Digital Communication Lab

Laboratory report submitted for the partial fulfillment
of the requirements for the degree of

Bachelor of Technology
in
Electronics and Communication Engineering

by

Mohit Akhouri - 19ucc023

Course Coordinator
Dr. Anirudh Agarwal , Dr. Nikhil Sharma

Department of Electronics and Communication Engineering
The LNM Institute of Information Technology, Jaipur

November 2021

Copyright © The LNMIIT 2021
All Rights Reserved

Contents

Chapter *1*

Experiment - 1

1.1 Name of the Experiment

To implement various Line Coding Schemes on MATLAB Simulink and analyse the parametric changes

1.2 Software Used

- MATLAB
- Simulink

1.3 Theory

In telecommunication, a **line code** is a pattern of voltage, current, or photons used to represent digital data transmitted down a transmission line.This process of coding is chosen so as to avoid overlap and distortion of signal such as **inter-symbol interference**. Line coding is carried out by a transmitter that converts data, in the form of binary digits, into a baseband digital signal that will represent the data on a transmission line.There are many different line coding techniques, ranging in complexity from very basic unipolar schemes in which the presence or absence of a voltage is used to represent a binary one or a binary zero, to highly sophisticated multilevel schemes in which different signal amplitudes are used, each representing a unique grouping of binary digits.
There are **3** types of Line Coding :

- Uni-polar
- Polar
- Bi-polar

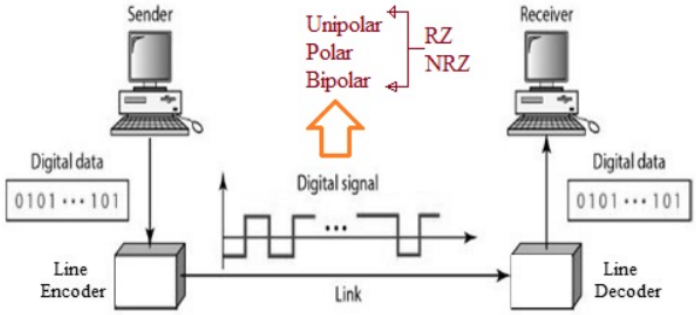

Figure 1.1 Line encoding and decoding

1.3.1 About Uni-polar Signalling :

Uni-polar signalling is also called as ON-OFF Keying (**OOK**). In this type of Line Coding scheme, presence of pulse is represented by 1 and absence of pulse is represented by 0. The bandwidth of a unipolar signal is inversely proportional to the duration of each data bit. There are two types of Uni-polar Signalling :

- Unipolar NRZ (Non-return to zero)
- Unipolar RZ (return to zero)

1.3.1.1 About Unipolar NRZ :

It is unipolar line coding scheme in which positive voltage defines bit 1 and the zero voltage defines bit 0. Signal does not return to zero at the middle of the **MARK bit** (1) thus it is called NRZ. In this type of line coding scheme , the pulse duration (τ) is **equal** to symbol duration (T_0). This scheme applies a DC bias to the line and unnecessarily wastes power.

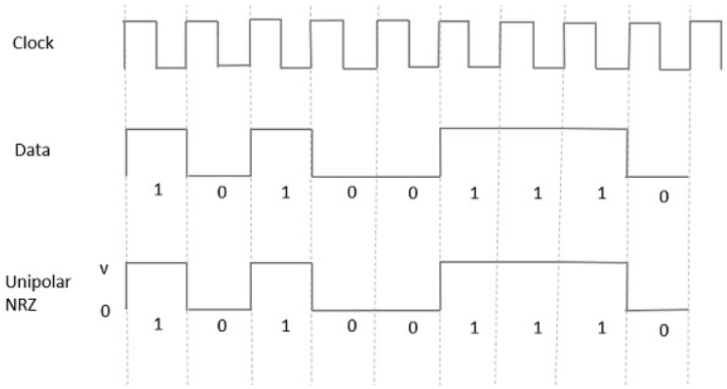

Figure 1.2 Unipolar NRZ Signalling

1.3.1.2 About Unipolar RZ :

In this type of signalling scheme, binary **one** is represented by pulse with high to low transition.Here during entire bit period, initially pulse remains high in the first half period and returns to zero in the next half bit duration. Binary **zero** is represented as absence of pulse. In this type of signalling, the pulse duration (τ) is **less than** to symbol duration (T_0).

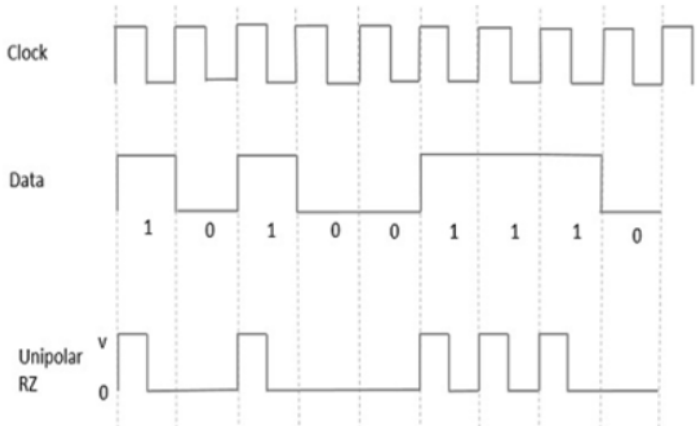

Figure 1.3 Unipolar RZ Signalling

1.3.2 About Polar Signalling:

Polar line coding schemes use both positive and negative voltage levels to represent binary values.Polar encoding uses two voltage levels. Binary 1 is represented by positive pulse **+V** and binary 0 is represented by negative (**antipodal**) pulse **-V**.
Polar signalling has many benefits over Unipolar signalling which are as follows :

- NRZ-I helps in synchronization at the receiver due to use of transition to map binary **1**.
- Level voltage representation helps in reducing DC voltage.
- Simplicity in implementation.

There are two types of Polar Signalling :

- Polar NRZ (Non-return to zero)
- Polar RZ (return to zero)

1.3.2.1 About Polar NRZ:

In this type of polar signalling , binary **one** maps to **+V** voltage level and binary **zero** to **-V** voltage level.

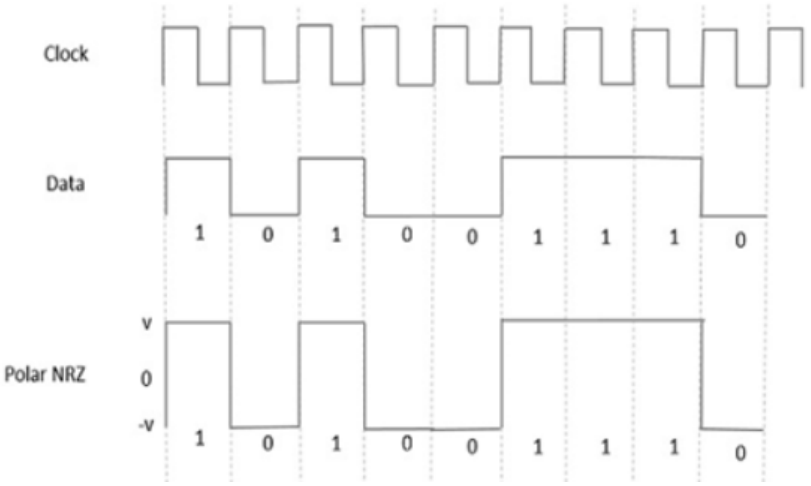

Figure 1.4 Polar NRZ Signalling

1.3.2.2 About Polar RZ:

In this type of polar signalling , binary **one** maps to **+V** for first half of the bit duration and returns to **0** for next half of the bit duration . Similarly, binary **zero** maps to **-V** for first half of the bit duration and returns to **0** for next half of the bit duration.

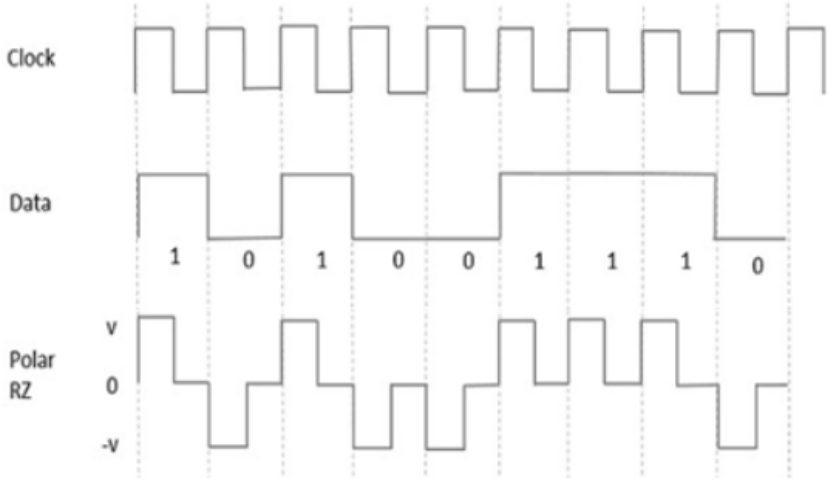

Figure 1.5 Polar RZ Signalling

1.3.3 About Bipolar Signalling:

In telecommunication, bipolar encoding is a type of return-to-zero (RZ) line code, where two nonzero values are used, so that the three values are +, , and zero. Such a signal is called a **duobinary** signal. Standard bipolar encodings are designed to be DC-balanced, spending equal amounts of time in the + and states.

1.3.3.1 Alternate Mark Inversion:

One kind of bipolar encoding is a paired disparity code, of which the simplest example is alternate mark inversion. In this code, a binary 0 is encoded as zero volts, as in unipolar encoding, whereas a binary 1 is encoded **alternately** as a positive voltage or a negative voltage. The name arose because, in the context of a T-carrier, a binary **1** is referred to as a **MARK**, while a binary **0** is called a **space**.

1.3.3.2 About Bipolar NRZ :

In this type of bipolar signalling , there are 3 voltage levels (0, +V and -V). In this scheme, binary **one** is represented **alternately** by +V and -V. There is **no return to zero** in middle of the bit duration of **MARK** pulse. Binary **zero** is represented by **0** voltage level.

1.3.3.3 About Bipolar RZ :

This is just the opposite of Bipolar NRZ , in this scheme, During the bit duration of the **MARK Pulse** , it returns to **zero** for second half of the duration and remains +V or -V (**alternately**) for first half of the duration.

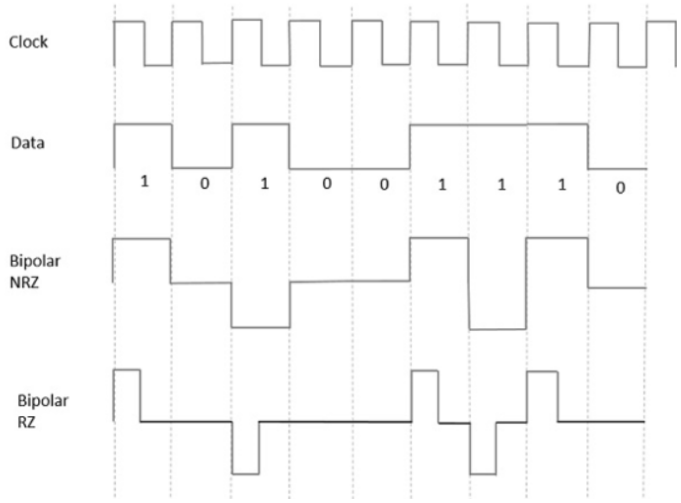

Figure 1.6 Bipolar Line coding scheme

1.3.4 About Manchester Signalling:

In telecommunication and data storage, Manchester code (also known as phase encoding, or PE) is a line code in which the encoding of each data bit is either low then high, or high then low, for equal time. It is a self-clocking signal with no DC component. Consequently, electrical connections using a Manchester code are easily galvanically isolated. Manchester code derives its name from its development at the **University of Manchester**, where the coding was used for storing data on the magnetic drums of the **Manchester Mark 1 computer**. Manchester code was widely used for magnetic recording on 1600 bpi computer tapes before the introduction of 6250 bpi tapes which used the more efficient group-coded recording.[1] Manchester code was used in early Ethernet physical layer standards and is still used in **consumer IR protocols**, **RFID** and **near-field communication**.

1.3.4.1 Features of Manchester Signalling:

Manchester coding is a special case of **binary phase-shift keying** (BPSK), where the data controls the phase of a square wave carrier whose frequency is the data rate. Manchester code ensures frequent line voltage transitions, directly proportional to the clock rate; this helps clock recovery.

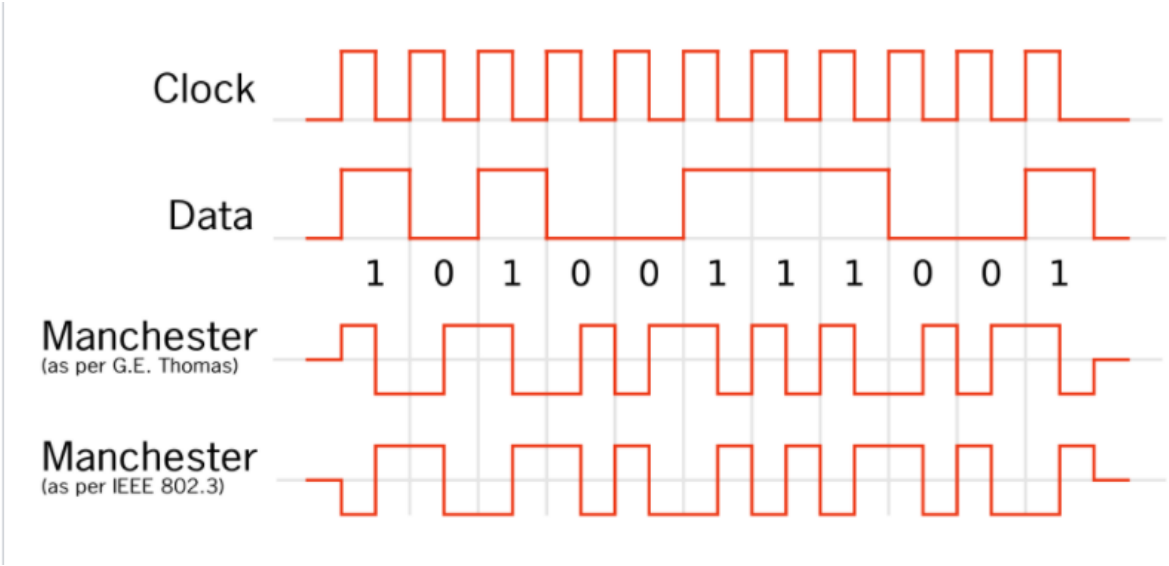

Figure 1.7 Manchester Line Coding Scheme

1.4 Code and Results

1.4.1 Unipolar RZ and Polar RZ Coding

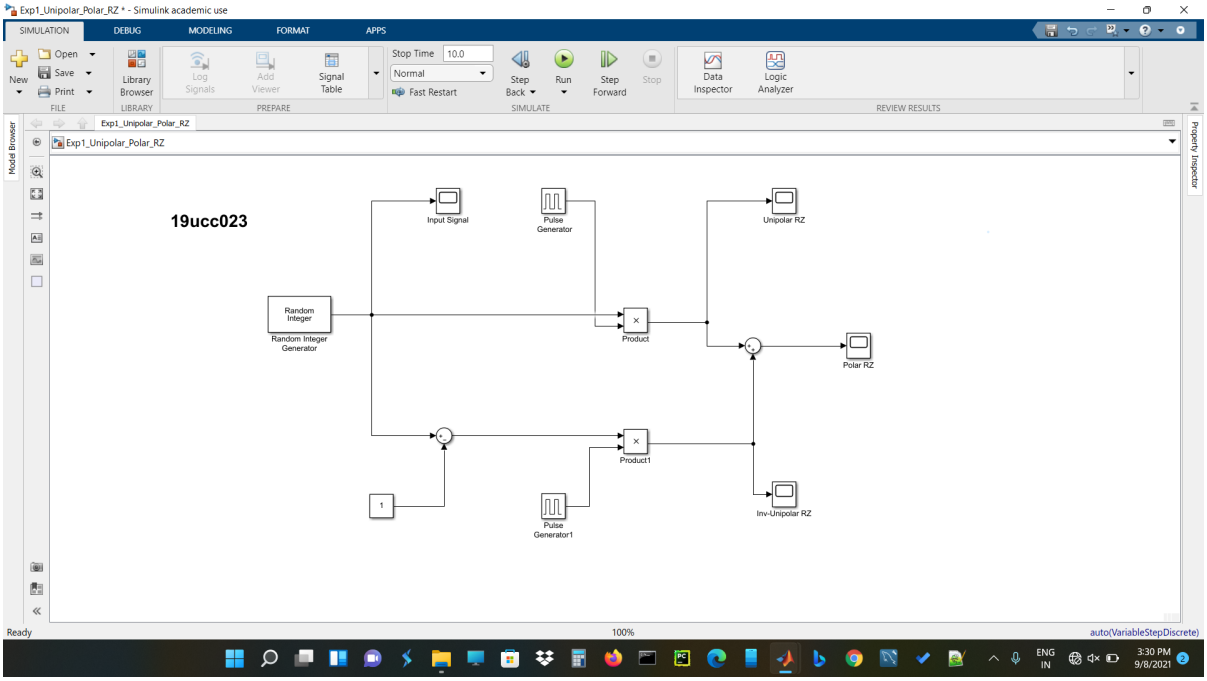

Figure 1.8 Unipolar RZ and Polar RZ Coding Simulink block diagram

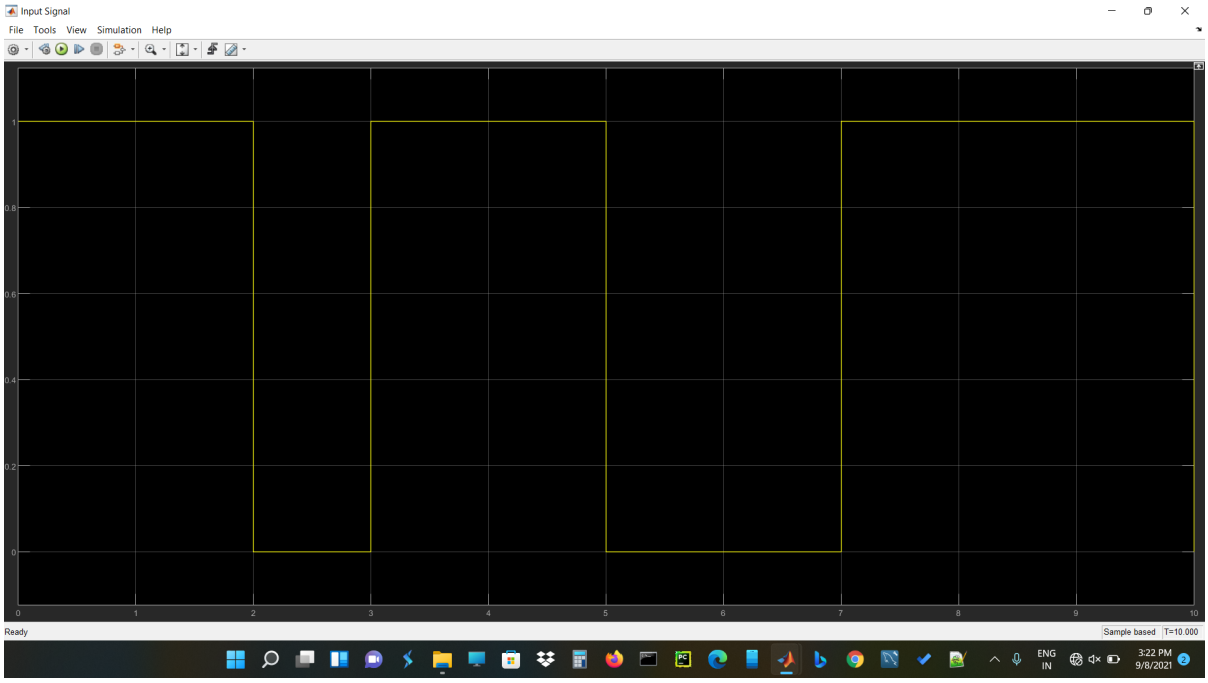

Figure 1.9 Plot of the Input Signal Block

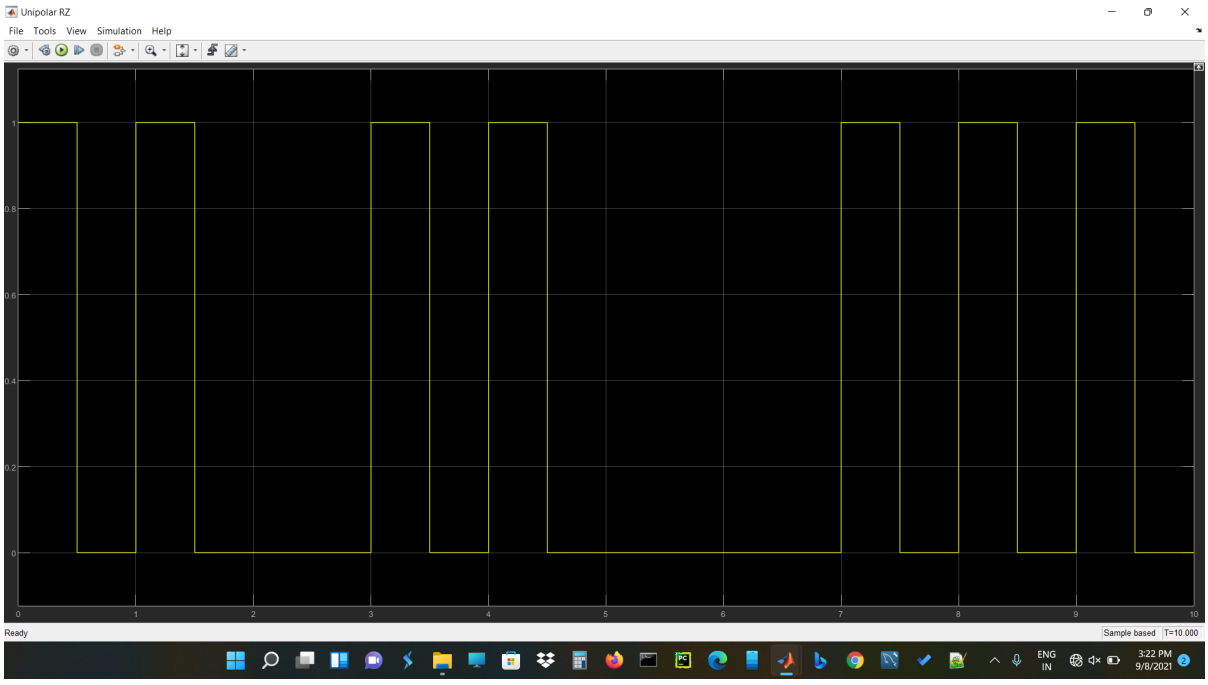

Figure 1.10 Plot of the Unipolar RZ

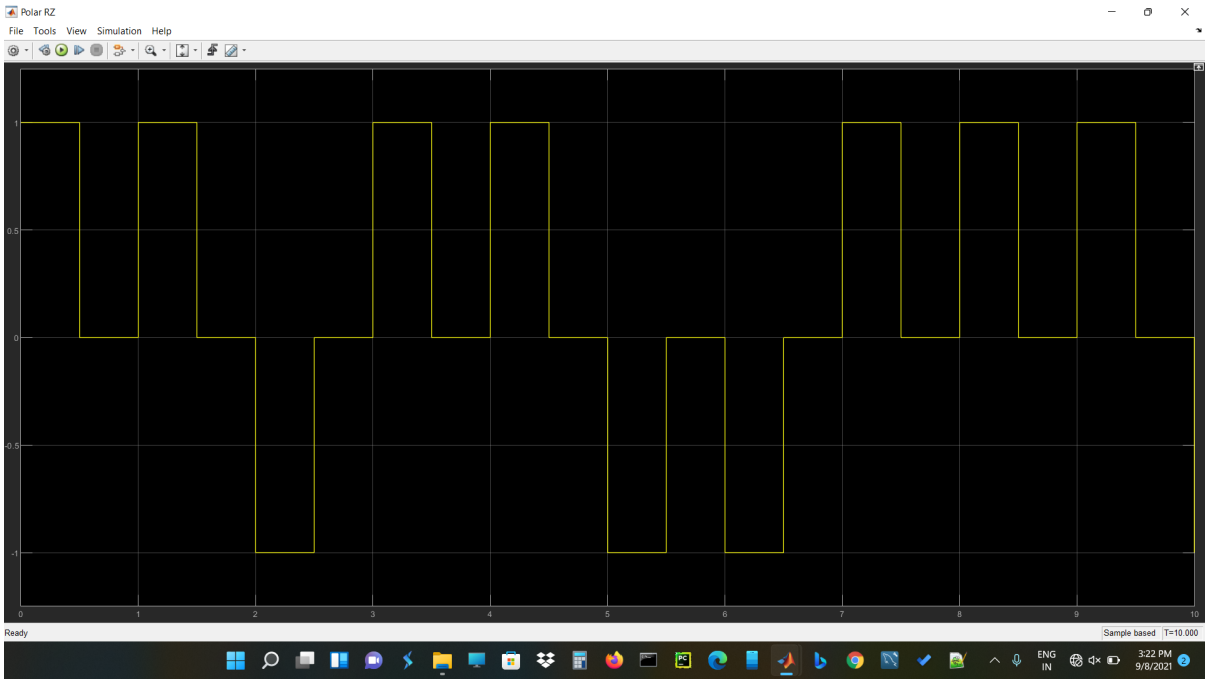

Figure 1.11 Plot of the Polar RZ

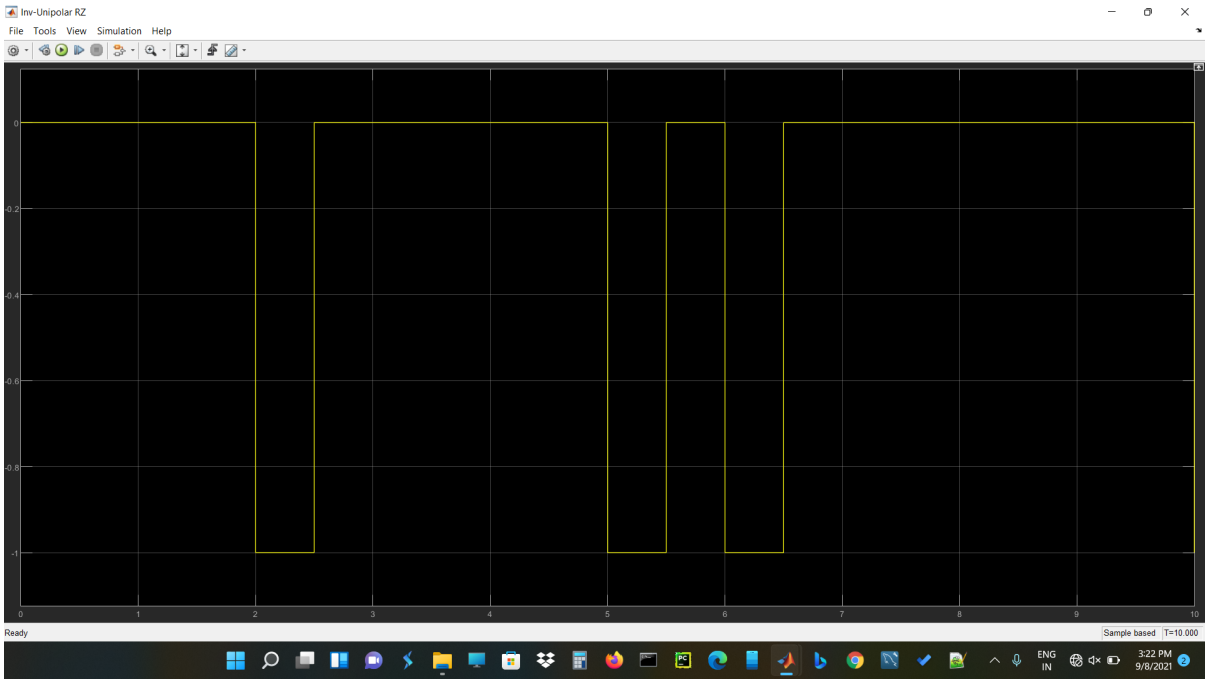

Figure 1.12 Plot of the Inverse Unipolar RZ

1.4.2 Bipolar NRZ-RZ Coding

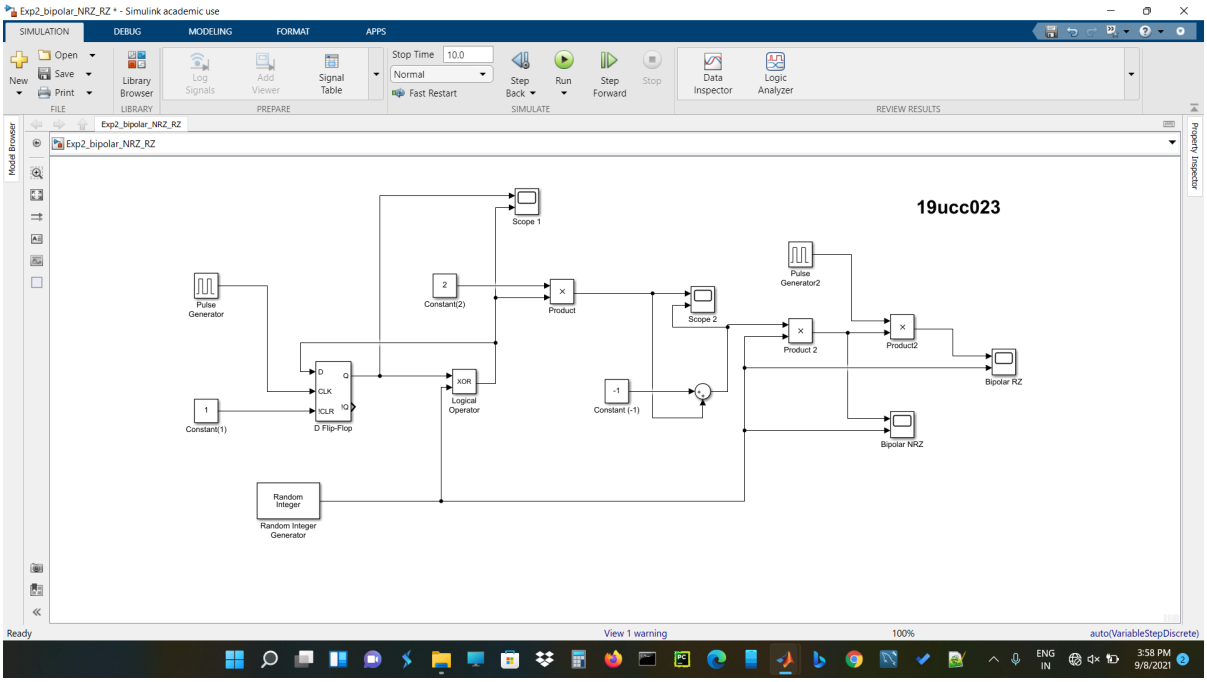

Figure 1.13 Bipolar NRZ-RZ Coding Simulink block diagram

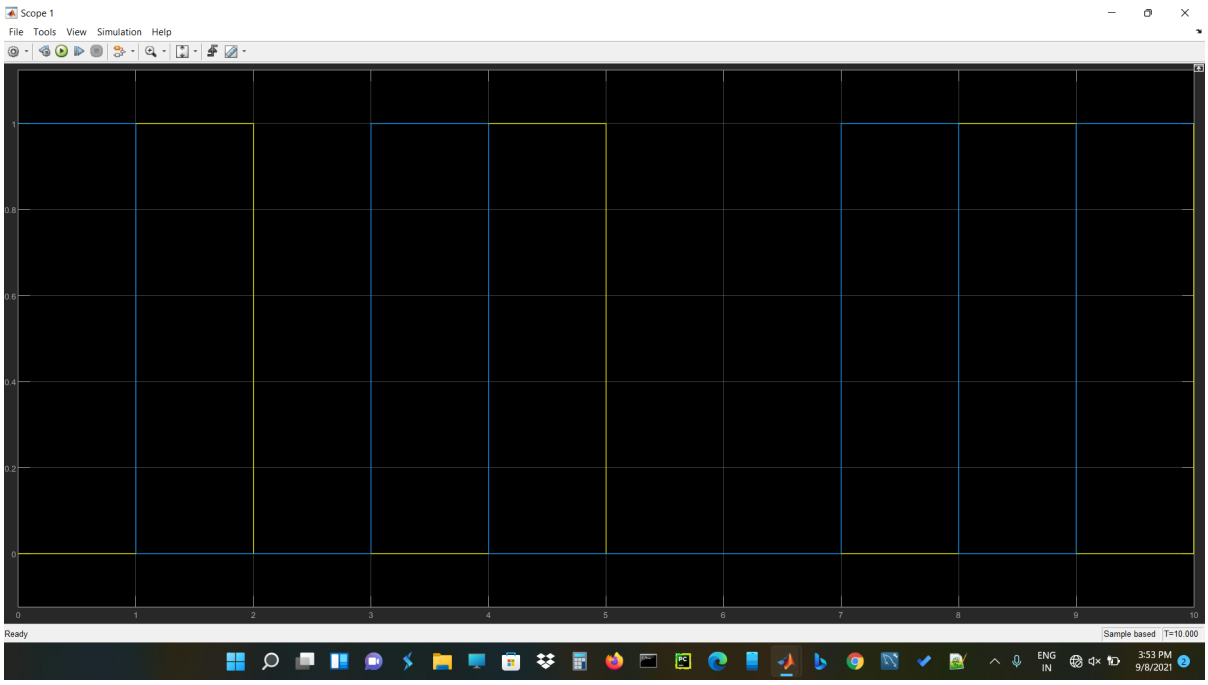

Figure 1.14 Plot of the Scope 1 in the simulink block diagram

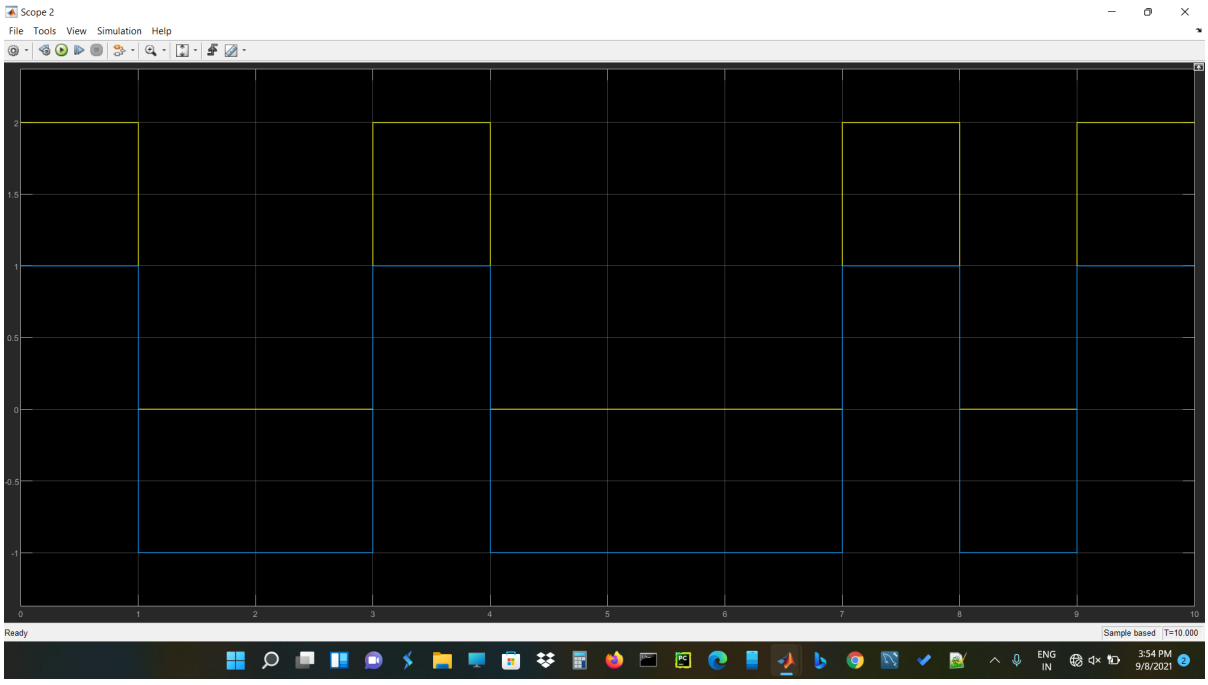

Figure 1.15 Plot of the Scope 2 in the simulink block diagram

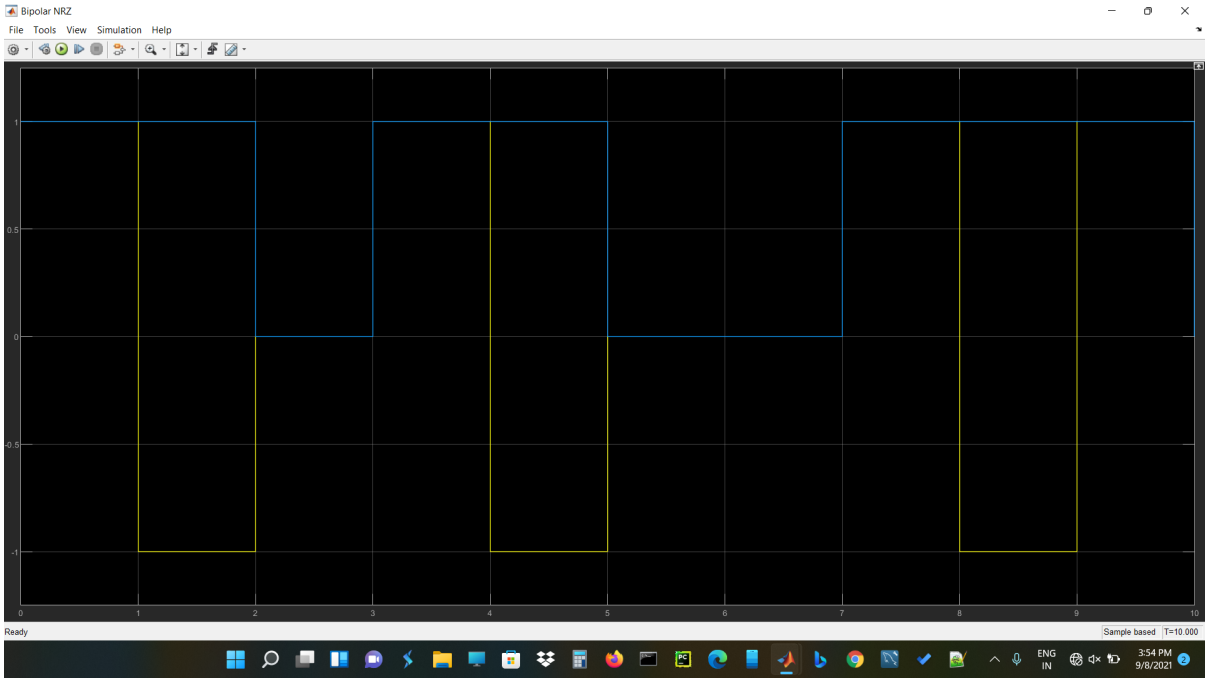

Figure 1.16 Plot of the Bipolar NRZ

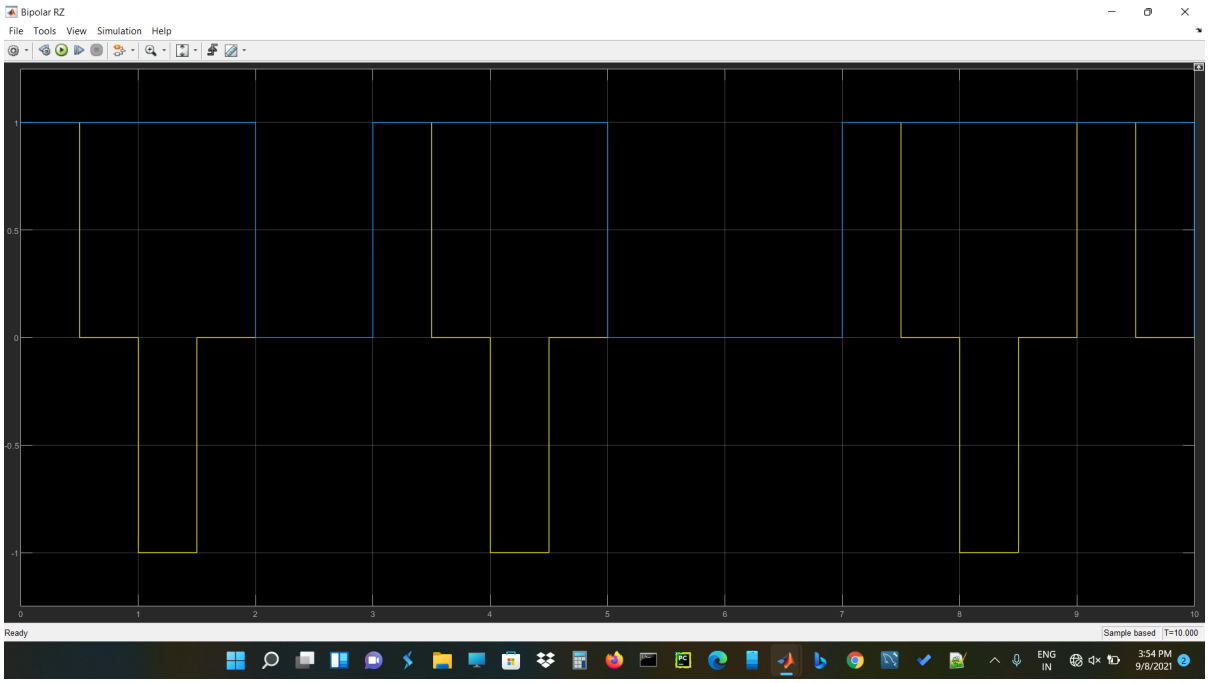

Figure 1.17 Plot of the Bipolar RZ

1.5 Conclusion

In this experiment, we have learnt about various **Line Coding Schemes** such as **Unipolar** , **Bipolar** and **Manchester** signalling. We also learnt about the Simulink software and how to implement various circuits using block diagrams. We learnt about new blocks of simulink such as Random Integer Generator, Scope and D-FlipFlop. We also learnt about how to simulate the block diagram and generate plots.

Chapter *2*

Experiment - 2

2.1 Name of the Experiment

- To implement Pulse Code Modulation (PCM) and Demodulation on MATLAB Simulink platform
- To implement Delta Modulation (DM) and Demodulation on MATLAB Simulink platform

2.2 Software Used

- MATLAB
- Simulink

2.3 Theory

2.3.1 About Pulse Code Modulation (PCM) :

Pulse-code modulation (PCM) is a method used to **digitally** represent sampled analog signals. It is the standard form of digital audio in computers, compact discs, digital telephony and other digital audio applications. In a PCM stream, the **amplitude** of the analog signal is **sampled** regularly at uniform intervals, and each sample is **quantized** to the **nearest value** within a range of digital steps. A PCM stream has two basic properties that determine the stream's fidelity to the original analog signal: the **sampling rate**, which is the number of times per second that samples are taken and the **bit depth**, which determines the number of possible digital values that can be used to represent each sample.

The stream of pulses and non-pulse streams of 1's and 0's are **not easily affected** by **interference** and **noise**. Even in the presence of noise, the presence or absence of a pulse can be easily determined. Since PCM is digital, a more general reason would be that digital signals are easy to process by cheap standard techniques. This makes it easier to implement complicated communication systems such as **telephone networks**.

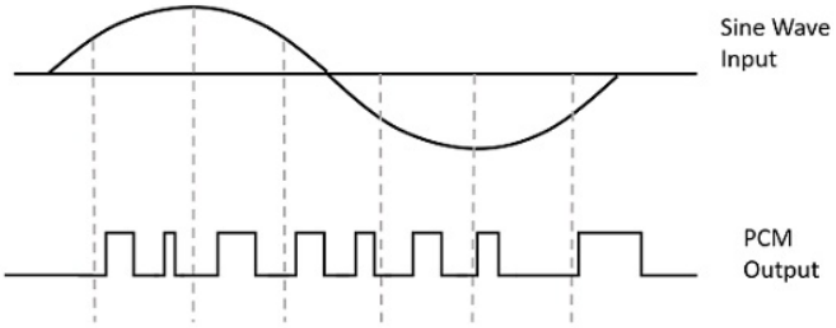

Figure 2.1 Pulse Code Modulation of Sine wave

2.3.1.1 Basic Elements of PCM :

The **transmitter section** of a Pulse Code Modulator circuit consists of **Sampling**, **Quantizing** and **Encoding**, which are performed in the **analog-to-digital converter** section. The **low pass filter** prior to sampling prevents aliasing of the message signal.

The basic operations in the **receiver section** are **regeneration** of impaired signals, **decoding**, and **reconstruction** of the quantized pulse train. Following is the block diagram of PCM which represents the basic elements of both the transmitter and the receiver sections.

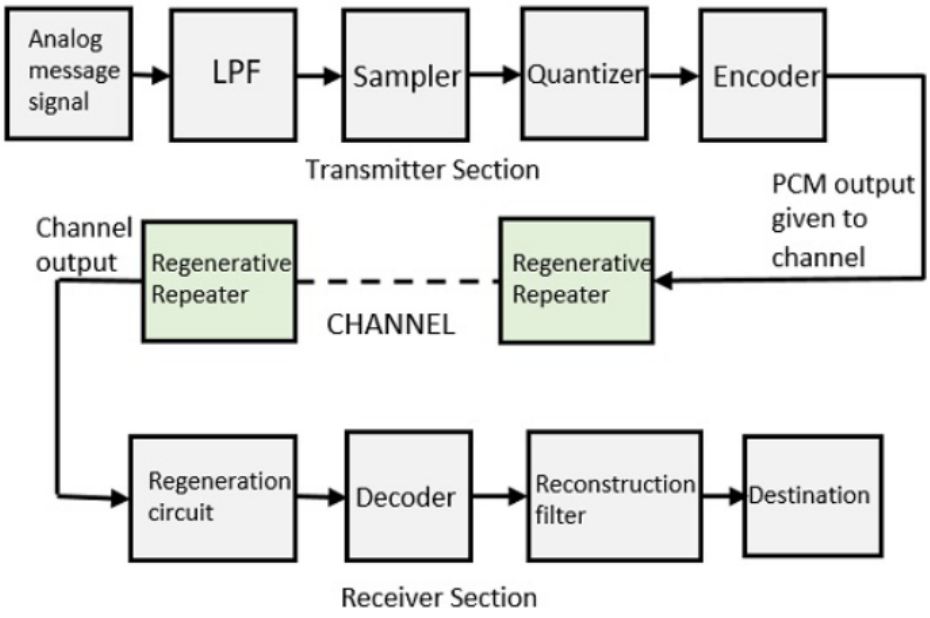

Figure 2.2 Basic Elements of PCM

2.3.1.2 Implementation of PCM :

- LPCM is used for the **lossless encoding** of audio data in the Compact disc Red Book standard (informally also known as Audio CD), introduced in 1982.
- **LaserDiscs** with digital sound have an LPCM track on the digital channel.
- LPCM is used by **HDMI** (defined in 2002), a single-cable digital audio/video connector interface for transmitting uncompressed digital data.

2.3.1.3 Limitations of PCM :

- Choosing a discrete value that is near but not exactly at the analog signal level for each sample leads to **quantization error**.
- As samples are dependent on time, an accurate **clock** is **required** for accurate reproduction. If either the encoding or decoding clock is not stable, these imperfections will directly affect the output quality of the device.
- Between samples no measurement of the signal is made; the **sampling theorem** guarantees non-ambiguous representation and recovery of the signal only if it has no energy at frequency $\frac{f_s}{2}$ or higher (one half the sampling frequency, known as the Nyquist frequency); higher frequencies will not be correctly represented or recovered and add aliasing distortion to the signal below the **Nyquist frequency**.

2.3.2 About Delta Modulation :

A **delta modulation** (DM or Δ-modulation) is an analog-to-digital and digital-to-analog signal conversion technique used for **transmission** of **voice information** where quality is not of primary importance. DM is the simplest form of **differential pulse-code modulation (DPCM)** where the difference between successive samples is encoded into n-bit data streams. In delta modulation, the transmitted data are reduced to a 1-bit data stream.

Main Features of Delta Modulation are :

- The analog signal is approximated with a series of segments.
- Each segment of the approximated signal is compared to the preceding bits and the successive bits are determined by this comparison.
- Only the change of information is sent, that is, only an increase or decrease of the signal amplitude from the previous sample is sent whereas a no-change condition causes the modulated signal to remain at the same 0 or 1 state of the previous sample.

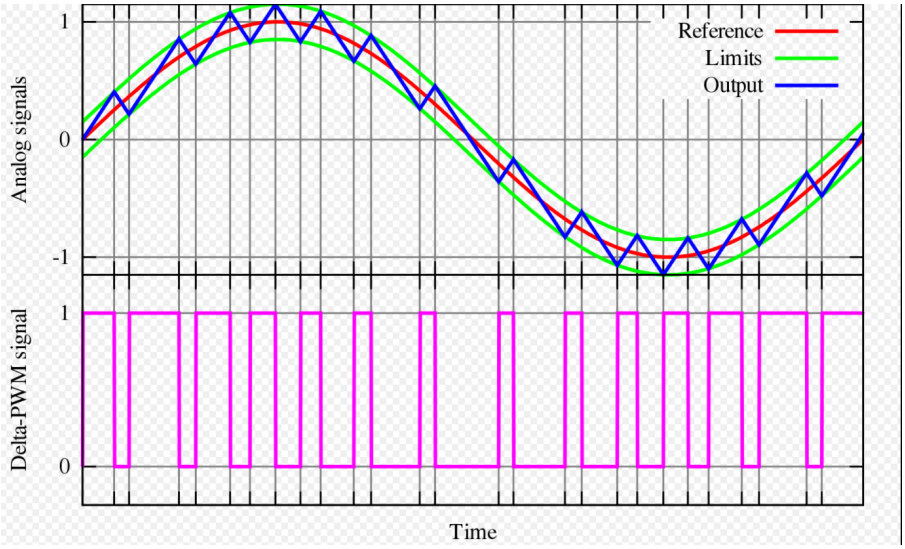

Figure 2.3 Delta Modulation

2.3.2.1 About Delta Modulator :

The Delta Modulator comprises of a **1-bit quantizer** and a **delay circuit** along with two summer circuits. The block diagram of Delta Modulator is as follows :

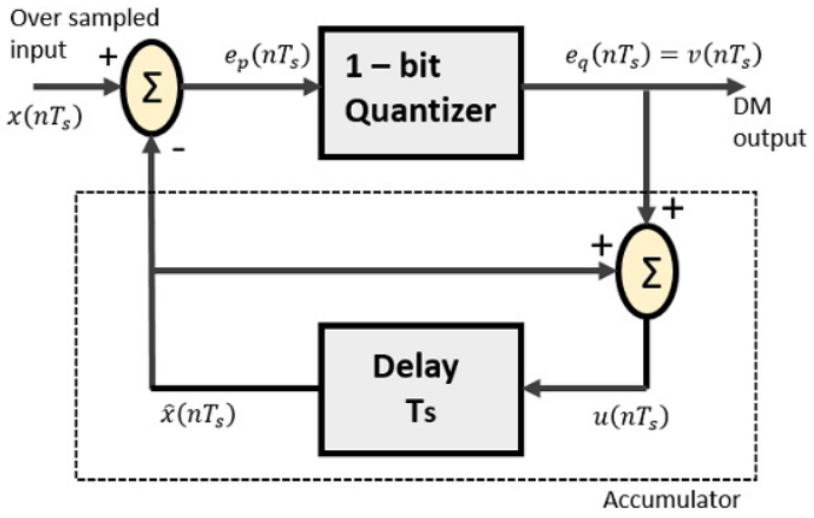

Figure 2.4 Block Diagram of Delta modulator

2.3.2.2 About Delta DeModulator :

The delta demodulator comprises of a **low pass filter**, a **summer**, and a **delay circuit**. The predictor circuit is eliminated here and hence no assumed input is given to the demodulator. The block diagram of Delta Demodulator is as follows :

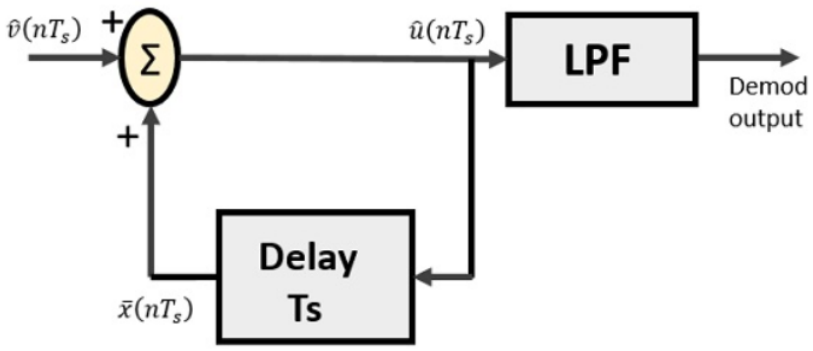

Figure 2.5 Block Diagram of Delta demodulator

2.3.2.3 Applications of Delta-modulation :

Contemporary applications of Delta Modulation includes, but is not limited to, recreating legacy synthesizer waveforms. With the increasing availability of **FPGAs** and **game-related ASICs**, sample rates are easily controlled so as to avoid slope overload and granularity issues. For example, the **C64DTV** used a 32 MHz sample rate, providing ample dynamic range to recreate the SID output to acceptable levels. Delta Modulation is most useful in systems where **timely data delivery** at the receiver is more important than the data quality. This modulation is applied to**ECG waveform** for **database reduction** and **real-time signal processing**. For **analog-to-PCM encoding**, Delta modulation is used.

2.3.2.4 Limitations of Delta modulation :

- Slope overload distortion
- Granular or idle noise
- High bit rate
- Poor start-up response
- It requires a predicator and hence it is very complex

2.4 Code and Results

2.4.1 PCM Modulation and Demodulation :

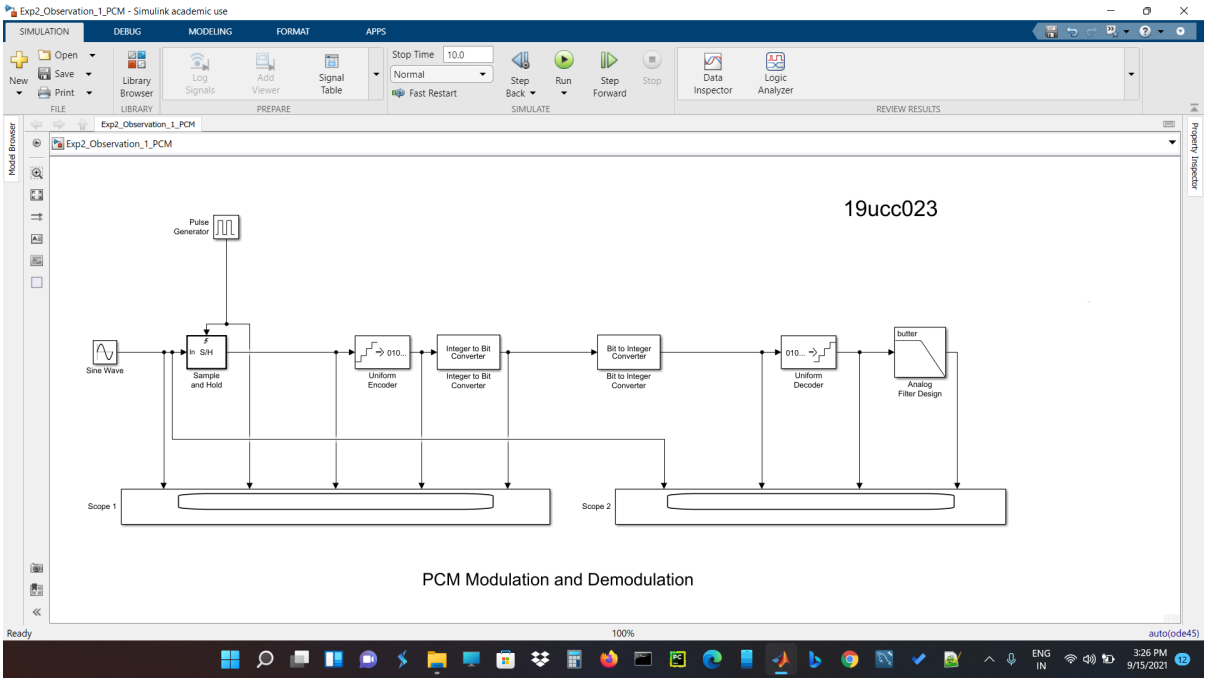

Figure 2.6 PCM Block Diagram

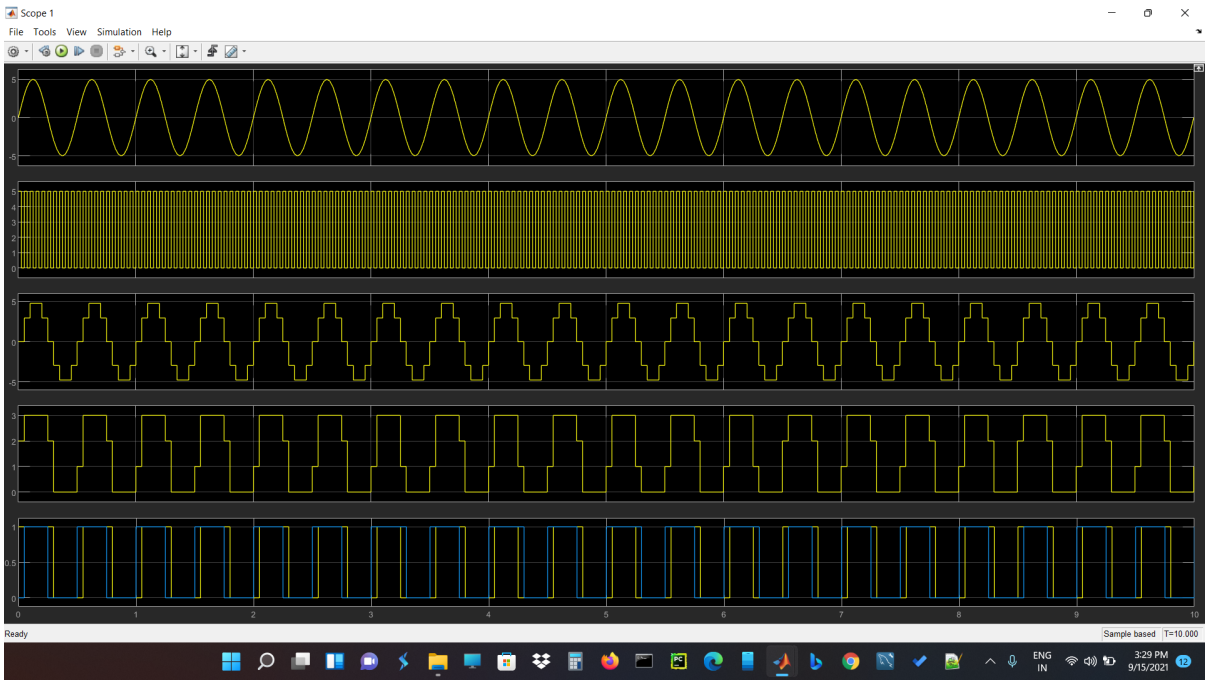

Figure 2.7 Plot of Scope 1 of PCM block diagram for **bits = 2**

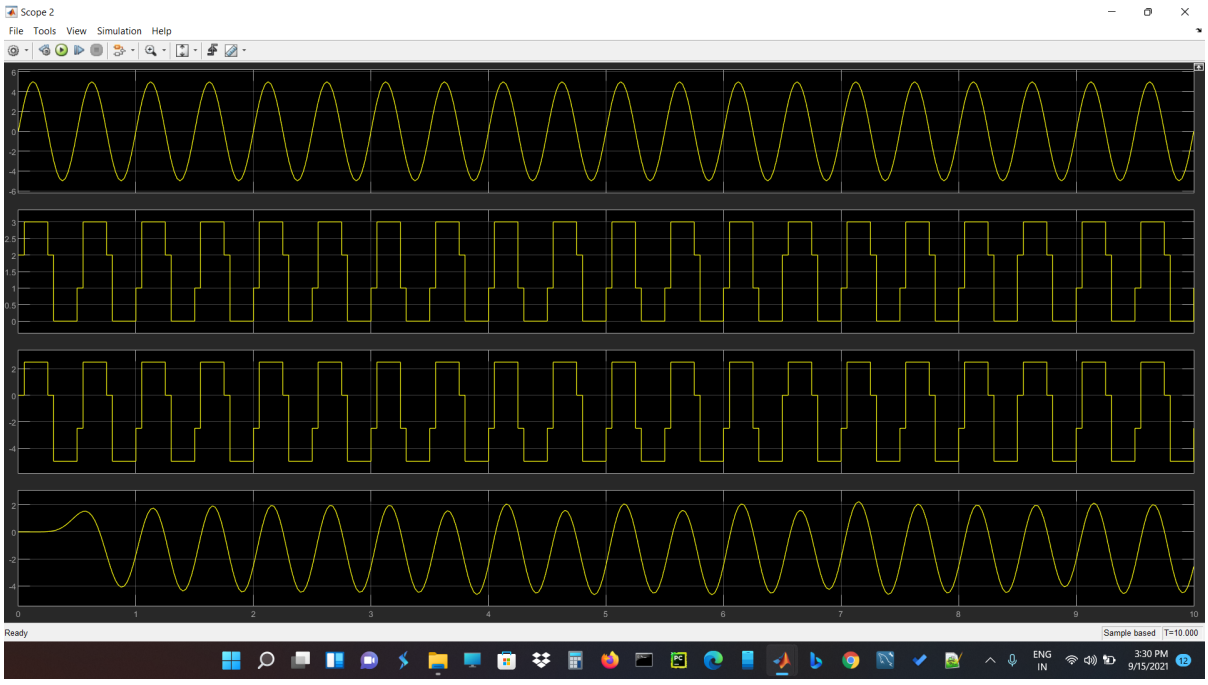

Figure 2.8 Plot of Scope 2 of PCM block diagram for **bits = 2**

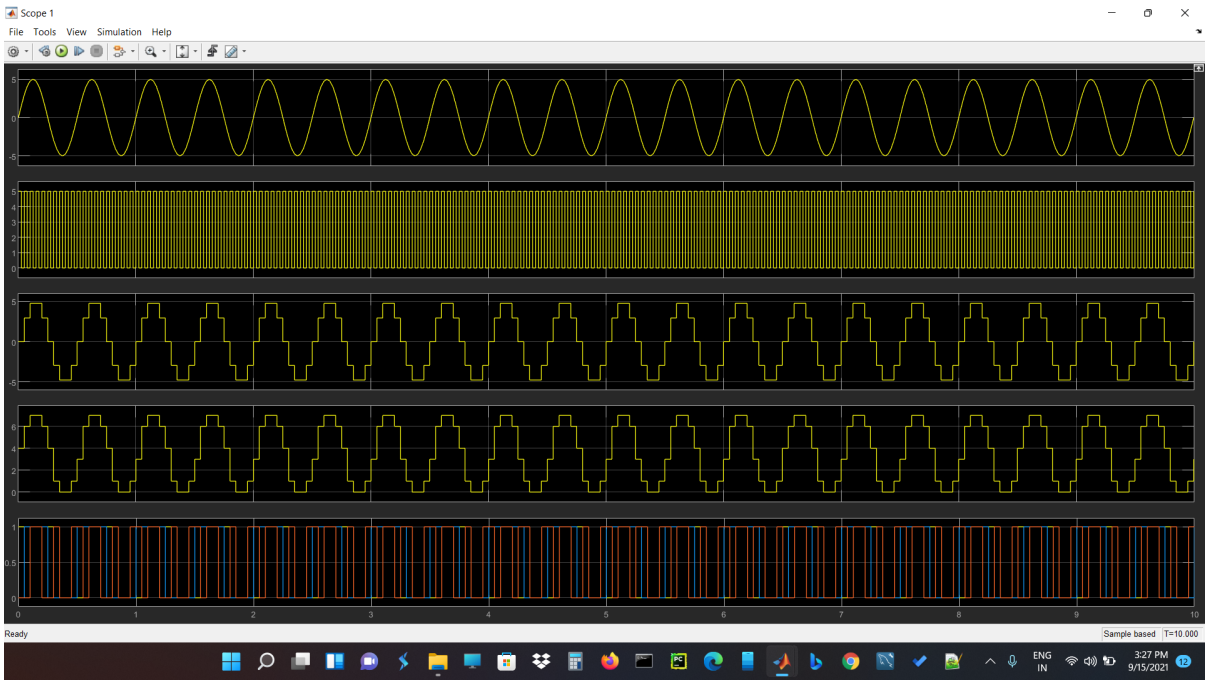

Figure 2.9 Plot of Scope 1 of PCM block diagram for **bits = 3**

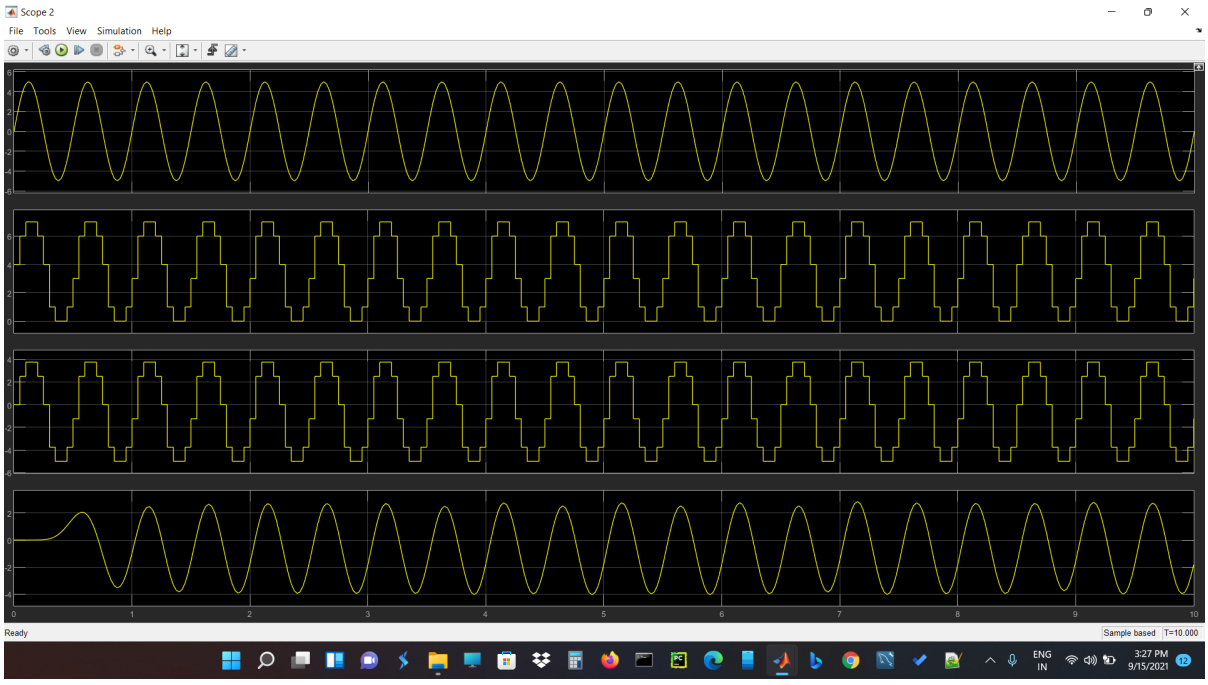

Figure 2.10 Plot of Scope 2 of PCM block diagram for **bits = 3**

2.4.2 Delta Modulation and Demodulation :

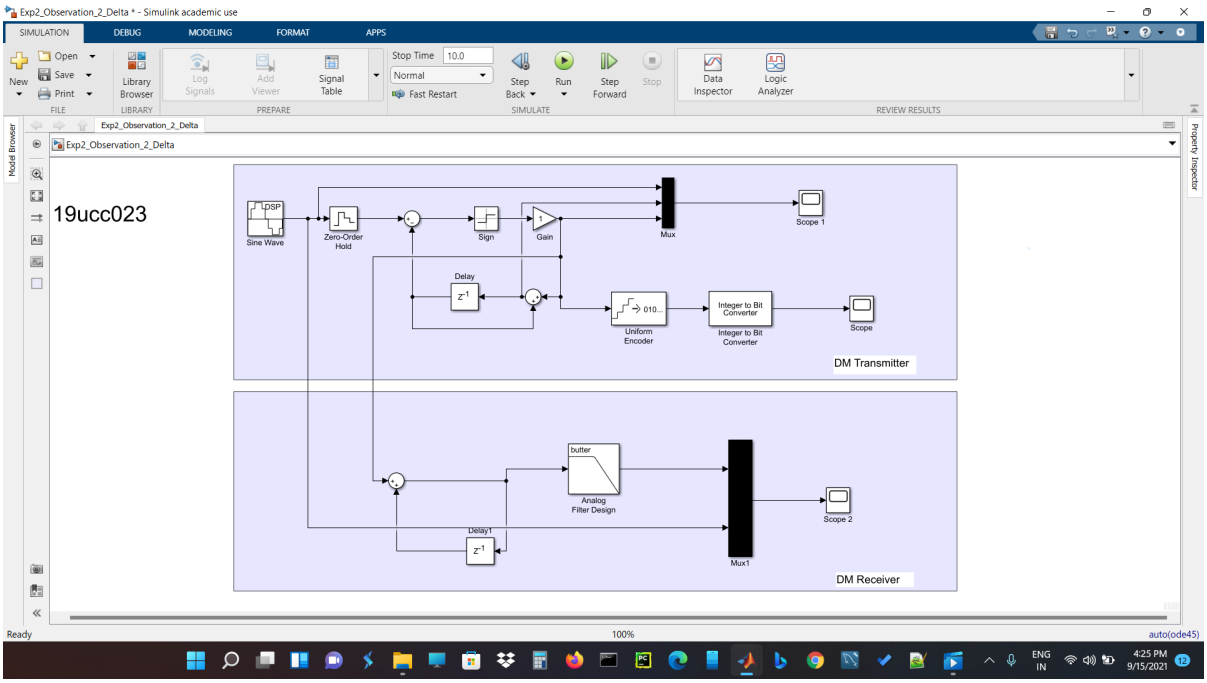

Figure 2.11 Delta Modulation Block Diagram

2.4.2.1 Plots for the Gain value = 1 :

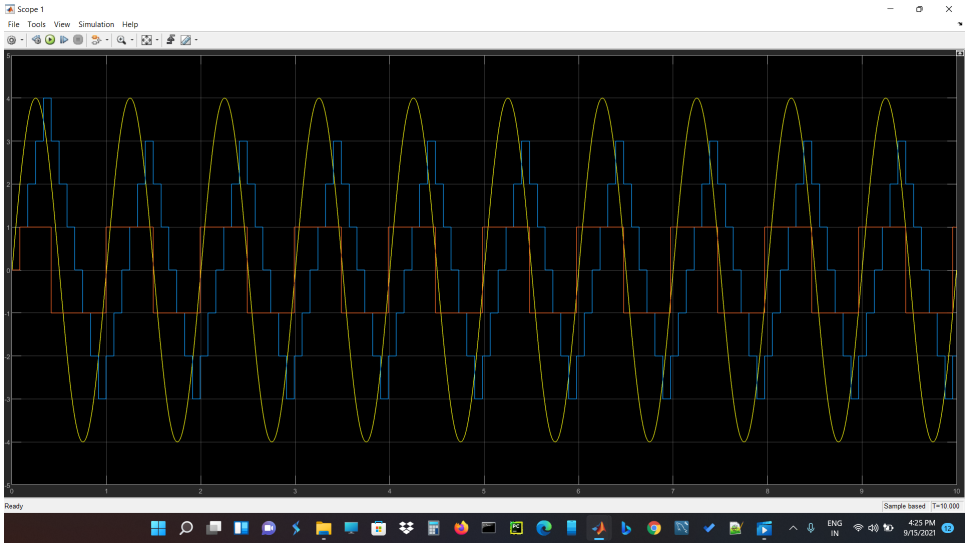

Figure 2.12 Plot of scope 1 of Delta Modulation Block diagram for **gain = 1**

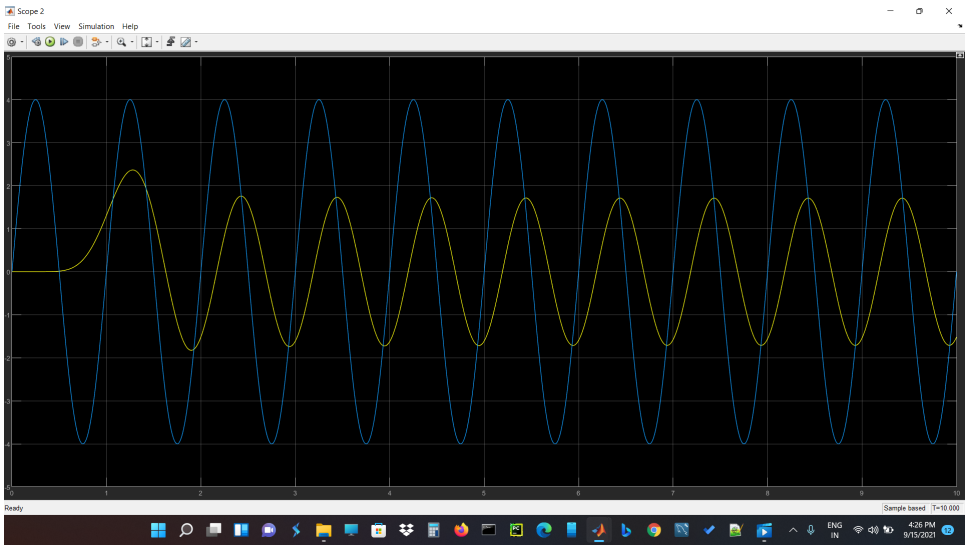

Figure 2.13 Plot of scope 2 of Delta Modulation Block diagram for **gain = 1**

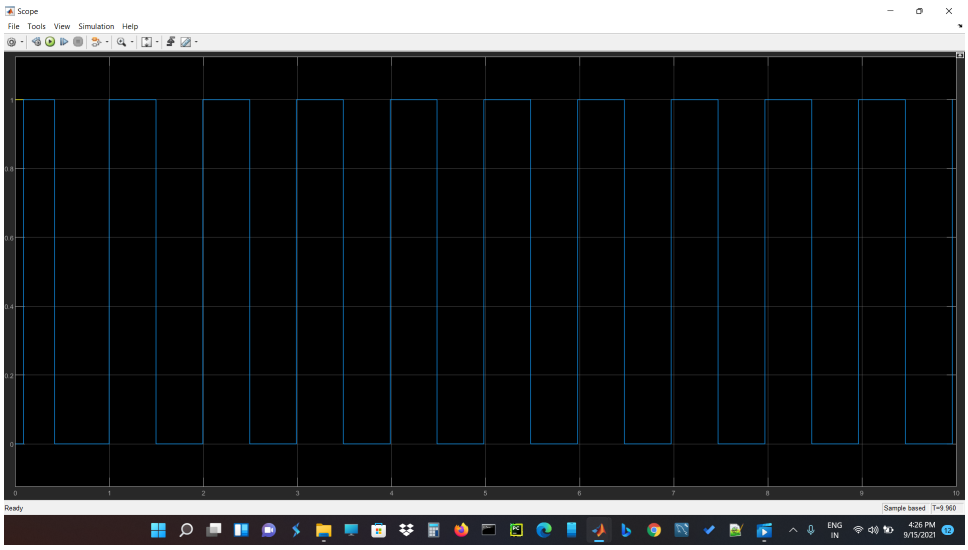

Figure 2.14 Plot of scope of Delta Modulation Block diagram for **gain = 1**

2.4.2.2 Plots for the Gain value = 10 :

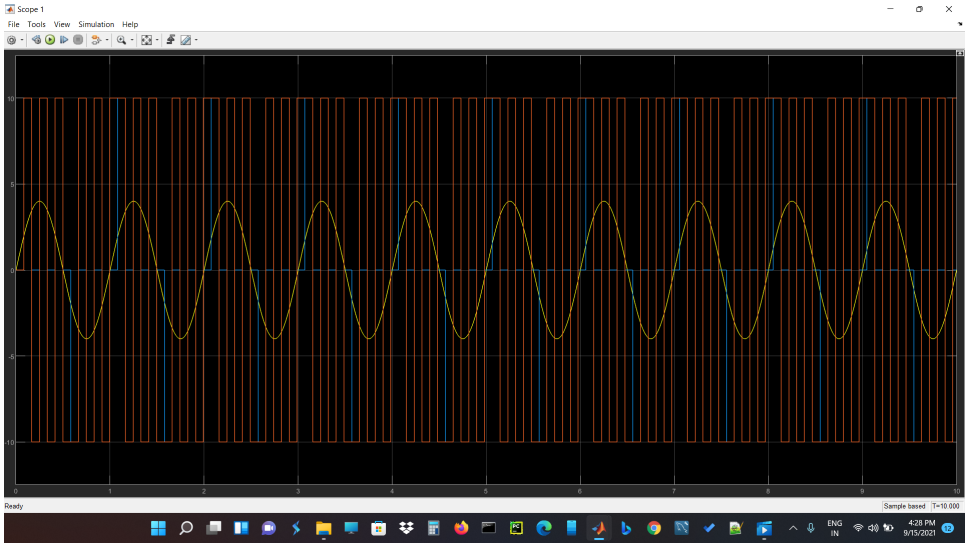

Figure 2.15 Plot of scope 1 of Delta Modulation Block diagram for **gain = 10**

Figure 2.16 Plot of scope 2 of Delta Modulation Block diagram for **gain = 10**

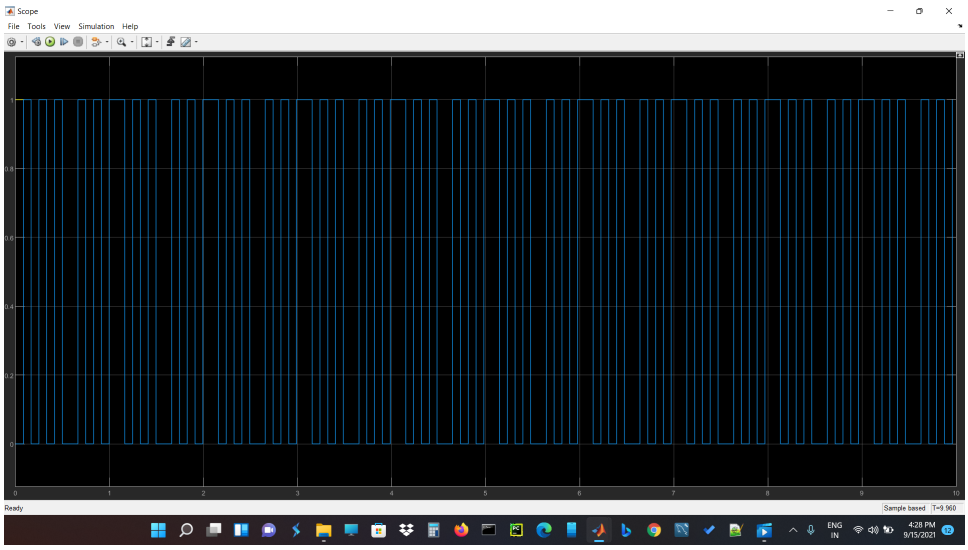

Figure 2.17 Plot of scope of Delta Modulation Block diagram for **gain = 10**

2.4.2.3 Plots for the Gain value = 0.2 :

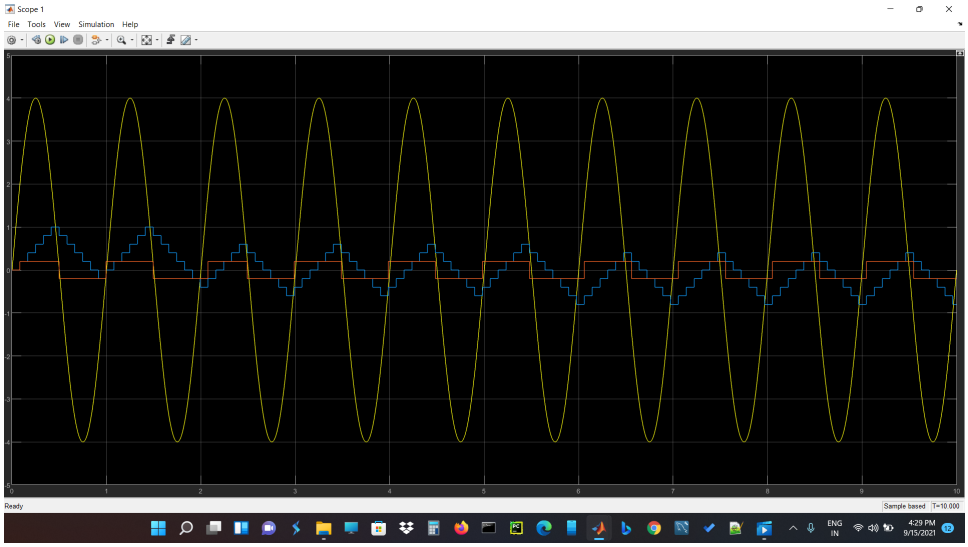

Figure 2.18 Plot of scope 1 of Delta Modulation Block diagram for **gain = 0.2**

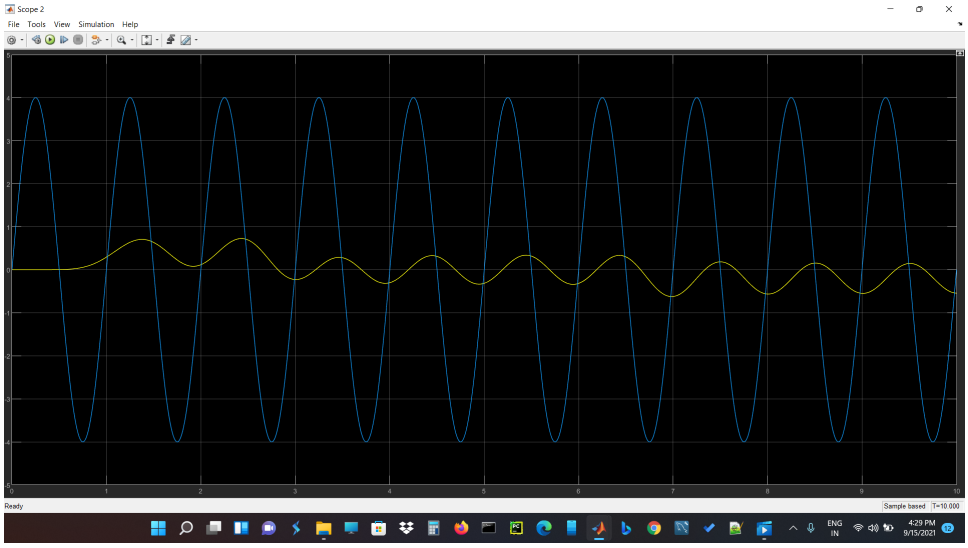

Figure 2.19 Plot of scope 2 of Delta Modulation Block diagram for **gain = 0.2**

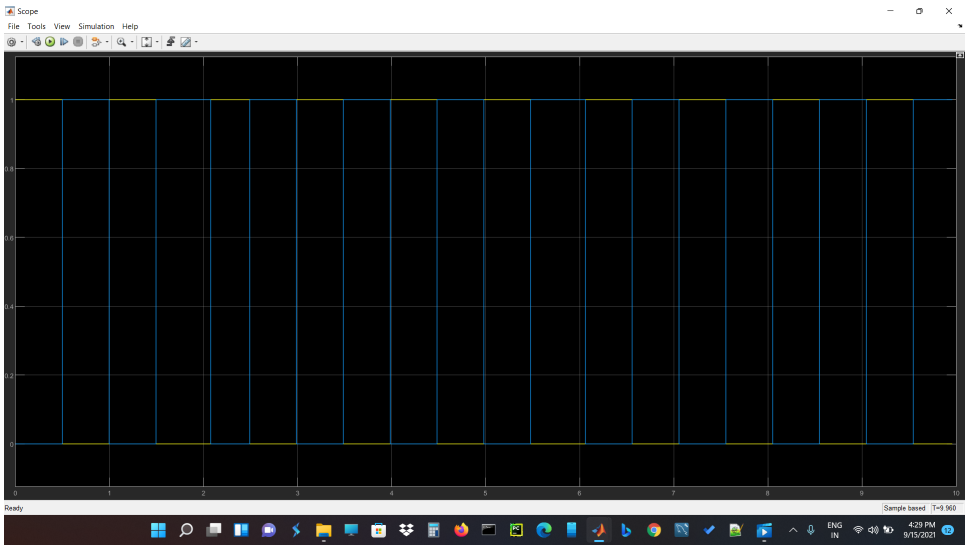

Figure 2.20 Plot of scope of Delta Modulation Block diagram for **gain = 0.2**

2.5 Conclusion

In this experiment, we have learnt about two types of modulation schemes - **PCM** and **Delta** Modulation and Demodulation. We also saw the effect of **number of bits** changing the Output of **PCM**. We also learnt about the effect of change of **gain** on the output of **Delta Modulator**. We also learnt how to implement these modulation schemes in Simulink platform.

Chapter *3*

Experiment - 3

3.1 Name of the Experiment

To implement BPSK Modulation and Demodulation (incorporating Matched Filter) on MATLAB Simulink Platform

3.2 Software Used

- MATLAB
- Simulink

3.3 Theory

3.3.1 About Phase Shift Keying :

Phase-shift keying (PSK) is a digital modulation process which conveys data by changing (modulating) the phase of a constant frequency reference signal (the carrier wave). The modulation is accomplished by varying the sine and cosine inputs at a precise time. It is widely used for wireless **LANs**, **RFID** and **Bluetooth communication**. **PSK** uses a **finite** number of **phases**, each assigned a unique pattern of binary digits. Usually, each phase encodes an equal number of bits. Each pattern of bits forms the symbol that is represented by the particular phase. The **demodulator**, which is designed specifically for the symbol-set used by the modulator, determines the phase of the received signal and maps it back to the symbol it represents, thus recovering the original data. This requires the receiver to be able to compare the phase of the received signal to a reference signal – such a system is termed **coherent**.

PSK is of **two types**:

- Binary Phase Shift Keying (BPSK)
- Quadrature Phase Shift Keying (QPSK)

3.3.2 About BPSK Modulation :

Binary Phase Shift Keying (BPSK) is a **two phase** modulation scheme, where the 0's and 1's in a binary message are represented by two different phase states in the carrier signal : θ = 0^o for binary 1 and θ = 180^o for binary 0. BPSK is the **simplest** form of phase shift keying (PSK). In BPSK, only **one sinusoid** is taken as the basis function. Modulation is achieved by varying the phase of the sinusoid depending on the message bits. Therefore, within a bit duration T_b, the two different phase states of the carrier signal are represented as :

$$S_1(t) = A_c cos(2\pi f_c t) \tag{3.1}$$

$$S_0(t) = A_c cos(2\pi f_c t + \pi) \tag{3.2}$$

In the above equations , $0 <= t <= T_b$, $S_1(t)$ is for **binary 1** and $S_0(t)$ is for **binary 0**, A_c is the **amplitude** of the sinusoidal signal , f_c is the **carrier frequency** , **t** is the **instantaneous frequency** and T_b is the **bit rate** in seconds.

The **constellation diagram** for BPSK will show **two** constellation points, lying entirely on the x axis (in-phase). It has **no projection on the y axis** (quadrature). This means that the BPSK modulated signal will have an **in-phase component** but no quadrature component. This is because it has only **one basis function**. It can be noted that the carrier phases are 180◦ apart and it has constant envelope. The carrier's phase contains all the information that is being transmitted.

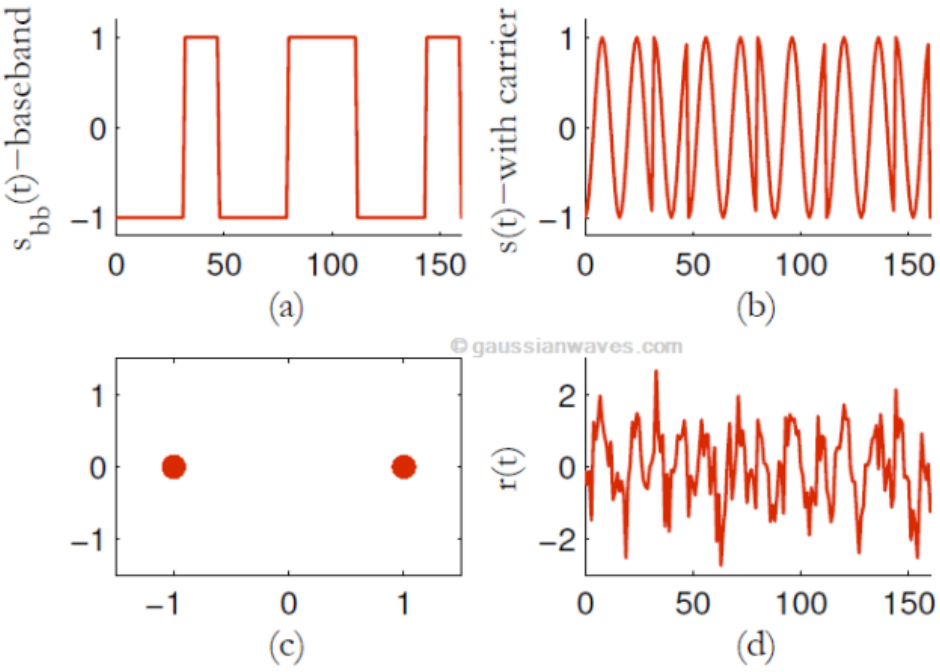

Figure 3.1 (a) Baseband BPSK signal , (b) transmitted BPSK signal - with carrier , (c) constellation at transmitter , (d) received signal with AWGN noise

3.3.2.1 About BPSK transmitter :

A **BPSK transmitter** is implemented by coding the message bits using **NRZ coding** (1 represented by positive voltage and 0 represented by negative voltage) and multiplying the output by a reference oscillator running at carrier frequency f_c. The MATLAB function **bpsk_mod** implements a baseband BPSK transmitter. The output of the function is in baseband and it can optionally be multiplied with the carrier frequency outside the function.

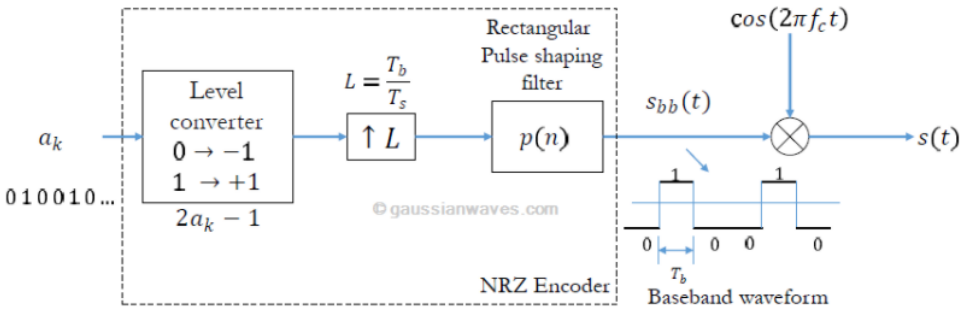

Figure 3.2 BPSK modulation block diagram

3.3.2.2 About BPSK receiver :

A **correlation type coherent detector** is used for **receiver** implementation. In coherent detection technique, the knowledge of the carrier frequency and phase must be known to the receiver. This can be achieved by using a **Costas loop** or a **Phase Lock Loop (PLL)** at the receiver. In the coherent receiver, the received signal is multiplied by a reference frequency signal from the carrier recovery blocks like PLL or Costas loop. The MATLAB function **bpsk_demod**, implements a baseband BPSK receiver.

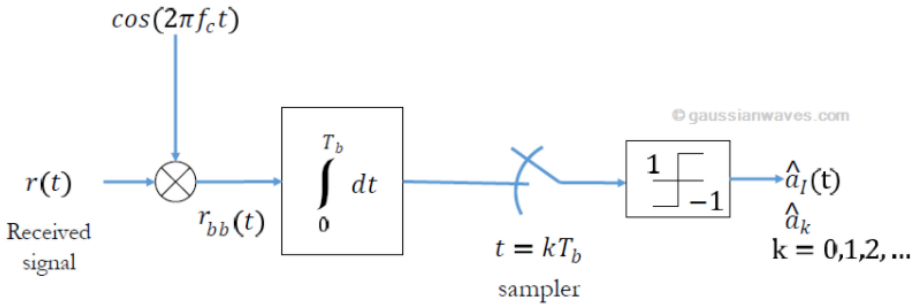

Figure 3.3 Coherent Detector used for BPSK Demodulation

3.4 Code and Results

3.4.1 BPSK Modulation and Demodulation :

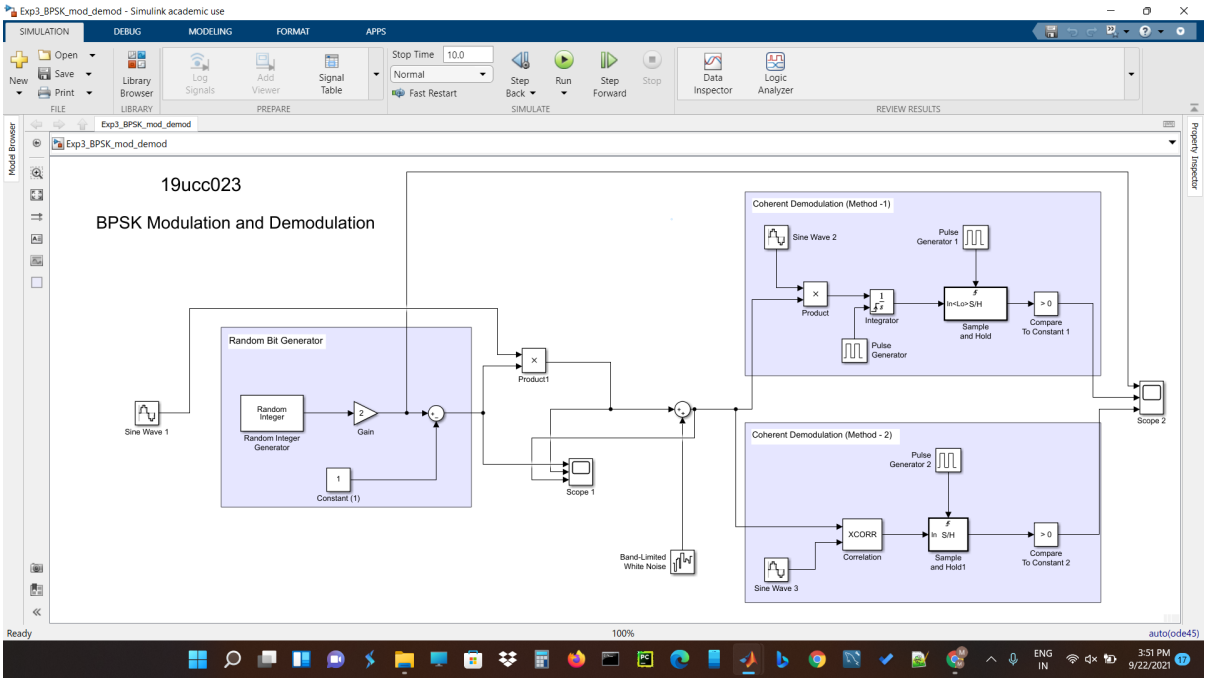

Figure 3.4 BPSK Modulation and Demodulation Simulink Block Diagram

3.4.1.1 Plots of Scope 1 of BPSK Block Diagram :

1. 2x-1 Waveform of Input Signal
2. BPSK Modulated waveform
3. BPSK Modulated waveform + Band Limited White Noise

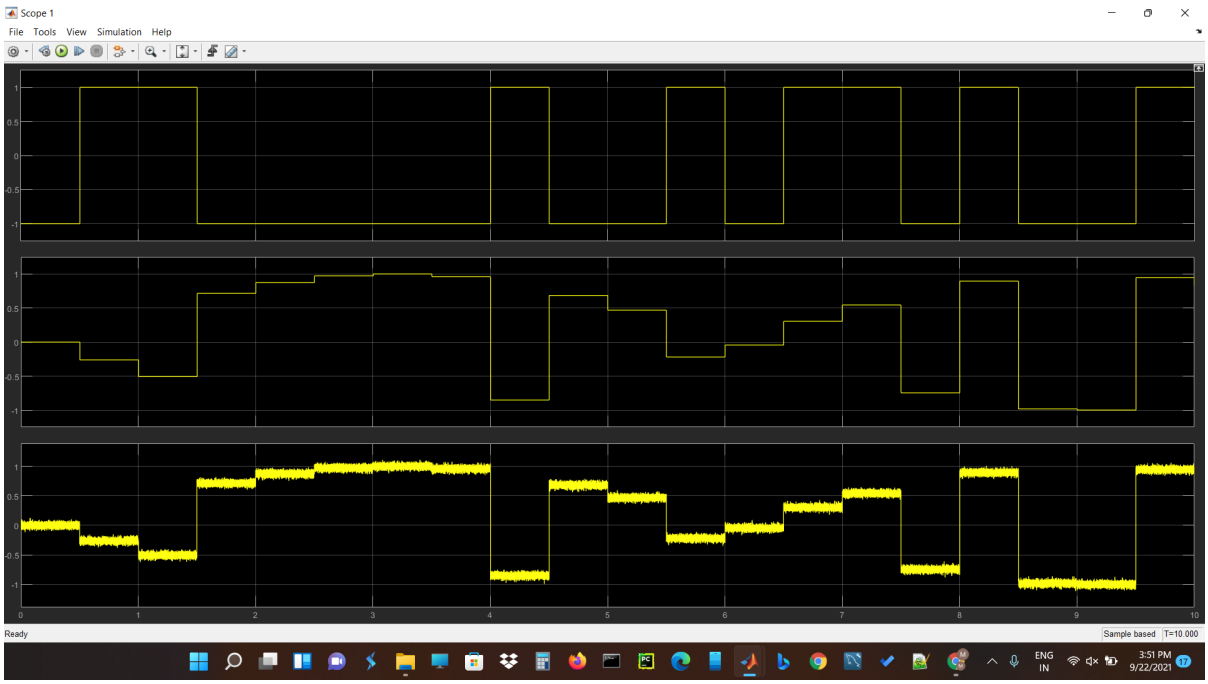

Figure 3.5 Plots of Scope 1 of BPSK Block Diagram

3.4.1.2 Plots of Scope 2 of BPSK Block Diagram :

1. Input waveform
2. BPSK Demodulated waveform from METHOD - 1
3. BPSK Demodulated waveform from METHOD - 2

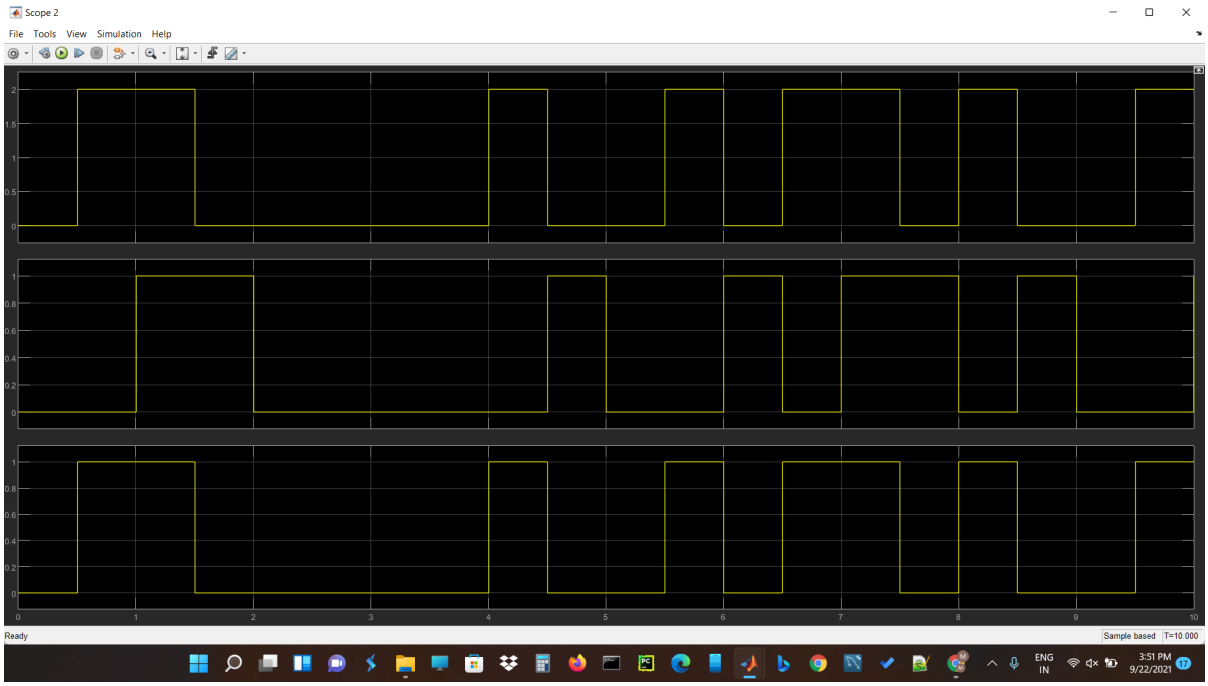

Figure 3.6 Plots of Scope 2 of BPSK Block Diagram

3.5 Conclusion

In this experiment, we learnt about **BPSK** modulation and demodulation. We learnt about their **Constellation Diagrams** and the structures of BPSK **transmitter** and **receiver**. We also implemented the BPSK Modulation-Demodulation in Simulink platform and observed the results. We implemented the Demodulation through 2 methods - in one we directly apply the Correlation block and in second , we implemented step-by-step procedure to calculate correlation. We learnt about **Band Limited White Noise** also. We learnt about new blocks in Simulink platform such as **Integrator** and **Correlation** blocks.

Chapter *4*

Experiment - 4

4.1 Name of the Experiment

To implement QPSK Modulation and Demodulation on MATLAB Simulink platform

4.2 Software Used

- MATLAB
- Simulink

4.3 Theory

4.3.0.1 About QPSK Modulation :

Quadrature Phase Shift Keying (QPSK) is a form of phase modulation technique, in which two information bits (combined as one symbol) are modulated at once, selecting one of the four possible carrier phase shift states. The mathematical analysis shows that QPSK can be used either to double the data rate compared with a BPSK system while maintaining the **same bandwidth** of the signal, or to maintain the data-rate of BPSK but halving the bandwidth needed. In this latter case, the BER of QPSK is exactly the same as the BER of BPSK – and believing differently is a common confusion when considering or describing QPSK. The transmitted carrier can undergo numbers of **phase changes**. The QPSK signal within a symbol duration T_{sym} is defined as :

$$S(t) = Acos(2\pi f_c t + \theta_n) \tag{4.1}$$

In the above equation , $0 <= t <= T_{sym}$. The **signal phase** is given by :

$$\theta_n = (2n-1)\pi/4 \tag{4.2}$$

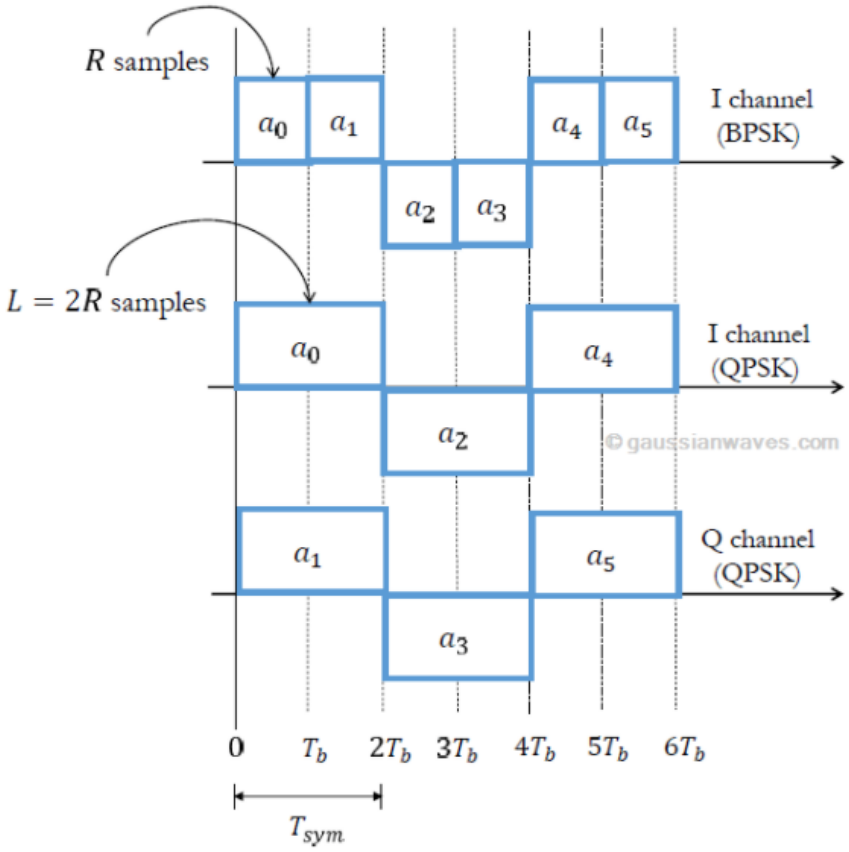

Figure 4.1 comparison of Timing Diagram of QPSK and BPSK modulations

From the timing diagram for BPSK and QPSK modulation , we can observe that for BPSK modulation the symbol duration for each bit is same as bit duration, but for QPSK the symbol duration is **twice** the **bit duration** : $T_{sym} = 2T_b$. Therefore, if the QPSK symbols were transmitted at same rate as BPSK, it is clear that QPSK sends twice as much data as BPSK does.

4.3.0.2 About QPSK transmitter :

In the QPSK transmitter, a splitter separates the odd and even bits from the generated information bits. Each stream of odd bits (quadrature arm) and even bits (in-phase arm) are converted to NRZ format in a parallel manner. The binary data stream is split into the **in-phase** and **quadrature-phase** components. These are then separately modulated onto two **orthogonal basis** functions. In this implementation, two sinusoids are used. Afterwards, the two signals are **superimposed**, and the resulting signal is the QPSK signal. Polar non-return-to-zero encoding is used here. QPSK Modulation is performed by the MATLAB function **qpsk_mod**.

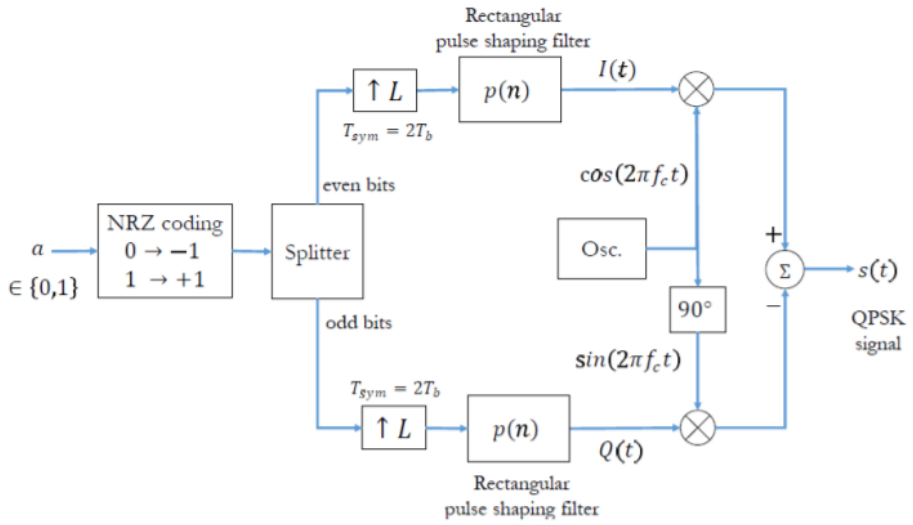

Figure 4.2 Transmitter section Block Diagram of QPSK Modulation

4.3.0.3 About QPSK receiver :

Due to its special relationship with BPSK, the QPSK receiver takes the simplest form. In this implementation, the I-channel and Q-channel signals are individually demodulated in the same way as that of BPSK demodulation. After demodulation, the I-channel bits and Q-channel sequences are combined into a single sequence. QPSK Demodulation is performed by the MATLAB function **qpsk_demod**.

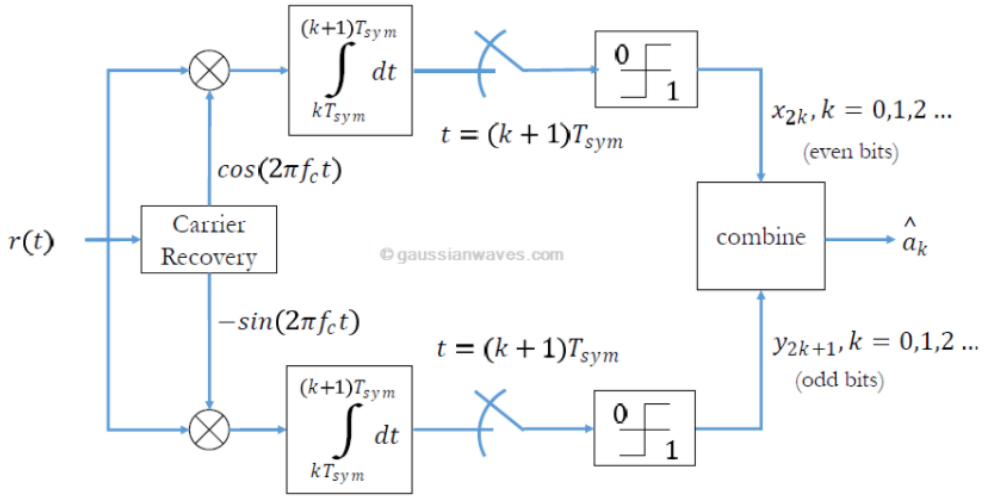

Figure 4.3 Reciever section Block Diagram of QPSK Modulation

4.3.0.4 Performance Analysis of QPSK over AWGN :

The simulation involves, generating **random message bits**, modulating them using QPSK modulation, addition of **AWGN channel noise** corresponding to the given signal-to-noise ratio and demodulating the noisy signal using a coherent QPSK receiver. The waveforms at the various stages of the modulator are shown in the below figure.

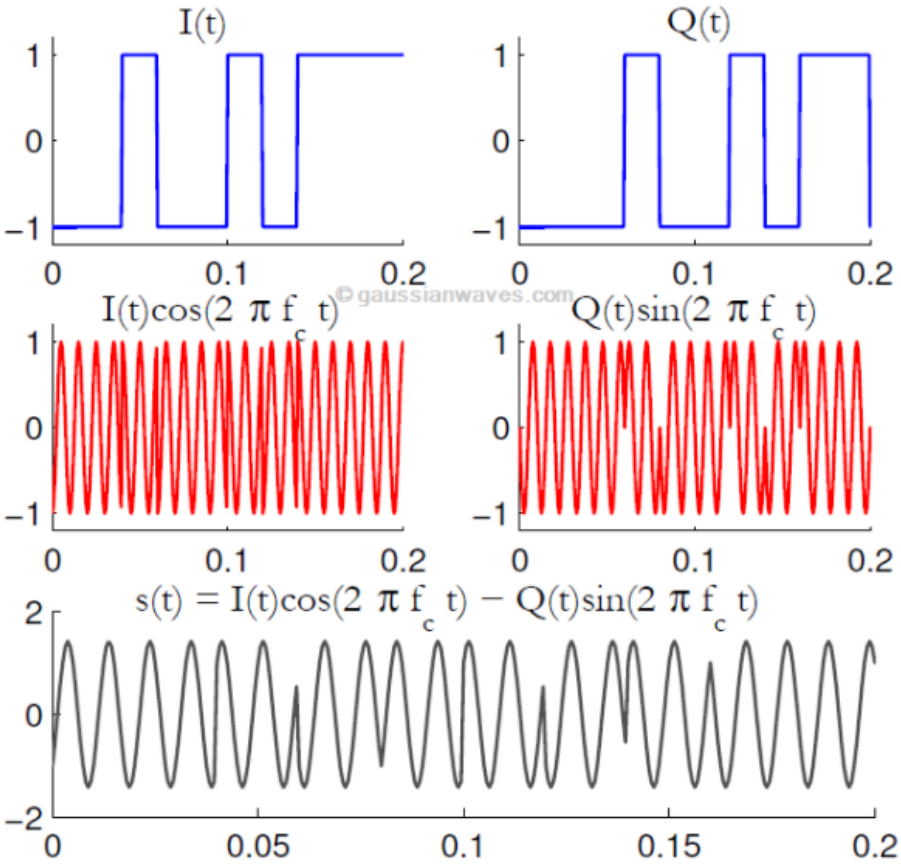

Figure 4.4 Perfomance analysis of QPSK over AWGN channel

4.3.0.5 Variations of QPSK :

There are three types of variation of QPSK which are as follows :

- Offset QPSK
- $\pi/4$-QPSK
- $\pi/4$-DQPSK.

4.4 Code and Results

4.4.1 QPSK Modulation and Demodulation :

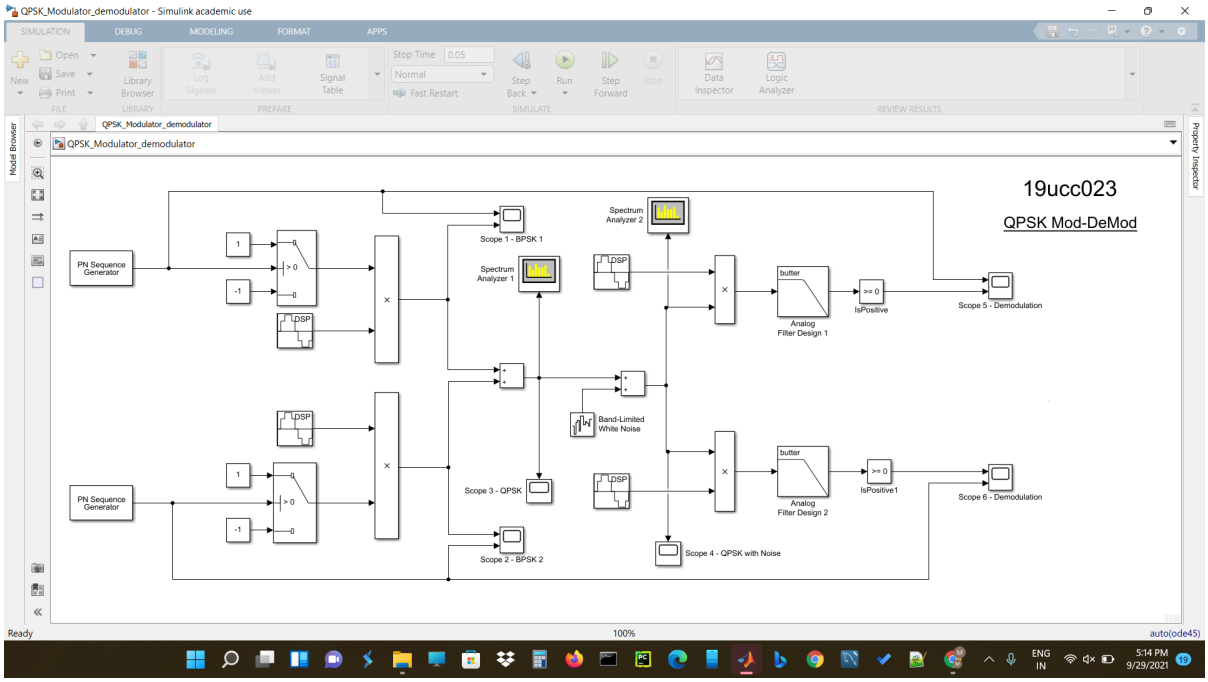

Figure 4.5 QPSK modulation demodulation Simulink Block Diagram

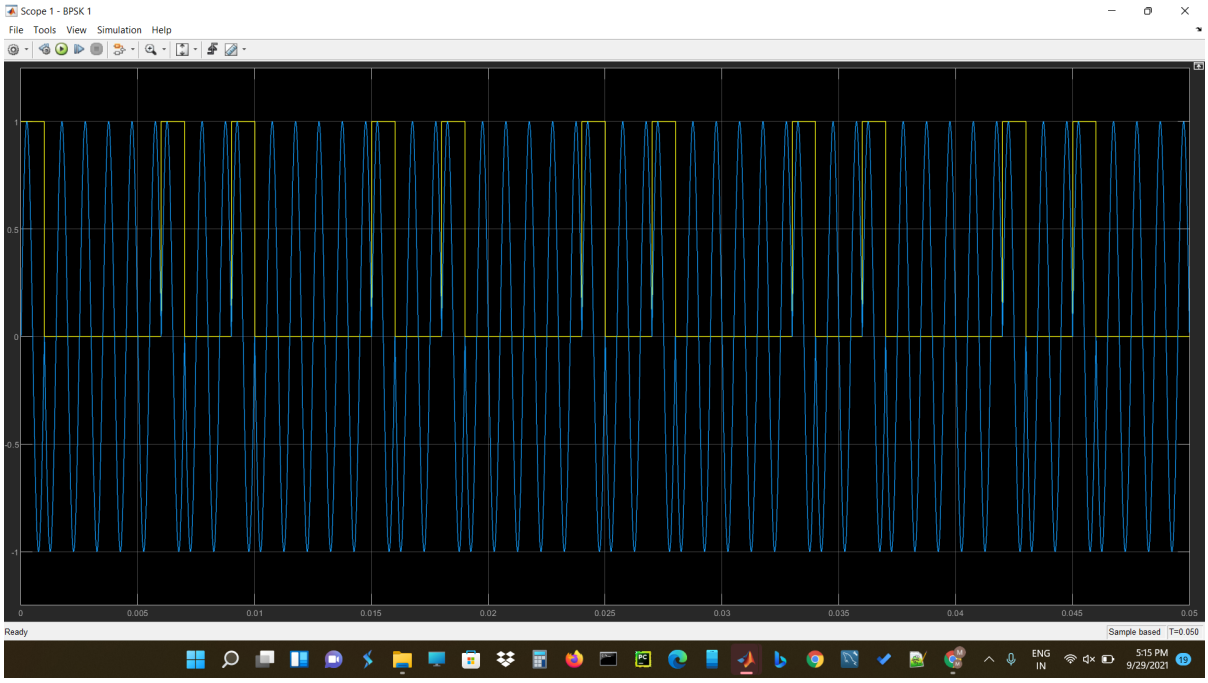

Figure 4.6 Plot of Scope 1 (BPSK 1)

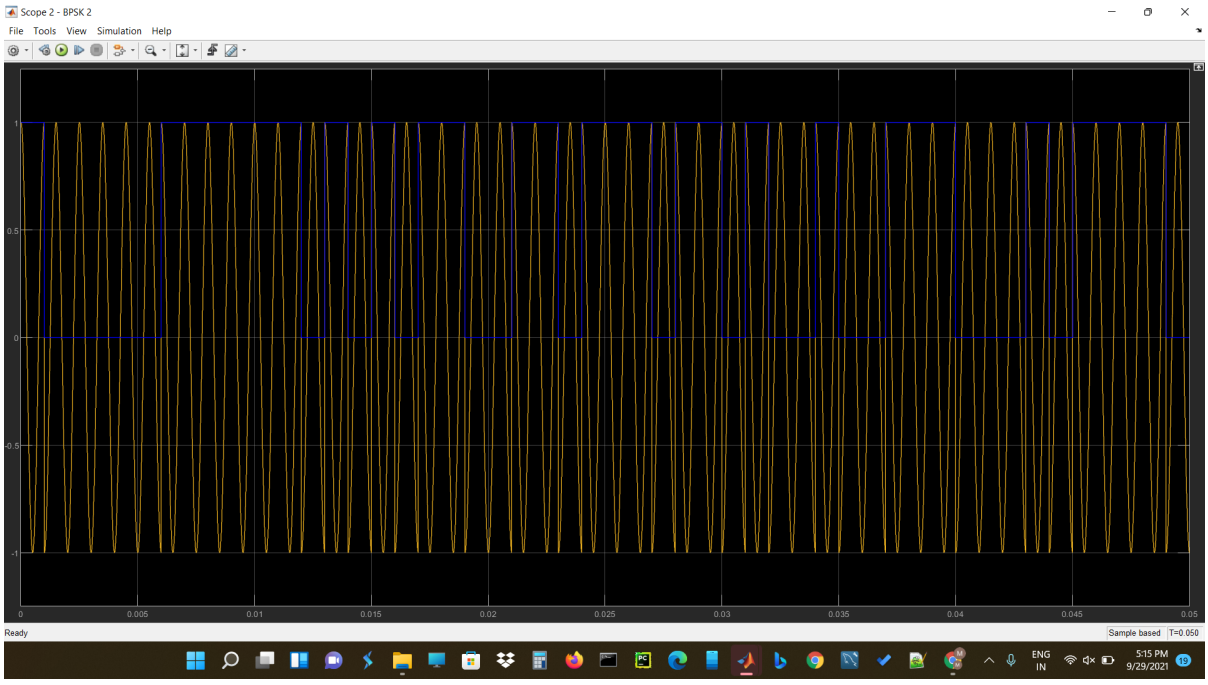

Figure 4.7 Plot of Scope 2 (BPSK 2)

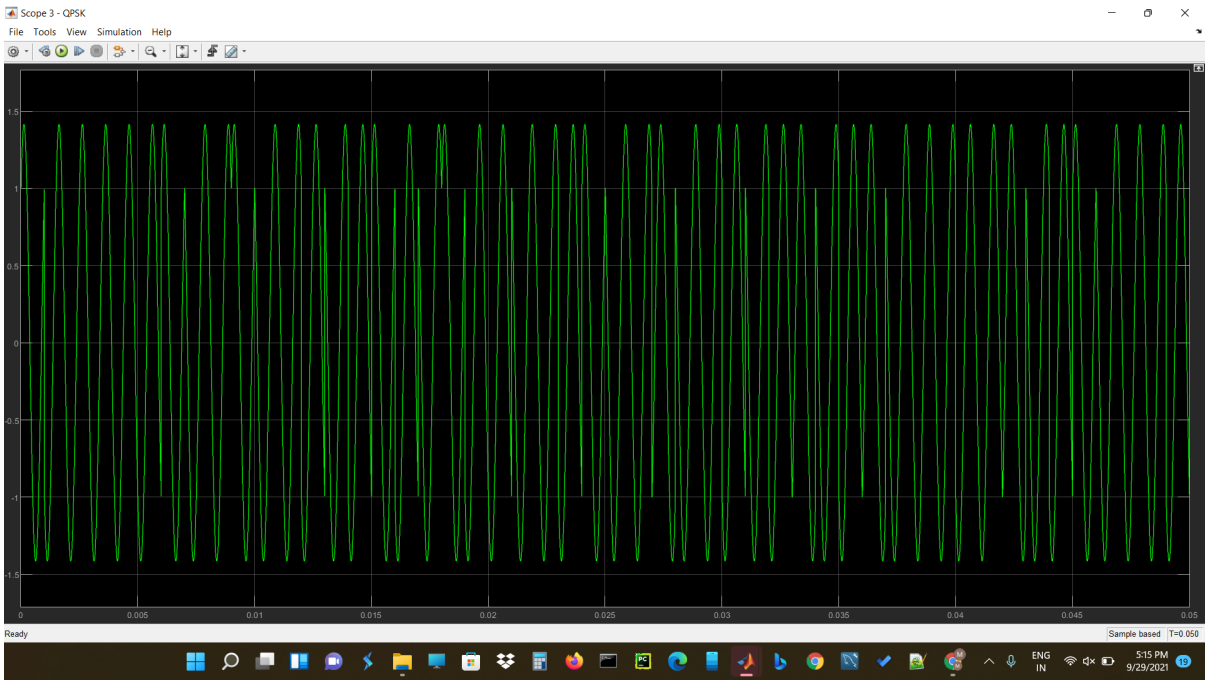

Figure 4.8 Plot of Scope 3 (QPSK waveform)

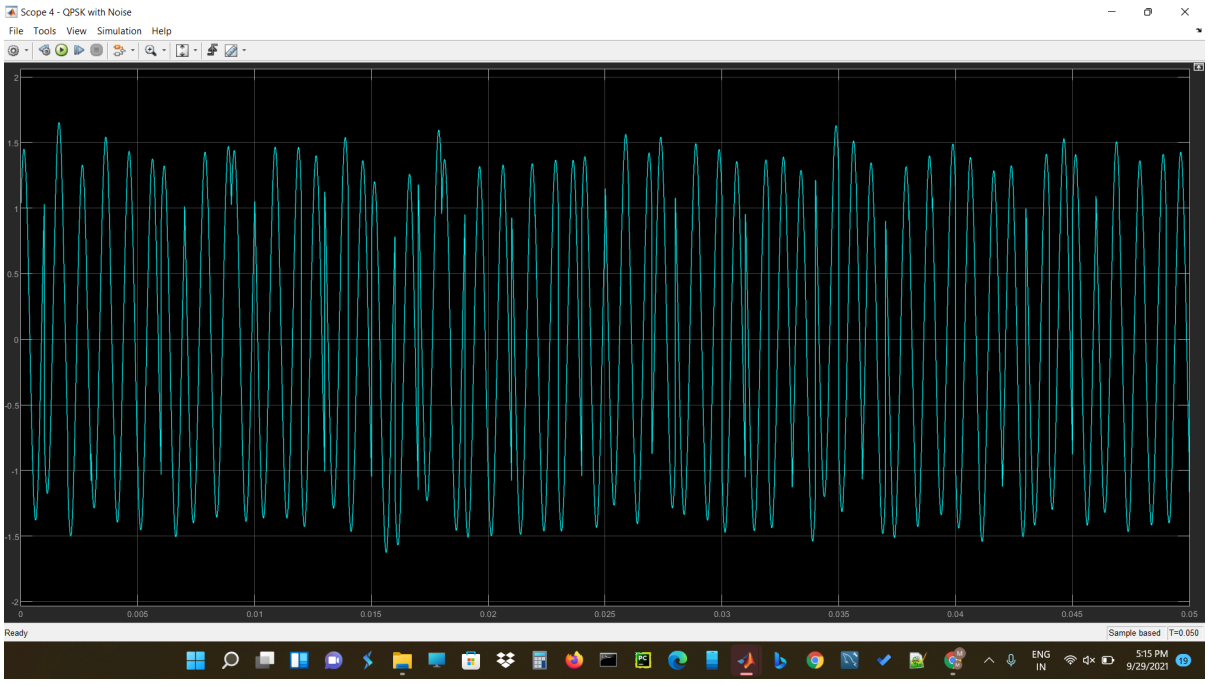

Figure 4.9 Plot of Scope 4 (QPSK waveform with Noise)

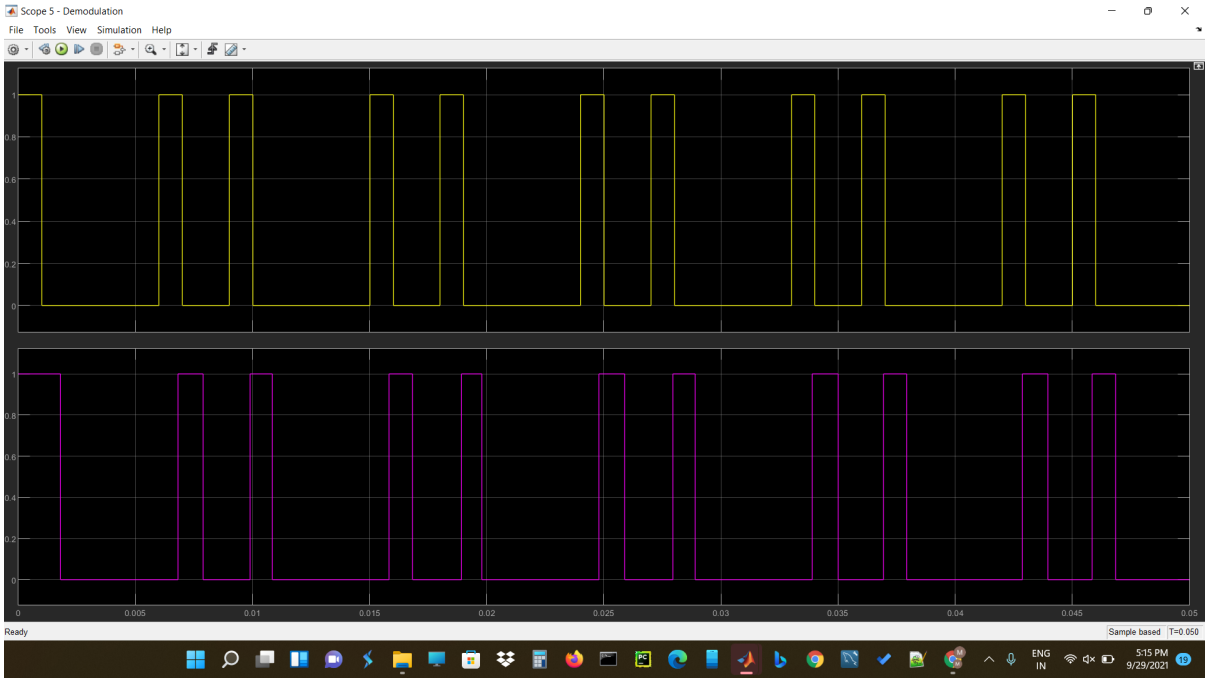

Figure 4.10 Plot of Scope 5 (demodulation)

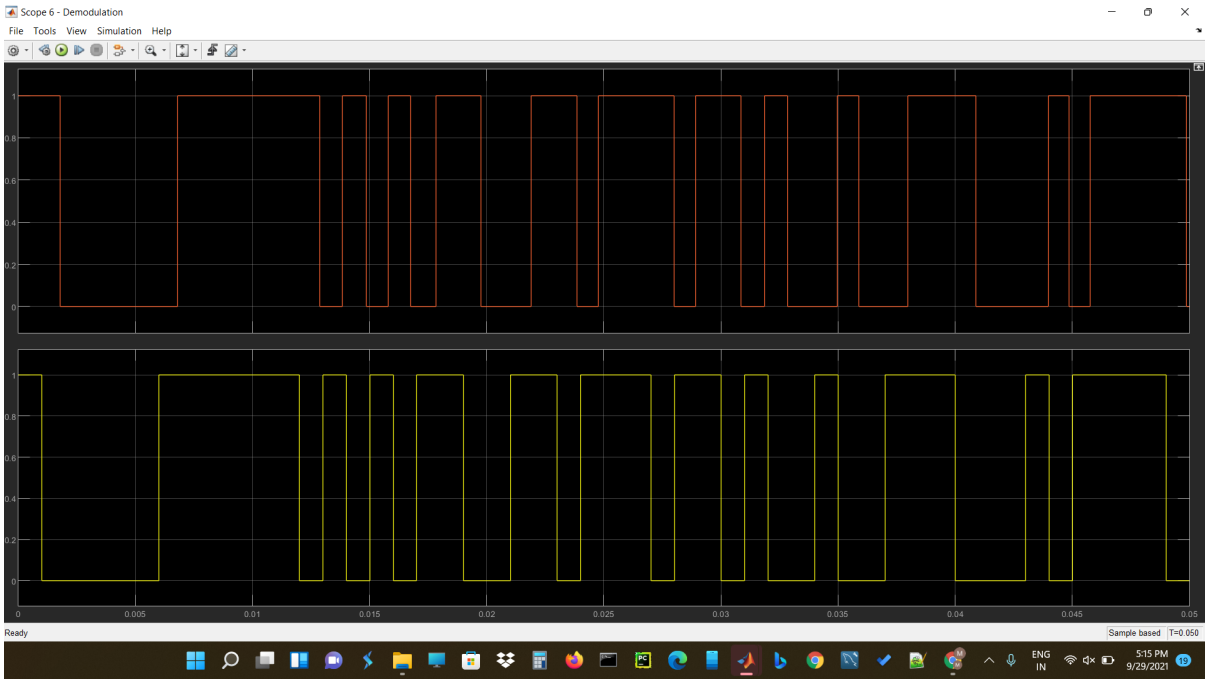

Figure 4.11 Plot of Scope 6 (demodulation)

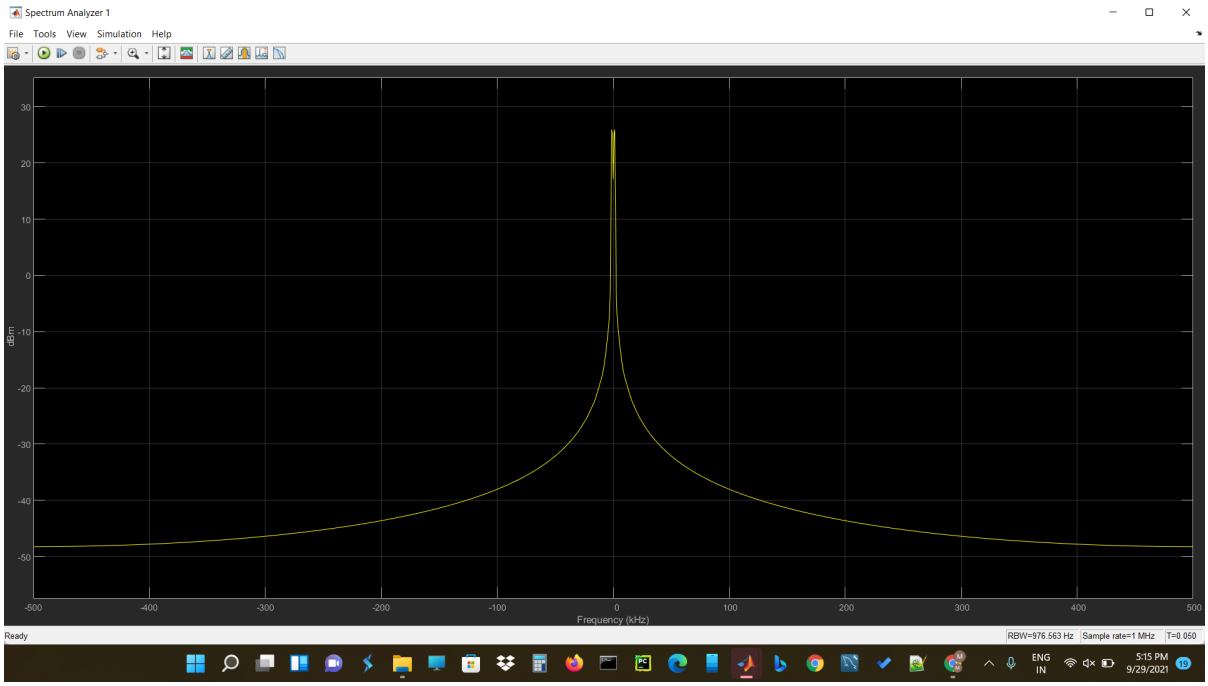

Figure 4.12 Plot of Spectrum Analyser 1

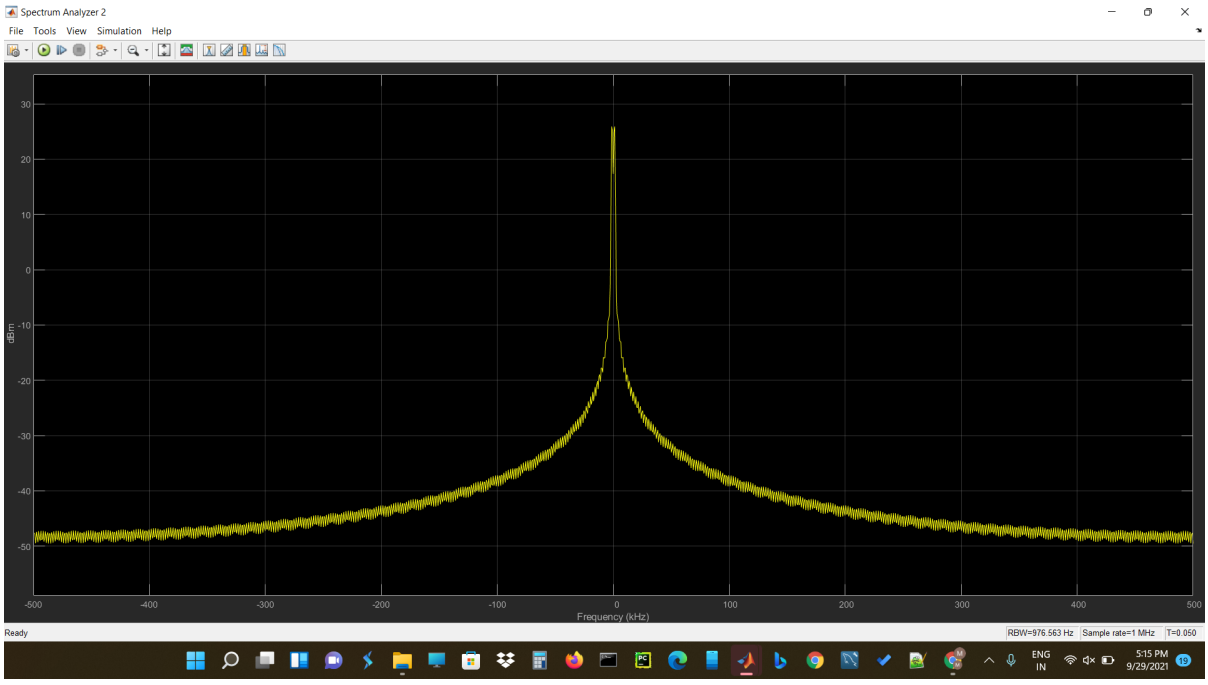

Figure 4.13 Plot of Spectrum Analyser 2

4.5 Conclusion

In this experiment, we learnt about **QPSK** modulation and demodulation. We learnt about their **Constellation Diagrams** and the structures of QPSK **transmitter** and **receiver**. We also implemented the QPSK Modulation-Demodulation in Simulink platform and observed the results. We learnt about new blocks in Simulink platform such as **PN-Sequence generator** , **Spectrum Analyser** and **switch** blocks.

Chapter *5*

Experiment - 5

5.1 Name of the Experiment

Performance analysis of binary modulation schemes (BPSK / OOK) over AWGN channel

5.2 Software Used

- MATLAB
- Simulink

5.3 Theory

5.3.1 About AWGN channel :

Additive white Gaussian noise (AWGN) is a basic noise model used in information theory to mimic the effect of many random processes that occur in nature. The modifiers denote specific characteristics:

- Additive : it is added to any noise that might be intrinsic to the information system.
- White : refers to the idea that it has **uniform power** across the frequency band for the information system. It is an analogy to the color white which has **uniform emissions** at all frequencies in the **visible spectrum**.
- Gaussian : because it has a normal distribution in the time domain with an average time domain value of zero and variance σ^2.

Wideband noise comes from many natural noise sources, such as the thermal vibrations of atoms in conductors (referred to as thermal noise or Johnson–Nyquist noise), shot noise, black-body radiation from the earth and other warm objects, and from celestial sources such as the Sun. The **central limit theorem** of probability theory indicates that the summation of many random processes will tend to have distribution called **Gaussian** or **Normal**.

5.3.1.1 About Gaussian Random Variable:

Consider the Gaussian random variable of mean μ and variance σ^2. This is represented using the notation N(μ,σ^2). The probability density function $f_X(x)$ is given as :

$$f_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.1}$$

The **Standard Gaussian Random Variable** is a random variable with **mean** equal to **zero** and **variance** equal to **unity**. The standard Gaussian random variable is represented by N(0,1) and its probability density function $f_X(x)$ is given as :

$$f_X(x) = \frac{1}{\sqrt{(2\pi)}} e^{-\frac{x^2}{2}} \tag{5.2}$$

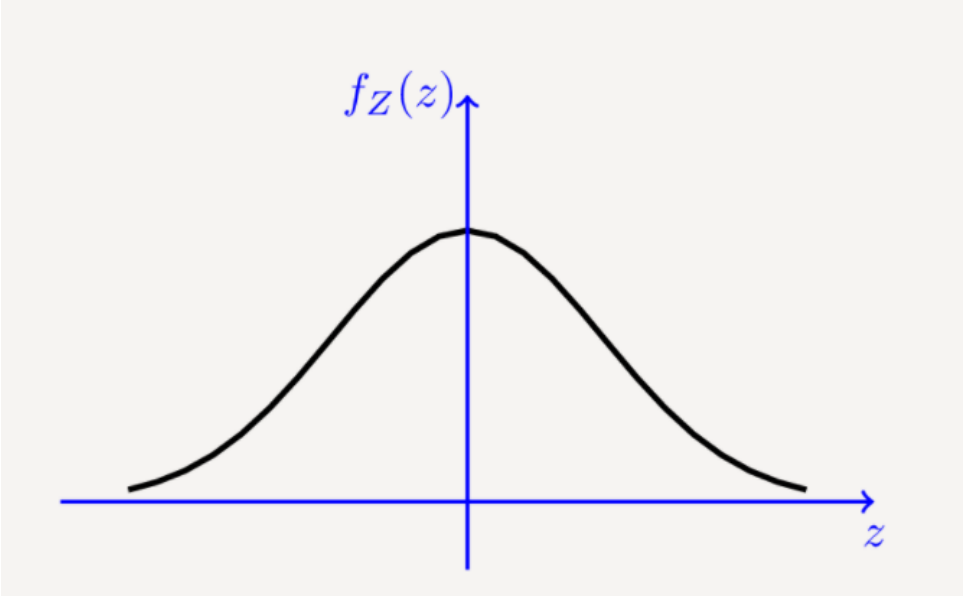

Figure 5.1 PDF of Gaussian Random Variable

Normal distributions are important in **statistics** and are often used in the **natural** and **social sciences** to represent real-valued random variables whose distributions are not known.Their importance is partly due to the **central limit theorem**. It states that, under some conditions, the average of many samples (observations) of a random variable with finite mean and variance is itself a random variable—whose distribution converges to a normal distribution as the number of samples increases. Therefore, physical quantities that are expected to be the sum of many **independent processes**, such as measurement errors, often have distributions that are nearly normal.

5.3.2 About Bit error rate of BPSK Modulation :

This is also called as **2-phase PSK** or **Phase Reversal Keying**. In this technique, the sine wave carrier takes two phase reversals such as 0° and 180°. The Pulse Energy is given by E_b and the transmitted symbols are given as $s_1(t)$ = $\sqrt{E_b}$ and $s_2(t)$ = -$\sqrt{E_b}$ for the information symbols 1 and 0 respectively. The **bit error** would refer to decoding a transmitted $\sqrt{E_b}$ (corresponding to bit 1) erroneously as the 0 bit and vice-versa. The corruption of the detected information symbol stream arises centrally due to presence of the white Gaussian noise at the receiver. Such a channel is called AWGN channel.

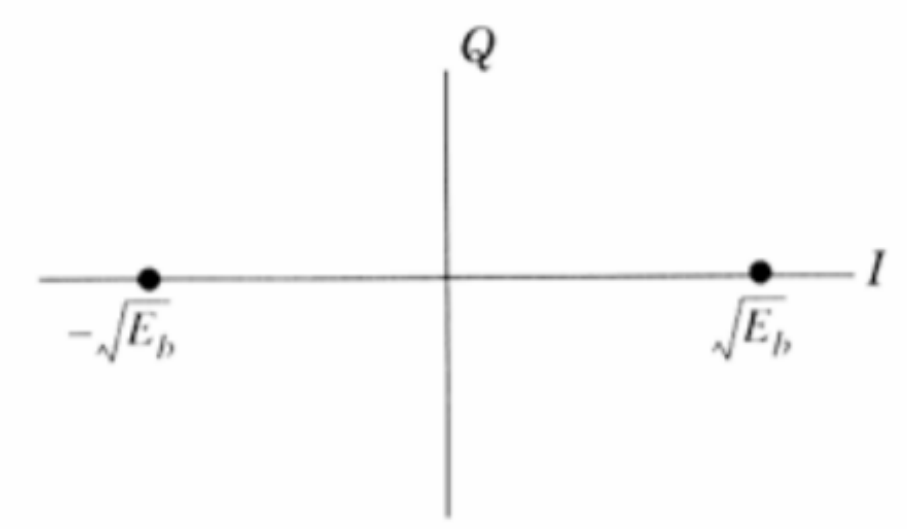

Figure 5.2 Constellation diagram of BPSK Modulation

The two signals are equally likely and let signal $s_1(t)$ is transmitted. Then, received signal from the matched filter (demodulator) is given as :

$$r = s_1 + n = \sqrt{E_b} + n \tag{5.3}$$

where n = N(0,σ^2) is the AWGN at receiver.

According to **ML detection rule** , if r¿=0 then decision is made that $s_1(t)$ is transmitted and if r¡0 then decision is made that $s_2(t)$ is transmitted. The **bit error probability** of BPSK signal for the AWGN channel is :

$$P_{e,BPSK} = Q(\sqrt{\frac{E_b}{\sigma^2}}) = Q(\sqrt{SNR}) \tag{5.4}$$

In the above equation , the term **SNR** = $\frac{E_b}{\sigma^2}$ is the **signal-to-noise ratio** of the AWGN channel.

5.3.3 About Bit error rate of OOK Modulation :

On–off keying (OOK) denotes the simplest form of **amplitude-shift keying (ASK)** modulation that represents digital data as the **presence** or **absence** of a carrier wave. In its simplest form, the **presence of a carrier** for a specific duration represents a **binary one**, while its **absence** for the same duration represents a **binary zero**. On–off keying is most commonly used to **transmit Morse code** over **radio frequencies** (referred to as CW (continuous wave) operation), although in principle any digital encoding scheme may be used. OOK has been used in the **ISM bands** to transfer data between computers. The **constellation diagram** of OOK Modulation is given as :

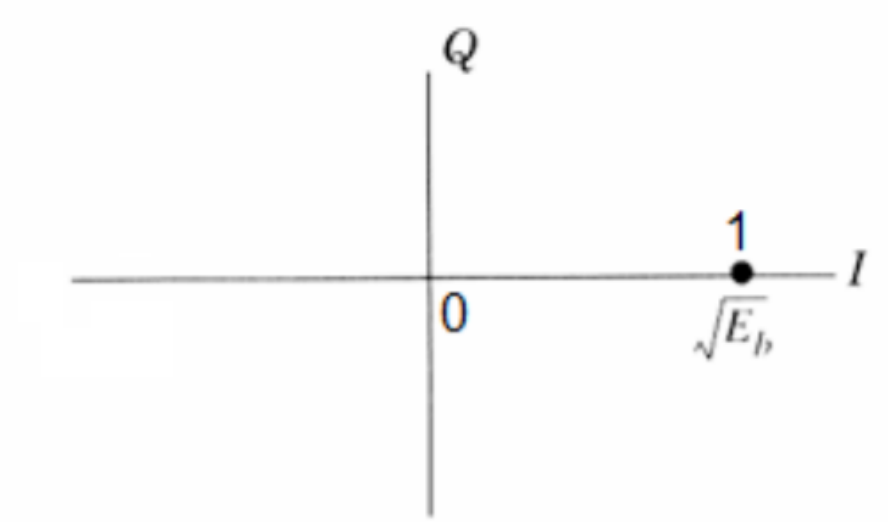

Figure 5.3 Constellation Diagram of OOK Modulation

According to **ML detection rule** , the decision is made that r = n if transmitted symbol is 0 and r = s_1 + n = $\sqrt{E_b}$ + n if transmitted symbol is 1.

The received signal is compared with threshold α = 1/2 and if r ¿= α then decision is made that 1 is transmitted otherwise decision is made that 0 is transmitted. The bit error rate of a OOK signal for the AWGN channel is :

$$P_{e,OOK} = Q(\sqrt{\frac{E_b}{2\sigma^2}}) = Q(\sqrt{SNR/2}) \tag{5.5}$$

5.4 Code and Results

5.4.1 Bit Error Rate of BPSK Modulation :

```
% 19ucc023
% Mohit Akhouri
% Observation 1 - Practical and Theoretical BER of BPSK Modulation

% This code will implement BPSK Modulation and compare the theoretical
 and
% analytical BER

% This code will also plot the graph between Theoretical and Practical
 BER
% vs. Signal to Noise Ratio ( SNR )

clc;
clear all;
close all;

size = 10000; % intializing the size for the random variable and input
 signal
BER_Practical = zeros(1,10); % Initializing the Array to store
 practical values of BER
BER_Theoretical = zeros(1,10); % Initializing the Array to store
 Theoretical values of BER

x=zeros(1,size); % Initializing the array to store the POLAR input
 signal x[n]

% ALGORITHM for initializing a POLAR SIGNALLING x[n]
for i=1:size
    rnd = rand();
    if(rnd>0.5)
        x(i)=1;  % +V in POLAR SIGNALLING
    else
        x(i)=-1; % -V in POLAR SIGNALLING
    end
end

SNR_dB = 0:9; % defining the range of Signal to Noise Ratio ( Measured
 in dB )

% Main loop algorithm for calculation of x[n],y[n], noise "n"
% and calculation of theoretical and practical BER
for i=1:length(SNR_dB)

    SNR=10^((i-1)/10);
    N = 1/SNR;
    M=sqrt(N/2);

    y=zeros(1,size); % to store the output signal y[n] = x[n] + n , n=
 AWGN noise
    n=zeros(1,size); % to store the AWGN noise

    % Loop for calculation of AWGN noise and storing in variable 'n'
```

Figure 5.4 Part 1 of the Code for calculation of BER of BPSK Modulation

```
    for j=1:size
        n(j)=M*randn(); % using randn function to randomly choose any
 integer
    end

    % Loop to calculate the output signal y[n] = x[n] + n, n = AWGN
 noise
    for j=1:size
        y(j)=x(j)+n(j);
    end

    % Main Loop algorithm for ML-Detection of BPSK modulation
    yn=zeros(1,size);
    for j=1:size
        if(y(j)>=0) % Based on decision rule , either +V(1) or -V(-1)
 is choosen
            yn(j)=1;
        else
            yn(j)=-1;
        end
    end

    % Comparing the transmitted and received message signal
    % and calculating the Practical BER
    for j=1:size
        if(x(j)~=yn(j))
            BER_Practical(i)=BER_Practical(i)+1;
        end
    end

    BER_Practical(i)=BER_Practical(i)/size; % Calculation of Practical
 BER
    BER_Theoretical(i)=qfunc(sqrt(2/N));  % Calculation of Theoretical
 BER using Q function

end

% Display of Theoretical and Practical BER
disp(sprintf('%-10s \t %-20s \t %-20s','index','Theoretical
 BER','Practical BER'));
for i=1:10
    disp(sprintf('%-10i %-20d \t
 %-20d',i,BER_Practical(i),BER_Theoretical(i)));
end

% Plots of Practical and Theoretical BER vs. Signal to Noise Ratio
 ( SNR )
% in dB
semilogy(SNR_dB,BER_Practical,'Color','blue'); % semilogy used for
 plotting on base-10 logarithmic scale on Y-axis
hold on;
semilogy(SNR_dB,BER_Theoretical,'Color','red'); % semilogy used for
 plotting on base-10 logarithmic scale on Y-axis
```

Figure 5.5 Part 2 of the Code for calculation of BER of BPSK Modulation

```
ylabel('Bit Error Rate (BER) ->');
xlabel('SNR(dB) ->');
legend('Practical BER','Theoretical BER');
title('19ucc023 - Mohit Akhouri','Plot of Theoretical and Practical
 BER vs. SNR(dB) for BPSK modulation');
grid on;

hold off;
```

Figure 5.6 Part 3 of the Code for calculation of BER of BPSK Modulation

Command Window

index	Theoretical BER	Practical BER
1	7.620000e-02	7.864960e-02
2	5.730000e-02	5.628195e-02
3	3.750000e-02	3.750613e-02
4	2.380000e-02	2.287841e-02
5	1.220000e-02	1.250082e-02
6	7.500000e-03	5.953867e-03
7	2.600000e-03	2.388291e-03
8	4.000000e-04	7.726748e-04
9	2.000000e-04	1.909078e-04
10	1.000000e-04	3.362723e-05

Figure 5.7 Table of Theoretical and Practical values of BER of BPSK Modulation

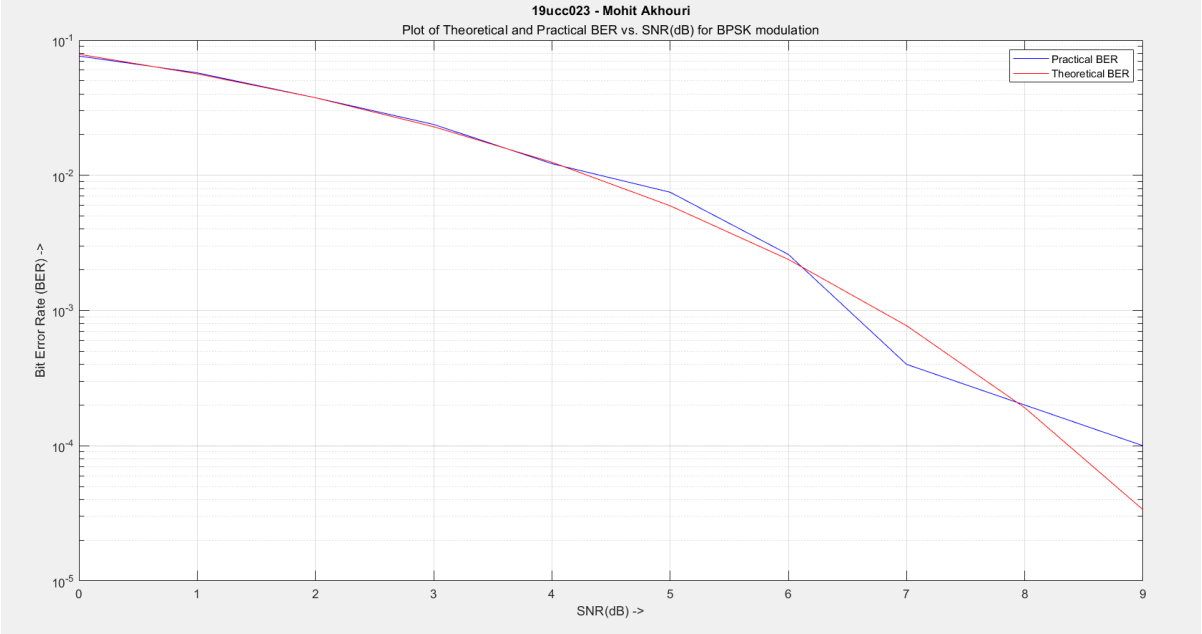

Figure 5.8 Plot of Theoretical and Practical BER vs. SNR (dB) for BPSK Modulation

5.4.1.1 Observation Table for BER of BPSK Modulation :

BER	Analytical BER	Simulated BER
1	0.0762	0.0786
2	0.0573	0.0562
3	0.0375	0.0375
4	0.0238	0.0228
5	0.0122	0.0125
6	0.0075	0.0059
7	0.0026	0.0023
8	0.0004	0.0007
9	0.0002	0.0001
10	0.0001	0.00003

Table 5.1 Analytical BER and Simulated BER for BPSK Modulation

5.4.2 Bit Error Rate of OOK Modulation :

```
% 19ucc023
% Mohit Akhouri
% Observation 2 - Practical and Theoretical BER of OOK Modulation

% This code will implement OOK Modulation and compare the theoretical
 and
% analytical BER

% This code will also plot the graph between Theoretical and Practical
 BER
% vs. Signal to Noise Ratio ( SNR )

clc;
clear all;
close all;

size = 10000; % intializing the size for the random variable and input
 signal
BER_Practical = zeros(1,10); % Initializing the Array to store
 practical values of BER
BER_Theoretical = zeros(1,10); % Initializing the Array to store
 Theoretical values of BER

x=zeros(1,size); % Initializing the array to store the POLAR input
 signal x[n]

% ALGORITHM for initializing a UNI-POLAR SIGNALLING x[n]
for i=1:size
    rnd = rand();
    if(rnd>0.5)
        x(i)=1;  % +V in UNI-POLAR SIGNALLING
    else
        x(i)=0; % 0 in UNI-POLAR SIGNALLING
    end
end

SNR_dB = 0:9; % defining the range of Signal to Noise Ratio ( Measured
 in dB )

% Main loop algorithm for calculation of x[n],y[n], noise "n"
% and calculation of theoretical and practical BER
for i=1:length(SNR_dB)

    SNR=10^((i-1)/10);
    N = 1/SNR;
    M=sqrt(N/2);

    y=zeros(1,size); % to store the output signal y[n] = x[n] + n , n=
 AWGN noise
    n=zeros(1,size); % to store the AWGN noise

    % Loop for calculation of AWGN noise and storing in variable 'n'
```

Figure 5.9 Part 1 of the Code for calculation of BER of OOK Modulation

```
    for j=1:size
        n(j)=sqrt(1/2)*M*randn(); % using randn function to randomly
 choose any integer
    end

    % Loop to calculate the output signal y[n] = x[n] + n, n = AWGN
 noise
    for j=1:size
        y(j)=x(j)+n(j);
    end

    % Main Loop algorithm for ML-Detection of OOK modulation
    yn=zeros(1,size);
    for j=1:size
        if(y(j)>=0.5) % Based on decision rule , either +V(1) or 0 is
 choosen
            yn(j)=1;
        else
            yn(j)=0;
        end
    end

    % Comparing the transmitted and received message signal
    % and calculating the Practical BER
    for j=1:size
        if(x(j)~=yn(j))
            BER_Practical(i)=BER_Practical(i)+1;
        end
    end

    BER_Practical(i)=BER_Practical(i)/size; % Calculation of Practical
 BER
    BER_Theoretical(i)=qfunc(sqrt(1/N));  % Calculation of Theoretical
 BER using Q function

end

% Display of Theoretical and Practical BER
disp(sprintf('%-10s \t %-20s \t %-20s','index','Theoretical
 BER','Practical BER'));
for i=1:10
    disp(sprintf('%-10i %-20d \t
 %-20d',i,BER_Practical(i),BER_Theoretical(i)));
end

% Plots of Practical and Theoretical BER vs. Signal to Noise Ratio
 ( SNR )
% in dB
semilogy(SNR_dB,BER_Practical,'Color','blue'); % semilogy used for
 plotting on base-10 logarithmic scale on Y-axis
hold on;
semilogy(SNR_dB,BER_Theoretical,'Color','red'); % semilogy used for
 plotting on base-10 logarithmic scale on Y-axis
```

Figure 5.10 Part 2 of the Code for calculation of BER of OOK Modulation

```
ylabel('Bit Error Rate (BER) ->');
xlabel('SNR(dB) ->');
legend('Practical BER','Theoretical BER');
title('19ucc023 - Mohit Akhouri','Plot of Theoretical and Practical
 BER vs. SNR(dB) for OOK modulation');
grid on;

hold off;
```

Figure 5.11 Part 3 of the Code for calculation of BER of OOK Modulation

index	Theoretical BER	Practical BER
1	1.565000e-01	1.586553e-01
2	1.282000e-01	1.309273e-01
3	1.030000e-01	1.040286e-01
4	7.780000e-02	7.889587e-02
5	5.390000e-02	5.649530e-02
6	3.420000e-02	3.767899e-02
7	2.290000e-02	2.300714e-02
8	1.420000e-02	1.258703e-02
9	5.700000e-03	6.004386e-03
10	3.100000e-03	2.413310e-03

Figure 5.12 Table of Theoretical and Practical values of BER of OOK Modulation

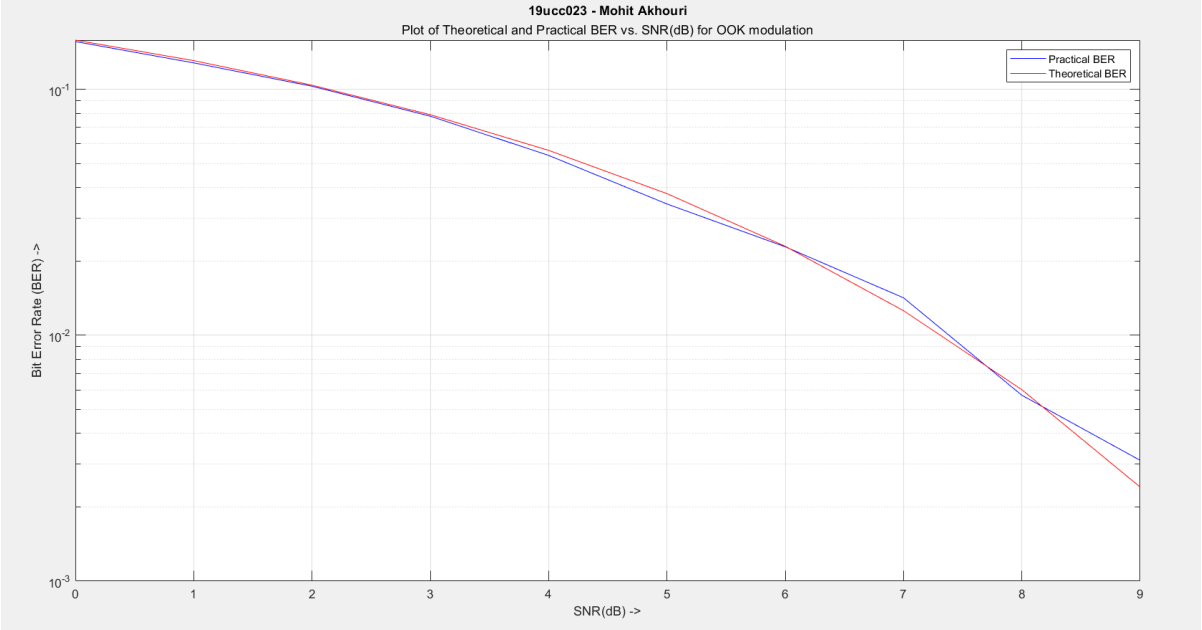

Figure 5.13 Plot of Theoretical and Practical BER vs. SNR (dB) for OOK Modulation

5.4.2.1 Observation Table for BER of OOK Modulation :

BER	Analytical BER	Simulated BER
1	0.1565	0.1586
2	0.1282	0.1309
3	0.1030	0.1040
4	0.0778	0.0788
5	0.0539	0.0564
6	0.0342	0.0376
7	0.0229	0.0230
8	0.0142	0.0125
9	0.0057	0.0060
10	0.0031	0.0024

Table 5.2 Analytical BER and Simulated BER for OOK Modulation

5.5 Conclusion

In this experiment , we learnt about two types of Modulation **BPSK Modulation** and **OOK Modulation**. We analysed how to generate the BPSK and OOK waveforms and calculate their Bit-Error rate. We learnt about important concepts like the **AWGN noise** and how it affects the Bit error rate. We also learnt about **Q-function** and how to calculate Theoretical BER of both BPSK and OOK Modulation. We implemented the codes in MATLAB and analysed the results. We also plotted the graph between BER and SNR (in dB) and verified the results.

Chapter *6*

Experiment - 6

6.1 Name of the Experiment

Performance analysis of M-ary phase-shift keying modulation scheme over AWGN channel

6.2 Software Used

- MATLAB
- Simulink

6.3 Theory

6.3.1 About AWGN channel :

Additive white Gaussian noise (AWGN) is a basic noise model used in information theory to mimic the effect of many random processes that occur in nature. The modifiers denote specific characteristics:

- Additive : it is added to any noise that might be intrinsic to the information system.
- White : refers to the idea that it has **uniform power** across the frequency band for the information system. It is an analogy to the color white which has **uniform emissions** at all frequencies in the **visible spectrum**.
- Gaussian : because it has a normal distribution in the time domain with an average time domain value of zero and variance σ^2.

Wideband noise comes from many natural noise sources, such as the thermal vibrations of atoms in conductors (referred to as thermal noise or Johnson–Nyquist noise), shot noise, black-body radiation from the earth and other warm objects, and from celestial sources such as the Sun. The **central limit theorem** of probability theory indicates that the summation of many random processes will tend to have distribution called **Gaussian** or **Normal**.

6.3.2 About Bit error rate of M-ary Phase Shift Keying :

M-ary phase-shift keying (MPSK) is employed in some of the digital cellular standards and communication geostationary satellite systems. MPSK employs a set of M equal-energy signals to represent M **equiprobable symbols**. This constant energy restriction (i.e., the constant envelope constraint) warrants a circular constellation for the signal points. In MPSK, the phase of the carrier takes on one of M possible values $\frac{2\pi(i-1)}{M}$, where i = 1, 2, ...,M. The MPSK signal set is thus analytically given by:

$$S_i t = \sqrt{\frac{2E_i}{T_s}} cos\left(2\pi f_c t - \frac{2\pi i - 1}{M}\right) \tag{6.1}$$

In the above equation , **t** ranges from 0 to T_s and values of **i** are from 1 to M. We assume a **uniform spacing** of phase values, i.e., the phase separation between any two adjacent signal points is **constant**. The Energy associated with the MPSK signal set is as follows :

$$E_i = E_s \tag{6.2}$$

In the above equation , **i** ranges from 1 to M. E_s represents the **average energy per symbol**.

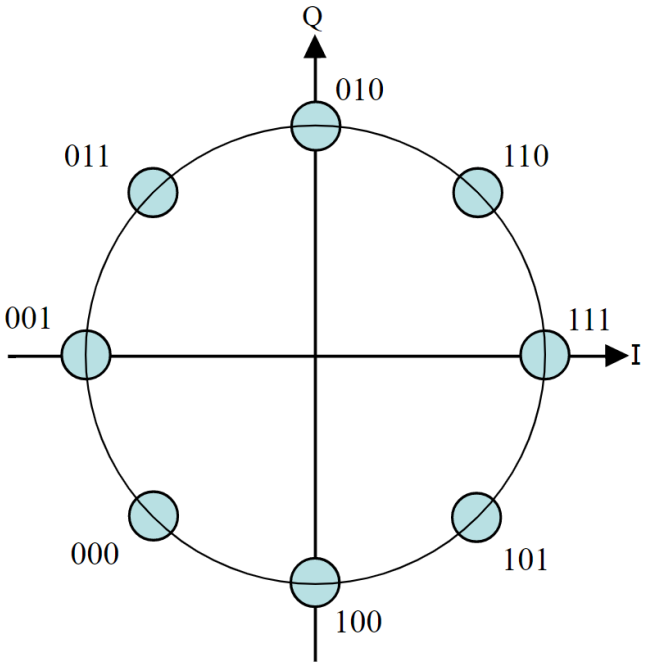

Figure 6.1 Constellation diagram of 8-ary Phase Shift Keying

The **MPSK signal set** is characterized by a **two dimensional signal space** and **M message points** :

$$S_i t = \sqrt{E_i} cos\left(\frac{2\pi(i-1)}{M}\right)\phi_1 t + \sqrt{E_i} sin\left(\frac{2\pi(i-1)}{M}\right)\phi_2 t \tag{6.3}$$

The **average symbol error probability** be tightly approximated as follows :

$$P_{SER-MPSK} \approx 2Q\sqrt{\frac{2E_s}{N_o}} sin\left(\frac{\pi}{M}\right) \tag{6.4}$$

6.3.3 About Bit error rate of 4-QPSK Modulation :

Quadrature Phase Shift Keying (QPSK) is a form of PSK which uses a combination of two bits (00, 01, 10 & 11). Each of this bit combination is represented by **four possible carrier phase shifts** (0, 90, 180 & 270). With QPSK **twice** as much information as ordinary PSK can be transmitted using the same bandwidth. In QPSK the **amplitude of carrier** remains **constant** for all symbols. QPSK modulation consists of **two BPSK modulation** on in-phase and **quadrature components** of the signal.

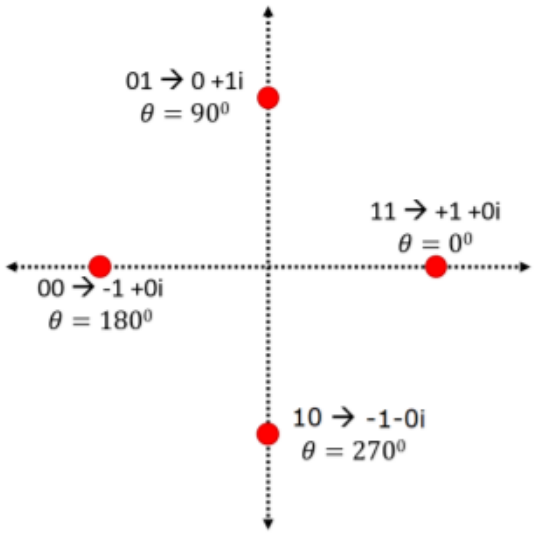

Figure 6.2 Constellation diagram of 4-QPSK Modulation

6.3.3.1 Symbol Error rate of 4-QPSK Modulation :

The BER of each branch is the same as BPSK :

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \tag{6.5}$$

The **symbol probability of error (SER)** is the probability of either branch has a bit error:

$$P_s = 1 - \left[1 - Q\left(\sqrt{\frac{2E_b}{N_o}}\right)\right]^2 \tag{6.6}$$

Since the **symbol energy** is **split** between the **two in-phase and quadrature components**, $\frac{E_s}{N_o} = \frac{2E_b}{N_o}$ and we have:

$$P_s = 1 - \left[1 - Q\left(\sqrt{\frac{E_s}{N_o}}\right)\right]^2 \tag{6.7}$$

6.4 Code and Results

6.4.1 BER and SER of M-ary Phase Shift Keying and 4-QPSK Modulation :

```
% 19ucc023
% Mohit Akhouri
% Observation 1 - Modulation order and Probability of Error for M-ary
 PSK

clc;
clear all;
close all;

size1 = 10; % intializing the size for BER and SER
size2 = 10000; % intializing the size for signal x[n]

SER_Practical = zeros(1,size1); % Initializing Practical SER array
SER_Theoretical = zeros(1,size1); % Initializing Theoretical SER array

BER_Real_part = zeros(1,size1); % Initializing Real part of
 BER_Practical
BER_Imag_part = zeros(1,size1); % Initializing Imaginary part of
 BER_Practical

BER_Practical = zeros(1,size1); % Initializing Practical BER array
BER_Theoretical = zeros(1,size1); % Initializing Theoretical BER array

x_real = zeros(1,size2); % Initializing Real part of x[n]
x_img = zeros(1,size2); % Initializing Imaginary part of x[n]
x = zeros(1,size2); % Intializing input signal x[n]
y = zeros(1,size2); % Initializing Output signal y[n]

% Main loop algorithm for calculation of transmitted signal x[n]
for i=1:size2

    rnd1 = rand(); % random value 1 generation
    rnd2 = rand(); % random value 2 generation

    % Constructing the real part of signal on the basis of decision
    if(rnd1 > 0.5)
        x_real(i)=1;
    else
        x_real(i)=-1;
    end

    % Constructing the Imaginary part of signal on the basis of
 decision
    if(rnd2 > 0.5)
        x_img(i)=1;
    else
        x_img(i)=-1;
    end
end

x = x_real + (1j * x_img); % Overall transmitted signal x[n]
```

Figure 6.3 Part 1 of the Code for Observation 1

```
SNR_dB = 0:9; % defining the range of Signal to Noise Ratio ( Measured
 in dB )

% Main loop algorithm for calculation of x[n],y[n], noise "n"
% and calculation of theoretical and practical BER and SER
for i=1:length(SNR_dB)

    SNR=10^((i-1)/10);
    N = 1/SNR;
    M = sqrt(N/2);

    n=zeros(1,size2); % Initializing noise signal n

    % loop for calculation of noise signal
    for j=1:size2
        n(j) = M*randn() + ( 1j * M * randn());
    end

    % loop for calculation of received AWGN + x[n] signal
    for j=1:size2
        y(j) = x(j) + n(j);
    end

    yn = zeros(1,size2);
    y_real = zeros(1,size2);
    y_img = zeros(1,size2);

    % Main Loop algorithm for ML-Detection of M-ary and QPSK
 Modulation
    for j=1:size2
        if(real(y(j)) >= 0)
            y_real(j) = 1;
        else
            y_real(j) = -1;
        end

        if(imag(y(j))>=0)
            y_img(j) = 1;
        else
            y_img(j) = -1;
        end

        yn = y_real + (1j * y_img);
    end

    % Comparing the transmitted and received message signal
    % and calculating the Practical BER and SER
    for j=1:size2
        if(x(j)~=yn(j))
            SER_Practical(i) = SER_Practical(i) + 1;
        end

        if(real(x(j)) ~= real(yn(j)))
            BER_Real_part(i) = BER_Real_part(i) + 1;
```

Figure 6.4 Part 2 of the Code for Observation 1

```
        end

        if(imag(x(j)) ~= imag(yn(j)))
            BER_Imag_part(i) = BER_Imag_part(i) + 1;
        end
    end

    BER_Practical(i) = ((BER_Real_part(i)/size2) + (BER_Imag_part(i)/
size2))/2;
    BER_Theoretical(i) = qfunc(sqrt(2/N));

    SER_Practical(i) = SER_Practical(i)/size2;
    SER_Theoretical(i) = 2 * qfunc(sqrt(2/N));
end

SER_matrix = zeros(10,5); % Matrix for storing SER values for
 different modulation orders
M = [2,4,8,16,32]; % array of modulation orders M

% Loop for calculation of SER for different modulation order M
for i=1:length(SNR_dB)
    SNR = 10^((i-1)/10);
    N = 1/SNR;

    for m = 1:length(M)
        SER_matrix(i,m) = 2 * qfunc(sqrt(2/N) * sin(pi/M(m)));
    end

end

% Displaying the SER matrix
disp('SER vs. Modulation order matrix is given as:');
disp(SER_matrix);

% Plot of SER for different modulation order
figure;
semilogy(SNR_dB,SER_matrix(:,1),'color','blue');
hold on;
semilogy(SNR_dB,SER_matrix(:,2),'color','black');
semilogy(SNR_dB,SER_matrix(:,3),'color','red');
semilogy(SNR_dB,SER_matrix(:,4),'color','magenta');
semilogy(SNR_dB,SER_matrix(:,5),'color','cyan');
ylabel('SER ->');
xlabel('SNR(dB) ->');
title('19ucc023 - Mohit Akhouri','Plots of SER for different values of
 Modulation order (M) for M-ary Phase Shift Keying');
legend('SER for M = 2','SER for M = 4','SER for M = 8','SER for M =
 16','SER for M = 32');
grid on;
hold off;

% Plots of practical and theoretical SER vs. SNR ( in dB )
figure;
semilogy(SNR_dB,SER_Practical,'Color','blue');
```

Figure 6.5 Part 3 of the Code for Observation 1

```
hold on;
semilogy(SNR_dB,SER_Theoretical,'Color','red');
xlabel('SNR(dB) ->');
ylabel("SER ->");
title('19ucc023 - Mohit Akhouri','Plots of Practical and Theoretical
 SER vs. SNR (dB) for 4-QPSK Modulation');
legend('Practical SER','Theoretical SER');
grid on;
hold off;

% Plots of practical and theoretical BER vs. SNR ( in dB )
figure;
semilogy(SNR_dB,BER_Practical,'Color','blue');
hold on;
semilogy(SNR_dB,BER_Theoretical,'Color','red');
xlabel('SNR(dB) ->');
ylabel("BER ->");
title('19ucc023 - Mohit Akhouri','Plots of Practical and Theoretical
 BER vs. BER (dB) for 4-QPSK Modulation');
legend('Practical BER','Theoretical SER');
grid on;
hold off;
```

Published with MATLAB® R2020b

Figure 6.6 Part 4 of the Code for Observation 1

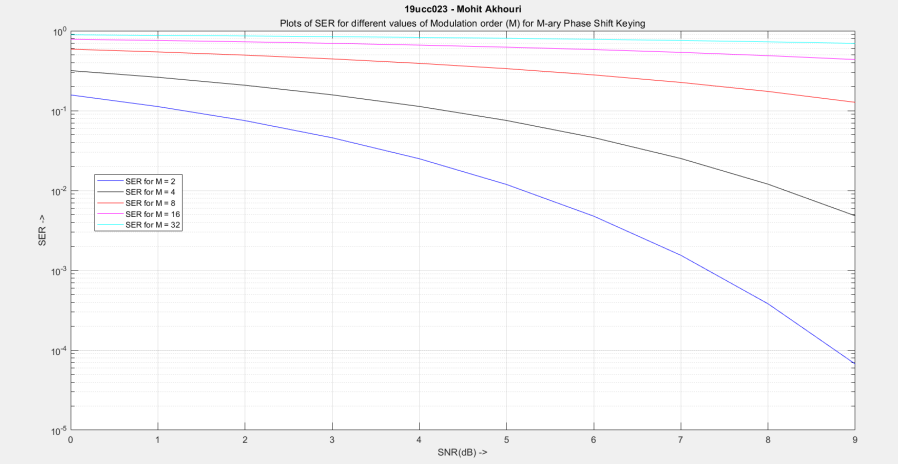

Figure 6.7 Plots of SER for different values of Modulation order M

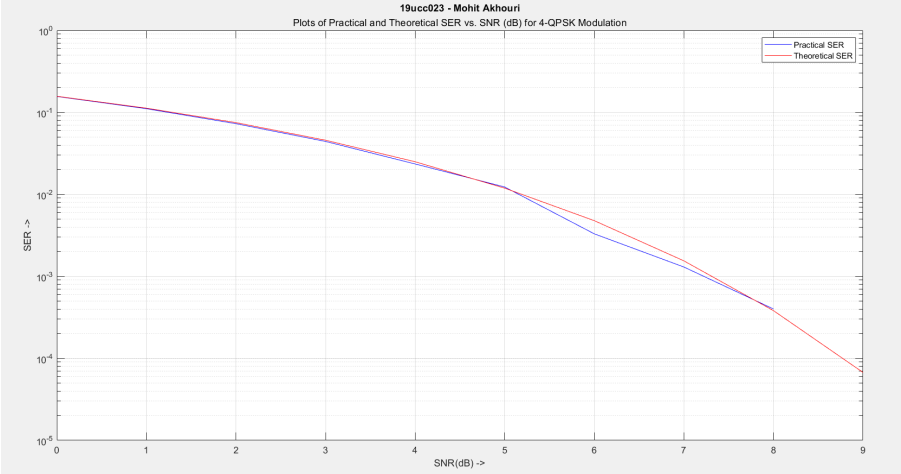

Figure 6.8 Plots of Theoretical and Practical SER vs. SNR (dB)

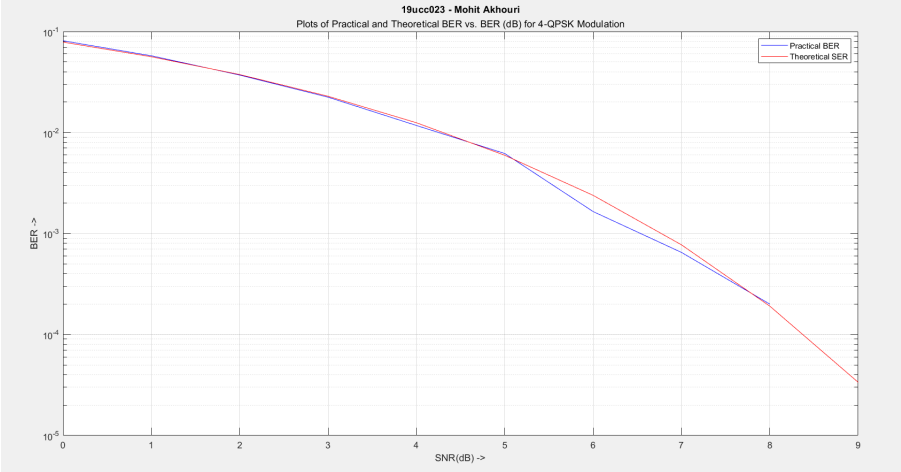

Figure 6.9 Plots of Theoretical and Practical BER vs. SNR (dB)

6.4.2 Relationship between Probability of error and Modulation order :

```
% 19ucc023
% Mohit Akhouri
% Observation 2 - Calculation and Plotting of Probability of Error vs.
% Modulation order

% This code will first calculate the Probability of error for
 different
% modulation order and Plot the graphs between them

clc;
clear all;
close all;

val1 = 5; % SNR = 5 dB stored in val1
val2 = 10; % SNR = 10 dB stored in val2
M=[2 4 8 16 32]; % Initializing the modulation order (M) array

PBE_for_SNR_5 = zeros(1,5); % Initializing array 1 for PBE for SNR = 5
 dB
PBE_for_SNR_10 = zeros(1,5); % Initializing array 2 for PBE for SNR =
 10 dB

% Calculation of Probability of Error for SNR = 5 dB

SNR_1 = 10.^(val1/10); % Calculating SNR 1

for i=1:5
    PBE_for_SNR_5(i) = 2 * qfunc(sqrt(SNR_1))*sin(pi/M(i));
end

% Calculation of Probability of Error for SNR = 10 dB

SNR_2 = 10.^(val2/10); % Calculating SNR 2

for i=1:5
    PBE_for_SNR_10(i) = 2*qfunc(sqrt(SNR_2))*sin(pi/M(i));
end

% Displaying the values of probability of error for different values
 of M
display('Probability of error (for SNR = 5dB) values for
 M=2,4,8,16,32 :');
display(PBE_for_SNR_5);

display('Probability of error (for SNR = 10dB) values for
 M=2,4,8,16,32 :');
display(PBE_for_SNR_10);

% Plots of Probability of error vs. Modulation order M for SNR = 5 dB
figure;
plot(PBE_for_SNR_5);
xlabel('Modulation order (M) ->');
```

Figure 6.10 Part 1 of the Code for Observation 2

```
ylabel('Probability of error ->');
title('19ucc023 - Mohit Akhouri','Probability of error vs. Modulation
 order (M) for SNR = 5 dB');
grid on;

% Plots of Probability of error vs. Modulation order M for SNR = 10 dB
figure;
plot(PBE_for_SNR_10);
xlabel('Modulation order (M) ->');
ylabel('Probability of error ->');
title('19ucc023 - Mohit Akhouri','Probability of error vs. Modulation
 order (M) for SNR = 10 dB');
grid on;
```

Published with MATLAB® R2020b

Figure 6.11 Part 2 of the Code for Observation 2

```
Probability of error (for SNR = 5dB) values for M=2,4,8,16,32 :

PBE_for_SNR_5 =

    0.0754    0.0533    0.0288    0.0147    0.0074

Probability of error (for SNR = 10dB) values for M=2,4,8,16,32 :

PBE_for_SNR_10 =

    0.0016    0.0011    0.0006    0.0003    0.0002
```

Figure 6.12 Display of values of Probability of error for different values of M

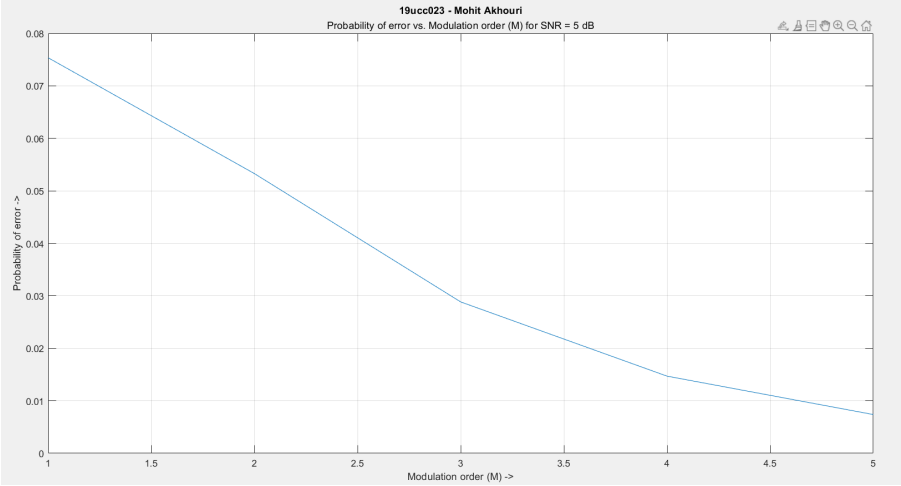

Figure 6.13 Plot of Probability of error vs. Modulation order for SNR = 5dB

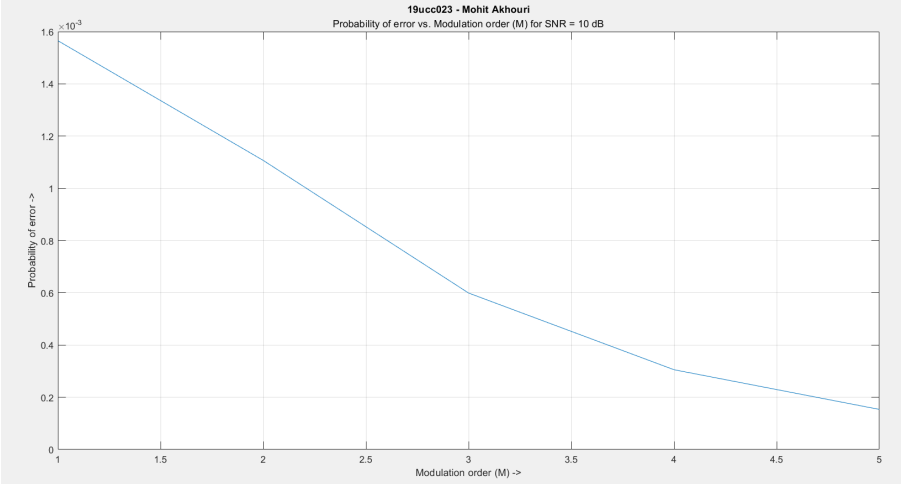

Figure 6.14 Plot of Probability of error vs. Modulation order for SNR = 10dB

6.5 Conclusion

In this experiment , we learnt about two types of Modulation **M-ary PSK Modulation** and **4-QPSK Modulation**. We analysed how to generate the M-ary PSK and 4-QPSK waveforms and calculate their Bit-Error rate and symbol error rate. We learnt about important concepts like the **AWGN noise** and how it affects the Bit error rate. We also learnt about **Q-function** and how to calculate Theoretical BER of both M-ary PSK and 4-QPSK Modulation. We implemented the codes in MATLAB and analysed the results. We also plotted the graph between BER and SNR (in dB) and also between SER and SNR (in dB) and verified the results.

Chapter *7*

Experiment - 7

7.1 Name of the Experiment

To perform encoding and decoding for a (7,4) Hamming code

7.2 Software Used

- MATLAB

7.3 Theory

7.3.1 About Hamming Code :

In computer science and telecommunication, **Hamming codes** are a family of **linear error-correcting codes**. Hamming codes can detect **one-bit** and **two-bit** errors, or correct one-bit errors without detection of uncorrected errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are perfect codes, that is, they achieve the highest possible rate for codes with their block length and **minimum distance of three**. **Richard W. Hamming** invented Hamming codes in 1950 as a way of automatically correcting errors introduced by punched card readers. In his original paper, Hamming elaborated his general idea, but specifically focused on the **Hamming(7,4) code** which adds **three parity bits** to **four bits of data**.

In mathematical terms, Hamming codes are a class of **binary linear code**. For each integer r $>= 2$ there is a code with block length n = 2^r - 1 and message length k = 2^r - r - 1. Hence the rate of Hamming codes is R = k / n = 1 - r / (2^r - 1), which is the highest possible for codes with minimum distance of three (i.e., the minimal number of bit changes needed to go from any code word to any other code word is three) and block length 2^r - 1.

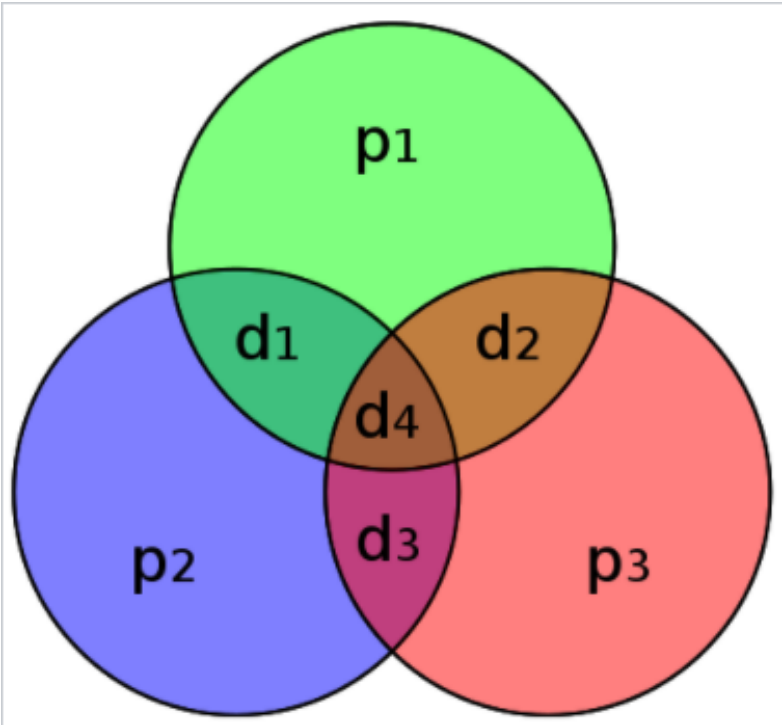

Figure 7.1 Graphical depiction of the four data bits and three parity bits - Hamming(7,4)

7.3.1.1 About (7,4) Hamming Code :

In **1950**, Hamming introduced the [7,4] Hamming code. It encodes **four data bits** into **seven bits** by adding **three parity bits**. It can detect and correct single-bit errors. With the addition of an overall parity bit, it can also detect (but not correct) double-bit errors.

The (7,4) Hamming Code is given by the generator matrix :

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{7.1}$$

The matrix **G** is called the **generator** of **canonical** matrix of linear (n,k) code. The matrix **H** is called **parity-check matrix**. The matrix **H** is given as :

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \tag{7.2}$$

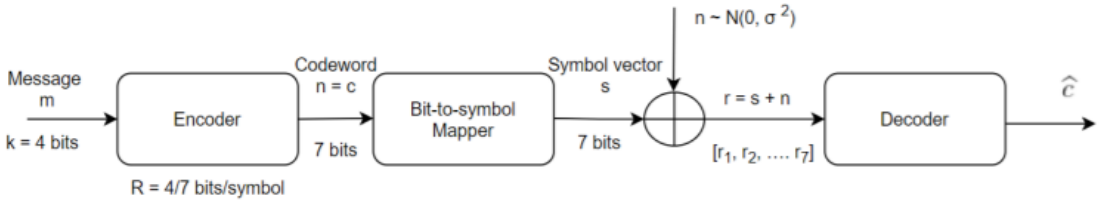

Figure 7.2 Block diagram of hamming code

7.3.1.2 About Hard Decision Decoder :

In **hard decision decoder** , we find codeword closest in **Hamming distance**. In this type of decoder , we will find distance of **b** from 2^k codeword. In this type of decoder , **complexity** increases **exponentially** with k.

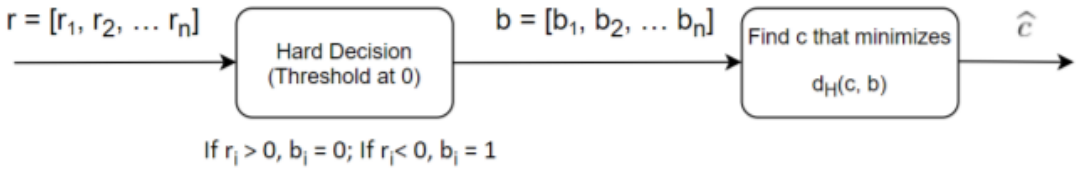

Figure 7.3 Hard Decision Decoder

7.3.1.3 About Soft Decision Decoder :

In **soft decision decoder** , we find codeword closest in **Euclidean distance**. In this type of decoder , we will find distance with 2^k code-symbol vectors . In this type of decoder , **Complexity** is **more** than hard decision decoder.

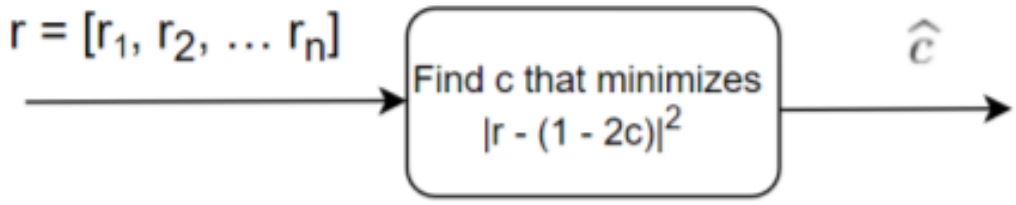

Figure 7.4 Soft Decision Decoder

7.4 Code and Results

7.4.1 Encoding and Decoding of (7,4) Hamming Code :

```
% 19ucc023
% Mohit Akhouri
% Observation - Performing the encoding and decoding for a (7,4)
 Hamming
% Code and finding BER for different values of SER

clc;
clear all;
close all;

% This code will perform the encoding and decoding of (7,4) Hamming
 Code
% The Hamming code is given by the generator function G and we will
 use the
% block diagram of encoder and decoder to perform the operations. We
 will
% use two types of decoder - SOFT DECISION decoder and HARD DECISION
% decoder.

% Finally we will find the values of BER for different SER and also
 plot
% the graph between SER and BER for hard and soft decision decoder.

N = 10000; % Size of input sequence

n = 7; % columns of Generating matrix G for Hamming Code
k = 4; % rows of Generating matrix G for Hamming Code

G = zeros(4,7); % Initializing the generating matrix G
G=[1 0 0 0 1 0 1 ; 0 1 0 0 1 1 1 ; 0 0 1 0 1 1 0 ; 0 0 0 1 0 1 1]; %
 Defining the generating matrix G

SER_dB = (0:7); % Defining the array to store SER values in dB from
 0dB to 7dB
SER_array = zeros(1,8); % array to store the SER values converted from
 dB to unitless

% Loop to calculate unitless value of SER
for i=1:size(SER_dB)
    SER_array(i) = 10^(SER_dB(i)/10);
end

size_SER = size(SER_array,2); % to store the number of elements in the
 SER_array

message = zeros(16,4); % to store the Message bits ( 0000 to 1111 )

% Loop to calculate the message in terms of bits and storing them in
% message matrix for further computation
for i=1:16
    str = dec2bin(i-1,4); % To get the binary equivalent of decimal
 number'i-1'
```

Figure 7.5 Part 1 of the Code for Encoding and Decoding of (7,4) Hamming Code

```
    temp_array = zeros(1,4); % To store the bits of the binary
 equivalent obtained from above

    % Loop to store the bits in the correct position of temp_array
    for j=1:size(str,2)
        if(str(j)=='0')
            temp_array(j) = 0;
        else
            temp_array(j) = 1;
        end
    end
    message(i,:) = temp_array; % to store the encoded bits in the
 message matrix
end

codeword = mod(message*G,2); % To store the corresponding codewords to
 each bit sequence of message matrix

% Displaying the bit sequence and corresponding codeword
disp('The codewords are as follows :');
disp(sprintf('%-15s \t %-15s','Message','Codeword'));
for i=1:16
    disp(sprintf('%-15s \t
 %-15s',int2str(message(i,:)),int2str(codeword(i,:))));
end

codeword_modf = codeword; % Initializing array to store the modified
 codewords
codeword_modf(codeword_modf==0) = -1; % modify the codewords by
 comparing them to 0 and then assigning -1

INFO_mat = randi([0,1],N,4,size_SER); % to store the information to be
 transmitted through AWGN channel

H = [G(:,k+1:n)',eye(n-k)]; % To store the H matrix obtained by
 transpose of G matrix and multiplying with Identity matrix I

% Loop to calculate the information ( bit sequence ) to be transmitted
for i=1:size_SER
    code_transm(:,:,i) = mod((INFO_mat(:,:,i)*G),2);
end

code_transm_modf = code_transm; % To store the modified transmitted
 codeword
code_transm_modf(code_transm_modf==0) = -1; % Modification of
 transmitted codeword done by comparing to 0 and then assigning -1

% Loop to determine the information ( bit sequence ) received
for i=1:size_SER
    code_recv(:,:,i) = awgn(code_transm_modf(:,:,i),SER_dB(i));
end

hard_dec_decoder = ones(size(code_recv)); % Initializing array to
 store the decoded bit sequence from hard decision decoder
```

Figure 7.6 Part 2 of the Code for Encoding and Decoding of (7,4) Hamming Code

```
hard_dec_decoder(code_recv<0) = 0; % Parity checking and modifying the
 hard_dec_decoder array

INFO_hard_dec = zeros(size(INFO_mat)); % To store the received
 information through hard decision decoder
% Main loop algorithm for calculation of bit error rate in case of
 hard
% decision decoder by comparing the distances
for h=1:size_SER
    for i=1:N
        dist = zeros(1,2^k); % to store the distance of codewords

        % loop to calculate the distance of codewords
        for j=1:2^k
            dist(j) = norm(mod(codeword(j,:) +
 hard_dec_decoder(i,:,h),2),1);
        end

        % finding minimum distance index
        [min_elem,ind] = min(dist);
        INFO_hard_dec(i,:,h) = message(ind,:);
    end
    BER_hard_dec(h) = length(find(INFO_hard_dec(:,:,h) -
 INFO_mat(:,:,h)))/(4*N); % Computing the bit error rate for hard
 decision decoder
end

% Displaying the values obtained for the BER for different values of
 SER
% (in dB ) for hard decision decoder
disp('BER value for different values of SER for hard decision decoder
 is as follows :');
disp(sprintf('%-8s \t %-8s','SER (dB)','BER'));
for i=1:8
    disp(sprintf('%-8d \t %-8f',SER_dB(i),BER_hard_dec(i)));
end

INFO_soft_dec = zeros(size(INFO_mat)); % To store the received
 information through soft decision decoder
% Main loop algorithm for calculation of bit error rate in case of
 soft
% decision decoder by comparing the distances
for h=1:size_SER
    for i=1:N
        dist = zeros(1,2^k); % to store the distance of codewords

        % loop to calculate the distance of codewords
        for j=1:2^k
            dist(j) = norm((codeword_modf(j,:) - code_recv(i,:,h)),2);
        end

        % finding minimum distance index
        [min_elem,ind] = min(dist);
        INFO_soft_dec(i,:,h) = message(ind,:);
```

Figure 7.7 Part 3 of the Code for Encoding and Decoding of (7,4) Hamming Code

```
    end
    BER_soft_dec(h) = length(find(INFO_soft_dec(:,:,h) -
 INFO_mat(:,:,h)))/(4*N); % Computing the bit error rate for soft
 decision decoder
end

% Displaying the values obtained for the BER for different values of
 SER
% (in dB ) for soft decision decoder
disp('BER value for different values of SER for soft decision decoder
 is as follows :');
disp(sprintf('%-8s \t %-8s','SER (dB)','BER'));
for i=1:8
    disp(sprintf('%-8d \t %-8f',SER_dB(i),BER_soft_dec(i)));
end

% Plotting graph of Bit error rate ( BER ) vs. Symbol error rate
 ( SER )
figure;
semilogy(SER_dB,BER_hard_dec,'red');
hold on;
semilogy(SER_dB,BER_soft_dec,'blue')
xlabel('SER (dB) ->');
ylabel('BER (dB) ->');
title('19ucc023 - Mohit Akhouri','Plots of Bit Error Rate ( BER )
 vs. Symbol Error Rate ( SER in dB ) for HARD and SOFT decision
 decoders');
legend('Hard Decision Decoder','Soft Decision Decoder');
grid on;
hold off;
```

Figure 7.8 Part 4 of the Code for Encoding and Decoding of (7,4) Hamming Code

```
The codewords are as follows :
Message              Codeword
0  0  0  0           0  0  0  0  0  0  0
0  0  0  1           0  0  0  1  0  1  1
0  0  1  0           0  0  1  0  1  1  0
0  0  1  1           0  0  1  1  1  0  1
0  1  0  0           0  1  0  0  1  1  1
0  1  0  1           0  1  0  1  1  0  0
0  1  1  0           0  1  1  0  0  0  1
0  1  1  1           0  1  1  1  0  1  0
1  0  0  0           1  0  0  0  1  0  1
1  0  0  1           1  0  0  1  1  1  0
1  0  1  0           1  0  1  0  0  1  1
1  0  1  1           1  0  1  1  0  0  0
1  1  0  0           1  1  0  0  0  1  0
1  1  0  1           1  1  0  1  0  0  1
1  1  1  0           1  1  1  0  1  0  0
1  1  1  1           1  1  1  1  1  1  1
```

Figure 7.9 Display of **Message** and corresponding **codeword**

```
BER value for different values of SER for hard decision decoder is as follows :
SER (dB)     BER
0            0.139775
1            0.100200
2            0.072525
3            0.047625
4            0.022400
5            0.011400
6            0.003600
7            0.001700
```

Figure 7.10 Display of the BER values for different SER values (in dB) for Hard Decision Decoder

```
BER value for different values of SER for soft decision decoder is as follows :
SER (dB)     BER
0            0.102750
1            0.063375
2            0.041000
3            0.023775
4            0.008925
5            0.003400
6            0.000900
7            0.000300
```

Figure 7.11 Display of the BER values for different SER values (in dB) for Soft Decision Decoder

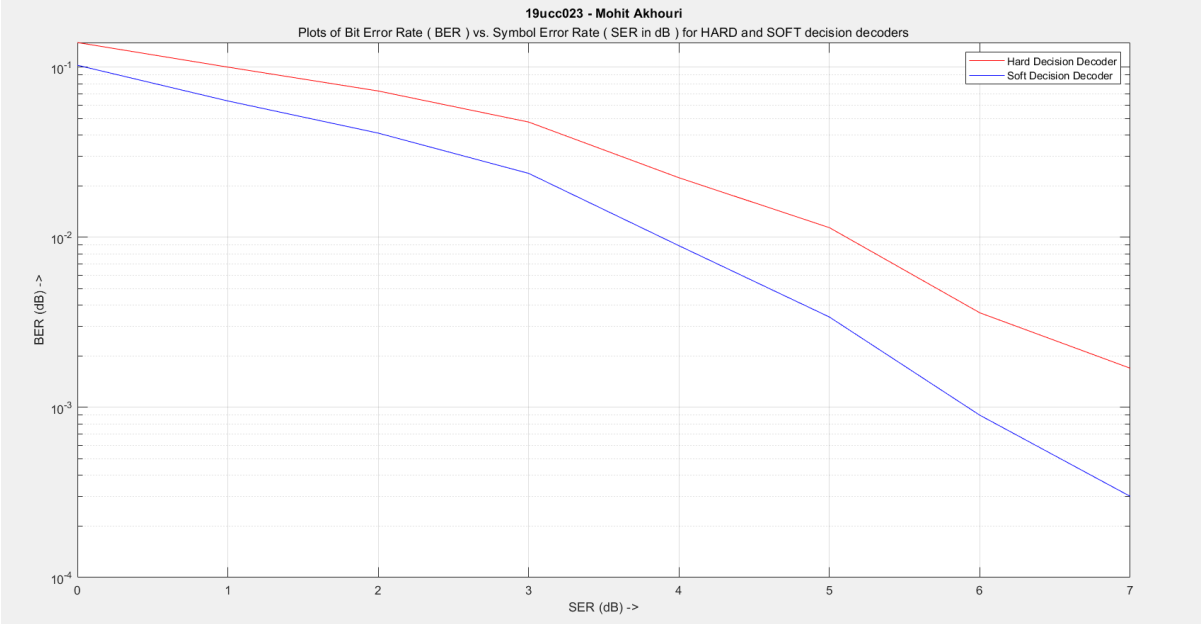

Figure 7.12 Plot of the BER vs. SER (dB) for Hard Decision Decoder and Soft Decision Decoder

7.4.2 Observation Table for BER vs. SER (dB) for Hard Decision and Soft Decision Decoders :

SER (in dB)	BER for Hard Decision Decoder	BER for Soft Decision Decoder
0	0.139775	0.102750
1	0.100200	0.063375
2	0.072525	0.041000
3	0.047625	0.023775
4	0.022400	0.008925
5	0.011400	0.003400
6	0.003600	0.000900
7	0.001700	0.000300

Table 7.1 BER for different values of SER for Hard and Soft Decision Decoders

7.5 Conclusion

In this experiment , We learnt about the **(7,4) Hamming Code** and how to implement it in MATLAB. We learnt about the block diagram of the Hamming Code for **encoding**. We also learnt about the **received** code after passing through AWGN channel and about two types of decoders - **Hard Decision Decoder** and **Soft Decision Decoder**. We also learnt about the concepts of **Euclidean distance** and **Hamming distance** and how they can be utilized in the calculation of distance. We also found BER values for different SER values (in dB) for both hard and soft decision decoders. We also plotted the graph between SER and BER. We also learnt about new MATLAB functions like **norm**,**mod** and **awgn** to name a few.

Chapter *8*

Experiment - 8

8.1 Name of the Experiment

Performance analysis of Convolutional Encoding and Viterbi Decoding

8.2 Software Used

- MATLAB

8.3 Theory

8.3.1 About Convolutional code :

In telecommunication, a **convolutional code** is a type of error-correcting code that generates parity symbols via the sliding application of a boolean polynomial function to a data stream. The sliding application represents the 'convolution' of the encoder over the data, which gives rise to the term 'convolutional coding'. The sliding nature of the convolutional codes facilitates trellis decoding using a time-invariant trellis. **Time invariant trellis decoding** allows convolutional codes to be **maximum-likelihood soft-decision decoded** with **reasonable complexity**.

Convolutional codes are often described as **continuous**. However, it may also be said that convolutional codes have arbitrary block length, rather than being continuous, since most real-world convolutional encoding is performed on blocks of data. Convolutionally encoded block codes typically employ termination. The arbitrary block length of convolutional codes can also be contrasted to classic block codes, which generally have fixed block lengths that are determined by **algebraic properties**.

The code rate of a convolutional code is commonly modified via **symbol puncturing**. The performance of a punctured convolutional code generally scales well with the amount of parity transmitted. The ability to perform economical soft decision decoding on convolutional codes, as well as the block length and code rate flexibility of convolutional codes, makes them very popular for **digital communications**.

8.3.2 About Convolutional Encoding :

To convolutionally encode data, start with **k memory registers**, each holding one input bit. Unless otherwise specified, all memory registers start with a value of 0. The encoder has n modulo-2 adders (a modulo 2 adder can be implemented with a single Boolean **XOR gate**, where the logic is: 0+0 = 0, 0+1 = 1, 1+0 = 1, 1+1 = 0), and n generator polynomials - one for each adder. An input bit m_1 is fed into the leftmost register. Using the **generator polynomials** and the existing values in the remaining registers, the encoder outputs n symbols. These symbols may be transmitted or punctured depending on the desired code rate. Now bit shift all register values to the right (m_1 moves to m_0, m_0 moves to m_{-1}) and wait for the next input bit. If there are no remaining input bits, the encoder continues shifting until all registers have returned to the zero state (**flush bit termination**).

8.3.2.1 1/2 Convolutional Encoder :

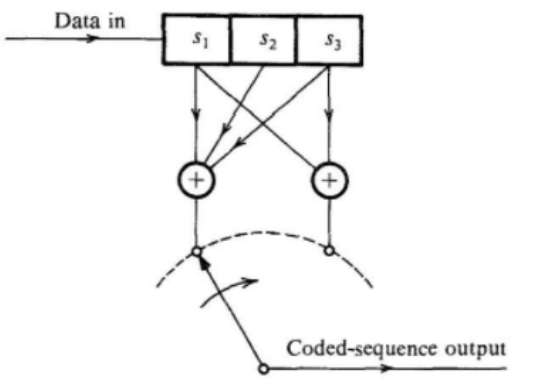

Figure 8.1 1/2 Convolutional Encoder

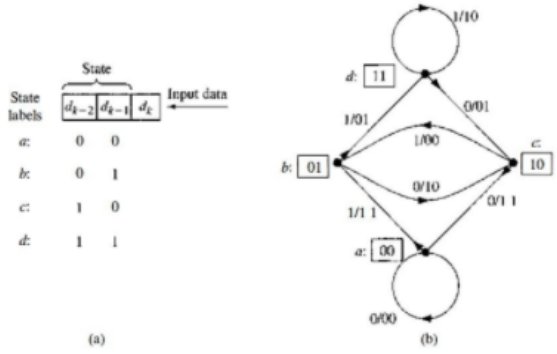

Figure 8.2 State and State Transition Diagram of 1/2 Convolutional Encoder

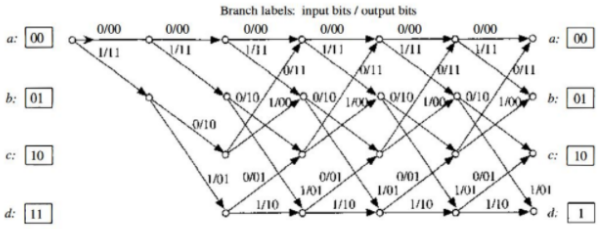

Figure 3: Trellis diagram for the encoder in Fig. 1

Figure 8.3 Trellis Diagram for 1/2 Convolutional Encoder

8.3.2.2 1/3 Convolutional Encoder :

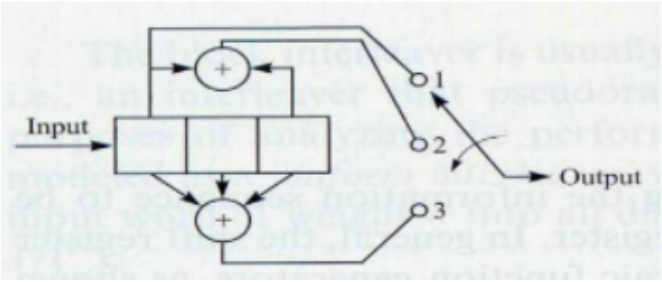

Figure 8.4 1/3 Convolutional Encoder

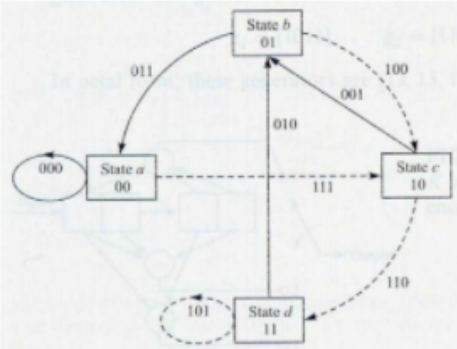

Figure 8.5 State Transition Diagram of 1/3 Convolutional Encoder

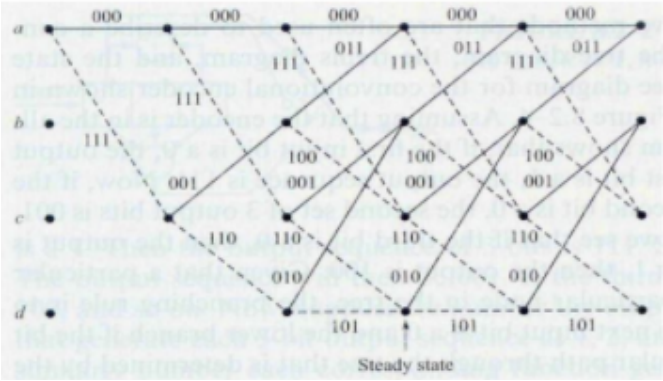

Figure 8.6 Trellis Diagram for 1/3 Convolutional Encoder

8.3.3 About Convolutional Decoding :

Several algorithms exist for decoding convolutional codes. For relatively small values of k, the **Viterbi algorithm** is universally used as it provides maximum likelihood performance and is **highly parallelizable**. Viterbi decoders are thus easy to implement in **VLSI hardware** and in software on CPUs with **SIMD instruction sets**.

Longer constraint length codes are more practically decoded with any of several sequential decoding algorithms, of which the **Fano algorithm** is the best known. Such codes were used in the Pioneer program of the early 1970s to Jupiter and Saturn, but gave way to shorter, Viterbi-decoded codes, usually concatenated with large **Reed–Solomon error correction codes** that steepen the overall bit-error-rate curve and produce extremely **low residual undetected error rates**.

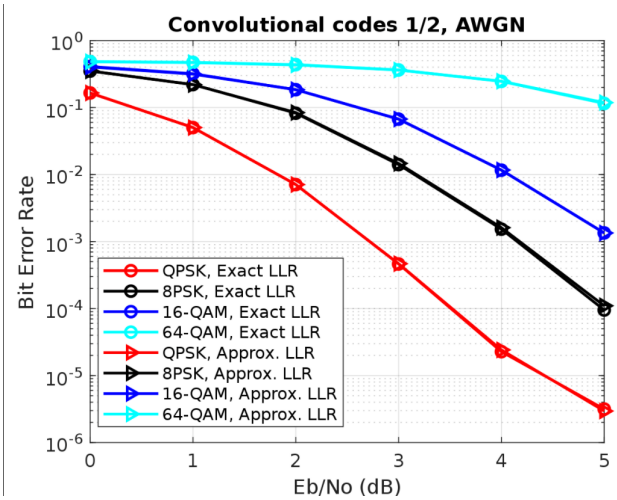

Figure 8.7 Bit error ratio curves for convolutional codes with different options of digital modulations

8.4 Code and Results

8.4.1 Generating Convolutional Encoded symbols through 2 methods :

```
% 19ucc023
% Mohit Akhouri
% Observation 1 - Generating Convolutional Encoded symbols ( via both
 INBUILT
% FUNCTION and MATLAB CODING )

% In this code, we will generate convolutionally encoded codewords
 using
% inbuilt function poly2trellis and also via MATLAB coding with help
 of
% arrays and loops.

clc;
clear all;
close all;

n = 5; % Number of random numbers to be generated
data = randi([0,1],1,n); % Generates a random sequence of 0's and 1's
 of length 1xn

% Display of random sequence of binary digits 0's and 1's generated
disp('The Random data generated is : ');
disp(data);

% Using Inbuilt function poly2trellis to generate convolutional codes
codes_trellis = poly2trellis(3,[7 5]); % Generating Trellis structure
 for convolutional code
codeword_inbuilt = convenc(data,codes_trellis); % Generating
 convolutional codeword with the help of trellis structure

% Display of convolutional codeword generated via INBUILT FUNCTION
% poly2trellis and convenc
disp('The Convolutional Codeword generated via INBUILT FUNCTION is :
 ');
disp(codeword_inbuilt);

% Using MATLAB coding to generate convolutional codes
codeword_matlab_coding = zeros(1,2*n); % To store the codeword
 generated
curr_data = zeros(1,3); % Temporary array to store the binary digits 0
 and 1

% Main Loop algorithm for the calculation of convolutional codewords
for i = 1:n
    curr_data(2:3) = curr_data(1:2); % Replacing bits 2 and 3 with
 bits 1 and 2
    curr_data(1) = data(i); % Starting point of codeword

    % Calculation of remaining convolutional codewords with the help
 of
    % 'mod' function
```

Figure 8.8 Part 1 of the Code for Generation of Convolutional Encoded symbols

```
    codeword_matlab_coding(2*i-1) = mod(curr_data(1) + curr_data(2) +
 curr_data(3) ,2);
    codeword_matlab_coding(2*i) = mod(curr_data(1) + curr_data(3) ,2);
end

% Display of convolutional codewords generated via MATLAB coding
disp('The Convolutional Codeword generated via MATLAB CODING is : ');
disp(codeword_matlab_coding);
```

Published with MATLAB® R2020b

Figure 8.9 Part 2 of the Code for Generation of Convolutional Encoded symbols

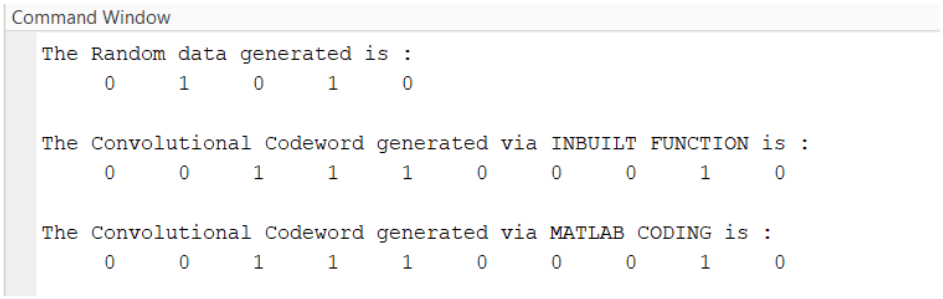

Figure 8.10 Display of **(a)** Random Code Sequence **(b)** Encoded Codeword using INBUILT function **(c)** Encoded Codeword using MATLAB coding

8.4.2 Performance analysis of convolutional encoded systems for BPSK Modulation :

```
% 19ucc023
% Mohit Akhouri
% Observation 2 - Performance analysis of convolutional encoded system

% In this code , we will do the performance analysis of convolutional
% encoded system and we plot the BER vs. SNR(dB)

clc;
clear all;
close all;

n = 1000000; % Size of random sequence of 0's and 1's

rate_1 = 1/2; % Rate for 1/2-Convolutional Encoder
rate_2 = 1/3; % Rate for 1/3-Convolutional Encoder

data = randi([0,1],1,n); % Generating Random Sequence of 0's and 1's

trellis_1_by_2 = poly2trellis(3,[7 5]);  % Trellis Structure of 1/2
 Convolutional Encoder
trellis_1_by_3 = poly2trellis(3,[4 5 7]); % Trellis Structure of 1/3
 Convolutional Encoder

codeword_1_by_2 = convenc(data,trellis_1_by_2); % Codeword generated
 for 1/2 Convolutional Encoder
codeword_1_by_3 = convenc(data,trellis_1_by_3); % Codeword generated
 for 1/3 Convolutional Encoder

max_SNR = 10; % Range for Maximum SNR value
BER_coded_using_1_by_2 = zeros(1,max_SNR); % BER in case of 1/2
 Convolutional Encoder
BER_coded_using_1_by_3 = zeros(1,max_SNR); % BER in case of 1/3
 Convolutional Encoder

BER_uncoded = zeros(1,max_SNR); % BER in case of uncoded system
SNR_dB = zeros(1,max_SNR); % Array to store the SNR values ( in dB )

% Main loop algorithm for calculation of BPSK Modulated waveform and
 Bit
% error rates for two types of convolutional encoders
for i = 1:max_SNR
    SNR_dB(i) = i; % SNR value (without any units)
    SNR = 10^(i/10); % SNR value (in dB)

    % Calculation of sigma factor for calculation of Noise
    Sigma_1 = sqrt(1/(2*rate_1*SNR));
    Sigma_2 = sqrt(1/(2*rate_2*SNR));
    % Calculation of Noise in case of BPSK Modulation
    Noise_1 = Sigma_1*randn(1,n/rate_1);
    Noise_2 = Sigma_2*randn(1,n/rate_2);

    % Computing Transmitted and Received codewords in case of BPSK
```

Figure 8.11 Part 1 of the Code for Performance analysis of convolutional encoded systems

```
    % Modulation for two types of encoder - 1/2 convolutional encoder
 and
    % 1/3 convolutional encoder
    transmitted_1 = 2*codeword_1_by_2 -1;
    received_1 = transmitted_1 + Noise_1;

    transmitted_2 = 2*codeword_1_by_3 -1;
    received_2 = transmitted_2 + Noise_2;

    % Detection of received codeword via HARD DECISION DECODER

    recovered_codeword_1_by_2 = (received_1>0); % Received codeword
 from 1/2 convolutional encoder
    recovered_codeword_1_by_3 = (received_2>0); % Received codeword
 from 1/3 convolutional encoder

    % Using 'vitdec' function to convolutionally decode binary data
    recovered_data_1_by_2 =
 vitdec(recovered_codeword_1_by_2,trellis_1_by_2,20,'trunc','hard');
    recovered_data_1_by_3 =
 vitdec(recovered_codeword_1_by_3,trellis_1_by_3,20,'trunc','hard');

    % Calculation of Net error rate in case of two types of encoders
    NER_hard_1_by_2 = sum(data~=recovered_data_1_by_2);
    NER_hard_1_by_3 = sum(data~=recovered_data_1_by_3);

    % Calculation of Bit error rates ( BER ) for both the encoders
    BER_coded_using_1_by_2(i) = NER_hard_1_by_2/n;
    BER_coded_using_1_by_3(i) = NER_hard_1_by_3/n;

    % Calculation of Bit error rate ( BER ) in case of uncoded system
    BER_uncoded(i) = qfunc(sqrt(SNR));
end

% Plots of BER vs. SNR (dB) for both types of encoders
figure;
semilogy(SNR_dB,BER_coded_using_1_by_2);
hold on;
semilogy(SNR_dB,BER_coded_using_1_by_3);
hold on;
semilogy(SNR_dB,BER_uncoded);
xlabel('SNR (dB) ->');
ylabel('BER ->');
title('19ucc023 - Mohit Akhouri','Plots of the BER vs. SNR(dB) for 1/2
 Convolutional Encoder and 1/3 Convolutional Encoder');
legend('BER in case of 1/2 Convolutional Encoder','BER in case of 1/3
 Convolutional Encoder','BER in case of uncoded system');
grid on;
hold off;
```

Published with MATLAB® R2020b

Figure 8.12 Part 2 of the Code for Performance analysis of convolutional encoded systems

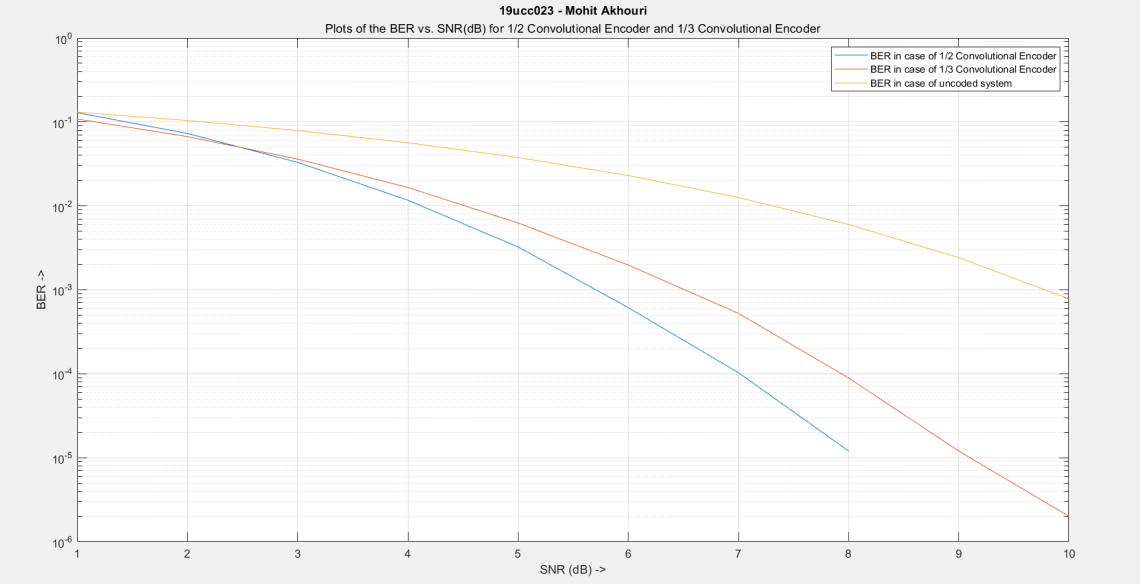

Figure 8.13 Plots of BER vs. SNR(dB) for Convolutionally encoded systems and uncoded system

8.5 Conclusion

In this experiment , We learnt about the **Convolutional Encoder** and **Convolutional Decoder**. We learnt about various concepts like **Trellis Diagram** and **Viterbi Decoding**. We implemented two types of convolutional encoder - for rates 1/2 and 1/3. We used the MATLAB inbuilt function - **poly2trellis** and designed MATLAB code for the construction of convolutional encoder. We performed the BPSK modulation and decoded the codewords using MATLAB function **vitdec**. We calculated values of BER for different values of SNR (dB) and plotted the graph between BER and SNR. We observed the graph for three types of cases - uncoded system , encoded system for 1/2 convolutional encoder and encoded system for 1/3 convolutional encoder. We learnt about many new MATLAB functions like - **poly2trellis**, **vitdec** and **mod**.