variable in the base MATLAB workspace contains the current value when the simulation ends. Pausing the simulation does not cause the block to write interim data to the variable.

- If you plan to use this block along with the Simulink® Coder™ software, then you should not use the Workspace option. Instead, use the Port option and connect the output port to a Simulink To Workspace block.

- If you set the Output data parameter to Port, then an output port appears. This output port contains the running error statistics.

B. Delays

The Receive delay and Computation delay parameters implement two different types of delays for this block. One delay is useful if you want this block to compensate for the delay in the received signal. The other is useful if you want to ignore the initial transient behavior of both input signals.

The Receive delay parameter represents the number of samples by which the received data lags behind the transmitted data. The transmit signal is implicitly delayed by that same amount before the block compares it to the received data. This value is helpful when you delay the transmit signal so that it aligns with the received signal. The receive delay persists throughout the simulation.

The Computation delay parameter represents the number of samples the block ignores at the beginning of the comparison.

If you do not know the receive delay in your model, you can use the Align Signals block, which automatically compensates for the delay. If you use the Align Signals block, set the Receive delay in the Error Rate Calculation block to 0 and the Computation delay to the value coming out of the Delay port of the Align Signals block.

Alternatively, you can use the Find Delay block to find the value of the delay, and then set the Receive delay parameter in the Error Rate Calculation block to the delay value.

If you use the Select samples from mask or Select samples from port option, then each delay parameter refers to the number of samples that the block receives, whether the block ultimately ignores some of them or not.

Stopping the Simulation Based on Error Statistics

You can configure this block so that its error statistics control the duration of simulation. This is useful for computing reliable steady-state error statistics without knowing in advance how long transient effects might last. To use this mode, check Stop simulation. The block attempts to run the simulation until it detects the number of errors the Target number of errors parameter specifies. However, the simulation stops before detecting enough errors if the time reaches the model's Stop time setting(in the Configuration Parameters dialog box),if the Error Rate Calculation block makes Maximum number of symbols comparisons, or if another block in the model directs the simulation to stop.

To ignore either of the two stopping criteria in this block, set the corresponding parameter (Target number of errors or Maximum number of symbols) to Inf. For example, to reach a target number of errors without stopping the simulation early, set Maximum number of symbols to Inf and set the model's Stop time to Inf.

3-9 BER calculation

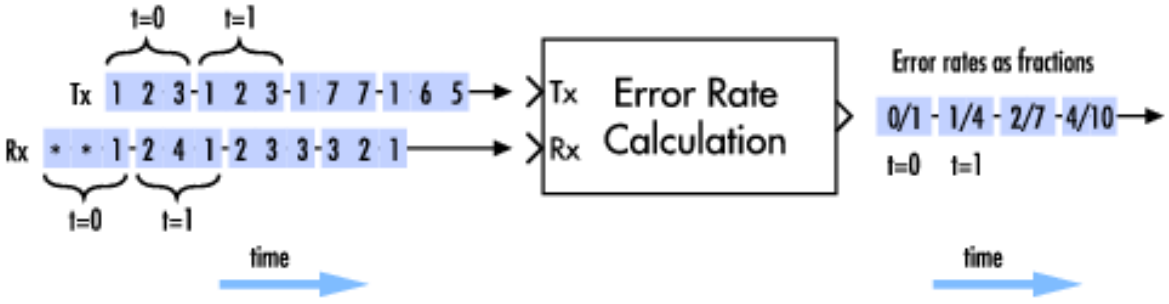

Figure 3.2 Error rate calculation

```
        error = step(hError,TX,RX);
```

Counts the number of differences between the transmitted data vector (see Figure 3.2), TX, and received data vector, RX. The step method outputs three-element vector consisting of the error rate, followed by the number of errors detected and the total number of samples compared. TX and RX inputs can be either scalars or column vectors of the same data type. Validate types are single, double, integer or logical. If TX isa scalar and RX is a vector, or vice-versa, then the block compares the scalar with each element of the vector.

```
        BER = [BER error(1,1)];
    end
    subplot(2,2,streamLengthN),plot(EbNo,BER),...
        xlabel('EbNo (dB)'),ylabel('BER (%)')
end
```

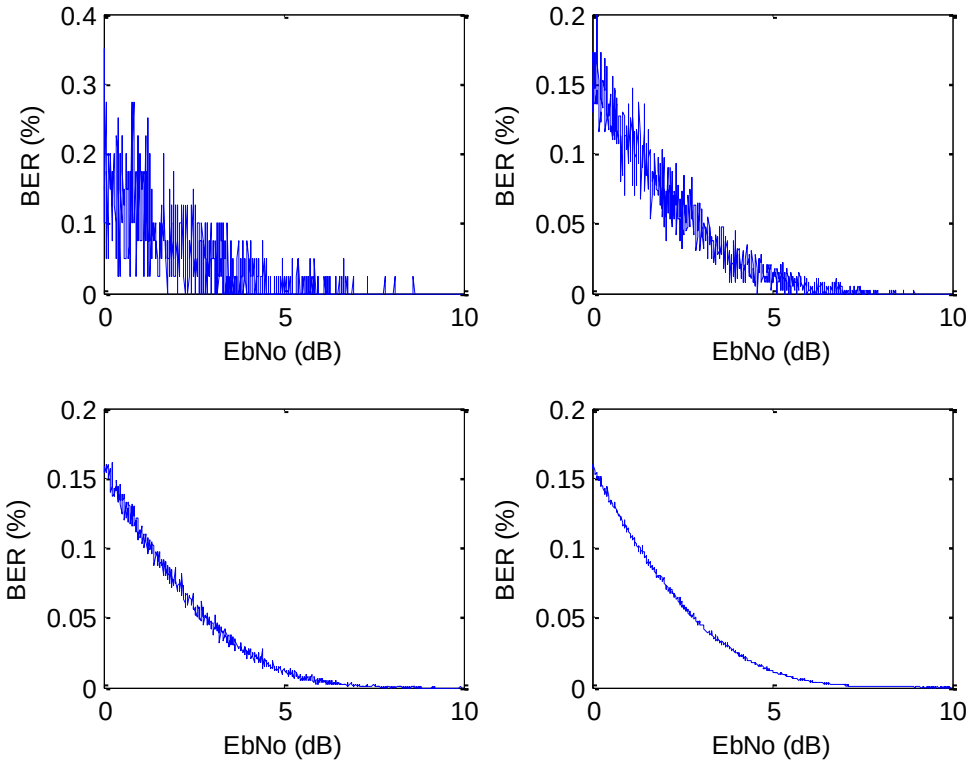

Figure 3.3 RER of number of data stream

As we can see from the Figure 3.3 above that as the number of data stream increase the Figures converge to theoretical results.

We create a GUI application that take the minimum and maximum number of data stream and the minimum and maximum values of Eb/No and calculate the BER as previous explanation. We can see the GUI application in the Figure 3.4 and Figure 3.5 below.

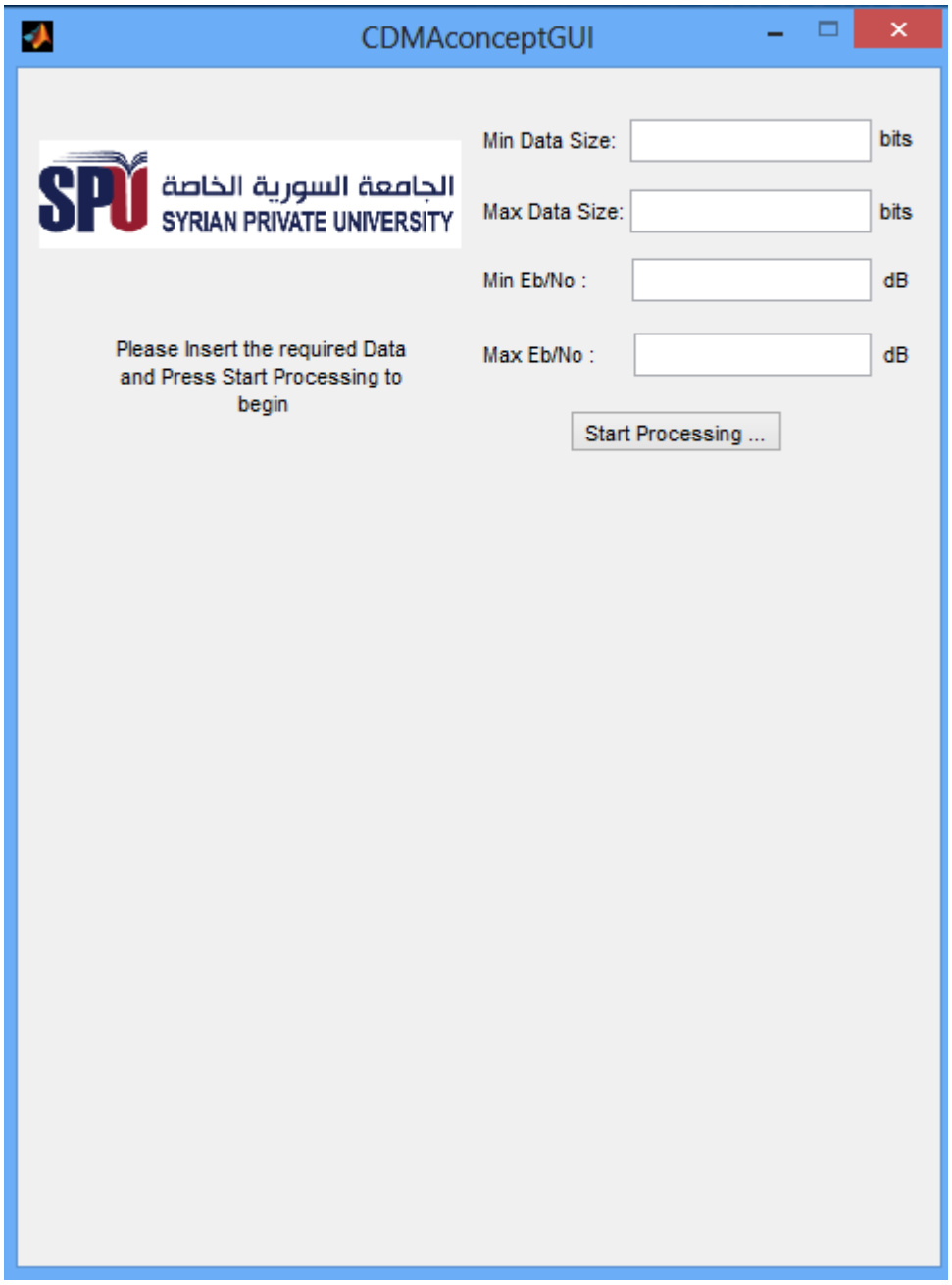

Figure 3.4 GUI interface

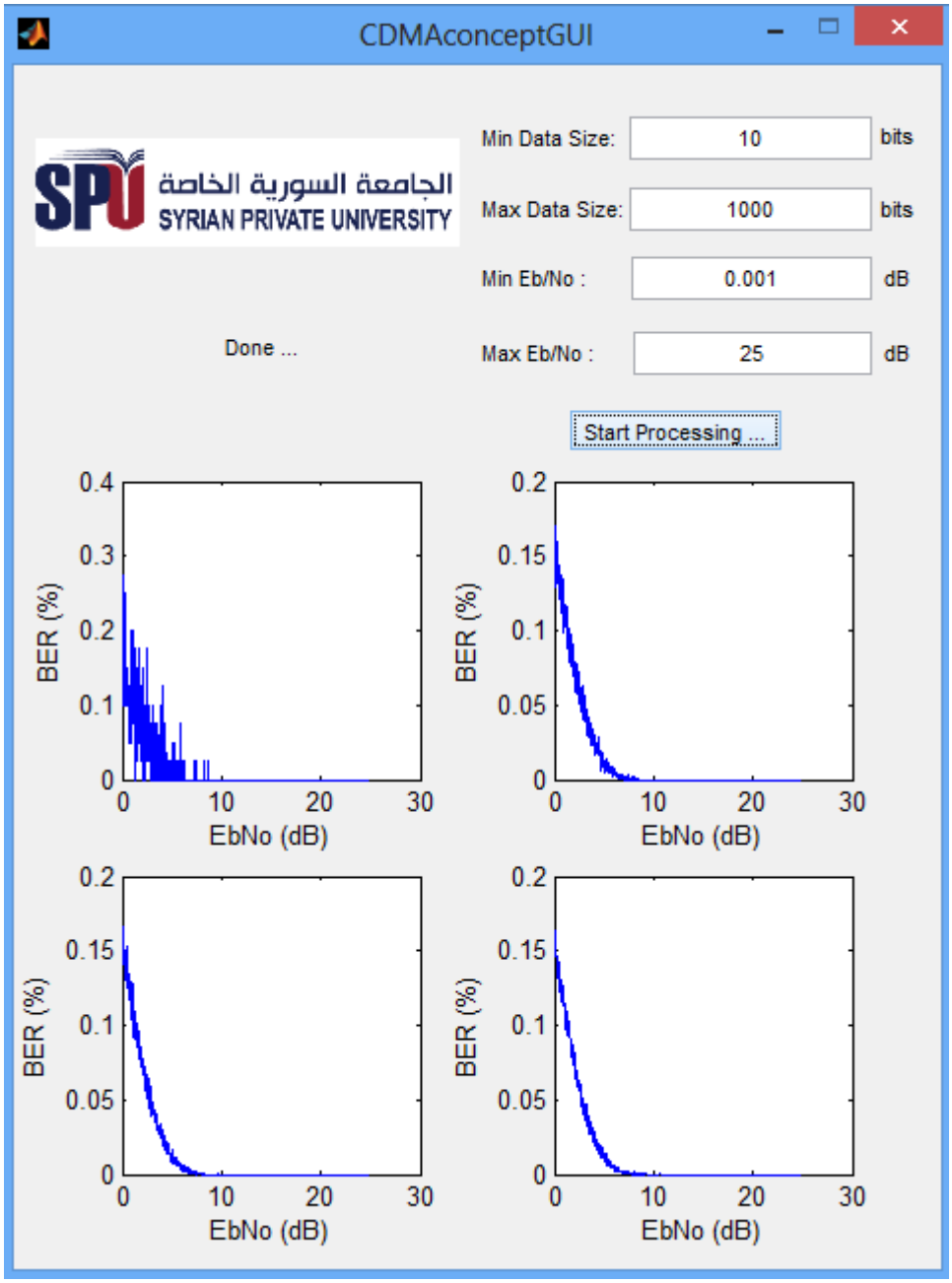

Figure 3.5 GUI results

Chapter 4

Project Summary & Future Development

Some of the observations that were made during the projects were, that first of all the system can use Walsh code instead of any other code because this system can implement at short distances or far distance, and therefore there will be negligible amount of multipath present. Secondly, the modulation technique used in the project would be BPSK due limitation of data rate provided by the sound card.

4-1 Final Overview

The implementation and simulation of system include the channel estimation of the environment. This requirement of the project has been completed. Furthermore a simulation of the CDMA system has been conducted for testing purposes.

After the completion of the project, the system can now transmit and receive N bit binary messages at a measurable BER. The system is properly synchronized, i.e. the receiver is listening all the time and when the signal of interest is transmitted it extracts it out of the noise.

4-2 Project Enhancements

There can be a number of developments in this project to take this to a new level. One of the major developments can be to use this system in Rael-Time application to know BER of AWGN channel or another communication channel.

Presently, the coding or orthogonal coder can use in this application, because it's better than PN-Sequence in Multi Carrier Modulation. In this way it would be a wireless system in the true essence.

REFERENCES

[1] Theodore.S.Rappaport, *Wireless communications Principles and practice (2nd edition)*, Prentice Hall PTR, INC.USA 2003.

[2] Shetty, Kiran Kumar "*A NOVEL ALGORITHM FOR UPLINK INTERFERENCE SUPPRESSION USING SMART ANTENNAS IN MOBILE COMMUNICATIONS"* Electronic Thesis & Dissertations, vol. 14, *pp 35-92, Feb.* 2004.

[3] *Spread Spectrum Communications and CDMA,* UCL Communications and Remote Sensing Lab, Bruxelles, Germany.

[4] Kamil Sh. Zigangirov, John B. Anderson *Theory of code division multiple access,* John Wiley & SONS, INC., 2004.

[5] Samuel.C.Yang, *CDMA RF system Engineering,* Artech House MA, USA, 1998.

[6] Bernard Sklar, *Digital Communications Fundamentals and applications* ,Prentice Hall INC. USA, Upper Saddle River, New Jersey, USA,1998.

Appendix

MATLAB Code

```
function varargout = CDMAconceptGUI(varargin)
% CDMACONCEPTGUI MATLAB code for CDMAconceptGUI.fig
%      CDMACONCEPTGUI, by itself, creates a new
CDMACONCEPTGUI or raises the existing
%      singleton*.
%
%      H = CDMACONCEPTGUI returns the handle to a new
CDMACONCEPTGUI or the handle to
%      the existing singleton*.
%
%
CDMACONCEPTGUI('CALLBACK',hObject,eventData,handles,...)
calls the local
%      function named CALLBACK in CDMACONCEPTGUI.M with
the given input arguments.
%
%      CDMACONCEPTGUI('Property','Value',...) creates a
new CDMACONCEPTGUI or raises the
%      existing singleton*.  Starting from the left,
property value pairs are
%      applied to the GUI before
CDMAconceptGUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes
property application
%      stop.  All inputs are passed to
CDMAconceptGUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose
"GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
CDMAconceptGUI

% Last Modified by GUIDE v2.5 27-Apr-2014 02:42:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @CDMAconceptGUI_OpeningFcn, ...
    'gui_OutputFcn',  @CDMAconceptGUI_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
```

```
    'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before CDMAconceptGUI is made
visible.
function CDMAconceptGUI_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
% varargin   command line arguments to CDMAconceptGUI
(see VARARGIN)

% Choose default command line output for CDMAconceptGUI
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes CDMAconceptGUI wait for user response (see
UIRESUME)
% uiwait(handles.figure1);
axes(handles.axes5)
I = imread('newlogo.png');
imshow(I);
set(handles.variablestatictxt,'String','Please Insert the
required Data and Press Start Processing to begin');
set(handles.axes1,'visible','off');
set(handles.axes2,'visible','off');
set(handles.axes3,'visible','off');
set(handles.axes4,'visible','off');

% --- Outputs from this function are returned to the
command line.
function varargout = CDMAconceptGUI_OutputFcn(hObject,
eventdata, handles)
```

```
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function mindatasizetxtbx_Callback(hObject, eventdata,
handles)
% hObject    handle to mindatasizetxtbx (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
mindatasizetxtbx as text
%        str2double(get(hObject,'String')) returns
contents of mindatasizetxtbx as a double


% --- Executes during object creation, after setting all
properties.
function mindatasizetxtbx_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to mindatasizetxtbx (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function maxdatasizetxtbx_Callback(hObject, eventdata,
handles)
% hObject    handle to maxdatasizetxtbx (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
maxdatasizetxtbx as text
%        str2double(get(hObject,'String')) returns
contents of maxdatasizetxtbx as a double


% --- Executes during object creation, after setting all
properties.
function maxdatasizetxtbx_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to maxdatasizetxtbx (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function minebnotxtbx_Callback(hObject, eventdata,
handles)
% hObject    handle to minebnotxtbx (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
minebnotxtbx as text
%        str2double(get(hObject,'String')) returns
contents of minebnotxtbx as a double


% --- Executes during object creation, after setting all
properties.
function minebnotxtbx_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to minebnotxtbx (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function maxebnotxtbx_Callback(hObject, eventdata,
handles)
% hObject    handle to maxebnotxtbx (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
maxebnotxtbx as text
%        str2double(get(hObject,'String')) returns
contents of maxebnotxtbx as a double


% --- Executes during object creation, after setting all
properties.
function maxebnotxtbx_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to maxebnotxtbx (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in startprocessingpushbtn.
function startprocessingpushbtn_Callback(hObject,
eventdata, handles)
```

```
% hObject    handle to startprocessingpushbtn (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

%% Get the data from the text boxes
mindatasize =
str2double(get(handles.mindatasizetxtbx,'String'));
maxdatasize =
str2double(get(handles.maxdatasizetxtbx,'String'));
minebno = str2double(get(handles.minebnotxtbx,'String'));
maxebno = str2double(get(handles.maxebnotxtbx,'String'));

%% Parameter Initialization
streamLength = linspace(mindatasize,maxdatasize,4);

for streamLengthN=1:length(streamLength)
    switch streamLengthN
        case 1
            axes(handles.axes1);
            set(handles.variablestatictxt,'String','1st
Iteration, Please Wait ...');
        case 2
            axes(handles.axes2);
            set(handles.variablestatictxt,'String','2nd
Iteration, Please Wait ...');
        case 3
            axes(handles.axes3);
            set(handles.variablestatictxt,'String','3rd
Iteration, Please Wait ...');
        otherwise
            axes(handles.axes4);
            set(handles.variablestatictxt,'String','Last
Iteration, Please Wait ...');
    end
    EbNo = linspace(minebno,maxebno,1000);
    BER = [];

    %% Generate Random data bit stream for each sender
    rng('shuffle');
    spf =
randi([streamLength(streamLengthN),streamLength(streamLen
gthN)],4,1);

    HPNSequence = comm.PNSequence('VariableSizeOutput',1,
'MaximumOutputSize',[streamLength(streamLengthN),1]);
    D1 = step(HPNSequence,spf(1))';
    D2 = step(HPNSequence,spf(2))';
    D3 = step(HPNSequence,spf(3))';
    D4 = step(HPNSequence,spf(4))';
```

```
    D = [D1;D2;D3;D4];
    [m,n] = size(D);

    % Convert to Biploar Coding
    for i=1:m
        for j=1:n
            if D(i,j)==0
                D(i,j) = -1;
            end
        end
    end

    %% Generate Orthogonal Code using OVSF algorithm
    hWalsh = comm.WalshCode('SamplesPerFrame',
4,'Length',4,'Index',0);
    C1=step(hWalsh)';
    hWalsh = comm.WalshCode('SamplesPerFrame',
4,'Length',4,'Index',1);
    C2=step(hWalsh)';
    hWalsh = comm.WalshCode('SamplesPerFrame',
4,'Length',4,'Index',2);
    C3=step(hWalsh)';
    hWalsh = comm.WalshCode('SamplesPerFrame',
4,'Length',4,'Index',3);
    C4=step(hWalsh)';

    C = [C1;C2;C3;C4];

    %% Processing Parameters
    M = length(C);      % length (number of bits) of code
    [N,I] = size(D);    %(N) number of unique senders /
bit streams (I) number of bits per stream
    T = [];             % sum of all transmitted and
encoded data on channel
    RECON = [];         % vector of reconstructed bits at
receiver
    TX = [];
    RX = [];

    % show data bits and codes
    % 'Vector of data bits to be transmitted:', D
    % 'Vector of codes used for transmission:', C

    %% Encode bits and transmit
    G = zeros(I,M);
    for n = 1:N
        Z = zeros(I,M);
        for i = 1:I
            for m = 1:M
                Z(i,m) = D(n,i)*C(n,m);
            end
```

```
        end
        G = G + Z;
    end


    % show channel traffic
    for i = 1:I
        T = [ T G(i,:) ];
    end
    % 'Resulting traffic on the channel:', G

    %% Channel Configuration
    HawgnChannel = comm.AWGNChannel;
    for EbNoValue=1:length(EbNo)
        TX = [];
        RX = [];
        RECON = [];
        HawgnChannel.EbNo = EbNo(EbNoValue);
        channelOutput = step(HawgnChannel,G);
        % 'Resulting traffic after the channel:',
channelOutput
        % decode and reconstruct
        for n = 1:N
            TOT = zeros(1,I);
            R   = zeros(I,M);
            for i = 1:I
                for m = 1:M
                    R(i,m) = channelOutput(i,m) * C
(n,m);
                    TOT(i) = TOT(i) + R (i,m);
                end
            end
            RECON = [RECON ; TOT / M];
        end
        RECON = round(RECON);
        %  'Reconstructed data at the receiver:'
        %  RECON

        for i = 1:N
            TX = [TX D(i,:)];
            RX = [RX RECON(i,:)];
        end
        TX = TX';
        RX = RX';
        hError = comm.ErrorRate();
        error = step(hError,TX,RX);
        BER = [BER error(1,1)];
    end
    plot(EbNo,BER),xlabel('EbNo (dB)'),ylabel('BER (%)')
end
set(handles.variablestatictxt,'String','Done ...');
```