Experiment **#1**

SINE WAVE GENERATION

<u>AIM</u>

Write a program to generate sine wave form with input amplitude and frequency.

<u>THEORY</u>

The sine wave is the most fundamental form of a periodic analog signal. Each cycle consists of a single arc above the time axis followed by a single arc below it.

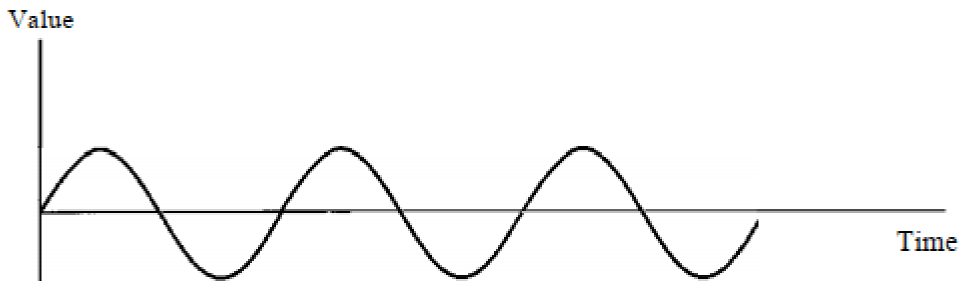


Figure 1 A sine wave

A sine wave can be represented by three parameters: the peak amplitude, the frequency, and the phase. These three parameters fully describe a sine wave.

The peak amplitude of a signal is the absolute value of its highest intensity, proportional to the energy it carries.

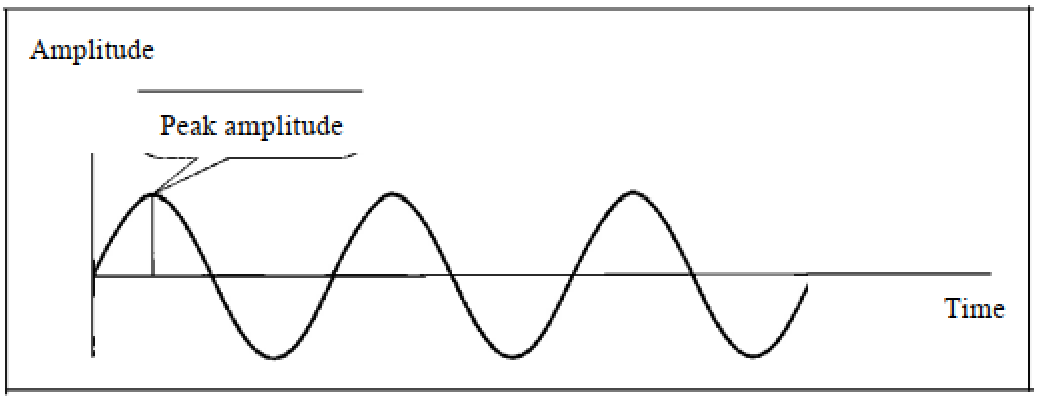


Figure 2 Amplitude

Period refers to the amount of time, in seconds, a signal needs to complete 1 cycle. Frequency refers to the number of periods in 1s.

$$f = \frac{1}{T} \quad \text{and} \quad T = \frac{1}{f}$$

Frequency and period are the inverse of each other.

Period is formally expressed in seconds. Frequency is formally expressed in Hertz (Hz), which is cycle per second.

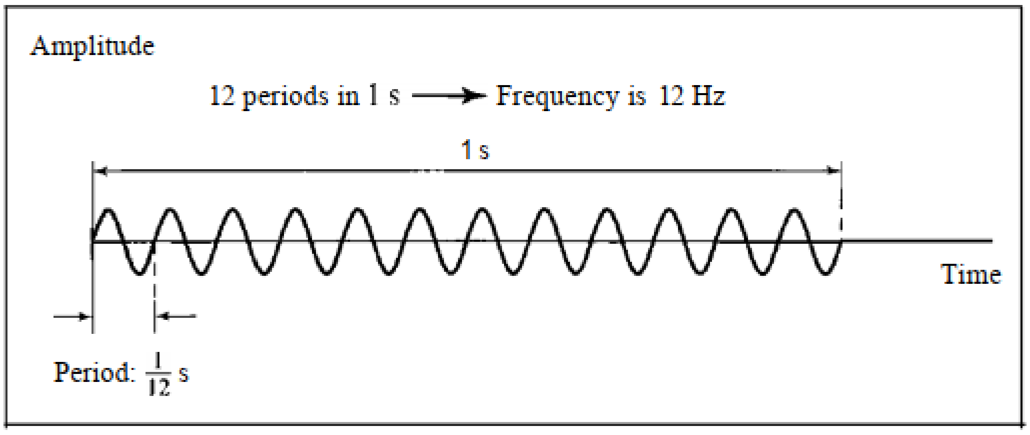
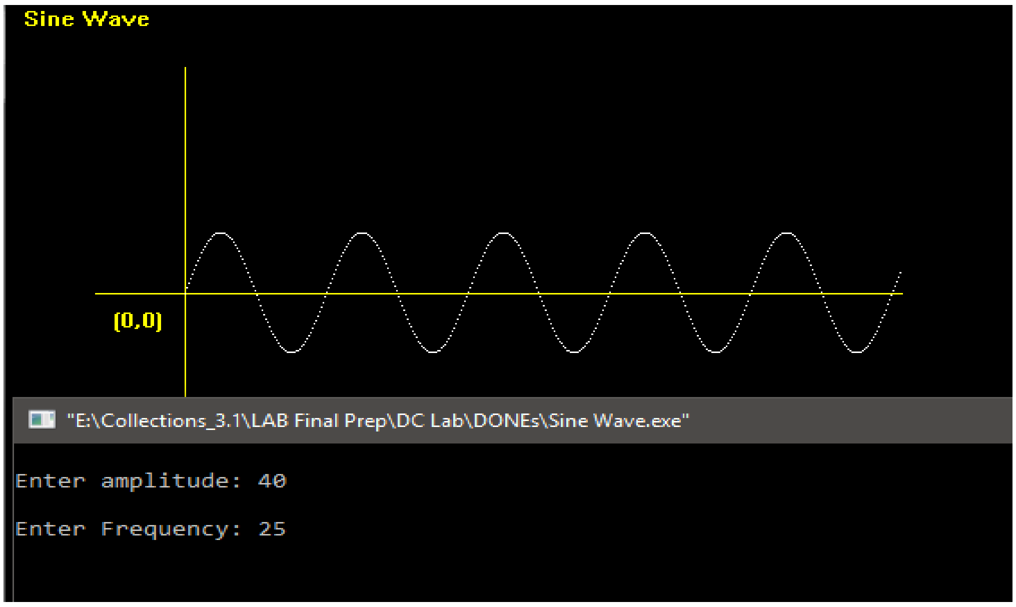


Figure 3 Frequency and Period

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
#include<dos.h>
using namespace std;

void draw_axis(){
    line ( 50, 200, 500, 200 );
    line ( 100, 50, 100, 400 );
    outtextxy ( 60, 210, "(0,0)" );
}
int main(){
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );
    setcolor ( YELLOW );
    while ( 1 ){
        int a, f;
        cleardevice();
        outtextxy(10,10, "Sine Wave");
        draw_axis();
        cout << "\nEnter amplitude: ";
        cin >> a;
        cout << "\nEnter Frequency: ";
        cin >> f;
        //draw_axis();
        for ( int t = 0; t < 400; t++ ){
            //y(t) = a*sin(2*PI*f*t);
            double y = a * sin ( 2 * 3.14 * f * t );
            delay ( 5 );
            putpixel ( t + 100, y + 200, WHITE );
        }
        getch();
    }
    return 0;
}
```

OUTPUT


Sine Wave
(0,0)
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Sine Wave.exe"

```
Enter amplitude: 40

Enter Frequency: 25
```

RESULT

Sine wave generation code is written, executed and the output is verified.

Experiment **#2**

COSINE WAVE GENERATION

AIM

Write a program to generate cosine wave form with input amplitude and frequency.

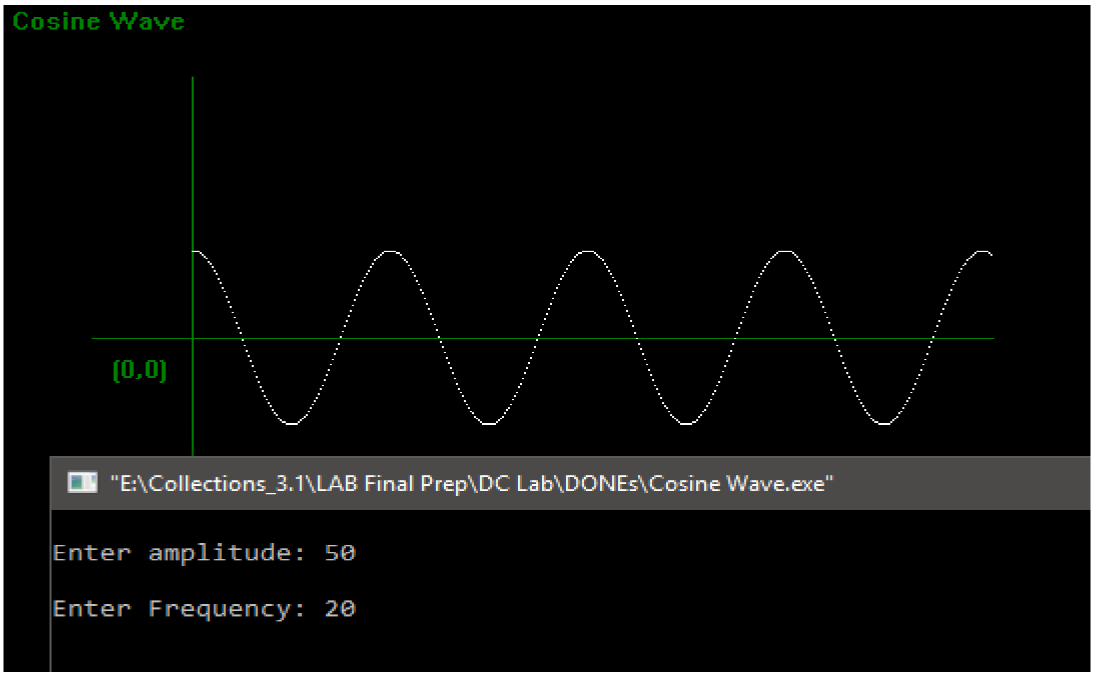
THEORY

The cosine waveform is a periodic analog signal. It is a signal waveform with a shape identical to that of a sine wave , except each point on the cosine wave occurs exactly 1/4 cycle earlier than the corresponding point on the sine wave. A cosine wave and its corresponding sine wave have the same frequency, but the cosine wave leads the sine wave by 90 degrees of phase.

CODE

```cpp
#include<bits/stdc++.h>
#include<graphics.h>
#include<dos.h>
using namespace std;

void draw_axis(){
    line ( 50, 200, 500, 200 );
    line ( 100, 50, 100, 400 );
    outtextxy ( 60, 210, "(0,0)" );
}
int main(){
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );
    setcolor ( GREEN );
    while ( 1 ){
        int a, f;
        cleardevice();
        outtextxy(10,10, "Cosine Wave");
        draw_axis();
        cout << "\nEnter amplitude: ";
        cin >> a;
        cout << "\nEnter Frequency: ";
        cin >> f;
        //draw_axis();
        for ( int t = 0; t < 400; t++ ){
            double y = ( -a ) * cos ( 2 * 3.14 * f * t );
            delay ( 5 );
            putpixel ( t + 100, y + 200, WHITE );
        }
        getch();
    }
    return 0;
}
```

OUTPUT


Cosine Wave
(0,0)
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Cosine Wave.exe"

```
Enter amplitude: 50

Enter Frequency: 20
```

RESULT

Cosine wave generation code is written, executed and the output is verified.

Experiment **#3**

SINE WAVE GENERATION WITH BINARY INPUT

AIM

Write a program to generate sine wave form with binary input.

THEORY

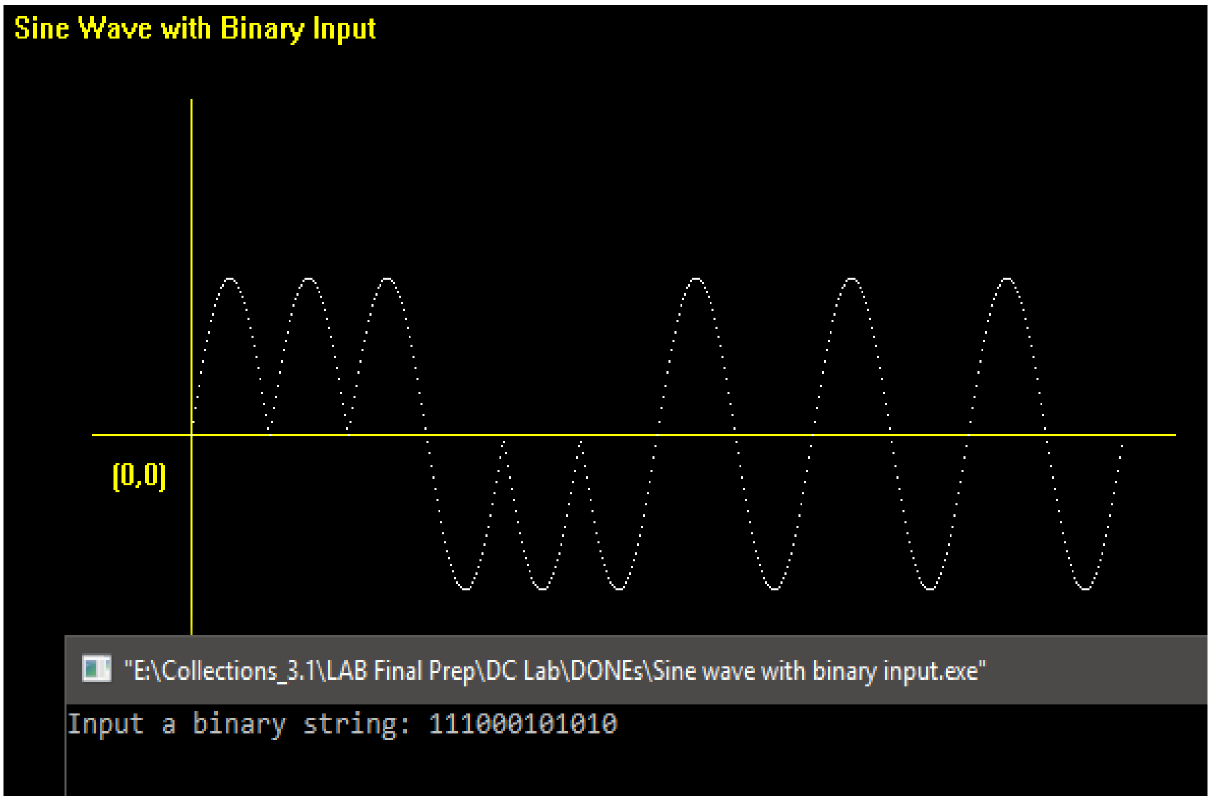
When the input is 1 then the amplitude is increasing. On the contrary if the input is 0 then the amplitude is decreasing.

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
#include<dos.h>
using namespace std;
float a,f;
int x;

void draw_axis()
{
    line ( 50, 200, 600, 200 );
    line ( 100, 50, 100, 400 );
    outtextxy ( 60, 210, "(0,0)" );
}

bool check_string(string str)
{
    int sz = str.size();
    for(int i = 0; i<sz; i++)
    {
        if(str[i]!='0' && str[i]!='1')
            return true;
    }
    return false;
}
void fun(int t)
{
    //y(t) = a*sin(2*PI*f*t);
    double y = a * sin ( 2 * 3.14 * f * t );
    delay ( 5 );
    putpixel ( x + 100, y + 200, WHITE );
    x++;

}
void draw_upWave()
{
    for ( int t = 0; t < 40; t++ )
        fun(t);
}
void draw_downWave()
{
    for ( int t = 40; t <79; t++ )
            fun(t);
}

int main()
{
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );
    setcolor ( YELLOW );
    while ( 1 )
    {
        string str;
        cleardevice();
        outtextxy ( 10, 10, "Sine Wave with Binary Input" );
        draw_axis();
        a = 70, f = 25;
        cout << "Input a binary string: ";
        cin >> str;
        if(check_string(str))
        {
            cout<< "Invalid string input! Enter only binary
digits\n\n";
            continue;
        }
        int sz = str.size();
        x = 0;
        for ( int i = 0; i < sz; i++ )
        {
            char c = str[i];
            if ( c == '0' )
                draw_downWave();

            else if ( c == '1' )
                draw_upWave();
        }
        cout<<endl;
```

```
          getch();
     }
     return 0;
}
```

OUTPUT


Sine Wave with Binary Input
(0,0)
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Sine wave with binary input.exe"

```
Input a binary string: 111000101010
```

RESULT

Sine wave generation with binary input code is written, executed and the output is verified.

Experiment **#4**

COSINE WAVE GENERATION WITH BINARY INPUT

AIM

Write a program to generate cosine wave form with binary input.

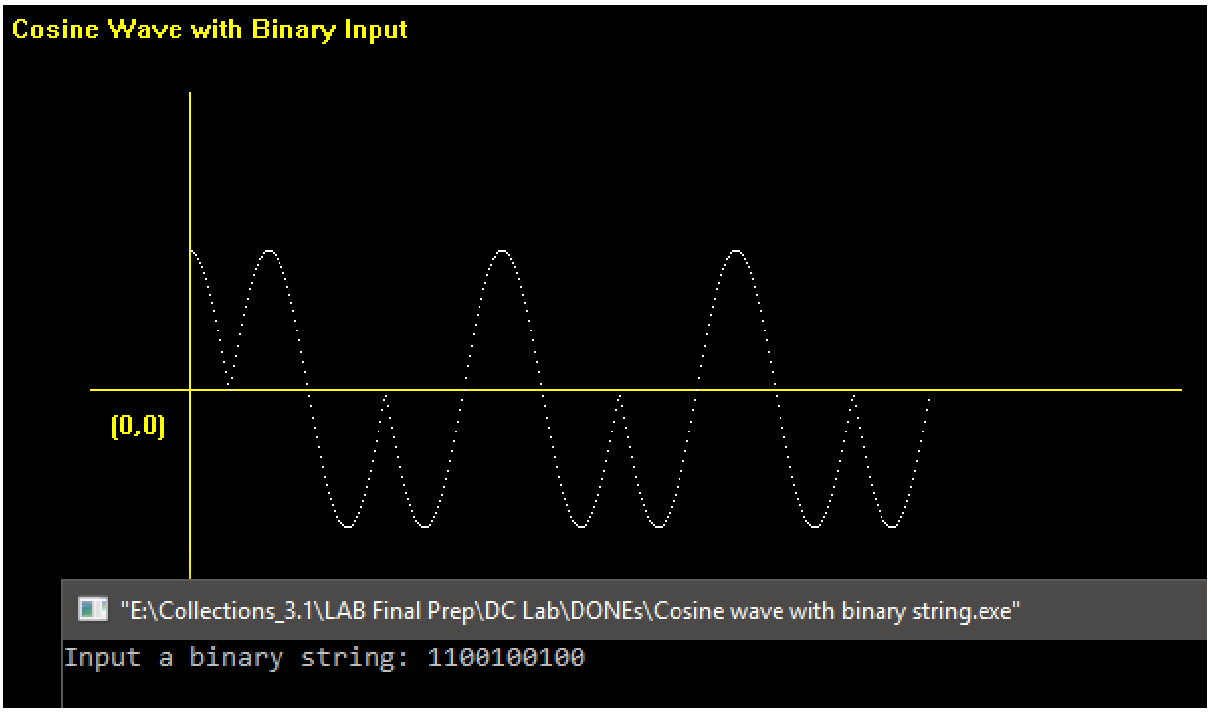
THEORY

When the input is 1 then the amplitude is increasing. On the contrary if the input is 0 then the amplitude is decreasing.

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
#include<dos.h>
using namespace std;
float a, f;
int x;
string str;

void draw_axis()
{
    line ( 50, 200, 600, 200 );
    line ( 100, 50, 100, 400 );
    outtextxy ( 60, 210, "(0,0)" );
}
bool check_string ( string str )
{
    int sz = str.size();
    for ( int i = 0; i < sz; i++ ){
        if ( str[i] != '0' && str[i] != '1' )
            return true;
    }
    return false;
}
void fun(int t)
{
    //y(t) = a*sin(2*PI*f*t);
    double y = ( a ) * sin ( 2 * 3.14 * f * t );
    delay ( 5 );
    putpixel ( x + 100, y + 200, WHITE );
    x++;
}
void draw_upWave ( int m )
{
    if ( m == 0 ){
        for ( int t = 20; t < 40; t++ )
            fun(t);
    }
    else{
        for ( int t = 0; t < 40; t++ )
            fun(t);
    }
}
void draw_downWave ( int m )
{
    if ( m == 0 ){
        for ( int t = 60; t < 79; t++ )
            fun(t);
    }
    else{
        for ( int t = 40; t < 79; t++ )
            fun(t);
    }
}

int main()
{
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );
    setcolor ( YELLOW );
    while ( 1 ){
        cleardevice();
        outtextxy ( 10, 10, "Cosine Wave with Binary Input" );
        draw_axis();
        a = 70, f = 25;
        cout << "Input a binary string: ";
        cin >> str;
        if ( check_string ( str ) ){
            cout<< "Invalid string input! Enter only binary
digits\n\n";
            continue;
        }
        int sz = str.size();
        x = 0;
        for ( int i = 0; i < sz; i++ ){
            char c = str[i];
            if ( c == '0' )
                draw_downWave ( i );

            else if ( c == '1' )
                draw_upWave ( i );
        }
        cout << endl;
        getch();
    }
    return 0;
}
```

OUTPUT


Cosine Wave with Binary Input
(0,0)
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Cosine wave with binary string.exe"

```
Input a binary string: 1100100100
```

RESULT

Cosine wave generation with binary input code is written, executed and the output is verified.

Experiment #5

TWO LEVEL DIGITAL SIGNAL GENERATION

AIM

Write a program to generate two level digital signal waveform with binary input.

THEORY

Information can also be represented by a digital signal. For example, a 1 can be encoded as a positive voltage and a 0 as zero voltage. In two level digital signal there are two levels of transition. Only 1 bit per level is sent.

In general, if a signal has L levels, each level needs log2 L bits. For this reason, we can send log24 = 2 bits in part b.

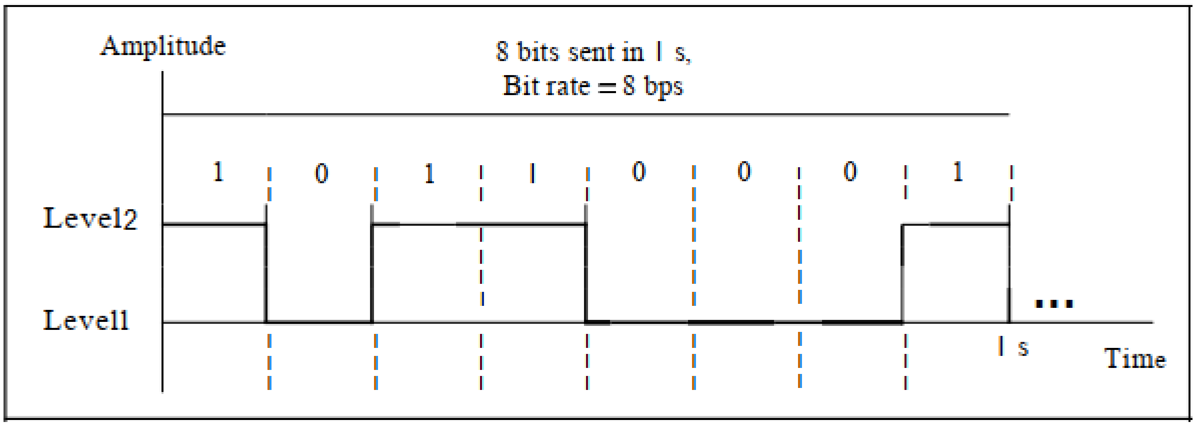


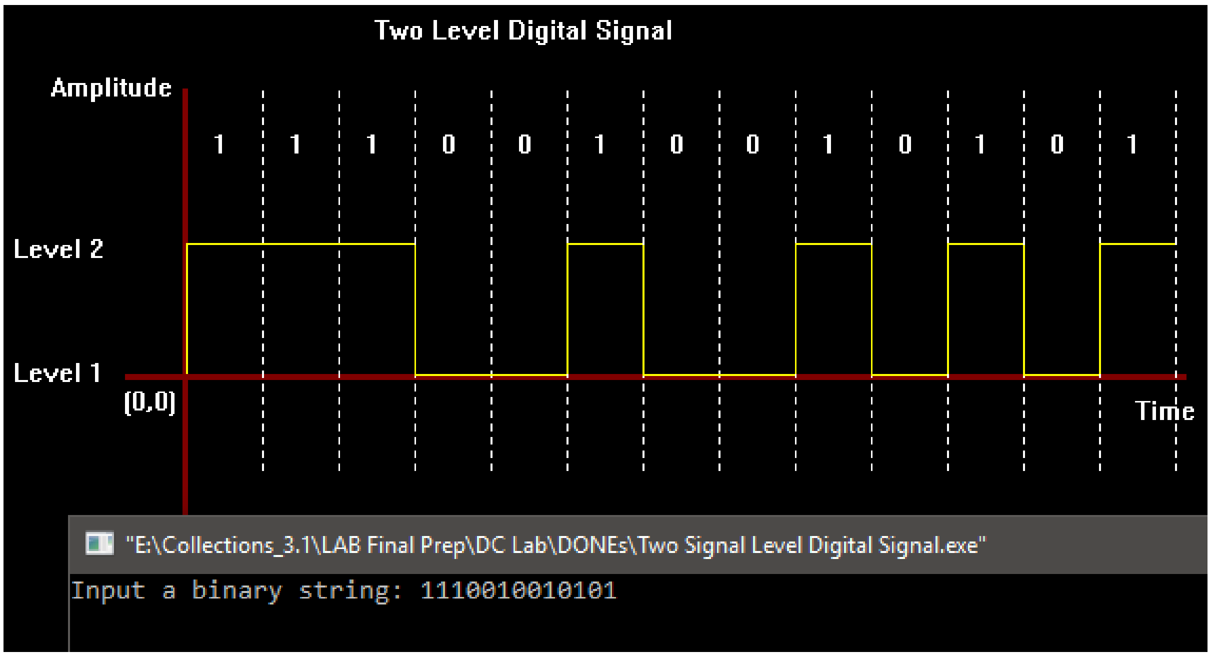
Figure 4 Two level Digital Signal

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define dot_color YELLOW
#define deri 5
bool khara = 0;
int x, y, px, py;

void decorate()
```

```
{
    setfillstyle ( SOLID_FILL, BLACK );
    floodfill ( 100, 100, BLACK );
    outtextxy(200, 10, "Two Level Digital Signal");
    outtextxy(68, 205, "(0,0)");
    outtextxy(30, 40, "Amplitude");
    outtextxy(600, 210, "Time");
    outtextxy(10,190, "Level 1");
    outtextxy(10,125, "Level 2");
    return;
}

void drawAxis()
{
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(RED);
    line ( 70, 200, 625, 200 );     // x axis
    line ( 100, 50, 100, 400 );     // y axis

    int z = 40;

    for(int i = 1; i<=13; i++)
    {
        setcolor(WHITE);
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        line(101+z, 50, 101+z, 250);
        z+=40;
    }
    return;
}

bool checkString ( string str )
{
    int sz = str.size();
    for ( int i = 0; i < sz; i++ )
    {
        if ( str[i] != '0' && str[i] != '1' )
            return true;
    }
    return false;
}

void jump(int z)
{
    for ( int i = 1; i < 70; i++ )
    {
        int m = z*i;
        putpixel ( x+1, y + m, dot_color );
        py = y + m;
        delay ( deri );
    }
    y = py;
    return;
}

void generateSignal(char c) {
    int z;
    if(khara == 0 && c=='1') {
        z = -1 , khara = 1;
        jump(z);
    }
    else if(khara==1 && c=='0') {
        z = 1 , khara = 0;
        jump(z);
    }
    for ( int i = 1; i <= 40; i++ ) {
        putpixel ( x + i, y, dot_color );
        px = x + i;
        delay ( deri );
    }
    x = px;
    setcolor(WHITE);
    char A[2];
    sprintf(A,"%c",c);
    outtextxy(x-25, 70, A);
    return;
}

void processSignal(string str){
    int sz = str.size();
    x = 100, y = 199, px = 100, py = 199, khara = 0;
    for ( int i = 0; i < sz; i++ ) {
        char c = str[i];
        generateSignal(c);
    }
    return;
}

int main()
{
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );

    while(1) {
        cleardevice();
        decorate();
        drawAxis();
        string str;
        cout << "Input a binary string: ";
        cin >> str;
        if ( checkString ( str ) ) {
            cout << "Invalid string input! Enter only binary 
digits\n\n";
            continue;
        }
        processSignal(str);
        getch();
    }
    return 0;
}
```

OUTPUT


Two Level Digital Signal
Amplitude
1
1
1
0
0
1
0
0
1
0
1
0
1
Level 2
Level 1
(0,0)
Time
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Two Signal Level Digital Signal.exe"

```
Input a binary string: 1110010010101
```

RESULT

Two level digital signal generation code is written, executed and the output is verified.

Experiment **#6**

FOUR LEVEL DIGITAL SIGNAL GENERATION

AIM

Write a program to generate four level digital signal waveform with binary input.

THEORY

In four level digital signal there are four levels of transition. 2 bit per level is sent. So there are $2^2 = 4$ are states – 00, 01, 10 & 11.

In general, if a signal has L levels, each level needs log2 L bits. For this reason, we can send log24 = 2 bits in part b.

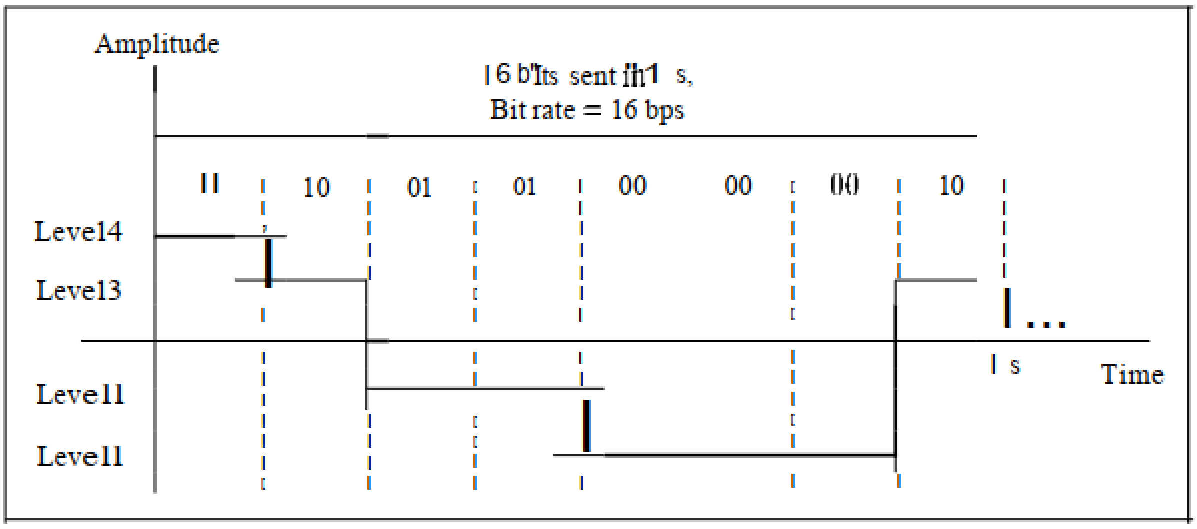


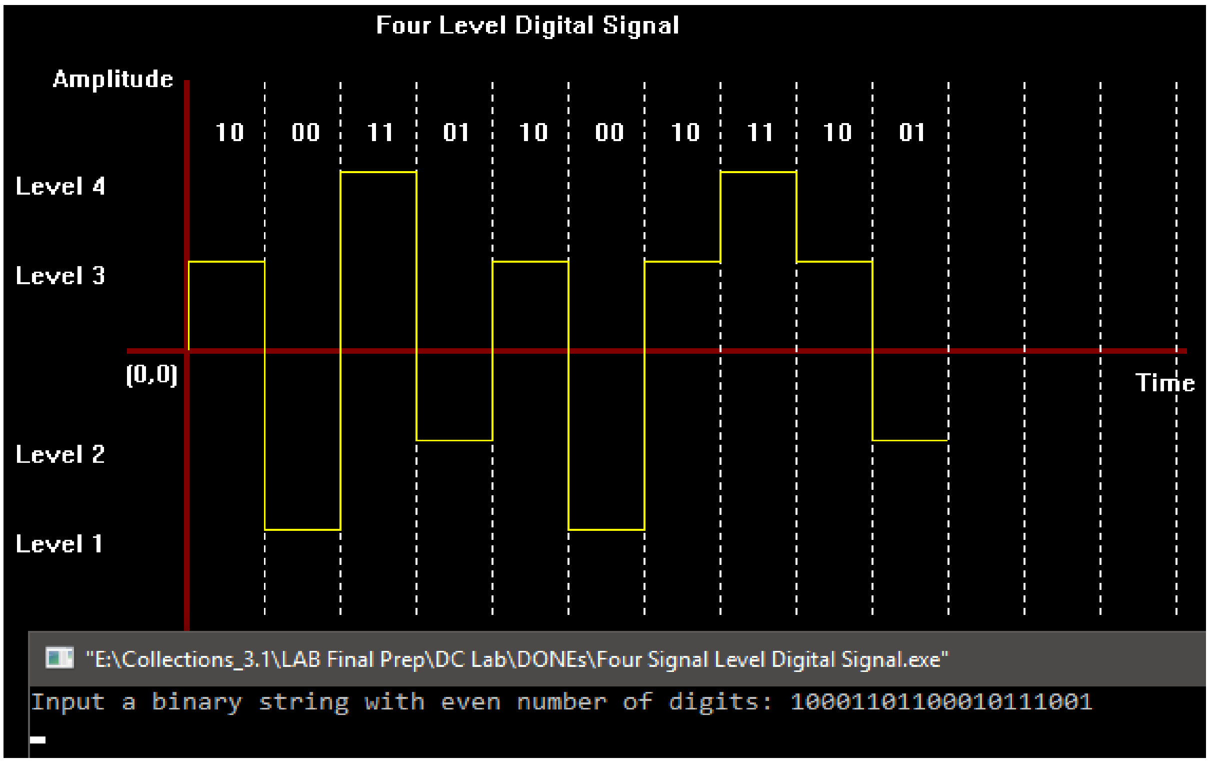
Figure 5 Four level Digital Signal

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define dot_color YELLOW
#define deri 5
int x, y, px, py, mx, my;
char cx='a', cy='a';
bool flag = 0;
```

```
void decorate()
{
    setfillstyle ( SOLID_FILL, BLACK );
    floodfill ( 100, 100, BLACK );
    outtextxy(200, 10, "Four Level Digital Signal");
    outtextxy(68, 205, "(0,0)");
    outtextxy(30, 40, "Amplitude");
    outtextxy(600, 210, "Time");
    outtextxy(10,300, "Level 1");
    outtextxy(10,250, "Level 2");
    outtextxy(10,150, "Level 3");
    outtextxy(10,100, "Level 4");
    return;
}

void drawAxis()
{
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(RED);
    line ( 70, 200, 625, 200 );     // x axis
    line ( 100, 50, 100, 400 );     // y axis
    int z = 40;
    for(int i = 1; i<=13; i++)
    {
        setcolor(WHITE);
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        line(101+z, 50, 101+z, 350);
        z+=40;
    }
    return;
}

void fun(int dir, char c1, char c2)
{
    int i = y;
    while(i!=my)
    {
        putpixel(x+1,i,dot_color);
        delay(deri);

        if ((cx=='0' && cy=='0') && (c1=='0' && c2=='1') )
            dir = -1;
        if( (cx=='1' && cy=='1') && (c1=='1' && c2=='0') )
            dir = 1;
        if(dir==1) i++;
        else if(dir==-1) i--;
        y = i;
    }
    return;
}
void generateSignal(char c1, char c2)
{
    int dir;
    if(c1=='1' && c2=='1')
    {
        dir = -1;
        my = 100;
        fun(dir, c1, c2);
    }
    else if(c1=='1' && c2=='0')
    {
        dir = -1;
        my = 150;
        fun(dir, c1, c2);
    }
    else if(c1=='0' && c2=='1')
    {
        dir = 1;
        my = 250;
        fun(dir, c1, c2);
    }
    else if(c1=='0' && c2=='0')
    {
        dir = 1;
        my = 300;
        fun(dir, c1, c2);
    }
    for(int i = 1; i<41; i++)
    {
        putpixel(x+i, y, dot_color);
        px = x+i;
        delay(deri);
    }
    x = px;
    cx = c1, cy = c2;
    setcolor(WHITE);
    char A[4];
    sprintf(A,"%c%c",c1,c2);
    outtextxy(x-25, 70, A);
    return;
}
void processSignal(string str)
{
    int sz = str.size();
    x = 100, y = 199, px = 100, py = 199;
    for ( int i = 0; i < sz; i+=2 )
    {
        char c1 = str[i];
        char c2 = str[i+1];
        generateSignal(c1, c2);
    }
    return;
}
bool checkString ( string str )
{
    int sz = str.size();
    if(sz%2)
        return true;
    for ( int i = 0; i < sz; i++ )
    {
        if ( str[i] != '0' && str[i] != '1' )
            return true;
    }
    return false;
}
int main()
{
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );

    while(1)
    {
        cleardevice();
        decorate();
        drawAxis();

        string str;
        cout << "Input a binary string with even number of digits:
";
        cin >> str;
        if ( checkString ( str ) )
        {
            cout << "Invalid string input! Enter only binary digits
with even length\n\n";
            continue;
        }
        processSignal(str);
        getch();
    }
    return 0;
}
```

OUTPUT



```
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Four Signal Level Digital Signal.exe"
Input a binary string with even number of digits: 10001101100010111001
_
```

RESULT

Four level digital signal generation code is written, executed and the output is verified.

Experiment #7

DIGITAL SIGNAL BIPOLAR ENCODING

Aim

Write a program to generate digital signal with bipolar encoding.

THEORY

Line coding is the process of converting digital data to digital signals. Line coding converts a sequence of bits to a digital signal. At the sender, digital data are encoded into a digital signal; at the receiver, the digital data are recreated by decoding the digital signal.

In bipolar encoding (sometimes called ***multilevel binary***), there are three voltage levels: positive, negative, and zero. The voltage level for one data element is at zero, while the voltage level for the other element alternates between positive and negative. A neutral zero voltage represents binary 0. Binary 1s are represented by alternating positive and negative voltages.

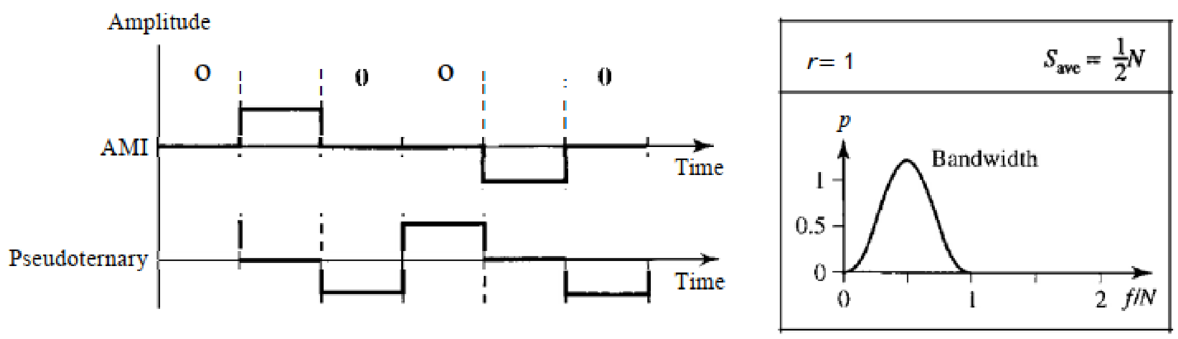


Figure 6 Bipolar Coding Scheme

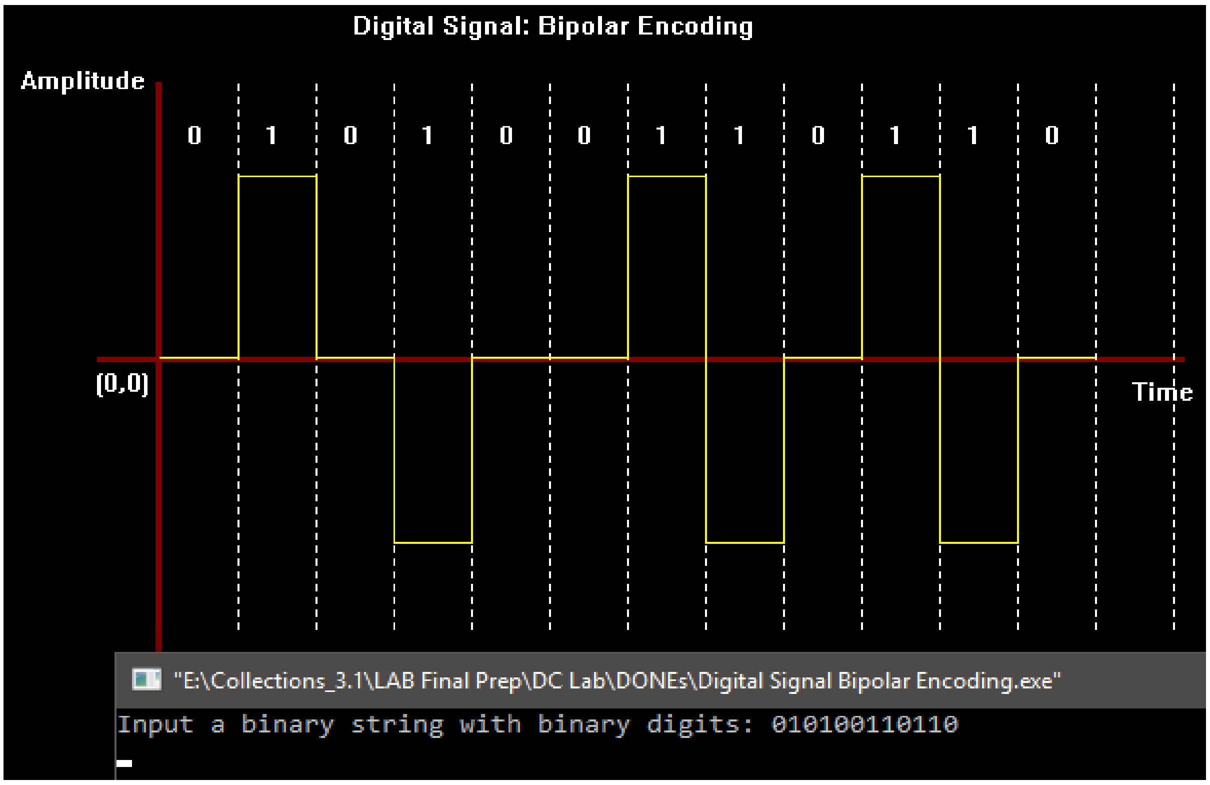
CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define dot_color YELLOW
#define deri 5
int x, y, px, py, mx, my;
```

```
char cx='a', cy='a';
bool counter;
bool online;

void decorate(){
    setfillstyle ( SOLID_FILL, BLACK );
    floodfill ( 100, 100, BLACK );
    outtextxy(200, 10, "Digital Signal: Bipolar Encoding");
    outtextxy(68, 205, "(0,0)");
    outtextxy(30, 40, "Amplitude");
    outtextxy(600, 210, "Time");
    return;
}
void drawAxis(){
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(RED);
    line ( 70, 200, 625, 200 );     // x axis
    line ( 100, 50, 100, 400 );     // y axis
    int z = 40;
    for(int i = 1; i<=13; i++){
        setcolor(WHITE);
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        line(101+z, 50, 101+z, 350);
        z+=40;
    }
    return;
}

void fun(int dir){
    int i = y;
    while(i!=my){
        putpixel(x+1, i, dot_color);
        delay(deri);
        if(dir==1) i++;
        else if(dir==-1) i--;
        y = i;
    }
    return;
}

void generateSignal(char c1){
    int dir;

    if(c1=='0')
    {
        if(online)
        {
            for(int i = 1; i<41; i++)
            {
                putpixel(x+i, 199, dot_color);
                px = x+i;
                delay(deri);
            }
        }
        else
        {
            my = 199;
            if(dir==-1) dir=1;
            else if(dir==1) dir = -1;
            fun(dir);
        }
    }

    if(c1=='1')
    {
        online = 0;
        if(!counter)
        {
            dir = -1;
            my = 100;
            fun(dir);
            counter = 1;
        }
        else
        {
            dir = 1;
            my = 300;
            fun(dir);
            counter = 0;
        }
    }

    if(!online) {
        for(int i = 1; i<41; i++) {
            putpixel(x+i, y, dot_color);
            px = x+i;
            delay(deri);
        }
    }
    x = px;
    cx = c1;
    setcolor(WHITE);
    char A[2];
    sprintf(A,"%c", c1);
    outtextxy(x-25, 70, A);
    return;
}
```

```
void processSignal(string str)
{
    int sz = str.size();
    x = 100, y = 199, px = 100, py = 199;
    counter=0, online = 1;
    for ( int i = 0; i < sz; i++ )
    {
        char c1 = str[i];
        generateSignal(c1);
    }
    return;
}
bool checkString ( string str )
{
    int sz = str.size();
    for ( int i = 0; i < sz; i++ )
    {
        if ( str[i] != '0' && str[i] != '1' )
            return true;
    }
    return false;
}
int main()
{
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );
    while(1)
    {
        cleardevice();
        decorate();
        drawAxis();

        string str;
        cout << "Input a binary string with binary digits: ";
        cin >> str;
        if ( checkString ( str ) )
        {
            cout << "Invalid string input! Enter only binary
digits\n\n";
            continue;
        }
        processSignal(str);
        getch();
    }
    return 0;
}
```

OUTPUT



```
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Digital Signal Bipolar Encoding.exe"
Input a binary string with binary digits: 010100110110
```

RESULT

Bipolar encoding scheme digital signal generation code is written, executed and the output is verified.

Experiment **#8**	NRZ-LEVEL OF DIGITAL SIGNAL

AIM

Write a program to generate digital signal with NRZ-Level encoding.

THEORY

In polar schemes, the voltages are on both sides of the time axis. For example, the voltage level for 0 can be positive and the voltage level for 1 can be negative.

In polar NRZ encoding, we use two levels of voltage amplitude. NRZ-L (**NRZ-Level**), the level of the voltage determines the value of the bit.

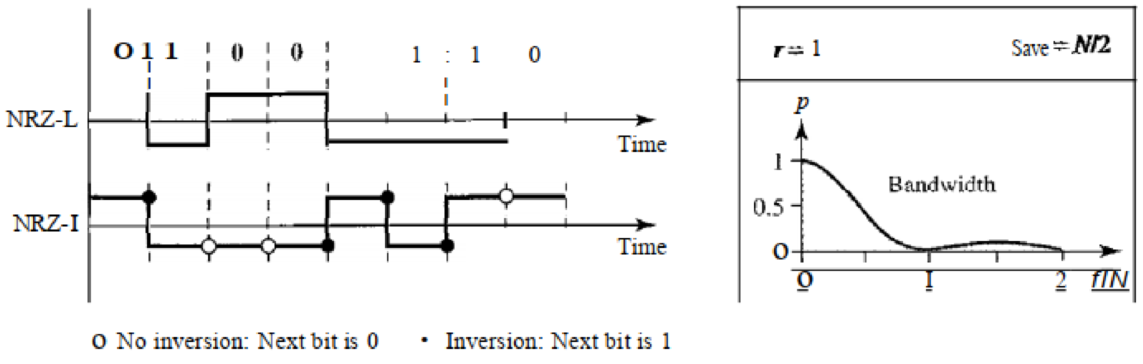
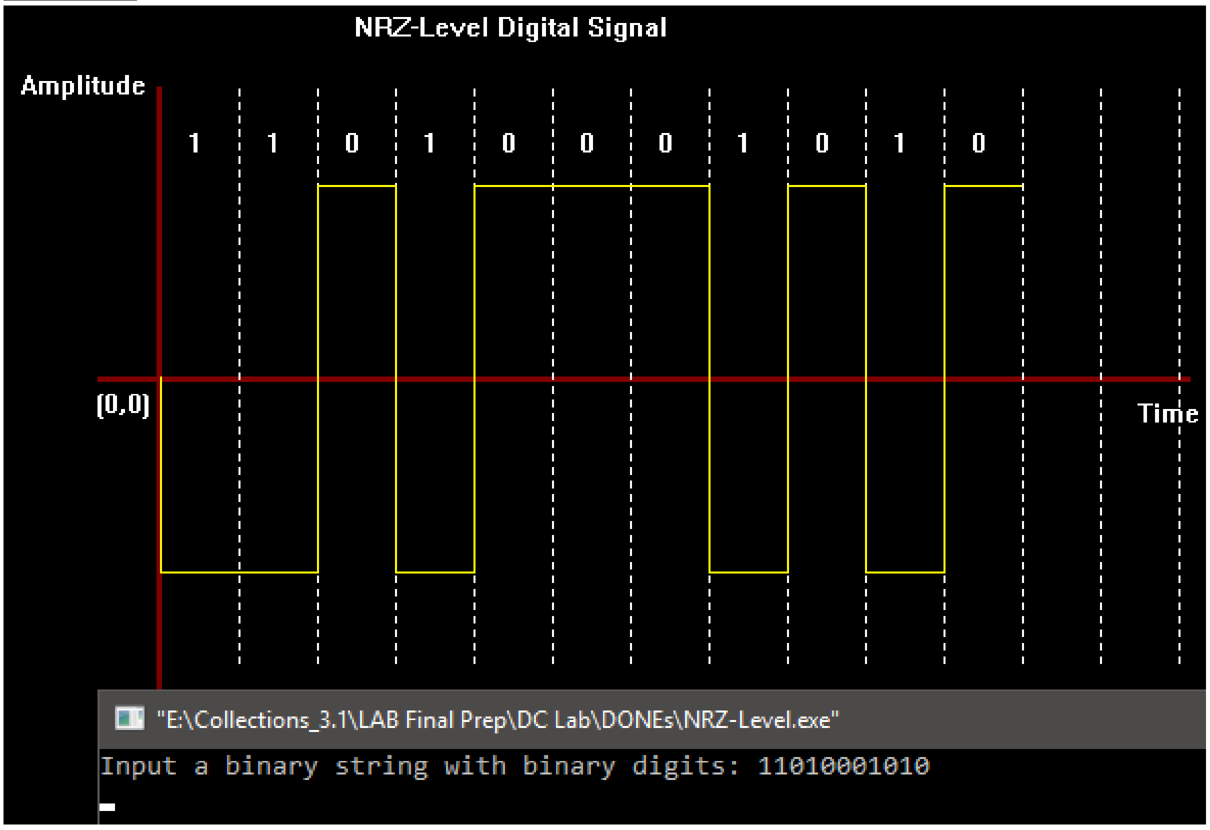


Figure 7 NRZ-Level Encoding

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define dot_color YELLOW
#define deri 5
int x, y, px, py, mx, my;
char cx='a', cy='a';

void decorate(){
    setfillstyle ( SOLID_FILL, BLACK );
    floodfill ( 100, 100, BLACK );
    outtextxy(200, 10, "NRZ-Level Digital Signal");
    outtextxy(68, 205, "(0,0)");
    outtextxy(30, 40, "Amplitude");
    outtextxy(600, 210, "Time");
    return;
}
void drawAxis(){
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(RED);
    line ( 70, 200, 625, 200 );     // x axis
    line ( 100, 50, 100, 400 );     // y axis
    int z = 40;
    for(int i = 1; i<=13; i++){
        setcolor(WHITE);
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        line(101+z, 50, 101+z, 350);
        z+=40;
    }
    return;
}

void fun(int dir)
{
    int i = y;
    while(i!=my) {
        putpixel(x+1,i,dot_color);
        delay(deri);
        if(dir==1) i++;
        else if(dir==-1) i--;
        y = i;
    }
    return;
}

void generateSignal(char c1)
{
    int dir;
    if(c1=='0'){
        dir = -1;
        my = 100;
        fun(dir);
    }
    else if(c1=='1'){
        dir = 1;
        my = 300;
        fun(dir);
    }
    for(int i = 1; i<41; i++){
        putpixel(x+i, y, dot_color);
        px = x+i;
        delay(deri);
    }
    x = px;
    cx = c1;
    setcolor(WHITE);
    char A[2];
    sprintf(A,"%c",c1);
    outtextxy(x-25, 70, A);
    return;
}
void processSignal(string str)
{
    int sz = str.size();
    x = 100, y = 199, px = 100, py = 199;
    for ( int i = 0; i < sz; i++ ){
        char c1 = str[i];
        generateSignal(c1);
    }
    return;
}

bool checkString ( string str )
{
    int sz = str.size();
    for ( int i = 0; i < sz; i++ ){
        if ( str[i] != '0' && str[i] != '1' )
            return true;
    }
    return false;
}

int main()
{
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );
    while(1){
        cleardevice();
        decorate();
        drawAxis();
        string str;
        cout << "Input a binary string with binary digits: ";
        cin >> str;
        if ( checkString ( str ) ){
            cout << "Invalid string input! Enter only binary 
digits\n\n";
            continue;
        }
        processSignal(str);
        getch();
    }
    return 0;
}
```

OUTPUT



```
Input a binary string with binary digits: 11010001010
```

RESULT

Digital signal generation code with NRZ-Level encoding is written, executed and the output is verified.

Experiment **#9**	NRZ-INVERT OF DIGITAL SIGNAL

AIM

Write a program to generate digital signal with NRZ-Invert encoding.

THEORY

In polar schemes, the voltages are on both sides of the time axis. For example, the voltage level for 0 can be positive and the voltage level for 1 can be negative.

In polar NRZ encoding, we use two levels of voltage amplitude. NRZ-I (NRZ-Invert), the change or lack of change in the level of the voltage determines the value of the bit. If there is no change, the bit is 0; if there is a change, the bit is 1.

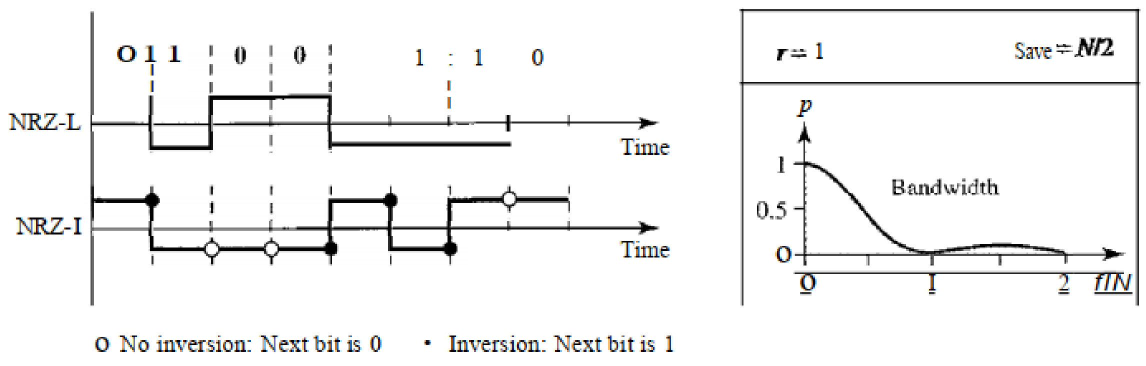


Figure 8 NRZ-Invert Encoding

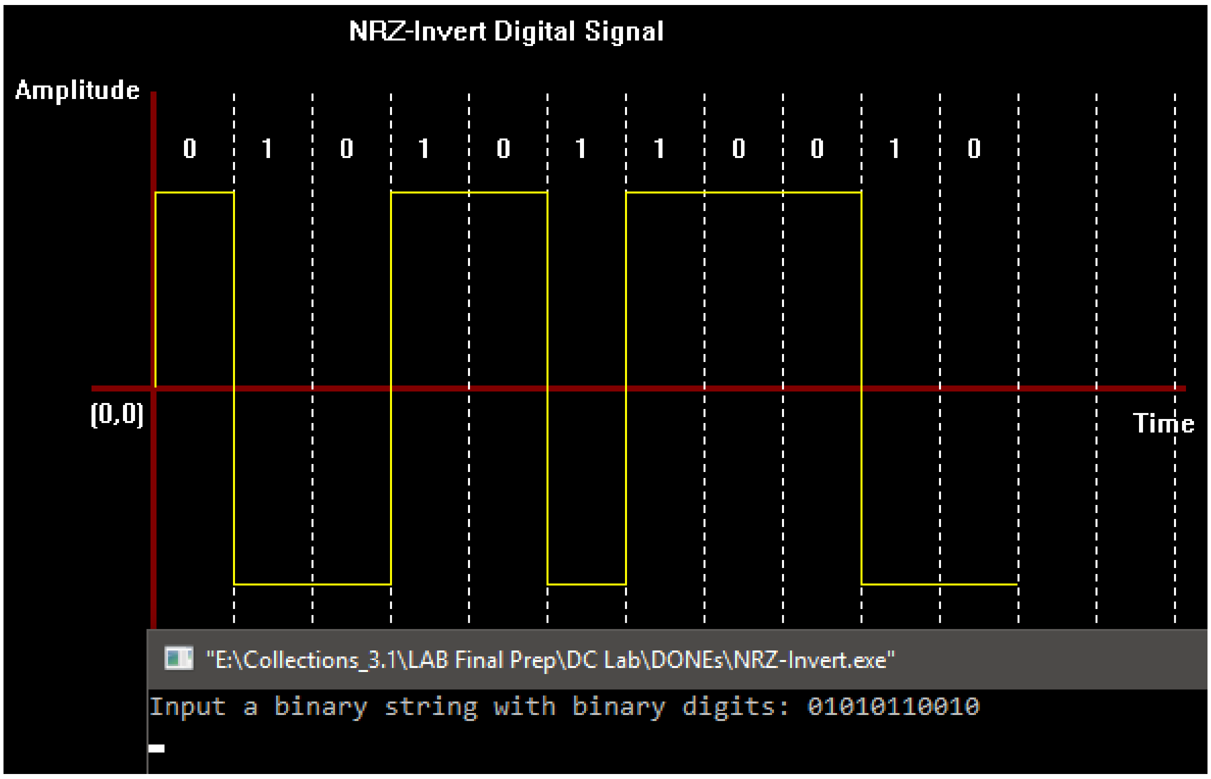
CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define dot_color YELLOW
#define deri 5
int x, y, px, py, mx, my, dir;
char cx='a', cy='a';
```

```
void decorate()
{
    setfillstyle ( SOLID_FILL, BLACK );
    floodfill ( 100, 100, BLACK );
    outtextxy(200, 10, "NRZ-Invert Digital Signal");
    outtextxy(68, 205, "(0,0)");
    outtextxy(30, 40, "Amplitude");
    outtextxy(600, 210, "Time");
    return;
}

void drawAxis()
{
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(RED);
    line ( 70, 200, 625, 200 );     // x axis
    line ( 100, 50, 100, 400 );     // y axis
    int z = 40;
    for(int i = 1; i<=13; i++)
    {
        setcolor(WHITE);
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        line(101+z, 50, 101+z, 350);
        z+=40;
    }
    return;
}

void fun(int dir, char c1)
{
    int i = y;
    while(i!=my)
    {
        putpixel(x+1,i,dot_color);
        delay(deri);

        if(dir==1) i++;
        else if(dir==-1) i--;
        y = i;
    }
    return;
}

void generateSignal(char c1)
{

    if(c1=='1')
    {
        if(dir==-1)
        {
            dir = 1;
            my = 300;
        }
        else
        {
            dir = -1;
            my = 100;
        }
    }
    fun(dir, c1);
    for(int i = 1; i<41; i++)
    {
        putpixel(x+i, y, dot_color);
        px = x+i;
        delay(deri);
    }
    x = px;
    cx = c1;
    setcolor(WHITE);
    char A[2];
    sprintf(A,"%c",c1);
    outtextxy(x-25, 70, A);
    return;
}

void processSignal(string str)
{
    int sz = str.size();
    x = 100, y = 199, px = 100, py = 199, my = 100, dir = -1;
    for ( int i = 0; i < sz; i++ )
    {
        char c1 = str[i];
        generateSignal(c1);
    }
    return;
}

bool checkString ( string str )
{
    int sz = str.size();

    for ( int i = 0; i < sz; i++ )
    {
        if ( str[i] != '0' && str[i] != '1' )
            return true;
    }
    return false;
}
```

```
int main()
{
    int gd = DETECT, gm;
    initgraph ( &gd, &gm, "C:\\TC\\BGI" );

    while(1)
    {
        cleardevice();
        decorate();
        drawAxis();

        string str;
        cout << "Input a binary string with binary digits: ";
        cin >> str;
        if ( checkString ( str ) )
        {
            cout << "Invalid string input! Enter only binary
digits\n\n";
            continue;
        }
        processSignal(str);
        getch();
    }
    return 0;
}
```

OUTPUT



```
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\NRZ-Invert.exe"
Input a binary string with binary digits: 01010110010
```

RESULT

Digital signal generation code with NRZ-Invert encoding is written, executed and the output is verified.

Experiment **#10**

MANCHESTER ENCODING OF DIGITAL SIGNAL

AIM

Write a program to generate digital signal with Manchester encoding.

THEORY

Manchester encoding is a synchronous clock encoding technique used by the physical layer of the Open System Interconnection [OSI] to encode the clock and data of a synchronous bit stream. A logic 0 is indicated by a 0 to 1 transition at the centre of the bit and logic 1 by 1 to 0 transition. The signal transitions do not always occur at the 'bit boundary' but there is always a transition at the centre of each bit.

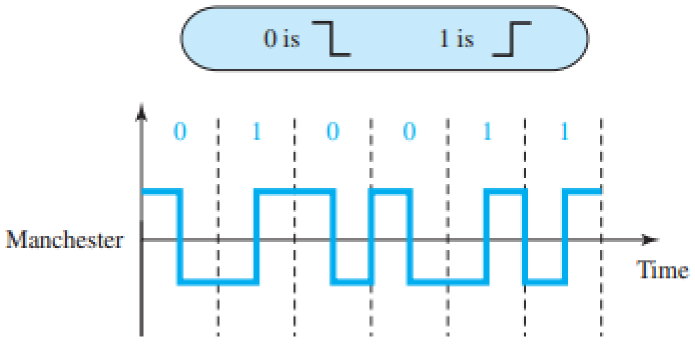
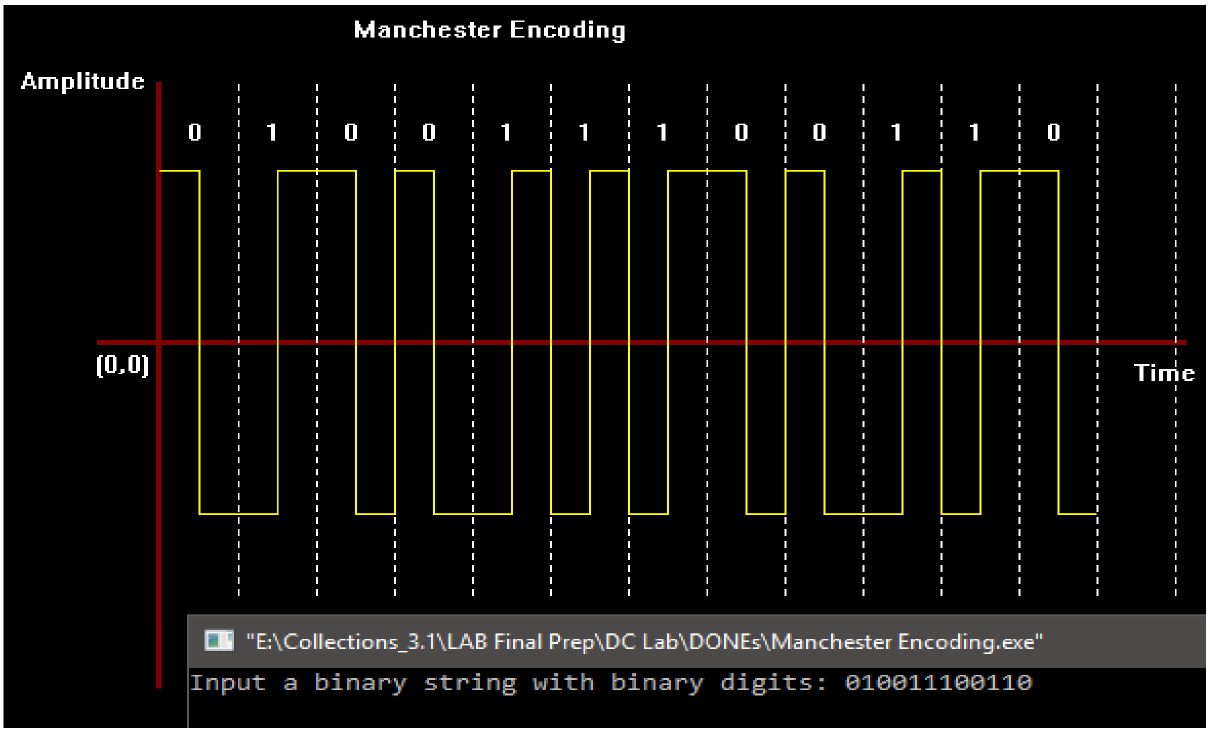


Figure 9 Manchester Encoding

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define dot_color YELLOW
#define deri 5
int sz, x, y, px, py;
char cx='a';
void decorate() {
    setfillstyle ( SOLID_FILL, BLACK );
    floodfill ( 100, 100, BLACK );
    outtextxy(200, 10, "Manchester Encoding");
    outtextxy(68, 205, "(0,0)");
    outtextxy(30, 40, "Amplitude");
    outtextxy(600, 210, "Time");
    return;
}
void drawAxis(){
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(RED);
    line ( 70, 200, 625, 200 );     // x axis
    line ( 100, 50, 100, 400 );     // y axis
    int z = 40;
    for(int i = 1; i<=13; i++) {
        setcolor(WHITE);
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        line(101+z, 50, 101+z, 350);
        z+=40;
    }
    return;
}
void horizontal_line(int m){
    for(int i = 1; i<21; i++){
        putpixel(x+i, m, dot_color);
        px = x+i;
        delay(deri);
    }
    return;
}
void vertical_line(int src, int des){
    int i = src;
    while(i!=des) {
        putpixel(x+1, i, dot_color);
        delay(deri);
        if(src>des) i--;
        else i++;
    }
    return;
}
void generateSignal(char c1) {
    if(c1=='0') {
        if(cx=='0')
            vertical_line(300,100);
        horizontal_line(100);
        x = px;
        vertical_line(100,300);
        horizontal_line(300);
    }
    else if(c1=='1') {
        if(cx=='1')
            vertical_line(100,300);
        horizontal_line(300);
        x = px;
        vertical_line(300,100);
        horizontal_line(100);
    }
    x = px, cx = c1;
    setcolor(WHITE);
    char A[2];
    sprintf(A, "%c", c1);
    outtextxy(x-25, 70, A);
    return;
}
void processSignal(string str) {
    x = 100, y = 100, px = 100, py = 100, cx = 'a';
    for(int i = 0 ; i<sz; i++) {
        char c1 = str[i];
        generateSignal(c1);
    }
    return;
}
bool checkString(string str) {
    sz = str.size();
    for(int i = 0; i<sz; i++) {
        if(str[i]!='0' && str[i]!='1')
            return true;
    }
    return false;
}
int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TC\\BGI");
    while(1) {
        cleardevice();
        decorate();
        drawAxis();
        string str;
        cout<< "Input a binary string with binary digits: ";
        cin>>str;
        if(checkString(str)) {
            cout<< "Invalid string input! Enter only binary
digits\n\n";
            continue;
        }
        processSignal(str);
        getch();
    }
    return 0;
}
```

OUTPUT



```
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Manchester Encoding.exe"
Input a binary string with binary digits: 010011100110
```

RESULT

Digital signal generation code with Manchester encoding is written, executed and the output is verified.

Experiment **#11**

DIFFERENTIAL MANCHESTER ENCODING OF DIGITAL SIGNAL

AIM

Write a program to generate digital signal with Differential Manchester encoding.

THEORY

Differential Manchester encoding. In differential Manchester encoding, the transition at the Middle of the bit is used only for synchronization. The bit representation is defined by the inversion "bit 0" or noninversion "bit 1" at the beginning of the bit.

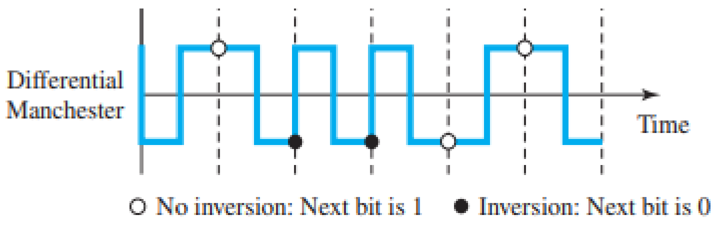


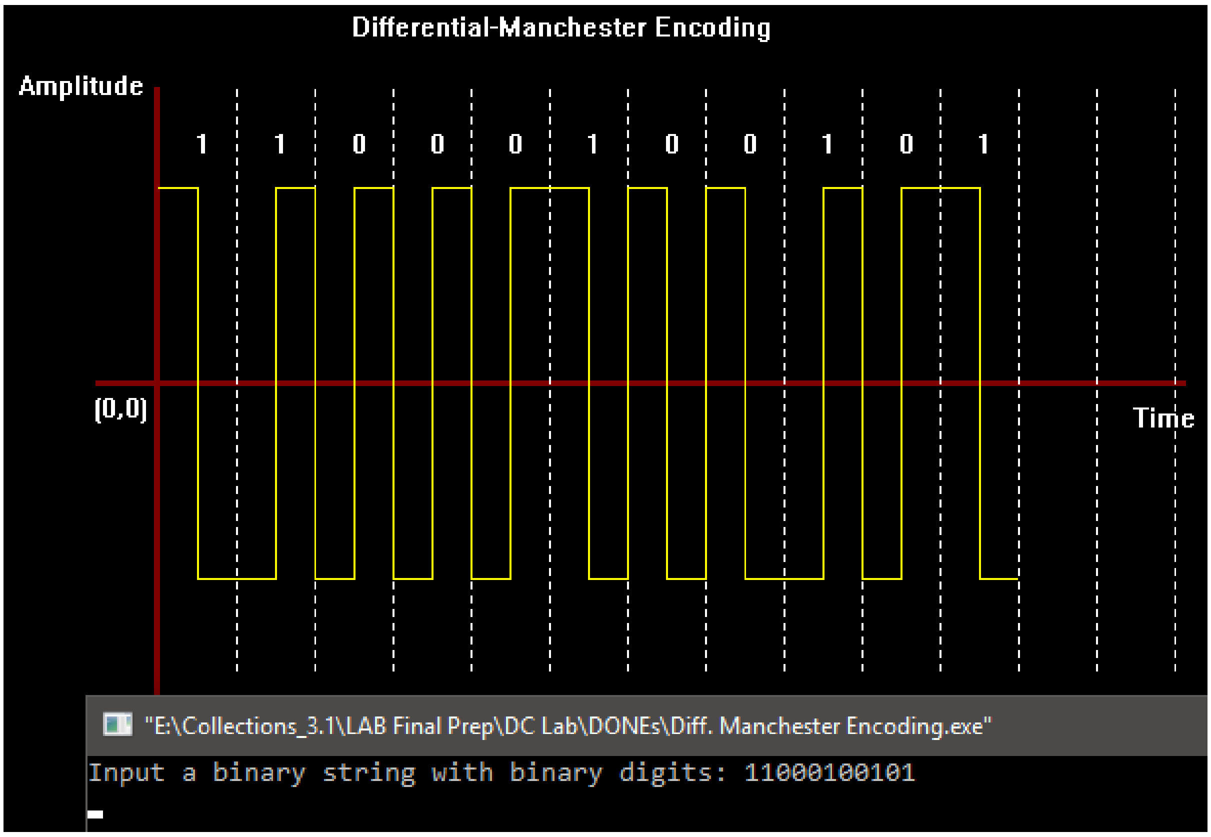
Figure 10 Differential Manchester Encoding

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;

#define dot_color YELLOW
#define deri 5
int sz, x, y, px, py;
int prev_des;
char cx='a';
bool flip;

void decorate() {
    setfillstyle ( SOLID_FILL, BLACK );
    floodfill ( 100, 100, BLACK );
    outtextxy(200, 10, "Differential-Manchester Encoding");
    outtextxy(68, 205, "(0,0)");
    outtextxy(30, 40, "Amplitude");
    outtextxy(600, 210, "Time");
    return;
}

void drawAxis()
{
    setlinestyle(SOLID_LINE,0,THICK_WIDTH);
    setcolor(RED);
    line ( 70, 200, 625, 200 );     // x axis
    line ( 100, 50, 100, 400 );     // y axis
    int z = 40;
    for(int i = 1; i<=13; i++) {
        setcolor(WHITE);
        setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
        line(101+z, 50, 101+z, 350);
        z+=40;
    }
    return;
}

void horizontal_line(int m)
{
    for(int i = 1; i<21; i++)
    {
        putpixel(x+i, m, dot_color);
        px = x+i;
        delay(deri);
    }
    return;
}

void vertical_line(int src, int des)
{
    int i = src;
    while(i!=des) {
        putpixel(x+1, i, dot_color);
        delay(deri);
        if(src>des) i--;
        else i++;
    }
    return;
}

bool liner(int src)
{
    if(src!=prev_des && src>99 && prev_des>99)
```

```
        return true;
    return false;
}

void generateSignal(char c1)
{
    if(c1=='1')
    {
        flip = 1-flip;
    }
    if(!flip)
    {
        if(liner(300))
            vertical_line(300,100);
        horizontal_line(100);
        x = px;
        vertical_line(100,300);
        horizontal_line(300);
        prev_des = 100;
    }
    else
    {
        if(liner(100))
            vertical_line(100,300);
        horizontal_line(300);
        x = px;
        vertical_line(300,100);
        horizontal_line(100);
        prev_des = 300;
    }
    x = px;
    setcolor(WHITE);
    char A[2];
    sprintf(A,"%c", c1);
    outtextxy(x-20, 70, A);
    return;
}

void processSignal(string str) {
    x = 100, y = 100, px = 100, py = 100, flip = prev_des = 1;
    cx = 'a';
    for(int i = 0 ; i<sz; i++) {
        char c1 = str[i];
        generateSignal(c1);
    }
    return;
}

bool checkString(string str) {
    sz = str.size();

    for(int i = 0; i<sz; i++) {
        if(str[i]!='0' && str[i]!='1')
            return true;
    }
    return false;
}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "C:\\TC\\BGI");

    while(1) {
        cleardevice();
        decorate();
        drawAxis();

        string str;
        cout<< "Input a binary string with binary digits: ";
        cin>>str;
        if(checkString(str)) {
            cout<< "Invalid string input! Enter only binary
digits\n\n";
            continue;
        }
        processSignal(str);
        getch();
    }
    return 0;
}
```

OUTPUT


Differential-Manchester Encoding
Amplitude
1
1
0
0
0
1
0
0
1
0
1
(0,0)
Time
"E:\Collections_3.1\LAB Final Prep\DC Lab\DONEs\Diff. Manchester Encoding.exe"

```
Input a binary string with binary digits: 11000100101
```

RESULT

Digital signal generation code with Differential Manchester encoding is written, executed and the output is verified.

Experiment **#12**

DIGITAL SIGNAL WITH FREQUENCY SHIFT KEYING (FSK)

AIM

Write a program to generate digital signal with Frequency Shift Keying (FSK).

THEORY

Digital-to-analog conversion is the process of changing one of the characteristics of an analog signal based on the information in digital data. Three mechanisms for modulating digital data into an analog signal: amplitude shift keying (ASK), frequency shift keying (FSK), and phase shift keying (PSK).

In frequency shift keying, the frequency of the carrier signal is varied to represent data. The frequency of the modulated signal is constant for the duration of one signal element, but changes for the next signal element if the data element changes. Both peak amplitude and phase remain constant for all signal elements.

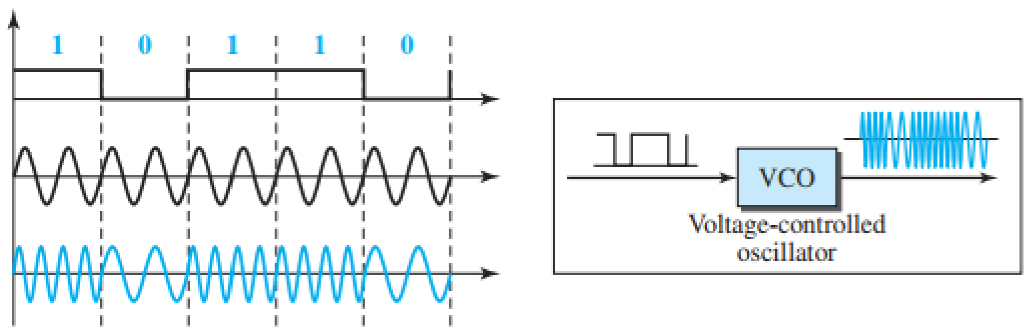


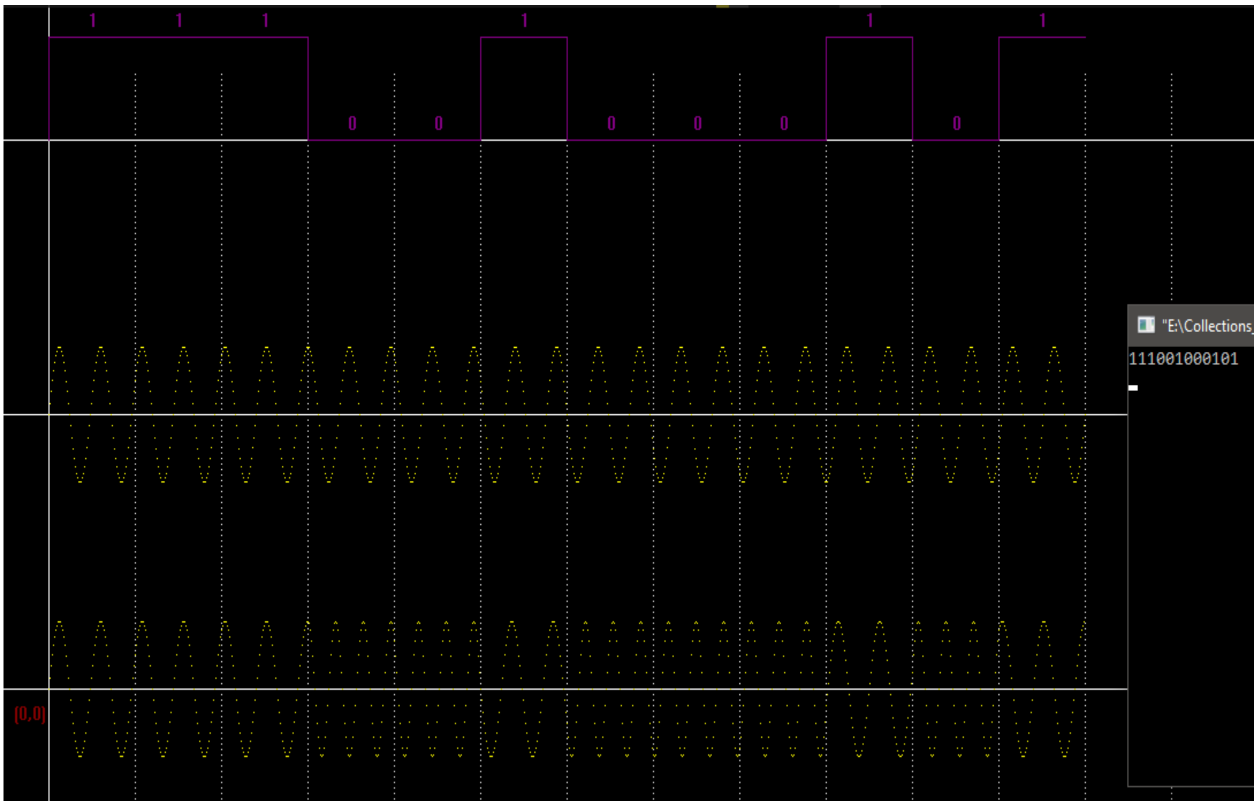
Figure 11 FSK

CODE

```
#include<bits/stdc++.h>
#include<graphics.h>
using namespace std;
#define deri 5
```

```
int x = 50;
int y = 500;
void  drawAxis()
{
    setcolor(WHITE);
    line(x,0,x,1500);
    line(0,y,1500,y);
    line(0,y-200,1500,y-200);
    line(0,y-400,1500,y-400);
    setcolor(RED);
    outtextxy(x-30,y+10,"(0,0)");
    for(int i = x+75; i<=1500; i=i+75) {
        for(int j = y+80; j>=50; j=j-4)
            putpixel(i,j,WHITE);
    }
}
void Pulse(string str)
{
    int ny = 100;
    int nx = x;
    setcolor(MAGENTA);
    int sz = str.size();
    for(int i = 0; i<sz; i++){
        int a = ny;
        if(str[i] == '0')
            ny = 100;
        else if(str[i] == '1')
            ny = 25;
        for(int j = a; j>=ny; j--)
            putpixel(nx,j,MAGENTA);
        for(int j = a; j<=ny; j++)
            putpixel(nx,j,MAGENTA);
        line(nx,ny,nx+75,ny);
        nx = nx+75;
        char A[3];
        sprintf(A,"%c",str[i]);
        outtextxy(nx-40,ny-20,A);
        delay(deri);
    }
}

void Base(string str)
{
    int angle = 0;
    int sz = str.size();
    for(int i = x; i<=(sz*75)+50; i = i+1){
        int yn = 50*sin(angle*(3.1416/180));
        int j = 300-yn;
        putpixel(i,j,YELLOW);
        delay(5);
        angle = angle+10;
    }
}
void FSK(string str)
{
    int angle = 0;
    setcolor(YELLOW);
    int sz = str.size();
    for(int i=0; i<sz; i++){
        int b;
        if(str[i] == '0')
            b = 15;
        else
            b = 10;
        for(int j = x; j<x+75; j = j+1){
            int yn = 50*sin(angle*(3.1416/180));
            int jy = 500-yn;
            putpixel(j,jy,YELLOW);
            delay(5);
            angle = angle+b;
        }
        x = x+75;
    }
}

int main()
{
    string str;
    cin>>str;
    initwindow(1500,700,"Digital Signal FSK");
    drawAxis();
    Pulse(str);
    Base(str);
    FSK(str);
    getch();
}
```

OUTPUT



```
111001000101
```

RESULT

Digital signal generation code with Frequency Shift Keying is written, executed and the output is verified.

Experiment **#13**

DIGITAL SIGNAL WITH AMPLITUDE SHIFT KEYING (ASK)

AIM

Write a program to generate digital signal with Amplitude Shift Keying (ASK).

THEORY

Digital-to-analog conversion is the process of changing one of the characteristics of an analog signal based on the information in digital data. Three mechanisms for modulating digital data into an analog signal: amplitude shift keying (ASK), frequency shift keying (FSK), and phase shift keying (PSK).

In amplitude shift keying, the amplitude of the carrier signal is varied to create signal elements. Both frequency and phase remain constant while the amplitude changes.

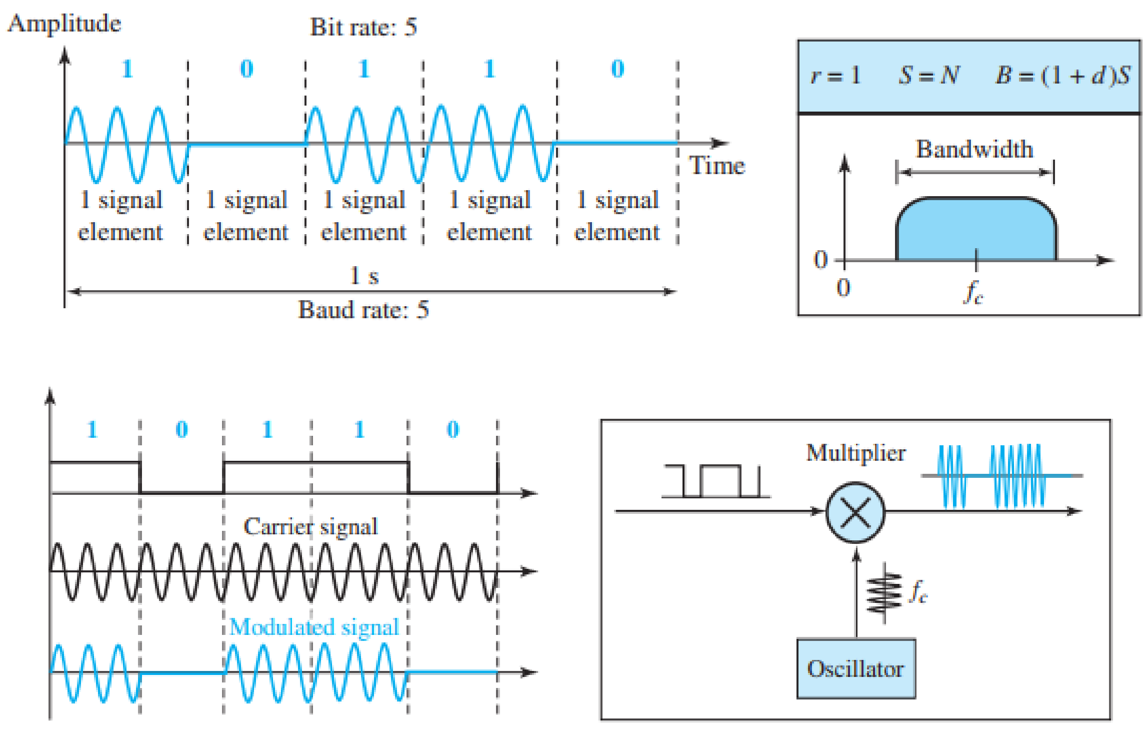
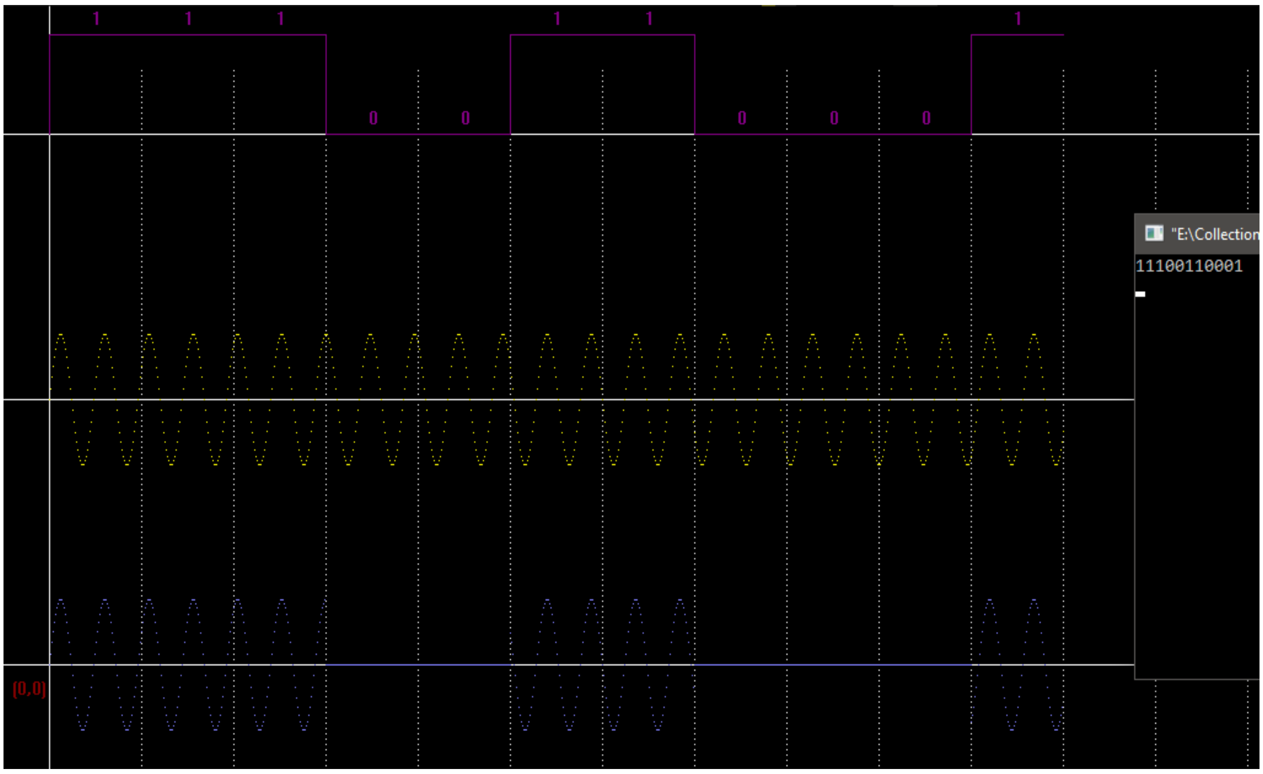


Figure 12 ASK

OUTPUT

```
#include<bits/stdc++.h>
#include<graphics.h>
#include<bits/stdc++.h>
using namespace std;
int x = 50, y = 500;

void drawAxis()
{
    setcolor(WHITE);
    line(x,0,x,1500);
    line(0,y,1500,y);
    line(0,y-200,1500,y-200);
    line(0,y-400,1500,y-400);
    setcolor(RED);
    outtextxy(x-30,y+10,"(0,0)");
    for(int i = x+75; i<=1500; i=i+75){
        for(int j = y+80; j>=50; j=j-4)
            putpixel(i,j,WHITE);
    }
}

void Pulse(string str)
{
    int ny = 100, nx = x;
    setcolor(MAGENTA);
    int sz = str.size();
    for(int i = 0; i<sz; i++){
        int a = ny;
        if(str[i] == '0')
            ny = 100;
        else if(str[i] == '1')
            ny = 25;
        for(int j = a; j>=ny; j--)
            putpixel(nx,j,MAGENTA);
        for(int j = a; j<=ny; j++)
            putpixel(nx,j,MAGENTA);
        line(nx,ny,nx+75,ny);
        nx = nx+75;
        char A[3];
        sprintf(A,"%c",str[i]);
        outtextxy(nx-40,ny-20,A);
    }
}

void Base(string str)
{
    int angle = 0, sz = str.size();
    for(int i = x; i<=(sz*75)+50; i = i+1){
        int yn = 50*sin(angle*(3.1416/180));
        int j = 300-yn;
        putpixel(i,j,YELLOW);
        delay(5);
        angle = angle+10;
    }
}

void ASK(string str)
{
    int angle = 0, sz = str.size();
    setcolor(LIGHTBLUE);
    for(int i=0; i<sz; i++){
        int k = 0;
        if(str[i] == '1'){
            for(int j = x; j<x+75; j = j+1){
                int yn = 50*sin(angle*(3.1416/180));
                int jy = 500-yn;
                putpixel(j,jy,LIGHTBLUE);
                delay(5);
                angle = angle+10;
            }
            k++;
        }
        if(k == 0){
            angle = angle + 750;
            line(x,500,x+75,500);
        }
        x = x+75;
    }
}

int main()
{
    string str;
    cin>>str;
    initwindow(1500,700,"Digital Signal ASK");
    drawAxis();
    Pulse(str);
    Base(str);
    ASK(str);
    getch();
}
```

OUTPUT



```
11100110001
```

RESULT

Digital signal generation code with Amplitude Shift Keying is written, executed and the output is verified.