IMPLEMENTATION / SIMULATION OF A UNIVERSAL MODULATION IDENTIFIER AND DEMODULATOR FOR COMMUNICATION INTELLIGENT (COMMINT) SYSTEM

By

GC Hassan Maqbool
GC Mehmood ur Rehman
GC Fahad Nadir
GC Umair Gillani

DS

Maj Imran Rashid
Maj Dr. Shoab Ahmad Khan
(Center for Advanced Research in Engineering)

Electrical Engineering Department
Military College of Signals, Rawalpindi

April 2005

To our beloved Parents

and

Friends

"Creating interest is a great way to give yourself the motivation to achieve your goals."

DECLARATION

NO PART OF THIS DOCUMENT HAS BEEN PRESENTED AND PUBLISHED ELSEWHERE IN WHOLE OR PARTIAL FORM.

PREFACE

The concept of modulation identification first evolved in the last quarter of the last century. Since then many researchers and developers have developed different methods and algorithms. Out of the many algorithms and methods available a few are based on the maximum likelihood method, statistical method, constellation rebuilding and recognition, wavelets analysis of the signal spectrum, vector space recognition and the power matrices methods. Many algorithms have been developed since then for the modulation identification. But none of the algorithms, however, has been able to identify more than 3 or 4 modulation techniques simultaneously. Hence a Universal Modulation Identification Algorithm (UMIA) has yet not been possible.

The concept of modulation identification relates to the idea of integration of different digital receivers to form a single, common receiver. This receiver shall be able to receive, demodulate and decode all the signals received by it. The integration of receivers here does not mean just the combination of the receivers into a single box but the working on a common platform with *'minimum number of resources'* and *'maximum processing speed'*.

Modulation identification is a specific area in the field of communication theory. In cooperative communication theory, almost of parameters involved in the identification at the receiver end are supposed to be known a priori or are accessible. In a general receiver, the parameters including modulation type, carrier frequency, bandwidth, symbol rate, and carrier phase is known. Therefore, the received signal can be demodulated without ambiguity and then the data can be extracted. Modulation identification, which is used to identify the modulation type of a received signal, however, is another story. Intrinsically, modulation identifier does not have much information about the signal emitter, in our case only the bit symbol duration is required to be known, and modulation identification is generally invisible to the signal emitter and is used as a means to pick up some useful information. The criterion of performance of receiver in digital communication systems is usually the bit error probability; in the meanwhile, the criterion for modulation identification is the probability of correct identification.

With the rapid growth in the radio communication technologies, a concept of software defined radio (SDR) has been introduced to integrate wireless applications working over any air interface and protocol. SDR can provide the standard communication platforms for adaptive receiver which enables non-restrictive wireless roaming across numerous radio technologies such as TDMA, CDMA, and GSM. Users with adaptive receiver can communicate in all the regions of the world where different communication standard is adopted. In order to realize the adaptive receiver, it is required to identify the digital modulation

type of a signal. There have been several efforts to develop a digital modulation identification (DMI) method.

Automatic identification of the digital modulation type of a signal has found applications in many areas, including electronic warfare, surveillance and threat analysis. The application of such an identifier has been the need both in the military as well as the commercial fields. In the military perspectives, its application lies in Electronic Support Measures (ESM), in Electronic Counter Measures (ECM) and in Electronic Counter-Counter Measures (ECCM). Further, in military applications, modulation identification can be employed for electronic surveillance, interference identification and monitoring. In the commercial field, its application is quite evident in the Software Defined Radio (SDR), spectrum management, network traffic administration, different data rate allocation etc.

Hassan Maqbool

ACKNOWLEDGEMENTS

We are indebted and grateful to ALMIGHTY ALLAH who gave us knowledge, motivation and strength to struggle for the completion of our project.

We are really grateful to MAJ IMRAN RASHID for his kind supervision for the project and guiding us as and when needed.

We also thank MAJ Dr. SHOAIB AHMAD KHAN for guiding us to handle the project in a sequential manner and helping us through whenever we needed his help. His advice as and when, we needed was available to us and his time for us out of his busy schedule.

We are obliged to MAJ IMTIAZ KHOKHAR for his kind guidance and direction.

We are also thankful to LEC. IRTIZA ALI for his guidance at the very beginning of the project.

We would also like to acknowledge the efforts of Mr. MUHAMMAD ALI CHAUDARY for his kind assistance in MATLAB programming.

We are thankful to our parents, families and teachers, whose motivation, guidance and prayers have made us to reach this point of completion of our project.

We are really grateful to our course mates and colleagues who have motivated us through to reach to the completion of the project. They may include AAMIR JAVED, RASHID IQBAL, YASIR BILAL, UMAIR KALIMI, HAMMAD JAVED and GHEESA ISRAR.

ABSTRACT

We, in our project, has developed and implemented a new algorithm for modulation identification in ideal channel conditions. This algorithm may be classified in the statistical method category. It has been totally our thinking and work that has led us to develop such an algorithm that is based truly on the modulating parameters. Hence, it could be said that any modulation technique can be identified as long as the modulating parameters can be extracted from the received signal. Our algorithm ha been termed as *'Histogram-Count Modulation Identification'* (HCMI) method. In the algorithm, we have derived the modulating parameters as the identification parameters. First we plot the histograms of the respective modulating parameters, which give us the number of these parameters. Basing upon which, we identify the modulation technique.

The implementation of the project has been divided into three stages. These stages are mentioned as follows: -

1. Implementation of the Digital Modulators.
2. Implementation of Digital Modulation Identifier.
3. Implementation of Digital Demodulators.

Our project is Matlab based. Basically the algorithm has been simulated in Matlab. The version used as the platform is Matlab 7. Our secondary goal is the implementation of the project on a DSP TMS320C6000. But the secondary goal is in view to the time line available for the project. Certainly due to non-availability of the card and the time we have not been able to achieve our secondary goal but we do not regret our achievement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

INTRODUCTION

This chapter covers the introductory part of the project. It introduces a few basic concepts and ideas, which forecasts the advantages of the project. On the completion of this chapter, the reader will be acquainting himself with the following ideas and concepts: -

1.1 Introduction to the Project.
1.2 Why Modulate.
1.3 Analog vs. Digital.
1.4 Fourier Transforms.
1.5 Phase Locked Loop.
1.6 Modulation Identification.
1.7 Applications of project modules.
1.8 Summary.

1.1 INTRODUCTION TO THE PROJECT

Space belongs to everyone and anything once transmitted through it knows no boundaries and could be interrupted, intercepted and decoded for information extraction by any third party. No one can ever wonder how powerful and efficient the third party would be.

Our project could be considered as the fundamental module of any communication system, with its multi-dimensional application in Electronic Support Measures (ESM), Electronic Counter Measures (ECM) and Electronic Counter-Counter Measures (ECCM).

In terms of ESM, a '*Modulation Hoping Transceiver System' (MHTS)* [1] would enable the forces to communicate in a more secure and swift environment, with a comparatively decreased probability of interruption, interception, interpretation and detection. The transmitter transmits the data or voice over the channel by changing the digital modulation technique after every few instances. Hence the third party is always in state of chaos, as how to know the modulation technique that is being used at that instant. Hence the third party either fails or works too hard to extract the original message or data out of it.

In terms of ECM and ECCM, the project module works on stand alone basis, meaning thereby the module is able to identify the modulation technique, the third party may be working on. Consequent upon which, the identified modulation's demodulator attached to it would retrieve the data out of the intercepted signal. But here one thing is to be made clear that the data retrieved might be encrypted using some algorithms and keys.

---

1. A new concept for secure communication been introduced by us.

Hence if encryption has been done on the data then our module would not be able to decrypt but would certainly be able to demodulate the encrypted data. In order to decrypt it, a decrypting module may be required. This is something beyond the scope of our project and hence be considered so.

Our main stress is on the modulation identification, neither on the demodulation nor on the decryption. So we have developed an algorithm altogether by ourselves for the modulation identification. This algorithm has been tested upon 11 modulation techniques, including the CDMA 2000 (QPSK), IS95 (BPSK) and the other communication based on different modulation techniques. The results for identification have been quite extra ordinary with a success rate of almost cent percent. The 11 modulation techniques are as follows: -

1. Binary Amplitude Shift Keying (BASK).
2. Quadrature Amplitude Shift Keying (QASK).
3. 8ary Amplitude Shift Keying (ASK8).
4. Binary Phase Shift Keying (BPSK).
5. Quadrature Phase Shift Keying (QPSK).
6. 8ary Phase Shift Keying (PSK8).
7. Quadrature Amplitude Modulation-4 (QAM-4).
8. Quadrature Amplitude Modulation-8 (QAM-8).
9. Quadrature Amplitude Modulation-16 (QAM-16).
10. Quadrature Amplitude Modulation-32 (QAM-32).
11. Quadrature Amplitude Modulation-64 (QAM-64).

Successfully we have been able to implement the demodulators too. Hence, a complete 'modulation-identification-demodulation' system or in short UMID [1], as we call it, system has been implemented, as shown in the block diagram 1.a.

The source may be data (text), image, audio signal or video signal. The signal from the source, whichever it may be, has to be converted to binary bits (digitized) as is the need for its transmission. The sink on the other hand could also be of one of the four types mentioned provided we know what was transmitted and how to convert it back to analog form.

---

1. UMID – Universal Modulation Identifier and Demodulator

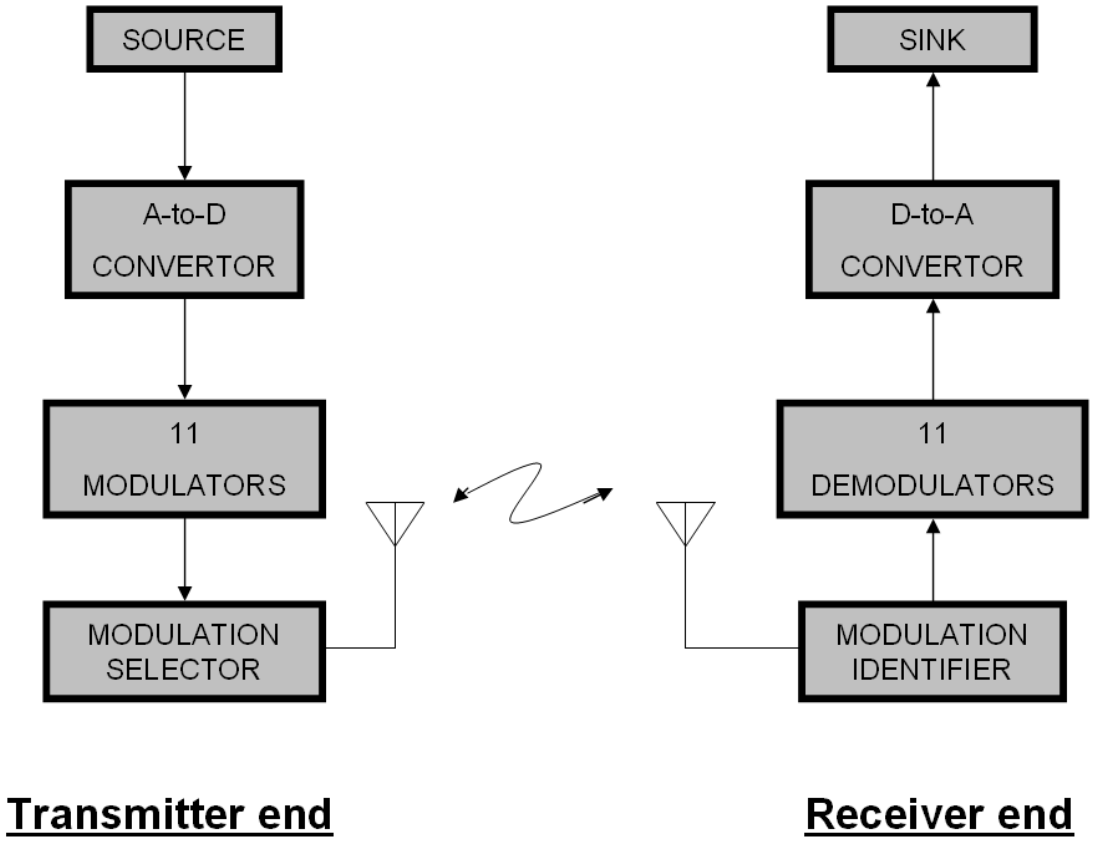

Figure 1.a. Block Diagram of UMID

1.2 **WHY MODULATE?** [1]

The technique of superimposing the message signal on the carrier is known as modulation. The process of shifting of spectrum of the signal from a lower frequency to a higher frequency is called modulation and the reverse process is known as demodulation. Modulation is the process by which a property or parameter of one signal (in this case the carrier) is varied in proportion to the second signal (in this case the message signal). Modulation techniques can be broadly classified as follows: -

1. Digital versus Analog Modulation
2. Baseband versus Bandpass (Passband) Modulation
3. Binary versus M-ary Modulation
4. Memory-less Modulation versus Modulation with memory
5. Linear versus Nonlinear Modulation
6. Constant envelope versus Non-constant envelope Modulation
7. Power efficient versus Bandwidth efficient Modulation

Basing upon the ways of transmission and the waveform types, the modulations can be classified into following two categories: -

1. Baseband Modulation.
2. Passband Modulation.

Baseband Modulation has pulses as the waveforms, but in case of passband modulation a carrier waveform is a sinusoid.

Digital modulation is a process by which digital symbols are transformed into waveforms that are compatible with the characteristics of the channel. In the case of baseband modulation, these waveforms are pulses, but in the case of bandpass modulation the desired information signal modulates a sinusoid called a carrier wave, or simply a carrier. For radio transmission the carrier is converted to Electromagnetic (EM) field for propagation to the desired destination.

Now the question arises, why it is necessary to use a carrier for the radio transmission of baseband signals. This is because the transmission of EM fields through the space is accomplished with the use of antennas. In order to efficiently couple the transmitted EM energy into space, the dimensions of the antenna aperture is to be as large as the wavelength being transmitted. Wavelength, λ, is equal to c / f, where c is the speed of light and f is the frequency of the wave being transmitted. For baseband signal with frequency $f = 3000\ Hz$, $\lambda = 1,00,000\ m$, antenna length $(l) \approx 60\ miles$. To efficiently transmit a 3000 Hz signal through space without carrier wave modulation, an antenna that spans at least 60 miles would be required. Even if we are willing to inefficiently transmit the EM energy with an antenna measuring $1/10^{th}$ of a wavelength, we are faced with an impossible antenna size. However, if the information to be transmitted is first modulated on a higher frequency carrier, for example 30 GHz carrier, the equivalent antenna diameter is then less than ½ in. For this reason, carrier wave or bandpass modulation is an essential step for all systems involving radio transmission. But certainly a higher frequency carrier may not be needed in case of baseband modulation, where the channel is usually wired or the distance is comparatively too small.

Bandpass modulation can provide other important benefits in signal transmission. If more than one signal utilizes a single channel, modulation may be used to separate the different signals. Such a technique is known as FDM [1], TDM [2]. Modulation can be used to minimize the effects of interference. A class of such modulation schemes, known as *Spread Spectrum Modulation*, requires a system band width that would be required by the message. Modulation can also be used to place a signal is frequency band where design requirements, such as

---

1. FDM – Frequency Division Multiplexing
2. TDM – Time Division Multiplexing

amplification and filtering, can easily be met. This is the case when RF (Radio Frequency) signals are converted to an IF (Intermediate Frequency) at the receiver.

Effects of Inter-symbol Interference

Detection may be treated as the detection of signals in the presence of AWGN under the assumption that there is no inter-symbol interference (ISI). Then the analysis would be straightforward, since the zero-mean AWGN process is characterized by its variance alone. In practice we find that ISI is often a second source of interference which must be accounted for. ISI can be generated by the use of band-limiting filters at the transmitter output, in the channel, or at the receiver input. The result of this additional interference is to degrade the error probabilities for coherent as well as for non coherent reception. Analysis involving ISI in addition to AWGN is much more complicated since it involves the impulse response of the channel.

1.3 ANALOG vs. DIGITAL

Analog signal is continuous both along the time and the amplitude axis. Discrete signal is continuous on amplitude axis but discrete along the time axis. Digital signal, however, is discrete both along the amplitude and time axis. Respectively, x(t), x(kT) and x[n] represent the mentioned signals.

More elaborately, an analog signal is continuous function of time; that is, x(t) is uniquely defined for all values of t. An electrical analog signal arises when a physical waveform is converted into an electrical signal by means of a transducer. By comparison, a discrete signal is one that exists only at discrete times; it is characterized by a sequence of numbers defined for each time, kT, where k is and integer and T is a fixed time interval or the time for which the signal is defined. On the contrary, x[n] would exist at discrete times and defined by discrete magnitudes only, where n is the step.

1.3.1 Analog-to-Digital Conversion [2]

a. Sampling

The conversion of an analog signal to a digital one involves a number of stages and processes. The primary step being the *sampling* of the analog signal, to convert analog signal to a discrete signal at first instance. The sampling process must satisfy the Nyquist's Criterion i.e. the sampling frequency must at least be twice the maximum frequency in the signal. Rather to be more precise it must satisfy the Engineer's version of Nyquist's Criterion i.e. 2.2 times the maximum frequency in the signal. This criterion

helps in avoiding aliasing [1]. The process of sampling is shown in figure 1.b.

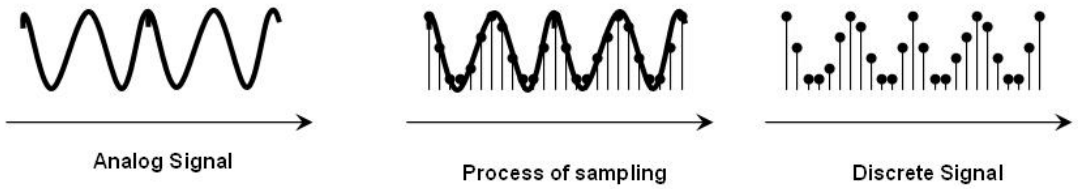

Figure 1.b. Process of sampling

b. **Quantization**

After the process of sampling has occurred, we have the discrete signal. The next step is the conversion of discrete signal to digital signal. This process is known as quantization. In quantization, each sample's amplitude is approximated or rounded off to the nearest quantized level. Consider that a signal be m(t) has amplitudes in the range ($-m_p$, m_p), which is partitioned into L levels, where L is known as the quantization level, each of magnitude $\Delta v = 2m_p / L$. Each sample magnitude is approximated to the midpoint of the interval in which the sample value falls. Each sample is now approximated to one of the L numbers. The information is thus digitized.

The quantized signal is an approximation of the original signal. The accuracy of quantized signal can be improved upon to any desired degree by increasing the number of levels L. For intelligibility of voice signals, for example L = 8 or L = 16 is sufficient. For commercial use L = 32 is a minimum, and for telephone communication, L = 128 or 256 is commonly used. The process of uniform quantization is illustrated in figure 1.c.

---

1. Aliasing is the process of overlapping of the samples at the receiver while detection / reconstruction take place.

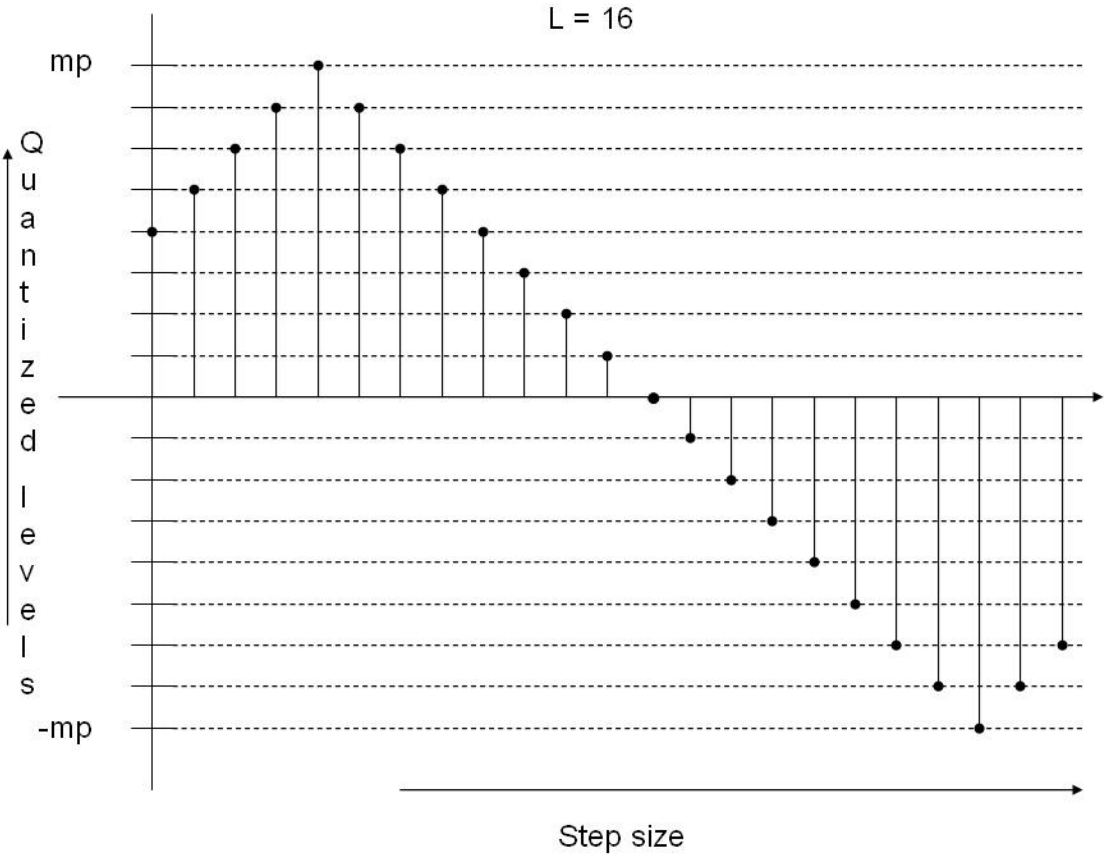

Figure 1.c. Process of uniform quantization

During each sampling interval, one quantized sample is transmitted, which has taken on one of the quantized levels. Each of the L levels is encoded to binary bits. This is known as *'binary encoding of data'*. The number of binary bits that each level requires to be encoded depends on the total number of quantized levels. For example if L = 16, we would need only, 2^4 = 16, i.e. 4 bits and if we increase the levels to 32, we would need, 2^5 = 32, i.e. 5 bits. Hence if the number of levels is increased, the number of encoding bits will also increase, resulting in large data to be transmitted, but good communication quality.

c. **Non-uniform Quantization** [3]

Uniform quantizers are the most common type of ADCs because they are the most robust. By '*robust*' we mean that they are relatively insensitive to small changes in the input statistics. They achieve this robustness by not being finely tuned to one specific set of input parameters. This allows them to perform well even in the face of uncertain input parameters, and it means that small changes in input statistics will results in only small changes in output statistics.

When there is small uncertainty in the input signal statistics, it is possible to design a non-uniform quantizer which exhibits a smaller SNR than a uniform quantizer using the same number of bits. This is accomplished by partitioning the input dynamic range into non-uniform intervals such that the noise power, weighted by the probability of occurrence in each interval, is the same. Iterative solutions for the decision boundaries and step sizes for an optimal quantizer can be found for specific density functions and for a small number of bits. This task is simplified by modeling the non-uniform quantizer as a sequence operators, as depicted in figure 1.d, below.

The input levels are first mapped, via a non-linear function called a *compressor*, to an alternative range of levels. These levels are uniformly quantized and the quantized signal levels are then mapped, via a complementary non-linear function called an *expander*, to the output range of levels. Borrowing part of the names from each of the operations COMpress and expand, we have the acronym by which this process is commonly identified as *companding.*

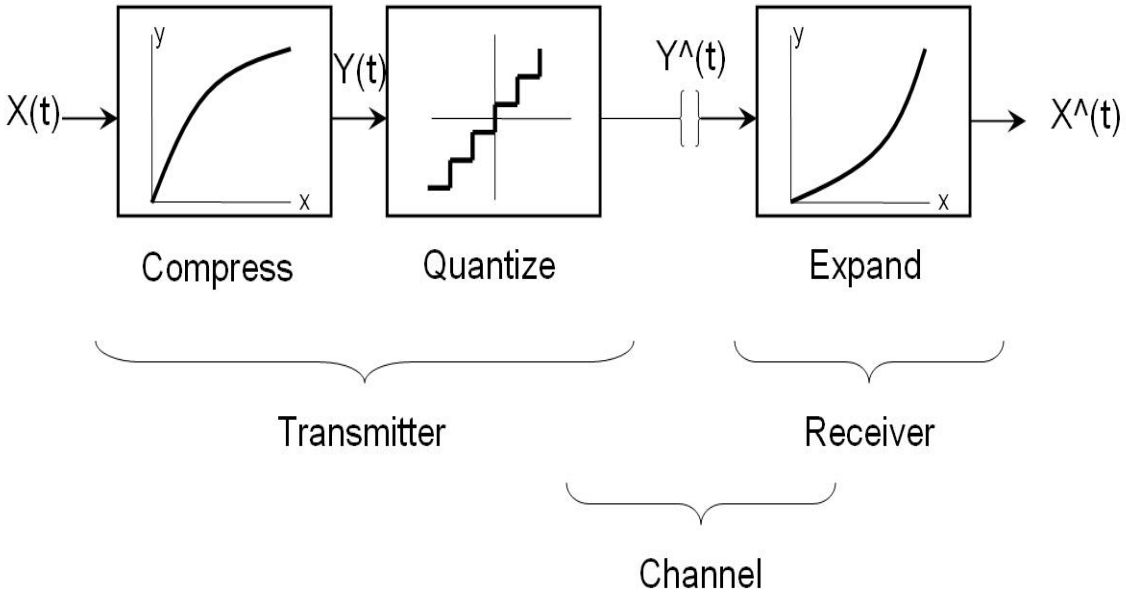

Figure 1.e. Non-uniform quantization as a sequence of compression, uniform quantization and expansion.

The approximation or rounding off during quantization creates an error, when decoded at the receiver during a process known as *'binary decoding of data'*. This error is known as the

quantization error. This can however be reduced by increasing the quantization levels. Consequently, increased number of encoding bits. However this results is good communication quality.

1.3.2 Advantages of Digital over Analog

a. Noise Immunity of Digital Signals

Digital message are transmitted by using a finite set of electrical waveforms. For example in the mores code, a mark can be transmitted by an electrical pulse of amplitude A/2, and a space can be transmitted by a pulse of amplitude –A/2. In an M-ary case, M-distinct electrical pulses (or waveforms) reused; each of the M pulses represents one of the M possible symbols. The task of the receiver is to extract a message from a distorted or noisy signal at the channel output. Message extraction is often easier from digital signals than from an analog signal. Consider a binary case; two symbols are encoded as rectangular pulses of amplitude A/2 and -A/2. The only decision at the receiver is the selection between two possible pulses received, not the details of the pulse shape. The decision is readily made with reasonable certainty if the pulses are noisy or distorted. This has been illustrated in figure 1.e.

The digital message in the figure is distorted by the channel as shown in figure yet if the distortion is with in limits we can recover the data without error because we need only to make a simple binary decision as to whether the received pulse is positive or negative. Figure shows the same data with channel distortion and noise. Here again, the data can be recovered correctly as long as the distortion and the noise are with in limits. In contrast, the waveform in an analog message is important, and even a slight distortion or interference in the waveform will cause an error in the received signal. Clearly, a digital communication system is more rugged then an analog communication system in the sense that it can better take care of the possible errors that may occur due to noise and distortion (as long as they are with in limits).

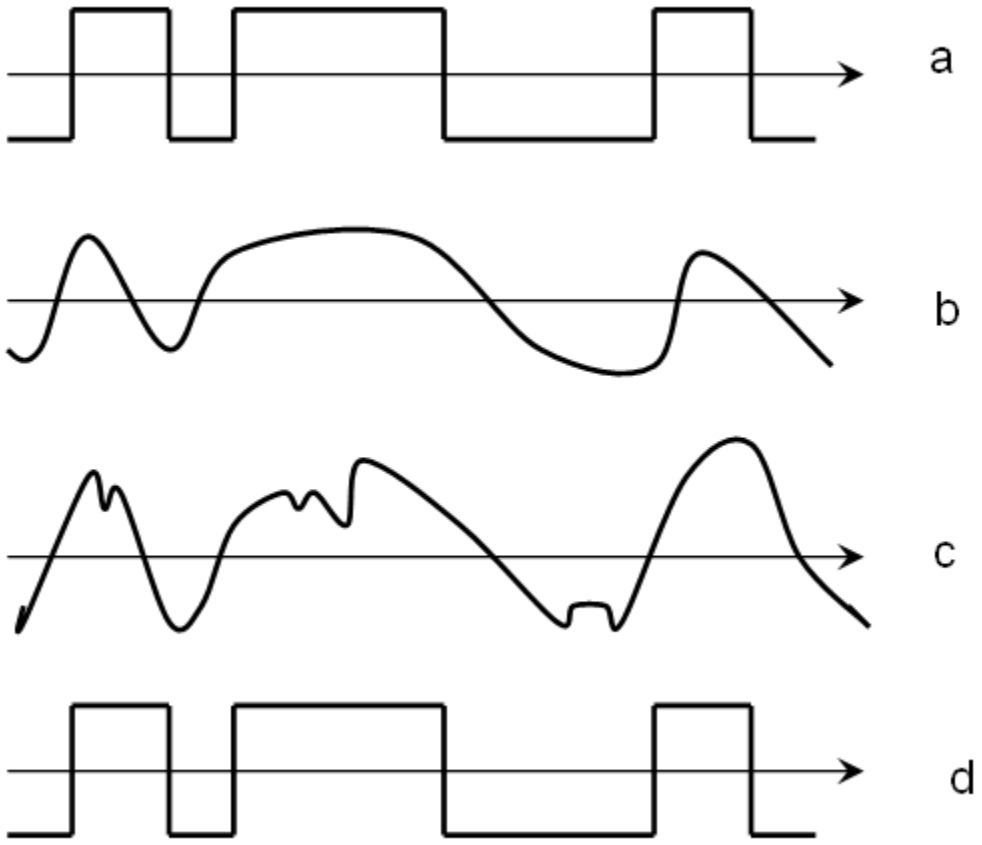

Figure 1.e. (a) transmitted signal (b) received distorted signal (without noise) (c) received distorted signal with noise (d) regenerated signal (delayed)

b. **Viability of Regenerative Repeaters in Digital Communication**

The main reason for the superiority of the digital system over the analog ones is the viability of regenerative repeaters in the former. Repeater stations are placed along the communication path of a digital system at distances short enough to ensure that noise and distortion remains within a limit. This allows pulse direction with high accuracy. At each repeater station the incoming pulses are detected and new clean pulses are transmitted to next repeater station. This process prevents the accumulation of noise and distortion along the path by cleaning the pulses periodically at the repeater station we can thus transmit messages over longer distances with greater accuracy.

For analog system, there is no way to avoid accumulation of noise and distortion along the path. As a result, the distortion and

the noise interference are accumulative over the entire transmission to compound the difficulty the signal is attenuated continuously over the transmission path thus with increasing distance the signal becomes weaker, where as distortion and the noise becomes stronger. Ultimately, the signal overwhelm by the distortion and noise, is mutilated. Amplification is of little help, because it enhances the signal and the noise in the same proportion i.e. SNR remains constant. Consequently, the distance over which the analog message can be transmitted is limited by the transmitter power. Despite these problems, analog communication was used widely and successfully in the past. Because of the advent of optical fiber and the dramatic cost reduction achieved in the fabrication of digital circuitry, almost all new communication system being installed are digital.

c. The two types (analog and digital signals) of communication are difficult to combine over the same medium in analog domain. Using digital techniques, it is possible to combine both formats for transmission through a common medium.

d. Packet Switching is possible in digital communication systems whereas Circuit Switching takes place in analog communication systems.

e. Encryption and other privacy techniques are easy to implement in digital communication systems.

f. Digital Communication Systems are inherently more efficient than analog in realizing the exchange of SNR for bandwidth.

g. Digital signals can be decoded to yield extremely of rates high fidelity as well as privacy.

h. **Two-state signal representation.**
The input to a digital system is in the form a sequence of bits (binary or M-ary).

i. **Hardware is more flexible.**
Digital hardware implementation is flexible and permits the use of microprocessors, mini-processors, digital switching and VLSI. Also it has shorter design and production cycle.

j. **Low Cost.**
The use of VLSI and LSI in the design of components and systems have resulted in lower cost.

k. **Easier and more efficient to multiplex several digital signals.**

Digital multiplexing techniques like Time Division Multiple Access (TDMA) and Code Division Multiple Access (CDMA) are easier to implement than analog techniques such as Frequency Division Multiple Access (FDMA).

1.4 FOURIER TRANSFORM [4]

Fourier Transform is a method or transformation that is helpful in order to know the frequency components in a signal.

This is an extension to Fourier Series representation of any signal. This is to the fact that if the time period of any signal goes to infinity, the Fourier Series representation converges to Fourier Transforms.

Consider a general sequence x[n] that is of finite duration, as shown in equation 1.1 below.

$$x[n]=0 \quad \begin{cases} n<N1 \\ n>N2 \end{cases}$$

…….. EQ 1.1

This signal is shown in figure 1.f. [3] from this aperiodic signal, we can construct a periodic sequence x~[n] for which x[n] is one period, as illustrated in figure 1.g. As we choose the period N to be large, x~[n] is identical to x[n] over a longer interval, and as N → ∞, x~[n] = x[n] for any finite value of n.

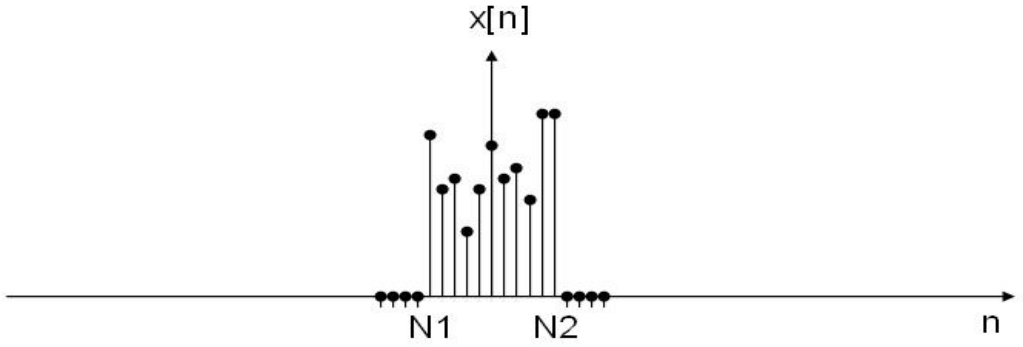

Figure 1.f. x[n]

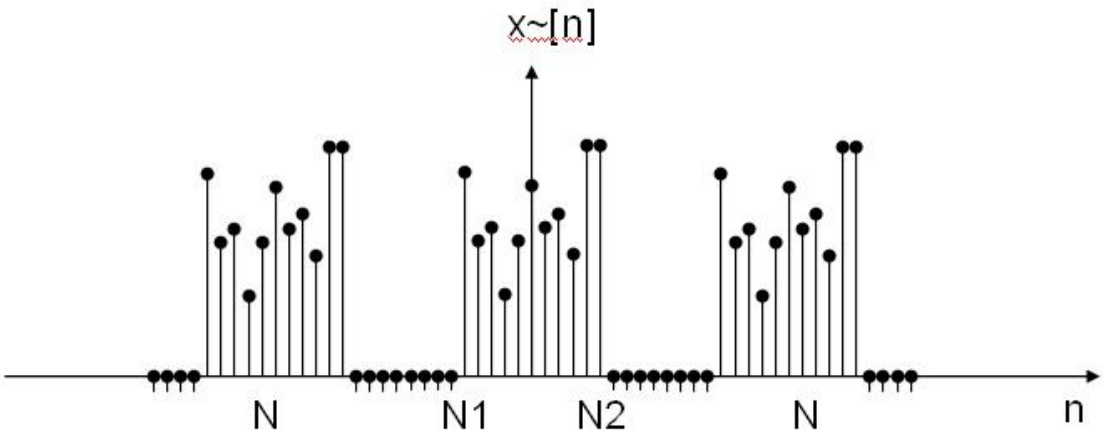

Figure 1.g. x~[n]

Let us now examine the Fourier Series representation of the signal x~[n]. Specifically, the analysis and the synthesis equations for the Fourier Series representation of the signal. These equations are equation 1.2 and 1.3, below.

$$\tilde{x}[n] = \sum_{K=<N>} a_K \, e^{jk(2\pi/N)\,n}$$

…… EQ 1.2

$$a_K = 1/N \sum_{K=<N>} \tilde{x}[n] \, e^{-jk(2\pi/N)\,n}$$

…… EQ 1.3

Since x[n] = x~[n] over a period that includes the interval N1 <= n <=N2 it is convenient to choose the interval of summation in equation 1.3 to include this interval, so that x~[n] can be replaced by x[n] in summation. Therefore,

$$a_k = 1/N \sum_{n=N_1}^{N_2} \tilde{x}[n] \, e^{-jk(2\pi/N)\,n} = 1/N \sum_{n=-\infty}^{\infty} \tilde{x}[n] \, e^{-jk(2\pi/N)\,n}$$

…… EQ 1.4

Where in the second equality in equation 1.4, we have used the fact that x[n] is zero outside the interval N1 <= n<= N2. Defining the function,

$$X(e^{jw}) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-jwn}$$

…… EQ 1.5

We see that the coefficients a_K are proportional to the samples of $X(e^{jw})$, i.e.,

$$a_K = 1/N\, X(e^{jw_0})$$

……... EQ 1.6

Where $w_0 = 2\pi /N$ is the spacing of the samples in the frequency domain, combining equations 1.2 and 1.6 yields,

$$\tilde{x}[n] = \sum_{K=<N>} 1/N\, X(e^{jw})\, e^{jkw_0 n}$$

……. EQ 1.7

Since $w_0 = 2\pi /N$, or equivalently, $1/N = w_0 / 2\pi$, equation 1.7 can be rewritten as,

$$\tilde{x}[n] = 1/2\pi \sum_{K=<N>} X(e^{jkw})\, e^{jkw_0 n}\, w_0$$

…….. EQ 1.8

As N increases w_0 decreases, and as $N \rightarrow \infty$ equation 1.8 passes to an integral. To see this more clearly, consider $X(e^{jw})\, e^{jwn}$ as sketched in figure 1.g. From equation 1.5, $X(e^{jw})$ is seen to be periodic in *w* with a period 2π, and so is e^{jwn}. Thus, the product $X(e^{jw})\, e^{jwn}$ will also be periodic. As depicted in the figure, each term in the summation in equation 1.8 represents the area of rectangle of height $X(e^{jwo})\, e^{jwon}$ and width w_0. As $w_0 \rightarrow 0$, the summation becomes an integral. Furthermore, since the summation is carried out over N consecutive intervals of width $w_0 = 2\pi /N$, the total interval of integration will always be 2π. Therefore, as $N \rightarrow \infty$, $\tilde{x}[n] = x[n]$, and equation 1.8 becomes,

$$x[n] = 1/2\pi \int_{2\pi} X(e^{jw_o})\, e^{jw_o n}\, dw$$

…… EQ 1.9

Where $X(e^{jwo})\ e^{jwon}$ is periodic with period 2 π, the interval of integration can be taken as any length 2 π. Thus, we have the following pairs of equations, 1.10 and 1.11. The function $X(e^{jw})$ is referred to as the *discrete-time Fourier transform* and the pair of equations as the *discrete-time Fourier transform pair.* Equation 1.10 is the synthesis equation and 1.11 is the analysis equation.

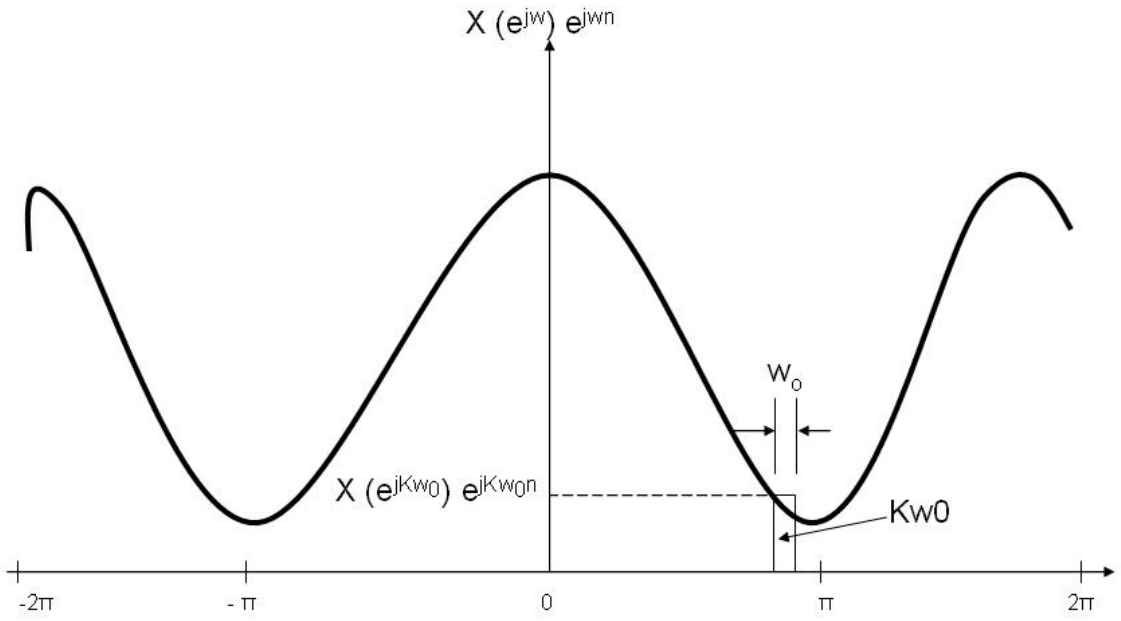

Figure 1.h. Graphical interpretation of equation 1.8

The derivation indicates how an aperiodic sequence can be thought of as a linear combination of complex exponentials. In particular the synthesis equation is infact a representation of x[n] as a linear combination of complex exponentials infinitesimally close in frequency and with amplitudes $X(e^{jw})(dw/2\pi)$. For this reason, as in continuous time, the Fourier transform $X(e^{jw})$ will often be referred to as the spectrum of x[n], because it provides the information an how x[n] is composed of complex exponentials of different frequencies.

$$x[n] = 1/2\pi \int_{2\pi} X(e^{jw})\, e^{jwn}\, dw$$

…… EQ 1.10

$$X(e^{jw}) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-jwn}$$

…… EQ 1.11

1.5 **PHASE LOCKED LOOP** [5]

PLL is short for phase-locked loop, an electronic circuit that controls an oscillator so that it maintains a constant phase angle (i.e., lock) on the frequency of an input, or reference, signal. A PLL ensures that a communication signal is locked on a specific frequency and can also be used to generate, modulate and demodulate a signal and divide a frequency.

PLL is used often in wireless communications where the oscillator is usually at the receiver and the input signal is extracted from the signal received from the remote transmitter.

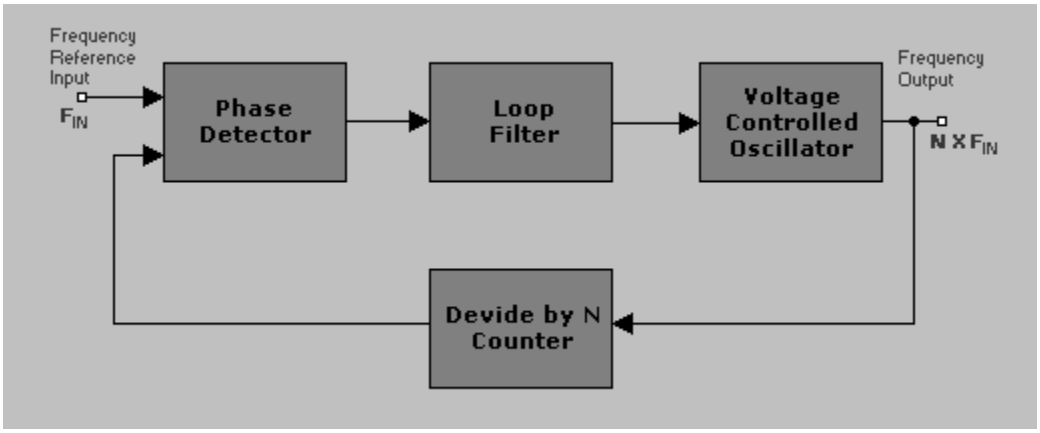

Figure 1.i. Block Diagram of PLL

PLL stands for '*Phase-Locked Loop*' and is basically a closed loop frequency control system, which functioning is based on the phase sensitive detection of phase difference between the input and output signals of the controlled oscillator (CO). First a little bit of history of the Phase-Locked Loop and prior to that with the super-heterodyne. In the early 1930's, the super-heterodyne receiver was king. Edwin Howard Armstrong is widely regarded as one of the foremost contributors to the field of radio-electronics. Among his principal contributions were regenerative feedback circuits, the super-heterodyne radio receiver, and a frequency-modulation radio broadcasting system. It superseded the tuned radio frequency receiver TRF also invented by Armstrong in 1918. He was inducted into the National Inventors Hall of Fame in 1980. Armstrong was born on December 18, 1890, in New York City, where he was to spend much of his professional career. He graduated with a degree in electrical engineering from Columbia University in 1913, and observed the phenomenon of regenerative feedback in vacuum-tube circuits while still an undergraduate. At Columbia, he came under the influence of the legendary professor-inventor.

Michael I. Pupin, who served as a role model for Armstrong and became an effective promoter of the young inventor. In 1915 Armstrong presented an influential paper on regenerative amplifiers and oscillators to the IRE.

Subsequently, regenerative feedback was incorporated into a comprehensive engineering science developed by Harold Black, Harry Nyquist, Hendrik Bode, and others in the period between 1915 and 1940. Armstrong conceived the super-heterodyne radio receiver principle in 1918, while serving in the Army Signal Corps in France. He played a key role in the commercialization of the invention during the early 1920's. The Radio Corporation of America (RCA) used his super-heterodyne patent to monopolize the market for this type of receiver until 1930. The super-heterodyne eventually extended its domain far beyond commercial broadcast receivers and, for example, proved ideal for microwave radar receivers developed during World War II.

However, because of the number of tuned stages in a super-heterodyne, a simpler method was desired. In 1932, a team of British scientists experimented with a method to surpass the super-heterodyne. This new type of receiver, called the '*homodyne*' and later renamed to '*synchrodyne*', first consisted of a local oscillator, a mixer, and an audio amplifier. When the input signal and the local oscillator were mixed at the same phase and frequency, the output was an exact audio representation of the modulated carrier. Initial tests were encouraging, but the synchronous reception after a period of time became difficult due to the slight drift in frequency of the local oscillator. To counteract this frequency drift, the frequency of the local oscillator was compared with the input by a phase detector so that a correction voltage would be generated and fed back to the local oscillator, thus keeping it on frequency. This technique had worked for electronic servo systems, so why wouldn't it work with oscillators? This type of feedback circuit began the evolution of the Phase-Locked Loop.

As a matter of fact, in 1932 a scientist in France by the name of H.de Bellescise, already wrote a subject on the findings of PLL called '*La Réception Synchrone*', published in *Onde Electrique, volume 11*. I guess he lacked the funding or did not know how to implement his findings. In either case it is my personal belief that the British scientist team developed further on the findings of Bellescise. No problem, good stuff. That's why papers like Bellescise are there for.

Although the synchronous, or homodyne, receiver was superior to the super-heterodyne method, the cost of a phase-locked loop circuit outweighed its advantages. Because of this prohibitive cost the widespread use of this principle did not begin until the development of the monolithic integrated circuit and incorporation of complete phased-lock loop circuits in low-cost IC packages, then things started to happen. In the 1940s, the first widespread use of the phase-locked loop was in the synchronization of the horizontal and vertical sweep oscillators in television receivers to the transmitted sync pulses. Such circuits carried the names '*Synchro-Lock*' and '*Synchro-Guide*'. Since that time, the electronic phase-locked loop principle has been extended to other applications.

For example, radio telemetry data from satellites used narrow-band, phase-locked loop receivers to recover low-level signals in the presence of noise.

Other applications now include AM and FM demodulators, FSK decoders, motor speed controls, Touch-Tone® decoders, light-coupled analog isolators, Robotics, and Radio Control transmitters and receivers. Nowadays our technology driven society would be at a loss without this technique; our cell phones and satellite TV's would be useless, well, actually they would not exist.

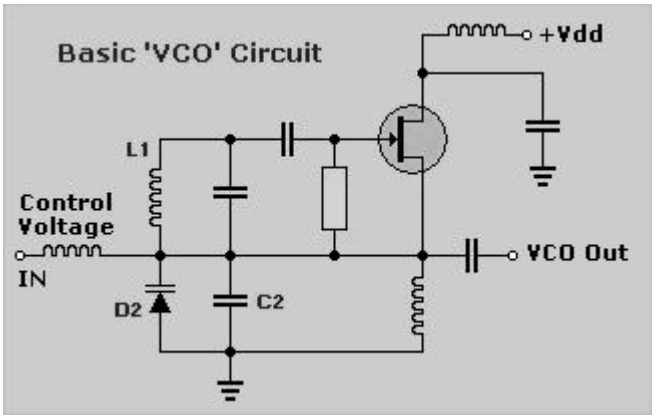

Figure 1.j. Circuit Diagram of Basic PLL

Applications

- Data and Tape Synchronization.
- Modems.
- FSK Modulation.
- FM Demodulation.
- Frequency Synthesizer.
- Tone Decoding.
- Frequency Multiplication and Division.
- SCA Demodulators (‘Hidden’ Radios).
- Telemetry Radios.
- Signal Regeneration.
- Coherent Demodulators.
- Satellite.
- Robotics and Radio Control.

1.6 MODULATION IDENTIFICATION

The process of classifying, identifying or recognizing modulation technique using some method is known as modulation identification.

The need of modulation identification arises from the advancement in Software Defined Radio (SDR). Many modulation techniques and many ways of transmission are used for various purposes of communication. TDM, FDM and CDMA [1] are different ways of transmission using separate modulation techniques.

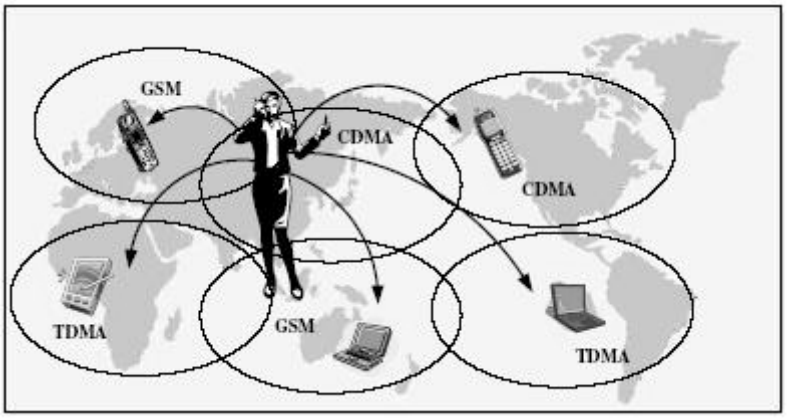

Figure 1.k. Non-restrictive wireless roaming with adaptive receiver.

The modern modulation techniques include the GSM, CDMA 2000 that uses the QPSK and the IS 95 that uses BPSK. The Phase Shift Keying modulations are the most commonly used modulation techniques with special application in satellite communication. They are most immuned to noise with the least probability of error on the detection side.

The receiver receives, demodulates and decodes only those signals, that it recognizes and has the demodulator of. It shall not be able to extract information out of the signal, whose modulation technique it does not recognize.

Therefore, in various commercial and military applications the need for modulation identification arises. Integrating different receivers, working on different modulation techniques, needs a module prior to demodulators that shall be able to identify modulation. Many algorithms have been developed by many researchers and developers. A few of the common methods include statistical methods, maximum likelihood methods, power matrices, wavelet analysis etc.

We, in our project, have devised another algorithm by ourselves for the said purpose of identification. Thus has been termed as 'Histogram-Count Modulation Identification' method and may be placed under the category of statistical methods of modulation identification. This is based on calculating the modulating parameters out of the received signal. The modulating parameters thus become the identification parameters. In the second phase of the project, we calculate the values of modulating parameters and feed them to demodulator of the identified modulation technique for demodulation.

The algorithm is explained in detail in the following chapters of the documentation.

---

1. CDMA – Code Division Multiple Access

1.7 APPLICATIONS OF PROJECT MODULE

With its wide applications in the military and commercial sector, the project module can be applied in the following fields: -

- ECCM.
- ECM.
- SDR.
- MHTS.
- Digital Receivers.
- Stenography.

1.8 SUMMARY

Space belongs to everyone and anything once transmitted through it knows no boundaries and could be interrupted, intercepted and decoded for information extraction by any third party. No one can ever wonder how powerful and efficient the third party would be.

The technique of superimposing the message signal on the carrier is known as modulation. The process of shifting of spectrum of the signal from a lower frequency to a higher frequency is called modulation and the reverse process is known as demodulation. Modulation is the process by which a property or parameter of one signal (in this case the carrier) is varied in proportion to the second signal (in this case the message signal).

Analog signal is continuous both along the time and the amplitude axis. Discrete signal is continuous on amplitude axis but discrete along the time axis. Digital signal, however, is discrete both along the amplitude and time axis. Respectively, x(t), x(kT) and x[n] represent the mentioned signals.

Fourier Transform is a method or transformation that is helpful in order to know the frequency components in a signal. This is an extension to Fourier Series representation of any signal. This is to the fact that if the time period of any signal goes to infinity, the Fourier Series representation converges to Fourier Transforms.

PLL is short for phase-locked loop, an electronic circuit that controls an oscillator so that it maintains a constant phase angle (i.e., lock) on the frequency of an input, or reference, signal. A PLL ensures that a communication signal is locked on a specific frequency and can also be used to generate, modulate and demodulate a signal and divide a frequency.

The process of classifying, identifying or recognizing modulation technique using some method is known as modulation identification. The need of

modulation identification arises from the advancement in Software Defined Radio (SDR). Many modulation techniques and many ways of transmission are used for various purposes of communication. TDM, FDM and CDMA are different ways of transmission using separate modulation techniques.

REFERENCES

[1, 3] Bernard Sklar, *Digital Communication Systems.*

[2] B.P.Lathi, *Modern Communication and Analog Communication Systems.*

[4] Openheim, *Signals and Systems.*

[5] *http://www.howstuffworks.com.*

CHAPTER 2

MODULATION TECHNIQUES

INTRODUCTION

The transmission of digital signals is increasing at a rapid rate. Low-frequency analogue signals are often converted to digital format (PAM) before transmission. The source signals are generally referred to as base band signals. Of course, we can send analogue and digital signals directly over a medium. From electro-magnetic theory, for efficient radiation of electrical energy from an antenna it must be at least in the order of magnitude of a wavelength in size; *c = f * lamda*, where c is the velocity of light, f is the signal frequency and lamda is the wavelength. For a 1 kHz audio signal, the wavelength is 300 km. An antenna of this size is not practical for efficient transmission. The low-frequency signal is often frequency-translated to a higher frequency range for efficient transmission. The process is called modulation. The use of a higher frequency range reduces antenna size. In the modulation process, the base band signals constitute the modulating signal and the high-frequency carrier signal is a sinusoidal waveform.

There are three basic ways of modulating a sine wave carrier. For binary digital modulation, they are called binary amplitude-shift keying (BASK), binary frequency-shift keying (BFSK) and binary phase shift keying (BPSK).

This chapter explains the following things: -

2.1 QUADRATURE AMPLITUDE MODULATION

QAM is a combination of ASK and PSKQAM is a modulation process that incorporates the techniques of *digital* (coded) Phase Shift Keying (PSK) and *digital* (quantized) Amplitude Modulation. In addition, QAM can process and transmit the codes of *two* input signals phased 90 apart: the I and Q signals. This is the "quadrature" part of QAM. Two different signals sent simultaneously on the same carrier frequency. The modulation technique can be represented using equation 2.1.

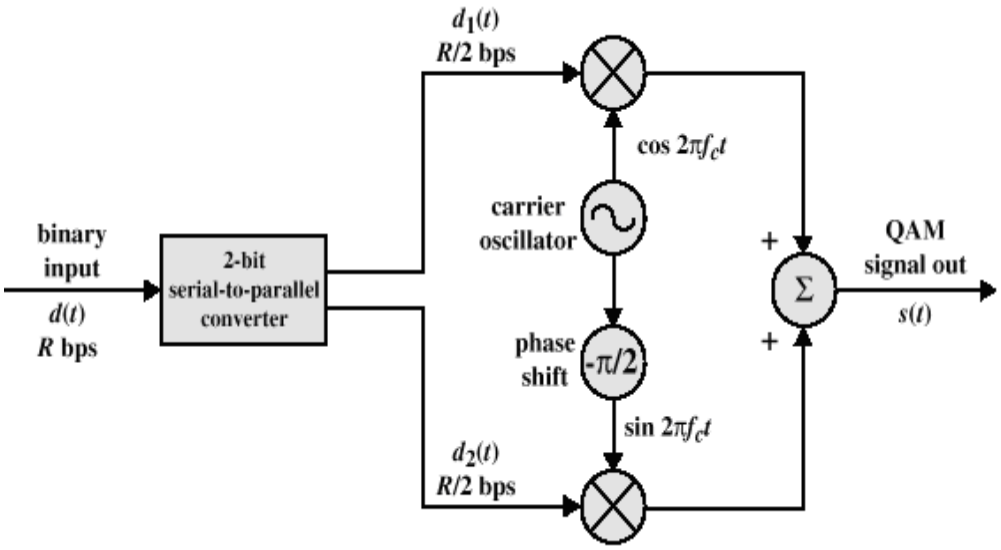

Figure 2.a. A simple QAM MODULATOR

$$s(t) = d_1(t)\cos 2\pi f_c t + d_2(t)\sin 2\pi f_c t$$

……Equation 2.1

2.1.1 **QAM-4**

QAM-4 consists of two amplitudes and two phases. For QAM-4, 2 bits constitue a symbol, resulting in a total of 4 bits The modulator of QAM-4 is shown in figure 2.b, below.

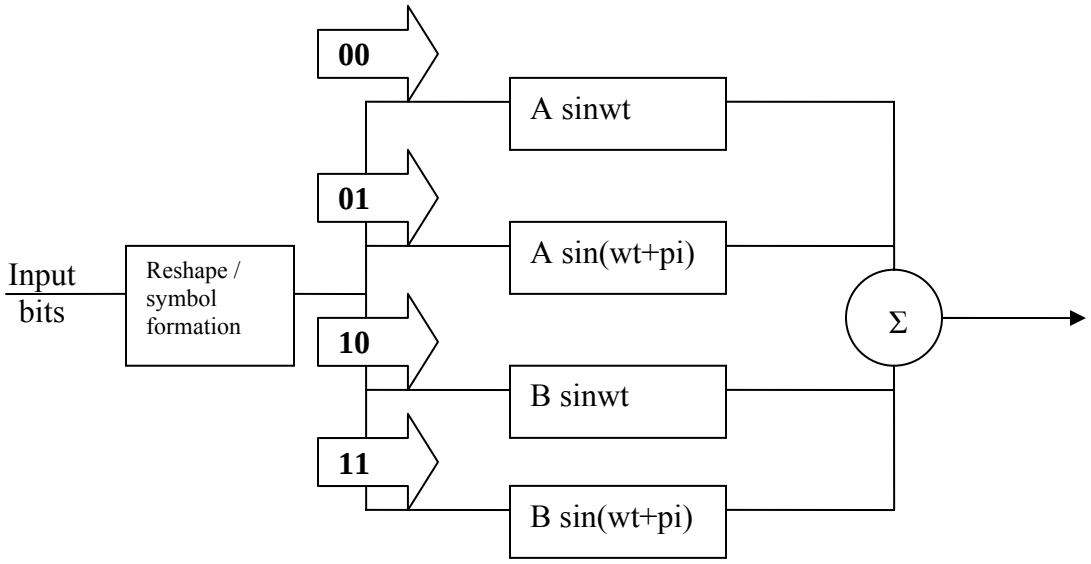

Figure 2.b. a QAM-4 modulator

The demodulator of QAM-4 is shown in figure 2.c, below:

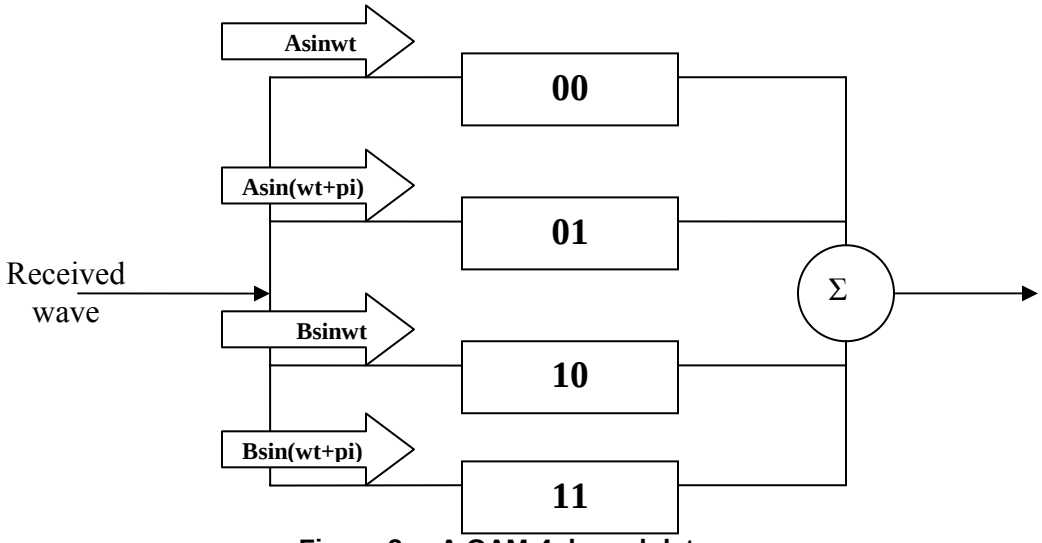

Figure 2.c. A QAM-4 demodulator

The waveform for QAM-4 is shown in figure 2.d, below:

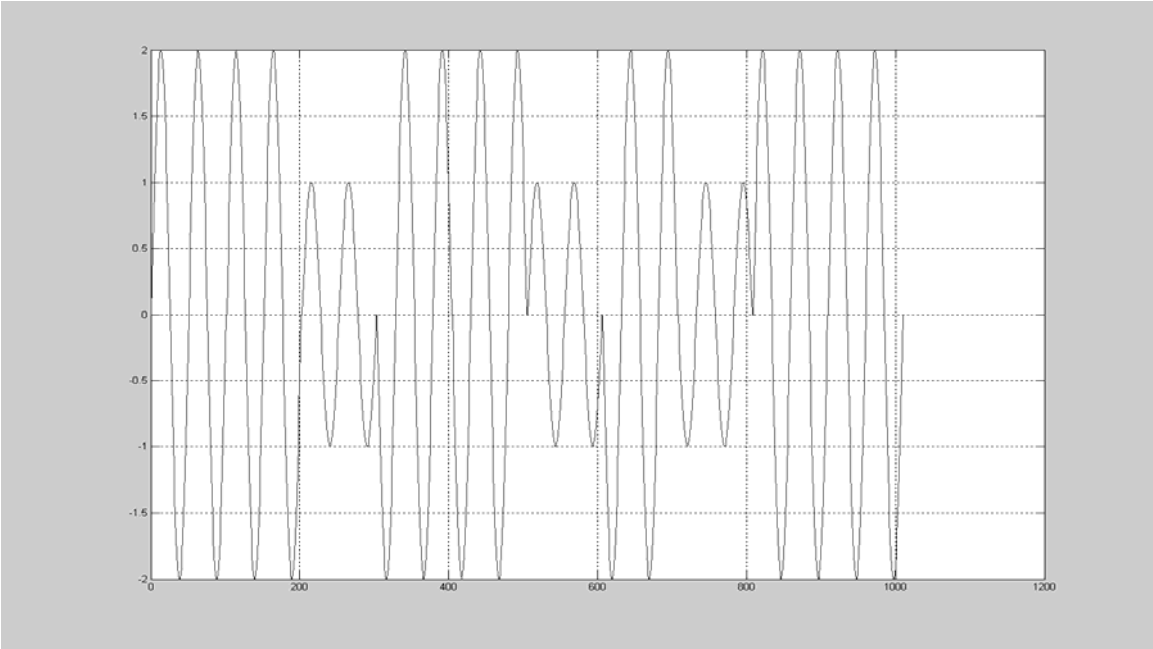

Figure 2.d. transmitted waveform of QAM-4

The vector form of QAM-4 is shown in figure 2.e, below:-

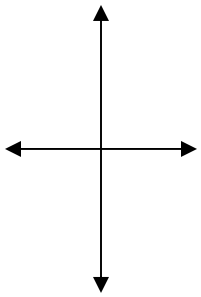

Figure 2.e. vector form of the QAM-4

2.1.2 **QAM-8**

QAM-8 consists of two amplitudes and four phases. The symbols size for QAM-8 modulation technique is 3 bits and they have been shown in table 2.1, below.

Table 2.1. Symbols for QAM 8

S.No.	Symbols
1.	000
2.	001
3.	010
4.	011
5.	100
6.	101
7.	110
8.	111

The waveform for QAM-8 is shown in figure 2.f, below.

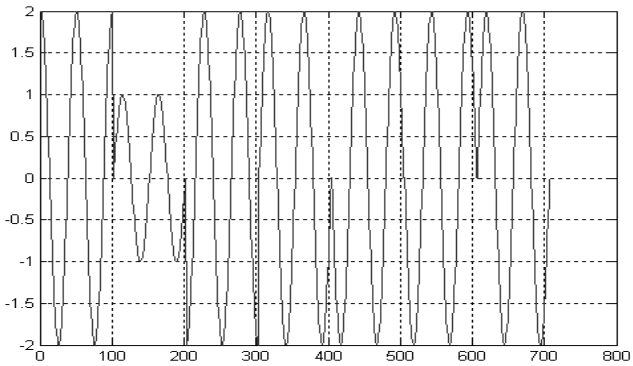

Figure2.f. transmitted waveform of QAM-8

2.1.3 **QAM-16**

In 16-QAM there are 16 possible states, that is, 16 possible combinations of signal amplitude and phase in its constellation. It consists of eight phases and two amplitudes. Its symbol consists of 4 binary bits. The waveform for QAM-16 is shown in figure 2.g, below.

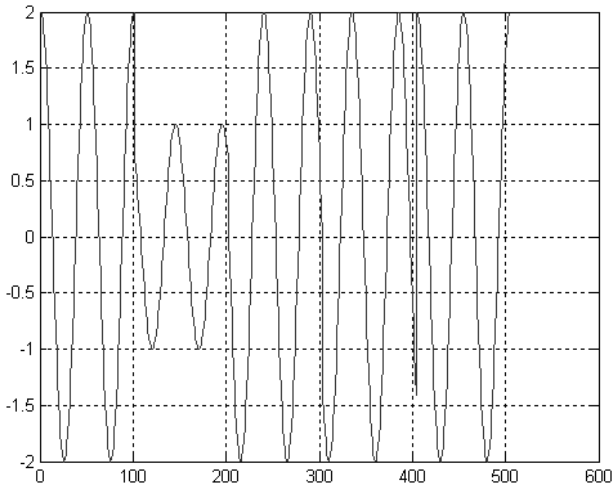

Figure 2.g. transmitted waveform of QAM-16

And the vector space diagram is shown in figure 2.h, below.

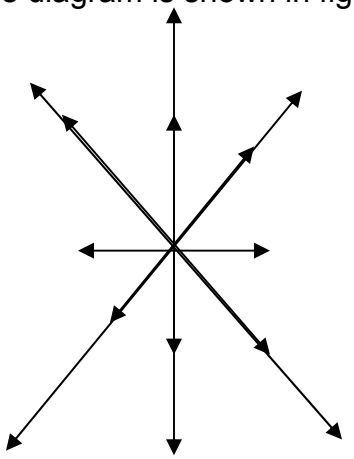

Figure 2.h. Vector space diagram for QAM-16

2.1.4 **QAM-32**

In QAM-32 we have four amplitudes and eight phases. The symbol of QAM-32 consists of 5 binary bits. As the type itself describes it will have 32 constellation points. The waveform of QAM-32 si shown in figure 2.i, below.

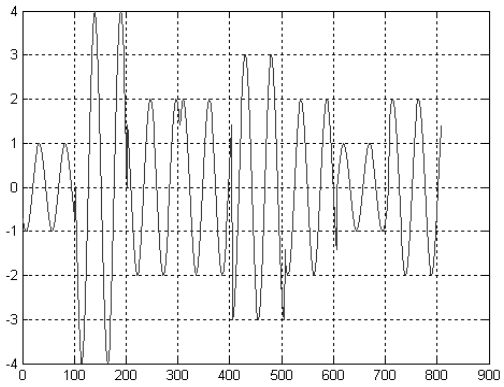

Figure 2.i. Transmitted waveform of QAM-32

2.1.5 **QAM-64**

The waveform for QAM-64 will be :-

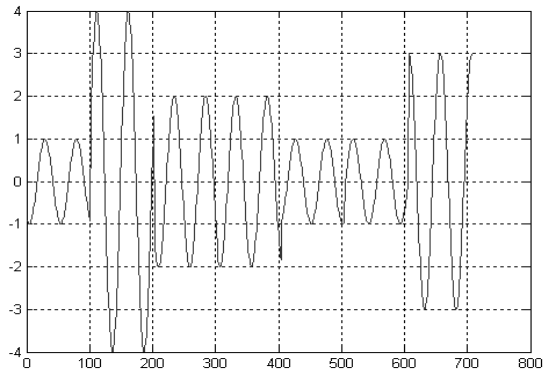

Figure 2.j. transmitted waveform of QAM-64

In QAM-64 we have four amplitudes and sixteen phases. The symbol of QAM-32 consists of 6 binary bits, each. Its constellation consists of 64 points.

2.2 **AMPLITUDE SHIFT KEYING**

In communication when there is need to transmit signal over radio link or satellites link we need to modulate them through different techniques ,because baseband signals have sizable power at low frequencies & cannot be transmitted over radio or steatite link. Hence, for such purpose, the signal spectrum must be shifted to high frequency range. The spectrum of a signal can be shifted to higher frequency by modulating a high frequency sinusoid by the baseband signal. An unmodulated carrier is shown in fig. The on-off baseband signal m(t) is also shown .When the carrier amplitude is varied in proportion to m(t).then we have a

modulated carrier m(t) cos w $_c$(t) as shown in figure . here modulated signal is still in on-off signal. This modulation scheme of transmitting binary data is known as on-off keying (OOK) or Amplitude shift keying(ASK).

Amplitude shift keying (ASK) in the context of digital communications is a modulation process, which imparts to a sinusoid two or more discrete amplitude levels. These are related to the number of levels adopted by the digital message. For a binary message sequence there are two levels, one of which is typically zero. Thus the modulated waveform consists of bursts of a sinusoid.

The BASK signal sequence generated by the binary sequence 1 0 1 1 0 0 1. The amplitude of a carrier is switched or keyed by the binary signal m (t) as shown in fig.2.2.b.This is sometimes called on-off keying (OOK).

A binary amplitude-shift keying (BASK) signal can be defined by

$$S(t) = A*m(t)\cos 2*pi*fc*t,\ 0 < t < T$$

and this carrier is varied in proportion.
Where,

A is a constant
m(t) = 1 or 0
fc is the carrier frequency
T is the bit duration.

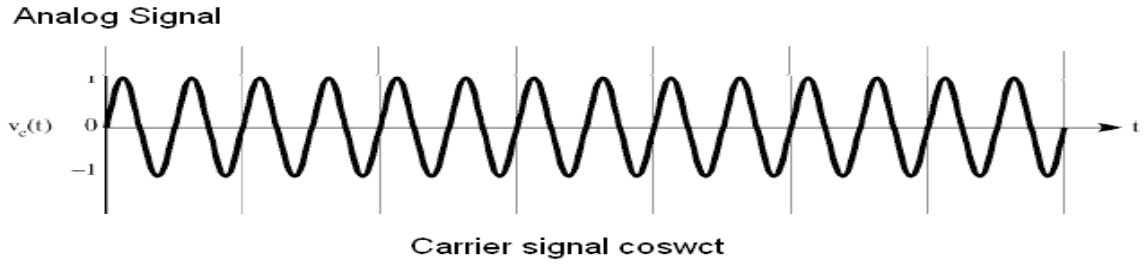

Figure 2.k. Carrier Signal

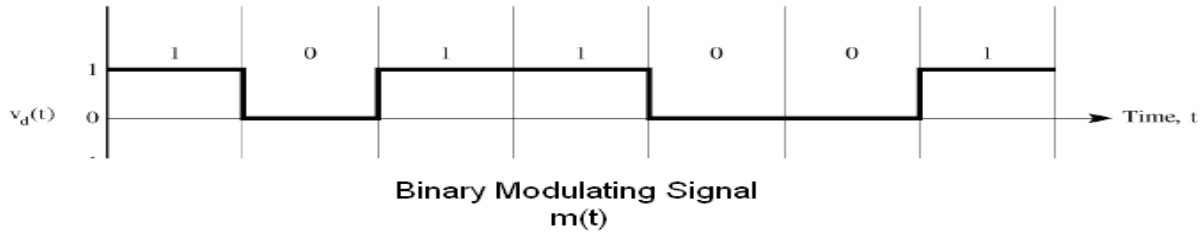

Figure 2.l. Modulating Signal

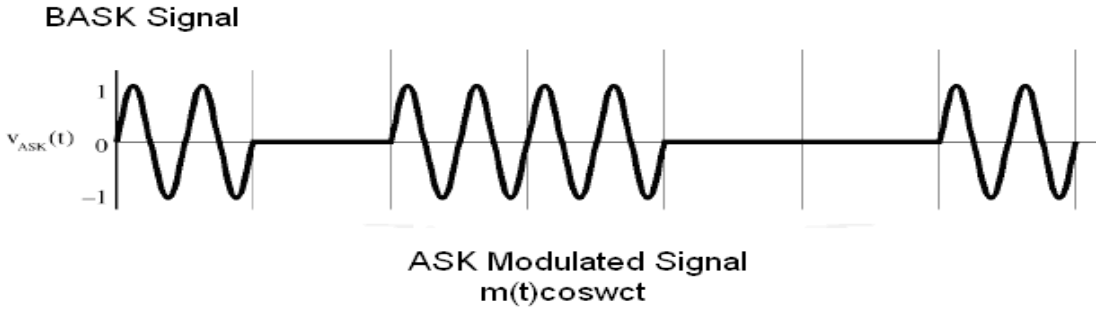

Figure 2.m. Modulated Signal

There are sharp discontinuities shown in fig 2.2.c at the transition points. These result in the signal having an unnecessarily wide bandwidth. Band limiting is generally introduced before transmission, in which case these discontinuities would be 'rounded off'. The band limiting may be applied to the digital message, or the modulated signal itself. The data rate is often made a sub-multiple of the carrier frequency.

The effect of multiplication by the carrier signal A*cos (2*pi*fc*t) is simply to shift the spectrum of the modulating signal m (t) to fc.

The sharp discontinuities in the ASK waveform imply a wide bandwidth. A significant reduction can be accepted before errors at the receiver increase unacceptably. This can be brought about by band limiting (pulse shaping) the message before modulation, or band limiting the ASK signal itself after generation. Having a very definite envelope, an envelope detector can be used as the first step in recovering the original sequence. Further processing can be employed to regenerate the true binary waveform.

ASK signal has a well defined envelope. Thus it is amenable to demodulation by an envelope detector.
Demodulation is a two-stage process:

1. Recovery of the band limited bit stream
2. Regeneration of the binary bit stream

Having a very definite envelope, an envelope detector can be used as the first step in recovering the original sequence. Further processing can be employed to regenerate the true binary waveform.

2.2.1 **ASK-2**

For a binary message sequence there are two levels, one of which is typically zero. The vector diagram for ASK-2 modulation technique is shown in figure 2.n.

Amplitudes	=	2
Normalized amplitude	=	2
Phases	=	1

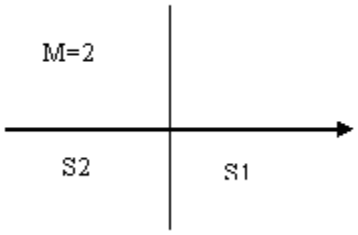

Figure 2.n. Vector Diagram for ASK-2

2.2.2 **ASK-4**

For a binary message sequence there are four levels and each level shows different amplitude. The vector diagram for ASK-4 is shown in figure 2.o, below.

Amplitudes	=	4
Normalized amplitude	=	2
Phases	=	1

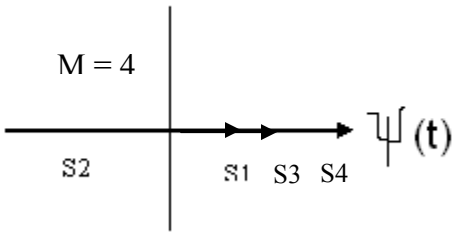

Figure 2.o. Vector Diagram for ASK-4

2.2.3 **ASK-8**

For a binary message sequence there are eight levels each showing different values. The vector diagram for ASK-8 is shown in figure 2.p, below.

Amplitudes	=	8
Normalized amplitude	=	2
Phases	=	1

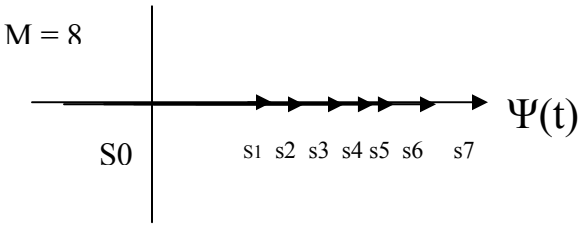

Figure 2.p. Vector Diagram for ASK-8

2.3 PHASE SHIFT KEYING

2.3.1 Phase shift:

The change in phase of a periodic signal with respect to a reference is called phase shift.

2.3.2 Phase modulation:

(PM) is the encoding of information into a carrier wave by variation of its phase in accordance with an input signal. Phase modulation can be regarded as a special case of frequency modulation where the carrier frequency modulation is the time derivative of the PM modulating signal. Phase modulation can also be regarded as a special case of quadrature amplitude modulation.

2.3.3 Phase-shift keying (PSK):

In digital transmission, angle modulation in which the phase of the carrier is discretely varied in relation either to a reference phase or to the phase of the immediately preceding signal element, in accordance with data being transmitted.Phase-shift keying refers to the simple case of phase modulation by a simple signal with a discrete number of states. For example, with only two states, the technique is Binary Phase Shift Keying (BPSK). With four states, it's known as Quadrature Phase Shift Keying (QPSK), with eight states, it's known as 8-PSK, 16 states is 16-PSK, and so on.

The simplest PSK technique is called binary phase-shift keying (BPSK). It uses two opposite signal phases (0 and 180 degrees). The digital signal is broken up time wise into individual bits (binary digits). The state of each bit is determined according to the state of the preceding bit. If the phase of the wave does not change, then the signal state stays the

same (0 or 1). If the phase of the wave changes by 180 degrees -- that is, if the phase reverses -- then the signal state changes (from 0 to 1, or from 1 to 0). Because there are two possible wave phases, BPSK is sometimes called biphase modulation. Phase-shift-keying offers a simple way of increasing the number of levels in the transmission without increasing the bandwidth by introducing smaller phase shifts

More sophisticated forms of PSK exist. In m-ary or multiple phase-shift keying (MPSK), there are more than two phases, usually four (0, +90, -90, and 180 degrees) or eight (0, +45, -45, +90, -90, +135, -135, and 180 degrees). If there are four phases (m = 4), the MPSK mode is called quadrature phase-shift keying or quaternary phase-shift keying (QPSK), and each phase shift represents two signal elements. If there are eight phases (m = 8), the MPSK mode is known as octal phase-shift keying (OPSK), and each phase shift represents three signal elements. In MPSK, data can be transmitted at a faster rate, relative to the number of phase changes per unit time, than is the case in BPSK.

It can also be define Phase shift keying is a technique which shifts the period of a wave.

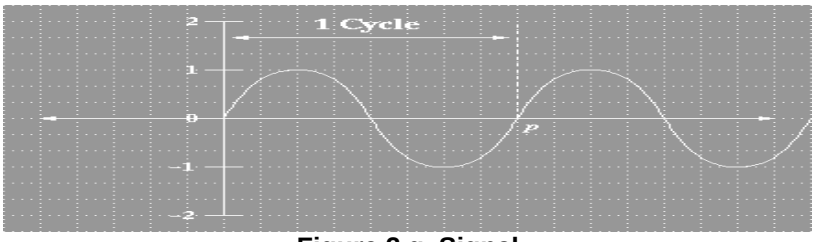

Figure 2.q. Signal

This wave has a period of p, & starting from point ‘0’

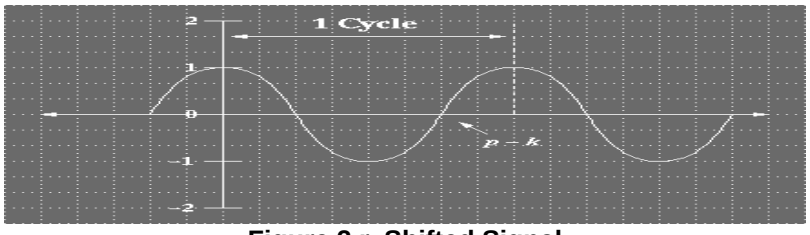

Figure 2.r. Shifted Signal

This is the same wave as the first, but its phase has been shifted. The period starts at the wave's highest point (1). Here we have shifted this wave by one quarter of the wave's full period. We can shift it another quarter, if we wanted to, so the original wave would be shifted by half its period. And we could do it one more time, so that it would be shifted three quarters of its original period.

This means we have 4 separate waves, if we consider each wave for some binary value. Since there are 4, we can let each wave signify 2 bits (00, 01, 10, 11):

Bit value	Phase Shift
00	0
01	Pi/4
10	Pi/2
11	Pi/4

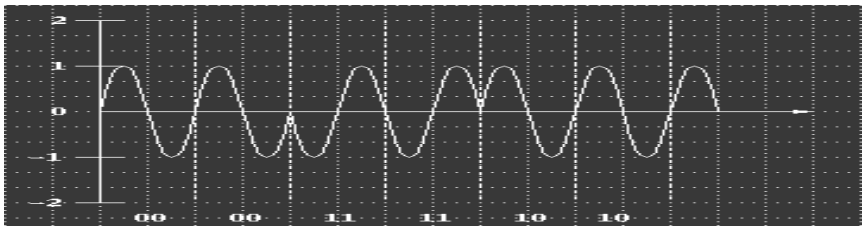

Figure 2.s. PSK Signal

This technique of letting each shift of a wave represent some bit value is **phase shift keying**. But the real key is to shift each wave relative to the wave that came before it. The correct pattern should be: 00 00 10 00 10 00.

In a communications system, the representing of characters, such as bits or quaternary digits, by a shift in the phase of an electromagnetic carrier wave with respect to a reference, by an amount corresponding to the symbol being encoded. For example, when encoding bits, the phase shift could be 0° for encoding a "0," and 180° for encoding a "1," or the phase shift could be -90° for "0" and +90° for a "1," thus making the representations for "0" and "1" a total of 180° apart.

In PSK systems designed so that the carrier can assume only two different phase angles, each change of phase carries one bit of information, i.e., the bit rate equals the modulation rate. If the number of recognizable phase angles is increased to 4, then 2 bits of information can be encoded into each signal element; likewise, 8 phase angles can encode 3 bits in each signal element.

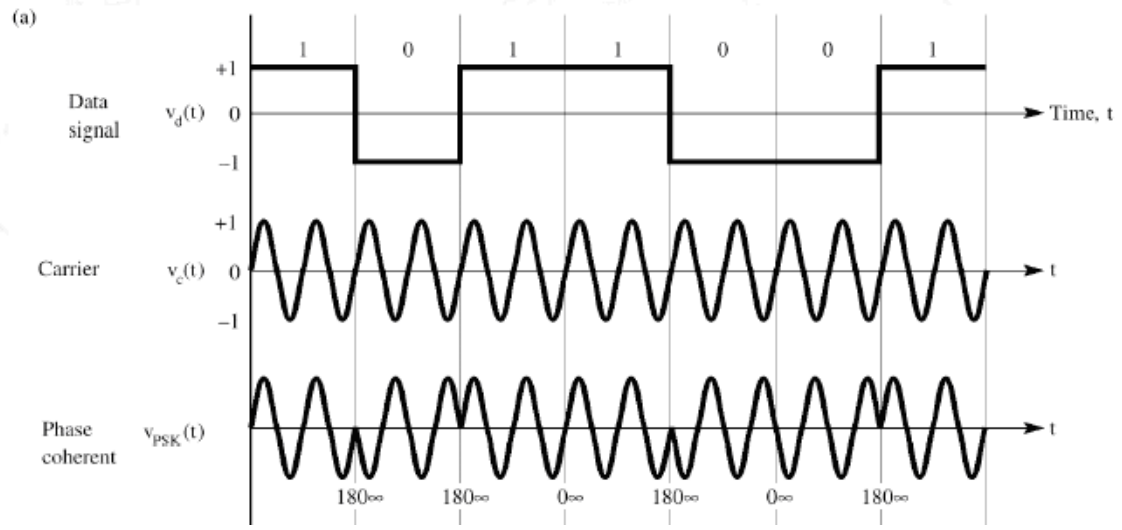

Figure 2.t. Binary phase shift keying

The number of times the signal parameter (amplitude, frequency, and phase) is changed per second is called the **signaling rate**. So Phase-shift keying (PSK) is a method of digital communication in which the phase of a transmitted signal is varied to convey information.

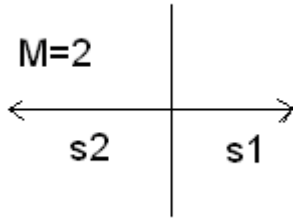

Figure 2.u. Vector Diagram for PSK-2

2.3.5 QUADRATURE PHASE-SHIFT KEYING (QPSK)

QPSK is a form of phase shift keying (PSK) using four phase states, normally 90 degrees apart. Quadrature phase-shift keying (quadriphase, quaternary phase-shift keying) is a form of modulation in which a carrier is sent in four phases, 45, 135, 225, and 315 degrees, and the change in phase from one symbol to the next encodes two bits per symbol. In QPSK, the four angles are usually out of phase by 90°.This a significant advantage over the normal Phase shift Keying of two values effectively doubling the bandwidth.

If we explained through figure then it describes a four-level (4-ary) PSK or quadriphase shift keying (QPSK) where M= 4.Binary Digit source digits are collected two at a time & for each symbol interval the two sequential digits

instruct the modulator as to which of the four waveforms to produce. For typical coherent M-ary PSK system, s(t) can be expressed as

$$s(t) = sqre (2E / T \cos (w t - 2 * pi / M)) \quad 0 <= t <= T$$

……..Equation 2.2

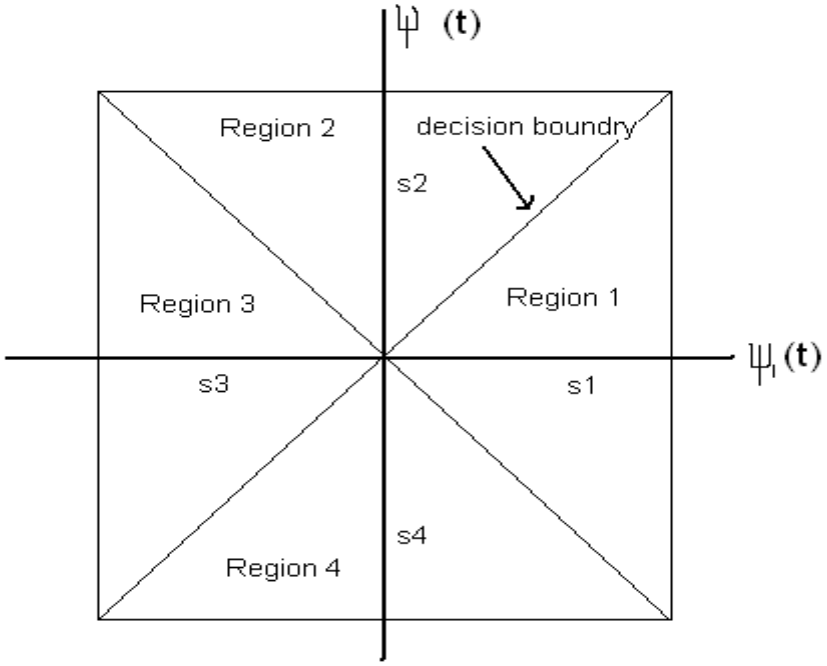

Figure 2.v. Signal Space & decision region for QPSK system

E is the energy constant of s(t) over each symbol duration T, and w is the carrier frequency.

QPSK is the unique among MPSK signal set in the sense that QPSK waveform set is represented by combination of antipodal and orthogonal members. the decision boundaries partition the signal space into M = 4 regions: the construction is similar to the procedure for M = 2.the decision rule for the detector is to decide that s(t) is transmitted if the if the received signal vector falls in region 1 and s2(t) is transmitted if the received signal vector falls in region 2 and so on .

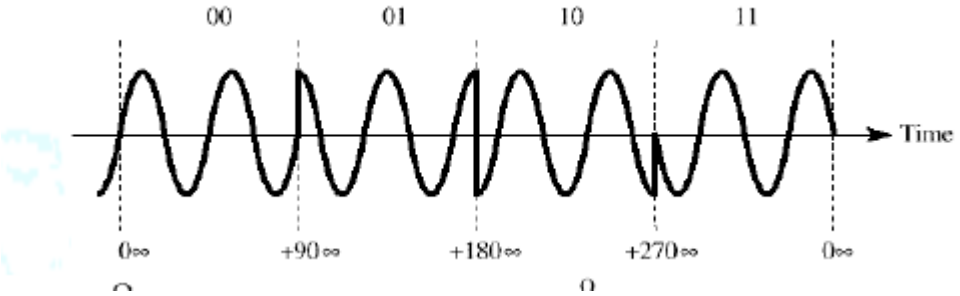

Figure 2.w. Phase Time Diagram

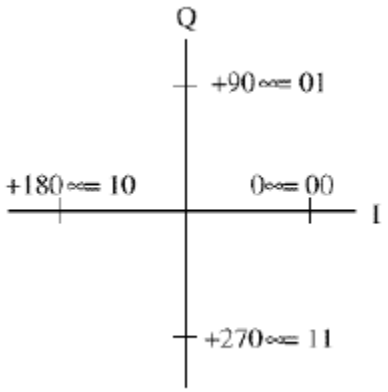

Figure 2.x. Phase Time Diagram

QPSK is a digital frequency modulation technique used for sending data over coaxial cable networks.It's both easy to implement and fairly resistant to noise, QPSK is used primarily for sending data from the cable subscriber upstream to the Internet

2.3.6 **PSK-8**

PSK 8 has a symbol size of three bits. The basic parameters are listed below.

Amplitudes	=	1
Normalized amplitude	=	1
Phases	=	5

2.4 SUMMARY

All modulated signals have three modulating parameters: -

Amplitude

The amplitude of a signal is the height of the wave above or below a given reference point.

Frequency

The frequency is the number of times a signal makes a complete cycle within a given time frame. Spectrum - The range of frequencies that a signal spans from minimum to maximum. Bandwidth - The absolute value of the difference between the lowest and highest frequencies of a signal. For example, consider an average voice: The average voice has a frequency range of roughly 300 Hz to 3100 Hz. The spectrum would thus be 300 - 3100 Hz the bandwidth would be 2800 Hz

Phase

The phase of a signal is the position of the waveform relative to a given moment of time or relative to time zero. A change in phase can be any number of angles between 0 and 360 degrees. Phase changes often occur on common angles, such as 45, 90, 135, etc.

CHAPTER 3

MODULATION HOPPING TRANSCEIVER SYSTEM (MHTS)

INTRODUCTION

This chapter explains the concept of the newly proposed transceiver system. It explains all the modules of the transceiver system briefly and then the working of the transceiver system in detail. The objectives set forth for this chapter are as follows: -

3.1 Background.
3.2 Modules of *MHTS*.
3.3 Working of *MHTS*.
3.4 Proposed Technical Characteristics of the *MHTS*.
3.5 Proposed Technical Characteristics of the *Universal Receiver*.
3.6 Summary.

3.1 BACKGROUND

Space belongs to everyone and anything once transmitted through it knows no boundaries and could be received, demodulated and decoded by any third party interrupting and intruding into the communication. No one ever knows how efficient the third party may be.

Our project is aimed at the integration or combination of different digital demodulators. This gives rise to an idea of *Universal Receiver*. This concept arises from its application and need in the *Communication Security*.

From military point of view, its application is quite understandable in the Electronic Support Measures (ESM), Electronic Counter Measures (ECM) and Electronic Counter-Counter Measures (ECCM). Any one worried about the communication taking place between two enemy terminals would certainly need to know about the modulation technique that is been used for communication. Hence an application that speaks by itself.

From commercial point of view, its application is in view of integrating different things to a common platform. What if a user can use his mobile for the following purposes: -

a. Receiving the radio broadcast.
b. Receiving the television broadcast.
c. Monitoring the satellite communication.
d. Monitoring the mobile communication in ones area.
e. Scanning any communication taking place on any modulation techniques.

Certainly the user would be pleased and would never mind paying anything for it. So this is what our proposed transceiver system is aiming to.

If I have to provide security to my communication, I would either resort to the ciphering device, encryption, frequency hopping or may be some of the other security techniques. But all that I shall be using would be commercial and the breaking codes and algorithms would have been developed already. Hence my communication is not cent percent secure.

Talking about the military communication systems, so far the security to the communication has been provided by ciphering devices and the frequency hopping devices, which themselves have not been able to provide 100 % interception-free and secure communication. These two methods though much secure but are modulation specific and hence have limited applications. Just as the ciphering device CD-411 (Ericsson), which is an accessory of RL-421 (Ericsson), works only on Frequency Modulation (FM).

Further the ciphering devices always have a deciphering device, thus it greatly requires key security etc. though the probability of detection is quite less but still much can be screened out through the spectrum of the signal, modulated on a single modulation technique transmitted through the space. This is even then not that much secure that one be sure what he has transmitted through the space and would be received only by that individual for whom it was transmitted.

Frequency hoping has been a much talked about concept for secure communication, but the transmission rate and hoping speed i.e. hops per second, decrease the security of the information flow and limits implementation in the field. For example, RL-421 has maximum transmission rate of 1024 KBPS in frequency hoping mode. Hence it cannot support any communication system that requires a 2 MB data stream that is required for its interface with PASCOM system. So its applications are limited.

3.2 MODULES OF MHTS

The block diagram for the Modulation Hopping Transceiver System is shown in figure 3.a and 3.b. Its modules are listed below: -

1. **Transmitter end**
   a. Source.
   b. ADC (Analog-to-Digital Converter).
   c. ECE (Error Correction Encoder).
   d. Encryption Module.
   e. Modulators.
   f. Modulation Selector.
   g. Frequency Hopping Module.
   h. Transmitter.

2. **Receiver End**
   a. Receiver.

b. Frequency Identifier.
c. Modulation Identifier.
d. Demodulators.
e. Decryption Module.
f. ECD (Error Correction Decoder).
g. DAC (Digital-to-Analog Converter).
h. Sink.

- **Source and Sink**

The source and sink may be of one of the under-mentioned data types. A header may be used in order to define the type of data that is been transmitted. The *'text'* may be referred to for the demonstration purposes. This is because of quick digitization and less number of bits. The data types and the respective headers have been shown in table 3.1, below. Any header other than the mentioned below may be used for identification of different data types.

Table 3.1. Data types and the respective headers

S.No.	Data Type	Header
1.	Data (Text)	00
2.	Image	01
3.	Audio	10
4.	Video	11

- **ADC and DAC**

Most of the data types would be in analog form. For any digital communication system the data must be in digital form. Hence the digitization of the data i.e. conversion of analog data into binary bits is needed to be done. This is done by ADC at the transmitter end. This conversion process has been described in Chapter 1 in detail. At the receiver end the reverse process takes place at the DAC. DAC is undertaken after the identification of the data type. Since this process is different for each type of the data. Text is converted using the ASCII codes. Image is digitized by pixel values. The audio and the video are converted by the normal process of digitization. An added process in video digitization is the separation of audio and video signal that usually exists in the video signal and the digitization is done after separation, separately.

- **ECE and ECD**

The Error Correction Information (ECI) is needed to be transmitted along with the data been transmitted. This is necessary because the errors are needed to be corrected after demodulation etc at the receiver end that might have occurred because of the due to channel effects and other circuitry noises etc. This is to make sure that all the data is received in the correct manner. This is one of the effective ways of FEC [1]. Previously the retransmission request was the only method available, if the data received was not understandable. This resulted in time delays and bandwidth utilization. The ECI is embedded at the transmitter into the data using the ECE and at the receiver it extracted out of the data using the ECD.

- **Encryption and Decryption Modules**

The data security is provided by one of the encryption techniques. In encryption process the data is transformed into a not understandable language. This is to say, if two people are able to speak and understand English and French simultaneously, and a third man, knowing only English, gets in between them and wants to listen to them and the two do not wish that the third should know what they are talking about, they would certainly like to switch to French. This rightly is encryption. We convert our data into some thing that seems like noise or something liked that to an intruder who is trying to get the information what he is not suppose to get.

- **Modulators and Demodulators**

Many different digital modulators are fed the digitized, encrypted data simultaneously. All the modulators modulate the data and feed it forward. At the receiver end, the identified modulation technique's demodulator is used for the demodulation purpose. It is to be kept in mind that if we are talking about *'MHTS'*, the modulators and demodulators at the transmitter and the receiver end must be same. Along with that, one other thing is to be taken care of that the modulation techniques must have the same data rate and the requirement if PASCOM system i.e. 2MB stream must be seen for. This will ensure the easy and effective interface of the MHTS with the PASCOM system that is not available with the PATCOM system, which is presently the backbone of the Pakistan Army field communication. But if we are talking in terms of the *'Universal*

---

1. FEC – Forward Error Correction

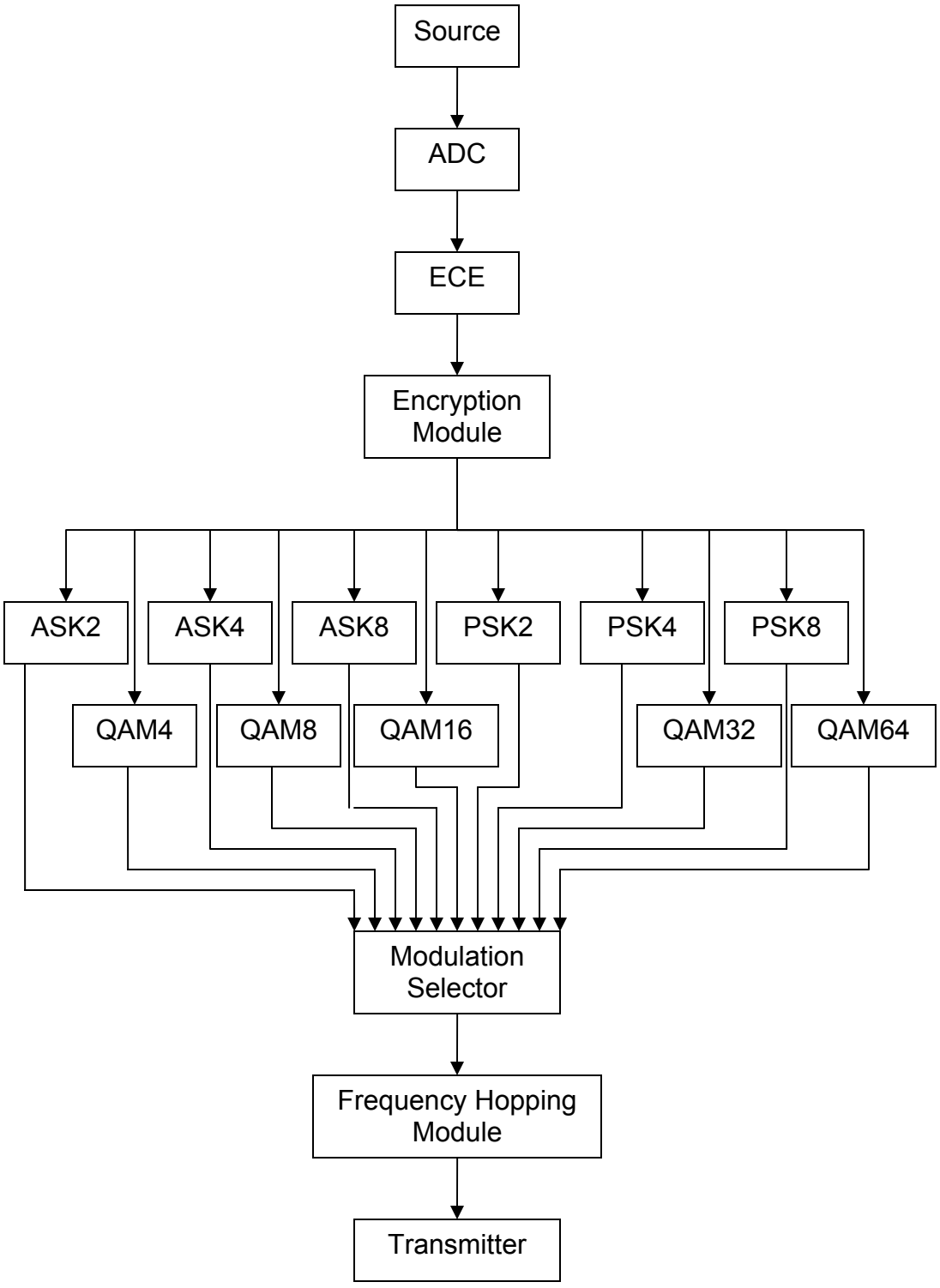

Figure 3.a. Transmitter block diagram of the 'MHTS'

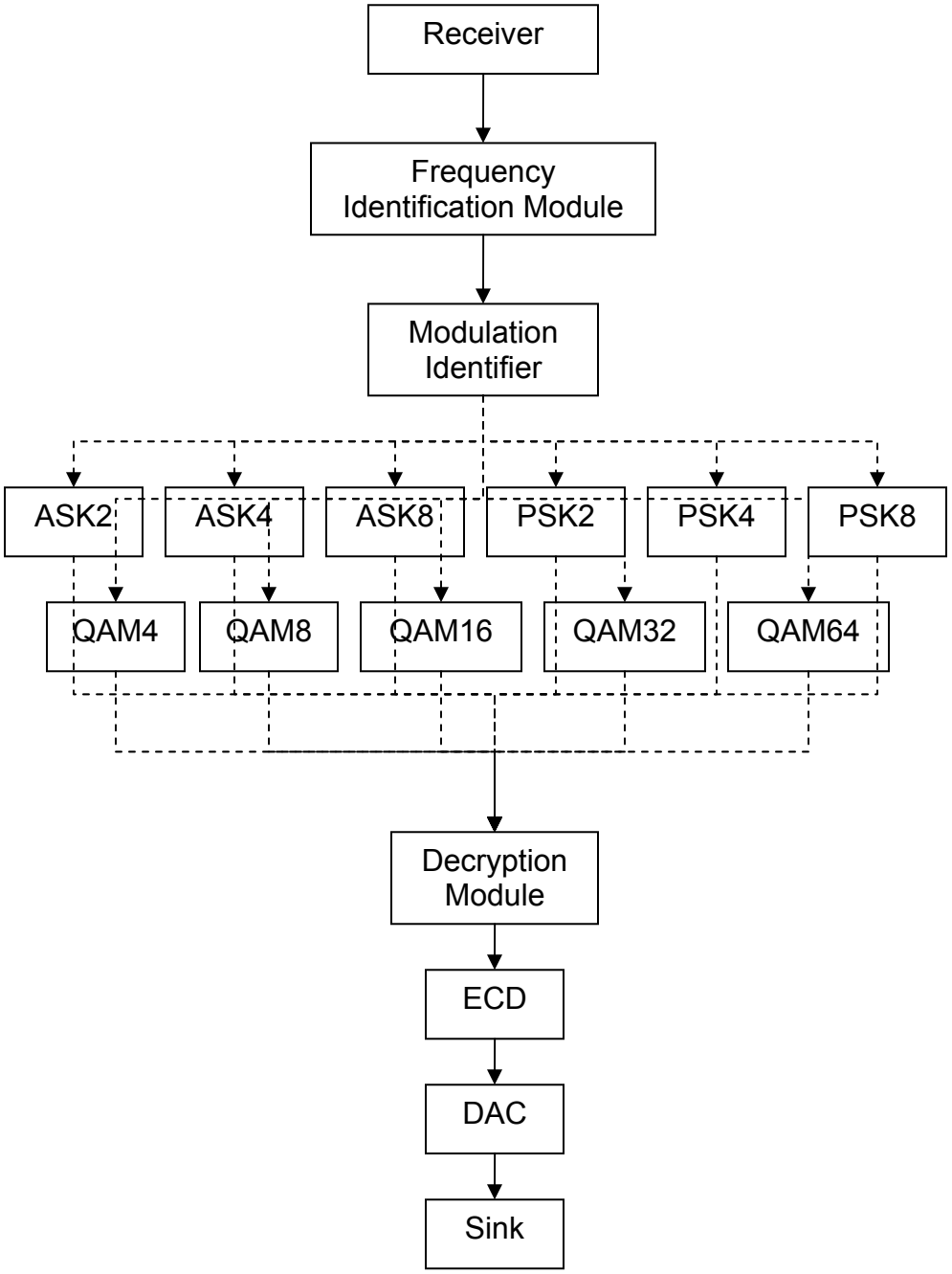

Figure 3.b. Receiver's block diagram of the 'MHTS'

---

* The dashed arrows show the one line feeding not parallel feeding as is done at the transmitter end.

Receiver', the demodulators are needed to be of all the modulation techniques that may exist in the area of its operation. This is to ensure that no data is left unchecked. This means that all that is transmitted and received through the space in your area of interest is monitored by you.

- **Modulation Selector and Modulation Identifier**

The modulators' output is fed to the Modulation Selector, where by one of the modulators' output is selected at random for transmission purpose. At the receiver end, the Modulation Identifier identifies the modulation technique using the algorithm been developed by us.

- **Frequency Hopping and Frequency Identification Module**

The frequency hopping is a much talked about topic. This is one of the effective ways of avoiding jamming during transmission. Normally the frequency hopping takes place using a key at the transmitter and the same key at the receiver for the easy identification of the frequency. But this some how seems unsafe. This is so because of the second application of frequency hopping. The second application may be the security aspect. This is to say if the third party is unable to identify the frequency of the signal it would not be able to get the data or information out of it. So if the key is secretly leaked out, the security of the communication is at risk. Hence in the *'MHTS'* we are talking about Frequency Hopping without keying.

- **Transmitter and Receiver**

The transmitter and receiver are basically the antennas, which shall be liable to receive any type of modulation at any frequency. Since in space nothing except the modulation technique and the frequency is to be catered for detection and evaluation. So the antenna shape and polarization etc must be able to get the unknown signal with unknown parameters and giving it a measurable gain both at the transmitter and the receiver.

3.3 WORKING OF MHTS

The working of the transceiver system can be explained in the following steps: -

1. The data is taken from the user on the transmitter. The data can be of one of the four types mentioned. The data is then digitized i.e. conversion to binary bits ('0' and '1'). The data can be an input from the user (audio signal) or a PC (text, stored image, stored video) or a camera (image or video signal) or from any other source.

2. After digitization the bits are fed to the ECE where the error correction information is added to the data. This would help in the error identification and correction at the receiver end, which might have occurred due to one reason or the other.

3. The data thus obtained is encrypted to provide security to the communication. The encryption may be done using one of the many algorithms available (in case more than one algorithm is implemented for the encryption at the transmitter).

4. The encrypted data is again in the binary forms and hence fed in-parallel to all the modulators available at the transmitter. The modulation is done in parallel and the output of each modulator is fed to the modulation selector.

5. The modulation selector then randomly selects one of the modulators' output for the transmission of data at that instant.

6. After modulation selection the data is fed to the frequency hoping module where the respective process occurs. This module may or may not be involved during operation, as is done in case of RL-421.

7. The transmitter than transmits the data through the space.

8. At the receiver end, the very first step would be the frequency identification. Since, we need to bring down the frequency to some IF for further processing. So the identification of the frequency is needed to be done prior to any other operation.

9. After IF signal has been obtained the next operation would be the modulation identification. This is done by our module. The improvement in this algorithm is still needed because it is working for ideal channel conditions and hence not reliable in real-time application. Once we talk about the transceiver system it has to be real-time and hence the improvement is needed. One way is to add a de-noiser module that shall be able to extract an ideal channel-like signal from the noised one.

10. Once identified with the modulation technique the next step is the demodulation of the signal. The signal received is fed to the identified modulation technique's demodulator. This in return would give us the encrypted data.

11. The encrypted data is needed to be decrypted that is done by the respective decryptor if in case more than one algorithm has been implemented for encryption at the transmitter.

12. The ECD must be able to identify error and correct it, if the error has occurred due to any reason.

13. The next step is the conversion of the demodulated data back to analog form. This is done once the data type has been identified using one of the headers that might be present there.

14. The last step is the sink by itself which is the user or a PC.

3.4 PROPSED TECHNICAL CHARACTERISTICS OF THE MHTS

S.No.	Characteristic	Remarks/Proposed Value
1.	Frequency Range	Shall mainly cover the HF band.
2.	Maximum Communication Range	25 km to 30 km (keeping in view the earth curvature restriction).
3.	Power Supply	AC (220 V) and DC.
4.	No. of Modulators	At least 10 (with equal data rate, but may differ depending on the requirement).
5.	Direct PASCOM Connectivity	YES.
6.	Receiver as Standalone	YES (for ECM utilization).
7.	Frequency Hopping	240 TPM.
8.	Modulation Hopping	60 TPM.
9.	Operating Temperature	Both in the Field and Office utilization.
10.	TDM (FF) Transmission	Any but must support 2048 kbps for PASCOM connectivity.
11.	TDM (FH) Transmission	Any but must support 2048 kbps for PASCOM

		connectivity.
12.	No. of Channels	2000
13.	PATCOM Interface	YES.
14.	Channel Spacing	100KHz.
15.	Power Output	Auto selection.
16.	Clock Input	NRZ.
17.	Encryption	YES. (as and when needed) By one of the implemented techniques or algorithms.

A few of the proposed technical characteristics for *MHTS* are based upon the already existing PATCOM equipment, presently in work in the PAK Army. By doing so a quick and easy development of the project may be the result. Added to this would be less utilization of resources. Since integrating the existing modules may be a bit easy than to develop new and then join them.

3.5 PROPSED TECHNICAL CHARACTERISTICS OF THE UNIVERSAL RECEIVER

S.No.	Characteristic	Remarks/Proposed Value
1.	Frequency Range	Almost infinity (at least till 10 GHz)
2.	Maximum Communication Range	25 km to 30 km (keeping in view the earth curvature restriction).
3.	Power Supply	Battery as used in cellular phones.
4.	No. of Modulators	All of those modulators that may be needed for commercial usage. For Pakistan it may include the following: - • TV.

		• Radio. • GSM. (Ufone, Mobilink, Telenor, Paktel GSM) • CDMA. • Satellite. Communication. • AMPS. (Instaphone, Paktel) • Other which may be region restricted.
5.	Receiver as Standalone	Only for TV and radio broadcast.

3.6 **SUMMARY**

Our project is aimed at the integration or combination of different digital demodulators. This gives rise to an idea of *Universal Receiver*. This concept arises from its application and need in the *Communication Security*.

The modules of the *MHTS* are as follows: -

1. **Transmitter end**
   - i. Source.
   - j. ADC (Analog-to-Digital Converter).
   - k. ECE (Error Correction Encoder).
   - l. Encryption Module.
   - m. Modulators.
   - n. Modulation Selector.
   - o. Frequency Hopping Module.
   - p. Transmitter.

2. **Receiver End**
   - a. Receiver.
   - b. Frequency Identifier.
   - c. Modulation Identifier.
   - d. Demodulators.
   - e. Decryption Module.
   - f. ECD (Error Correction Decoder).
   - g. DAC (Digital-to-Analog Converter).
   - h. Sink.

Data is always to be digitized before feeding to the modulators for transmission. The signal received at the receiver would give us the parameters for identification of modulation technique. Remember no header has been added for modulation identification. It is based totally on the modulated signal received at the receiver.

CHAPTER 4

MODULATION IDENTIFCATION ALGORITHM

INTRODUCTION

This chapter explains the modulation identification algorithm. The theoretical background is given in chapter 2. Refer to ha chapter for better understanding as and when needed. The objectives for this chapter are: -

4.1 Background.
4.2 Algorithm Explanation.
4.3 Identification Statements.
4.4 MFSK Identification.
4.5 Phase Count Ambiguity.
4.6 Summary.

4.1 BACKGROUND

Many algorithms have been developed for modulation identification. The need for modulation identification arises from the development and advancements in the field of communication especially in SDR. Many modulation techniques with certain advantages over the other are in use. The transmission method has also been varying with development and need. Different modulation techniques have come into existence looking into the needs of data type to be transmitted, extent of precision needed, data rate, probability of error, spectrum available, pdf 1 and other parameters.

PSK 2 -16 and QAM 3 -64 are widely used in satellite communication. QPSK is used in CDMA 2000 and BPSK in IS 95 4. Further other modulation techniques have rendered their usage and application in various fields. But the Phase Shift Keying modulation technique and the Amplitude-Phase keying modulation technique are widely used because of the existence of the constellation diagram or state space diagram of these modulation techniques.

The ease in the detection and reconstruction of the constellation diagram at the receiving end has enhanced the application of such modulation techniques many folds.

Since the advancement in communication, the world has diverted to digital communication systems from their analog counter parts. This is because of enhanced data rate and higher noise immunity that almost all the modern communication systems are now digital ones.

---

1. Power Density Function.
2. Phase Shift Keying.
3. Quadrature Amplitude Modulation.
4. American mobile communication standards.

Since there exists a large bank of digital modulation techniques, therefore, there is a need for modulation identification to cater for advancements in SDR. For this a large number of methods and algorithms have been proposed be researchers and developer, each with some advantages over the other. A few of the techniques and methods are as follows: -

1. Maximum Likelihood method.
2. Power Matrices method.
3. Statistical methods.
4. Vector Analysis.
5. Wavelet Analysis.
6. Spectrum Analysis.
7. DMRA (Digital Modulation Recognition Algorithms).

There may be other methods for modulation identification that have not been mentioned above. Each method has certain advantages over the other. Basing on the application area and fields, different methods have been proposed.

4.2 ALGORITHM EXPLAINATION

The algorithm that is been used in our project has been developed and simulated by us. The algorithm may be placed under the statistical method heading. It has been termed as *'Histogram-Count Modulation Identification (HCMI)'* method. There are two ways the algorithm can be made to work. These two ways are shown in figure 3.a and 3.b.

Method I

For the first way there are two stages of the algorithm, which are as follows: -

a. Amplitude Histogram.
b. Phase Histogram.

For the first way the techniques are identified at various stages of the algorithm. As has been shown in figure 3.a ASK 2, ASK 4, ASK 8 and QAM-M are identified at the first stage. QAM-M is further identified at the second stage to QAM-4, QAM-8, QAM-16, QAM-32 and QAM-64. MPSK is further identified by repeating stage two. The values in the brackets show the obtained values of the identification parameters. The shaded boxes show the technique that has been identified, then.

All the modulation techniques have certain parameters basing upon which modulation takes place. The explanation below describes them in a better way. Suppose the received unknown signal at the receiver end be s(t). Then, s(t) can generally be represented with the help of following equation:

$$s(t) = s\sim(t) * \sin (w_c t + \theta_c)$$

where,

w_c = carrier frequency

θ_c = carrier phase

and s~(t) for each modulation technique is defined here. For ASK

$$s\sim(t) = d(t) * A$$

where,

d(t) = bit stream (0,1)

A = Amplitude of wave

and *wc and θc* as constants.

For PSK,

$$s\sim(t) = d_i(t) \text{ with } \theta_c = \theta_i$$

where,

i = bit position

w_c ,amplitude as constant.

For FSK,

$$s\sim(t) = d_i(t) \text{ with } w_c = w_i$$

where,

i = bit position

θ_c , amplitude as constant.

For QAM,

$$s\sim(t) = d_i(t) * Ai \text{ with } w_c = w_i$$

where,

i = symbol position

w_c , frequency as constant.

Let's take an example of QAM 8. QAM 8 has two amplitudes and four phases resulting into eight possible combinations, one each for the respective symbols. For QAM 8, we have eight symbols of 3 bits each. Hence this becomes the identification parameter for the QAM 8. Such are the parameters of each modulation technique that helps us in the developed algorithm to identify the modulation technique occurring, then.

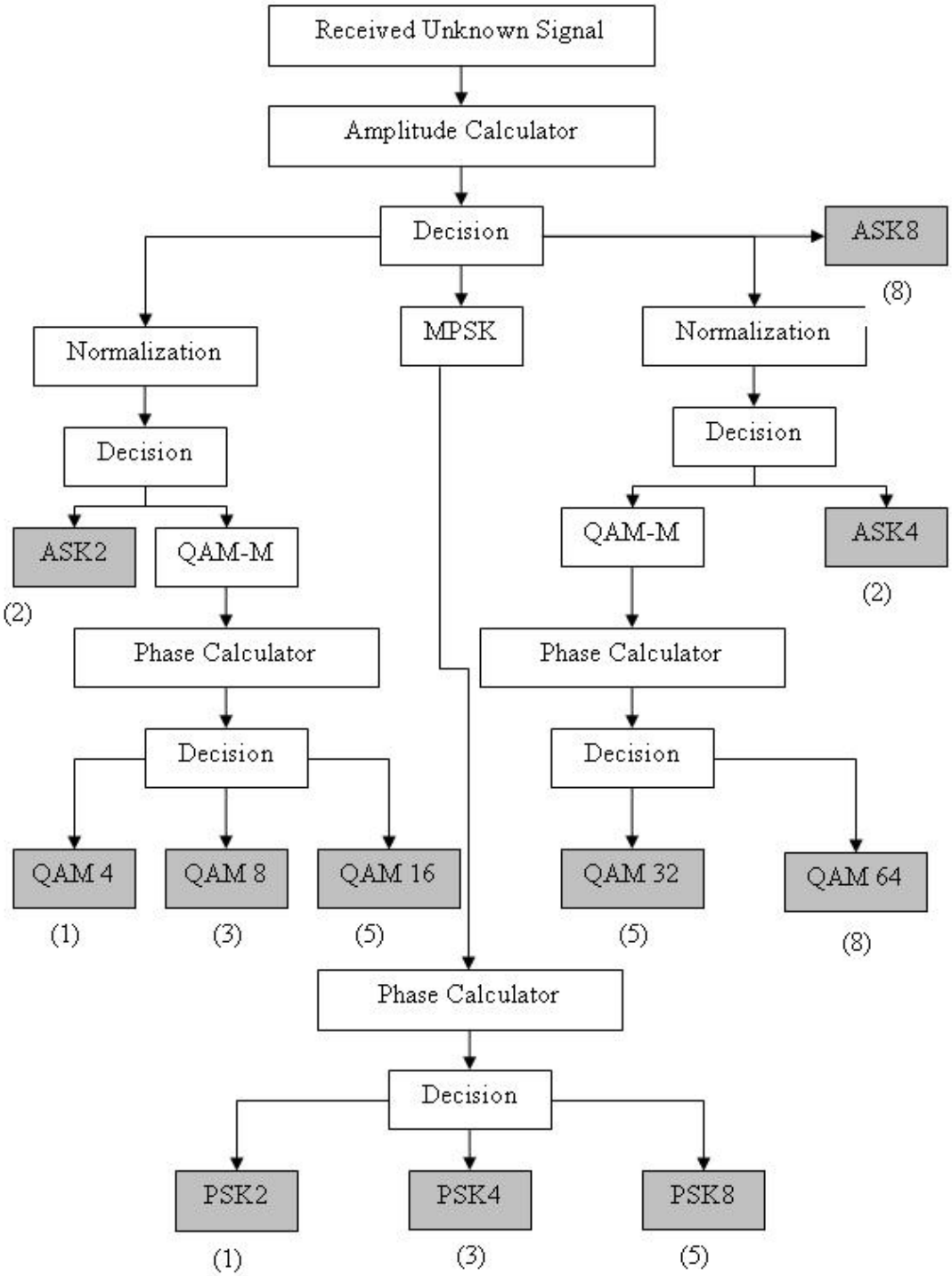

Figure 4.a. Method I for the working of the Algorithm

The received unknown signal with unknown modulation technique is fed to *'Amplitude Calculator'*. The amplitude calculator plots the instantaneous amplitude histogram with the number of instantaneous amplitudes on y-axis and the values of instantaneous amplitudes of x-axis. This then counts the number of instantaneous

amplitudes in the unknown received signal. The identification parameter is the number of instantaneous amplitudes, which is independent of the values of the instantaneous amplitudes. Hence, the identification parameter is the number of instantaneous amplitudes not the value of the instantaneous amplitudes. This number is counted and this majorly identifies the ASK modulation techniques.

If the number of instantaneous amplitudes is '4', QASK is identified as the modulation technique occurring. Likewise, if the number of instantaneous amplitudes is '8', ASK 8 is identified as the modulation technique occurring. But the problem arises when ASK 2 is needed to be identified uniquely. Since ASK 2, QAM 4, QAM 8 and QAM 16, all the three have two instantaneous amplitudes as the modulating parameter. Therefore, amplitude normalization is done to identify ASK 2 from the three. After the instantaneous amplitude normalization has occurred, ASK 2 would still have '2' instantaneous amplitudes (one would be '0' and the other will be '1'). Where as, the other three modulation techniques would have single constant amplitude. Hence ASK 2 is identified uniquely from QAM4, QAM 8 and QAM 16. Till this point, the three Amplitude modulation techniques and the QAM-M has been identified. Further if the number of instantaneous amplitudes in the received unknown signal, before the normalization is '1', MPSK is the modulation technique. Since, it has constant amplitude throughout its occurrence.

The unknown received signal with its normalized amplitude is fed to the *'Phase Calculator'*. Here, again the identification parameter is the number of instantaneous phases not the value of these instantaneous phases. The 'Phase Calculator' plots the histogram of the instantaneous phases with number of instantaneous phases on the y-axis and values of instantaneous phases on x-axis, just like 'Amplitude Calculator', which gives the number of instantaneous phases. If the number of instantaneous phases is '1' along with the number of instantaneous amplitudes as 2, QAM 4 is the identified modulation technique. If the number of instantaneous phases is '3' along with the known '2' instantaneous amplitudes, the modulation technique identified would be QAM 8. Likewise, if the number of instantaneous phases is '5' along with the known '2' instantaneous amplitudes, the identified modulation technique would be QAM 16. If the number of instantaneous phases is '5' along with the number of instantaneous amplitudes as '4', QAM32 is the identified modulation technique. Lastly if the number of instantaneous phases is '8' along with the number of instantaneous amplitudes as '4', QAM 64 is the identified modulation technique.

Now if the number of instantaneous amplitudes is '1', further identification with in MPSK is needed. With the number of instantaneous amplitudes, both before and after normalization as '1', MPSK is needed to be further identified with in its own domain. Like QAM, the MPSK signal is fed to the 'Phase Calculator', which plots the histogram of the instantaneous phase and gives their number. If the number of instantaneous phases is '1', PSK 2, if '3', PSK4 and if '5', PSK 8 is identified. The obtained numbers are different from the expected numbers because of calculation restrictions.

Hence we have identified 11 modulation techniques that include ASK 2, ASK 4, ASK 8, PSK 2, PSK 4, PSK 8, QAM 4, QAM 8, QAM 16, QAM 32 and QAM 64, using this algorithm. Here the symbol duration of the received unknown signal is known only. Generally speaking, we can say that this algorithm would help us in the identification of Mary-ASK, Mary-PSK, Mary-FSK and QAM-M. For the first three modulations, we have

$$M > 2 \;\&\; M = 2\ ^{\wedge} n$$

For QAM-M, M is defined as

$$M > 4 \;\&\; M = 2^{\wedge}\ n$$

where,

n = positive integer

Method II

The second way of working of the algorithm is shown in figure 4.b. It has three stages, which are listed below: -

a. Amplitude Calculator.
b. Phase Calculator.
c. Identifier.

The parameters are calculated just like in Method I. here the difference arises during the identification process. The first two stages calculate the three identification parameters values, and feed them in parallel to the identifier where a set of logical statements help in the identification of the digital modulation technique. Here the techniques are identified at the last stage.

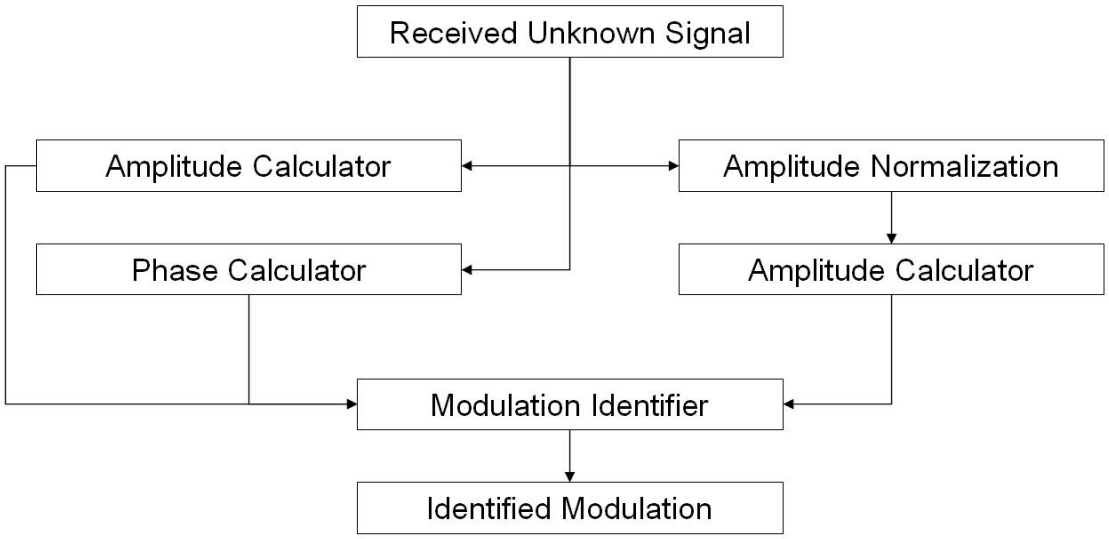

Figure 4.b. Method II for the working of the Algorithm

Comparison

Method I	Method II
Requires less processing	Requires more processing
Early identification i.e. stage-wise identification	Identification at the last stage
Some probability of error occurs	Confirm identification

4.3 IDENTIFICATION STATEMENTS

The eleven modulation techniques that have been implemented in our project are identified using the respective basic properties i.e. the parameters basing upon which each modulation takes place. These have been numerically listed in table 4.1, below. Followings are the identification statements for each: -

ASK 2

1. No variation in Phase i.e. the instantaneous phase count from the phase histogram is '1'.
2. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is '1'.
3. Two amplitudes both before and after normalization i.e. the instantaneous amplitude count from the amplitude histogram is '2' both before and after normalization.

ASK 4

1. No variation in Phase i.e. the instantaneous phase count from the phase histogram is '1'.
2. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is '1'.
3. Four amplitudes both before and after normalization i.e. the instantaneous amplitude count from the amplitude histogram is '4' both before and after normalization.

ASK 8

1. No variation in Phase i.e. the instantaneous phase count from the phase histogram is '1'.
2. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is '1'.

3. Eight amplitudes both before and after normalization i.e. the instantaneous amplitude count from the amplitude histogram is ‘8’ both before and after normalization.

PSK 2

1. No variation in Amplitude i.e. the instantaneous amplitude count from the amplitude histogram is ‘1’.
2. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is ‘1’.
3. Two phases i.e. the instantaneous phase count from the phase histogram is ‘2’.

PSK 4

1. No variation in Amplitude i.e. the instantaneous amplitude count from the amplitude histogram is ‘1’.
2. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is ‘1’.
3. Four phases i.e. the instantaneous phase count from the phases histogram is ‘4’.

PSK 8

1. No variation in Amplitude i.e. the instantaneous amplitude count from the amplitude histogram is ‘1’.
2. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is ‘1’.
3. Four phases i.e. the instantaneous phase count from the phase histogram is ‘8’.

QAM 4

1. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is ‘1’.
2. Two amplitudes before normalization and one amplitude after normalization i.e. the instantaneous amplitude count from the amplitude histogram is ‘2’ before normalization and ‘1’ after normalization.
3. Two phases i.e. the instantaneous phase count from the phase histogram is ‘2’.

QAM 8

1. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is ‘1’.
2. Two amplitudes before normalization and one amplitude after normalization i.e. the instantaneous amplitude count from the amplitude histogram is ‘2’ before normalization and ‘1’ after normalization.
3. Four phases i.e. the instantaneous phase count from the phase histogram is ‘4’.

QAM 16

1. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is '1'.
2. Two amplitudes before normalization and one amplitude after normalization i.e. the instantaneous amplitude count from the amplitude histogram is '2' before normalization and '1' after normalization.
3. Eight phases i.e. the instantaneous phase count from the phase histogram is '8'.

QAM 32

1. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is '1'.
2. Four amplitudes before normalization and one amplitude after normalization i.e. the instantaneous amplitude count from the amplitude histogram is '4' before normalization and '1' after normalization.
3. Eight phases i.e. the instantaneous phase count from the phase histogram is '2'.

QAM 64

1. No variation in Frequency i.e. the instantaneous frequency count from the frequency histogram is '1'.
2. Four amplitudes before normalization and one amplitude after normalization i.e. the instantaneous amplitude count from the amplitude histogram is '4' before normalization and '1' after normalization.
3. Sixteen phases i.e. the instantaneous phase count from the phase histogram is '16'.

Table 4.1 Numerical Values of Identification Parameters

S. No.	Technique	No. of Identification Parameters			
		Amp	Norm. Amp	Phases	Freq
1.	ASK 2	2	2	1	1
2.	ASK 4	4	2	1	1
3.	ASK 8	8	2	1	1
4.	PSK 2	1	1	1	1
5.	PSK 4	1	1	3	1
6.	PSK 8	1	1	5	1
7.	QAM 4	2	1	1	1
8.	QAM 8	2	1	3	1
9.	QAM 16	2	1	5	1
10.	QAM 32	4	1	5	1
11.	QAM 64	4	1	8	1

4.4 IDENTIFICATION OF MFSK

MFSK can also be identified uniquely basing on its modulating parameters. It has constant phase and amplitude. The only varying parameter in it is its frequency. Hence if we are able to identify the number of instantaneous frequencies in the received unknown signal, we shall be able to identify MFSK. A few of the methods for calculation of frequencies are as follows: -

a. Fourier Transformation.
b. PLL.
c. Filters.

Hence a slight modification in the algorithm would certainly help us in the identification of modulation techniques where the frequency is also a varying modulating parameter.

4.5 PHASE COUNT AMBIGUITY

Following questions may come to an individual's mind once the number of phases is detected during the Phase Calculation: -

a. Why there is '1' phase instead of '2' for BPSK and QAM-4?
b. Why there is '3' phases instead of '4' for QPSK and QAM-8?
c. Why there is '5' phases instead of '8' for PSK-8 and QAM-16, QAM32?
d. Why there is '8' phases instead of '16' for QAM-64?

The answer is quite simple: the calculation restriction. Figure 4.c illustrates this fact in a better manner.

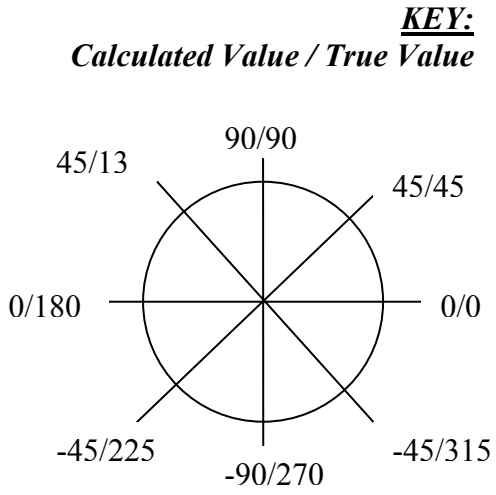

Figure 4.c. Phase Ambiguity illustration for PSK-8, QAM-16 and QAM-32

Very clearly the calculation restriction is shown in the figure above. It is so because

$$\sin(45) = \sin(135) \rightarrow 45 = 135$$
$$\sin(315) = \sin(225) \rightarrow 225 = 315$$
$$\sin(0) = \sin(180) \rightarrow 0 = 180$$

Therefore, we have '5' phases instead of '8' phases for PSK-8, QAM-16 and QAM-32. And likewise we have for other modulation techniques, as has been mentioned above in the questions.

4.7 SUMMARY

Many algorithms have been developed for modulation identification. The need for modulation identification arises from the development and advancements in the field of communication especially in SDR. Many modulation techniques with certain advantages over the other are in use. The transmission method has also been varying with development and need. Different modulation techniques have come into existence looking into the needs of data type to be transmitted, extent of precision needed, data rate, probability of error, spectrum available, pdf and other parameters.

There are three parameters for modulation identification: -

- Number of Amplitudes.
- Number of Normalized Amplitudes.
- Number of Phases.

A novelistic approach has been described for identification of M-FSK. Added to this the phase ambiguity is due to calculation restriction.

CHAPTER 5

MANUAL FOR THE PROJECT 'GUI'

INTRODUCTION

The Graphical User Interface (GUI) for the project is also Matlab 7.0 based. This chapter would help the user to go through the GUI for project demonstration. Behind every figure of the GUI files same .m files are running that have been written for the command window purpose. But here it is to be kept in mind that many things have been added in the GUI .m files. Therefore both the .m files may be consulted in case of any ambiguity in understanding. To use the GUI proceed with the following steps: -

STEP 1:

Type *'gui1'* in the command window and press *'enter'*. A figure, like 5.a as shown below, would appear.

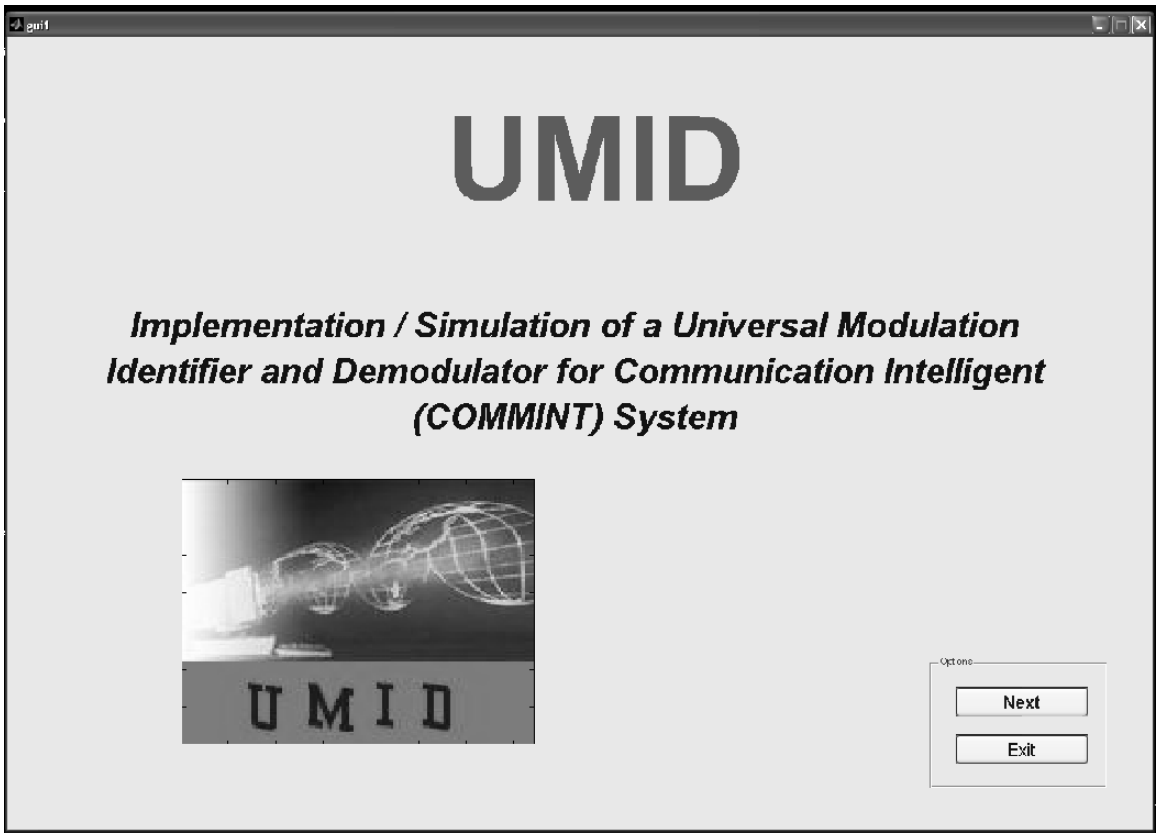

Figure 5.a. gui1 figure of the GUI

This is the opening window of the GUI. The window is showing the project name and the project logo.

STEP 2:

Press *'Next'* to proceed to the next GUI window. The figure 5.b would appear. This figure is showing the name of the syndicate members, the college and the project DS. This figure is labeled as *'gui2'.*

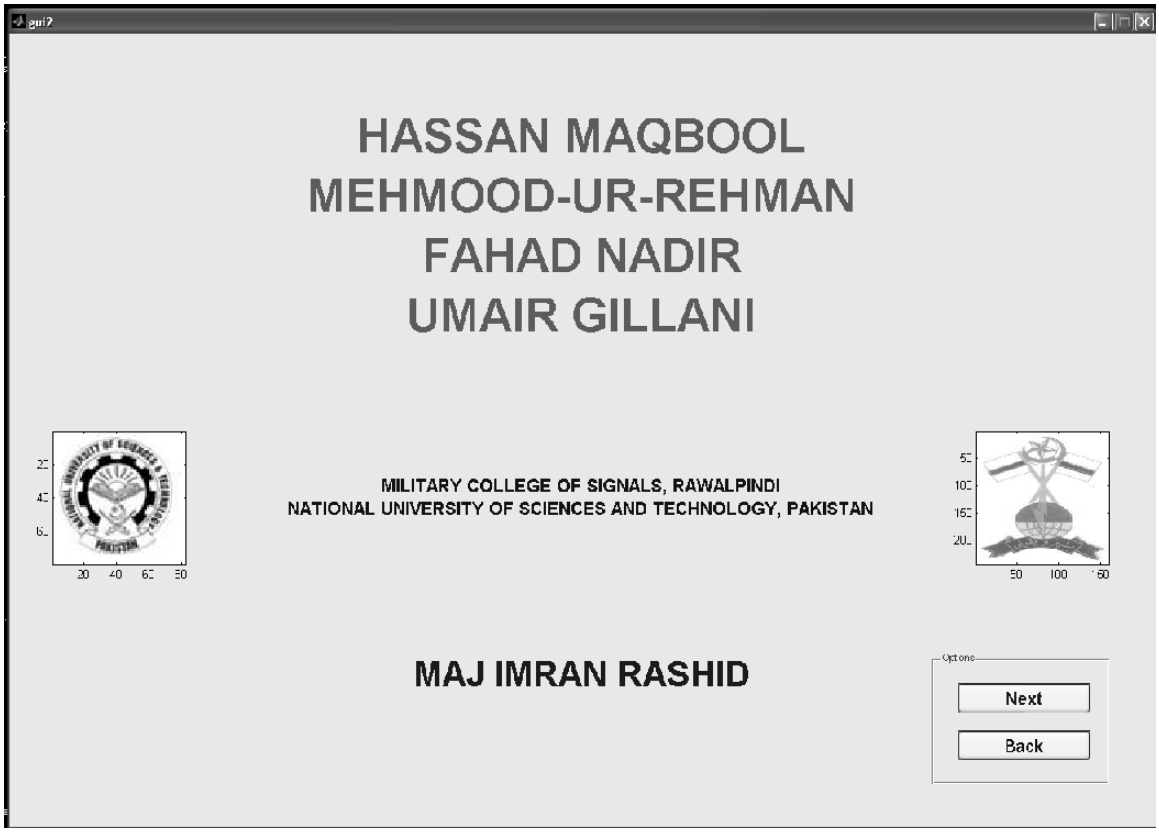

Figure 5.b. gui2 figure of the GUI

STEP 3:

Press *'Next'* to proceed to the next window of GUI titled *'GUI CHOICES'.* This figure looks like figure 5.c. This figure shows the GUI choices. It is titled as *'gui3'.* The choices include the followings: -

- Introduction to the Project.
- MATLAB codes.
- Identification Algorithm.

These are the radio buttons options. Select either of the four options and then press *'NEXT'* to proceed with the options. These three options are the theoretical part of the project and hence do not require any further guidance. Go through them to acquaint yourself with the theory related problems of the project. A big push button in the middle of the GUI titled *'PROJECT DEMONSTRATION'*.

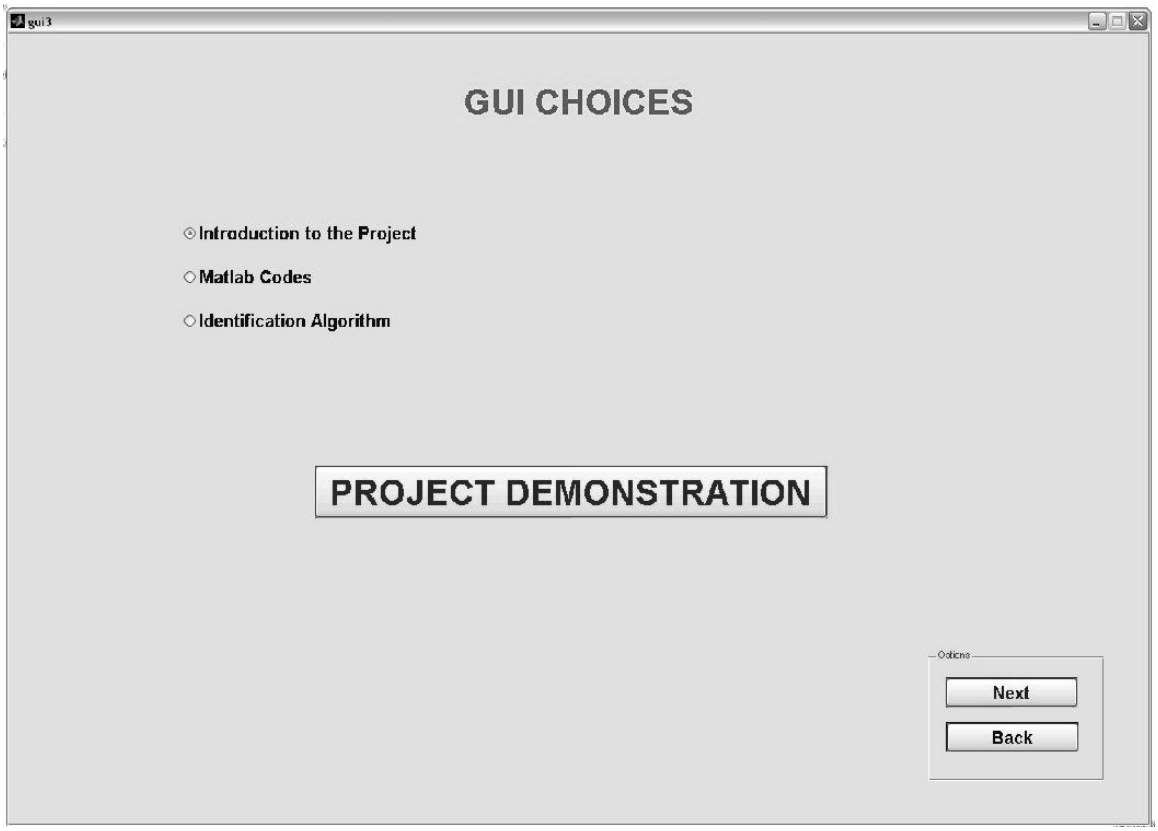

Figure 5.c. gui3 figure of the GUI

STEP 4:

Press *'PROJECT DEMONSTRATION'* push button for project demo. The following figure, figure 5.d will appear. Pressing this push button also executes the *'clear all*;' command, that has been written so in its *.m file*. This figure is titled as *'datasel'* as shown in figure 5.d.

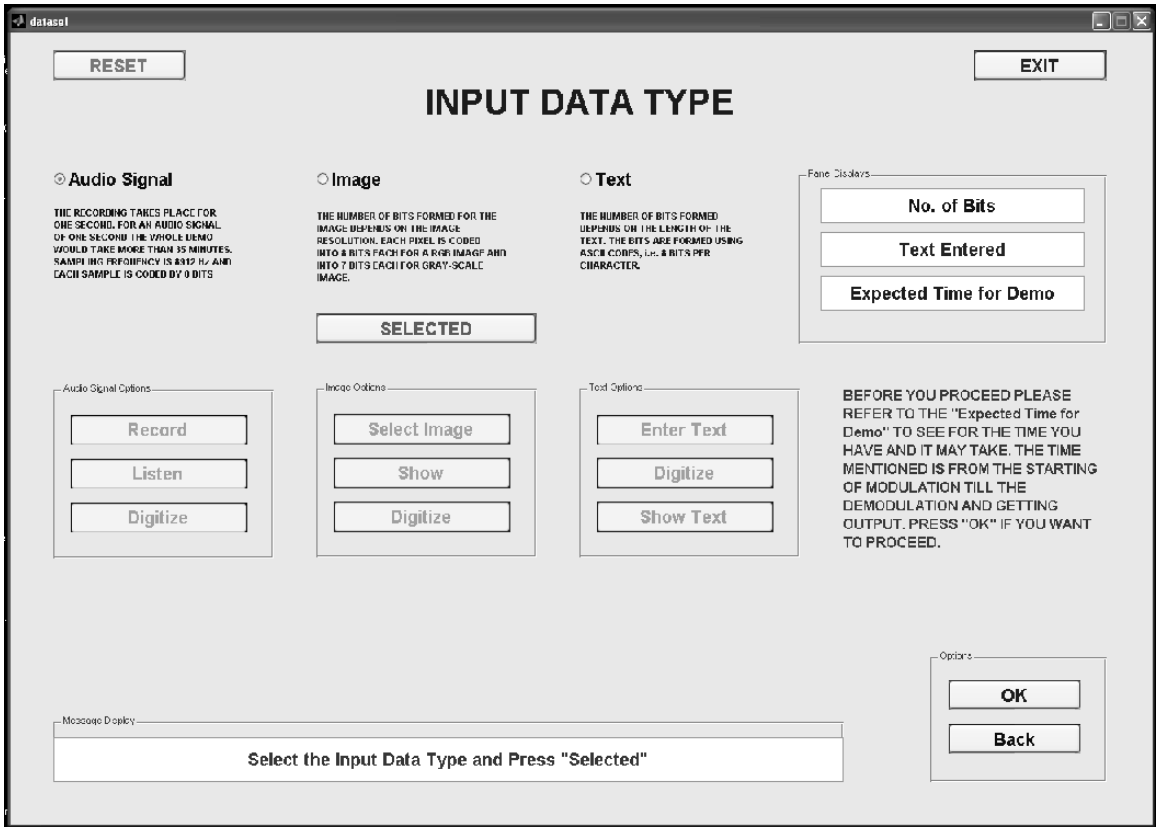

Figure 5.d. 'datasel'

STEP 5:

Select the type of data to be transmitted. press *'SELECTED'* to enter the data type as the input. Once done press *'OK'* to proceed. This figure is titled as *'gui3demo'* that would appear. This has been shown in figure 5.e. Here the GUI asks the user to select whether to select the modulation technique manually or let it be selected automatically. Let's first explain the process by manually selecting the modulation technique.

Figure 5.e. gui3demo figure of the GUI

STEP 6:

Select *'Manual'* radio button and press *'Proceed'.* This figure is titled as *'demomanual'*. The following figure 5.f. appears then. This is the transmitter window of the UMID. The data has been entered already, so select the modulation technique. The entered text would appear in the edit text box 1 titled *'Text to be transmitted'*. The next step is to select the modulation technique from popup menu 1 and press *'Modulation Selected'*. Here the modulation takes place. The values of the input data, modulator's output are stored as follows: -

- *save('manual','s');* ➔ *the bit source*
- *save('man','y');* ➔ *the modulator's output*

They are retrieved afterwards for identification and demodulation. Select the data to be plotted from popup menu 2 and press *'Update Figure'*. The axes 1 would display the plotted data. Press *'Transmit'* to go the receiver's end or press *'Back'* to go back to the *'gui3demo'*. The *'RESET TRANSMITTER'* button on the top resets all the fields of the transmitter.

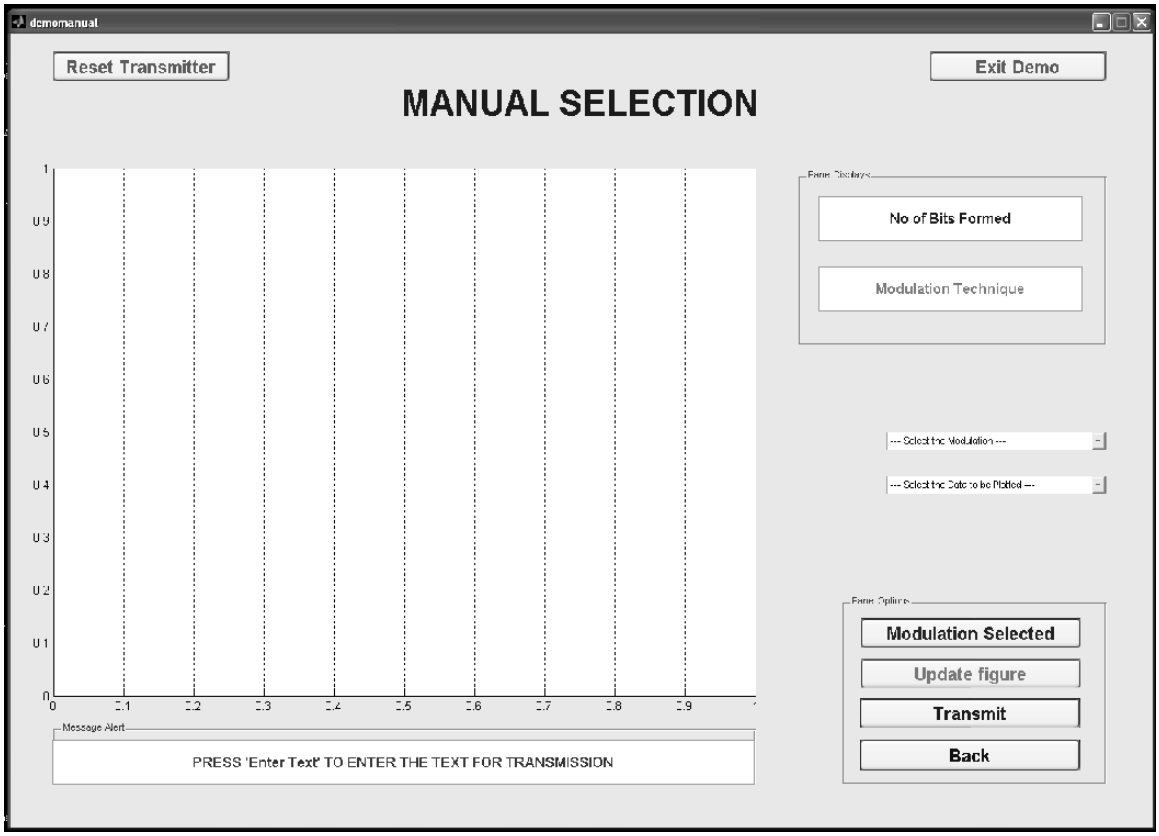

Figure 5.f. demomanual figure for GUI

STEP 7:

Pressing *'Transmit'* on the *demomanual* leads us to *manualtx*. Now we are at the receiver end. Here the data of modulator's data is again loaded.

- *load('man','y'); ➔ load the modulator's output*

Figure 5.h would appear. This figure has been shown below. Here the identification algorithm has been sequentially shown. The push buttons on the right size are the algorithm steps. Follow them in the sequence they are because many parameters are saved during the algorithm process. They are as follows: -

- *save('Amplitude','Amp','M'); ➔ AmpHist.m executes*
- *save('NAmp','NO'); ➔ amp_norm.m executes*
- *save('NA','Amp1','N') ; ➔ AmpHist.m executes again after normalization*
- *save('Ph','Phase','P'); ➔ PhaseDetect.m executes*
- *save('Fop','FOP'); ➔ Fop.m executes for Fourier transforms.*

Like for the previous figure, select the data to be plotted from the popup menu 1 and press *'Update Figure'*. The top three edit text boxes shows the identification parameters values. The fourth long edit text box displays the

identified modulation technique. Press *'Demodulate'* to go to the next figure. The *'Reset Receiver'* clears all the fields of the figure.

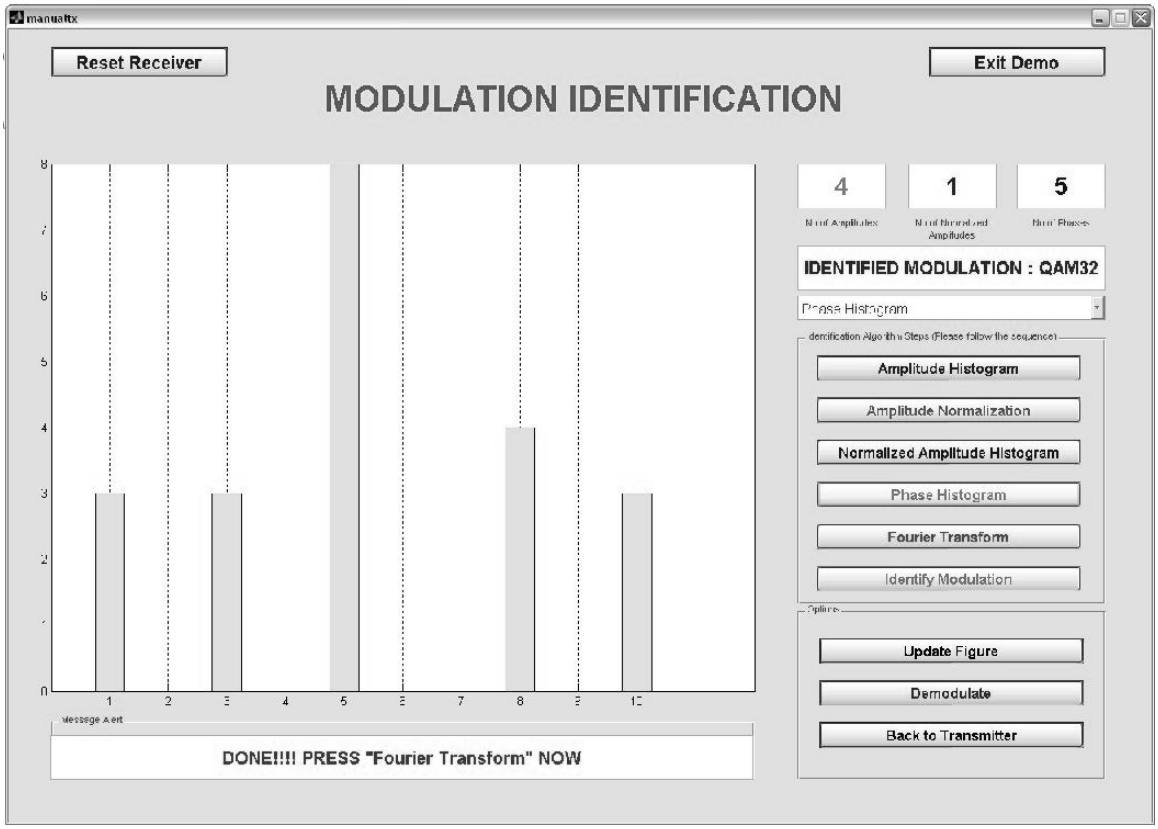

Figure 5.g. manualtx figure for the GUI

STEP 8:

The next step is the *'manual demodulation'*. Here the waveform is demodulated. Select the demodulator manually from the popup menu 1 and press demodulate. Edit box 1 display the identified data type. Edit box 2 and 3 display the *'transmitted bits'* and the *'demodulated bit'* respectively. Select the data to be plotted from popup menu 2 and press update figure for the plot. The next button is the *'BER Calculation'*, which calculates the BER. It is shown in Edit box 4. The *'Back to Transmitter'* would take the user back to the transmitter figure. Here the input data is loaded for BER calculation.

- *load('manual','s');* ➔ *load the bit stream that was given as input to the modulator at the transmitter end*

.

The *'Reset Demodulator'* would clear all the fields of the figure. Press '*Exit Demo'* to end the demonstration. A dialogue box would appear as shown in figure 5.i. Confirm you action, then.

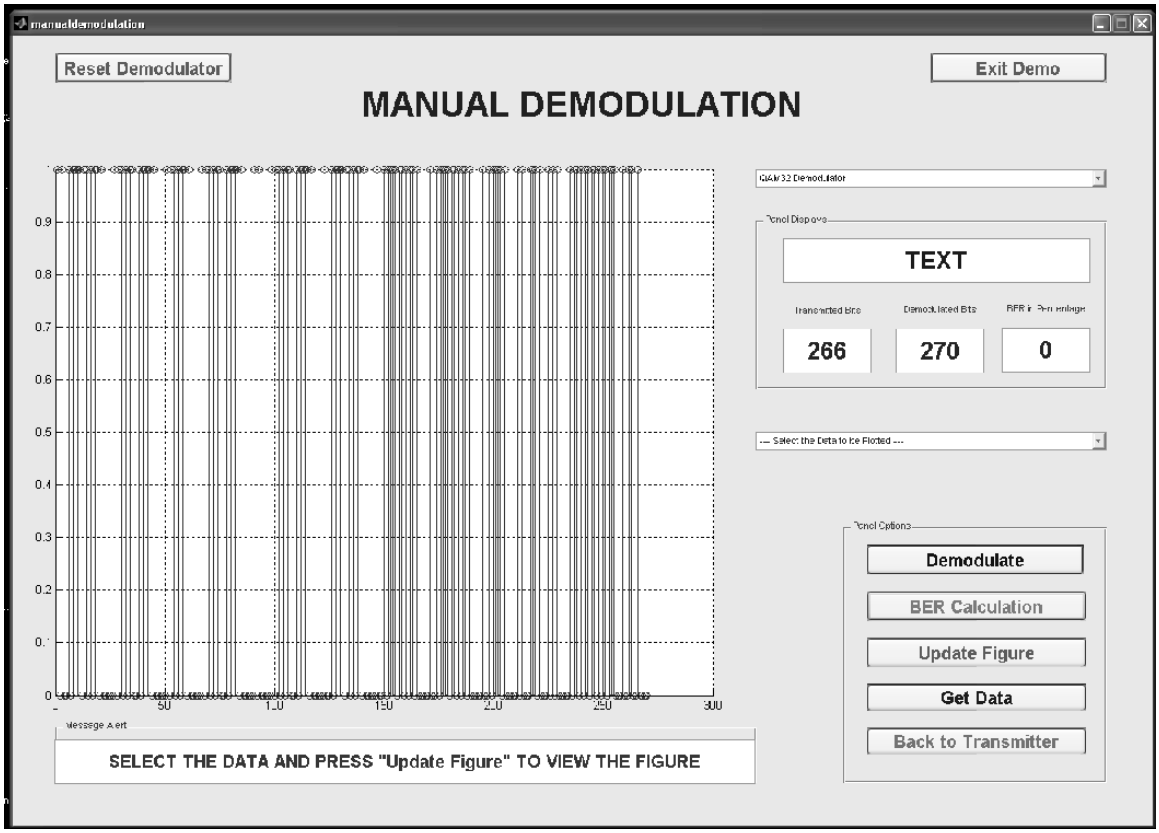

Figure 5.h. manualdemodulation figure for GUI

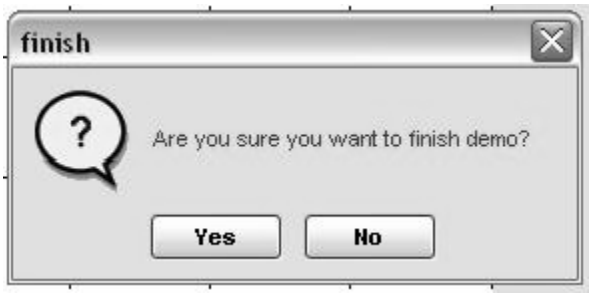

Figure 5.i. Finish figure for GUI

STEP 9:

The identified data type is obtained by pressing the *'get data'* button. The figure shown in figure 5.j. appears and shows the data that was transmitted.

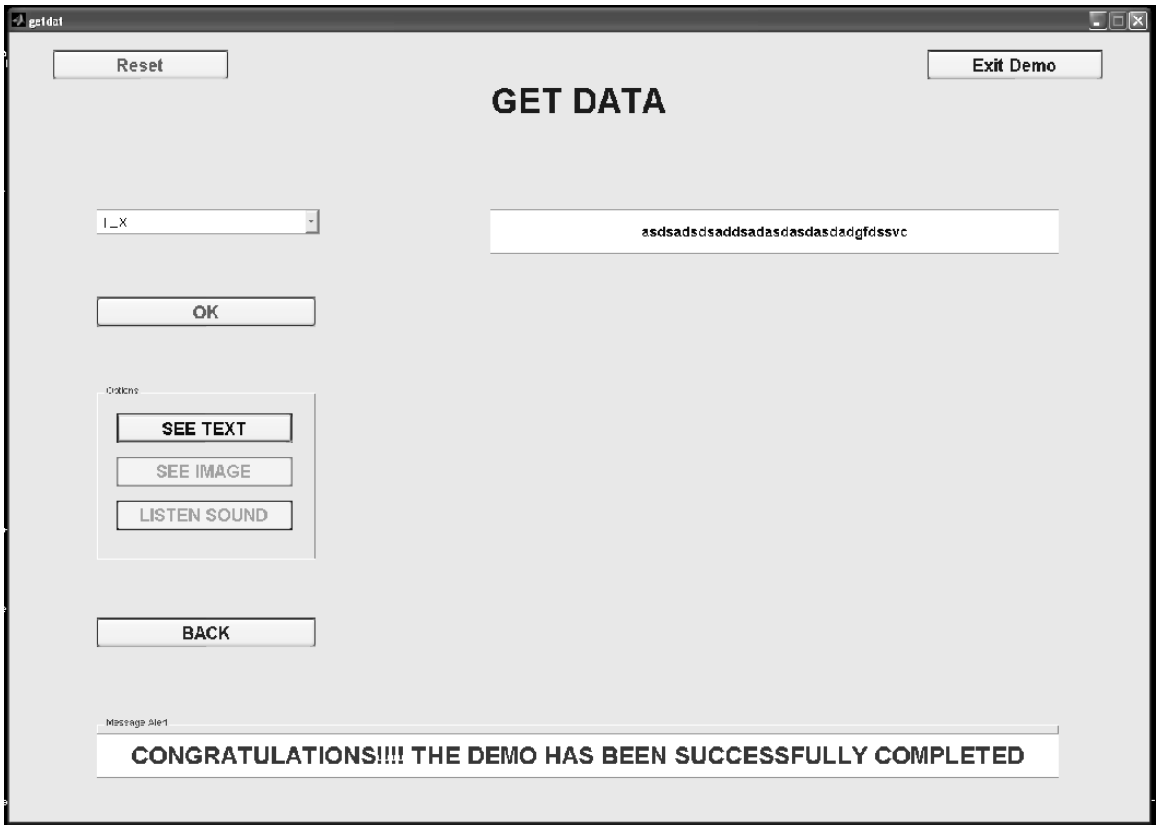

Figure 5.j. figure for data showing

STEP 10:

Now let's proceed with the automatic demonstration method. Select *'Automatic'* instead of *'Manual'* in step 5 and click *'Proceed'*. The *'Demoauto'* figure will be opened. This is shown in figure 5.k below. Rest all the working is same instead of the selection method of the modulation technique. Press *'Autoselect Modulation'* push button to auto select the modulation. Once the modulation technique has been auto selected, the push button becomes inactive. Select the data to be plotted and press *'Update Figure',* or press *'Reset Transmitter'* push button to reset all the fields, or press *'Transmit*' to go to the receiver's end. Here again the source data and modulator's output is stored as follows: -

- save('modop','y','s'); ➔ *where modop is the file name, y is the modulator's output and s is the source data.*

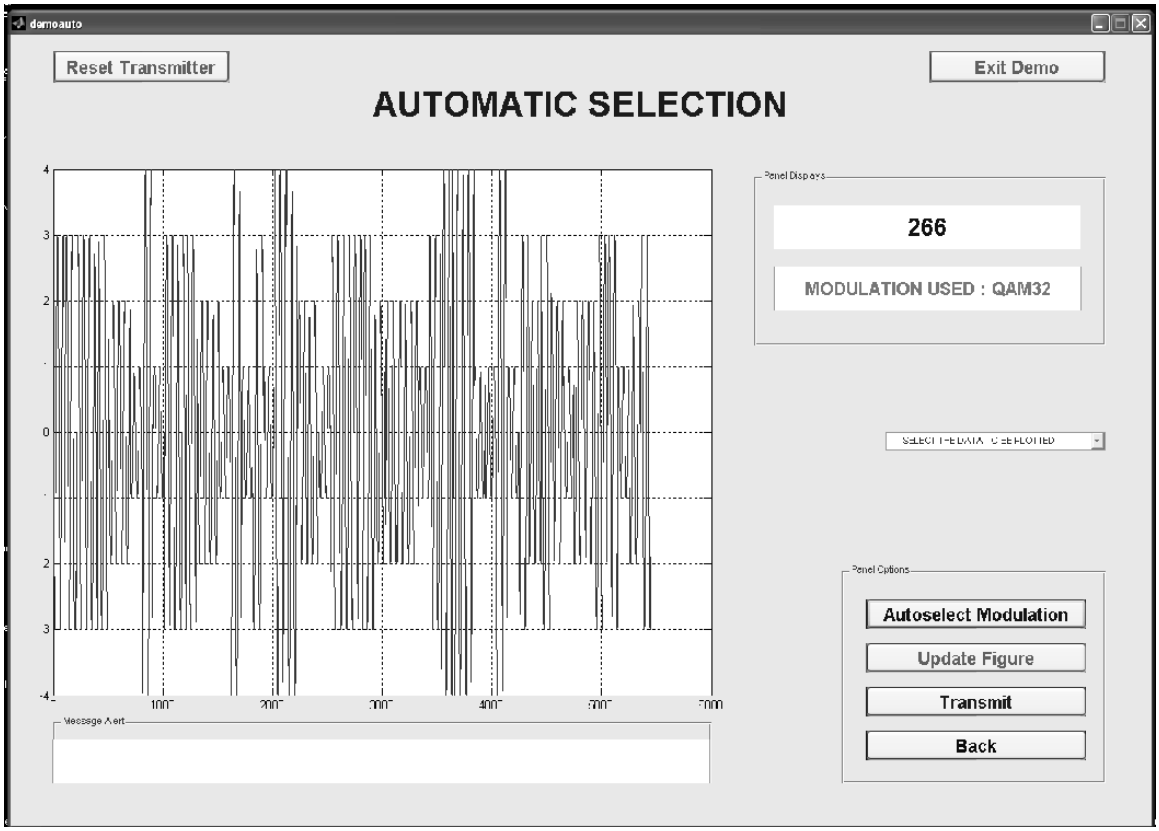

Figure 5.k. demoauto figure for the GUI

STEP 11:

Once on the receiver's side all the working is same less the identification algorithm works at its own instead of the user making it through its stages. Press *'Identify Modulation and Demodulate'* to allow the autorun of the algorithm and let it demodulate using the respective demodulator by itself. The top three left edit text boxes shows the values achieved. The righjt most edit text box shows the identified modulation technique. The edit boxes below shows the number of transmitted bits, number of received bits and BER respectively. The longer edit box below the figures shows the received text. The figure is named as demoalgorithm and is shown if figure 5.lbelow. The *'Back to Transmitter'* push button will take the user back to the transmitter's end.

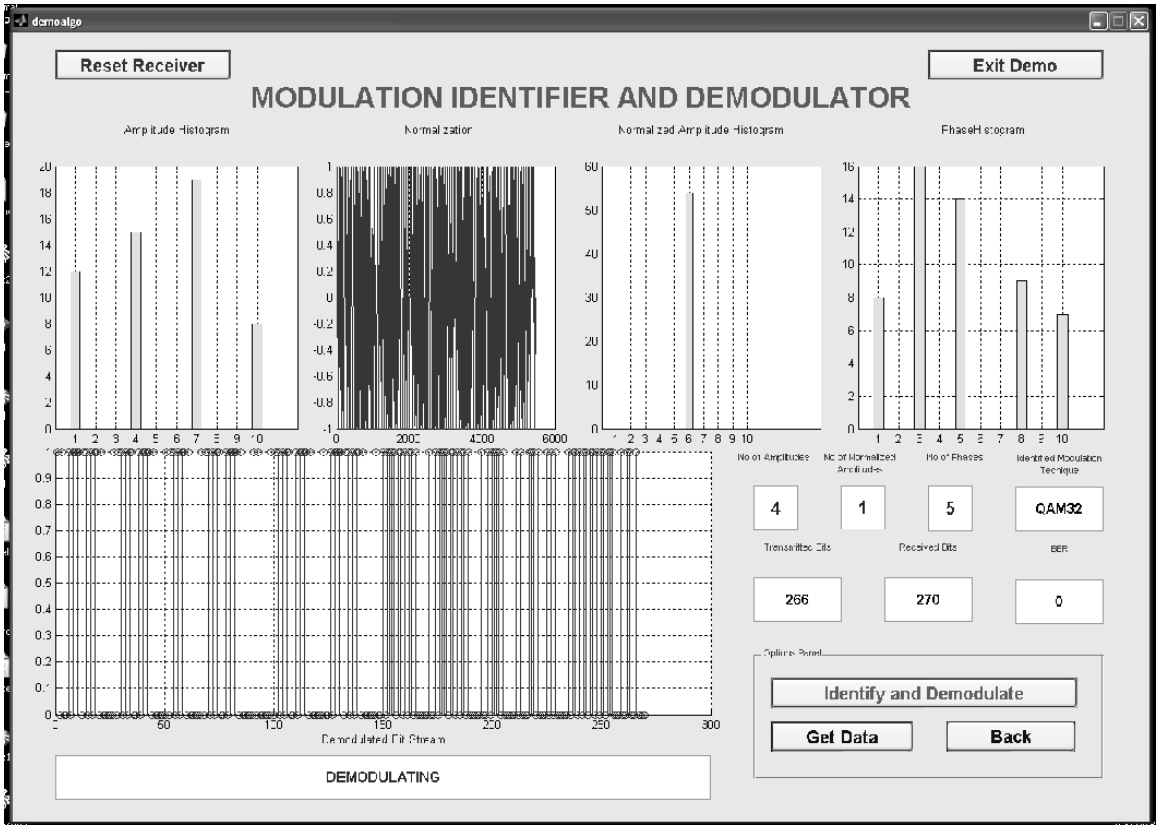

Figure 5.l. demoalgorithm figure for GUI

STEP 11:

Once demodulated press *'GET DATA'* to see which data type was transmitted and what was transmitted. See figure 5.m for figure illustration. All you need to do is to press *SHOW'.*

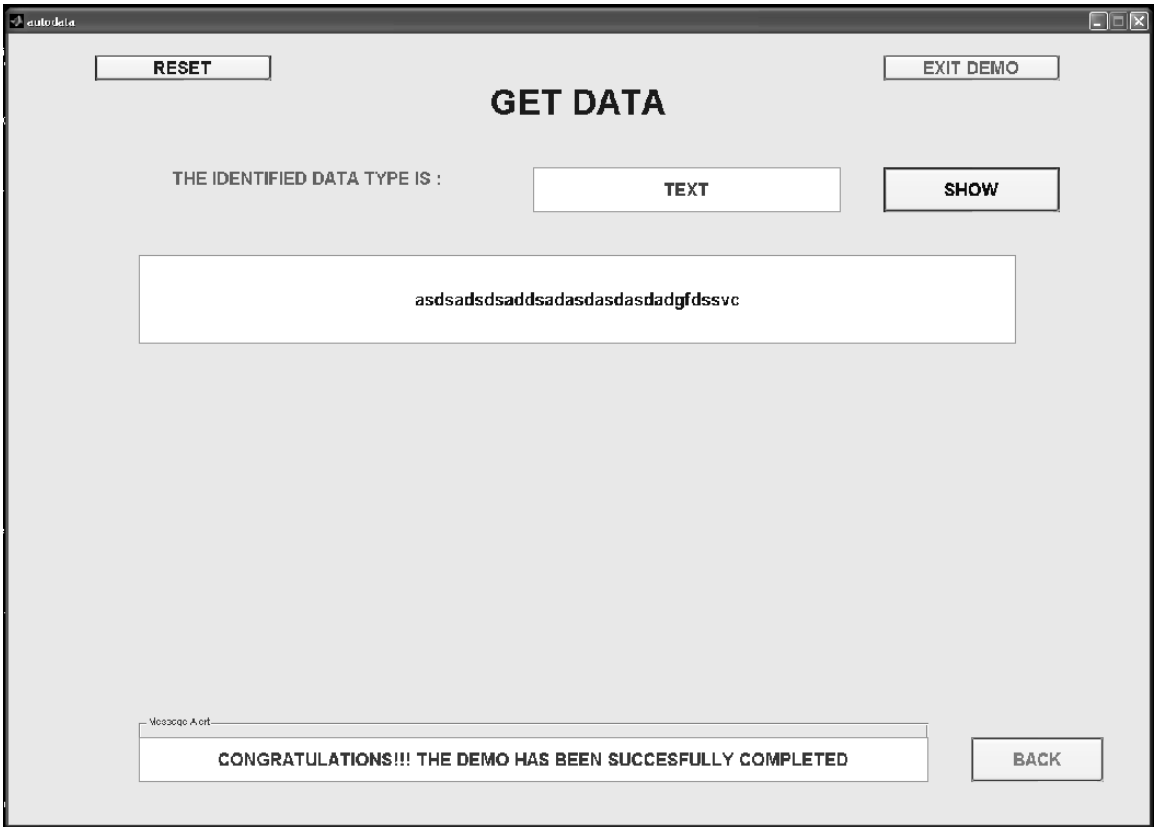

Figure 5.m. figure showing the transmitted data

NOTE:

A *'Message Alert'* appears in almost all the figures of the GUI during the demonstration of the project in the bottom of the axes. If anything goes wrong or a wrong button is pressed, a warning would appear in the area. The user is required to act according to the warning that appears in the area. Else the identification may have certain problems and may go wrong.

CHAPTER 6

SIMULATION AND RESULTS

INTRODUCTION

This chapter will show the working of the project as has been implemented during its course time. The working has been explained by taking one of the modulation techniques (QAM 16) as an example. On the completion of this chapter, the reader shall have in-depth understanding of the working of the project, its simulation and the results. This chapter has the following objectives: -

6.1 Steps of Simulation.
6.2 Simulation of Source and its Results.
6.3 Simulation of modulators and its Results.
6.4 Simulation of Demodulators and its Results.
6.5 Simulation of Modulation Identification Algorithm and its results.

6.1 STEPS OF SIMULATION

The flow of the simulation goes in a number of steps. Each step is important at its stage and hence cannot be omitted or overseen no matter which so ever be the modulation technique being used. The sequential flow is shown in figure 6.a. These steps are as follows: -

1. Source

The source may be of one of the following four types: -

a. Data (Text).
b. Image.
c. Audio.
d. Video.

2. A-to-D Conversion

The source has to be digitized i.e. converted to binary bits as is the requirement for digital modulation to occur. We have been able to convert the first three sources' types to binary bits and are working on the fourth one. The binary stream thus generated is saved for BER calculation after demodulation has occurred for confirmation of complete data recovery. It is to be made clear that certain headers are used to identify the source type for its recovery at the sink. For simplicity the headers are listed in table 6.1 below.

Table 6.1. Headers for Source type recognition

Source Type	Header
Data	00
Image	01
Audio	10
Video	11

3. Modulation

The binary stream is fed to all the 11 modulators as input, where the respective modulations take place. One of the modulator's output is chosen and the output is virtually transmitted

4. Receive

The virtually transmitted wave is virtually received with no channel effects, since the channel has been considered as ideal channel. Hence no attenuation or distortion occurs.

5. Amplitude Histogram

After receiving the wave, the first step at the receiver is the *'Modulation Identification'*. For this we start with plotting the instantaneous amplitude histogram. This would give us the number if instantaneous amplitudes. This is our first identification parameter. The values of these amplitudes are calculated for use at the demodulator for demodulation.

6. Sort Amplitude Array

The values of instantaneous amplitude calculated in step 5 are sorted in an ascending order. This is the shape of the amplitude array that is useable at the demodulator for demodulation.

7. Amplitude Normalization

The received signal is fed to the amplitude normalizing module that normalizes every amplitude to unity. After normalization has occurred, the wave has constant '1' amplitude for a all modulation except M-ASK. Hence resulting in one single amplitude. For M-ASK, all the amplitudes except '0' are normalized to unity. '0' amplitude however remain undisturbed at '0'.

8. Normalized Amplitude Histogram

Again after normalization, we plot the amplitude histogram. This gives is the number of normalized amplitudes. This is our second identification parameter. The amplitude normalized waveform is the input.

9. Phase Histogram

Here again the normalized amplitude waveform is the input. This module plots the instantaneous phase histogram. This plot gives us the number of instantaneous phases that becomes our third identification parameter.

10. Phase Values

In this module the number of instantaneous phases is taken as input from step 9. This gives us the array of values of the phases in ascending order, which is used for demodulation.

11. Identification of Modulation Technique

As shown in figure 3.1 and 3.2, we have two ways in which the algorithm identifies the modulation technique. For the first case the techniques are identified at various stages. However in case 2, value from step 5, 8, 9 and 11 are taken and modulation is identified then.

12. Demodulation

The received waveform, values from step 6, 10 and 11 are fed to that demodulator whose modulation technique has been identified during the identification process. The output is the data stream in terms of ‘0’ and ‘1’.

13. BER Calculation

The BER is calculated by using the output of step 2 and step 13. Since the channel is ideal, so the BER must always be 0 %, i.e. the ideal case.

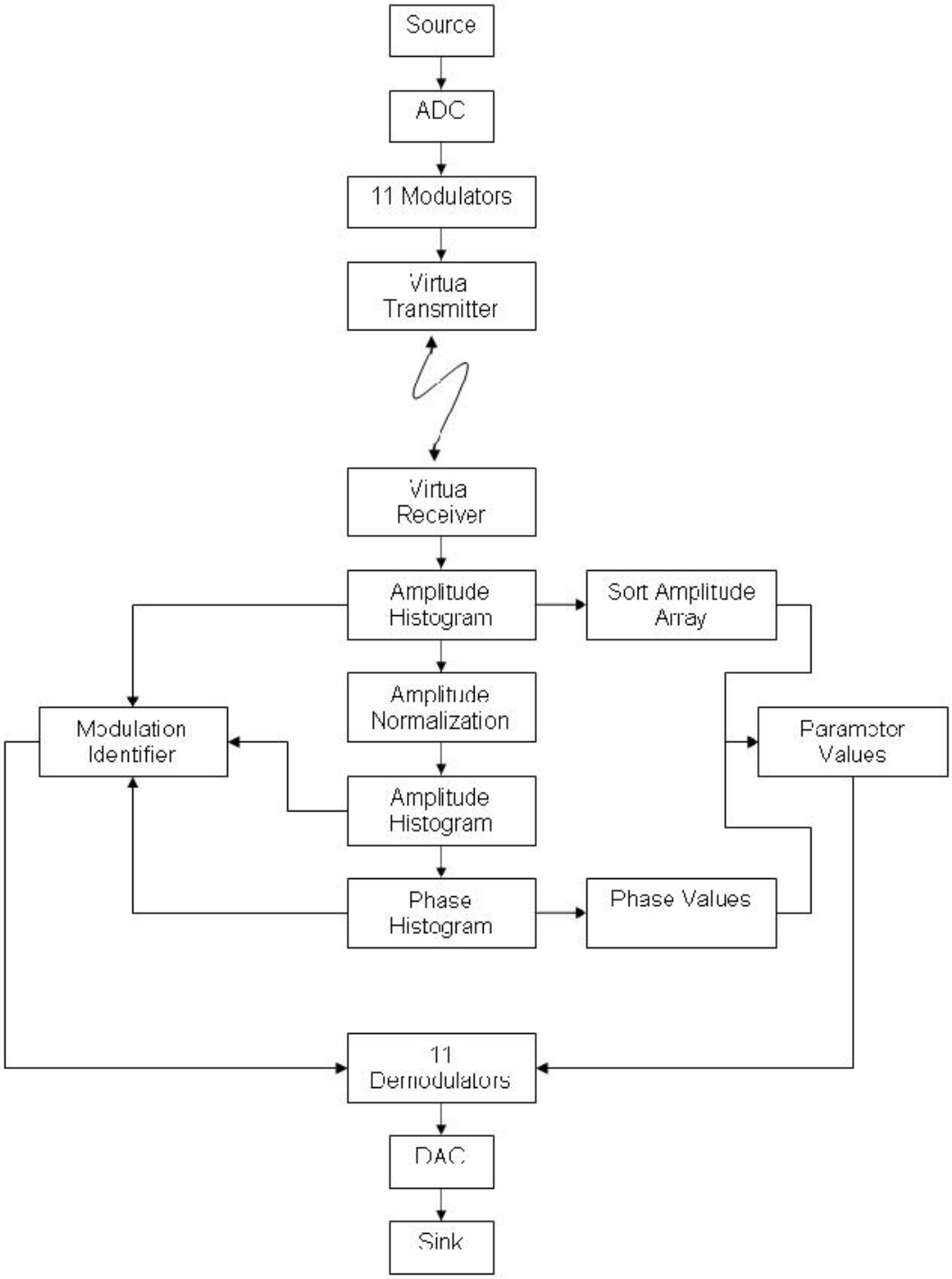

Figure 6.a. Steps of Simulation

14. D-to-A Conversion

The header identifies the source type that was transmitted. Once identified with the source modulation using the header, the demodulated bit stream is

converted back to the source type by D-to-A conversion. After this the actual shape of the data is retrieved.

6.2 SIMULATION OF SOURCE AND ITS RESULTS

Depending on the source type, the conversion to binary bits takes place. For data as an example, the text is taken as the input from the user. The string thus entered is converted to the respective ASCII codes, which on conversion would give us the binary bits. Figure 6.b. below shows the ASCII codes and the binary stream then generated for the character *'identification'*.

```
>> double( identification )

ans =

   105   100   101   110   116   105   102   105    99    97   116   105   111   110

>> str2bin('identification')

ans =

  Columns 1 through 28

    1    0    1    0    0    1    0    1    1    0    1    0    0    0    0    0    1    0    1    0    0    0    1    1    0    1    1    1

  Columns 29 through 56

    0    0    0    0    1    0    1    1    0    1    1    0    1    0    1    0    0    1    0    1    1    0    1    0    0    0    1    0

  Columns 57 through 84

    1    0    1    0    0    1    0    1    1    0    0    1    1    0    0    1    1    0    0    1    0    1    1    1    1    0    1    1

  Columns 85 through 112

    0    1    1    0    1    0    1    0    0    1    0    1    1    0    1    1    0    0    0    1    1    0    1    1    0    0    0    0

>>
```

Figure 6.b. 'Identification', ASCII Codes and Binary Bits

6.2 SIMULATION OF MODULATORS AND ITS RESULTS

To explain form here onwards, we shall take an example of one of the modulation techniques, say QAM-16. The input to the modulator is the bit steam generated by the source. The file for the simulation of QAM-16 modulation is *'qam16_mod.m'*. The first step for QAM-16 is the conversion of serial data to parallel data, since the symbol for QAM-16 constitutes 4 bits each, resulting in a total of 16 possible symbols. Hence a bank of 112 bit is converted to a bank of 28 symbols only.

The second step is the generation of respective symbol waveforms. 16 sinusoidal waveforms, one for each symbol, are generated. The symbol number, the symbol formation and the assigned waveform for QAM-16 are listed in table 6.1.

As and when the symbol is formed one of the 16 sinusoidal waveforms is assigned to it. The waves are then accumulated till the time all the data has been assigned one waveform or the other. Once accumulated, the modulation is complete and the wave is virtually transmitted.

Here we have two sets of data for plotting. One is the input bit stream plot shown in figure 6.c and the second is the output waveform plot, shown in figure 6.d, below.

Table 6.1. Symbol and Assigned Wave for QAM-16

Decimal	Symbol Formation	Assigned Waveform
0	0000	Amp(1)*sin((w*t)+Phase(1))
1	0001	Amp(1)*sin((w*t)+Phase(2))
2	0010	Amp(1)*sin((w*t)+Phase(3))
3	0011	Amp(1)*sin((w*t)+Phase(4))
4	0100	Amp(1)*sin((w*t)+Phase(5))
5	0101	Amp(1)*sin((w*t)+Phase(6))
6	0110	Amp(1)*sin((w*t)+Phase(7))
7	0111	Amp(1)*sin((w*t)+Phase(8))
8	1000	Amp(2)*sin((w*t)+Phase(1))
9	1001	Amp(2)*sin((w*t)+Phase(2))
10	1010	Amp(2)*sin((w*t)+Phase(3))
11	1011	Amp(2)*sin((w*t)+Phase(4))
12	1100	Amp(2)*sin((w*t)+Phase(5))
13	1101	Amp(2)*sin((w*t)+Phase(6))
14	1110	Amp(2)*sin((w*t)+Phase(7))
15	1111	Amp(2)*sin((w*t)+Phase(8))

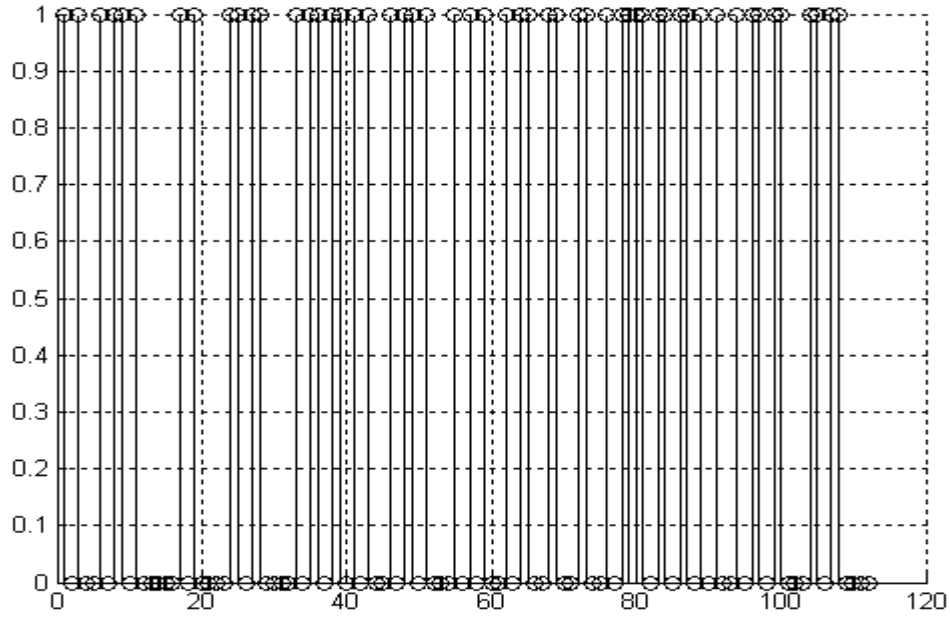

Figure 6.c. Input bit stream

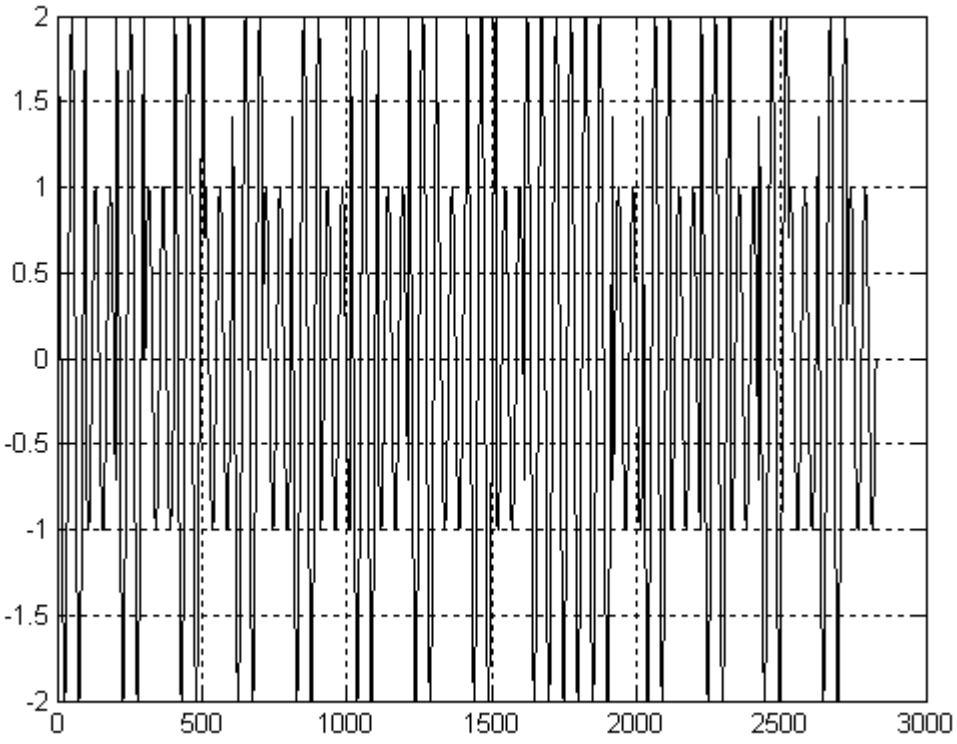

Figure 6.d. Output Waveform for QAM-16 Modulator

6.4 SIMULATION OF DEMODULATORS AND ITS RESULTS

The identification is a long process and shall be described in section 6.5 later. However at the demodulator the inputs are: -

1. Modulator's Output.
2. Amplitude Array.
3. Phase Array.

The inputs excluding the first one are obtained at the receiver and during the identification process of the modulation technique. At the demodulator, the modulator's output is accessed symbol by symbol. For each symbol duration, we have 101 data points. So each of the 101 data points are checked point-by-point for its correspondence with all the 16 waveforms. If all the data points correspond to one of the 16 waveforms, the corresponding symbol is taken as the data been carried by that wave. However, in certain cases if 100 out 0f 101 data points correspond correctly to either of the waves, then the corresponding symbol is taken as the data carried. This is so because in certain techniques the first data point of the symbol duration is 0 and is taken as a part of previous data set, resulting in 100 data points for the next one. So we confine ourselves to this and take the respective symbol as the data carried. Here all the waves are checked and the data is accumulated symbol by symbol. Here we do not need the parallel to serial conversion of the data demodulated, because the obtained shape of the data is already the bit stream. Here, we have only of set of data for plot, which is the demodulated bits stream, shown in figure 6.e, below. The BER is calculated for confirmation of recovery of complete data. The demodulated bit stream is converted back to the source type.

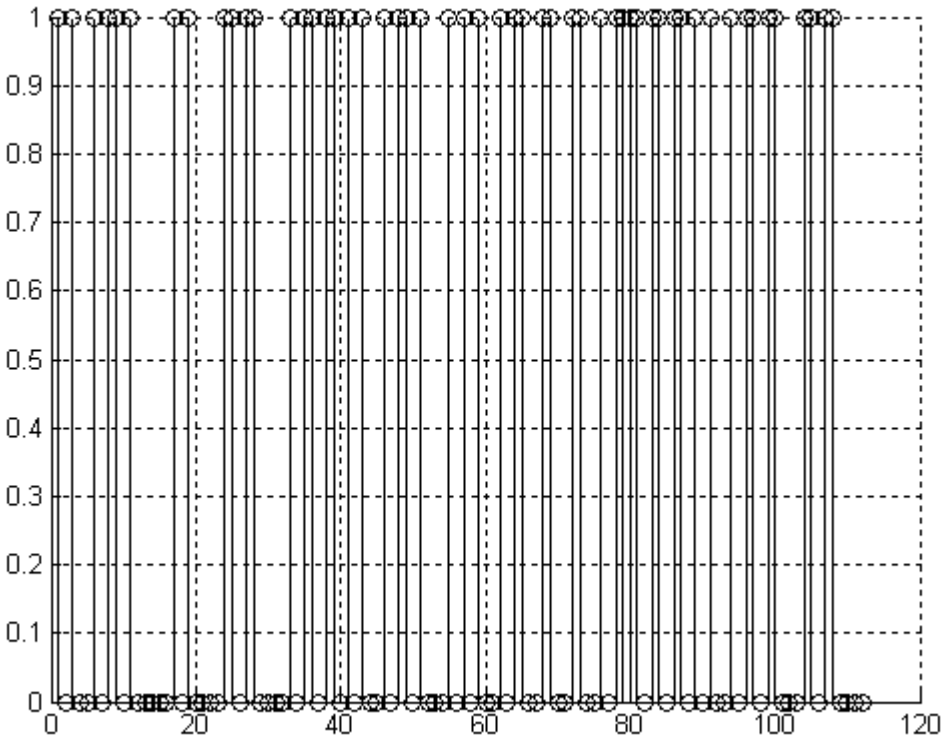

Figure 6.e. Demodulated Stream for QAM-16 Demodulator

6.5 SIMULATION OF MODULATION IDENTIFICATION ALGORITHM AND ITS RESULTS

The modulation identification algorithm has *five* stages as mentioned below. They are explained one-by-one, then.

1. Amplitude Histogram.
2. Amplitude Normalization.
3. Normalized Amplitude Histogram.
4. Phase Histogram.
5. Identification.

1. The input to the identifier is the modulator's output only. For the first step, it is fed to *'AmpHist.m'*. This m file plots the amplitude histogram and gives us the number of instantaneous amplitudes. The input is divided symbol by symbol and is checked for maximum value. This gives us the amplitude of the symbol. The amplitude is rounded off to the nearest integer, since it is assumed that the amplitudes can be integer only. This is done for each symbol. The values are stored separately. The amplitude histogram for QAM-16, which has been taken as an example for explanation, is shown in figure 6.f, below. The values of the

amplitudes are stored in ascending order to be used for demodulation. The *'count'* is our first identification parameter.

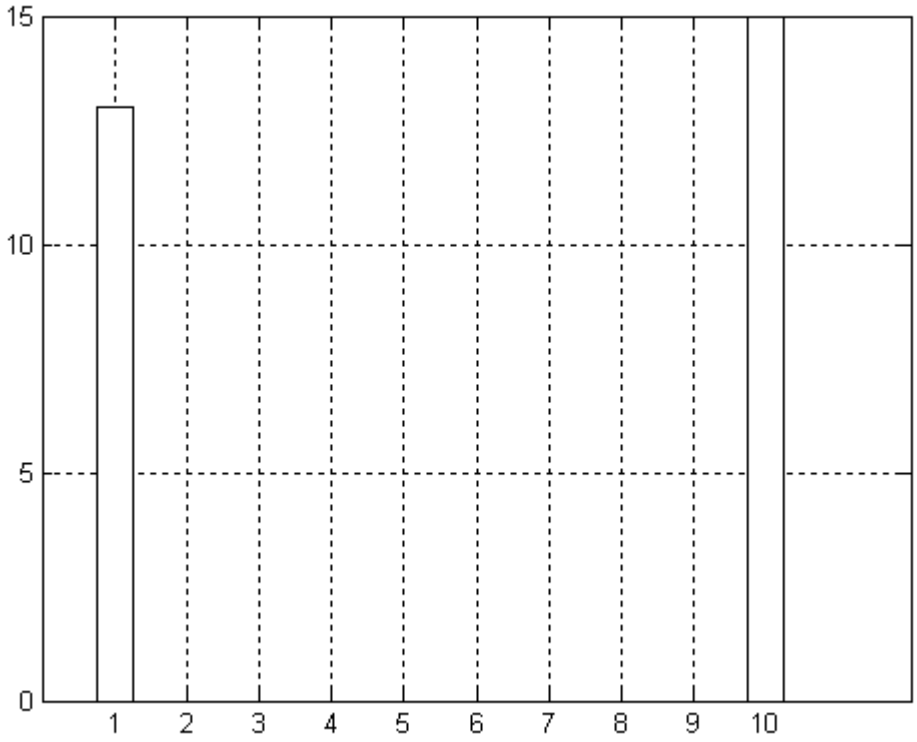

Figure 6.f Amplitude Histogram for QAM-16

2. The next step ts the amplitude normalization. This process is undertaken by *'amp_norm.m'* that takes the modulators output as the input. This is also undertaken symbol by symbol. The amplitude for each symbol is calculated and whole symbol data points are divided by it. This will give us a normalized output. Here one thing is to be kept in mind, that a zero amplitude is not normalized to one, since it helps in identification between ASK2 and QAM, mainly. The zero amplitude normalization can neither be done because 0/0 is calculated to be NaN 1 by Matlab. Hence a calculation restriction. This waveform is used for further processing. The normalized output waveform for QAM-16 is shown in figure 6.g, below.

---

1. NaN – Not a number

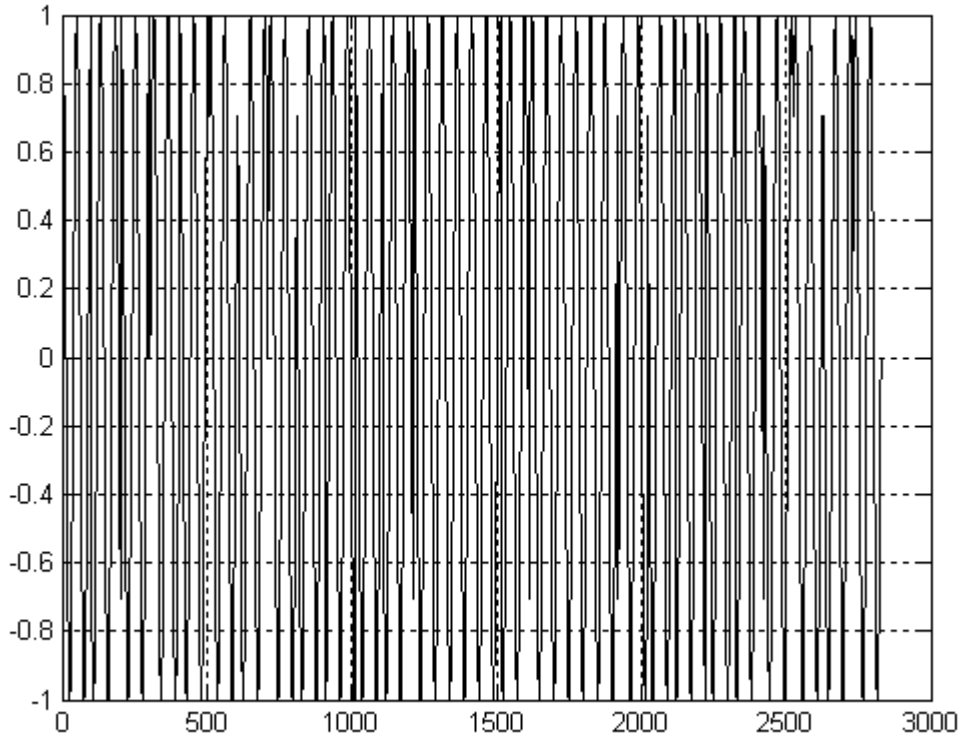

Figure 6.g. Normalized Output of QAM-16

3. The next step is the Normalized Amplitude Histogram. The normalized amplitude waveform is fed as the input to *'amp_norm.m'* and the amplitude histogram is plotted much in the same way as is done in step 1. Here the difference is that the values of the amplitudes are of no use to us and hence are not stored. But the count is stored, which is our second identification parameter. The normalized amplitude histogram for QAM-16 is shown in figure 6.h, below.

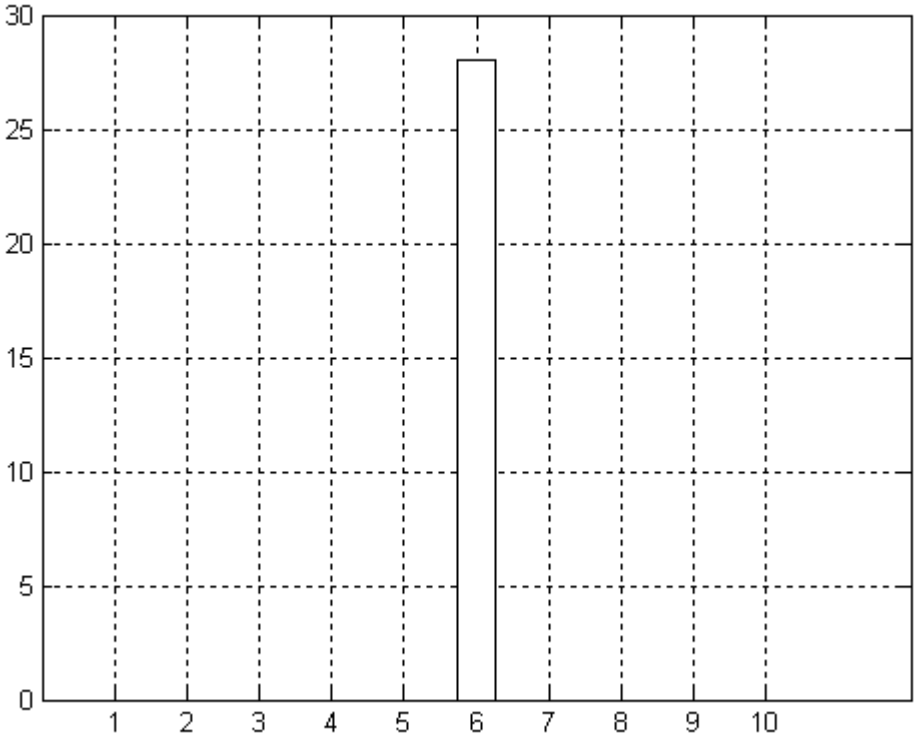

Figure 6.h. Normalized Amplitude Histogram for QAM-16

4. The fourth step is the Phase Histogram that uses *'PhaseDetect.m'*. The normalized amplitude waveform is fed as the input. Like for amplitude histogram, the waveform is accessed symbol by symbol. The phase for each symbol is calculated. The values are not stored due to calculation restriction. However, the count is stored, which is our third identification parameter. The calculation restriction is some what the calculation of two angles as same. This is well illustrated in the following equation 6.1 and the figure 6.i, below.

$$\begin{aligned} \sin(45) &= \sin(135) \quad \Rightarrow 45 = 135 \\ \sin(315) &= \sin(225) \quad \Rightarrow 225 = 315 \\ \sin(0) &= \sin(180) \quad \Rightarrow 0 = 180 \end{aligned}$$

..... EQ 6.1

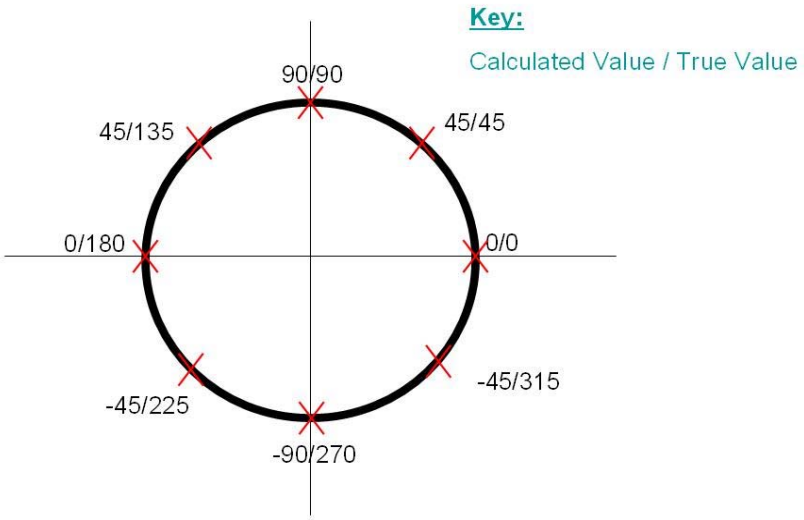

Figure 6.i. Explanatory diagram for Calculation Restriction of Phase

The phase histogram is obtained, as shown in figure 6.j, below. The phase count obtained is used to get the array of he phases in ascending order from *'Phase.m'*.

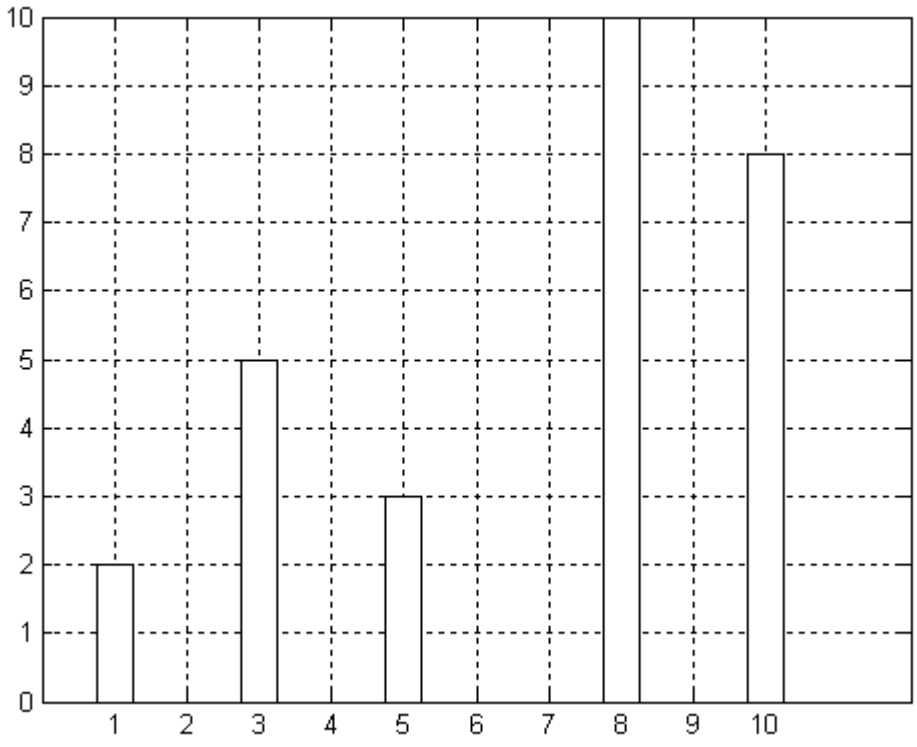

Figure 6.j. Phase Histogram for QAM-16

5. The last step is identification module. Here either of the two ways of identification is followed, which is shown in figure 4.a and 4.b. The comparison between the two has also been short listed in table 4.1. Depending on the usage and application any of the methods may be used for the identification purpose.

CONCLUSION

We, in our project, have developed and simulated a new algorithm for digital modulation identification in ideal channel conditions. This algorithm may be classified in the statistical method category. It has been totally our thinking and work that has led us to develop such an algorithm that is based truly on the modulating parameters. Hence, it could be said that any modulation technique can be identified through this algorithm, as long as the modulating parameters of the modulation technique can be extracted from the received signal. Our algorithm has been termed as ***'Histogram - Count Modulation Identification' (HCMI)*** method. The identified demodulator demodulates the data into bit streams. The data is further identified as whether to be ***image, audio or text,*** and is thus brought back to its original shape, via respective converters.

With its wide ***applications*** in the military and commercial sector, the project module can be applied in the following fields: -

- ECCM.
- ECM.
- SDR.
- MHTS.
- Digital Receivers.
- Stenography.

FUTURE RECOMMENDATIONS

Added to it, we have proposed a new concept of secure communication, which is the ***Modulation Hopping Transceiver System (MHTS)***. In this the modulation technique would change many times a unit time, along with other security devices like ciphering devices etc.

The development and improvement areas are as follows: -

1. Improvement in algorithm.
2. Development of encryption module for MHTS.
3. Development of frequency hopping module for MHTS.
4. Interfacing of modules.
5. Hardware implementation.

We are really thankful to ***ALMIGHTY ALLAH,*** our ***FAMILIES***, our ***FRIENDS*** and all those who have helped us to attain what we have attained.

REFERENCES / BIBLIOGRAPHY

BOOKS

[1] Digital Communication Systems, Second Edition by Bernard Sklar
[2] Modern Digital and Analog Communication Systems by B.P. Lathi
[3] Signal and Systems, Second Edition by A.V. Oppenheim and A.V. Willsky
[4] Discrete Time Signal Processing by A.V. Oppenheim and P.W. Sehaffer
[5] Automatic Modulation Recognition Algorithms by E.E. Azzouz and A.K. Nandi

RESEARCH PAPERS

[6] E.E. Azzouz and A.K. Nandi "*Automatic Modulation Identification of Digital Modulation Types"* Signal processing, vol.47, no.1, pp.55-69, November 1995

[7] B.F. Beidas and C.L. Weber, "Higher-*order correlation based approach to modulation classification of digitally frequency-modulated signal,"* IEEE journal on Selected areas in communication vol.13,no.1,pp.89-101,jaunary 1995.

[8] Char-Dir Chung and A. Polydoros, "*Envelope based classification schemes for continuous-phase binary frequency-shift-keyed modulations*" in proceeding of MILCOM'94,Fort Monmouth ,NJ,USA , October 1994,vol. 3,pp. 796 -800

[9] S.-Z. Hsue and S.S Soliman,"*Automatic modulation classification using zero crossing"* IEEE Proceedings F(Radar and Signal Processing), vol. 137,no .6, pp.459-64,December 1990.

[10] Chung –Yu Huang and A. polydoros,"*Likelyhood methods for MPSK modulation classification,*" IEEE Transaction on Communications, vol. 43, no. 2-4, pp.1493-504, February 1995.

[11] A.K Nandi and E.E Azzouz, "*Automatic analog modulation recognition,"* Signal processing vol. 46, no. 2, pp.211-22, October 1995.

[12] A.K Nandi and E.E Azzouz, *"Modulation recognition using artificial neural networks*" Signal processing, vol. 56, no. 2, pp.165-75, January 1997.

[13] A .Polydoros and K.Kim, "*On the detection and classification of quadrature digital modulation in broadband noise,"* IEEE Transactions on Communications,vol. 38,no.8,pp.1190-211,August 1990.

[14] S.Z Hsue and S.S Soliman,*"Automatic modulation classification using zero crossing,*" in IEEE proceedings F (Radar and Signal processing),vol.137,no.6,pp.459-464,December 1990.

[15] B.F Beidas and C.L.Weber, "*Higher order correlation based approach to modulation classification of digitally modulated signals.*" In IEEE journal on selected area in Communications, vol. 13, no.1, pp.89-101, January 1995.

[16] Alfred O. Hero III and Hafez Hadinjed Mahram, "*Digital Modulation classification and power moment matrices,*" in proceedings IEEE ICASSP-98,May 1998.
[17] Xiaming Huo and David Donoho,"*A Simple and robust Modulation classification method via counting,*" in proceeding IEEE ICASSP-98, May 1998.
[18] A. Polydoros and K.Kim, "*On the detection and classification of Quadrature digital modulation in broad-band noise,*" in IEEE transaction on communication, vol. 38, no.8, pp.1199-1211, August 1990.

APPENDIX 'A'

MATLAB CODES

CODE FOR ASK2 MODULATOR

```
function [ASK2Output]=ask2_mod(Data)

% ----------- MODULATION PARAMETERS -----------------

DataLength=length(Data);
Freq=2;
Amp=[0 1];
t=0:0.01:1;
% bit duration
T=length(t);
w=2*pi*Freq;
% obtained angular frequency

% ------------ WAVES GENERATED FOR MODULATION ---------

x1=Amp(1)*sin(w*t);              % wave for bit '1'
x2=Amp(2)*sin(w*t);              % wave for bit '0'

% ---------- MODULATION --------------------

ASK2Output=[];   % initializing output of the transmitter
for i=1:DataLength       % assigning waves to the bits
    switch Data(i)
        case 1
            ASK2Output=[ASK2Output,x1];
        case 0
            ASK2Output=[ASK2Output,x2];
    end
end

% ------------- OUTPUT WAVEFORM THAT IS TRANSMITTED -----

% plot(ASK2Output,'r'); % output waveform of the modulator
% grid;
```

CODE FOR ASK2 DEMODULATOR

```
function [outputd]=ask2_demod(outputm,amp,P)

% ------------ KNOWN PARAMETERS ------------

t=0:0.01:1;
T=length(t);
```

```
Freq=2;

% ------------- CALCULATED PARAMETERS --------------

datalen=length(outputm);   % total data points
pw=datalen/T;              % duration data points
w=2*pi*Freq;               % obtained angular frequency

% ------------- DEMODULATION ------------------

outputd=[];                % initializing bit stream
count=0;                   % initializing step count
for i=1:pw                 % loop for next bit duration
    bit0=0;                % calculation of amplitude 0
    bit1=0;                % calculation of amplitude 1
    for j=1:T
        switch outputm(count+j)
            case amp(1)*sin(w*t(j))     % for bit '1'
                bit0=bit0+1;   % addition of amplitude zero
            case amp(2)*sin(w*t(j))     % for bit '0'
                bit1=bit1+1;   % addition of amplitude one
        end
    end
    count=count+T;             % count added with number of
                               data points in one duration
    if bit1==T-1      % first point of sine wave is also 0
        outputd=[outputd,0];   % accumulation of bit '0'
    elseif bit0==T
        outputd=[outputd,1];   % accumulation of bit '1'
    end
    stem(outputd,'r');
    grid;
    drawnow;
end
```

CODE FOR ASK4 MODULATOR

```
function [ASK4Output]=ask4_mod(Data)

% ------------ MODULATION PARAMETERS --------------

DataLength=length(Data);
Phase=0;
Freq=2;
Amp=[0 1 2 3];
```

```
t=0:0.01:1;
% bit duration
T=length(t);
w=2*pi*Freq;            % obtained angular frequency

% ----------- WAVES GENERATED FOR MODULATION -------------

x1=Amp(1)*(sin((w*t)+Phase));     % wave for symbol '00'
x2=Amp(2)*(sin((w*t)+Phase));     % wave for symbol '01'
x3=Amp(3)*(sin((w*t)+Phase));     % wave for symbol '10'
x4=Amp(4)*(sin((w*t)+Phase));     % wave for symbol '11'

% --- SERIAL TO PARALLEL CONVERSION (SYMBOL FORMATION) ---

sym=2;
z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);

% --------------- MODULATION ----------------------

ASK4Output=[];            % initializing empty vectors
for i=1:n                 % assigning waves to the symbols
    if z([1,2],i)'==[0 0]
        ASK4Output=[ASK4Output,x1];
    elseif z([1,2],i)'==[0 1]
        ASK4Output=[ASK4Output,x2];
    elseif z([1,2],i)'==[1 0]
        ASK4Output=[ASK4Output,x3];
    elseif z([1,2],i)'==[1 1]
        ASK4Output=[ASK4Output,x4];
    end
end

% ------ OUTPUT WAVEFORM THAT IS TRANSMITTED ----------

% plot(ASK4Output,'r'); % output waveform of the modulator
% grid;
```

CODE FOR ASK4 DEMODULATOR

```
function [outputd]=ask4_demod(outputm,amp,P)

% ------------ KNOWN PARAMETERS ----------------

t=0:0.01:1;
T=length(t);
```

```
Freq=2;

% ------------ CALCULATED PARAMETERS -------------

datalen=length(outputm);      % total data points
pw=datalen/T;                 % duration data points
w=2*pi*Freq;                  % obtained angular frequency

% ------------ DEMODULATION ----------------------

outputd=[];                  % initializing bit stream
count=0;                     % initializing step count
for i=1:pw                   % loop for next bit duration
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    for j=1:T
        if outputm(count+j)==amp(1)*sin(w*t(j))
                                    % for symbol '00'
            bit0=bit0+1;
        elseif outputm(count+j)==amp(2)*sin(w*t(j))
                                    % for symbol '01'
            bit1=bit1+1;
        elseif outputm(count+j)==amp(3)*sin(w*t(j))
                                    % for symbol '10'
            bit2=bit2+1;
        elseif outputm(count+j)==amp(4)*sin(w*t(j))
                                    % for symbol '11'
            bit3=bit3+1;
        end
    end

    count=count+T;      % count added with number of data
                          points in one duration
    if bit0==T
        outputd=[outputd,0 0];% accumulation of symbol '00'
    elseif bit1==T-1   % first point of sine wave is also 0
        outputd=[outputd,0 1];% accumulation of symbol '01'
    elseif bit2==T-1   % first point of sine wave is also 0
        outputd=[outputd,1 0];% accumulation of symbol '10'
    elseif bit3==T-1   % first point of sine wave is also 0
        outputd=[outputd,1 1];% accumulation of symbol '11'
    end
    stem(outputd,'r');
    grid;
    drawnow;
```

```
end
```

CODE FOR ASK8 MODULATOR

```
function [ASK8Output]=ask8_mod(Data)

% -------------- MODULATION PARAMETERS ----------------

DataLength=length(Data);
Phase=0;
Freq=2;
Amp=[0 1 2 3 4 5 6 7];
t=0:0.01:1;                 % bit duration
T=length(t);
w=2*pi*Freq;                % obtained angular frequency

% ----------- WAVES GENERATED FOR MODULATION --------

x1=Amp(1)*(sin((w*t)+Phase));      % wave for symbol '000'
x2=Amp(2)*(sin((w*t)+Phase));      % wave for symbol '001'
x3=Amp(3)*(sin((w*t)+Phase));      % wave for symbol '010'
x4=Amp(4)*(sin((w*t)+Phase));      % wave for symbol '011'
x5=Amp(5)*(sin((w*t)+Phase));      % wave for symbol '100'
x6=Amp(6)*(sin((w*t)+Phase));      % wave for symbol '101'
x7=Amp(7)*(sin((w*t)+Phase));      % wave for symbol '110'
x8=Amp(8)*(sin((w*t)+Phase));      % wave for symbol '111'

% --- SERIAL TO PARALLEL CONVERSION (SYMBOL FORMATION) ---

sym=3;
z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);

% --------- MODULATION ----------

ASK8Output=[];              % initializing empty vectors
for i=1:n
    if z([1,2,3],i)'==[0 0 0]
        ASK8Output=[ASK8Output,x1];
    elseif z([1,2,3],i)'==[0 0 1]
        ASK8Output=[ASK8Output,x2];
    elseif z([1,2,3],i)'==[0 1 0]
        ASK8Output=[ASK8Output,x3];
    elseif z([1,2,3],i)'==[0 1 1]
        ASK8Output=[ASK8Output,x4];
    elseif z([1,2,3],i)'==[1 0 0]
```

```
        ASK8Output=[ASK8Output,x5];
    elseif z([1,2,3],i)'==[1 0 1]
        ASK8Output=[ASK8Output,x6];
    elseif z([1,2,3],i)'==[1 1 0]
        ASK8Output=[ASK8Output,x7];
    elseif z([1,2,3],i)'==[1 1 1]
        ASK8Output=[ASK8Output,x8];
    end
end

% --------- OUTPUT WAVEFORM THAT IS TRANSMITTED --------

% plot(ASK8Output,'r');% output waveform of the modulator
% grid;
```

CODE FOR ASK8 DEMODULATOR

```
function [outputd]=ask8_demod(outputm,amp,P)

% ------------------ KNOWN PARAMETERS -------------

t=0:0.01:1;
T=length(t);
Freq=2;

% ----------- CALCULATED PARAMETERS ---------------

datalen=length(outputm);     % total data points
pw=datalen/T;                % duration data points
w=2*pi*Freq;                 % obtained angular frequency

% ------------ DEMODULATION --------------------

outputd=[];                  % initializing bit stream
count=0;                     % initializing step count
for i=1:pw                   % loop for next bit duration
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    bit4=0;
    bit5=0;
    bit6=0;
    bit7=0;
    for j=1:T
```

```
    if outputm(count+j)==amp(1)*sin(w*t(j))
                                    % for symbol '000'
        bit0=bit0+1;
    elseif outputm(count+j)==amp(2)*sin(w*t(j))
                                    % for symbol '001'
        bit1=bit1+1;
    elseif outputm(count+j)==amp(3)*sin(w*t(j))
                                    % for symbol '010'
        bit2=bit2+1;
    elseif outputm(count+j)==amp(4)*sin(w*t(j))
                                    % for symbol '011'
        bit3=bit3+1;
    elseif outputm(count+j)==amp(5)*sin(w*t(j))
                                    % for symbol '100'
        bit4=bit4+1;
    elseif outputm(count+j)==amp(6)*sin(w*t(j))
                                    % for symbol '101'
        bit5=bit5+1;
    elseif outputm(count+j)==amp(7)*sin(w*t(j))
                                    % for symbol '110'
        bit6=bit6+1;
    elseif outputm(count+j)==amp(8)*sin(w*t(j))
                                    % for symbol '111'
        bit7=bit7+1;
    end
end
count=count+T;       % count added with number of data
                     points in one duration
if bit0==T
    outputd=[outputd,0 0 0];    % accumulation of
                                 symbol '000'
elseif bit1==T-1  % first point of sine wave is also 0
    outputd=[outputd,0 0 1];    % accumulation of
                                 symbol '001'
elseif bit2==T-1  % first point of sine wave is also 0
    outputd=[outputd,0 1 0];    % accumulation of
                                 symbol '010'
elseif bit3==T-1  % first point of sine wave is also 0
    outputd=[outputd,0 1 1];    % accumulation of
                                 symbol '011'
elseif bit4==T-1  % first point of sine wave is also 0
    outputd=[outputd,1 0 0];    % accumulation of
                                 symbol '000'
elseif bit5==T-1  % first point of sine wave is also 0
    outputd=[outputd,1 0 1];    % accumulation of
                                 symbol '001'
elseif bit6==T-1  % first point of sine wave is also 0
```

```
        outputd=[outputd,1 1 0];    % accumulation of
                                     symbol '010'
    elseif bit7==T-1  % first point of sine wave is also 0
        outputd=[outputd,1 1 1];    % accumulation of
                                     symbol '011'
    end
    stem(outputd,'r');
    grid;
    drawnow;
end
```

CODE FOR PSK2 MODULATOR

```
function [PSK2Output]=psk2_mod(Data)

% ------------ MODULATION PARAMETERS ----------------

DataLength=length(Data);
Freq=2;
Amp=1;
Phase=[0 pi];
t=0:0.01:1;                % bit duration
T=length(t);
w=2*pi*Freq;               % obtained angular frequency

% -------- WAVES GENERATED FOR MODULATION ------------

x1=Amp*(sin((w*t)+Phase(1)));         % wave for bit '1'
x2=Amp*(sin((w*t)+Phase(2)));         % wave for bit '0'

% ------------ MODULATION ----------------

PSK2Output=[];  % initialization output of the transmitter
for j=1:DataLength          % assigning waves to the bits
    if Data(j)==1;
        PSK2Output=[PSK2Output,x1];
    elseif Data(j)==0;
        PSK2Output=[PSK2Output,x2];
    end
end

% ----------- OUTPUT WAVEFORM THAT IS TRANSMITTED --------

% plot(PSK2Output,'r');     % output waveform of modulator
% grid;
```

CODE FOR PSK2 DEMODULATOR

```
function [output_demod]=psk2_demod(output_mod,amp,phase)

% ------------------ KNOWN DEMODULATION PARAMETERS ------

freq=2;                          % intermediate frequency
t=0:0.01:1;                      % bit duration
T=length(t);

% --------- CALCULATED DEMODULATION PARMETERS ------------

datalen=length(output_mod);    % total data points
pw=datalen/T;                  % duration data points
w=2*pi*freq;                   % obtained angular frequency

% ------------------ DEMODULATION -------------------

output_demod=[];               % initializing bit stream
count=0;                       % step count
for i=1:pw                     % loop for next bit duration
    bit0=0;                % calculation of points for '0'
    bit1=0;                % calculation of points for '1'
    for j=1:T
        if output_mod(count+j)==amp*sin((w*t(j))+phase(1))
                               % for bit '1'
            bit1=bit1+1;
        elseif output_mod(count+j)==amp*sin((w*t(j))+
                               phase(2)) % for bit '0'
            bit0=bit0+1;
        end
    end
    count=count+T;
% count added with number of data points in one duration
    if bit0==T
        output_demod=[output_demod,0];       % accumulation
                                             of bit '0'
    elseif bit1==T
        output_demod=[output_demod,1];       % accumulation
                                             of bit '1'
    end
    stem(output_demod,'r');        % plot of output of PSK
                                   demodulator
    grid;
    drawnow;
end
```

CODE FOR PSK4 MODULATOR

```
function [PSK4Output]=psk4_mod(Data);

% ----------- MODULATING PARAMETERS ------------------

freq=2;
amp=1;
phase=[0 pi/2 pi 3*pi/2];
t=0:0.01:1;

% ---------- CALCULATED MODULATING PARAMETERS --------

DataLength=length(Data);
w=2*pi*freq;

% ----------- WAVES GENERATED FOR MODULATION --------

x1=sin((w*t)+phase(1));
x2=sin((w*t)+phase(4));
x3=sin((w*t)+phase(3));
x4=sin((w*t)+phase(2));

% ------------ SERIAL TO PARALLEL CONVERSION -----------

sym=2;
r=rem(DataLength,sym);
if r~=0
    r1=sym-r;
    r2=zeros(1,r1);
    Data=[Data,r2];
end

z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);

% -------------- MODULATION -------------------

PSK4Output=[];           % initializing empty vectors
for i=1:n                % assigning waves to the symbols
    if z([1,2],i)'==[0 0]
        PSK4Output=[PSK4Output,x1];
    elseif z([1,2],i)'==[0 1]
        PSK4Output=[PSK4Output,x2];
    elseif z([1,2],i)'==[1 0]
        PSK4Output=[PSK4Output,x3];
    elseif z([1,2],i)'==[1 1]
        PSK4Output=[PSK4Output,x4];
```

```
    end
end

% ---------- MODULATOR'S OUTPUT -------------------------

% plot(PSK4Output,'m');
% grid;
% xlabel('output wave of QPSK modulator');
```

CODE FOR PSK4 DEMODULATOR

```
function [output_demod]=psk4_demod(output_mod,amp,phase)

% ----------- KNOWN DEMODULATING PARAMETERS ----------

freq=2;
t=0:0.01:1;

% -------- CALCULATED MODULATING PARAMETERS ----------

T=length(t);
datalen=length(output_mod);
pw=datalen/T;
w=2*pi*freq;

% ------------------- DEMODULATION ------------

output_demod=[];
count=0;
for i=1:pw
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    for j=1:T
        if output_mod(count+j)==amp*sin((w*t(j))+phase(1))
            bit0=bit0+1;
        elseif
output_mod(count+j)==amp*sin((w*t(j))+phase(3))
            bit1=bit1+1;
        elseif
output_mod(count+j)==amp*sin((w*t(j))+phase(4))
            bit2=bit2+1;
        elseif
output_mod(count+j)==amp*sin((w*t(j))+phase(2))
            bit3=bit3+1;
```

```
        end
    end
    count=count+T;
    if bit0==T
        output_demod=[output_demod,0 0];
    elseif bit1==T
        output_demod=[output_demod,1 0];
    elseif bit2==T
        output_demod=[output_demod,0 1];
    elseif bit3==T
        output_demod=[output_demod,1 1];
    end
    stem(output_demod,'r');
    grid;
    drawnow;
end
```

CODE FOR PSK8 MODULATOR

```
function [PSK8Output]=psk8_mod(Data)

% ------------- MODULATION PARAMETERS --------------------

DataLength=length(Data);
Freq=2;
Amp=1;
Phase=[0 pi/4 pi/2 3*pi/4 pi 5*pi/4 3*pi/2 7*pi/4];
t=0:0.01:1;            % symbol duration
T=length(t);
w=2*pi*Freq;          % obtained angular frequency

% ------------- WAVES GENERATED FOR MODULATION ----

x1=Amp*(sin((w*t)+(Phase(1))));  % wave for symbol '000'
x2=Amp*(sin((w*t)+(Phase(2))));  % wave for symbol '001'
x3=Amp*(sin((w*t)+(Phase(3))));  % wave for symbol '010'
x4=Amp*(sin((w*t)+(Phase(4))));  % wave for symbol '011'
x5=Amp*(sin((w*t)+(Phase(5))));  % wave for symbol '100'
x6=Amp*(sin((w*t)+(Phase(6))));  % wave for symbol '101'
x7=Amp*(sin((w*t)+(Phase(7))));  % wave for symbol '110'
x8=Amp*(sin((w*t)+(Phase(8))));  % wave for symbol '111'

% -- SERIAL TO PARALLEL CONVERSION (SYMBOL FORMATION) ---

sym=3;
r=rem(DataLength,sym);
if r~=0
    r1=sym-r;
```

```
    r2=zeros(1,r1);
    Data=[Data,r2];
end

z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);

% -------------- MODULATION ------------------------

PSK8Output=[];                % initializing empty vector
for i=1:n
    if z([1,2,3],i)'==[0 0 0]
        PSK8Output=[PSK8Output,x1];
    elseif z([1,2,3],i)'==[0 0 1]
        PSK8Output=[PSK8Output,x2];
    elseif z([1,2,3],i)'==[0 1 0]
        PSK8Output=[PSK8Output,x3];
    elseif z([1,2,3],i)'==[0 1 1]
        PSK8Output=[PSK8Output,x4];
    elseif z([1,2,3],i)'==[1 0 0]
        PSK8Output=[PSK8Output,x5];
    elseif z([1,2,3],i)'==[1 0 1]
        PSK8Output=[PSK8Output,x6];
    elseif z([1,2,3],i)'==[1 1 0]
        PSK8Output=[PSK8Output,x7];
    elseif z([1,2,3],i)'==[1 1 1]
        PSK8Output=[PSK8Output,x8];
    end
end

% ------------- OUTPUT WAVEFORM THAT IS TRANSMITTED ------

% plot(PSK8Output,'r');     % output waveform of modulator
% grid;
```

CODE FOR PSK8 DEMODULATOR

```
function [DemodStream]=psk8_demod(ModOutput,Amp,Phases)

% ----------- KNOWN DEMODULATION PARAMETERS ----------

Freq=2;
w=2*pi*Freq;
t=0:0.01:1;
T=length(t);
```

```
% ------- CALCULATED DEMODULATION PARMETERS -----

RecLength=length(ModOutput);
RecSymbols=RecLength/T;
DemodStream=[];
count=0;

% -------------- DEMODULATION --------------

for i=1:RecSymbols
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    bit4=0;
    bit5=0;
    bit6=0;
    bit7=0;
    for j=1:T
        if ModOutput(count+j)==Amp*sin(w*t(j)+Phases(1))
            bit0=bit0+1;
        elseif
ModOutput(count+j)==Amp*sin(w*t(j)+Phases(2))
            bit1=bit1+1;
        elseif
ModOutput(count+j)==Amp*sin(w*t(j)+Phases(3))
            bit2=bit2+1;
        elseif
ModOutput(count+j)==Amp*sin(w*t(j)+Phases(4))
            bit3=bit3+1;
        elseif
ModOutput(count+j)==Amp*sin(w*t(j)+Phases(5))
            bit4=bit4+1;
        elseif
ModOutput(count+j)==Amp*sin(w*t(j)+Phases(6))
            bit5=bit5+1;
        elseif
ModOutput(count+j)==Amp*sin(w*t(j)+Phases(7))
            bit6=bit6+1;
        elseif
ModOutput(count+j)==Amp*sin(w*t(j)+Phases(8))
            bit7=bit7+1;
        end
    end

    count=count+T;
```

```
    if bit0==T
        DemodStream=[DemodStream,0 0 0];
    elseif bit1==T
        DemodStream=[DemodStream,0 0 1];
    elseif bit2==T
        DemodStream=[DemodStream,0 1 0];
    elseif bit3==T
        DemodStream=[DemodStream,0 1 1];
    elseif bit4==T
        DemodStream=[DemodStream,1 0 0];
    elseif bit5==T
        DemodStream=[DemodStream,1 0 1];
    elseif bit6==T
        DemodStream=[DemodStream,1 1 0];
    elseif bit7==T
        DemodStream=[DemodStream,1 1 1];
    end
    stem(DemodStream,'r');
    grid;
    drawnow;
end
```

CODE FOR QAM4 MODULATOR

```
function [QAM4Output]=qam4_mod(Data)

% ---------- MODULATING PARAMETERS -----------------

t=0:0.01:1;
f=2;
w=2*pi*f;
amp=[1 2];
phase=[0 pi];
DataLength=length(Data);

% ---------------- MODULATING WAVES --------------

x1=amp(1)*sin((w*t)+phase(1));
x2=amp(1)*sin((w*t)+phase(2));
x3=amp(2)*sin((w*t)+phase(1));
x4=amp(2)*sin((w*t)+phase(2));

% ------ PARALLEL TO SERIAL CONVERSION OF DATA -------

sym=2;
r=rem(DataLength,sym);
```

```
if r~=0
    r1=sym-r;
    r2=zeros(1,r1);
    Data=[Data,r2];
end

z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);


% ------------------ MODULATION --------------------------

QAM4Output=[];            % initializing empty vectors
for i=1:n                 % assigning waves to the symbols
    if z([1,2],i)'==[0 0]
        QAM4Output=[QAM4Output,x1];
    elseif z([1,2],i)'==[0 1]
        QAM4Output=[QAM4Output,x2];
    elseif z([1,2],i)'==[1 0]
        QAM4Output=[QAM4Output,x3];
    elseif z([1,2],i)'==[1 1]
        QAM4Output=[QAM4Output,x4];
    end
end

% ------------ MODULATOR'S OUTPUT -----------------

% plot(ModulatorOutput,'r');
% grid;
% title('OUTPUT WAVEFORM OF QAM-4 MODULATOR');
```

CODE FOR QAM4 DEMODULATOR

```
function [Demod]=qam4_demod(MO,amp,phase)

% ------------------ KNOWN PARAMETERS ---------

t=0:0.01:1;
T=length(t);
Freq=2;

% -------------- CALCULATED PARAMETERS ----------------

datalen=length(MO);           % total data points
pw=datalen/T;                 % duration data points
w=2*pi*Freq;                  % obtained angular frequency
```

```
% --------------- DEMODULATION --------------------

Demod=[];                   % initializing bit stream
count=0;                    % initializing step count
for i=1:pw                  % loop for next bit duration
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    for j=1:T
        if MO(count+j)==amp(1)*sin(w*t(j)+phase(1))
                                % for symbol '00'
            bit0=bit0+1;
        elseif MO(count+j)==amp(1)*sin(w*t(j)+phase(2))
                                % for symbol '01'
            bit1=bit1+1;
        elseif MO(count+j)==amp(2)*sin(w*t(j)+phase(1))
                                % for symbol '10'
            bit2=bit2+1;
        elseif MO(count+j)==amp(2)*sin(w*t(j)+phase(2))
                                % for symbol '11'
            bit3=bit3+1;
        end
    end
    count=count+T;      % count added with number of data
                        points in one duration
    if bit0==T
        Demod=[Demod,0 0]; % accumulation of symbol '00'
    elseif bit1==T    % first point of sine wave is also 0
        Demod=[Demod,0 1]; % accumulation of symbol '01'
    elseif bit2==T-1  % first point of sine wave is also 0
        Demod=[Demod,1 0]; % accumulation of symbol '10'
    elseif bit3==T    % first point of sine wave is also 0
        Demod=[Demod,1 1]; % accumulation of symbol '11'
    end
    stem(Demod,'r');
    grid;
    drawnow;
end
```

CODE FOR QAM8 MODULATOR

```
function [QAM8Output]=qam8_mod(Data)

% ---------- MODULATION PARAMETERS ---------------------
```

```
DataLength=length(Data);
Freq=2;
Amp=[1 2];
Phase=[0 pi/2 pi 3*pi/2];
t=0:0.01:1;                    % symbol duration
T=length(t);
w=2*pi*Freq;                   % obtained angular frequency

% ----------- WAVES GENERATED FOR MODULATION -----------

x1=Amp(1)*(sin((w*t)+Phase(1)));          % wave for '000'
x2=Amp(1)*(sin((w*t)+Phase(2)));          % wave for '001'
x3=Amp(1)*(sin((w*t)+Phase(3)));          % wave for '010'
x4=Amp(1)*(sin((w*t)+Phase(4)));          % wave for '011'
x5=Amp(2)*(sin((w*t)+Phase(1)));          % wave for '100'
x6=Amp(2)*(sin((w*t)+Phase(2)));          % wave for '101'
x7=Amp(2)*(sin((w*t)+Phase(3)));          % wave for '110'
x8=Amp(2)*(sin((w*t)+Phase(4)));          % wave for '111'

% --- SERIAL TO PARALLEL CONVERSION (SYMBOL FORMATION) ---

sym=3;
r=rem(DataLength,sym);
if r~=0
    r1=sym-r;
    r2=zeros(1,r1);
    Data=[Data,r2];
end

z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);

% ---------------- MODULATION ------------------

QAM8Output=[];              % initializing empty vectors
for i=1:n
    if z([1,2,3],i)'==[0 0 0]
        QAM8Output=[QAM8Output,x1];
    elseif z([1,2,3],i)'==[0 0 1]
        QAM8Output=[QAM8Output,x2];
    elseif z([1,2,3],i)'==[0 1 0]
        QAM8Output=[QAM8Output,x3];
    elseif z([1,2,3],i)'==[0 1 1]
        QAM8Output=[QAM8Output,x4];
    elseif z([1,2,3],i)'==[1 0 0]
        QAM8Output=[QAM8Output,x5];
    elseif z([1,2,3],i)'==[1 0 1]
```

```
        QAM8Output=[QAM8Output,x6];
    elseif z([1,2,3],i)'==[1 1 0]
        QAM8Output=[QAM8Output,x7];
    elseif z([1,2,3],i)'==[1 1 1]
        QAM8Output=[QAM8Output,x8];
    end
end

% ---------- OUTPUT WAVEFORM THAT IS TRANSMITTED -------

% plot(QAM8Output,'r');     % output waveform of modulator
% grid;
```

CODE FOR QAM8 DEMODULATOR

```
function [output_demod]=qam8_demod(output_mod,amp,phase)

% -------- KNOWN DEMODULATION PARAMETERS ----------

freq=2;
t=0:0.01:1;                                % bit duration
T=length(t);

% --------- CALCULATED DEMODULATION PARMETERS ----------

datalen=length(output_mod);  % total data points
pw=datalen/T;            % duration data points
w=2*pi*freq;             % obtained angular frequency

% ----------------- DEMODULATION --------------------

output_demod=[];             % initializing bit stream
count=0;
% step count
for i=1:pw                   % loop for next bit duration
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    bit4=0;
    bit5=0;
    bit6=0;
    bit7=0;
    for j=1:T
        if
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(1))
           bit0=bit0+1;
```

```
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(2))
            bit1=bit1+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(3))
            bit2=bit2+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(4))
            bit3=bit3+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(1))
            bit4=bit4+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(2))
            bit5=bit5+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(3))
            bit6=bit6+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(4))
            bit7=bit7+1;
        end
    end

    count=count+T;
% count added with number of data points in one duration

    if bit0==T
        output_demod=[output_demod,0 0 0];
    elseif bit1==T
        output_demod=[output_demod,0 0 1];
    elseif bit2==T
        output_demod=[output_demod,0 1 0];
    elseif bit3==T
        output_demod=[output_demod,0 1 1];
    elseif bit4==T-1
        output_demod=[output_demod,1 0 0];
    elseif bit5==T
        output_demod=[output_demod,1 0 1];
    elseif bit6==T
        output_demod=[output_demod,1 1 0];
    elseif bit7==T
        output_demod=[output_demod,1 1 1];
    end
    stem(output_demod,'r');
    grid;
    drawnow;
```

```
end
```

CODE FOR QAM16 MODULATOR

```
function [QAM16Output]=qam16_mod(Data)

% -------------- MODULATION PARAMETERS -------------

DataLength=length(Data);
Freq=2;
Amp=[1 2];
Phase=[0 pi/4 pi/2 3*pi/4 pi 5*pi/4 3*pi/2 7*pi/4];
t=0:0.01:1;                 % symbol duration
T=length(t);
w=2*pi*Freq;                % obtained angular frequency

% ------- WAVES GENERATED FOR MODULATION -------------

x1=Amp(1)*(sin((w*t)+Phase(1)));       % wave for '0000'
x2=Amp(1)*(sin((w*t)+Phase(2)));       % wave for '0001'
x3=Amp(1)*(sin((w*t)+Phase(3)));       % wave for '0010'
x4=Amp(1)*(sin((w*t)+Phase(4)));       % wave for '0011'
x5=Amp(1)*(sin((w*t)+Phase(5)));       % wave for '0100'
x6=Amp(1)*(sin((w*t)+Phase(6)));       % wave for '0101'
x7=Amp(1)*(sin((w*t)+Phase(7)));       % wave for '0110'
x8=Amp(1)*(sin((w*t)+Phase(8)));       % wave for '0111'
x9=Amp(2)*(sin((w*t)+Phase(1)));       % wave for '1000'
x10=Amp(2)*(sin((w*t)+Phase(2)));      % wave for '1001'
x11=Amp(2)*(sin((w*t)+Phase(3)));      % wave for '1010'
x12=Amp(2)*(sin((w*t)+Phase(4)));      % wave for '1011'
x13=Amp(2)*(sin((w*t)+Phase(5)));      % wave for '1100'
x14=Amp(2)*(sin((w*t)+Phase(6)));      % wave for '1101'
x15=Amp(2)*(sin((w*t)+Phase(7)));      % wave for '1110'
x16=Amp(2)*(sin((w*t)+Phase(8)));      % wave for '1111'

% --- SERIAL TO PARALLEL CONVERSION (SYMBOL FORMATION) ---

sym=4;
r=rem(DataLength,sym);
if r~=0
    r1=sym-r;
    r2=zeros(1,r1);
    Data=[Data,r2];
end

z=reshape(Data,sym,length(Data)/sym);
```

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

% ------------------ MODULATION ----------------------

QAM16Output=[]; % initializing output of the transmitter

for i=1:n
    if z([1,2,3,4],i)'==[0 0 0 0]
        QAM16Output=[QAM16Output,x1];
    elseif z([1,2,3,4],i)'==[0 0 0 1]
        QAM16Output=[QAM16Output,x2];
    elseif z([1,2,3,4],i)'==[0 0 1 0]
        QAM16Output=[QAM16Output,x3];
    elseif z([1,2,3,4],i)'==[0 0 1 1]
        QAM16Output=[QAM16Output,x4];
    elseif z([1,2,3,4],i)'==[0 1 0 0]
        QAM16Output=[QAM16Output,x5];
    elseif z([1,2,3,4],i)'==[0 1 0 1]
        QAM16Output=[QAM16Output,x6];
    elseif z([1,2,3,4],i)'==[0 1 1 0]
        QAM16Output=[QAM16Output,x7];
    elseif z([1,2,3,4],i)'==[0 1 1 1]
        QAM16Output=[QAM16Output,x8];
    elseif z([1,2,3,4],i)'==[1 0 0 0]
        QAM16Output=[QAM16Output,x9];
    elseif z([1,2,3,4],i)'==[1 0 0 1]
        QAM16Output=[QAM16Output,x10];
    elseif z([1,2,3,4],i)'==[1 0 1 0]
        QAM16Output=[QAM16Output,x11];
    elseif z([1,2,3,4],i)'==[1 0 1 1]
        QAM16Output=[QAM16Output,x12];
    elseif z([1,2,3,4],i)'==[1 1 0 0]
        QAM16Output=[QAM16Output,x13];
    elseif z([1,2,3,4],i)'==[1 1 0 1]
        QAM16Output=[QAM16Output,x14];
    elseif z([1,2,3,4],i)'==[1 1 1 0]
        QAM16Output=[QAM16Output,x15];
    elseif z([1,2,3,4],i)'==[1 1 1 1]
        QAM16Output=[QAM16Output,x16];
    end
end


% ----------- OUTPUT WAVEFORM THAT IS TRANSMITTED -------

% plot(QAM16Output,'r');     % output waveform of modulator
% grid;
```

CODE FOR QAM16 DEMODULATOR

```
function [output_demod]=qam16_demod(output_mod,amp,phase)

% --------KNOWN DEMODULATION PARAMETERS ---------------

freq=2;                    % intermediate frequency
t=0:0.01:1;                % bit duration
T=length(t);

% ---------- CALCULATED DEMODULATION PARMETERS --------

datalen=length(output_mod);    % total data points
pw=datalen/T;                  % duration data points
w=2*pi*freq;                   % obtained angular frequency

% ------------ DEMODULATION -----------------

output_demod=[];           % initializing bit stream
count=0;                   % step count
for i=1:pw                 % loop for next bit duration
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    bit4=0;
    bit5=0;
    bit6=0;
    bit7=0;
    bit8=0;
    bit9=0;
    bit10=0;
    bit11=0;
    bit12=0;
    bit13=0;
    bit14=0;
    bit15=0;
    for j=1:T
        if
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(1))
            bit0=bit0+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(2))
            bit1=bit1+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(3))
            bit2=bit2+1;
```

```
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(4))
            bit3=bit3+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(5))
            bit4=bit4+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(6))
            bit5=bit5+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(7))
            bit6=bit6+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(8))
            bit7=bit7+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(1))
            bit8=bit8+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(2))
            bit9=bit9+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(3))
            bit10=bit10+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(4))
            bit11=bit11+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(5))
            bit12=bit12+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(6))
            bit13=bit13+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(7))
            bit14=bit14+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(8))
            bit15=bit15+1;
        end
    end

    count=count+T;
% count added with number of data points in one duration

    if bit0==T
        output_demod=[output_demod,0 0 0 0];
```

```
    elseif bit1==T
        output_demod=[output_demod,0 0 0 1];
    elseif bit2==T
        output_demod=[output_demod,0 0 1 0];
    elseif bit3==T
        output_demod=[output_demod,0 0 1 1];
    elseif bit4==T
        output_demod=[output_demod,0 1 0 0];
    elseif bit5==T
        output_demod=[output_demod,0 1 0 1];
    elseif bit6==T
        output_demod=[output_demod,0 1 1 0];
    elseif bit7==T
        output_demod=[output_demod,0 1 1 1];
    elseif bit8==T-1
        output_demod=[output_demod,1 0 0 0];
    elseif bit9==T
        output_demod=[output_demod,1 0 0 1];
    elseif bit10==T
        output_demod=[output_demod,1 0 1 0];
    elseif bit11==T
        output_demod=[output_demod,1 0 1 1];
    elseif bit12==T
        output_demod=[output_demod,1 1 0 0];
    elseif bit13==T
        output_demod=[output_demod,1 1 0 1];
    elseif bit14==T
        output_demod=[output_demod,1 1 1 0];
    elseif bit15==T
        output_demod=[output_demod,1 1 1 1];
    end
    stem(output_demod,'r');
    grid;
    drawnow;
end
```

CODE FOR QAM32 MODULATOR

```
function [QAM32Output]=qam32_mod(Data)

% ---------- MODULATION PARAMETERS --------------------

DataLength=length(Data);
Freq=2;
Amp=[1 2 3 4];
Phase=[0 pi/4 pi/2 3*pi/4 pi 5*pi/4 3*pi/2 7*pi/4];
```

```
t=0:0.01:1;                % symbol duration
T=length(t);
w=2*pi*Freq;               % obtained angular frequency

% ------------- WAVES GENERATED FOR MODULATION ---------

x1=Amp(1)*(sin((w*t)+Phase(1)));
x2=Amp(1)*(sin((w*t)+Phase(2)));
x3=Amp(1)*(sin((w*t)+Phase(3)));
x4=Amp(1)*(sin((w*t)+Phase(4)));
x5=Amp(1)*(sin((w*t)+Phase(5)));
x6=Amp(1)*(sin((w*t)+Phase(6)));
x7=Amp(1)*(sin((w*t)+Phase(7)));
x8=Amp(1)*(sin((w*t)+Phase(8)));
x9=Amp(2)*(sin((w*t)+Phase(1)));
x10=Amp(2)*(sin((w*t)+Phase(2)));
x11=Amp(2)*(sin((w*t)+Phase(3)));
x12=Amp(2)*(sin((w*t)+Phase(4)));
x13=Amp(2)*(sin((w*t)+Phase(5)));
x14=Amp(2)*(sin((w*t)+Phase(6)));
x15=Amp(2)*(sin((w*t)+Phase(7)));
x16=Amp(2)*(sin((w*t)+Phase(8)));
x17=Amp(3)*(sin((w*t)+Phase(1)));
x18=Amp(3)*(sin((w*t)+Phase(2)));
x19=Amp(3)*(sin((w*t)+Phase(3)));
x20=Amp(3)*(sin((w*t)+Phase(4)));
x21=Amp(3)*(sin((w*t)+Phase(5)));
x22=Amp(3)*(sin((w*t)+Phase(6)));
x23=Amp(3)*(sin((w*t)+Phase(7)));
x24=Amp(3)*(sin((w*t)+Phase(8)));
x25=Amp(4)*(sin((w*t)+Phase(1)));
x26=Amp(4)*(sin((w*t)+Phase(2)));
x27=Amp(4)*(sin((w*t)+Phase(3)));
x28=Amp(4)*(sin((w*t)+Phase(4)));
x29=Amp(4)*(sin((w*t)+Phase(5)));
x30=Amp(4)*(sin((w*t)+Phase(6)));
x31=Amp(4)*(sin((w*t)+Phase(7)));
x32=Amp(4)*(sin((w*t)+Phase(8)));

% --- SERIAL TO PARALLEL CONVERSION (SYMBOL FORMATION) ---

sym=5;
z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);

% -------------- MODULATION ------------------
```

```
QAM32Output=[];% initializing output of the transmitter

for i=1:n
    if z([1,2,3,4,5],i)'==[0 0 0 0 0]
        QAM32Output=[QAM32Output,x1];
    elseif z([1,2,3,4,5],i)'==[0 0 0 0 1]
        QAM32Output=[QAM32Output,x2];
    elseif z([1,2,3,4,5],i)'==[0 0 0 1 0]
        QAM32Output=[QAM32Output,x3];
    elseif z([1,2,3,4,5],i)'==[0 0 0 1 1]
        QAM32Output=[QAM32Output,x4];
    elseif z([1,2,3,4,5],i)'==[0 0 1 0 0]
        QAM32Output=[QAM32Output,x5];
    elseif z([1,2,3,4,5],i)'==[0 0 1 0 1]
        QAM32Output=[QAM32Output,x6];
    elseif z([1,2,3,4,5],i)'==[0 0 1 1 0]
        QAM32Output=[QAM32Output,x7];
    elseif z([1,2,3,4,5],i)'==[0 0 1 1 1]
        QAM32Output=[QAM32Output,x8];
    elseif z([1,2,3,4,5],i)'==[0 1 0 0 0]
        QAM32Output=[QAM32Output,x9];
    elseif z([1,2,3,4,5],i)'==[0 1 0 0 1]
        QAM32Output=[QAM32Output,x10];
    elseif z([1,2,3,4,5],i)'==[0 1 0 1 0]
        QAM32Output=[QAM32Output,x11];
    elseif z([1,2,3,4,5],i)'==[0 1 0 1 1]
        QAM32Output=[QAM32Output,x12];
    elseif z([1,2,3,4,5],i)'==[0 1 1 0 0]
        QAM32Output=[QAM32Output,x13];
    elseif z([1,2,3,4,5],i)'==[0 1 1 0 1]
        QAM32Output=[QAM32Output,x14];
    elseif z([1,2,3,4,5],i)'==[0 1 1 1 0]
        QAM32Output=[QAM32Output,x15];
    elseif z([1,2,3,4,5],i)'==[0 1 1 1 1]
        QAM32Output=[QAM32Output,x16];
    elseif z([1,2,3,4,5],i)'==[1 0 0 0 0]
        QAM32Output=[QAM32Output,x17];
    elseif z([1,2,3,4,5],i)'==[1 0 0 0 1]
        QAM32Output=[QAM32Output,x18];
    elseif z([1,2,3,4,5],i)'==[1 0 0 1 0]
        QAM32Output=[QAM32Output,x19];
    elseif z([1,2,3,4,5],i)'==[1 0 0 1 1]
        QAM32Output=[QAM32Output,x20];
    elseif z([1,2,3,4,5],i)'==[1 0 1 0 0]
        QAM32Output=[QAM32Output,x21];
    elseif z([1,2,3,4,5],i)'==[1 0 1 0 1]
        QAM32Output=[QAM32Output,x22];
```

```
    elseif z([1,2,3,4,5],i)'==[1 0 1 1 0]
        QAM32Output=[QAM32Output,x23];
    elseif z([1,2,3,4,5],i)'==[1 0 1 1 1]
        QAM32Output=[QAM32Output,x24];
    elseif z([1,2,3,4,5],i)'==[1 1 0 0 0]
        QAM32Output=[QAM32Output,x25];
    elseif z([1,2,3,4,5],i)'==[1 1 0 0 1]
        QAM32Output=[QAM32Output,x26];
    elseif z([1,2,3,4,5],i)'==[1 1 0 1 0]
        QAM32Output=[QAM32Output,x27];
    elseif z([1,2,3,4,5],i)'==[1 1 0 1 1]
        QAM32Output=[QAM32Output,x28];
    elseif z([1,2,3,4,5],i)'==[1 1 1 0 0]
        QAM32Output=[QAM32Output,x29];
    elseif z([1,2,3,4,5],i)'==[1 1 1 0 1]
        QAM32Output=[QAM32Output,x30];
    elseif z([1,2,3,4,5],i)'==[1 1 1 1 0]
        QAM32Output=[QAM32Output,x31];
    elseif z([1,2,3,4,5],i)'==[1 1 1 1 1]
        QAM32Output=[QAM32Output,x32];
    end
end


% ---------- OUTPUT WAVEFORM THAT IS TRANSMITTED ----

% plot(QAM32Output,'r');    % output waveform of modulator
% grid;
```

CODE FOR QAM32 DEMODULATOR

```
function [output_demod]=qam32_demod(output_mod,amp,phase)

% --------- KNOWN DEMODULATION PARAMETERS ----------

freq=2;
t=0:0.01:1;              % bit duration
T=length(t);

% -------- CALCULATED DEMODULATION PARMETERS ----------

datalen=length(output_mod);  % total data points
pw=datalen/T;                % duration data points
w=2*pi*freq;                 % obtained angular frequency

% ---------- DEMODULATION -------------------
```

```
output_demod=[];                % initializing bit stream
count=0;                        % step count
for i=1:pw                      % loop for next bit duration
    bit0=0;
    bit1=0;
    bit2=0;
    bit3=0;
    bit4=0;
    bit5=0;
    bit6=0;
    bit7=0;
    bit8=0;
    bit9=0;
    bit10=0;
    bit11=0;
    bit12=0;
    bit13=0;
    bit14=0;
    bit15=0;
    bit16=0;
    bit17=0;
    bit18=0;
    bit19=0;
    bit20=0;
    bit21=0;
    bit22=0;
    bit23=0;
    bit24=0;
    bit25=0;
    bit26=0;
    bit27=0;
    bit28=0;
    bit29=0;
    bit30=0;
    bit31=0;

    for j=1:T
        if
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(1))
            bit0=bit0+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(2))
            bit1=bit1+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(3))
            bit2=bit2+1;
```

```
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(4))
            bit3=bit3+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(5))
            bit4=bit4+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(6))
            bit5=bit5+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(7))
            bit6=bit6+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(8))
            bit7=bit7+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(1))
            bit8=bit8+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(2))
            bit9=bit9+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(3))
            bit10=bit10+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(4))
            bit11=bit11+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(5))
            bit12=bit12+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(6))
            bit13=bit13+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(7))
            bit14=bit14+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(8))
            bit15=bit15+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(1))
            bit16=bit16+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(2))
            bit17=bit17+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(3))
```

```
            bit18=bit18+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(4))
            bit19=bit19+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(5))
            bit20=bit20+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(6))
            bit21=bit21+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(7))
            bit22=bit22+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(8))
            bit23=bit23+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(1))
            bit24=bit24+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(2))
            bit25=bit25+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(3))
            bit26=bit26+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(4))
            bit27=bit27+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(5))
            bit28=bit28+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(6))
            bit29=bit29+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(7))
            bit30=bit30+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(8))
            bit31=bit31+1;
        end
    end

    count=count+T;
% count added with number of data points in one duration

    if bit0==T
```

```
    output_demod=[output_demod,0 0 0 0 0];
elseif bit1==T
    output_demod=[output_demod,0 0 0 0 1];
elseif bit2==T
    output_demod=[output_demod,0 0 0 1 0];
elseif bit3==T
    output_demod=[output_demod,0 0 0 1 1];
elseif bit4==T
    output_demod=[output_demod,0 0 1 0 0];
elseif bit5==T
    output_demod=[output_demod,0 0 1 0 1];
elseif bit6==T
    output_demod=[output_demod,0 0 1 1 0];
elseif bit7==T
    output_demod=[output_demod,0 0 1 1 1];
elseif bit8==T-1
    output_demod=[output_demod,0 1 0 0 0];
elseif bit9==T
    output_demod=[output_demod,0 1 0 0 1];
elseif bit10==T
    output_demod=[output_demod,0 1 0 1 0];
elseif bit11==T
    output_demod=[output_demod,0 1 0 1 1];
elseif bit12==T
    output_demod=[output_demod,0 1 1 0 0];
elseif bit13==T
    output_demod=[output_demod,0 1 1 0 1];
elseif bit14==T
    output_demod=[output_demod,0 1 1 1 0];
elseif bit15==T
    output_demod=[output_demod,0 1 1 1 1];
elseif bit16==T-1
    output_demod=[output_demod,1 0 0 0 0];
elseif bit17==T
    output_demod=[output_demod,1 0 0 0 1];
elseif bit18==T
    output_demod=[output_demod,1 0 0 1 0];
elseif bit19==T
    output_demod=[output_demod,1 0 0 1 1];
elseif bit20==T
    output_demod=[output_demod,1 0 1 0 0];
elseif bit21==T
    output_demod=[output_demod,1 0 1 0 1];
elseif bit22==T
    output_demod=[output_demod,1 0 1 1 0];
elseif bit23==T
    output_demod=[output_demod,1 0 1 1 1];
```

```
    elseif bit24==T-1
        output_demod=[output_demod,1 1 0 0 0];
    elseif bit25==T
        output_demod=[output_demod,1 1 0 0 1];
    elseif bit26==T
        output_demod=[output_demod,1 1 0 1 0];
    elseif bit27==T
        output_demod=[output_demod,1 1 0 1 1];
    elseif bit28==T
        output_demod=[output_demod,1 1 1 0 0];
    elseif bit29==T
        output_demod=[output_demod,1 1 1 0 1];
    elseif bit30==T
        output_demod=[output_demod,1 1 1 1 0];
    elseif bit31==T
        output_demod=[output_demod,1 1 1 1 1];
    end
    stem(output_demod,'r');
    grid;
    drawnow;
end
```

CODE FOR QAM64 MODULATOR

```
function [QAM64Output]=qam64_mod(Data)

% ----------- MODULATION PARAMETERS ---------------

DataLength=length(Data);
Freq=2;
Amp=[1 2 3 4];
Phase=[0 pi/8 pi/4 3*pi/8 pi/2 5*pi/8 3*pi/4 7*pi/8 pi
9*pi/8 5*pi/4 11*pi/8 4*pi/3 13*pi/8 7*pi/8 15*pi/8];
t=0:0.01:1;         % symbol duration
T=length(t);
w=2*pi*Freq;        % obtained angular frequency

% --------- WAVES GENERATED FOR MODULATION --------

x1=Amp(1)*(sin((w*t)+Phase(1)));
x2=Amp(1)*(sin((w*t)+Phase(2)));
x3=Amp(1)*(sin((w*t)+Phase(3)));
x4=Amp(1)*(sin((w*t)+Phase(4)));
x5=Amp(1)*(sin((w*t)+Phase(5)));
x6=Amp(1)*(sin((w*t)+Phase(6)));
x7=Amp(1)*(sin((w*t)+Phase(7)));
x8=Amp(1)*(sin((w*t)+Phase(8)));
```

```
x9=Amp(1)*(sin((w*t)+Phase(9)));
x10=Amp(1)*(sin((w*t)+Phase(10)));
x11=Amp(1)*(sin((w*t)+Phase(11)));
x12=Amp(1)*(sin((w*t)+Phase(12)));
x13=Amp(1)*(sin((w*t)+Phase(13)));
x14=Amp(1)*(sin((w*t)+Phase(14)));
x15=Amp(1)*(sin((w*t)+Phase(15)));
x16=Amp(1)*(sin((w*t)+Phase(16)));
x17=Amp(2)*(sin((w*t)+Phase(1)));
x18=Amp(2)*(sin((w*t)+Phase(2)));
x19=Amp(2)*(sin((w*t)+Phase(3)));
x20=Amp(2)*(sin((w*t)+Phase(4)));
x21=Amp(2)*(sin((w*t)+Phase(5)));
x22=Amp(2)*(sin((w*t)+Phase(6)));
x23=Amp(2)*(sin((w*t)+Phase(7)));
x24=Amp(2)*(sin((w*t)+Phase(8)));
x25=Amp(2)*(sin((w*t)+Phase(9)));
x26=Amp(2)*(sin((w*t)+Phase(10)));
x27=Amp(2)*(sin((w*t)+Phase(11)));
x28=Amp(2)*(sin((w*t)+Phase(12)));
x29=Amp(2)*(sin((w*t)+Phase(13)));
x30=Amp(2)*(sin((w*t)+Phase(14)));
x31=Amp(2)*(sin((w*t)+Phase(15)));
x32=Amp(2)*(sin((w*t)+Phase(16)));
x33=Amp(3)*(sin((w*t)+Phase(1)));
x34=Amp(3)*(sin((w*t)+Phase(2)));
x35=Amp(3)*(sin((w*t)+Phase(3)));
x36=Amp(3)*(sin((w*t)+Phase(4)));
x37=Amp(3)*(sin((w*t)+Phase(5)));
x38=Amp(3)*(sin((w*t)+Phase(6)));
x39=Amp(3)*(sin((w*t)+Phase(7)));
x40=Amp(3)*(sin((w*t)+Phase(8)));
x41=Amp(3)*(sin((w*t)+Phase(9)));
x42=Amp(3)*(sin((w*t)+Phase(10)));
x43=Amp(3)*(sin((w*t)+Phase(11)));
x44=Amp(3)*(sin((w*t)+Phase(12)));
x45=Amp(3)*(sin((w*t)+Phase(13)));
x46=Amp(3)*(sin((w*t)+Phase(14)));
x47=Amp(3)*(sin((w*t)+Phase(15)));
x48=Amp(3)*(sin((w*t)+Phase(16)));
x49=Amp(4)*(sin((w*t)+Phase(1)));
x50=Amp(4)*(sin((w*t)+Phase(2)));
x51=Amp(4)*(sin((w*t)+Phase(3)));
x52=Amp(4)*(sin((w*t)+Phase(4)));
x53=Amp(4)*(sin((w*t)+Phase(5)));
x54=Amp(4)*(sin((w*t)+Phase(6)));
x55=Amp(4)*(sin((w*t)+Phase(7)));
```

```
x56=Amp(4)*(sin((w*t)+Phase(8)));
x57=Amp(4)*(sin((w*t)+Phase(9)));
x58=Amp(4)*(sin((w*t)+Phase(10)));
x59=Amp(4)*(sin((w*t)+Phase(11)));
x60=Amp(4)*(sin((w*t)+Phase(12)));
x61=Amp(4)*(sin((w*t)+Phase(13)));
x62=Amp(4)*(sin((w*t)+Phase(14)));
x63=Amp(4)*(sin((w*t)+Phase(15)));
x64=Amp(4)*(sin((w*t)+Phase(16)));

% --- SERIAL TO PARALLEL CONVERSION (SYMBOL FORMATION) ---

sym=6;
r=rem(DataLength,sym);
if r~=0
    r1=sym-r;
    r2=zeros(1,r1);
    Data=[Data,r2];
end

z=reshape(Data,sym,length(Data)/sym);
[m,n]=size(z);

% ------------------- MODULATION ------------------

QAM64Output=[];   % initializing output of the transmitter

for i=1:n
    if z([1,2,3,4,5,6],i)'==[0 0 0 0 0 0]
        QAM64Output=[QAM64Output,x1];
    elseif z([1,2,3,4,5,6],i)'==[0 0 0 0 0 1]
        QAM64Output=[QAM64Output,x2];
    elseif z([1,2,3,4,5,6],i)'==[0 0 0 0 1 0]
        QAM64Output=[QAM64Output,x3];
    elseif z([1,2,3,4,5,6],i)'==[0 0 0 0 1 1]
        QAM64Output=[QAM64Output,x4];
    elseif z([1,2,3,4,5,6],i)'==[0 0  0 1 0 0]
        QAM64Output=[QAM64Output,x5];
    elseif z([1,2,3,4,5,6],i)'==[0 0 0 1 0 1]
        QAM64Output=[QAM64Output,x6];
    elseif z([1,2,3,4,5,6],i)'==[0 0 0 1 1 0]
        QAM64Output=[QAM64Output,x7];
    elseif z([1,2,3,4,5,6],i)'==[0 0 0 1 1 1]
        QAM64Output=[QAM64Output,x8];
    elseif z([1,2,3,4,5,6],i)'==[0 0 1 0 0 0]
        QAM64Output=[QAM64Output,x9];
    elseif z([1,2,3,4,5,6],i)'==[0 0 1 0 0 1]
```

```
    QAM64Output=[QAM64Output,x10];
elseif z([1,2,3,4,5,6],i)'==[0 0 1 0 1 0]
    QAM64Output=[QAM64Output,x11];
elseif z([1,2,3,4,5,6],i)'==[0 0 1 0 1 1]
    QAM64Output=[QAM64Output,x12];
elseif z([1,2,3,4,5,6],i)'==[0 0 1 1 0 0]
    QAM64Output=[QAM64Output,x13];
elseif z([1,2,3,4,5,6],i)'==[0 0 1 1 0 1]
    QAM64Output=[QAM64Output,x14];
elseif z([1,2,3,4,5,6],i)'==[0 0 1 1 1 0]
    QAM64Output=[QAM64Output,x15];
elseif z([1,2,3,4,5,6],i)'==[0 0 1 1 1 1]
    QAM64Output=[QAM64Output,x16];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 0 0 0]
    QAM64Output=[QAM64Output,x17];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 0 0 1]
    QAM64Output=[QAM64Output,x18];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 0 1 0]
    QAM64Output=[QAM64Output,x19];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 0 1 1]
    QAM64Output=[QAM64Output,x20];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 1 0 0]
    QAM64Output=[QAM64Output,x21];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 1 0 1]
    QAM64Output=[QAM64Output,x22];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 1 1 0]
    QAM64Output=[QAM64Output,x23];
elseif z([1,2,3,4,5,6],i)'==[0 1 0 1 1 1]
    QAM64Output=[QAM64Output,x24];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 0 0 0]
    QAM64Output=[QAM64Output,x25];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 0 0 1]
    QAM64Output=[QAM64Output,x26];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 0 1 0]
    QAM64Output=[QAM64Output,x27];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 0 1 1]
    QAM64Output=[QAM64Output,x28];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 1 0 0]
    QAM64Output=[QAM64Output,x29];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 1 0 1]
    QAM64Output=[QAM64Output,x30];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 1 1 0]
    QAM64Output=[QAM64Output,x31];
elseif z([1,2,3,4,5,6],i)'==[0 1 1 1 1 1]
    QAM64Output=[QAM64Output,x32];
elseif z([1,2,3,4,5,6],i)'==[1 0 0 0 0 0]
    QAM64Output=[QAM64Output,x33];
```

```
elseif z([1,2,3,4,5,6],i)'==[1 0 0 0 0 1]
    QAM64Output=[QAM64Output,x34];
elseif z([1,2,3,4,5,6],i)'==[1 0 0 0 1 0]
    QAM64Output=[QAM64Output,x35];
elseif z([1,2,3,4,5,6],i)'==[1 0 0 0 1 1]
    QAM64Output=[QAM64Output,x36];
elseif z([1,2,3,4,5,6],i)'==[1 0 0 1 0 0]
    QAM64Output=[QAM64Output,x37];
elseif z([1,2,3,4,5,6],i)'==[1 0 0 1 0 1]
    QAM64Output=[QAM64Output,x38];
elseif z([1,2,3,4,5,6],i)'==[1 0 0 1 1 0]
    QAM64Output=[QAM64Output,x39];
elseif z([1,2,3,4,5,6],i)'==[1 0 0 1 1 1]
    QAM64Output=[QAM64Output,x40];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 0 0 0]
    QAM64Output=[QAM64Output,x41];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 0 0 1]
    QAM64Output=[QAM64Output,x42];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 0 1 0]
    QAM64Output=[QAM64Output,x43];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 0 1 1]
    QAM64Output=[QAM64Output,x44];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 1 0 0]
    QAM64Output=[QAM64Output,x45];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 1 0 1]
    QAM64Output=[QAM64Output,x46];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 1 1 0]
    QAM64Output=[QAM64Output,x47];
elseif z([1,2,3,4,5,6],i)'==[1 0 1 1 1 1]
    QAM64Output=[QAM64Output,x48];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 0 0 0]
    QAM64Output=[QAM64Output,x49];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 0 0 1]
    QAM64Output=[QAM64Output,x50];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 0 1 0]
    QAM64Output=[QAM64Output,x51];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 0 1 1]
    QAM64Output=[QAM64Output,x52];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 1 0 0]
    QAM64Output=[QAM64Output,x53];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 1 0 1]
    QAM64Output=[QAM64Output,x54];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 1 1 0]
    QAM64Output=[QAM64Output,x55];
elseif z([1,2,3,4,5,6],i)'==[1 1 0 1 1 1]
    QAM64Output=[QAM64Output,x56];
elseif z([1,2,3,4,5,6],i)'==[1 1 1 0 0 0]
```

```
        QAM64Output=[QAM64Output,x57];
    elseif z([1,2,3,4,5,6],i)'==[1 1 1 0 0 1]
        QAM64Output=[QAM64Output,x58];
    elseif z([1,2,3,4,5,6],i)'==[1 1 1 0 1 0]
        QAM64Output=[QAM64Output,x59];
    elseif z([1,2,3,4,5,6],i)'==[1 1 1 0 1 1]
        QAM64Output=[QAM64Output,x60];
    elseif z([1,2,3,4,5,6],i)'==[1 1 1 1 0 0]
        QAM64Output=[QAM64Output,x61];
    elseif z([1,2,3,4,5,6],i)'==[1 1 1 1 0 1]
        QAM64Output=[QAM64Output,x62];
    elseif z([1,2,3,4,5,6],i)'==[1 1 1 1 1 0]
        QAM64Output=[QAM64Output,x63];
    elseif z([1,2,3,4,5,6],i)'==[1 1 1 1 1 1]
        QAM64Output=[QAM64Output,x64];
    end
end


% ------- OUTPUT WAVEFORM THAT IS TRANSMITTED -----------

% plot(QAM64Output,'r');   % output waveform of modulator
% grid;
```

CODE FOR QAM64 DEMODULATOR

```
function [output_demod]=qam64_demod(output_mod,amp,phase)

% ------ KNOWN DEMODULATION PARAMETERS ---------------

freq=2;
t=0:0.01:1;                    % bit duration
T=length(t);

% ------------ CALCULATED DEMODULATION PARMETERS ----------

datalen=length(output_mod);    % total data points
pw=datalen/T;                  % duration data points
w=2*pi*freq;                   % obtained angular frequency

% -------------- DEMODULATION ---------------

output_demod=[];              % initializing bit stream
count=0;                      % step count
for i=1:pw                    % loop for next bit duration
    bit0=0;
```

```
bit1=0;
bit2=0;
bit3=0;
bit4=0;
bit5=0;
bit6=0;
bit7=0;
bit8=0;
bit9=0;
bit10=0;
bit11=0;
bit12=0;
bit13=0;
bit14=0;
bit15=0;
bit16=0;
bit17=0;
bit18=0;
bit19=0;
bit20=0;
bit21=0;
bit22=0;
bit23=0;
bit24=0;
bit25=0;
bit26=0;
bit27=0;
bit28=0;
bit29=0;
bit30=0;
bit31=0;
bit32=0;
bit33=0;
bit34=0;
bit35=0;
bit36=0;
bit37=0;
bit38=0;
bit39=0;
bit40=0;
bit41=0;
bit42=0;
bit43=0;
bit44=0;
bit45=0;
bit46=0;
bit47=0;
```

```
    bit48=0;
    bit49=0;
    bit50=0;
    bit51=0;
    bit52=0;
    bit53=0;
    bit54=0;
    bit55=0;
    bit56=0;
    bit57=0;
    bit58=0;
    bit59=0;
    bit60=0;
    bit61=0;
    bit62=0;
    bit63=0;
    bit64=0;

    for j=1:T
        if
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(1))
            bit0=bit0+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(2))
            bit1=bit1+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(3))
            bit2=bit2+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(4))
            bit3=bit3+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(5))
            bit4=bit4+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(6))
            bit5=bit5+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(7))
            bit6=bit6+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(8))
            bit7=bit7+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(9))
            bit8=bit8+1;
```

```
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(10))
            bit9=bit9+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(11))
            bit10=bit10+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(12))
            bit11=bit11+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(13))
            bit12=bit12+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(14))
            bit13=bit13+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(15))
            bit14=bit14+1;
        elseif
output_mod(count+j)==amp(1)*sin((w*t(j))+phase(16))
            bit15=bit15+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(1))
            bit16=bit16+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(2))
            bit17=bit17+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(3))
            bit18=bit18+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(4))
            bit19=bit19+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(5))
            bit20=bit20+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(6))
            bit21=bit21+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(7))
            bit22=bit22+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(8))
            bit23=bit23+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(9))
```

```
            bit24=bit24+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(10))
            bit25=bit25+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(11))
            bit26=bit26+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(12))
            bit27=bit27+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(13))
            bit28=bit28+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(14))
            bit29=bit29+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(15))
            bit30=bit30+1;
        elseif
output_mod(count+j)==amp(2)*sin((w*t(j))+phase(16))
            bit31=bit31+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(1))
            bit32=bit32+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(2))
            bit33=bit33+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(3))
            bit34=bit34+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(4))
            bit35=bit35+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(5))
            bit36=bit36+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(6))
            bit37=bit37+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(7))
            bit38=bit38+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(8))
            bit39=bit39+1;
```

```
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(9))
            bit40=bit40+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(10))
            bit41=bit41+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(11))
            bit42=bit42+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(12))
            bit43=bit43+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(13))
            bit44=bit44+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(14))
            bit45=bit45+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(15))
            bit46=bit46+1;
        elseif
output_mod(count+j)==amp(3)*sin((w*t(j))+phase(16))
            bit47=bit47+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(1))
            bit48=bit48+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(2))
            bit49=bit49+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(3))
            bit50=bit50+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(4))
            bit51=bit51+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(5))
            bit52=bit52+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(6))
            bit53=bit53+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(7))
            bit54=bit54+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(8))
```

```
            bit55=bit55+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(9))
            bit56=bit56+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(10))
            bit57=bit57+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(11))
            bit58=bit58+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(12))
            bit59=bit59+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(13))
            bit60=bit60+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(14))
            bit61=bit61+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(15))
            bit62=bit62+1;
        elseif
output_mod(count+j)==amp(4)*sin((w*t(j))+phase(16))
            bit63=bit63+1;
        end
    end

    count=count+T;
% count added with number of data points in one duration

    if bit0==T
        output_demod=[output_demod,0 0 0 0 0 0];
    elseif bit1==T
        output_demod=[output_demod,0 0 0 0 0 1];
    elseif bit2==T
        output_demod=[output_demod,0 0 0 0 1 0];
    elseif bit3==T
        output_demod=[output_demod,0 0 0 0 1 1];
    elseif bit4==T
        output_demod=[output_demod,0 0 0 1 0 0];
    elseif bit5==T-1
        output_demod=[output_demod,0 0 0 1 0 1];
    elseif bit6==T
        output_demod=[output_demod,0 0 0 1 1 0];
    elseif bit7==T
        output_demod=[output_demod,0 0 0 1 1 1];
```

```
elseif bit8==T
    output_demod=[output_demod,0 0 1 0 0 0];
elseif bit9==T
    output_demod=[output_demod,0 0 1 0 0 1];
elseif bit10==T
    output_demod=[output_demod,0 0 1 0 1 0];
elseif bit11==T
    output_demod=[output_demod,0 0 1 0 1 1];
elseif bit12==T
    output_demod=[output_demod,0 0 1 1 0 0];
elseif bit13==T
    output_demod=[output_demod,0 0 1 1 0 1];
elseif bit14==T
    output_demod=[output_demod,0 0 1 1 1 0];
elseif bit15==T
    output_demod=[output_demod,0 0 1 1 1 1];
elseif bit16==T-1
    output_demod=[output_demod,0 1 0 0 0 0];
elseif bit17==T
    output_demod=[output_demod,0 1 0 0 0 1];
elseif bit18==T
    output_demod=[output_demod,0 1 0 0 1 0];
elseif bit19==T
    output_demod=[output_demod,0 1 0 0 1 1];
elseif bit20==T
    output_demod=[output_demod,0 1 0 1 0 0];
elseif bit21==T-1
    output_demod=[output_demod,0 1 0 1 0 1];
elseif bit22==T
    output_demod=[output_demod,0 1 0 1 1 0];
elseif bit23==T
    output_demod=[output_demod,0 1 0 1 1 1];
elseif bit24==T-1
    output_demod=[output_demod,0 1 1 0 0 0];
elseif bit25==T
    output_demod=[output_demod,0 1 1 0 0 1];
elseif bit26==T
    output_demod=[output_demod,0 1 1 0 1 0];
elseif bit27==T
    output_demod=[output_demod,0 1 1 0 1 1];
elseif bit28==T
    output_demod=[output_demod,0 1 1 1 0 0];
elseif bit29==T
    output_demod=[output_demod,0 1 1 1 0 1];
elseif bit30==T
    output_demod=[output_demod,0 1 1 1 1 0];
elseif bit31==T
```

```
    output_demod=[output_demod,0 1 1 1 1 1];
elseif bit32==T-1
    output_demod=[output_demod,1 0 0 0 0 0];
elseif bit33==T
    output_demod=[output_demod,1 0 0 0 0 1];
elseif bit34==T
    output_demod=[output_demod,1 0 0 0 1 0];
elseif bit35==T
    output_demod=[output_demod,1 0 0 0 1 1];
elseif bit36==T
    output_demod=[output_demod,1 0 0 1 0 0];
elseif bit37==T
    output_demod=[output_demod,1 0 0 1 0 1];
elseif bit38==T
    output_demod=[output_demod,1 0 0 1 1 0];
elseif bit39==T
    output_demod=[output_demod,1 0 0 1 1 1];
elseif bit40==T-1
    output_demod=[output_demod,1 0 1 0 0 0];
elseif bit41==T
    output_demod=[output_demod,1 0 1 0 0 1];
elseif bit42==T
    output_demod=[output_demod,1 0 1 0 1 0];
elseif bit43==T
    output_demod=[output_demod,1 0 1 0 1 1];
elseif bit44==T
    output_demod=[output_demod,1 0 1 1 0 0];
elseif bit45==T
    output_demod=[output_demod,1 0 1 1 0 1];
elseif bit46==T
    output_demod=[output_demod,1 0 1 1 1 0];
elseif bit47==T
    output_demod=[output_demod,1 0 1 1 1 1];
elseif bit48==T-1
    output_demod=[output_demod,1 1 0 0 0 0];
elseif bit49==T
    output_demod=[output_demod,1 1 0 0 0 1];
elseif bit50==T
    output_demod=[output_demod,1 1 0 0 1 0];
elseif bit51==T
    output_demod=[output_demod,1 1 0 0 1 1];
elseif bit52==T
    output_demod=[output_demod,1 1 0 1 0 0];
elseif bit53==T
    output_demod=[output_demod,1 1 0 1 0 1];
elseif bit54==T
    output_demod=[output_demod,1 1 0 1 1 0];
```

```
    elseif bit55==T
        output_demod=[output_demod,1 1 0 1 1 1];
    elseif bit56==T-1
        output_demod=[output_demod,1 1 1 0 0 0];
    elseif bit57==T
        output_demod=[output_demod,1 1 1 0 0 1];
    elseif bit58==T
        output_demod=[output_demod,1 1 1 0 1 0];
    elseif bit59==T
        output_demod=[output_demod,1 1 1 0 1 1];
    elseif bit60==T
        output_demod=[output_demod,1 1 1 1 0 0];
    elseif bit61==T
        output_demod=[output_demod,1 1 1 1 0 1];
    elseif bit62==T
        output_demod=[output_demod,1 1 1 1 1 0];
    elseif bit63==T
        output_demod=[output_demod,1 1 1 1 1 1];
    end
    stem(output_demod,'m');
    title('DEMODULATED BITS TREAM');
    grid;
    drawnow;
end
```

CODE FOR AMPLITUDE HISTOGRAM

```
function [M,Amplitude]=AmpHist(ModulatorOutput)

t=0:0.01:1;
BitDuration=length(t);
DataLength=length(ModulatorOutput);
BitPoints=DataLength/BitDuration;

Amplitude=[];
count=0;
for i=1:BitPoints
    Points=[];
    for j=1:BitDuration
        points(j)=ModulatorOutput(count+j);
        Points=[Points,points(j)];
    end
    amp=max(Points);
    Amp=round(amp);
    Amplitude=[Amplitude,Amp];
    count=count+BitDuration;
```

```
    Histogram=hist(Amplitude);
    bar(Histogram,0.5);
    grid;
    drawnow;
    colormap([127/255 1 212/255]);
end

M=0;
for j=1:length(Histogram)
    if Histogram(j)~=0
        M=M+1;
    end
end
```

CODE FOR AMPLITUDE NORMALIZATION

```
function [NormalizedOutput]=amp_norm(output_mod)

t=0:0.01:1;
T=length(t);
DataLength=length(output_mod);
DataBits=DataLength/T;

count=0;             % initializing variables and vectors
NormalizedOutput=[];
amp=[];
points=[];

for i=1:DataBits
    for j=1:T
        Points(j)=output_mod(count+j);
        points=[points,Points(j)];
    end
    amp=max(points); % amplitude per symbol duration
    count=count+T;
    points=[];       % reinitializing to 0
    if amp~=0
        Normalization=Points/amp;    % normalizing
        NormalizedOutput=[NormalizedOutput,Normalization];
                    % accumulating if amplitude not equal 0
    elseif amp==0
        NormalizedOutput=[NormalizedOutput,Points];
                    % accumulating if amplitude 0
    end
    plot(NormalizedOutput,'r');
    grid;
    drawnow;
end
```

CODE FOR SORT AMPLITUDE ARRAY

```
function [SortAmp]=Ampl(Array)

amp=[];
count=0;
amp=Array(1);

for i=2:length(Array)
    if amp~=Array(i);
    amp=[amp,Array(i)];
    end
end
SortAmp=sort(amp);
```

CODE FOR AUDIO-TO-BINARY CONVERSION

```
function c=au2bin(s)

s=map(s,0,2);

a=dec2bin(s);

[m,n]=size(a);

c=[];
for i=1:m
    for j=1:n
        c=[c,a(i,j)];
    end
end

c=c';
c=str2num(c);
c=c';
```

CODE FOR BER CALCULATION

```
function [BER]=ber(s,a)


if length(s)<length(a)
    for i=1:length(s)
        z(i)=a(i);
    end
```

```
    sum1=sum(s~=z);
elseif length(a)<length(s)
    for i=1:length(a)
        z(i)=s(i);
    end
    sum1=sum(a~=z);
elseif length(a)==length(s)
    sum1=sum(a~=s);
end

BER=sum1/length(s)*100;
```

CODE FOR BINARY-TO-IMAGE CONVERSION

```
function [image]=bin2im(a,a1,a2)

% a = binary bits of the image
% a1 = width resolution
% a2 = height resolution

a=a';
a=num2str(a);
a=a';

b=length(a);
c=b/8;

n=1;
for i=1:c
    for j=1:8
        bin(i,j)=a(n);
        n=n+1;
    end
end

v=bin2dec(bin);

m=1;
for l=1:3  % because the bit depth for RGB is always 3
    for j=1:a1
        for k=1:a2
            image(k,j,l)=v(m);
            m=m+1;
        end
    end
end
```

CODE FOR IMAGE-TO-BINARY TRANSFORM

```
function c=im2bin(s)

a=dec2bin(s);

[m,n]=size(a);

c=[];
for i=1:m
    for j=1:n
        c=[c,a(i,j)];
    end
end

c=c';
c=str2num(c);
c=c';
```

CODE FOR STRING-TO-BINARY CONVERSION

```
function [c]=str2bin(word)

u=double(word);
u=hex(u);
v=hex2dec(u);
bin=dec2bin(v);
[m,n]=size(bin);

c=[];
for i=1:m
    for j=1:n
        c=[c,bin(i,j)];
    end
end

c=c';
c=str2num(c);
c=c';

% ------------------------------------------------------

function [u] = hex(u)

u=u';
a=u;
u=num2str(u);
```

```
for i=1:length(u)
    r=mod(a(i),100);
    if(r>=0 & r<10 )
        m=num2str(a(i));
        u(i,:)=strcat('0a',m(1,3));
    elseif(r>=10 & r<20)
        m=num2str(a(i));
        u(i,:)=strcat('0b',m(1,3));
    elseif(r>=20 & r<29)
        m=num2str(a(i));
        u(i,:)=strcat('0c',m(1,3));
    elseif(r>=30 & r<39)
        m=num2str(a(i));
        u(i,:)=strcat('0d',m(1,3));
    elseif(r>=40 & r<49)
        m=num2str(a(i));
        u(i,:)=strcat('0e',m(1,3));
    elseif(r>=50 & r<59)
        m=num2str(a(i));
        u(i,:)=strcat('0f',m(1,3));
    end
end
```

CODE FOR BINARY-TO-STRING CONVERSION

```
function [str]=bin2str(nbin)

nbin=nbin';
nbin=num2str(nbin);
nbin=nbin';

l=length(nbin)/8;
n=1;
for(i=1:l)
    for(j=1:8)
        bin(i,j)=nbin(n);
        n=n+1;
    end
end

v=bin2dec(bin);
w=dec2hex(v);
len=length(w);
x=zeros(len,1);
x=num2str(x);
```

```
x(:,2)=w(:,1);
x(:,3)=w(:,2);

for(i=1:len)
    if x(i,2)=='A'
        x(i,:)=strcat('10',x(i,3));
    elseif x(i,2)=='B'
        x(i,:)=strcat('11',x(i,3));
    elseif x(i,2)=='C'
        x(i,:)=strcat('12',x(i,3));
    elseif x(i,2)=='D'
        x(i,:)=strcat('13',x(i,3));
    elseif x(i,2)=='E'
        x(i,:)=strcat('14',x(i,3));
    elseif x(i,2)=='F'
        x(i,:)=strcat('15',x(i,3));
    end
end
x=str2num(x)';
str=char(x);
```

CODE FOR DATA DISTRIBUTION FOR MODULATION

```
function [y]=datadist(s,r)

DataLength=length(s);

switch r
    case 2
        d=1;
        a=floor(length(s)/d);
        n=0;
        y=[];
        for i=1:a
            z=ask2_mod(s(1,1+n:d+n));
            n=n+d;
            y=[y,z];
            plot(y,'r');
            grid;
            drawnow;
        end
        save('man','y');
    case 3
        sym=2;
        r=rem(DataLength,sym);
```

```
    if r~=0
        r1=sym-r;
        r2=zeros(1,r1);
        s=[s,r2];
    end
    a=floor(length(s)/sym);
    n=0;
    y=[];
    for i=1:a
        z=ask4_mod(s(1,1+n:sym+n));
        n=n+sym;
        y=[y,z];
        plot(y,'r');
        grid;
        drawnow;
    end
    save('man','y');
case 4
    sym=3;
    r=rem(DataLength,sym);
    if r~=0
        r1=sym-r;
        r2=zeros(1,r1);
        s=[s,r2];
    end
    a=floor(length(s)/sym);
    n=0;
    y=[];
    for i=1:a
        z=ask8_mod(s(1,1+n:sym+n));
        n=n+sym;
        y=[y,z];
        plot(y,'r');
        grid;
        drawnow;
    end
    save('man','y');
case 5
    d=1;
    a=floor(length(s)/d);
    n=0;
    y=[];
    for i=1:a
        z=psk2_mod(s(1,1+n:d+n));
        n=n+d;
        y=[y,z];
        plot(y,'r');
```

```
        grid;
        drawnow;
    end
    save('man','y');
case 6
    sym=2;
    r=rem(DataLength,sym);
    if r~=0
        r1=sym-r;
        r2=zeros(1,r1);
        s=[s,r2];
    end
    a=floor(length(s)/sym);
    n=0;
    y=[];
    for i=1:a
        z=psk4_mod(s(1,1+n:sym+n));
        n=n+sym;
        y=[y,z];
        plot(y,'r');
        grid;
        drawnow;
    end
    save('man','y');
case 7
    sym=3;
    r=rem(DataLength,sym);
    if r~=0
        r1=sym-r;
        r2=zeros(1,r1);
        s=[s,r2];
    end
    a=floor(length(s)/sym);
    n=0;
    y=[];
    for i=1:a
        z=psk8_mod(s(1,1+n:sym+n));
        n=n+sym;
        y=[y,z];
        plot(y,'r');
        grid;
        drawnow;
    end
    save('man','y');
case 8
    sym=2;
    r=rem(DataLength,sym);
```

```
    if r~=0
        r1=sym-r;
        r2=zeros(1,r1);
        s=[s,r2];
    end
    a=floor(length(s)/sym);
    n=0;
    y=[];
    for i=1:a
        z=qam4_mod(s(1,1+n:sym+n));
        n=n+sym;
        y=[y,z];
        plot(y,'r');
        grid;
        drawnow;
    end
    save('man','y');
case 9
    sym=3;
    r=rem(DataLength,sym);
    if r~=0
        r1=sym-r;
        r2=zeros(1,r1);
        s=[s,r2];
    end
    a=floor(length(s)/sym);
    n=0;
    y=[];
    for i=1:a
        z=qam8_mod(s(1,1+n:sym+n));
        n=n+sym;
        y=[y,z];
        plot(y,'r');
        grid;
        drawnow;
    end
    save('man','y');
case 10
    sym=4;
    r=rem(DataLength,sym);
    if r~=0
        r1=sym-r;
        r2=zeros(1,r1);
        s=[s,r2];
    end
    a=floor(length(s)/sym);
    n=0;
```

```
        y=[];
        for i=1:a
            z=qam16_mod(s(1,1+n:sym+n));
            n=n+sym;
            y=[y,z];
            plot(y,'r');
            grid;
            drawnow;
        end
        save('man','y');
    case 11
        sym=5;
        r=rem(DataLength,sym);
        if r~=0
            r1=sym-r;
            r2=zeros(1,r1);
            s=[s,r2];
        end

        a=floor(length(s)/sym);
        n=0;
        y=[];
        for i=1:a
            z=qam32_mod(s(1,1+n:sym+n));
            n=n+sym;
            y=[y,z];
            plot(y,'r');
            grid;
            drawnow;
        end
        save('man','y');
end
```

CODE FOR FOURIER TRANSFORM

```
function [FOP]=fop(mod_op)

a=fft(mod_op);
FOP=abs(a);
```

CODE FOR MAPPING AUDIO SIGNAL

```
function xx=map(kk,low,high)

zz=(kk-min(min(kk))./(max(max(kk))-min(min(kk)));
```

```
xx=zz.*(high-low)+low;
```

CODE FOR PHASE HISTOGRAM

```
function [M,Phase]=PhaseDetect(output_mod)

t=0:0.01:1;
T=length(t);

DataLength=length(output_mod);
DataBits=DataLength/T;

count=0;
Phase=[];
count=0;
amp1=[];

for i=1:DataBits
    for j=1:T
        Points(j)=output_mod(count+j);
        amp1=[amp1,Points(j)];
    end
    amp=amp1(T);            % amplitude per symbol duration
    if abs(amp)<0.01
        amp=0;
    end
    y=asin(amp);
    z=180*y/pi;
    Phase=[Phase,z];
    count=count+T;
    amp1=[];                % reinitializing to 0
    Histogram=hist(Phase);
    bar(Histogram,0.5);
    grid;
    colormap([127/255 1 212/255]);
    drawnow;
end

M=0;
for j=1:length(Histogram)
    if Histogram(j)~=0
        M=M+1;
    end
end
```

CODE FOR ASSIGNING PHASES

```
function [Phase]=Phase(M)

if M==1
    Phase=[0 pi];
elseif M==3
    Phase=[0 pi/2 pi 3*pi/2];
elseif M==5
    Phase=[0 pi/4 pi/2 3*pi/4 pi 5*pi/4 3*pi/2 7*pi/4];
elseif M==8
    Phase=[0 pi/8 pi/4 3*pi/8 pi/2 5*pi/8 3*pi/4 7*pi/8 pi
9*pi/8 5*pi/4 11*pi/8 4*pi/3 13*pi/8 7*pi/8 15*pi/8];
end
```

CODE FOR AUDIO RECORD

```
function s=recsound

r=audiorecorder(8912,8,1);
recordblocking(r,1);
s=getaudiodata(r);
```

CODE FOR AUTO MODULATION SELECTION

```
function [y,X,a]=transmitter(s)

a=randint(1,1,10);
if a==0
    [y]=ask2_mod(s);
    X=('MODULATION USED: ASK2');
elseif a==1
    [y]=ask4_mod(s);
    X=('MODULATION USED: ASK4');
elseif a==2
    [y]=ask8_mod(s);
    X=('MODULATION USED: ASK8');
elseif a==3
    [y]=psk2_mod(s);
    X=('MODULATION USED: PSK2');
elseif a==4
    [y]=psk4_mod(s);
    X=('MODULATION USED: PSK4');
elseif a==5
```

```
    [y]=psk8_mod(s);
    X=('MODULATION USED: PSK8');
elseif a==6
    [y]=qam8_mod(s);
    X=('MODULATION USED: QAM8');
elseif a==7
    [y]=qam16_mod(s);
    X=('MODULATION USED: QAM16');
elseif a==8
    [y]=qam4_mod(s);
    X=('MODULATION USED: QAM4');
elseif a==9
    [y]=qam32_mod(s);
    X=('MODULATION USED: QAM32');
end
```

CODE FOR AUTO MODULATION IDENTIFICATION AND DEMODULATION

```
function
[M,N,P,X,NormalizedOutput,Amplitude,Ph,Amp1,Fop]=algorithm(
ModulatorOutput)

t=0:0.01:1;
T=length(t);

[M,Amplitude]=AmpHist(ModulatorOutput);
[NormalizedOutput]=amp_norm(ModulatorOutput);
[N,Amp]=AmpHist(NormalizedOutput);
Amp1=Ampl(Amplitude);
P=PhaseDetect(NormalizedOutput);
Ph=Phase(P);
save('array','Amplitude','P');
[Fop]=fop(ModulatorOutput);

if M==2 & N==2 & P==1
    X=('IDENTIFIED MODULATION : ASK2');
elseif M==4 & N==2 & P==1
        X=('IDENTIFIED MODULATION : ASK4');
elseif M==8 & N==2 & P==1
        X=('IDENTIFIED MODULATION : ASK8');
elseif M==1 & N==1 & P==1
        X=('IDENTIFIED MODULATION : PSK2');
elseif M==1 & N==1 & P==3
        X=('IDENTIFIED MODULATION : PSK4');
elseif M==1 & N==1 & P==5
```

```
        X=('IDENTIFIED MODULATION : PSK8');
elseif M==2 & N==1 & P==3
        X=('IDENTIFIED MODULATION : QAM8');
elseif M==2 & N==1 & P==5
        X=('IDENTIFIED MODULATION : QAM16');
elseif M==2 & N==1 & P==1
        X=('IDENTIFIED MODULATION : QAM4');
elseif M==4 & N==1 & P==5
        X=('IDENTIFIED MODULATION : QAM32');
end
```