SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM NAGAR, KATTANKULATHUR-603 203.

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

LABORATORY MANUAL

1906508 – COMMUNICATION SYSTEMS AND NETWORKS LABORATORY

Regulation - 2019

Semester/Branch	**: V semester ECE**
Academic Year	**: 2022-23 (ODD)**
Prepared By	**: Mr. S. Mari Rajan**, *Assistant Professor* (Sr.G) **Mr. D. Murugesan,** *Assistant Professor* (Sel.G) **Mr. J. Charles Vinoth,** *Assistant Professor* (O.G)

SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)
SRM Nagar, Kattankulathur – 603 203

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

VISION OF THE INSTITUTE

Educate to excel in social transformation

MISSION OF THE INSTITUTE

• To contribute to the development of human resources in the form of professional engineers and managers of international excellence and competence with high motivation and dynamism, who besides serving as ideal citizen of our country will contribute substantially to the economic development and advancement in their chosen areas of specialization.

• To build the institution with international repute in education in several areas at several levels with specific emphasis to promote higher education and research through strong institute-industry interaction and consultancy.

VISION OF THE DEPARTMENT

To excel in the field of electronics and communication engineering and to develop highly competent technocrats with global intellectual qualities.

MISSION OF THE DEPARTMENT

M1: To educate the students with the state of art technologies to compete internationally, able to produce creative solutions to the society`s needs, conscious to the universal moral values, adherent to the professional ethical code.

M2: To encourage the students for professional and software development career.

M3: To equip the students with strong foundations to enable them for continuing education and research.

SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)
SRM Nagar, Kattankulathur – 603 203

PROGRAM OUTCOMES
1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOME (PSOs)
PSO1: Ability to apply the acquired knowledge of basic skills, mathematical foundations, principles of electronics, modelling and design of electronics based systems in solving engineering Problems.
PSO2: Ability to understand and analyze the interdisciplinary problems for developing innovative sustained solutions with environmental concerns.
PSO3: Ability to update knowledge continuously in the tools like MATLAB, NS2, XILINIX and technologies like VLSI, Embedded, Wireless Communications to meet the industry requirements. **PSO4**: Ability to manage effectively as part of a team with professional behavior and ethics.

SYLLABUS

		L	T	P	C
1906508	**COMMUNICATION SYSTEMS AND NETWORKS LABORATORY**	**0**	**0**	**4**	**2**

OBJECTIVES:

The student should be able to:

- ✓ Implement the Analog and Digital Modulation schemes.
- ✓ Simulate Digital Modulation schemes and Error Control Techniques.
- ✓ Implement the different protocols in communication network.
- ✓ Familiarize with IP Configuration in a network.
- ✓ Analyze the various routing algorithms.

LIST OF EXPERIMENTS:

1. AM Modulator and Demodulator, FM Modulator and Demodulator.
2. Pulse Code Modulation and Demodulation.
3. Delta Modulation and Demodulation.
4. Line coding schemes.
5. Simulation of ASK, FSK, BPSK, DPSK, QPSK and QAM generation schemes.
6. Simulation of Error control techniques.
7. Implementation of Error Detection / Error Correction Techniques.
8. Implementation of Flow control Algorithms.
9. Implementation of IP address configuration and Commands such as ping, Trace route, ns lookup.
10. To create scenario and study the performance of network with CSMA / CA protocol. & compare with CSMA/CD protocols.
11. Network Topology - Star, Bus, Ring.
12. Implementation of distance vector and Link state routing algorithm.

TOTAL PERIODS: 60

OUTCOMES:

On completion of this laboratory course, the student would be able to,

- ✓ Simulate & validate the various functional modules of a communication system.
- ✓ Apply various Error coding schemes & demonstrate their capabilities towards the improvement of the noise performance of communication system
- ✓ Implement the different networking protocols.
- ✓ Configure a network using socket programming.
- ✓ Implement the various routing algorithms.

LIST OF EXPERIMENTS

1. Amplitude Modulation and Demodulation
2. Frequency Modulation and Demodulation
3. Pulse Code Modulation and Demodulation
4. Delta Modulation and Demodulation
5. Line coding schemes
6. Simulation of ASK, FSK, BPSK, DPSK, QPSK and QAM generation schemes.
7. Simulation of Error control techniques.
8. Implementation of Error Detection / Error Correction Techniques.
9. Implementation of Flow control Algorithms.
10. Implementation of IP address configuration and Commands such as ping, Trace route, ns lookup.
11. To create scenario and study the performance of network with CSMA / CA protocol & compare with CSMA/CD protocols.
12. Network Topology - Star, Bus, Ring.
13. Implementation of distance vector and Link state routing algorithm.

ADDITIONAL EXPERIMENTS

1. Signal Sampling and reconstruction
2. Time Division Multiplexing

Ex. No:1

AMPLITUDE MODULATION AND DEMODULATION

AIM:

To study and understand amplitude modulation & demodulation circuit and calculate modulation index for various modulating voltages.

APPARATUS REQUIRED:

1. AM transmitter trainer kit
2. AM receiver trainer kit
3. CRO
4. Patch chords

PRE LAB EXERCISE:

1. What is meant by modulation?
2. Explain Amplitude modulation and Demodulation?
3. Define modulation index.

THEORY:

Amplitude Modulation is a process by which amplitude of the carrier signal is varied in accordance with the instantaneous value of the modulating signal, but frequency and phase of carrier wave remains constant.

The modulating and carrier signal are given by

$$V_m(t) = V_m \sin \omega_m t$$

$$V_C(t) = V_C \sin \omega_C t$$

The modulation index is given by, $m_a = V_m / V_C$.

$$V_m = V_{max} - V_{min} \text{ and } V_C = V_{max} + V_{min}$$

The amplitude of the modulated signal is given by,

$$V_{AM}(t) = V_C (1+m_a \sin \omega_m t) \sin \omega_C t$$

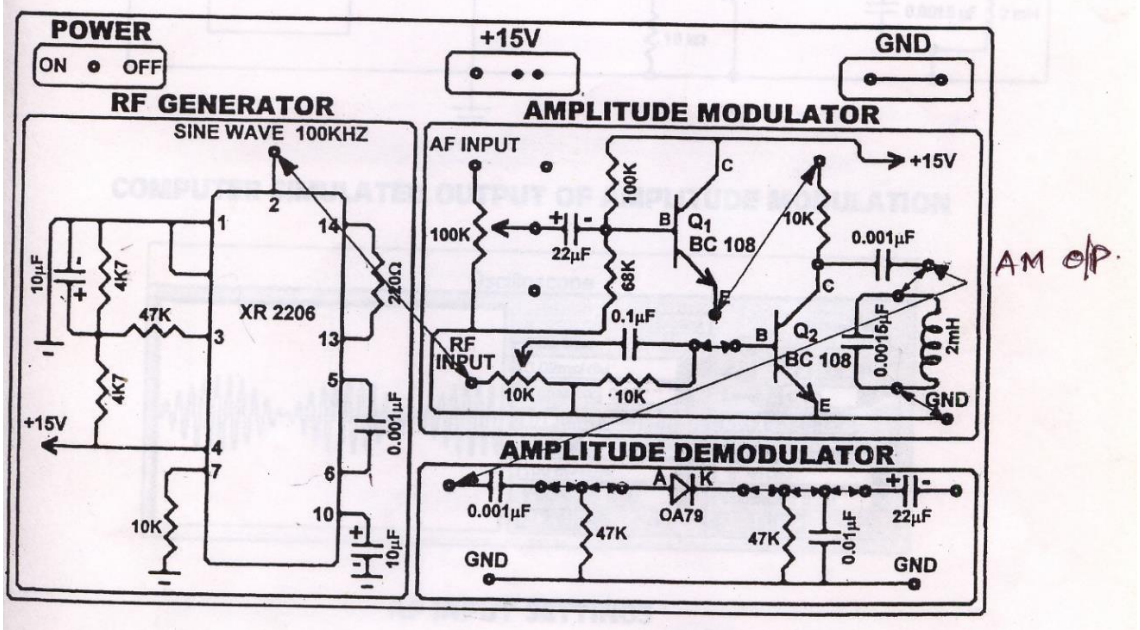
Where

V_m = maximum amplitude of modulating signal
V_C = maximum amplitude of carrier signal

V_{max} = maximum variation of AM signal

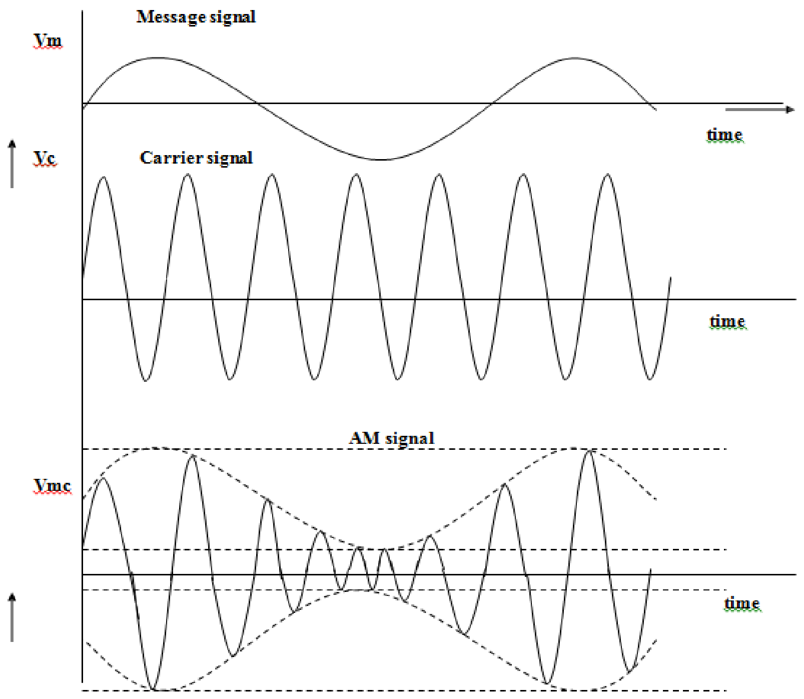
V_{min} = minimum variation of AM signal

AMPLITUDE MODULATION & DEMODULATION



INDICATES THE PATCHING CONNECTIONS

MODEL GRAPH



TABULAR COLUMN:

V_m(p– p) (Volts)	E_{MAX} (Volts)	E_{Min} (Volts)	Practical $M = \frac{E\,max - E\,min}{E\,max + E\,min}$ %	Theoretical $M = \frac{A\,max - A\,min}{A\,max + A\,min}$ %

PROCEDURE:

1. The circuit wiring is done as shown in diagram
2. A modulating signal input given to the Amplitude modulator
3. Now increase the amplitude of the modulating signal to the required level.
4. The amplitude and the time duration of the modulating signal are observed using CRO.
5. Finally the amplitude modulated output is observed from the output of amplitude modulator stage and the amplitude and time duration of the AM wave are noted down.
6. Calculate the modulation index by using the formula and verify them.
7. The final demodulated signal is viewed using a CRO at the output of audio power amplifier stage. Also the amplitude and time duration of the demodulated wave are noteddown.

POST LAB EXERCISE:

1. What was the value of modulation index observed?
2. As related to AM, what is over modulation, under modulation and 100%modulation?
3. What type of AM signals was generated in the kit?

RESULT:

The amplitude modulation circuit was designed and its modulation index was calculated for various modulating voltages and was compared with theoretical values.

Ex. No: 2

FREQUENCY MODULATION AND DEMODULATION

AIM:

To study the characteristics of FM transmitter and receiver using trainer kit.

APPARATUS REQUIRED:

1. FM transmitter trainer kit
2. FM receiver trainer kit
3. CRO
4. Patch chords

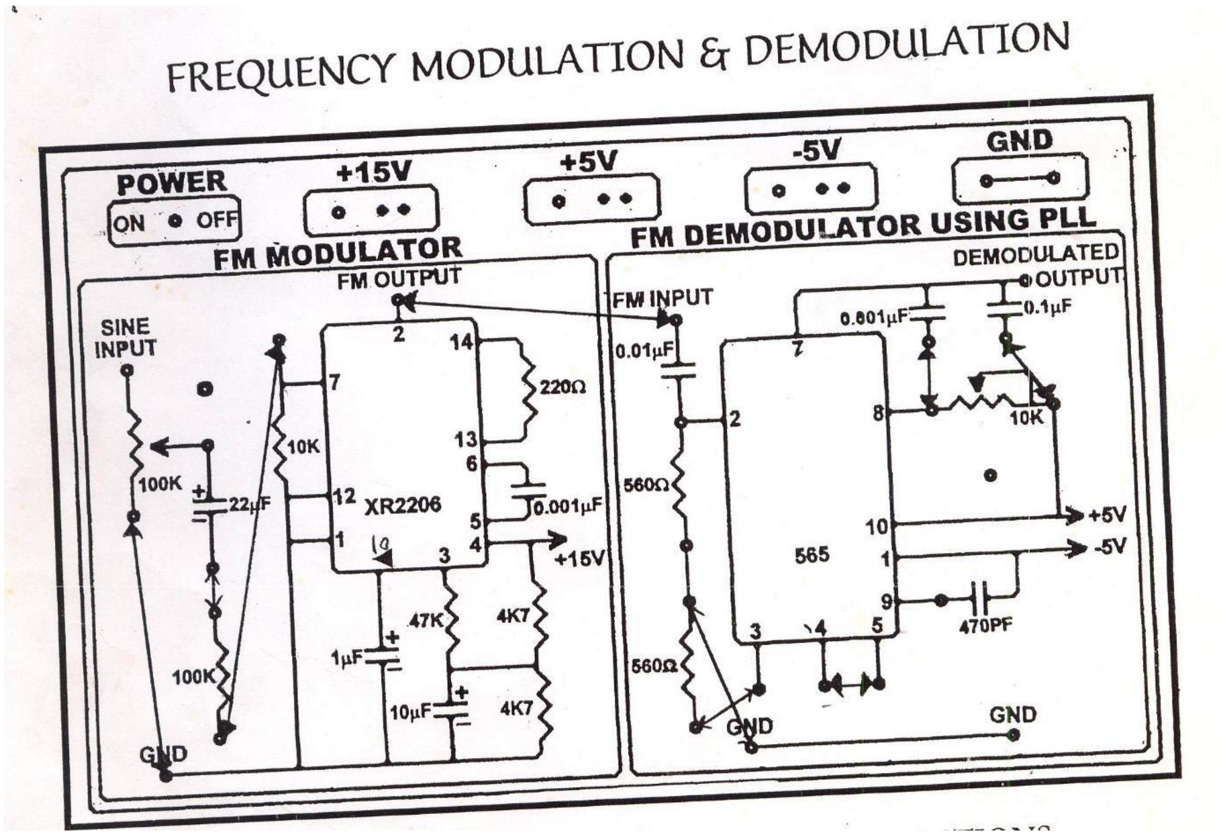
PRE LAB EXERCISE:

1. What is FM?
2. Define frequency deviation in FM.
3. What are the advantages of FM?

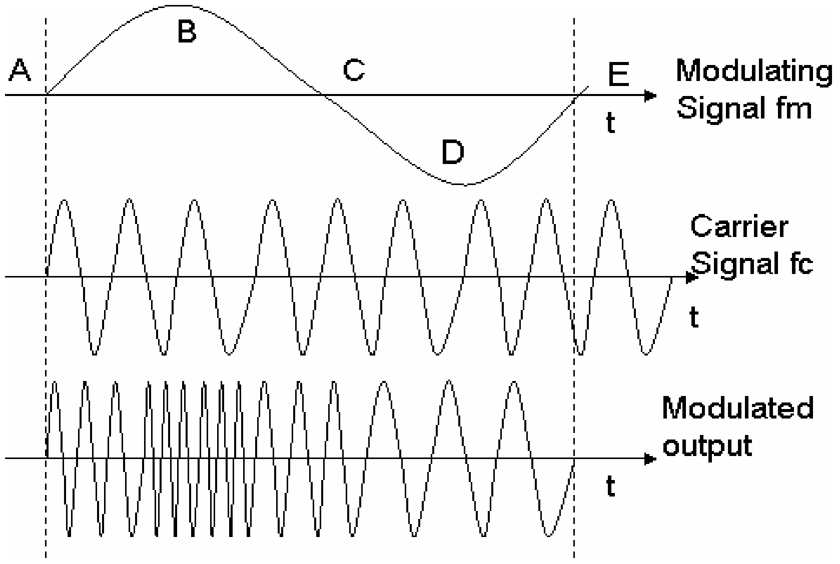
THEORY:

Frequency modulation (FM) is a form of modulation that represents information as variations in the instantaneous frequency of a carrier wave. (Contrast this with amplitude modulation, in whichthe amplitude of the carrier is varied while its frequency remains constant.) In analog applications, the carrier frequency is varied in direct proportion to changes in the amplitude of an input signal. Shifting the carrier frequency among a set of discrete values can represent digital data, a technique known as frequency-shift keying. FM is commonly used at VHF radio frequencies for high-fidelity broadcasts of music and speech (see FM broadcasting). Normal (analog) TV sound is also broadcast using FM. A narrowband form is used for voice communications in commercial and amateur radio settings. The type of FM used in broadcast is generally called wide-FM, or W-FM. In two-way radio,narrowband narrow-fm (N-FM) is used to conserve bandwidth. In addition, it is used to send signals into space.

FM is also used at intermediate frequencies by most analog VCR systems, including VHS, to record the luminance (black and white) portion of the video signal. FM is the only feasible method of recording video to and retrieving video from magnetic tape without extreme distortion, as video signals have a very large range of frequency components — from a few hertz to several megahertz, toowide for equalizers to work with due to electronic noise below -60 dB. FM also keeps the tape at saturation level, and therefore acts as a form of noise reduction, and a simple limiter can mask variations in the playback output, and the FM capture effect removes print-through and pre-echo. A continuous pilot-tone, if added to the signal — as was done on V2000 and many Hi-band formats
— can keep mechanical jitter under control and assist time base correction.


FREQUENCY MODULATION & DEMODULATION
POWER
ON OFF
+15V
+5V
-5V
GND
FM MODULATOR
FM DEMODULATOR USING PLL
FM OUTPUT
SINE INPUT
10K
100K
22μF
XR2206
220Ω
0.001μF
+15V
47K
4K7
1μF
10μF
100K
GND
FM INPUT
0.01μF
560Ω
560Ω
565
0.001μF
0.1μF
DEMODULATED OUTPUT
10K
+5V
-5V
470PF
GND
GND

MODEL GRAPH:


A
B
C
D
E
t
Modulating Signal fm
Carrier Signal fc
t
Modulated output
t

TABULAR COLUMN:

SIGNAL	**AMPLITUDE(V)**	**TIME PERIOD(ms)**
INPUT SIGNAL		
CARRIER SIGNAL		
FM SIGNAL		
DEMODULATED SIGNAL		

PROCEDURE:

1. The circuit wiring is done as shown in diagram
2. A modulating signal input given to the Frequency modulator
3. Now increase the modulated signal to the required level.
4. The amplitude and the time duration of the modulating signal are observed using CRO.
5. Finally the frequency modulated output is observed from the output of frequency modulator stage and the amplitude and time duration of the FM wave arenoted down.

POST LAB EXERCISE:

1. What type of FM generation was done in the kit?
2. What do you mean by narrow band and wideband FM?
3. Was the demodulated output same as the message signal?

RESULT:

Thus the FM signal was transmitted using FM trainer kit and the FM signal detected using FM detector kit.

Ex. No: 3

PULSE CODE MODULATION AND DEMODULATION

AIM:

To perform pulse code modulation and demodulation and to plot the waveforms for Binary data at different frequencies.

APPARATUS REQUIRED:

1. FM transmitter trainer kit
2. FM receiver trainer kit
3. CRO
4. Patch chords

PRE LAB EXERCISE:

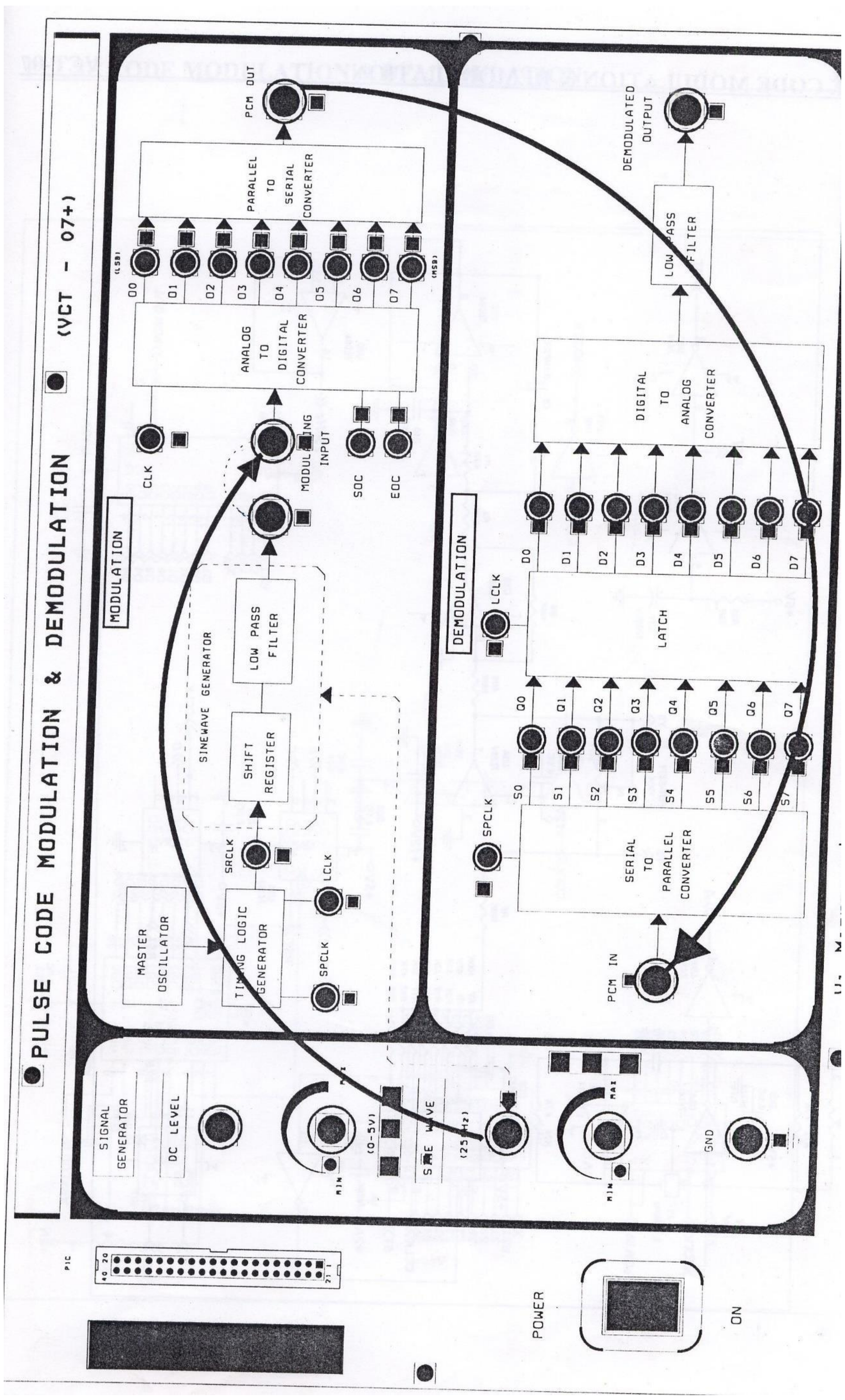
1. What is PCM?
2. List the advantages of PCM.
3. Summarize the application of PCM.

THEORY

Pulse code modulation is a process of converting an analog signal into digital. The voice or any data input is first sampled using a sampler (which is a simple switch) and then quantized. Quantization is the process of converting a given signal amplitude to an equivalent binary number with fixed number of bits. This quantization can be either mid- tread or mid-raise and it can be uniform or non-uniform based on the requirements. For example in speech signals, the higher amplitudes will be less frequent than the low amplitudes. So higher amplitudes are given less step size than the lower amplitudes and thus quantization is performed non-uniformly. After quantization the signal is digital and the bits are passed through a parallel to serial converter and then launched into the channel serially.

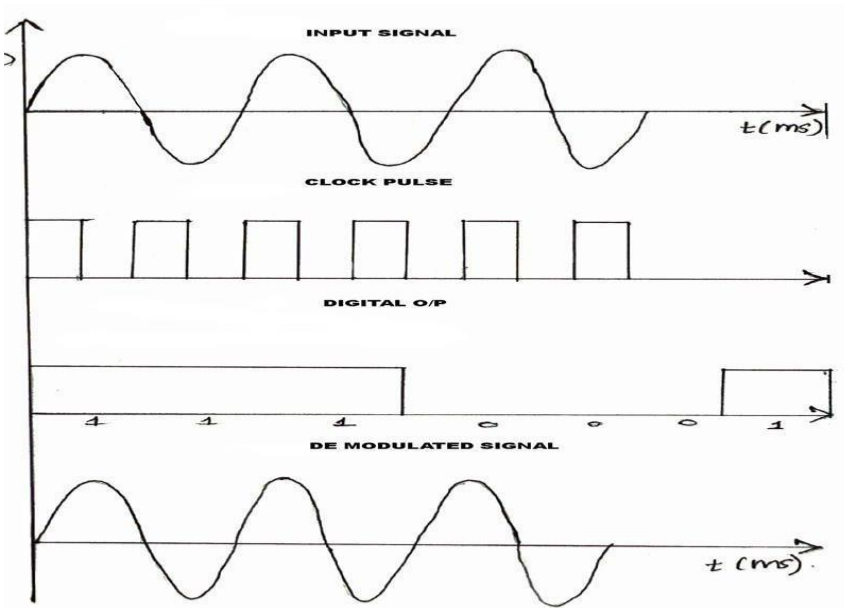
At the demodulator the received bits are first converted into parallel frames and each frame is de-quantized to an equivalent analog value. This analog value is thus equivalent to a sampler output. This is the demodulated signal.

In the kit this is implemented differently. The analog signal is passed through a ADC (Analog to Digital Converter) and then the digital code word is passed through a parallel to serial converter block. This is modulated PCM. This is taken by the Serial to Parallel converter and then through a DAC to get the demodulated signal. The clock is given to all these blocks for synchronization. The input signal can be either DC or AC according to the kit. The waveforms can be observed on a CRO for DC without problem. AC also can be observed but with poor resolution.



PULSE CODE MODULATION & DEMODULATION (VCT - 07+)
MODULATION
MASTER OSCILLATOR
TIMING LOGIC GENERATOR
SRCLK
LCLK
SPCLK
SINEWAVE GENERATOR
SHIFT REGISTER
LOW PASS FILTER
CLK
MODULATING INPUT
SOC
EOC
ANALOG TO DIGITAL CONVERTER
D0 (LSB)
D1
D2
D3
D4
D5
D6
D7 (MSB)
PARALLEL TO SERIAL CONVERTER
PCM OUT
DEMODULATION
PCM IN
SPCLK
SERIAL TO PARALLEL CONVERTER
S0
S1
S2
S3
S4
S5
S6
S7
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
LCLK
LATCH
D0
D1
D2
D3
D4
D5
D6
D7
DIGITAL TO ANALOG CONVERTER
LOW PASS FILTER
DEMODULATED OUTPUT
SIGNAL GENERATOR
DC LEVEL
MIN
MAX
(0-5V)
SINE WAVE
(250Hz)
MIN
MAX
GND
POWER
ON

MODEL GRAPH:



TABULAR COLUMN

S.No	Name of the signal	Amplitude in V	Time period in Sec	Frequency in Hz
1	Modulating Signal			
2	Carrier Signal			
3	Modulated Signal			
4	Demodulated Signal			

PROCEDURE:

1. Power on the PCM kit.
2. Measure the frequency of sampling clock.
3. Apply the DC voltage as modulating signal.
4. Connect the DC input to the ADC and measure the voltage.
5. Connect the clock to the timing and control circuit.
6. Note the binary work from LED display. The serial data through the channel can be observed in the CRO.
7. Also observe the binary word at the receiver end.
8. Now apply the AC modulating signal at the input.
9. Observe the waveform at the output of DAC.
10. Note the amplitude of the input voltage and the code word. Also note the value of theoutput voltage. Show the code word graphically for a DC input.

POST LAB EXERCISE:

1. What is the expression for transmission bandwidth in a PCM system?
2. What is the expression for quantization noise/error in PCM system?
3. What are the disadvantages of PCM?

RESULT:

Thus the PCM signal was generated using PCM modulator and the message signal wasdetected from PCM signal by using PCM demodulator and graphs were plotted.

Ex. No:4

DELTA MODULATION AND DEMODULATION

AIM:

To transmit an analog message signal in its digital form and again reconstruct back the original analog message signal at receiver by using Delta modulator.

APPARATUS REQUIRED

1. Delta Modulator kit
2. CRO and
3. Connecting probes

PRE LAB EXERCISE:

1. Explain DM.
2. What is the advantage of DM?
3. Differentiate PCM and DM.

THEORY:

Delta Modulation is a form of pulse modulation where a sample value is represented as a single bit. This is almost similar to differential PCM, as the transmitted bit is only one per sample just to indicate whether the present sample is larger or smaller than the previous one. The comparison of samples is accomplished by converting the digital to analog form and then comparing with the present sample. This is done using an up counter and DAC as shown in block diagram. The delta modulated signal is given to up counter and then a DAC and the analog input is given to OPAMP and a LPF to obtain the demodulated output.

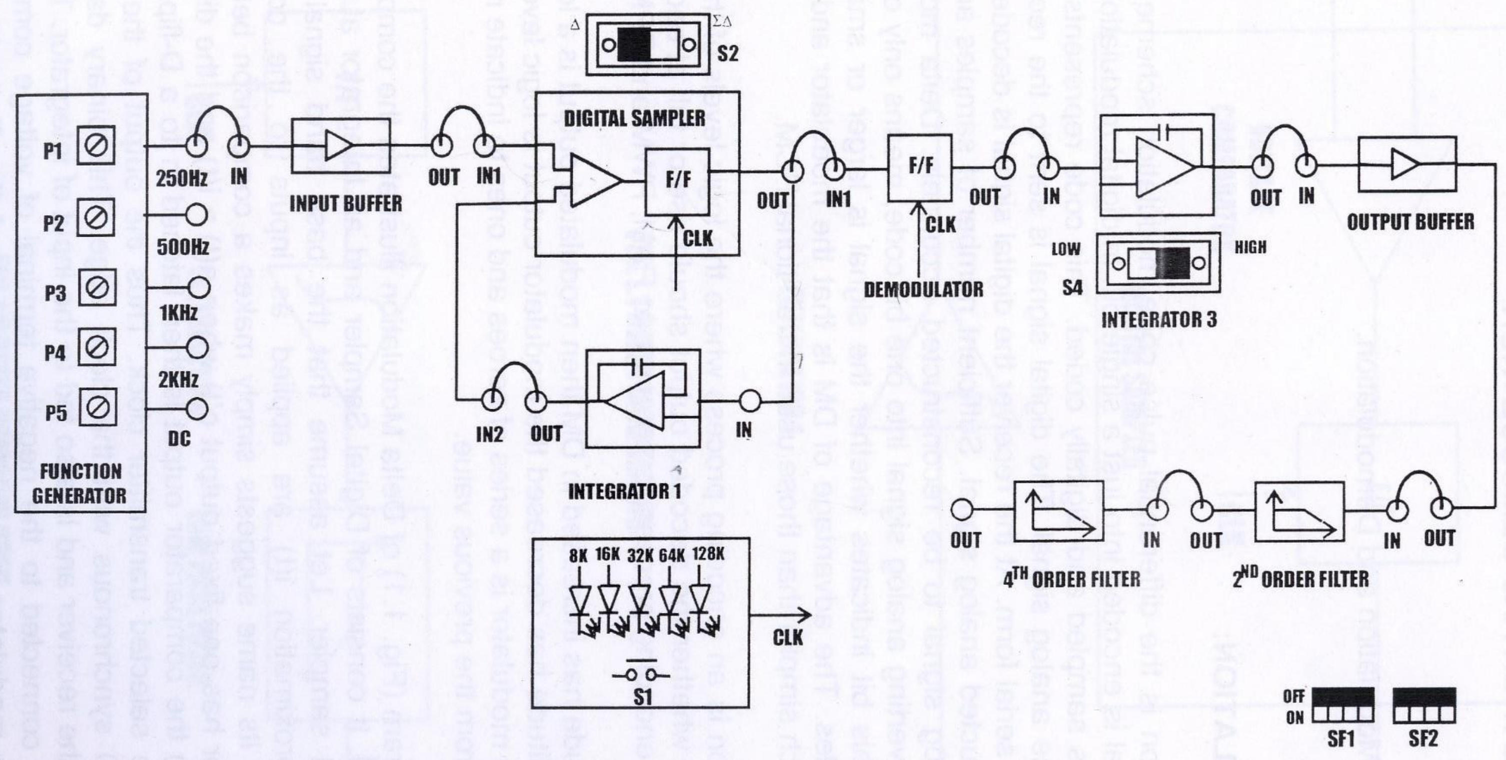


FIG.1.3 BLOCK DIAGRAM FOR DELTA MODULATION AND DEMODULATION

MODEL GRAPH

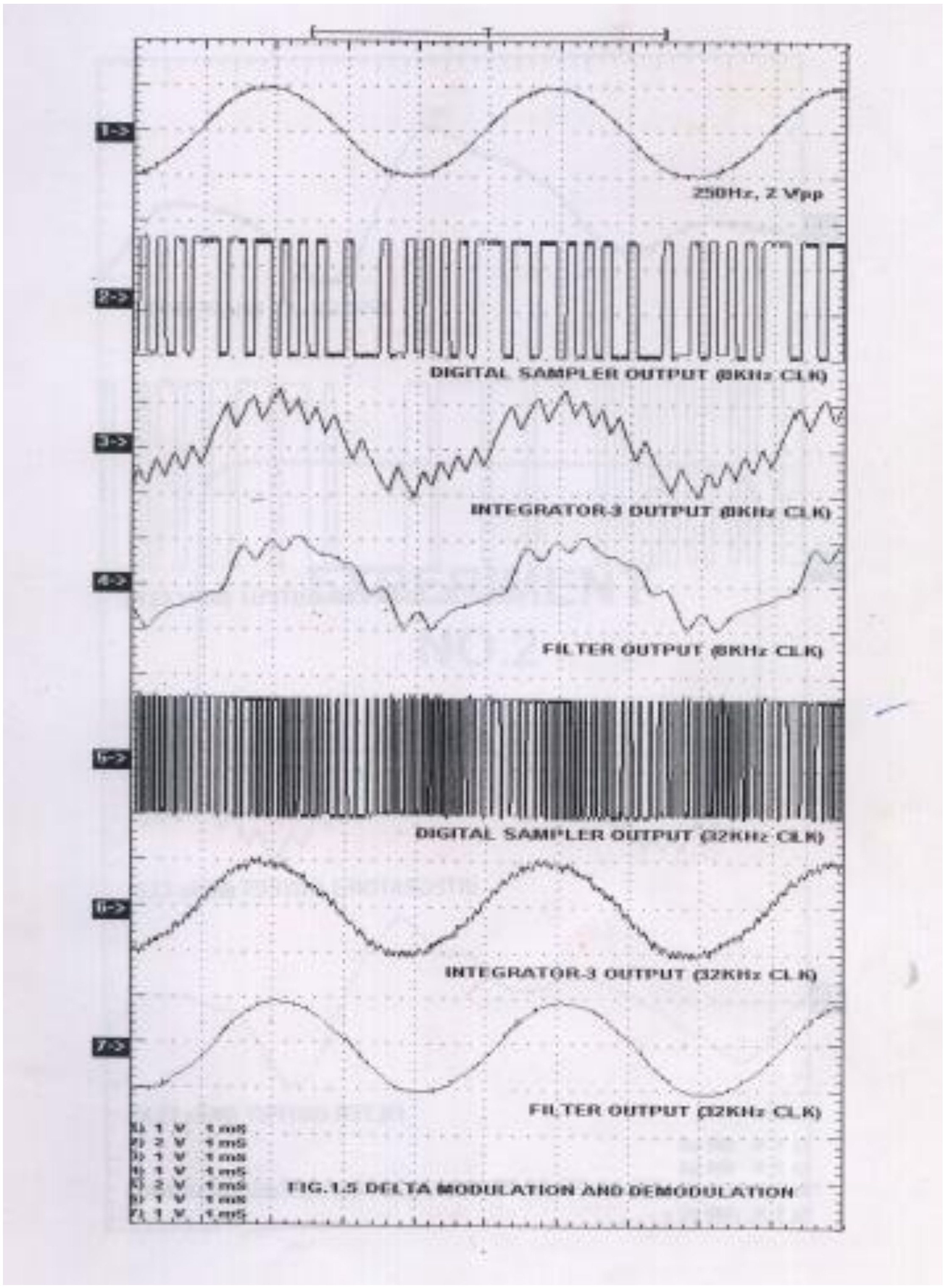


FIG.1.5 DELTA MODULATION AND DEMODULATION

TABULAR COLUMN

SIGNAL	AMPLITUDE (Volts)	TIME PERIOD (sec)
INPUT INTEGRATEDOUTPUT DELTA MODULATED WAVE DELTA DEMODULATED WAVE		

PROCEDURE

1. Switch on the kit. Connect the clock signal and the modulating input signal to the modulator block. Observe the modulated signal in the CRO.
2. Connect the DM output to the demodulator circuit. Observe the demodulator output on the CRO.
3. Also observe the DAC output on the CRO.
4. Change the amplitude of the modulating signal and observe the DAC output. Notice the slope overload distortion. Keep the tuning knob so that the distortion is gone. Note this value of the amplitude. This is the minimum required value of the amplitude to overcome slope overload distortion.
5. Calculate the sampling frequency required for no slope overload distortion.
6. Compare the calculated and measured values of the sampling frequency.

POST LAB EXERCISE:

1. How to overcome slope overload distortion?
2. What is the other name of Granular noise?
3. What is meant by staircase approximation?

RESULT:

Thus the analog message signal in its digital form was transmitted and again the originalanalog message signal was reconstructed at receiver by using Delta modulator and Demodulator

Ex. No:5

LINE CODING TECHNIQUES

AIM:

To study different line coding techniques and its characteristics.

APPARATUS REQUIRED:

1. Communication trainer kit
2. Multi Output Power Supply.
3. Patch chords.
4. DSO/CRO

PRE LAB EXERCISE:

1. What is the purpose of line codingschemes?
2. State NRZ unipolar format
3. Explain Manchester format.

THEORY:

We need to represent PCM binary digits by electrical pulses in order to transmit them through a base band channel. The most commonly used PCM popular data formats are being realized here. Line coding refers to the process of representing the bit stream (**1**‟ s and **0**‟ s) in the form of voltage or current variations optimally tuned for the specific propertiesof the physical channel being used. The selection of a proper line code can help in so many ways: One possibility is to aid in clock recovery at the receiver. A clock signal is recovered by observing transitions in the received bit sequence, and if enough transitions exist, a good recovery of the clock is guaranteed, and the signal is said to be self-clocking.

Another advantage is to get rid of DC shifts. The DC component in a line code is called the *bias* or the *DC coefficient*. Unfortunately, most long-distance communication channels cannot transport a DC component. This is why most line codes try to eliminate the DC component before being transmitted on the channel. Such codes are called *DC balanced*, *zero-DC*, *zero-bias*, or *DC equalized*. Some common types of line encoding in common-use nowadays are unipolar, polar, bipolar, Manchester, MLT-3 and Duo binary encoding. These codes are explained here:

1. Unipolar (Unipolar **NRZ** and Unipolar **RZ**):

Unipolar is the simplest line coding scheme possible. It has the advantage of being compatible with TTL logic. Unipolar coding uses a positive rectangular pulse *p*(*t*) to represent binary **1**, and the absence of a pulse (i.e., zero voltage) to represent a binary **0**. Two possibilities for the pulse *p*(*t*) exist3: Non-Return-to-Zero (NRZ) rectangular pulse and Return-to-Zero (RZ) rectangular pulse. The difference between Unipolar NRZ and Unipolar RZ codes is that the rectangular pulse in NRZ stays at a positive value (e.g., +5V) for the full duration of the logic **1** bit, while the pule in RZ drops from +5V to 0V in the middle of the bittime.

A drawback of unipolar (RZ and NRZ) is that its average value is not zero, which means it creates a significant DC-component at the receiver (see the impulse at zero frequency in the corresponding power spectral density (PSD) of this line code. The disadvantage of unipolar RZ compared to unipolar NRZ is that each rectangular pulse in RZ is only half the length of NRZ pulse. This means that unipolar RZ requires twice the bandwidth of the NRZ code.

Polar (Polar NRZ and Polar RZ):

In Polar NRZ line coding binary **1**‟ s are represented by a pulse *p*(*t*) and binary **0**‟ s are represented by the negative of this pulse -*p*(*t*) (e.g., -5V). Polar (NRZ and RZ) signals .Using the assumption that in a regular bit stream a logic **0** is just as likely as a logic **1**,polar signals (whether RZ or NRZ) have the advantage that the resulting Dc component is very close to zero.

BIPHASE SIGNALS (PHASE ENCODED SIGNALS):

a) Bi phase – LEVEL (Manchester Coding)
b) Bi phase – MARK and
c) Bi phase – Space Signals

These schemes are used in magnetic recording, optical communications and in satellite telemetry links. This phase encoded signals are special in the sense that they are composed of both the in phase and out-of-phase components of the clock.

Manchester Coding (Biphase –L):

With the Biphase – L, a ‘one’ is represented by a half bit wide pulse positioned during the first half of the bit interval and a ‘zero, is represented by a half bit wide pulse positioned during the second half of the bit interval.

Biphase Mark Coding (Biphase –M):

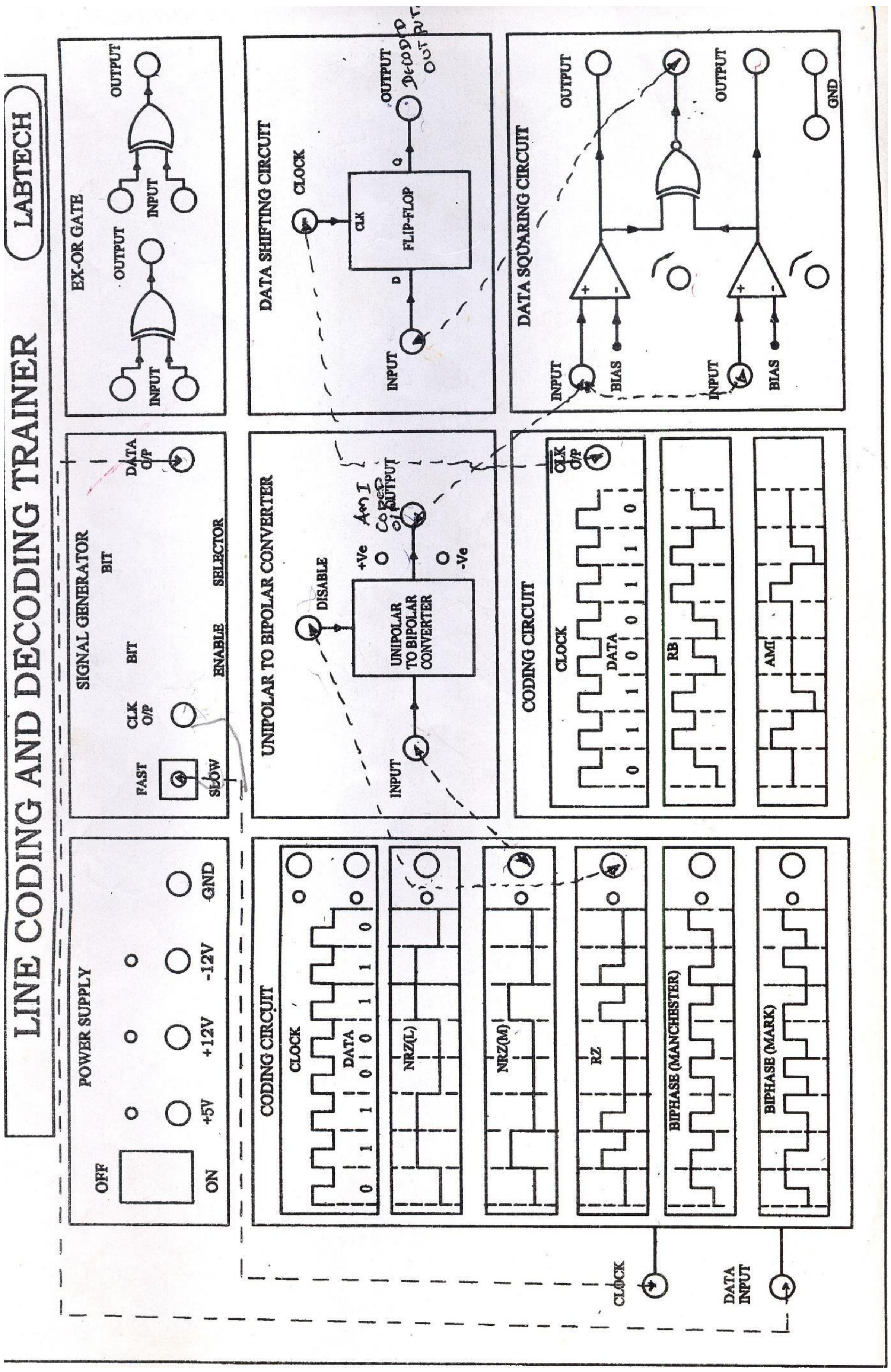
With the Biphase – M, a transition occurs at the beginning of every bit interval. A ‘one’ is represented by a second transition, one half bit later whereas a zero has no second transition.

Bi-Phase Space coding (Biphase –S):

With a Biphase – S also a transition occurs at the beginning of every bit interval. A ‘zero’ is arked by a second transition, one half bit later, where as a ‘one’ has no second transition.

Return to zero – Alternate Mark Inversion Coding (RZ-AMI):

This coding scheme is most often used in telemetry systems. This scheme comes under both the category of return to zero scheme and multilevel scheme. The one’s are represented by pulse width of half the bit duration existing in the alternate direction whereas zero’s are represented by absence of the pulse.


LINE CODING AND DECODING TRAINER
LABTECH
POWER SUPPLY
OFF
ON
+5V
+12V
-12V
GND
SIGNAL GENERATOR
FAST
SLOW
CLK O/P
BIT ENABLE
BIT SELECTOR
DATA O/P
EX-OR GATE
INPUT
OUTPUT
INPUT
OUTPUT
CODING CIRCUIT
CLOCK
DATA
0 1 1 0 0 1 1 0
NRZ(L)
NRZ(M)
RZ
BIPHASE (MANCHESTER)
BIPHASE (MARK)
UNIPOLAR TO BIPOLAR CONVERTER
INPUT
DISABLE
UNIPOLAR TO BIPOLAR CONVERTER
+Ve
-Ve
OUTPUT
AMI coded o/p
DATA SHIFTING CIRCUIT
CLOCK
CLK
D
Q
FLIP-FLOP
INPUT
OUTPUT
DECODED OUTPUT
CODING CIRCUIT
CLK O/P
CLOCK
DATA
0 1 1 0 0 1 1 0
RB
AMI
DATA SQUARING CIRCUIT
INPUT
BIAS
OUTPUT
INPUT
BIAS
OUTPUT
GND
CLOCK
DATA INPUT

<u>OBSERVATIONS</u>

DATA	**:**
AMPLITUDE	**:**
TIME PERIOD	**:**

PROCEDURE

1. Give the connections as per the experimental setup.
2. Observe the clock signal &the data and measure them.
3. Observe the standard data &the coded data formats and verify with the known formats.

POST LAB EXERCISE:

1. Which among the above format is widely used?
2. List the applications of Line coding schemes.
3. Which coding format has mean around zero?

RESULT:

Thus the different line coding techniques was studied and its characteristics are plotted.

Ex. No:6 a

SIMULATION OF ASK, FSK, BPSK GENERATION SCHEMES

AIM:

To obtain the signal constellation of ASK, FSK, BPSK waveforms using MATLAB.

SOFTWARE REQUIRED:

SYSTEM with MATLAB software

PRE-LAB QUESTIONS:

1. What is ASK modulation?
2. Compare the performance of ASK, FSK and BPSK.
3. What is Constellation in the digital modulation techniques?

THEORY:

Any digital modulation scheme uses a finite number of distinct signals to represent digital data. 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 (and referred to as CPSK).Alternatively, instead of using the bit patterns to *set* the phase of the wave, it can instead be used to *change* it by a specified amount. The demodulator then determines the *changes* in the phase of the received signal rather than the phase itself. Since this scheme depends on the difference between successive phases, it is termed **differential phase-shift keying (DPSK)**. DPSK can be significantly simpler to implement than ordinary PSK since there is no need for the demodulator to have a copy of the reference signal to determine the exact phase of the received signal (it is a non- coherent scheme). In exchange, it produces more erroneous demodulations.

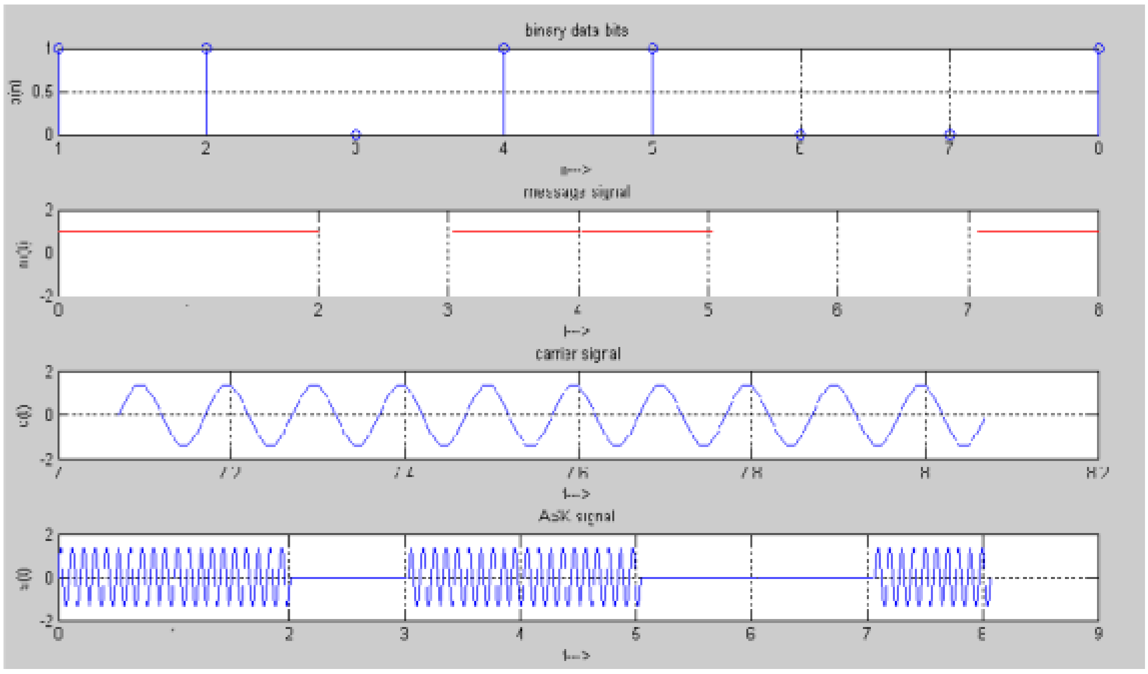
ALGORITHM:

1. Start.
2. Get the data bits and compute its length.
3. Get the input frequency.
4. Generate the BPSK waveforms with their corresponding carrier frequency.
5. Plot the output waveform.
6. Stop

PROGRAM FOR ASK

```
%%%%%       ASK    %%%%%%
  clc;
  clear all;
  close all;
  %GENERATE CARRIER SIGNAL
  Tb=1; fc=10;
  t=0:Tb/100:1;
  c=sqrt(2/Tb)*sin(2*pi*fc*t);
  %generate message signal
  N=8;
  m=rand(1,N);
  t1=0;t2=Tb
  for i=1:N
  t=[t1:.01:t2]
   if m(i)>0.5
   m(i)=1;
   m_s=ones(1,length(t));
   else
   m(i)=0;
   m_s=zeros(1,length(t));
   end
   message(i,:)=m_s;
   %product of carrier and message
   ask_sig(i,:)=c.*m_s;
   t1=t1+(Tb+.01);
   t2=t2+(Tb+.01);
   %plot the message and ASK signal
   subplot(5,1,2);axis([0 N -2 2]);plot(t,message(i,:),'r');
   title('message signal');xlabel('t--->');ylabel('m(t)');grid on
   hold on
   subplot(5,1,4);plot(t,ask_sig(i,:));
   title('ASK signal');xlabel('t--->');ylabel('s(t)');grid on
   hold on
   end
  hold off
  %Plot the carrier signal and input binary data
  subplot(5,1,3);plot(t,c);
  title('carrier signal');xlabel('t--->');ylabel('c(t)');grid on
  subplot(5,1,1);stem(m);
  title('binary data bits');xlabel('n--->');ylabel('b(n)');grid on
```

Output

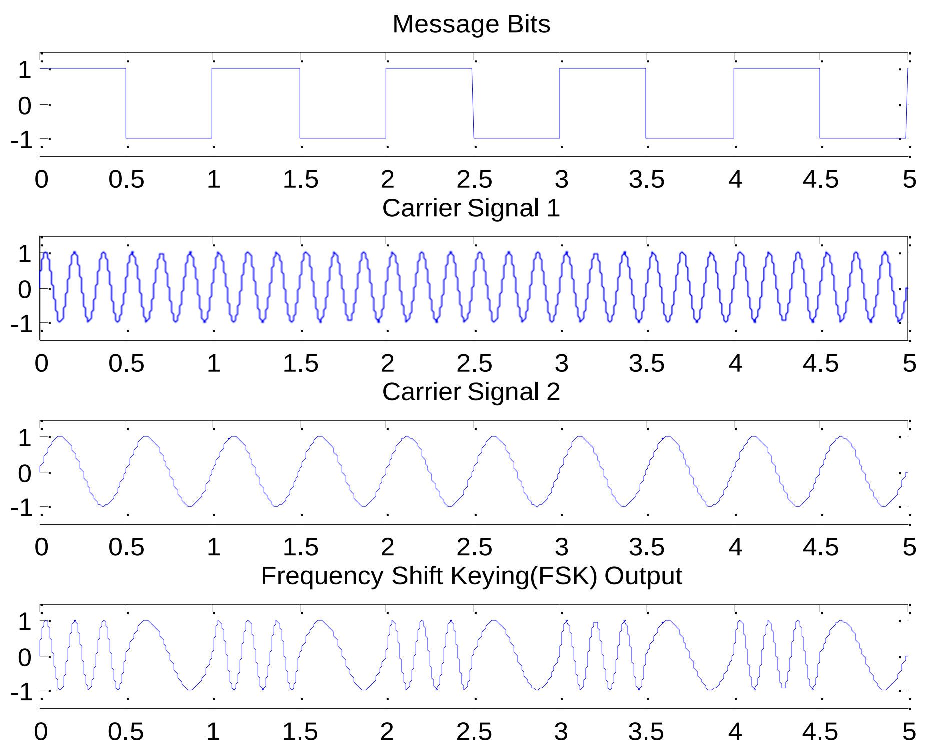


PROGRAM FOR FSK:

```
%%%%%%%%%%%    FSK    %%%%%%%%%%%%%%%

clc;
clear all;
close all;
t=[0:0.0001:5];
s=square(2*pi*t,100);
s1=square(2*pi*t);
s2=0.5*(s+s1);
s3=-square(2*pi*t);
s4=0.5*(s+s3);
x1=sin(2*pi*6*t)
x=x1.*s2;
y1=sin(2*pi*2*t)
y=y1.*s4;
z= x+y;
subplot(4,1,1)
plot(t,s1)
axis([0 5 -1.5 1.5])
title('Message Bits')
subplot(4,1,2)
plot(t,x1)
axis([0 5 -1.5 1.5])
```

```
title('Carrier Signal 1');
subplot(4,1,3)
plot(t,y1)
axis([0 5 -1.5 1.5])
title('Carrier Signal 2');
subplot(4,1,4)
plot(t,z)
axis([0 5 -1.5 1.5])
title('Frequency Shift Keying(FSK) Output');
```

Output



PROGRAM FOR BPSK:

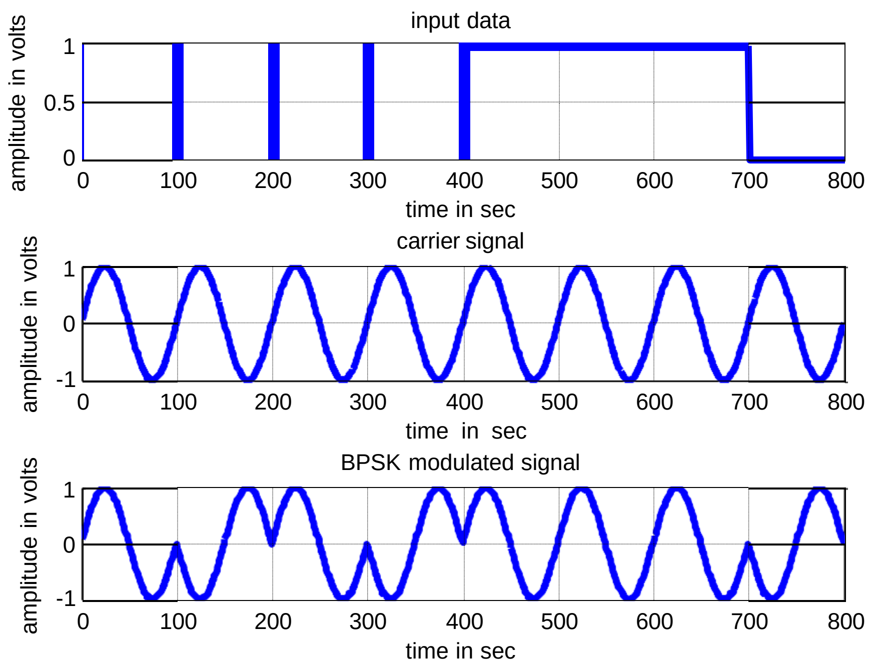
```
%BPSK Modulation
 clc;
 clear all;
 close all;
 n=input('Enter the data bits'); y=length(n);
 freq=input('enter the frequency');
 for i=1:y
```

```
        if n(1,i)==0
    for t=((i-1)*100+1):(i*100)
    y(t)=sin(2*pi*freq*t/1000+pi); x(t)=0;
    end
    else
    for t=(i-1)*100+1:(i*100)
    y(t)=sin(2*pi*freq*t/1000); x(t)=1;
    end
    end
 end
 figure(1);
 subplot(2,1,1); plot(x);
 title('inputdata');
 subplot(2,1,2); plot(y);
 xlabel('time in sec');
 ylabel('amplitude in volts');
 title('PSK');
grid on;
```

Command Window input

```
 Enter the data bits            [1 0 1 0 1 1 1 0]
  Enter the frequency            10
```

Output



POST-LAB QUESTIONS:

1. Plot Constellation diagram for ASK signal.
2. Obtain FSK Constellation with binary mapping.

3.Generate the BPSK signal for the data 11101110.

RESULT:

Thus the ASK PSK and FSK signals are generated using MATLAB.

Ex. No:6 b

SIMULATION OF GENERATION OF QPSK, QAM AND DPSK SCHEMES

AIM:

To obtain the signal constellation of QPSK, QAM and DPSK waveforms using MATLAB.

SOFTWARE REQUIRED:

SYSTEM with MATLAB software

PRE-LAB QUESTIONS:

1. What is M-ary modulation?
2. Compare the performance of QPSK and QAM.
3. What is Constellation in the digital modulation techniques?

THEORY:

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 (and referred to as CPSK).Alternatively, instead of using the bit patterns to *set* the phase of the wave, it can instead be used to *change* it by a specified amount. The demodulator then determines the *changes* in the phase of the received signal rather than the phase itself. Since this scheme depends on the difference between successive phases, it is termed **differential phase-shift keying (DPSK)**. DPSK can be significantly simpler to implement than ordinary PSK since there is no need for the demodulator to have a copy of the reference signal to determine the exact phase of the received signal (it is a non-coherent scheme). In exchange, it produces more erroneous demodulations. The exact requirements of the particular scenario under consideration determine which scheme is used. Quadrature amplitude modulation (QAM) is both an analog and a digital modulation scheme. It conveys two analog message signals, or two digital bit streams, by changing (modulating) the amplitudes of two carrier waves, using the amplitude-shift keying (ASK) digital modulation scheme or amplitude modulation (AM) analog modulation scheme. QAM is used extensively as a modulation scheme for digital telecommunication systems. Arbitrarily high spectral efficiencies can be achieved with QAM by setting a suitable constellation size, limited only by the noise level and linearity of the communications channel.

ALGORITHM:

1. Start.
2. Get the data bits and compute its length.
3. Get the input frequency.
4. Generate the waveforms with their corresponding carrier frequency.
5. Plot the output waveform.
6. Stop

PROGRAM FOR QPSK:

```
%%%%%%%%%%%    QPSK    %%%%%%%%%%%%%%%%
```

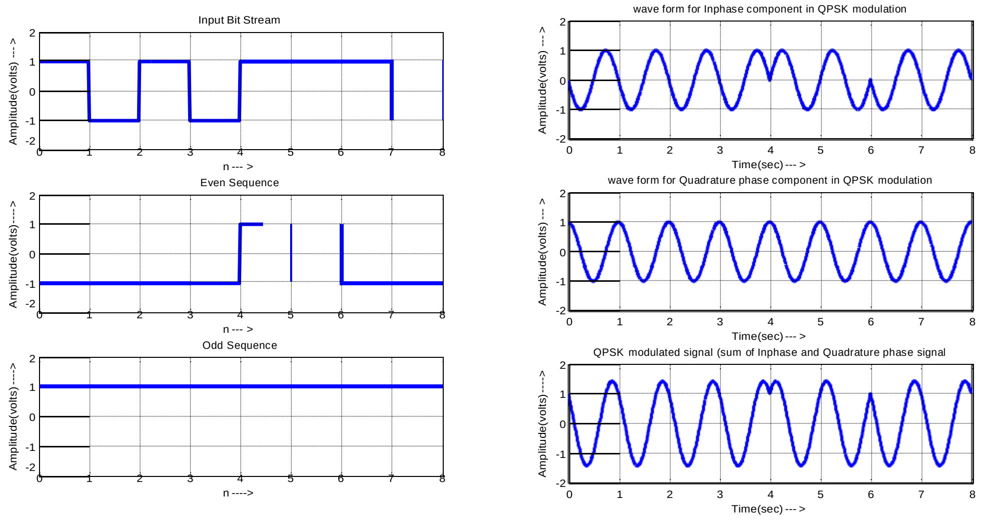
```
clear;
clc;
b = input('Enter the bit stream = ');
n = length(b); % length of the input bit stream
t = 0:0.01:n;
x = 1:1:(n+2)*100;
for i = 1:n
if (b(i) == 0)
u(i) = -1;
else
u(i) = 1;
end
for j = i:0.1:i+1
input_bits(x(i*100:(i+1)*100)) = u(i);
if (mod(i,2) == 0)
even_bits(x(i*100:(i+1)*100)) = u(i);
even_bits(x((i+1)*100:(i+2)*100)) = u(i);
else
odd_bits(x(i*100:(i+1)*100)) = u(i);
odd_bits(x((i+1)*100:(i+2)*100)) = u(i);
end
if (mod(n,2)~= 0)
even_bits(x(n*100:(n+1)*100)) = -1;
even_bits(x((n+1)*100:(n+2)*100)) = -1;
end
end
end
input_bits = input_bits(100:end);
odd_bits = odd_bits(100:(n+1)*100);
even_bits = even_bits(200:(n+2)*100);
cost = cos(2*pi*t);
sint = sin(2*pi*t);
x = odd_bits.*cost; % In-Phase Signal component
y = even_bits.*sint; % Quadrature phase Signal Component
z = x+y; % QPSK Signal
subplot(3,2,1);
plot(t,input_bits, 'linewidth',3);
xlabel('n ---- >');
ylabel('Amplitude(volts)---- >');
title('Input Bit Stream');
grid on ;
axis([0 n -2 +2]);
subplot(3,2,2);
plot(t,y,'linewidth',3);
```

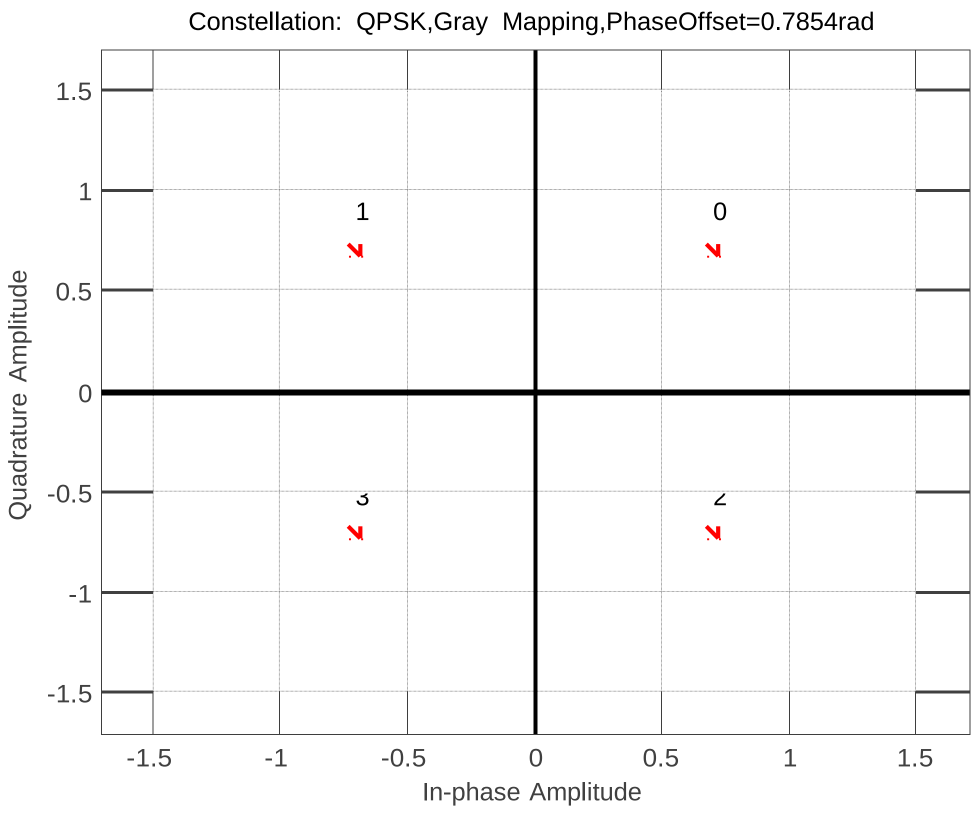
```
xlabel('Time(sec) ---->');
ylabel('Amplitude(volts)---- >');
title('wave form for Inphase component in QPSK modulation');
grid on ;
axis([0 n -2 +2]);
subplot(3,2,3);
plot(t,even_bits,'linewidth',3);
xlabel('n ---- >');
ylabel('Amplitude(volts)---- >');
title('Even Sequence');
grid on ;
axis([0 n -2 +2]);
subplot(3,2,4);
plot(t,x, 'linewidth',3);
xlabel('Time(sec) ---->');
ylabel('Amplitude(volts)---- >');
title('wave form for Quadrature phase component in QPSK modulation');
grid on ;
axis([0 n -2 +2]);
subplot(3,2,5);
plot(t,odd_bits,'linewidth',3);
xlabel('n ---- >');
ylabel('Amplitude(volts)---- >');
title('Odd Sequence');
grid on ;
axis([0 n -2 +2]);
subplot(3,2,6);
plot(t,z,'linewidth',3);
xlabel('Time(sec) ---->');
ylabel('Amplitude(volts)---- >');
title('QPSK modulated signal (sum of Inphase and Quadrature phase signal');
grid on ;
axis([0 n -2 +2]);
```

%Command Window Input

```
Enter the bit stream = [1 0 1 0 1 1 1 0]
```

Output


Input Bit Stream
Amplitude(volts) --- >
n --- >
Even Sequence
Amplitude(volts)---->
n --- >
Odd Sequence
Amplitude(volts)---->
n ---->
wave form for Inphase component in QPSK modulation
Amplitude(volts) --- >
Time(sec) --- >
wave form for Quadrature phase component in QPSK modulation
Amplitude(volts) --- >
Time(sec) --- >
QPSK modulated signal (sum of Inphase and Quadrature phase signal
Amplitude(volts) --- >
Time(sec) --- >


Constellation: QPSK,Gray Mapping,PhaseOffset=0.7854rad
Quadrature Amplitude
In-phase Amplitude
1
0
3
2

PROGRAM FOR QAM:

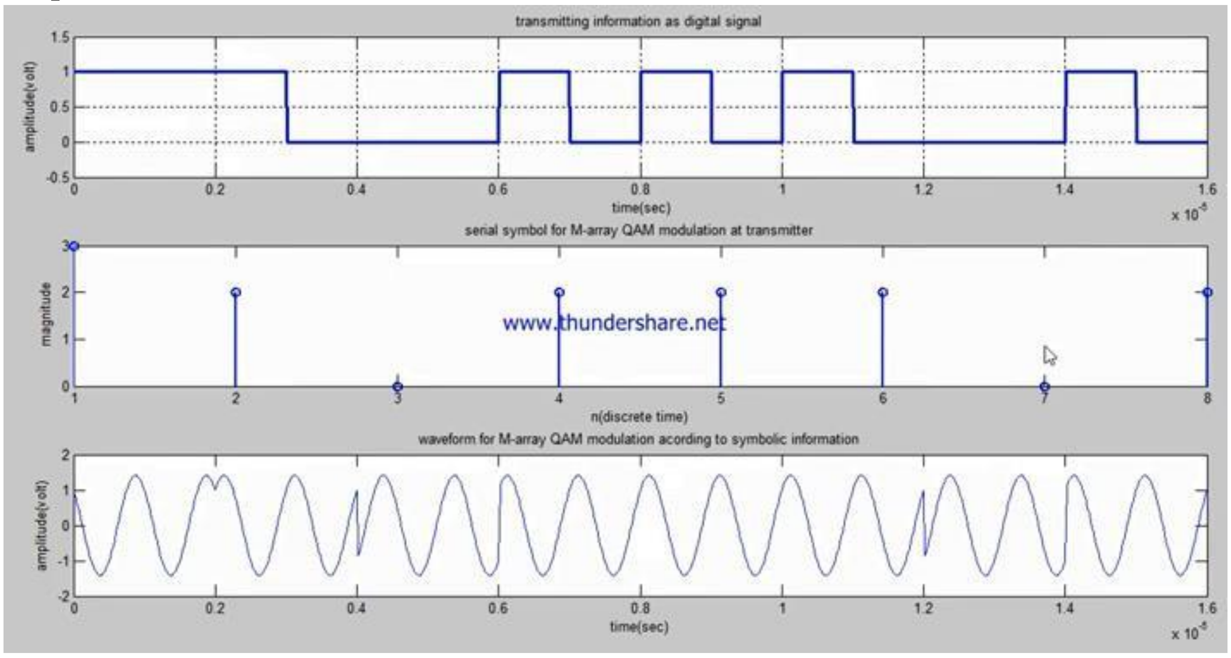
```
nbit=16;                                %number of information bits
msg=round(rand(nbit,1));           % information generation as binary form
disp(' binary information at transmitter ');
disp(msg);
fprintf('\n\n');

%XX representation of transmitting binary information as digital signal XXX
x=msg;
bp=.000001;                                          % bit period
bit=[];
for n=1:1:length(x)
    if x(n)==1;
       se=ones(1,100);
    else x(n)==0;
        se=zeros(1,100);
    end
     bit=[bit se];

end
t1=bp/100:bp/100:100*length(x)*(bp/100);
figure(1)
subplot(3,1,1);
plot(t1,bit,'lineWidth',2.5);grid on;
axis([ 0 bp*length(x) -.5 1.5]);
ylabel('amplitude(volt)');
xlabel(' time(sec)');
title('transmitting information as digital signal');


% binary information convert into symbolic form for M-array QAM modulation
M=M;                                    % order of QAM modulation
msg_reshape=reshape(msg,log2(M),nbit/log2(M))';
disp(' information are reshaped for convert symbolic form');
disp(msg_reshape);
fprintf('\n\n');

size(msg_reshape);
for(j=1:1:nbit/log2(M))
   for(i=1:1:log2(M))
       a(j,i)=num2str(msg_reshape(j,i));
   end
end


as=bin2dec(a);
ass=as';
figure(1)
subplot(3,1,2);
stem(ass,'Linewidth',2.0);
title('serial symbol for M-array QAM modulation at transmitter');
xlabel('n(discrete time)');
ylabel(' magnitude');


disp('symbolic form information for M-array QAM ');
disp(ass);
fprintf('\n\n');
```

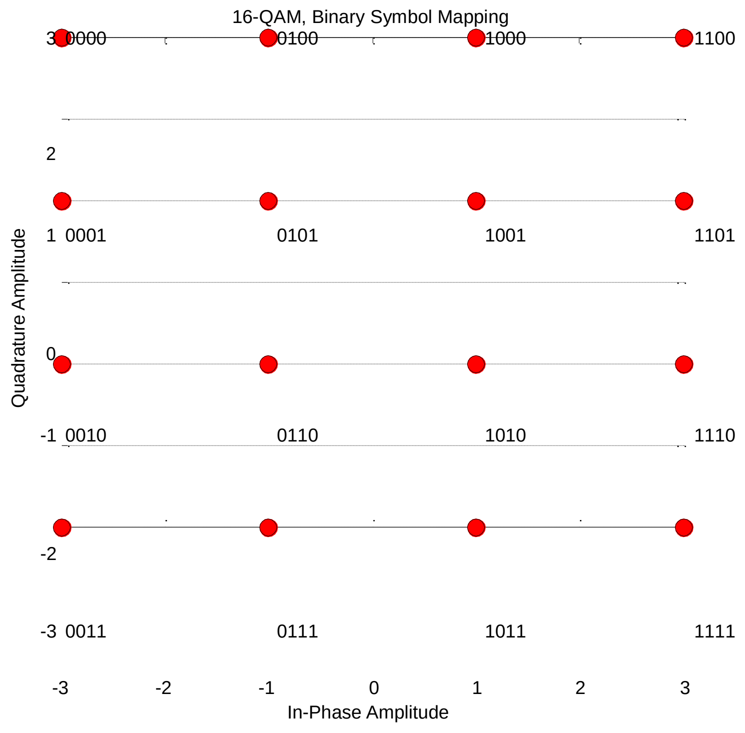
```
%XXXXXXXXXXXXXX Mapping for M-array QAM modulation XXXXXXXXXXXXXXXXXXXXXXXXX
M=M;                                    %order of QAM modulation
x1=[0:M-1];
p=qammod(ass,M)     %constalation design for M-array QAM acording to symbol
sym=0:1:M-1;     % considerable symbol of M-array QAM, just for scatterplot
pp=qammod(sym,M);                  %constalation diagram for M-array QAM
scatterplot(pp),grid on;
title('consttelation diagram for M-array QAM');

%XXXXXXXXXXXXXXXXXXXXXXX  M-array QAM modulation XXXXXXXXXXXXXXXXXXXXXXXXXXX
RR=real(p)
II=imag(p)
sp=bp*2;                              %symbol period for M-array QAM
sr=1/sp;                                      % symbol rate
f=sr*2;
t=sp/100:sp/100:sp;
ss=length(t);
m=[];
for(k=1:1:length(RR))
   yr=RR(k)*cos(2*pi*f*t);                  % inphase or real component
   yim=II(k)*sin(2*pi*f*t);           % Quadrature or imagenary component
   y=yr+yim;
   m=[m y];
end
tt=sp/100:sp/100:sp*length(RR);
figure(1);
subplot(3,1,3);
plot(tt,m);
title('waveform for M-array QAM modulation acording to symbolic information');
xlabel('time(sec)');
ylabel('amplitude(volt)');


%XXXXXXXXXXXXXXXXXXXXX M-array QAM demodulation XXXXXXXXXXXXXXXXXXXXXXXXXXX
m1=[];
m2=[];
for n=ss:ss:length(m)
  t=sp/100:sp/100:sp;
  y1=cos(2*pi*f*t);                          % inphase component
  y2=sin(2*pi*f*t);                        % quadrature component
  mm1=y1.*m((n-(ss-1)):n);
  mm2=y2.*m((n-(ss-1)):n);
  z1=trapz(t,mm1)                                % integration
  z2=trapz(t,mm2)                                % integration
  zz1=round(2*z1/sp)
  zz2=round(2*z2/sp)
  m1=[m1 zz1]
  m2=[m2 zz2]
end



%XXXXXXXXXXXXXXXXXXXX de-mapping for M-array QAM modulation XXXXXXXXXXXXXXXX
clear i;
clear j;
for (k=1:1:length(m1))
gt(k)=m1(k)+j*m2(k);
```

```
end
gt

ax=qamdemod(gt,M);
figure(3);
subplot(2,1,1);
stem(ax,'linewidth',2);
title(' re-obtain symbol after M-array QAM demodulation ');
xlabel('n(discrete time)');
ylabel(' magnitude');

disp('re-obtain symbol after M-array QAM demodulation ');
disp(ax);
fprintf('\n\n');


bi_in=dec2bin(ax);
[row col]=size(bi_in);
p=1;
 for(i=1:1:row)
     for(j=1:1:col)
         re_bi_in(p)=str2num(bi_in(i,j));
         p=p+1;
     end
 end
disp('re-obtain binary information after M-array QAM demodulation');
disp(re_bi_in')
fprintf('\n\n');


%XX representation of receiving binary information as digital signal XXXXXX
x=re_bi_in;
bp=.000001;                                        % bit period
bit=[];
for n=1:1:length(x)
    if x(n)==1;
       se=ones(1,100);
    else x(n)==0;
        se=zeros(1,100);
    end
     bit=[bit se];

end
t1=bp/100:bp/100:100*length(x)*(bp/100);
figure(3)
subplot(2,1,2);
plot(t1,bit,'lineWidth',2.5);grid on;
axis([ 0 bp*length(x) -.5 1.5]);
ylabel('amplitude(volt)');
xlabel(' time(sec)');
title('receiving information as digital signal after M-array QAM demoduation');


%>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> end of program <<<<<<<<<<<<<<<<<<<<<<<<<

%16-QAM Modulation Signal Constellation
clc;
M = 16;                  % Modulation order
x = (0:15)';             % Integer input
```

```
y1 = qammod(x, 16, 0);          % 16-QAM output, phase offset = 0
plot(y1,'ok','MarkerSize',10,'MarkerFaceColor', 'r')
text(real(y1)+0.1, imag(y1), dec2bin(x))
title('16-QAM, Binary Symbol Mapping')
axis([-3 3 -3 3])
xlabel('In-Phase Amplitude ');
ylabel('Quadrature Amplitude');
grid on
```

<u>Output</u>

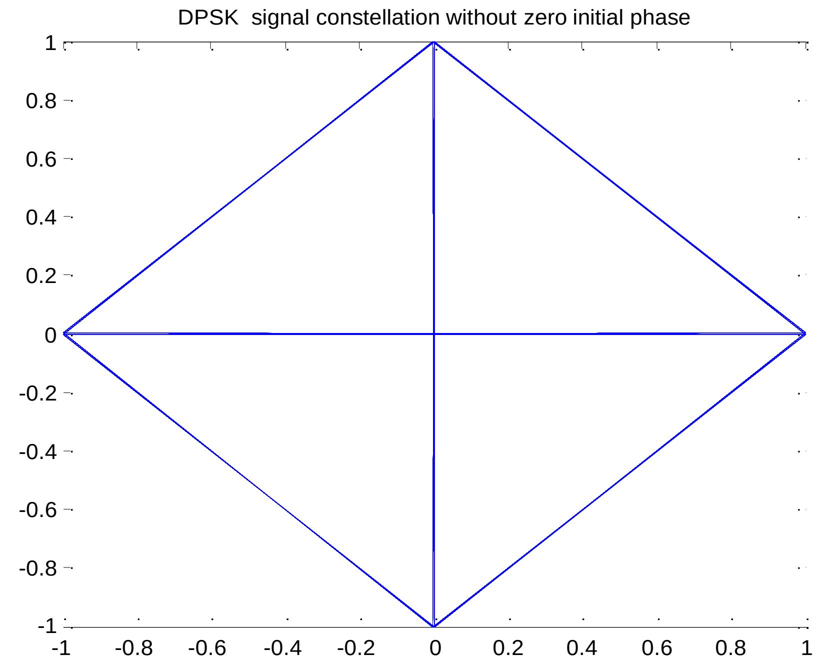


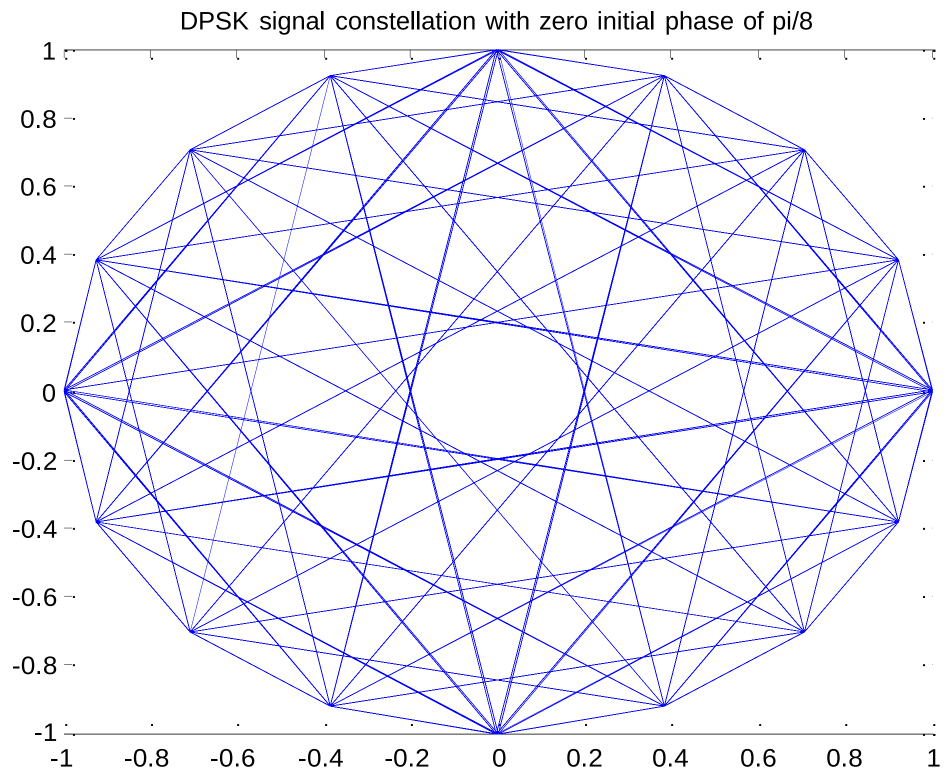


PROGRAM FOR DPSK:

```
clc;
clear all; close all;
M = 4; % Alphabet size
x = randi([0 M-1],1000,1); % Random message
y = dpskmod(x,M); % Modulate.
z = dpskdemod(y,M); % Demodulate.
s1 = symerr(x,z) % Expect one symbol error, namely, the first symbol.
s2 = symerr(x(2:end),z(2:end)) % Ignoring 1st symbol, expect no errors. The output is below.
s1 =1
s2 =0
y = dpskmod(x,M)
s = RandStream.create('mt19937ar', 'seed',131);
prevStream = RandStream.setGlobalStream(s); % seed for repeatability
M = 4; % Use DQPSK in this example, so M is 4.
x = randi([0 M-1],500,1); % Random data
y1 = dpskmod(x,M,pi/8); % Modulate using a nonzero initial phase.
plot(y) % Plot all points, using lines to connect them
title(' DPSK  signal constellation without zero initial phase')
figure (2)
plot(y1)
title(' DPSK signal constellation with zero initial phase of pi/8')
```

Output:


DPSK signal constellation without zero initial phase
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1


DPSK signal constellation with zero initial phase of pi/8
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1

POST-LAB QUESTIONS:

1. Plot Constellation diagram for 64-QAM signal.
2. Obtain QPSK Constellation with binary mapping.
3.Generate the QPSK signal for the data 11101110.

<u>RESULT</u>:

Thus the Constellation of DPSK, QPSK and QAM signals are generated using MATLAB.

Ex. No:7

SIMULATION OF LINEAR BLOCK CODING SCHEMES

AIM:

To write a program in MATLAB for Linear Block coding technique.

SOFTWARE REQUIRED:

SYSTEM with MATLAB

PRE-LAB QUESTIONS:

1. What is linear block coding ?
2. Define error controlling technique.
3. How to compute syndrome table?

THEORY:

In coding theory, a linear code is an error-correcting code for which any linear combination of code words is also a codeword. Linear codes are traditionally partitioned into block codes and convolutional codes, although turbo codes can be seen as a hybrid of these two types. Linear codes allow for more efficient encoding and decoding algorithms than other codes.

Linear codes are used in forward error correction and are applied in methods for transmitting symbols (e.g., bits) on a communications channel so that, if errors occur in the communication, some errors can be corrected or detected by the recipient of a message block. The codewords in a linear block code are blocks of symbols which are encoded using more symbols than the original value to be sent. A linear code of length *n* transmits blocks containing *n* symbols.

ALGORITHM:

1. Generate the message sequence of length ‘k’.
2. Generate the coefficient matrix ‘p’ of order k by (n-k).
3. Generate the identity matrix G and multiply with p matrix to form generator matrix.
4. Linear block code is calculated as, LBC=[M]*[G].
5. Obtain the parity check matrix.
6. Add noise to code word, [r]=[c]+[e].
7. Construct the syndrome table.
8. Decode the desired message from code word.

PROGRAM:

```
clc;
clear all;
close all;
n=7; k=4;
```

```
% message of length ‘k’
M=[0 0 0 0; 0 0 0 1; 0 0 1 0; 0 0 1 1; 0 1 0 0; 0 1 0 1; 0 1 1 0; 0 1 1 1];
disp('message');
disp(M);
%p is coefficient matrix k by(n-k)
p=[1 0 1; 0 1 0; 1 0 0; 1 1 0];
display('parity');
disp(p);
%Generator matri
G=[[p],eye(k)];
display('generator matrix'); disp(G);
%Linear Block Code
c=encode(M,n,k, 'linear fmt',G);
c=rem(M*G,2);
display('code word');
disp(c);
%parity check matrix
H=[eye(n-k),[p]']; display('parity check matrix'); disp(H);
%Addition of noise
display('error');
e=randerr(8,n);
display('received matrix');
r=rem(plus(c,e),2); disp(r);
%decoding error matrix
disp('syndrome error');
e=rem(r*H',2);
disp(e);
%syndrome table
disp('Decoding table');
t=syndtable(H);
disp(t);
[msg,err,cc]=decode(r,n,k,'linear fmt',G,t);
disp('decoded message');
disp(cc);
```

Output

```
  message
0   0   0   0
0   0   0   1
0   0   1   0
0   0   1   1
0   1   0   0
0   1   0   1
0   1   1   0
0   1   1   1
```

```
         parity
         1   0   1
         0   1   0
         1   0   0
         1   1   0

    generator matrix
 1   0   1   1   0   0   0
 0   1   0   0   1   0   0
 1   0   0   0   0   1   0
 1   1   0   0   0   0   1

       code word
 0   0   0   0   0   0   0
 1   1   0   0   0   0   1
 1   0   0   0   0   1   0
 0   1   0   0   0   1   1
 0   1   0   0   1   0   0
 1   0   0   0   1   0   1
 1   1   0   0   1   1   0
 0   0   0   0   1   1   1


   parity check matrix
 1   0   0   1   0   1   1
 0   1   0   0   1   0   1
 0   0   1   1   0   0   0

         error
    received matrix
 0   0   0   0   0   1   0
 1   1   0   0   1   0   1
 1   1   0   0   0   1   0
 0   1   0   0   0   1   0
 0   1   0   1   1   0   0
 1   0   1   0   1   0   1
 1   1   0   0   1   0   0
 0   1   0   0   1   1   1


     syndrome error
         1   0   0
         0   1   0
         0   1   0
         1   1   0
         1   0   1
         0   0   1
         1   0   0
         0   1   0
```

```
                    Decoding table
Single-error patterns loaded in decoding table.  2 rows remaining.
        2-error patterns loaded.  0 rows remaining.
     0    0    0    0    0    0    0
     0    0    1    0    0    0    0
     0    1    0    0    0    0    0
     0    1    1    0    0    0    0
     1    0    0    0    0    0    0
     0    0    0    1    0    0    0
     0    0    0    0    0    0    1
     0    1    0    1    0    0    0

             decoded message
     1    0    0    0    0    1    0
     1    0    0    0    1    0    1
     1    0    0    0    0    1    0
     0    1    0    0    0    1    1
     0    1    0    0    1    0    0
     1    0    0    0    1    0    1
     0    1    0    0    1    0    0
     0    0    0    0    1    1    1
```

RESULT:

Thus the implementation of linear block code (7, 4) was done using MATLAB.

Ex-No :8 **Error Correction and Error Detection Techniques**

AIM:

To develop a program to study and implement the error correction and error detection using *CRC-CCITT (16bit).*

ALGORITHM:

1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).
2. Divide the result by G(x). The remainder = C(x).
3. If: x div y gives remainder c that means: x = n y + c Hence (x-c) = n y (x-c) div y gives remainder 0
   Here (x-c) = (x+c) Hence (x+c) div y gives remainder 0
4. Transmit: T(x) = M(x) + C(x)
5. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

THEORY :

Whenever digital data is stored or interfaced, data corruption might occur. Since the beginning of computer science, developers have been thinking of ways to deal with this type of problem. For serial data they came up with the solution to attach a parity bit to each sent byte. This simple detection mechanism works if an odd number of bits in a byte changes, but an even number of false bits in one byte will not be detected by the parity check. To overcome this problem developers have searched for mathematical sound mechanisms to detect multiple false bits. The **CRC** calculation or *cyclic redundancy check* was the result of this. Nowadays CRC calculations are used in all types of communications. All packets sent over a network connection are checked with a CRC.
The idea behind CRC calculation is to look at the data as one large binary number. This number is divided by a certain value and the remainder of the calculation is called the CRC. Dividing in the CRC calculation at first looks to cost a lot of computing power, but it can be performed very quickly if we use a method similar to the one learned at school.

Example: (Write in the left side of the observation/Record)

We will as an example calculate the remainder for the character 'm'—which is 1101101 in binary notation—by dividing it by 19 or 10011. Please note that 19 is an odd number. This is necessary as we will see further on. Please refer to your schoolbooks as the binary calculation method here is not very different from the decimal method you learned when you were young. It might only look a little bit strange. Also notations differ between countries, but the method is similar

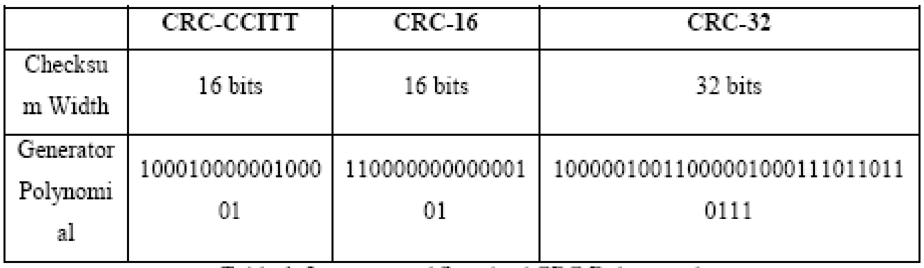
```
                    1 0 1 = 5
            -------------
1 0 0 1 1 / 1 1 0 1 1 0 1
            1 0 0 1 1 | |
            --------- | |
              1 0 0 0 0 |
              0 0 0 0 0 |
              --------- |
              1 0 0 0 0 1
                1 0 0 1 1
                ---------
                  1 1 1 0 = 14 = remainder
```

Basic mathematical process:

The message bits are appended with *c* zero bits; this *augmented message* is the dividend

- A predetermined *c+1*-bit binary sequence, called the *generator polynomial*, is the divisor
- The checksum is the *c*-bit remainder that results from the division operation

Table 1 lists some of the most commonly used generator polynomials for 16- and 32-bit CRCs. Remember that the width of the divisor is always one bit wider than the remainder. So, for example, you'd use a 17-bit generator polynomial whenever a 16-bit checksum is required.



	CRC-CCITT	CRC-16	CRC-32
Checksum Width	16 bits	16 bits	32 bits
Generator Polynomial	10001000000100001	11000000000000101	100000100110000010001110110110111

Table 1. International Standard CRC Polynomials

Error detection with CRC

Consider a message represented by the polynomial M(x)

Consider a *generating polynomial* G(x) This is used to generate a CRC = C(x) to be appended to M(x). Note this G(x) is prime.

Steps:

1. Multiply M(x) by highest power in G(x). i.e. Add So much zeros to M(x).
2. Divide the result by G(x). The remainder = C(x). Special case: This won't work if bitstring =all zeros. We don't allow such an M(x).But M(x) bitstring = 1 will work, for example. Can divide 1101 into 1000.
3. If: x div y gives remainder c that means: x = n y + c Hence (x-c) = n y (x-c) div y gives remainder 0 Here (x-c) = (x+c) Hence (x+c) div y gives remainder 0
4. Transmit: T(x) = M(x) + C(x)
5. Receiver end: Receive T(x). Divide by G(x), should have remainder 0.

Note if G(x) has order n - highest power is xn, then G(x) will cover (n+1) bits and the *remainder* will cover n bits. i.e. Add n bits (Zeros) to message.

Source Code:

```
#include<stdio.h>
int a[100],b[100],i,j,len,k,count=0;
//Generator Polynomial:g(x)=x^16+x^12+x^5+1
int gp[]={1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,1,};
int main()
{
void div();
system("clear");
printf("\nEnter the length of Data Frame :");
scanf("%d",&len);
printf("\nEnter the Message :");
for(i=0;i<len;i++)
scanf("%d",&a[i]);
//Append r(16) degree Zeros to Msg bits
for(i=0;i<16;i++)
a[len++]=0;
//Xr.M(x) (ie. Msg+16 Zeros)
for(i=0;i<len;i++)
b[i]=a[i];
//No of times to be divided ie. Msg Length
k=len-16;
div();
for(i=0;i<len;i++)
b[i]=b[i]^a[i]; //MOD 2 Substraction
printf("\nData to be transmitted : ");
for(i=0;i<len;i++)
printf("%2d",b[i]);
printf("\n\nEnter the Reveived Data : ");
for(i=0;i<len;i++)
scanf("%d",&a[i]);
div();
for(i=0;i<len;i++)
if(a[i]!=0)
{
printf("\nERROR in Recived Data");
return 0;
}
printf("\nData Recived is ERROR FREE");
}
void div()
{
for(i=0;i<k;i++)
{
if(a[i]==gp[0])
{
for(j=i;j<17+i;j++)
a[j]=a[j]^gp[count++];
}
count=0;
}
}
```

Output:

```
[root@localhost ]# cc prg1.c
[root@localhost ]# ./a.out
Enter the length of Data Frame :4
Enter the Message :1 0 1 1
Data to be transmitted : 1 0 1 1 1 0 1 1 0 0 0 1 0 1 1 0 1 0 1 1
Enter the Reveived Data : 1 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1
ERROR in Recived Data
Reminder is : 0000000100000000
```

RESULT: Thus the error correction and error detection using *CRC-CCITT (16bit)* was implemented.

Ex.No: 9a **Implementation of Stop and Wait Protocol**

AIM:

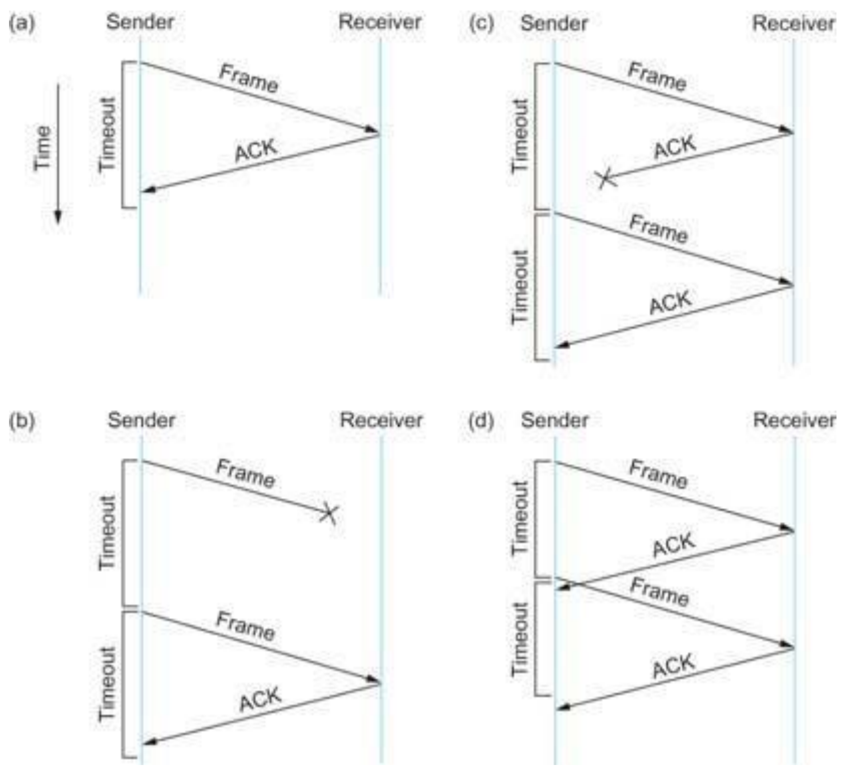
To write a C program to demonstrate the working of simple Stop and Wait Protocol.

THEORY:

The **Stop-and-Wait protocol** uses both flow and error control. In this protocol, the sender sends one frame at a time and waits for an acknowledgment before sending the next one. To detect corrupted frames, we need to add a CRC to each data frame. When a frame arrives at the receiver site, it is checked. If its CRC is incorrect, the frame is corrupted and silently discarded. The silence of the receiver is a signal for the sender that a frame was either corrupted or lost.

Every time the sender sends a frame, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next frame (if it has one to send). If the timer expires, the sender resends the previous frame, assuming that the frame was either lost or corrupted. This means that the sender needs to keep a copy of the frame until its acknowledgment arrives. When the corresponding acknowledgment arrives, the sender discards the copy and sends the next frame if it is ready. Only one frame and one acknowledgment can be in the channels at any time.

The advantages are Limited buffer size, sooner Errors detection and prevents one station occupying medium for long periods.



PRE-LAB QUESTIONS:

1. What are the responsibilities of data link layer?
2. Define flow control.
3. What is the function of stop and wait flow control?

4. Mention the advantage and disadvantage of stop and wait flow control.

ALGORITHM:

(i) Start Program

1. Start
2. Declare the required variables and file pointers.
3. Get the choice from the user if he needs a Selective Retransmission or Go-Back-N protocol.
4. Select “1” for Sending the data and “2” for the end of transmission
5. For sending the data open a text file in write mode and write the data that has to be sent.
6. Once written close the file.
7. Check the ack.txt file in which the acknowledgement from the receiver is stored.
8. If the acknowledgment is positive, then send the data to the receiver.
9. If all the data are sent, then select Option “2” to end the transmission.

(ii) Stop Program

1. Start
2. Declare the required variables and file pointers.
3. For receiving the data, open a data.txt text file in read mode and read the data that was sent by the receiver.
4. Open the ack.txt file in write mode and write as “Yes” if received correctly.
5. Else, write as ‘No” in the ack.txt file.
6. Close the file.

Sender Module

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>
#include<string.h>
void main()
{
int ch,strt=1;
char input[20],ack[4];
FILE*in,*ak;
clrscr();
printf("\nStop and Wait Protocol\n");
printf("\n1.Send\n2.End Of Transmission\n");
strcpy(ack,"yes");
while(1)
{
if(strt|=1)
printf("\nEnter your choice...");
scanf("%d",&ch);
switch(ch)
{
case 1:
```

```
if (strt|=1)
{
ak=fopen("ack.txt","r");
fscanf(ak,"%s",ack);
fclose(ak);
}
if(((strcmp(ack,"yes"))==0)||(strt==1))
{
in=fopen("data.txt","w");
printf("\nEnter the Data...");
scanf("%s",input);
fprintf(in,"%s",input);
fclose(in);
printf("\nData Sent\n");
strt=0;
}
else
{exit(0);}
break;
case 2:
{exit(0);
break;}
}

getch();
}
}
```

Receiver Module

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<conio.h>
void main()
{
int one,choice,frst=1;
char output[20],akstr[4];
FILE*out,*ak1;
clrscr();
while(1)
{
if(frst==0)
{
out=fopen("data.txt","r");
fscanf(out,"%s",output);
printf("\nReceived data:");
printf("%s",output);
}
fclose(out);
printf("\n1.Yes\n2.No");
printf("\nDo you want to Acknowledge the Data?\t");
scanf("%d",&choice);
frst=0;
if(choice==1)
  {
  strcpy(akstr,"yes");
  }
else
  {
  strcpy(akstr,"no");
  }
ak1=fopen("ack.txt","w");
fprintf(ak1,"%s",akstr);
fclose(ak1);
if(choice==2)
break;
}
}
```

RESULT: Thus the working of simple error control functionality was implemented using the simple Stop and wait protocol in C language.

Ex.No: 9b **Implementation and study of selective repeat and Go back-N protocols**

AIM:

To write a C program to demonstrate the working of selective repeat and Goback-N Protocol.

THEORY:

Four protocols have been defined for the data-link layer to deal with flow and error control. Simple, Stop-and-Wait, Go-Back-N, and Selective-Repeat.

The sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window.

Go-Back-N : It is based on sliding window for error control. If no error occurs ACK is sent as usual with next frame expected. If error occurs then discard that frame and all future frames until error frame received correctly. Transmitter must go back and retransmit that frame and all subsequent frames.

Selective Reject:

It also called selective retransmission for error control. Here only the rejected frames are retransmitted. Subsequent frames are accepted by the receiver and buffered. This minimizes retransmission. Receiver must maintain large enough buffer.

ALGORITHM:

1. Start the program
2. Include the required header files.
3. Declare and Initialize the variables and file pointers.
4. Enter the choice to choose protocol - Selective Retransmission or Go-Back-N
5. Get the size of the data to be sent
6. Check for the acknowledgement. If yes, then enter the data one by one\.
7. If the acknowledgement is not received, then ask enter the position that has be sent again.
8. If the protocol is Selective Retransmission, then the data of that particular position will be sent.
9. If the protocol is Go-Back-N, then all the data from the position will be sent**.**
10. It the data are transmitted then choose the Exit option to quit the program.

Selective repeat and GoBack N

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<string.h>
int main()
{
int data[5],rec[5];
int n,m,a,i,c=0,j;
char r[5],ch='y';
clrscr();
do
{
```

```
printf("\n Enter the choice:");
printf("\n 1.Select Retransmission\n");
printf("\n 2.Go back n");
printf("\n 3.Exit\n");
scanf("%d",&a);
printf("\n Enter the size of data: ");
scanf("%d",&n);
printf("\n Enter the data one by one :");
for(i=0;i<n;i++)
{
scanf("%d", &data[i]);
}
switch(a)
{
case 1:
{
printf("\n data is ready to send");
for(i=0;i<n;i++)
 {
rec[i]=data[i];
printf("\n %d is sent \n", data[i]);
printf("\nIf the data is received (y/n)?\n");
scanf("%s", r);
if(strcmp(r,"n"))
printf("\n ack is rec %d\n",i+1);
else
c++;
 }
if(c!=0)
{
   printf("\n enter position of data to send:");
   scanf("%d",&m);
   }
while(m>n)
{
     printf("\n Wrong choice");
     printf("\n Enter position of data again:");
    scanf("%d",&m);
}
if(a==1)
             printf("\n %d is sent again\n", data[m-1]);
getch();
}
break;
case 2:
{
    for(i=0;i<n;i++)
    {

      for(j=0;j<n;j++)
      {
           rec[j]=data[j];
           printf("\n %d is sent\n",data[j]);
           printf("\nIf the data is received (y/n)?\n");
           scanf("%s",r);
           if(strcmp(r,"n"))
                     printf("\n ack is rec %d\n",j);
       else
                     c++;
      }
if(c!=0)
{

    printf("\n enter starting position of data to be sent");
    scanf("%d",&m);
    for(i=m;i<n;i++)
      {
             printf("\n %d is sent",data[i]);


      }
 }
}
}
getch();
break;
default:exit(0);
}}
while(ch=='y');
    return(0);
}
```

RESULT: Thus the working of selective repeat and GoBack n were implemented and understood in C language.

Ex.No: 10a **Implementation of IP address**

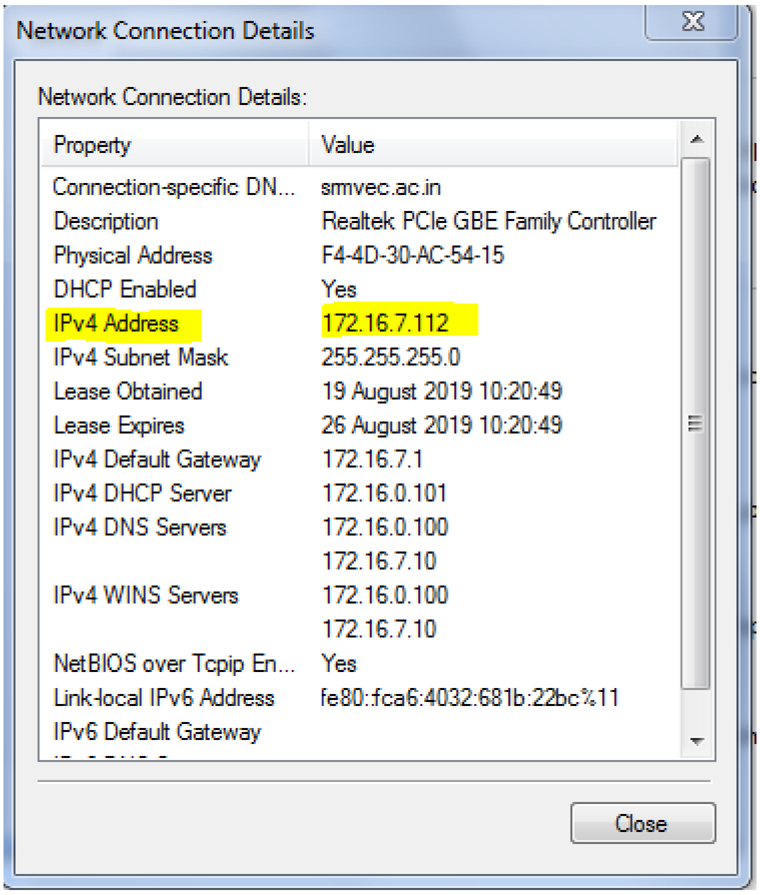
A. IP Address Configuration:

An Internet Protocol (IP) address is a unique number assigned to every device on a network. Just as a street address determines where a letter should be delivered, an IP address identifies computers on the Internet. Network devices use IP addresses to communicate with each other.

IP addresses are required by any network adapter on any computer that needs to connect to the Internet or another computer. Addresses are given out to network computers in one of two manners, dynamically or statically.

To set a static IP address in Windows 7, 8, and 10:

- Click Start Menu > Control Panel > Network and Sharing Center or Network and Internet > Network and Sharing Center.
- Click on Local Area Connection.
- Click Details.
- View for the Internet Protocol Version 4 (TCP/IPv4) address.



Network Connection Details:

Property	Value
Connection-specific DN...	srmvec.ac.in
Description	Realtek PCIe GBE Family Controller
Physical Address	F4-4D-30-AC-54-15
DHCP Enabled	Yes
IPv4 Address	172.16.7.112
IPv4 Subnet Mask	255.255.255.0
Lease Obtained	19 August 2019 10:20:49
Lease Expires	26 August 2019 10:20:49
IPv4 Default Gateway	172.16.7.1
IPv4 DHCP Server	172.16.0.101
IPv4 DNS Servers	172.16.0.100
	172.16.7.10
IPv4 WINS Servers	172.16.0.100
	172.16.7.10
NetBIOS over Tcpip En...	Yes
Link-local IPv6 Address	fe80::fca6:4032:681b:22bc%11
IPv6 Default Gateway	

RESULT:

Thus the IP address configuration was implemented successfully.

Ex.No: 10b Implementation of IP Commands such as ping, Trace route, nslookup.

IP Commands

(i)Ping

Ping is a standard application found on most laptop and desktop computers. Apps that support ping can also be installed on smartphones and other mobile devices. Additionally, websites that support Internet speed test services often include ping as one of their features.

A ping utility sends test messages from the local client to a remote target over the TCP/IP network connection. The target can be a Web site, a computer, or any other device with an IP address. Besides determining whether the remote computer is currently online, ping also provides indicators of the general speed or reliability of network connections.

Running Ping:

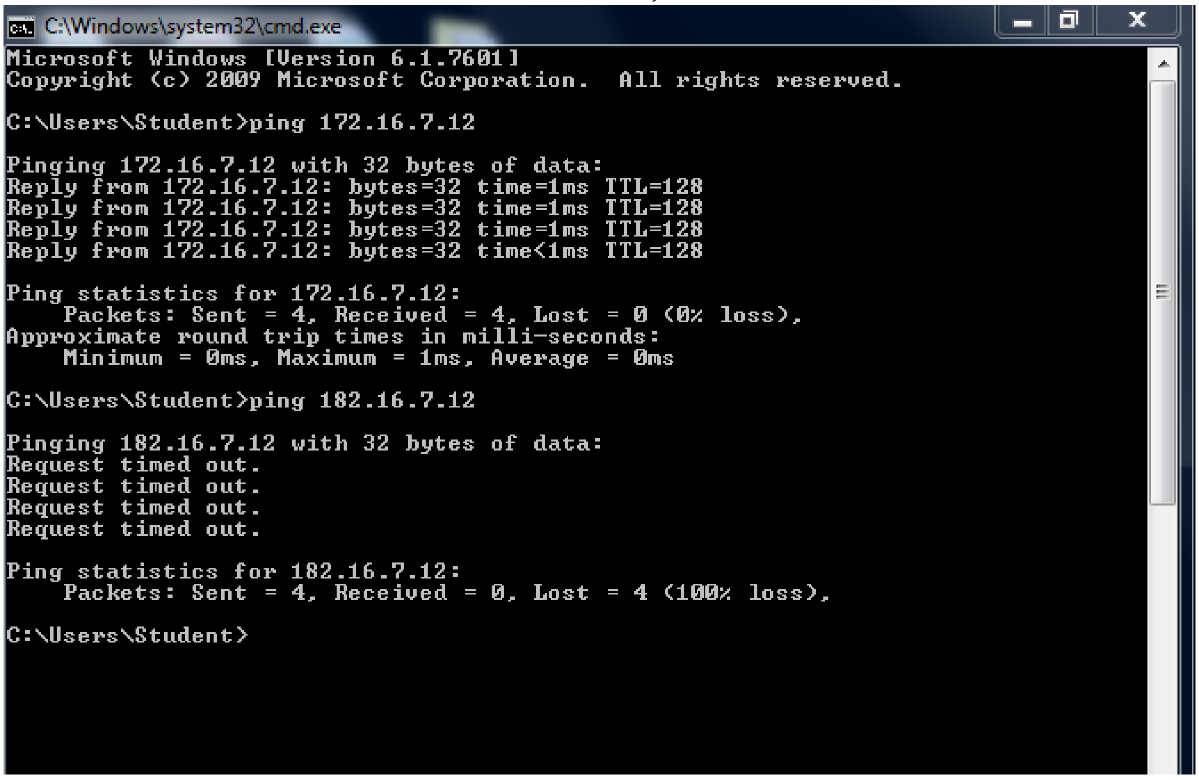
Microsoft Windows, Mac OS X, and Linux provide command line ping programs that can be run from the operating system shell. Computers can be pinged by either IP address or by name.

To ping a computer by IP address:

- Open a shell prompt (in Microsoft Windows, the Command Prompt or MS-DOS Prompt on the Start Menu).
- Type ping followed by a space and then the IP address.
- Press the Enter (or Return) key.

Interpreting the Results of Ping

- **Reply from**: By default, Microsoft Windows ping sends a series of four messages to the address. The program outputs a confirmation line for each response message received from the target computer.
- **Bytes**: Each ping request is 32 bytes in size by default.
- **Time:** Ping reports the amount of time (in milliseconds) between the sending of requests and receipt of responses.
- **TTL (Time-to-Live):** A value between 1 and 128, TTL can be used to count how many different networks the ping messages passed through before reaching the target computer. A value of 128 indicates the device is on the local network, with 0 other networks in between.



C:\Windows\system32\cmd.exe

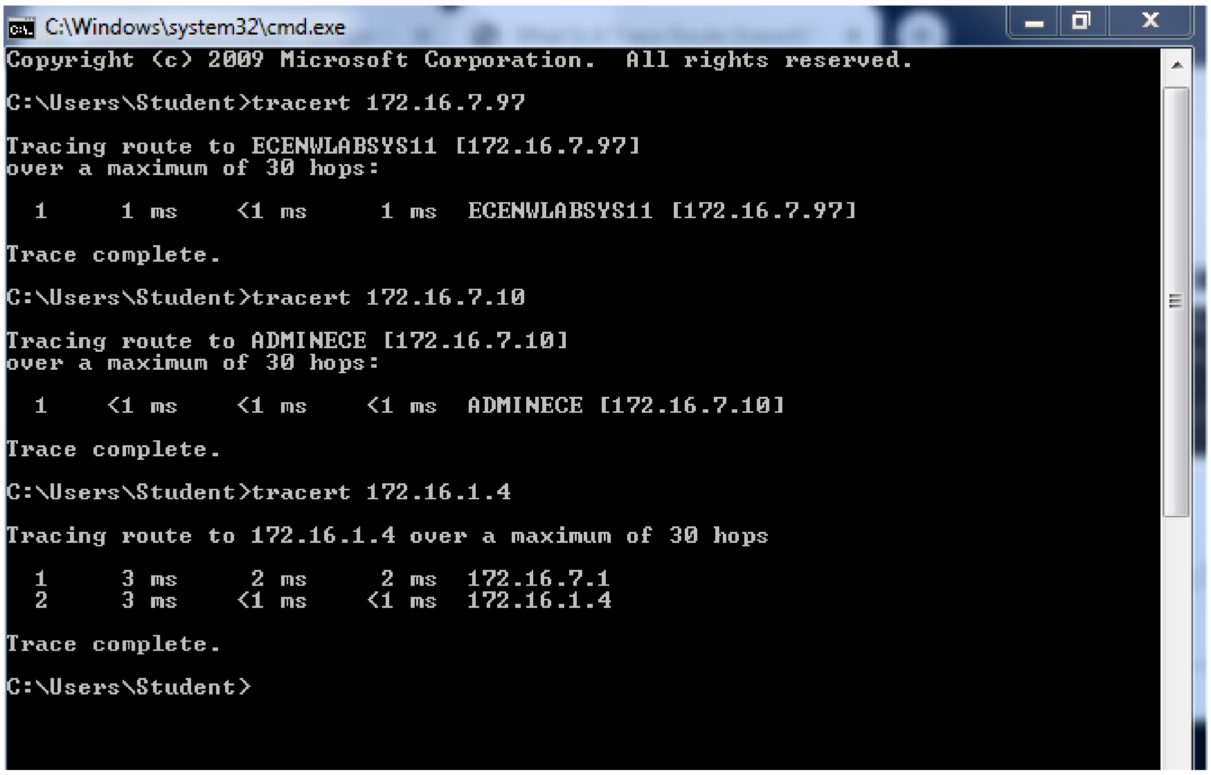
```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Student>ping 172.16.7.12

Pinging 172.16.7.12 with 32 bytes of data:
Reply from 172.16.7.12: bytes=32 time=1ms TTL=128
Reply from 172.16.7.12: bytes=32 time=1ms TTL=128
Reply from 172.16.7.12: bytes=32 time=1ms TTL=128
Reply from 172.16.7.12: bytes=32 time<1ms TTL=128

Ping statistics for 172.16.7.12:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\Users\Student>ping 182.16.7.12

Pinging 182.16.7.12 with 32 bytes of data:
Request timed out.
Request timed out.
Request timed out.
Request timed out.

Ping statistics for 182.16.7.12:
    Packets: Sent = 4, Received = 0, Lost = 4 (100% loss),

C:\Users\Student>
```

(ii) Traceroute

A *traceroute* is a function which traces the path from one network to another. It allows us to diagnose the source of many problems.

To run traceroute on Windows:

- Go to *Start* > *Run*.
- To open the command prompt type **cmd** and press the Enter key. This will bring up a command prompt window. It has a line that looks like this: **C:\Documents and Settings\yourname> _** with a cursor blinking next to the ">"symbol.
- In the command prompt, type: **tracert hostname** where **hostname** is the name of the server connection you are testing. (IP address)
- You may have to wait up to a minute or more for the test to complete. It will generate a list of the connections along the way and some information about the speed of the steps along the way.
- It sends us the complete results (every line) for analysis.



```
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Student>tracert 172.16.7.97

Tracing route to ECENWLABSYS11 [172.16.7.97]
over a maximum of 30 hops:

  1     1 ms    <1 ms     1 ms  ECENWLABSYS11 [172.16.7.97]

Trace complete.

C:\Users\Student>tracert 172.16.7.10

Tracing route to ADMINECE [172.16.7.10]
over a maximum of 30 hops:

  1    <1 ms    <1 ms    <1 ms  ADMINECE [172.16.7.10]

Trace complete.

C:\Users\Student>tracert 172.16.1.4

Tracing route to 172.16.1.4 over a maximum of 30 hops

  1     3 ms     2 ms     2 ms  172.16.7.1
  2     3 ms    <1 ms    <1 ms  172.16.1.4

Trace complete.

C:\Users\Student>
```

iii) Nslookup

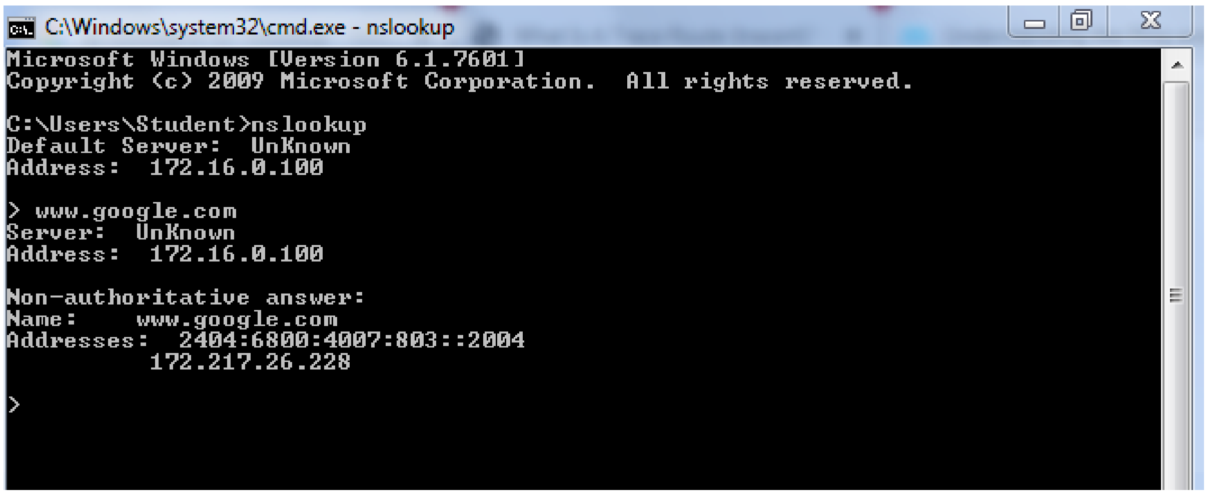
nslookup is a network administration command-line tool available for many computer operating systems.

- The main use of **nslookup** is for troubleshooting DNS related problems.
- Nslookup can be used in **interactive** and **non-interactive** mode.
- To use in interactive mode **type nslookup** at the command line and hit return.

To run nslookup on Windows:

1. Go to Start > Run and type cmd.
2. At a command prompt, type nslookup, and then press Enter.

3. Type server <IP address>;,where IP address is the IP address of your external DNS server
4. Type set q=MX, and then press Enter
5. Type <domain name>, where domain name is the name of your domain, and then press Enter.



```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Student>nslookup
Default Server:  UnKnown
Address:  172.16.0.100

> www.google.com
Server:  UnKnown
Address:  172.16.0.100

Non-authoritative answer:
Name:    www.google.com
Addresses:  2404:6800:4007:803::2004
          172.217.26.228

>
```

RESULT:

Thus the IP Commands such as ping, Traceroute, nslookup was implemented.

INTRODUCTION TO NETWORK SIMULATOR 2

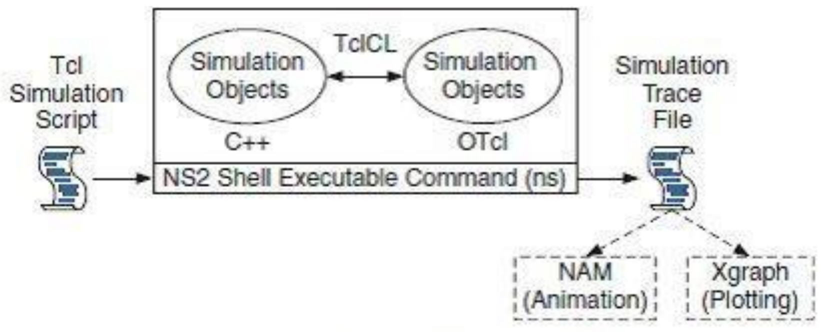
NS2 stands for Network Simulator Version 2. It is an open-source event-driven simulator designed specifically for research in computer communication networks.

Features of NS2

1. It is a discrete event simulator for networking research.
2. It provides substantial support to simulate bunch of protocols like TCP, FTP, UDP, https and DSR.
3. It simulates wired and wireless network.
4. It is primarily Unix based.
5. Uses TCL as its scripting language.
6. Otcl: Object oriented support
7. Tclcl: C++ and otcl linkage
8. Discrete event scheduler

Basic Architecture

NS2 consists of two key languages: C++ and Object-oriented Tool Command Language (OTcl). While the C++ defines the internal mechanism (i.e., a backend) of the simulation objects, the OTcl sets up simulation by assembling and configuring the objects as well as scheduling discrete events. The C++ and the OTcl are linked together using TclCL



Basic architecture of NS.

- Nam (**Network Animator**) is an animation tool to graphically represent the network and packet traces.
- **UDP** (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss-tolerating connections between applications on the internet.

Some basic Otcl script syntax :

- **Basic Command :**

```
set a 8

set b [expr $a/8]
```

In the first line, the variable **a** is assigned the value 8. In the second line, the result of the command [expr $a/8], which equals 1, is then used as an argument to another command, which in turn assigns a value to the variable **b**. The "$" sign is used to obtain a value contained in a variable and square brackets are an indication of a command substitution.

- **Define new procedures with *proc* command**

```
proc  factorial  fact  {
    if {$fact <= 1} {
        return 1
    }
expr $fact * [factorial [expr $fact-1]]
 }
```

- **To open a file for reading :**

```
set testfile [open hello.dat r]
```

Similarly, **put** command is used to write data into the file

```
set testfile [open hello.dat w]
puts $testfile "hello1"
```

- To call sub processes within another process, **exec** is used, which creates a sub process and waits for it to complete.

```
exec rm $testfile
```

- To be able to run a simulation scenario, a network topology must first be created. In ns2, the topology consists of a collection of nodes and links.

```
set ns [new Simulator]
```

- The simulator object has member functions which enables to create the nodes and define the links between them. The class simulator contains all the basic functions. Since ns was defined to handle the Simulator object, the command $ns is used for using the functions belonging to the simulator class.

- In the network topology nodes can be added in the following manner :

```
set n0 [$ns node]
set n1 [$ns node]
```

- Traffic agents (TCP, UDP etc.) and traffic sources (FTP, CBR etc.) must be set up if the node is not a router. It enables to create CBR traffic source using UDP as transport protocol or an FTP traffic source using TCP as a transport protocol.

CBR traffic source using UDP :

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packet_size_ 512
```

FTP traffic source using TCP :

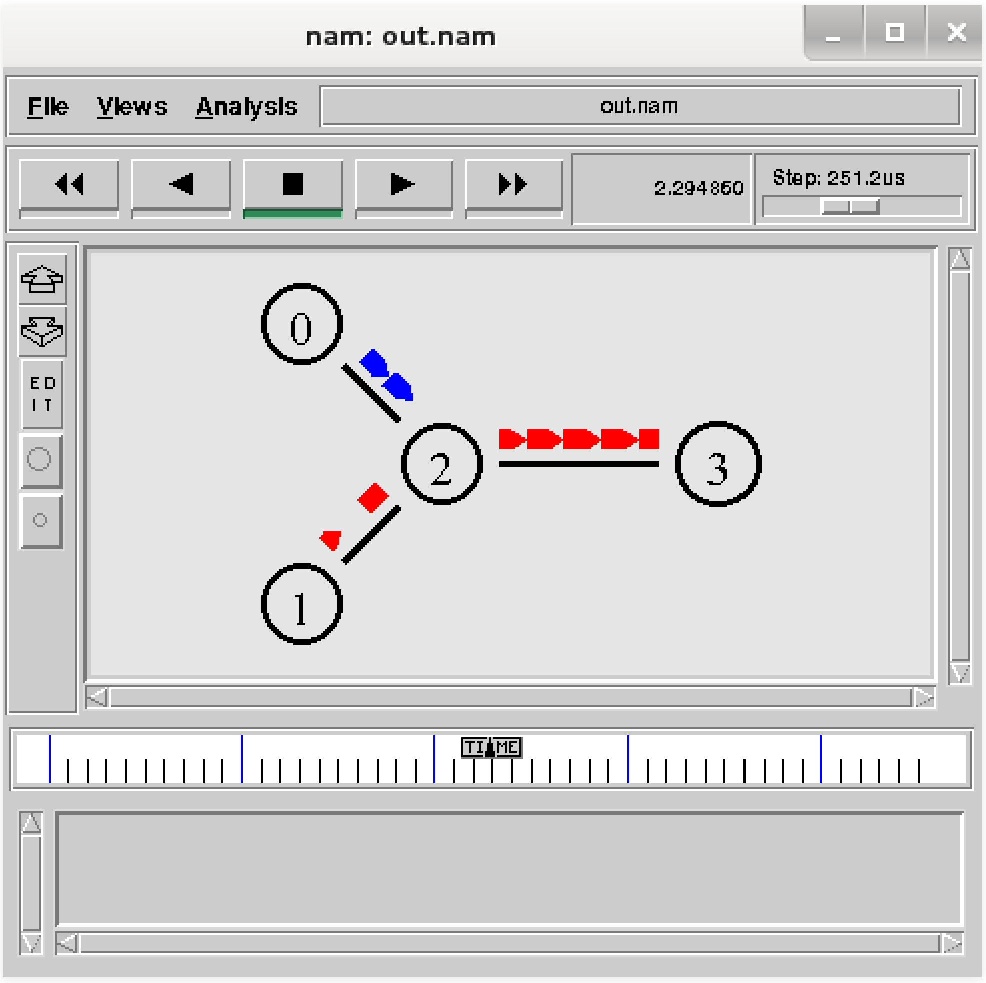
```
set tcp0 [new Agent/TCP]
$ns attach-agent $n0 $tcp0
set ftp0 [new Application/FTP]
$ftp0 attach-agent $tcp0
$tcp0 set packet_size_ 512
```

<u>Below is the implementation of creating links between the source and destination using both ftp and tcp :</u>

```
# Create a simulator object
set ns [new Simulator]

# Define different colors
# for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

# Open the NAM trace file
set nf [open out.nam w]
$ns namtrace-all $nf

# Define a 'finish' procedure
proc finish {}
{
   global ns nf
   $ns flush-trace

   # Close the NAM trace file
   close $nf

   # Execute NAM on the trace file
   exec nam out.nam &
   exit 0
}

# Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
# Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns duplex-link $n2 $n3 1.7Mb 20ms DropTail

# Set Queue Size of link (n2-n3) to 10
$ns queue-limit $n2 $n3 10

# Give node position (for NAM)
$ns duplex-link-op $n0 $n2 orient right-down
$ns duplex-link-op $n1 $n2 orient right-up
$ns duplex-link-op $n2 $n3 orient right

# Monitor the queue for link (n2-n3). (for NAM)
$ns duplex-link-op $n2 $n3 queuePos 0.5

# Setup a TCP connection
set tcp [new Agent/TCP]
$tcp set class_ 2
$ns attach-agent $n0 $tcp

set sink [new Agent/TCPSink]
$ns attach-agent $n3 $sink
$ns connect $tcp $sink
$tcp set fid_ 1

# Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP

# Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]

$ns attach-agent $n3 $null
$ns connect $udp $null
$udp set fid_ 2

# Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
```

```
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 1000
$cbr set rate_ 1mb
$cbr set random_ false

# Schedule events for the CBR and FTP agents
$ns at 0.1 "$cbr start"
$ns at 1.0 "$ftp start"
$ns at 4.0 "$ftp stop"
$ns at 4.5 "$cbr stop"

# Detach tcp and sink agents
# (not really necessary)
$ns at 4.5 "$ns detach-agent $n0 $tcp ; $ns detach-agent $n3 $sink"

# Call the finish procedure after
# 5 seconds of simulation time
$ns at 5.0 "finish"

# Print CBR packet size and interval
puts "CBR packet size = [$cbr set packet_size_]"
puts "CBR interval = [$cbr set interval_]"

# Run the simulation
$ns run
```

Output :


nam: out.nam
File
Views
Analysis
out.nam
2.294860
Step: 251.2us
EDIT
0
1
2
3
TIME

Ex.No :11 To create scenario and study the performance of network with CSMA / CA protocol and compare with CSMA/CD protocols.

AIM : To create a scenario and study the performance of network with CSMA protocol

THEORY:
Carrier Sense Multiple Access (CSMA) is a network protocol that listens to or senses network signals on the carrier/medium before transmitting any data. CSMA is implemented in Ethernet networks with more than one computer or network device attached to it. CSMA is part of the Media Access Control (MAC) protocol. CSMA works on the principle that only one device can transmit signals on the network, otherwise a collision will occur resulting in the loss of data packets or frames. CSMA works when a device needs to initiate or transfer data over the network. Before transferring, each CSMA must check or listen to the network for any other transmissions that may be in progress. If it senses a transmission, the device will wait for it to end. Once the transmission is completed, the waiting device can transmit its data/signals. However, if multiple devices access it simultaneously and a collision occurs, they bothhave to wait for a specific time before reinitiating the transmission process.
CSMA protocol was developed to overcome the problem found in ALOHA i.e. to minimize the chances of collision, so as to improve the performance. CSMA protocol is based on the principle of 'carrier sense'. The station senses the carrier or channel before transmitting a frame. It means the station checks the state of channel, whether it is idle or busy.
Even though devices attempt to sense whether the network is in use, there is a good chance that two stations will attempt to access it at the same time. On large networks, the transmission time between one end of the cable and another is enough that one station may access the cable even though another has already just accessed it.
The chances of collision still exist because of propagation delay. The frame transmitted by one station takes some time to reach other stations. In the meantime, other stations may sense the channel to be idle and transmit their frames. This results in the collision.
CSMA with collision detection:
CSMA/CD is used to improve CSMA performance by terminating transmission as soon as a collision is detected, thus shortening the time required before a retry can be attempted\
CSMA with Collision detection:
CSMA/CA collision avoidance is used to improve the performance of CSMA. If the transmission medium is sensed busy before transmission, then the transmission is deferred for a random interval. This random interval reduces the likelihood that two or more nodes waiting to transmit simultaneously begin transmission upon termination of the detected transmission, thus reducing the incidence of collision

ALGORITHM:
Step 1 : Start.
Step 2 : Create a simulator object.
Step 3 : Configure the simulator to use Link state routing.
Step 4 : Open the nam trace file.
Step 5 : Open the output file.
Step 6 : Define the finish procedure.
Step 7 : Close the trace file.
Step 8 : Create the required nodes.

Step 10 : Create links between the nodes.
Step 11 : Create a UDP agent to attach the node.
Step 12 : Create a CBR traffic router and attach.
Step 13 : Create a Null agent to the traffic sink.
Step 14 : Connect the traffic source to the sink.
Step 15 : Schedule the events for CBR agent.
Step 16 : Stop.

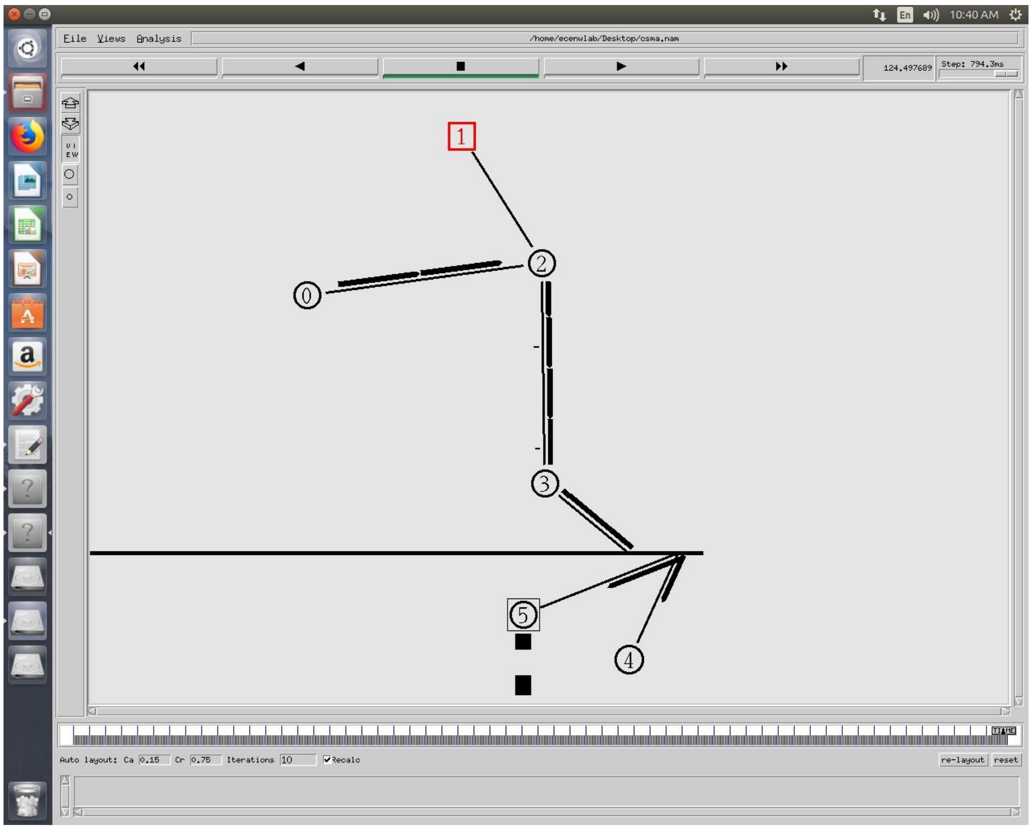
CSMA Protocol

```
set ns [new Simulator]

#Define different colors for data flows (for NAM)
$ns color 1 Blue
$ns color 2 Red

#Open the Trace files
set file1 [open out.tr w]
set winfile [open WinFile w]
$ns trace-all $file1

#Open the NAM trace file
set file2 [open out.nam w]
$ns namtrace-all $file2

#Define a 'finish' procedure
proc finish {} {
        global ns file1 file2
        $ns flush-trace
        close $file1
        close $file2
        exec nam out.nam &
        exit 0
}

#Create six nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$n1 color red
$n1 shape box

$n5 color red
$n5 shape box
```

```
#Create links between the nodes
$ns duplex-link $n0 $n2 2Mb 10ms DropTail
$ns duplex-link $n1 $n2 2Mb 10ms DropTail
$ns simplex-link $n2 $n3 0.3Mb 100ms DropTail
$ns simplex-link $n3 $n2 0.3Mb 100ms DropTail




set lan [$ns newLan "$n3 $n4 $n5" 0.5Mb 40ms LL Queue/DropTail MAC/Csma/Cd Channel]

#Setup a TCP connection
set tcp [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp
set sink [new Agent/TCPSink/DelAck]
$ns attach-agent $n4 $sink
$ns connect $tcp $sink
$tcp set fid_ 1
$tcp set window_ 80
$tcp set packetSize_ 5

#Setup a FTP over TCP connection
set ftp [new Application/FTP]
$ftp attach-agent $tcp
$ftp set type_ FTP


#Setup a UDP connection
set udp [new Agent/UDP]
$ns attach-agent $n1 $udp
set null [new Agent/Null]
$ns attach-agent $n5 $null
$ns connect $udp $null
$udp set fid_ 2

#Setup a CBR over UDP connection
set cbr [new Application/Traffic/CBR]
$cbr attach-agent $udp
$cbr set type_ CBR
$cbr set packet_size_ 100
$cbr set rate_ 0.01mb
$cbr set random_ false

$ns at 0.1 "$cbr start"
$ns at 2.0 "$ftp start"
$ns at 3.0 "$ftp stop"
$ns at 4.5 "$cbr stop"
```

```
$ns at 5.0 "finish"
$ns run
```

OUTPUT:


File
Views
Analysis
/home/ecenwlab/Desktop/csma.nam
124.497689
Step: 794.3ms
1
2
0
3
5
4
Auto layout: Ca 0.15 Cr 0.75 Iterations 10 Recalc
re-layout
reset
10:40 AM

RESULT:
Thus the study on CSMA protocol is carried out and implemented using NS2.

Ex.No :12 **Network Topology – Star, Bus, Ring, Mesh**

AIM:

To build and simulate a network under different topologies like Star,bus, mesh and ring, and observe the performance parameters.

REQUIREMENTS:

Operating System	:	Windows NT/2000/XP or LINUX
Programming Tool	:	Network Simulator (NS2)

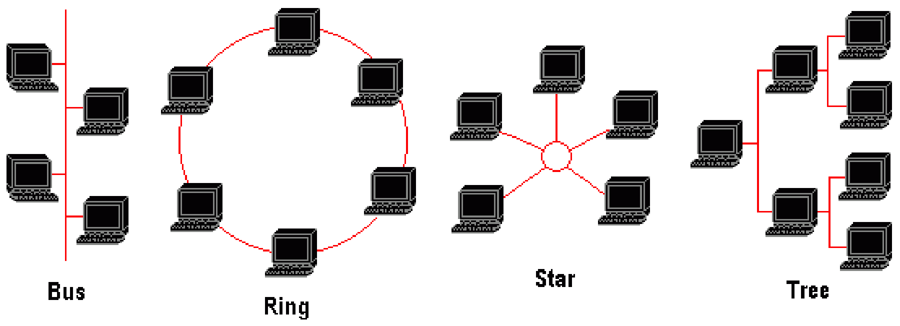
THEORY:

Network topology refers to the physical or logical layout of a network. It defines the way different nodes are placed and interconnected with each other. Alternately, network topology may describe how the data is transferred between these nodes.

There are two types of network topologies: physical and logical. Physical topology emphasizes the physical layout of the connected devices and nodes, while the logical topology focuses on the pattern of data transfer between network nodes.

The physical and logical network topologies of a network do not necessarily have to be identical. However, both physical and network topologies can be categorized into five basic models:

- **Bus Topology**: All the devices/nodes are connected sequentially to the same backbone or transmission line. This is a simple, low-cost topology, but its single point of failure presents a risk.
- **Star Topology**: All the nodes in the network are connected to a central device like a hub or switch via cables. Failure of individual nodes or cables does not necessarily create downtime in the network but the failure of a central device can. This topology is the most preferred and popular model.
- **Ring Topology**: All network devices are connected sequentially to a backbone as in bus topology except that the backbone ends at the starting node, forming a ring. Ring topology shares many of bus topology's disadvantages so its use is limited to networks that demand high throughput.
- **Tree Topology**: A root node is connected to two or more sub-level nodes, which themselves are connected hierarchically to sub-level nodes. Physically, the tree topology is similar to bus and star topologies; the network backbone may have a bus topology, while the low-level nodes connect using star topology.
- **Mesh Topology**: The topology in each node is directly connected to some or all the other nodes present in the network. This redundancy makes the network highly fault tolerant but the escalated costs may limit this topology to highly critical networks.



PROCEDURE:

1. Open a terminal and typethe TCL file in the vim editor with the command “vifilename.tcl” and save it.
2. Run the saved file with the command “ns filename.tcl”.
3. If any errors, edit them in the vim editor and rerun it again.
4. The output is displayed in the nam console.

STAR TOPOLOGY

```
set ns [new Simulator]
set nf [open out.nam w]
$ns namtrace-all $nf
proc finish {} {
    global ns nf
    $ns flush-trace
    close $nf
    exec nam out.nam &
    exit 0

}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$n0 shape square
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 10ms DropTail
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n0 $n5 1Mb 10ms DropTail
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
$ns connect $tcp0 $sink0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
```

```
$ns run
```

BUS TOPOLOGY

```
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   #Close the trace file
   close $nf
   #Executenam on the trace file
   exec nam out.nam &
   exit 0
}
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set lan0 [$ns newLan "$n0 $n1 $n2 $n3 $n4 $n5" 0.5Mb 80ms LL Queue/DropTail MAC/Csma/Cd
Channel]

#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 5
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
```

```
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

RING TOPOLOGY

```
#Create a simulator object
set ns [new Simulator]
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   #Close the trace file
   close $nf
   #Executenam on the trace file
   exec nam out.nam &
   exit 0
}
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
$ns duplex-link $n4 $n5 1Mb 10ms DropTail
$ns duplex-link $n5 $n0 1Mb 10ms DropTail

#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0
# Create a CBR traffic source and attach it to tcp0
```

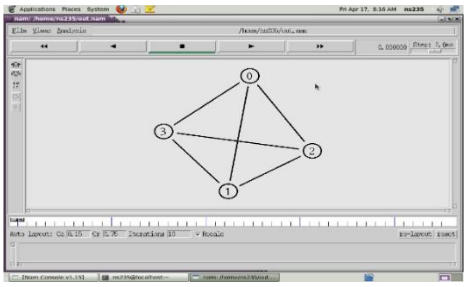
```
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"

#Run the simulation
$ns run
```

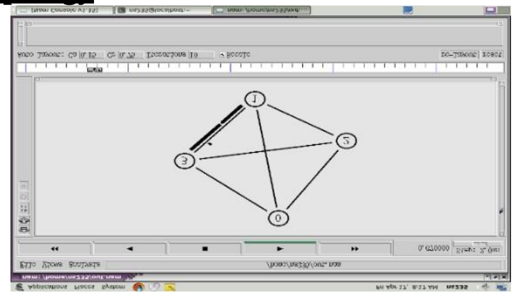
MESH TOPOLOGY

```
#Create a simulator object
set ns [new Simulator]

#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf

#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   #Close the trace file
   close $nf
   #Executenam on the trace file
   exec nam out.nam &
   exit 0
}
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]

#Create links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n2 1Mb 10ms DropTail
$ns duplex-link $n0 $n3 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n1 $n3 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail

#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
```

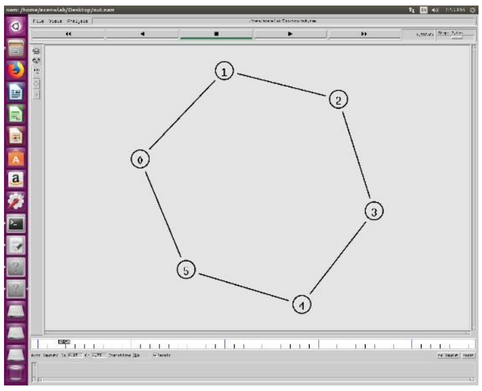
```
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0

# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

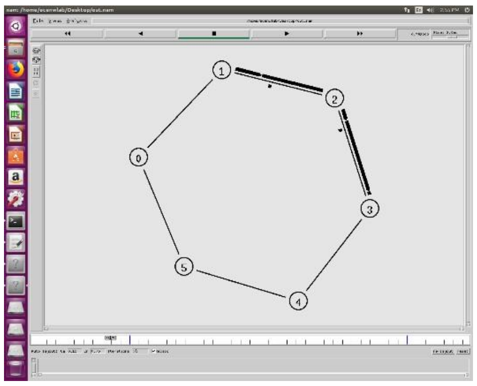
Mesh Topology



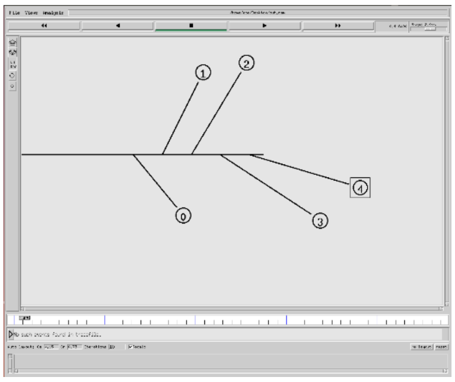


Ring Topology

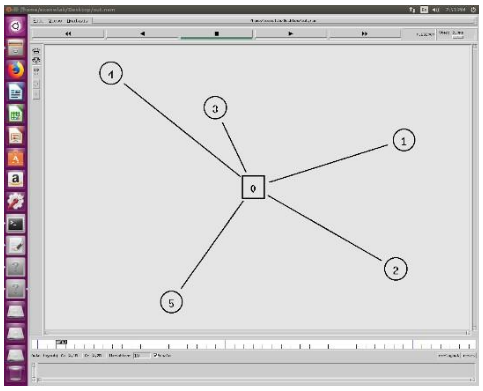




Bus Topology



Star Topology



RESULT:

Thus the network topologies like star, mesh, bus and ring have been implemented using network simulator.

Ex.No: 13a Implementation of Distance Vector Routing Algorithm

AIM:

To implement and simulate the distance vector routing algorithm using NS2 simulator.

REQUIREMENTS:

Operating System	:	Windows NT/2000/XP or LINUX
Programming Tool	:	Network Simulator (NS2)

THEORY:

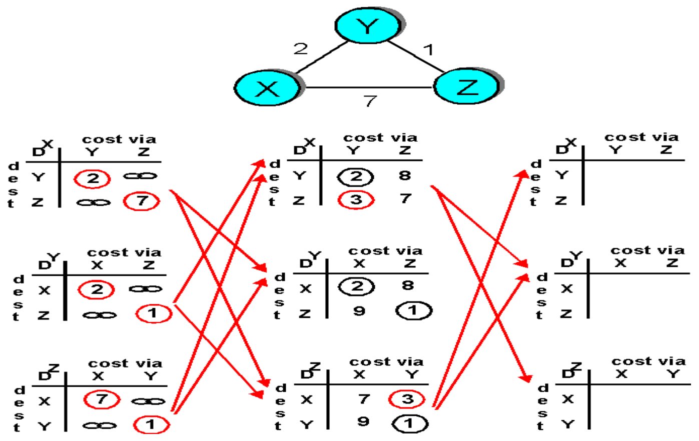
Computer networks generally use dynamic routing algorithms that are more complex than flooding, but more efficient because they find shortest paths for the current topology. Two dynamic algorithms in particular, distance vector routing and link state routing, are the most popular.

Distance Vector Routing

A **distance vector routing** algorithm operates by having each router maintain a table (i.e., a vector) giving the best known distance to each destination and which link to use to get there. These tables are updated by exchanging information with the neighbors. Eventually, every router knows the best link to reach each destination.

The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing algorithm, after the researchers who developed it (Bellman, 1957; and Ford and Fulkerson, 1962). It was the original ARPANET routing algorithm and was also used in the Internet under the name RIP.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for each router in the network. This entry has two parts: the preferred outgoing line to use for that destination and an estimate of the distance to that destination. The distance might be measured as the number of hops or using another metric, as we discussed for computing shortest paths.



The router is assumed to know the ‘‘distance’’ to each of its neighbors. If the metric is hops, the distance is just one hop. If the metric is propagation delay, the router can measure it directly with special ECHO packets that the receiver just timestamps and sends back as fast as it can. As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbors. Once every *T* msec, each router sends to each neighbor a list of its estimated delays to each destination. It also receives a similar list from each neighbor. Imagine that one of these tables has just come in from neighbor *X*, with *Xi* being *X*’s estimate of how long it takes to get to router *i*.

If the router knows that the delay to *X* is *m* msec, it also knows that it can reach router *i* via *X* in *Xi* + *m* msec. By performing this calculation for each neighbor, a router can find out which estimate seems the best and use that estimate and the corresponding link in its new routing table. Note that the old routing table is not used in the calculation.

The Count-to-Infinity Problem

The settling of routes to best paths across the network is called **convergence**. Distance vector routing is useful as a simple technique by which routers can collectively compute shortest paths, but it has a serious drawback in practice: although it converges to the correct answer, it may do so slowly.

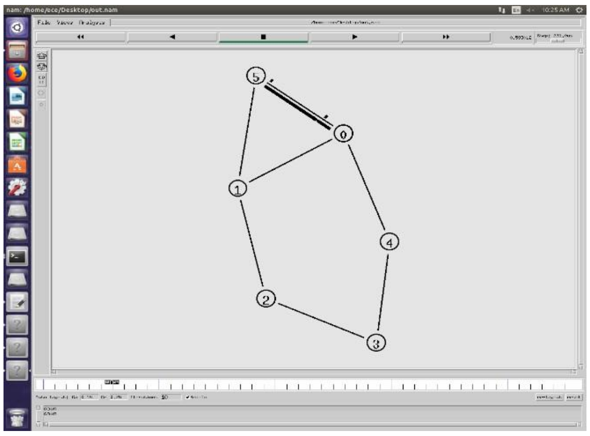
ALGORITHM:

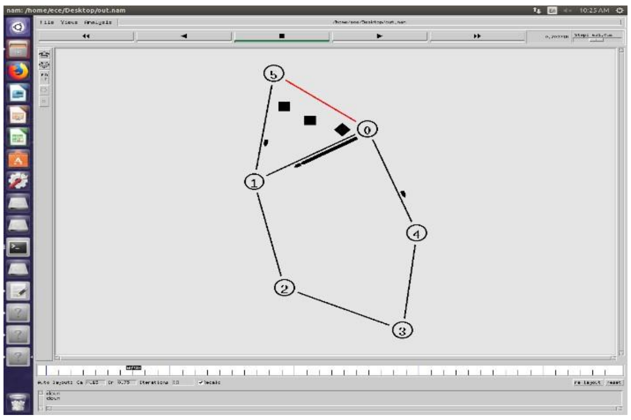
Step 1 : Start.
Step 2 : Create a simulator object.
Step 3 : Configure the simulator to use dynamic routing.
Step 4 : Open the nam.trace file.
Step 5 : Open the output file.
Step 6 : Define the finish procedure.
Step 7 : Close the trace file.
Step 8 : Create all the nodes.
Step 09 : Create links between the nodes.
Step 10 : Create a TCP agent to attach the node.
Step 11 : Create a CBR traffic router and attach.
Step 12 : Create a Null agent to the traffic sink.
Step 13 : Connect the traffic source to the sink.
Step 14 : Schedule the events for CBR agent.
Step 15 : Stop.

Distance Vector Routing Protocol

```
#Create a simulator object
set ns [new Simulator]
$ns rtproto DV
#Open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
#Define a 'finish' procedure
proc finish {} {
   global ns nf
   $ns flush-trace
   #Close the trace file
   close $nf
   #Executenam on the trace file
   exec nam out.nam &
   exit 0
}
#Create four nodes
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
#Create links between the nodes
$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n5 1Mb 10ms DropTail
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
#$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail
#Create a TCP agent and attach it to node n0
set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n1 $tcp0
#Create a TCP Sink agent (a traffic sink) for TCP and attach it to node n3
set sink0 [new Agent/TCPSink]
$ns attach-agent $n3 $sink0
#Connect the traffic sources with the traffic sink
$ns connect $tcp0 $sink0
# Create a CBR traffic source and attach it to tcp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
#Schedule events for the CBR agents
$ns at 0.5 "$cbr0 start"
$ns at 4.5 "$cbr0 stop"
#Call the finish procedure after 5 seconds of simulation time
$ns at 5.0 "finish"
#Run the simulation
$ns run
```

OUTPUT:





RESULT:

Thus the distance vector routing algorithm have been implemented and simulated using NS2 simulator.

Ex. No: 13b Implementation of Link State Routing Algorithm

AIM:

To implement and simulate the link state routing algorithm using NS2 simulator.

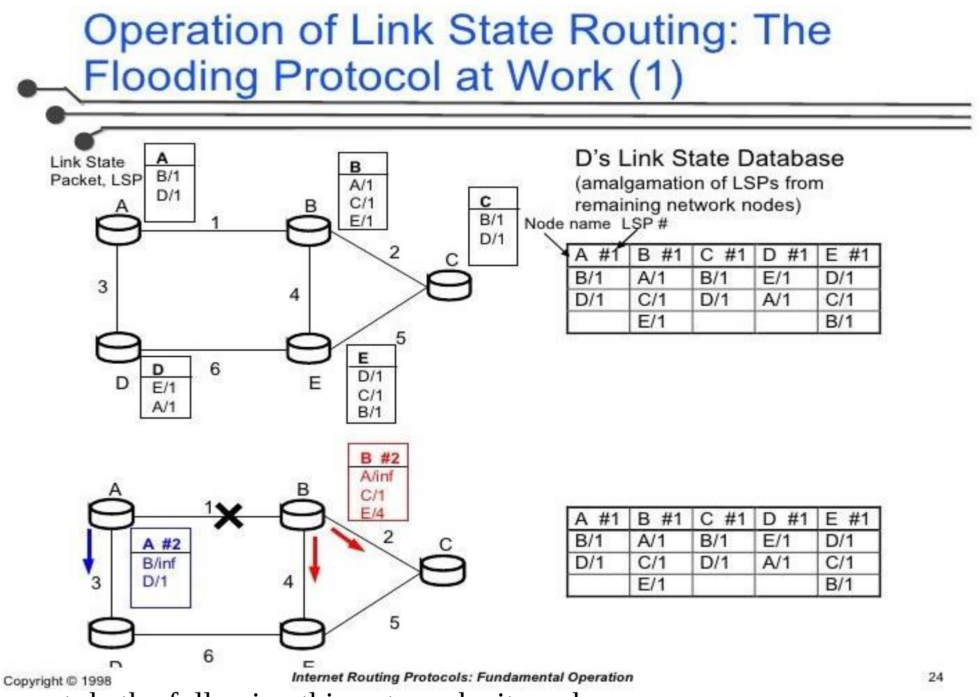
REQUIREMENTS:

Operating System	:	Windows NT/2000/XP or LINUX
Programming Tool	:	Network Simulator (NS2)

THEORY:

Link State Routing

Distance vector routing was used in the ARPANET until 1979, when it was replaced by link state routing. The primary problem that caused its demise was that the algorithm often took too long to converge after the network topology changed (due to the count-to-infinity problem). Consequently, it was replaced by an entirely new algorithm, now called **link state routing**. Variants of link state routing called IS-IS and OSPF are the routing algorithms that are most widely used inside large networks and the Internet today. The idea behind link state routing is fairly simple and can be stated as five parts.


Operation of Link State Routing: The Flooding Protocol at Work (1)
Link State Packet, LSP
D's Link State Database
(amalgamation of LSPs from remaining network nodes)
Node name LSP #
A #1 B #1 C #1 D #1 E #1
B #2
A #2
Copyright © 1998
Internet Routing Protocols: Fundamental Operation
24

Each router must do the following things to make it work:

1. Discover its neighbors and learn their network addresses.
2. Set the distance or cost metric to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to and receive packets from all other routers.
5. Compute the shortest path to every other router.

In effect, the complete topology is distributed to every router. Then Dijkstra's algorithm can be run at each router to find the shortest path to every other router. Link state routing is widely used in actual networks, so a few words about some example protocols are in order. Many ISPs use the **IS-IS** (**Intermediate System-Intermediate System**) link state protocol (Oran, 1990). It was designed for an early network called DECnet, later adopted by ISO for use with the OSI protocols and then modified to handle other protocols as well, most notably, IP. **OSPF** (**Open Shortest Path First**) is the other main link state protocol. It was designed by IETF several years after IS-IS and adopted many of the innovations designed for IS-IS. These innovations include a self-stabilizing method of flooding link state updates, the concept of a designated router on a LAN, and the method of computing and supporting path splitting and multiple metrics. As a consequence, there is very little difference between IS-IS and OSPF. The most important difference is that IS-IS can carry information about multiple network layer protocols at the same time (e.g., IP, IPX, and AppleTalk). OSPF does not have this feature, and it is an advantage in large multiprotocol environments.

ALGORITHM:

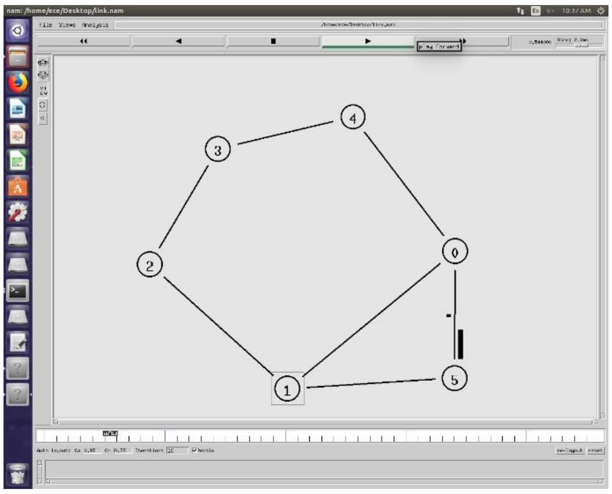
```
    Step 1 : Start.
    Step 2 : Create a simulator object.
    Step 3 : Configure the simulator to use Link state routing.
    Step 4 : Open the nam trace file.
    Step 5 : Open the output file.
    Step 6 : Define the finish procedure.
    Step 7 : Close the trace file.
    Step 8 : Call x-graph to display the result.
    Step 9 : Create 7 nodes.
    Step 10 : Create links between the nodes.
    Step 11 : Create a UDP agent to attach the node.
    Step 12 : Create a CBR traffic router and attach.
    Step 13 : Create a Null agent to the traffic sink.
    Step 14 : Connect the traffic source to the sink.
    Step 15 : Schedule the events for CBR agent.
    Step 16 : Stop.
```

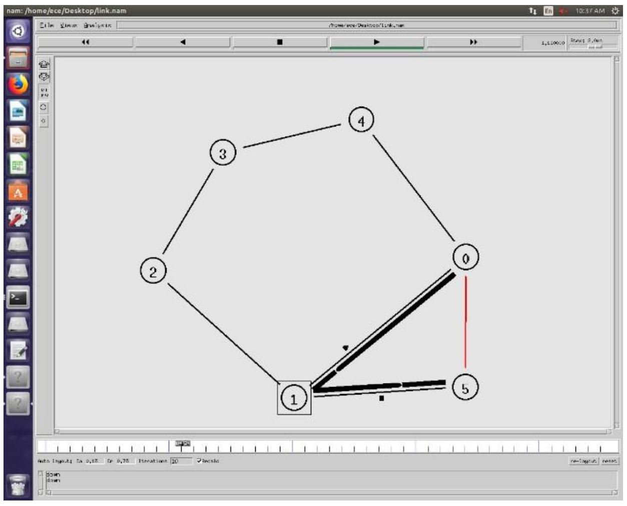
Link State Routing

```
set ns [new Simulator]

$ns rtproto LS

set nf [open link.nam w]
$ns namtrace-all $nf
set tf [open link.tr w]
$ns trace-all $tf
proc finish {} {
global ns nf tf
$ns flush-trace
close $nf
exec nam out.nam &
exit 0
}
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]

$ns duplex-link $n0 $n1 1Mb 10ms DropTail
$ns duplex-link $n0 $n5 1Mb 10ms DropTail
```

```
$ns duplex-link $n0 $n4 1Mb 10ms DropTail
$ns duplex-link $n1 $n5 1Mb 10ms DropTail
$ns duplex-link $n1 $n2 1Mb 10ms DropTail
$ns duplex-link $n2 $n3 1Mb 10ms DropTail
$ns duplex-link $n3 $n4 1Mb 10ms DropTail


set tcp0 [new Agent/TCP]
$tcp0 set class_ 1
$ns attach-agent $n0 $tcp0
set sink0 [new Agent/TCPSink]
$ns attach-agent $n5 $sink0
$ns connect $tcp0 $sink0
set cbr0 [new Application/Traffic/CBR]
$cbr0 set packetSize_ 500
$cbr0 set interval_ 0.01
$cbr0 attach-agent $tcp0
$ns at 0.5 "$cbr0 start"
$ns rtmodel-at 0.7 down $n0 $n5
$ns rtmodel-at 2.5 up $n0 $n5
$ns at 4.5 "$cbr0 stop"
$ns at 5.0 "finish"
$ns run
```

OUTPUT:





RESULT:
Thus the Link state routing was implemented and analyzed using NS2 program.