1. GENERATION OF SUM OF SINUSOIDAL SIGNALS

AIM: Write a **MATLAB** program to generate sum of sinusoidal signals

APPARATUS: Software:-Mat lab 7.0v, PC

THEORY: A continuous-time signal x(t) is said to be periodic with period T if it satisfy the condition

$x(t+T) = x(t)$ for all t $\quad -\infty<t<+\infty$ ……eq1

A signal is periodic if the above condition is not satisfied for at least one value of T.

The smallest value of T that satisfies the above condition is known as fundamental period.

Complex exponential and sinusoidal signals are examples for continuous-time periodic signals. Consider a sinusoidal signal

$x(t)=A\sin(wt+\theta)$ ……eq2

Where A is tha amplitude, w is tha frequency in radians per second (rad/sec), and θ the phase in radians. The frequency f_0 in hertz is given by $f_0 = w/2\pi$ or periodic signal we have

$X(t+T)=x(t)$ ……eq3

For $x(t)=A\sin(wt+\theta)$ ……eq4

$X(t+T) = A\sin[w(t+T)+\theta)]$

$=A\sin[wt+wT+\theta]$ ……eq5

Equation 5 and 2 are equal if

$wT = 2\pi$, $T = 2\pi / w$

Thu the sinusoidal is periodic with period $2\pi / w$

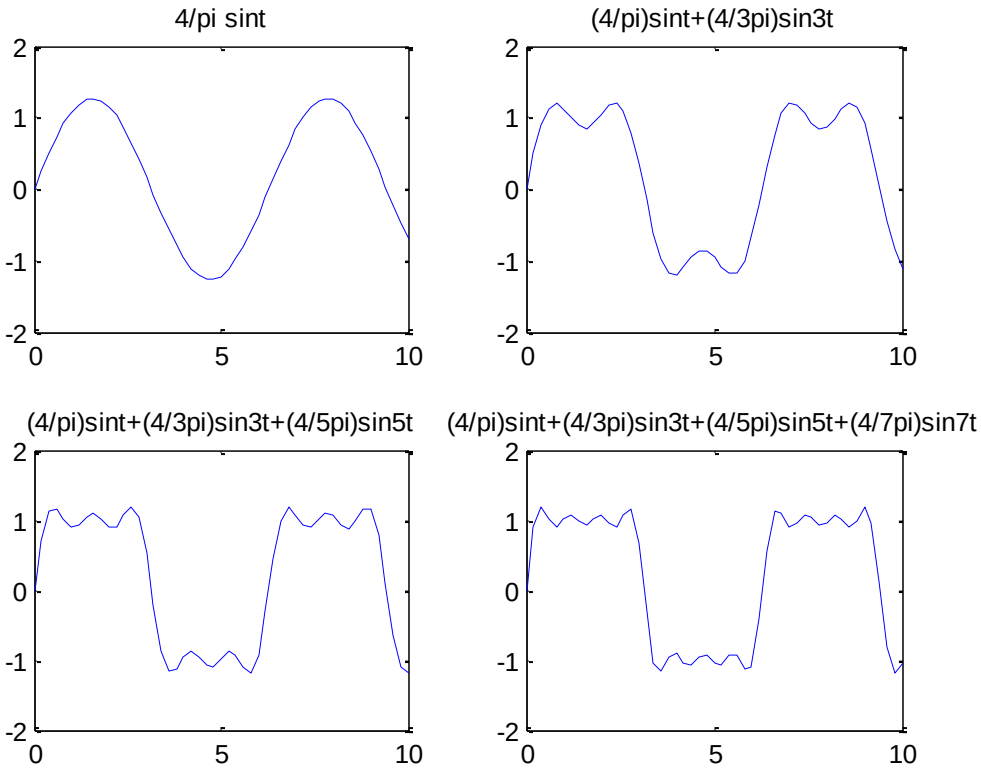
The sum of two periodic signals $x_1(t)$ and $x_2(t)$ with periods T_1 and T_2 respectively may or may not be periodic, then the ratio T_1/T_2 can be written as the ratio a/b of two integers a and b. If $T_1/T_2 = a/b$, then $bT_1 = aT_2$, and since a and b are integers, $x_1(t)$ and $x_2(t)$ are periodic with period bT_1. If a and b are co prime (i. e. a and b have no common integer factors other than 1) then $T = bT_1$ is the fundamental period of the sum of two signals.

PROGRAM:

```
clc;
clear all;
close all;
t=0: .2:10;
x1=(4/pi)*sin(t);
subplot(2,2,1);
plot(t,x1);
title('4/pi sint');
x2=(4/pi)*1/3*sin(3*t);
xa=x1+x2;
subplot(2,2,2);
plot(t,xa);
title('(4/pi)sint+(4/3pi)sin3t');
```

```
x3=(4/pi)*1/5*sin(5*t);

xb=x1+x2+x3;

subplot(2,2,3);

plot(t,xb);

title('(4/pi)sint+(4/3pi)sin3t+(4/5pi)sin5t');

x4=4/pi*1/7*sin(7*t);

xc=x1+x2+x3+x4;

subplot(2,2,4);

plot(t,xc);

title('(4/pi)sint+(4/3pi)sin3t+(4/5pi)sin5t+(4/7pi)sin7t');
```

OUTPUT WAVEFORMS :



RESULT:

Output waveform is observed by using MATLAB.

2. DFT AND IDFT OF A GIVEN SEQUENCE

AIM: To obtain DFT/IDFT of a given sequence

APPARATUS:

1. Software:-Matlab 7.0v
2. PC

THEORY:

Discrete Fourier Transform (DFT) is used for performing frequency analysis of discrete time signals. DFT gives a discrete frequency domain representation whereas the other transforms are continuous in frequency domain. The N point DFT of discrete time signal x[n] is given by the equation .The inverse DFT allows us to recover the sequence x[n] from the frequency samples.

X(k) is a complex number (remember ejw=cosw + jsinw). It has both magnitude and phase which are plotted versus k. These plots are magnitude and phase spectrum of x[n]. The 'k' gives us the frequency information. Here k=N in the frequency domain corresponds to sampling frequency (fs). Increasing N, increases the frequency resolution, i.e. it improves the spectral characteristics of the sequence. For example if fs=8kHz and N=8 point DFT, then in the resulting spectrum, k=1 corresponds to 1kHz frequency. For the same fs and x[n], if N=80 point DFT is computed, then in the resulting spectrum, k=1 corresponds to 100Hz frequency. Hence, the resolution in frequency is increased. Since $N \geq L$, increasing N to 8 from 80 for the same x[n] implies x[n] is still the same sequence (<8), the rest of x[n] is padded with zeros. This implies that there is no further information in time domain, but the resulting spectrum has higher frequency resolution. This spectrum is known as 'high density spectrum' (resulting from zero padding x[n]). Instead of zero padding, for higher N, if more number of points of

x[n] are taken (more data in time domain), then the resulting spectrum is called a 'high resolution spectrum'.

PROGRAM:

A) DFT of a given sequence x(n)

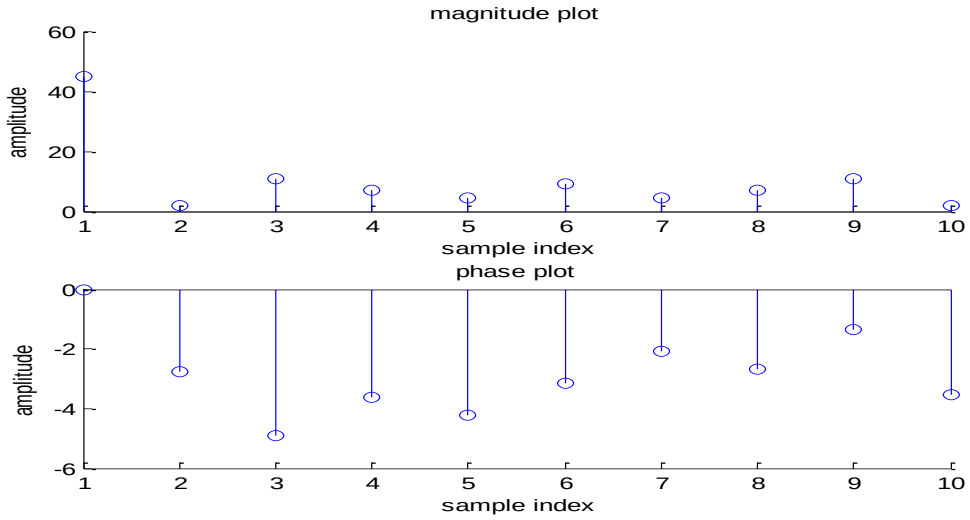
```
% file name DFT generation
close all;
clc;
clear all;
X=input('enter the input sequence')
N=length(X)
for k=0:N-1
   for n=0:N-1
      p=exp(-j*2*pi*k*n/N);
      f(k+1,n+1)=p
   end
end
X1=X*f
figure(1)
subplot(2,1,1)
stem(abs(X1))
xlabel(' sample index')
```

```
ylabel('amplitude')
title('magnitude plot')
subplot(2,1,2)
stem(phase(X1))
xlabel(' sample index')
ylabel('amplitude')
title('phase plot DFT of a given sequence X(k)')
```

OUTPUT:

```
enter the input sequence[ 2 3 4 5 6 7 2 3 4 9]
X = 2    3    4    5    6    7    2    3    4    9
N=10
X1 =  45.0000          -1.7639 - 0.7265i   2.2361 +10.6861i  -6.2361 + 3.0777i
      -2.2361 + 3.9757i  -9.0000 - 0.0000i  -2.2361 - 3.9757i  -6.2361 - 3.0777i
       2.2361 -10.6861i  -1.7639 + 0.7265i
```

OUTPUT :



B) IDFT of a given sequence X(k)

```
%file name IDFT generation
close all;
clc;
clear all;
X=input('enter the input sequence')
N=length(X)
for k=0:N-1
   for n=0:N-1
      p=exp(j*2*pi*k*n/N);
      f(k+1,n+1)=p
   end

end
X1=X*f/N
figure(1)
subplot(2,1,1)
stem(abs(X1))
xlabel(' sample index')
ylabel('amplitude')
title('magnitude plot')
subplot(2,1,2)
stem(phase(X1))
xlabel(' sample index')
ylabel('amplitude')
title('phase plot')
```

OUTPUT:

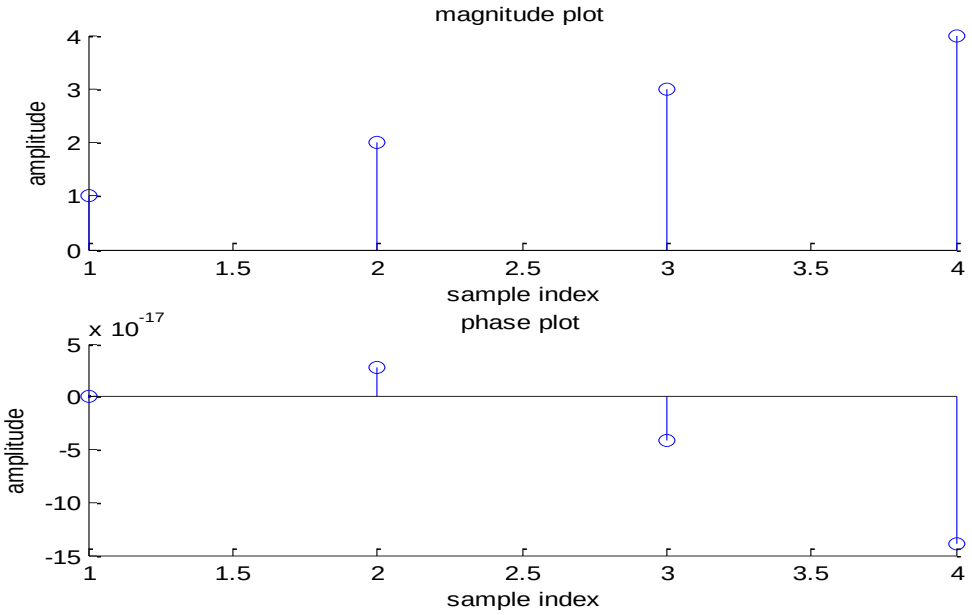
enter the input sequence[10 -2+2i -2 -2-2i]

X =10.0000 -2.0000 + 2.0000i -2.0000 -2.0000 - 2.0000i

N =4

X1 = 1.0000 2.0000 + 0.0000i 3.0000 - 0.0000i 4.0000 - 0.0000i

OUTPUT :



RESULT:

Output waveform is observed by using MATLAB.

3. FREQUENCY RESPONSE OF A SYSTEM

AIM: To plot the frequency response of the given system

APPARATUS: Software:- Mat lab 7.0v, PC

THEORY:

If we know a transfer function model of the system, we can calculate the frequency response from the transfer function, as explained below.

Suppose that system has the transfer function H(s) from input u to output y, that is,

$$y(s) = H(s)u(s)$$ By setting

$s = j\omega$ (j is the imaginary unit) into H(s), we get the complex quantity $H(j\omega)$, which is the frequency response (function).

The gain function is

$$A(\omega) = |H(j\omega)| \ .$$

The phase shift function is the angle or argument of $H(j\omega)$:

$$\varphi(\omega) = \arg H(j\omega)$$

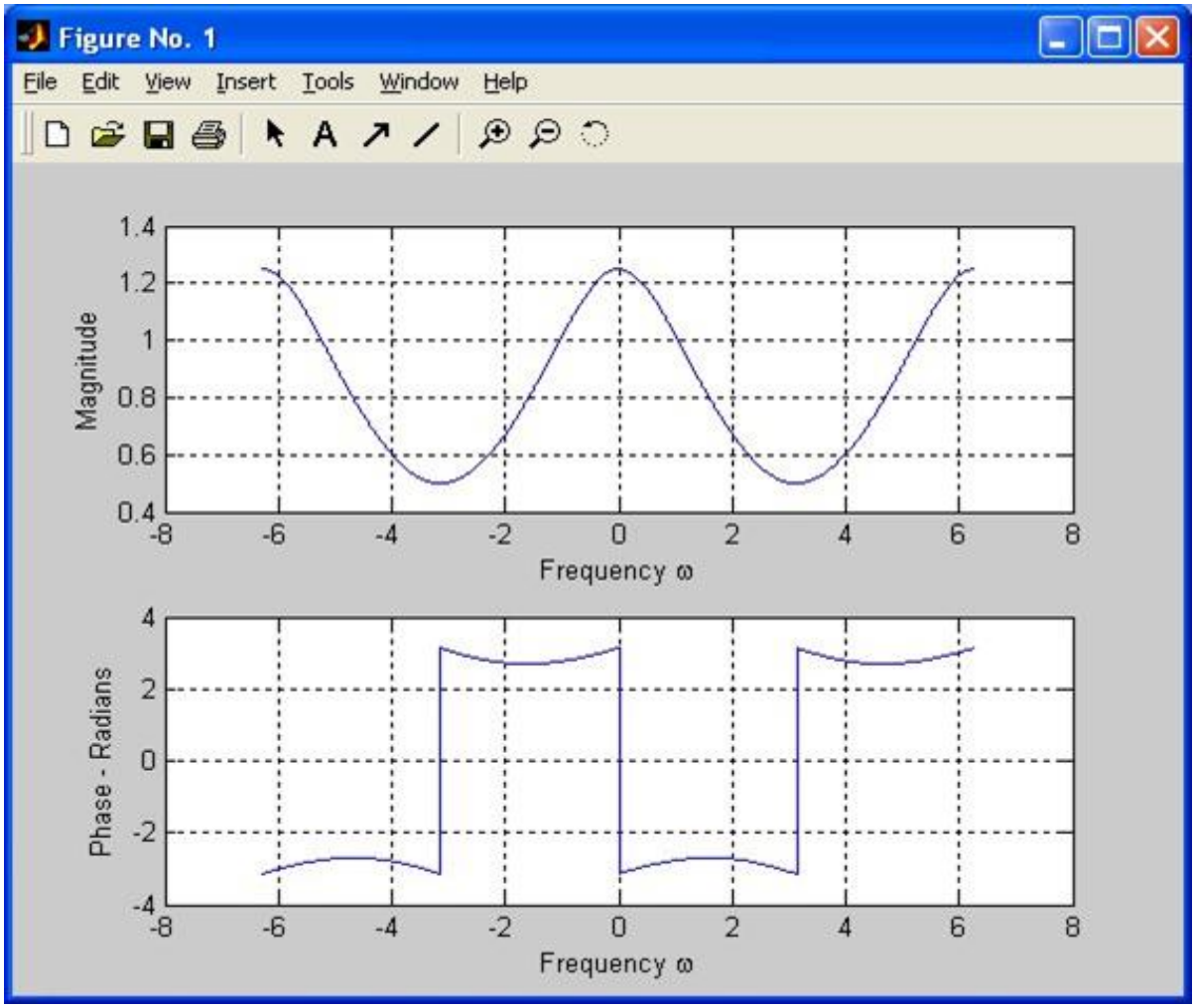
PROGRAM:

```
clc;
clear all;
close all;
b = [1, 4]; %Numerator coefficients
a = [1, -5]; %Denominator coefficients
w = -2*pi: pi/256: 2*pi;
[h] = freqz(b, a, w);
subplot(2, 1, 1), plot(w, abs(h));
xlabel('Frequency \omega'), ylabel('Magnitude'); grid
```

subplot(2, 1, 2), plot(w, angle(h));

xlabel('Frequency \omega'), ylabel('Phase - Radians'); grid

WAVEFORMS:



RESULT:

Output waveform is observed by using MATLAB.

4. FAST FOURIER TRANSFORM

AIM: To obtain FFT of a given sequence

APPARATUS: Software:-Matlab 7.0v

THEORY: The DFT Equation is given by

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$where \quad W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \text{ [TWIDDLE FACTOR]}$$

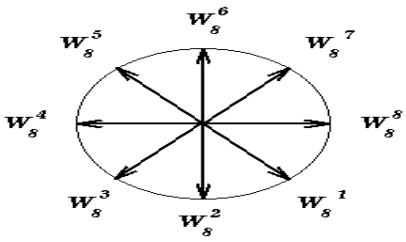
Twiddle Factor

In the Definition of the DFT, there is a factor called the *Twiddle Factor*

$$W_N^{nk} = e^{-j\frac{2\pi nk}{N}}$$

where N = number of samples.

If we take an 8 bit sample sequence we can represent the twiddle factor as a vector in the unit circle. e.g.



Note :

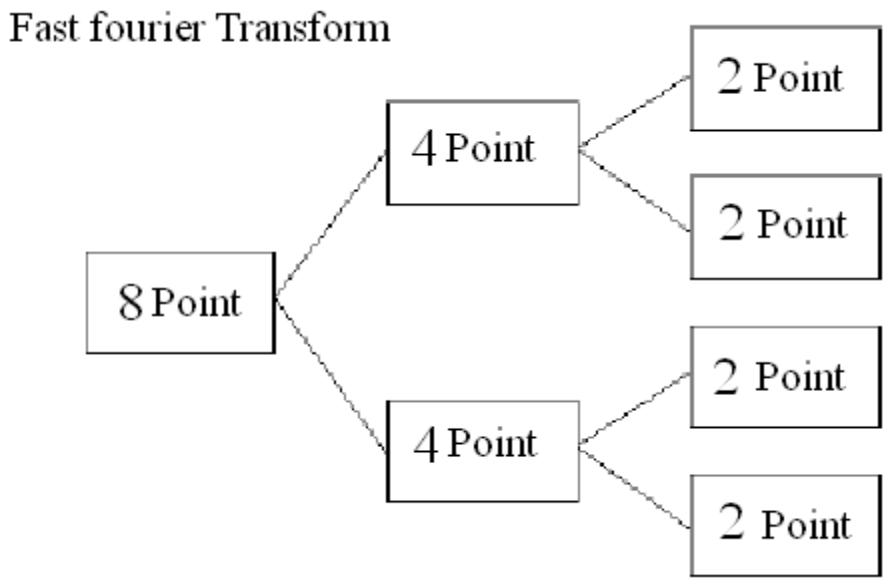
1. It is periodic. (i.e. it goes round and round the circle !!)
2. That the vectors are symmetric
3. The vectors are equally spaced around the circle.

Need for FFT

If you look at the equation for the Discrete Fourier Transform you will see that it is quite complicated to work out as it involves many additions and multiplications involving complex numbers. Even a simple eight sample signal would require 49 complex multiplications and 56 complex additions to work out the DFT. At this level it is still manageable; however a realistic signal could have 1024 samples which requires over 20,000,000 complex multiplications and additions. As you can see the number of calculations required soon mounts up to unmanageable proportions.

The Fast Fourier Transform is a simply a method of laying out the computation, which is much faster for large values of N, where N is the number of samples in the sequence. It is an ingenious way of achieving rather than the DFT's clumsy P^2 timing.

The idea behind the FFT is the *divide and conquer* approach, to break up the original N point sample into two (N/2) sequences. This is because a series of smaller problems is easier to solve than one large one. The DFT requires (N-1)^2 complex multiplications and N(N-1) complex additions as opposed to the FFT's approach of breaking it down into a series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal.



Algorithm:

1. Get the input sequence
2. Number of DFT point(m) is 8
3. Find out the FFT functionusing MATLAB function.
4. Display the input & outputs sequence using stem function

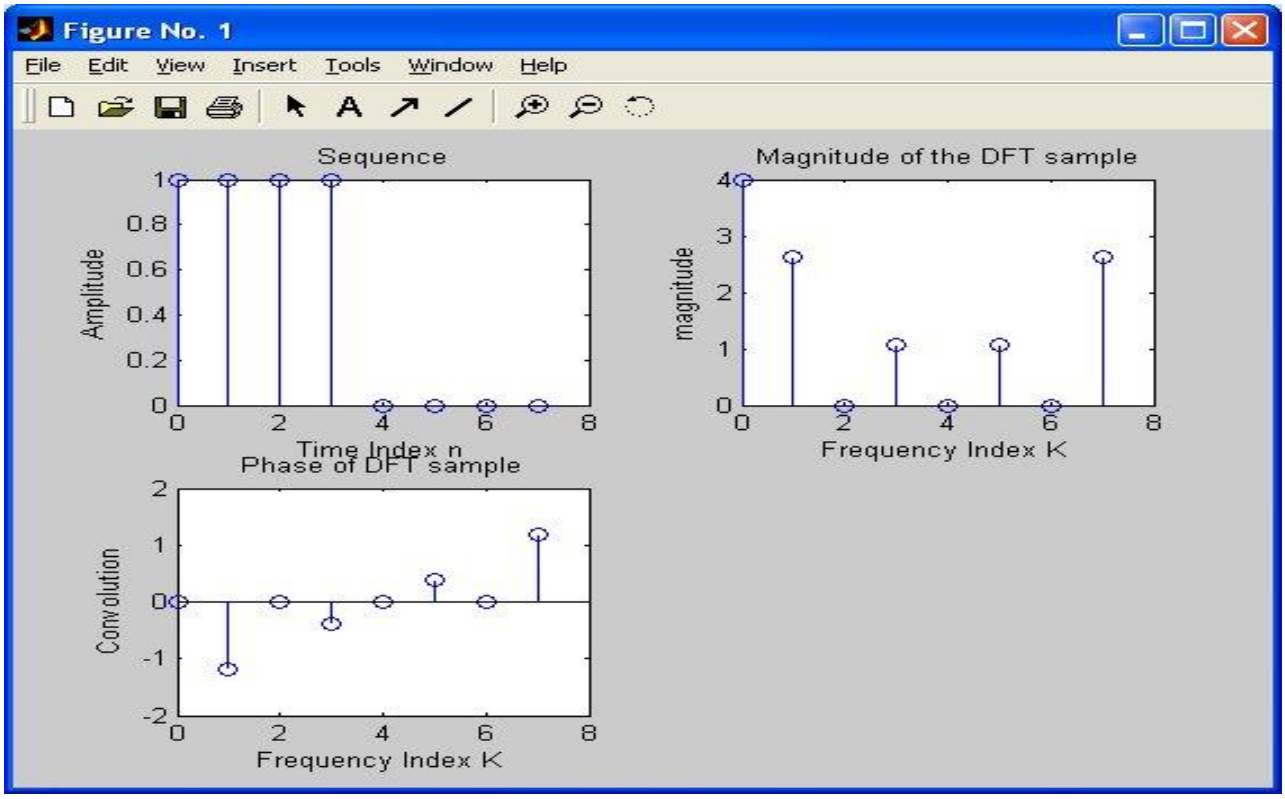
Program:

```
clear all;
N=8;
m=8;
a=input('Enter the input sequence');
n=0:1:N-1;
subplot(2,2,1);
stem(n,a);
xlabel('Time Indexn');
ylabel('Amplitude');
title('Sequence');
x=fft(a,m);
```

```
k=0:1:N-1;
subplot(2,2,2);
stem(k,abs(x));
ylabel('magnitude');
xlabel('Frequency Index K');
title('Magnitude of the DFT sample');
subplot(2,2,3);
stem(k,angle(x));
xlabel('Frequency Index K');
ylabel('Phase');
title('Phase of DFT sample');
ylabel('Convolution');
```

Output:-

Enter the input sequence [1 1 1 1 0 0 0 0]



Result: Output waveform is observed by using MATLAB.

5. POWER SPECTRAL DENSITY

AIM: To find the power spectral density of a given signal

APPARATUS: 1. Software:-Matlab 7.0v

2. PC

THEORY:

The total or the average power in a signal is often not of as great an interest. We are most often interested in the PSD or the Power Spectrum. We often want to see is how the input power has been redistributed by the channel and in this frequency-based redistribution of power is where most of the interesting information lies. The total area under the Power Spectrum or PSD is equal to the total average power of the signal. The PSD is an even function of frequency or in other words

To compute PSD: The value of the auto-correlation function at zero-time equals the total power in the signal. To compute PSD we compute the auto-correlation of the signal and then take its FFT. The auto-correlation function and PSD are a Fourier transform pair. (Another estimation method called "period gram" uses sampled FFT to compute the PSD.)

E.g.: For a process *x(n)* correlation is defined as:

$$R(\tau) = E\{x(n)x(n+\tau)\}$$

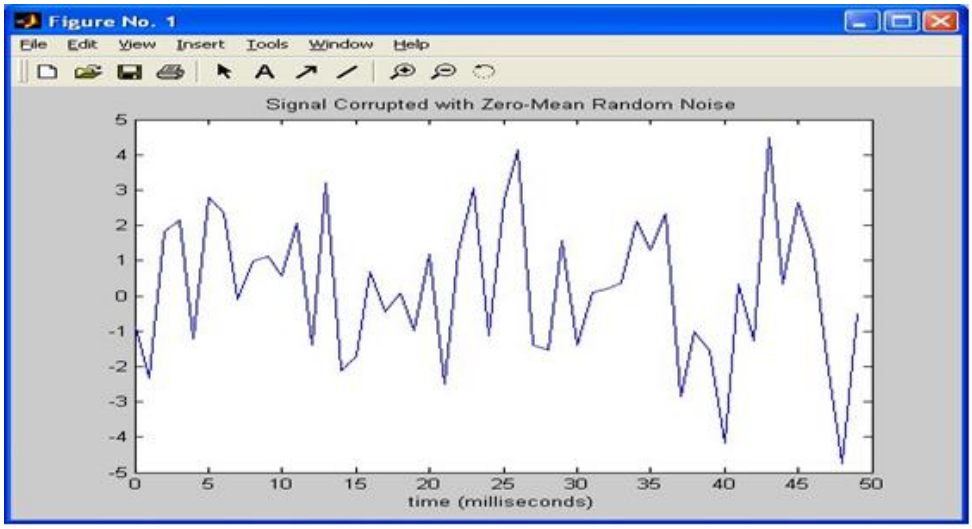
$$= \lim_{N \to \infty} \frac{1}{N} \sum_{n=1}^{N} x(n)x(n+\tau)$$

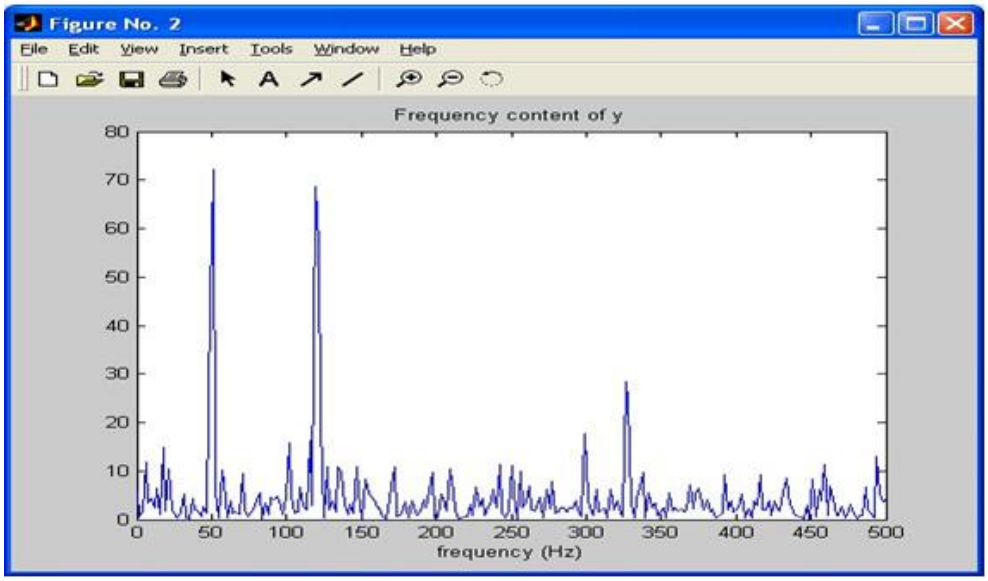
Power Spectral Density is a Fourier transform of the auto correlation.

PROGRAM:

```
%Power spectral density
t = 0:0.001:0.6;
x = sin(2*pi*50*t)+sin(2*pi*120*t);
y = x + 2*randn(size(t));
figure, plot(1000*t(1:50),y(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('time (milliseconds)');
Y = fft(y,512);
%The power spectral density, a measurement of the energy at various
frequencies, is:
Pyy = Y.* conj(Y) / 512;
f = 1000*(0:256)/512;
figure,plot(f,Pyy(1:257))
title('Frequency content of y');
xlabel('frequency (Hz)');
```

OUTPUT:





RESULT:

Output waveform is observed by using MATLAB.

6. IMPLEMENTATION OF LP FIR FILTER

AIM: To design FIR low pass filter for given specifications using

a) Kaiser window

APPARATUS:

1. Software:-Matlab 7.0v
2. PC

THEORY:

A Finite Impulse Response (FIR) filter is a discrete linear time-invariant system whose output is based on the weighted summation of a finite number of past inputs. An FIR transversal filter structure can be obtained directly from the equation for discrete-time convolution.

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k) \quad 0 < n < N-1$$

In this equation, x(k) and y(n) represent the input to and output from the filter at time n. h(n-k) is the transversal filter coefficients at time n. These coefficients are generated by using FDS (Filter Design Software or Digital filter design package).

FIR – filter is a finite impulse response filter. Order of the filter should be specified. Infinite response is truncated to get finite impulse response. placing a window of finite length does this. Types of windows available are Rectangular, Barlett, Hamming, Hanning, Blackmann window etc. This FIR filter is an all zero filter.

KAISER WINDOW:

W = KAISER(N) returns an N-point Kaiser window in the column vector W.

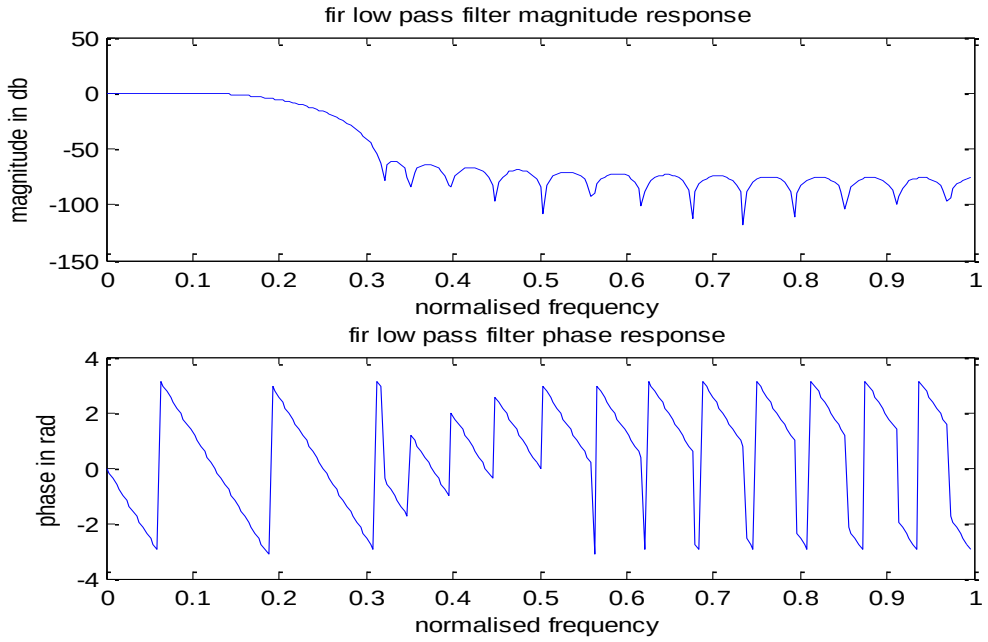
W = KAISER(N,BTA) returns the BETA-valued N-point Kaiser window.

PROGRAM:

```
clc;
clear all;
close all;
rp=input('enter the pass band ripple');
rs=input('enter the stop band ripple');
fp=input('enter the pass band freq');
fs=input('enter the stop band freq');
f=input('enter sampling freq');
beta=input('enter the beta value');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
    n1=n;n=n-1;
end
y=kaiser(n1,beta);
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
```

```
figure(1)
subplot(2,1,1)
plot(o/pi,m);
xlabel('normalised frequency')
ylabel('magnitude in db');
title('fir low pass filter magnitude response');
subplot(2,1,2)
plot(o/pi,angle(h));
xlabel('normalised frequency')
ylabel('phase in rad');
title('fir low pass filter phase response');
```

OUTPUT WAVE FORMS:



RESULT: Output waveform is observed by using MATLAB.

7. IMPLEMENTATION OF HP FIR FILTER

AIM: To design FIR high pass filter for given specifications using

a) Kaiser window

APPARATUS:

1. Software:-Matlab 7.10.0v
2. PC

THEORY:

A Finite Impulse Response (FIR) filter is a discrete linear time-invariant system whose output is based on the weighted summation of a finite number of past inputs. An FIR transversal filter structure can be obtained directly from the equation for discrete-time convolution.

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k) \quad 0 < n < N-1$$

In this equation, x(k) and y(n) represent the input to and output from the filter at time n. h(n-k) is the transversal filter coefficients at time n. These coefficients are generated by using FDS (Filter Design Software or Digital filter design package).

FIR – filter is a finite impulse response filter. Order of the filter should be specified. Infinite response is truncated to get finite impulse response. placing a window of finite length does this. Types of windows available are Rectangular, Barlett, Hamming, Hanning, Blackmann window etc. This FIR filter is an all zero filter.

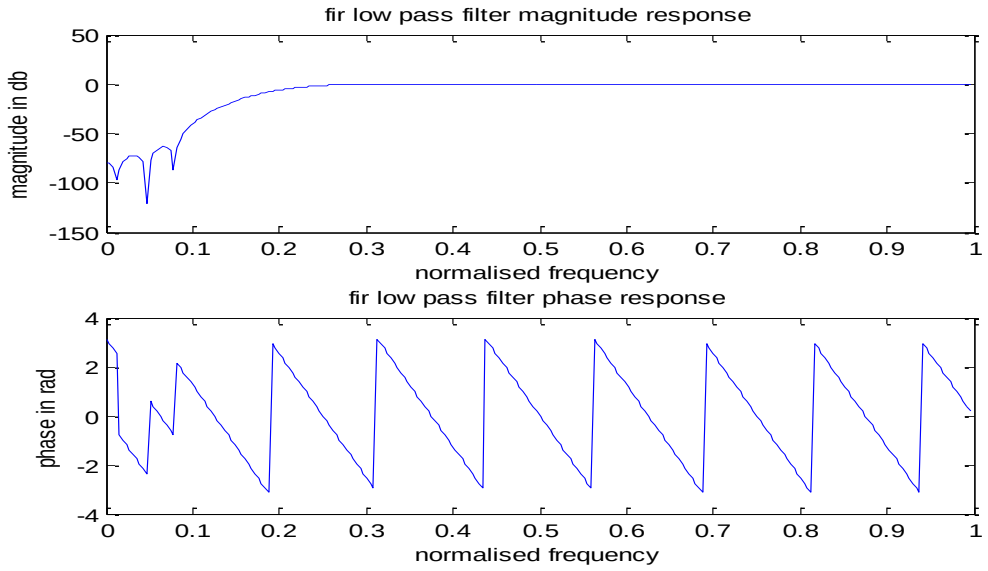
PROGRAM:

```
clc;
clear all;
```

```
close all;
rp=input('enter the pass band ripple');
rs=input('enter the stop band ripple');
fp=input('enter the pass band freq');
fs=input('enter the stop band freq');
f=input('enter sampling freq');
beta=input('enter the beta value');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
    n1=n;n=n-1;
end
y=kaiser(n1,beta);
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
figure(2)
subplot(2,1,1)
plot(o/pi,m);
```

```
xlabel('normalised frequency')
ylabel('magnitude in db');
title('fir low pass filter magnitude response');
subplot(2,1,2);
plot(o/pi,angle(h));
xlabel('normalised frequency');
ylabel('phase in rad');
title('fir low pass filter phase response');
```

OUTPUT WAVE FORMS:



RESULT:

Output waveform is observed by using MATLAB.

8. IMPLEMENTATION OF IIR LPF

AIM: To design IIR low pass filter

APPARATUS:

1. Software:-Matlab 7.0v
2. PC

THEORY:

The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

$$H(z)\frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=1}^{N} a_k Z^{-k}}$$

The difference equation for such a system is described by the following:

$$y(n)=\sum_{k=0}^{M} b_k x(n-k) \; + \; \sum_{k=1}^{N} a_k y(n-k)$$

M and N are order of the two polynomials bk and ak are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That's why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

PROGRAM:

A) BUTTERWORTH LPF

```
clc;
close all;
clear all;
format long
rp=input('enter the passband ripple')
rs=input('enter the stopband ripple')
wp=input('enter the passband freq')
ws=input('enter the stopband freq')
fs=input('enter the sampling freq')
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn);
w=0:.01:pi;
[h,om]=freqz(b,a,w,'whole');
m=20*log(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in dB -->');
xlabel('(a) normalised frequency -->');
title('magnitude plot')
```

```
subplot(2,1,2);plot(om/pi,an);
ylabel('phase in radians -->');
xlabel('(b) normalised frequency -->');
title('phase plot')
```

OUTPUT:

enter the passband ripple 4

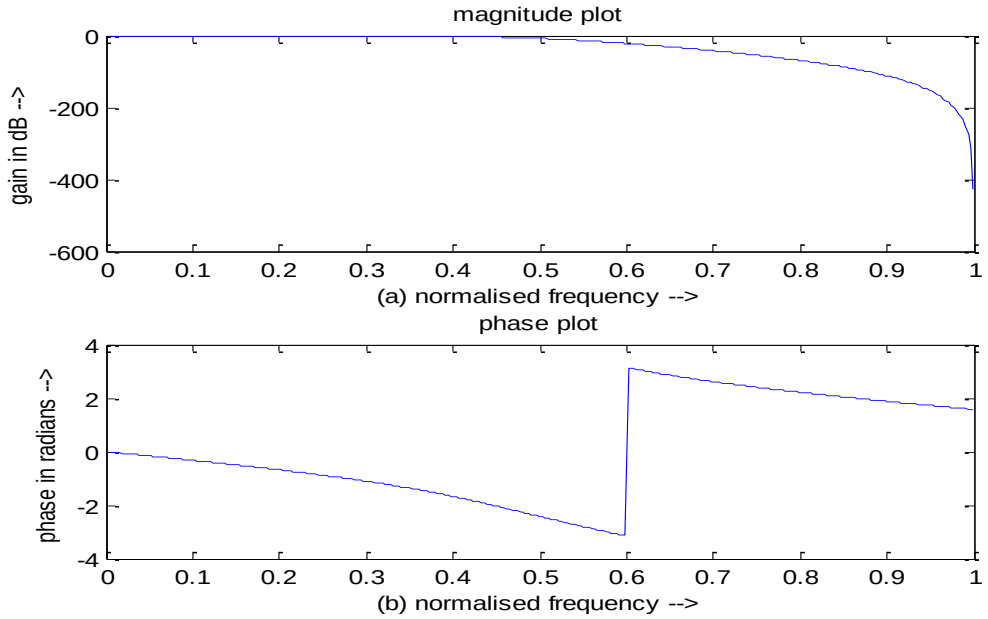
enter the stopband ripple 30

enter the passband freq 400

enter the stopband freq 800

enter the sampling freq 2000

OUTPUT WAVEFORMS:



B) CHEBYSHEV TYPE-1 LP FILTER

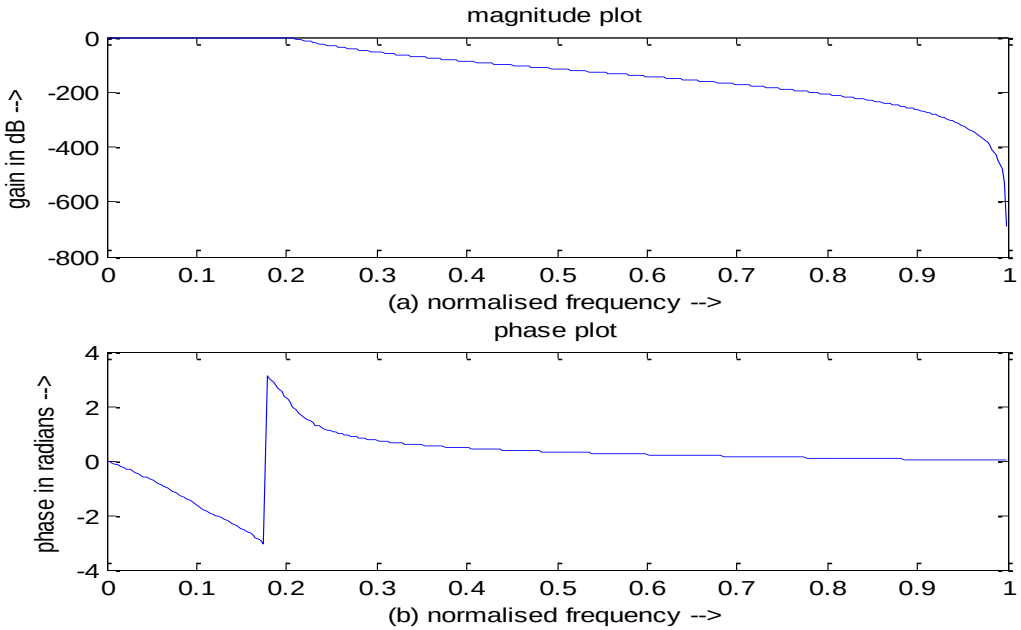
```
clc;
```

```
close all;
clear all;
rp=1
rs=15
wp=.2*pi;
ws=.3*pi;
[n,wn]=cheb1ord(wp/pi,ws/pi,rp,rs);
[b,a]=cheby1(n,rp,wn);
w=0:.01:pi;
[h,om]=freqz(b,a,w);
m=20*log(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in dB -->');
xlabel('(a) normalised frequency -->');
title('magnitude plot')
subplot(2,1,2);plot(om/pi,an);
ylabel('phase in radians -->');
xlabel('(b) normalised frequency -->');
title('phase plot')
```

OUTPUT:

```
b =    0.0018    0.0073    0.0110    0.0073    0.0018

a =    1.0000   -3.0543    3.8290   -2.2925    0.5507
```

OUTPUT WAVEFORMS:



C) CHEBYSHEV TYPE-2 LP FILTER

```
clc;

close all;

clear all;

rp=1

rs=20

wp=.2*pi;

ws=.3*pi;

[n,wn]=cheb2ord(wp/pi,ws/pi,rp,rs);

[b,a]=cheby2(n,rs,wn);

w=0:.01:pi;
```

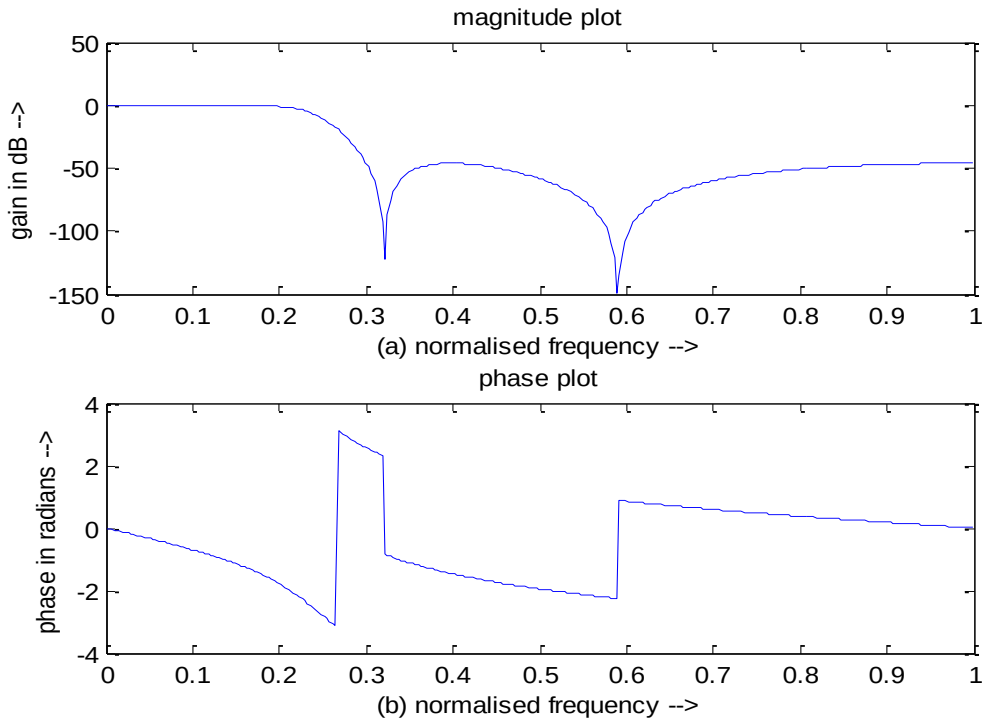
```
[h,om]=freqz(b,a,w);
m=20*log(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in dB -->');
xlabel('(a) normalised frequency -->');
title('magnitude plot')
subplot(2,1,2);plot(om/pi,an);
ylabel('phase in radians -->');
xlabel('(b) normalised frequency -->');
title('phase plot')
```

OUTPUT:

b = 0.1160 -0.0591 0.1630 -0.0591 0.1160

a =1.0000 -1.8076 1.5891 -0.6201 0.1153

OUTPUT WAVEFORMS:



RESULT:

Output waveform is observed by using MATLAB.

9. IMPLEMENTATION OF IIR HPF

AIM: To design IIR high pass filter

APPARATUS:

1. Software:- Mat lab 7.0v
2. PC

THEORY:

The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

$$H(z)\frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=1}^{N} a_k Z^{-k}}$$

The difference equation for such a system is described by the following:

$$y(n)=\sum_{k=0}^{M} b_k x(n-k) \; + \; \sum_{k=1}^{N} a_k y(n-k)$$

M and N are order of the two polynomials bk and ak are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That's why the filters are named

as butter worth filters. Filter coefficients can be found and the response can be plotted.

PROGRAM:

A) BUTTERWORTH HPF

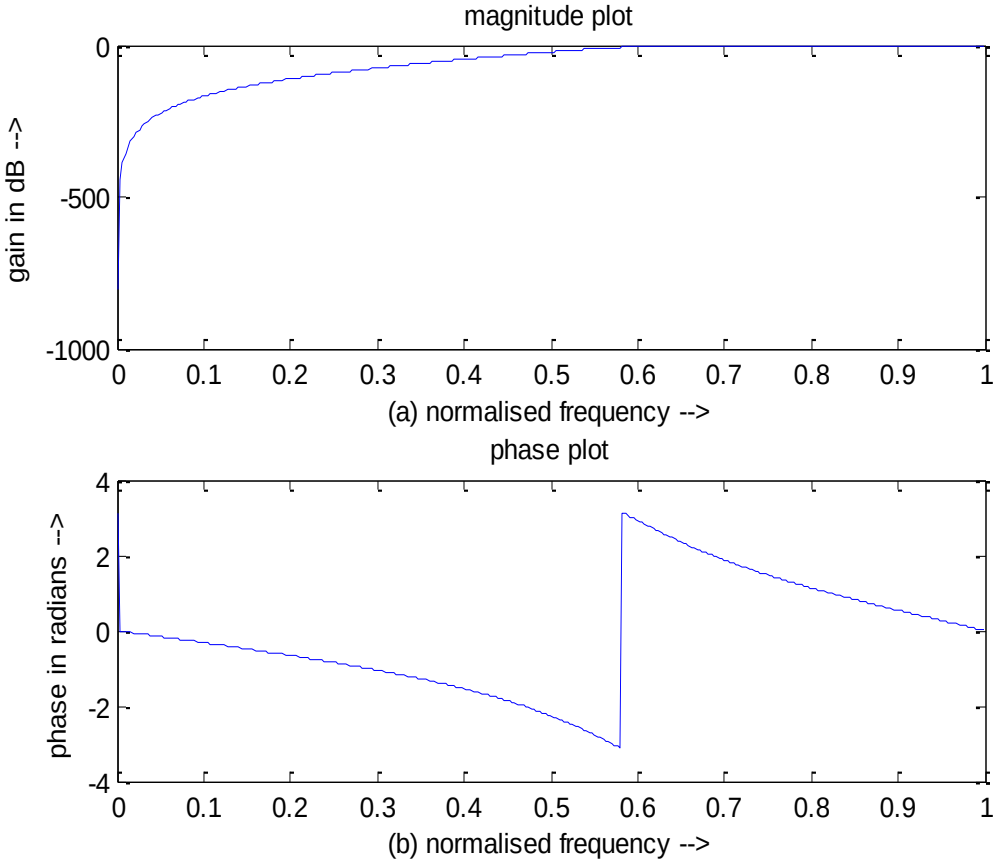
```
clc;
close all;clear all;
format long
rp=input('enter the passband ripple')
rs=input('enter the stopband ripple')
wp=input('enter the passband freq')
ws=input('enter the stopband freq')
fs=input('enter the sampling freq')
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn,'high');
w=0:.01:pi;
[h,om]=freqz(b,a,w);
m=20*log(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in dB -->');
xlabel('(a) normalised frequency -->');
title('magnitude plot')
```

```
subplot(2,1,2);plot(om/pi,an);
ylabel('phase in radians -->');
xlabel('(b) normalised frequency -->');
title('phase plot')
```

OUTPUT:

```
enter the passband ripple  .4
enter the stopband ripple  30
enter the passband freq  400
enter the stopband freq  800
enter the sampling freq  2000
```

OUTPUT WAVEFORMS:



B) CHEBYSHEV TYPE-1 HP FILTER

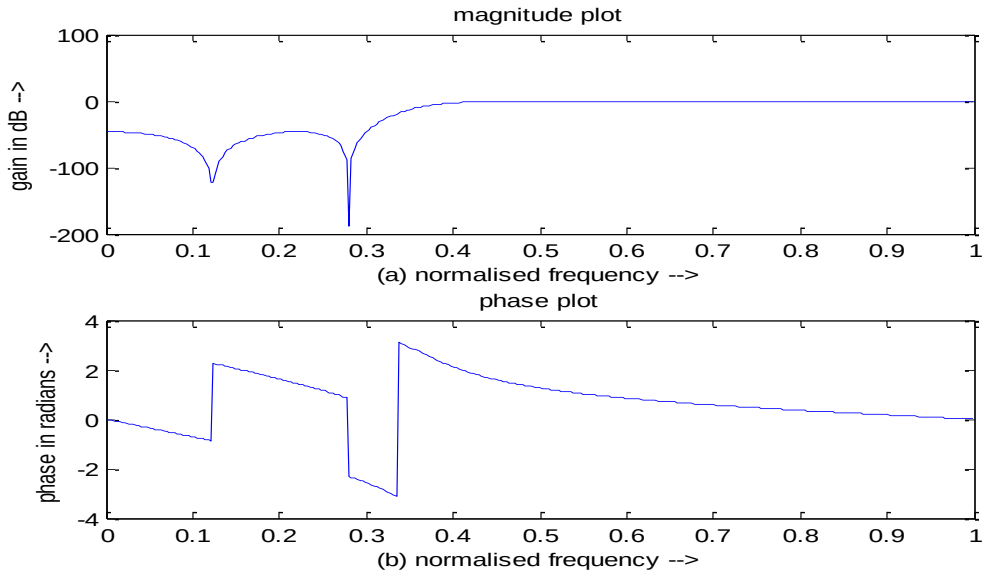
```
clc;

close all;

clear all;

rp=1

rs=20

wp=.2*pi;

ws=.3*pi;

[n,wn]=cheb2ord(wp/pi,ws/pi,rp,rs);
```

```
[b,a]=cheby2(n,rs,wn,'high');
w=0:.01:pi;
[h,om]=freqz(b,a,w);
m=20*log(abs(h));
an=angle(h);
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in dB -->');
xlabel('(a) normalised frequency -->');
title('magnitude plot')
subplot(2,1,2);plot(om/pi,an);
ylabel('phase in radians -->');
xlabel('(b) normalised frequency -->');
title('phase plot')
```

OUTPUT:

```
b = 0.3723   -1.1645    1.6240   -1.1645    0.3723
a =1.0000   -1.4971    1.4075   -0.6538    0.1392
```

OUTPUT WAVE FORMS:



C) CHEBYSHEV TYPE2 HP FILTER

```
clc;
close all;
clear all;
rp=1
rs=20
wp=.2*pi;
ws=.3*pi;
[n,wn]=cheb2ord(wp/pi,ws/pi,rp,rs);
[b,a]=cheby2(n,rp,wn,'high');
w=0:.01:pi;
[h,om]=freqz(b,a,w);
m=20*log(abs(h));an=angle(h);
```

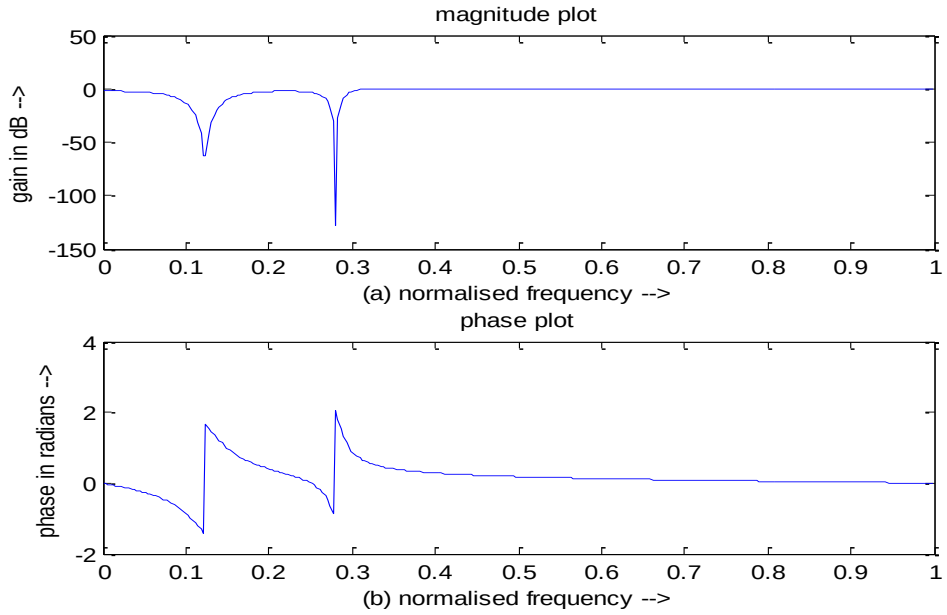
```
subplot(2,1,1);plot(om/pi,m);
ylabel('gain in dB -->');
xlabel('(a) normalised frequency -->');
title('magnitude plot')
subplot(2,1,2);plot(om/pi,an);
ylabel('phase in radians -->');
xlabel('(b) normalised frequency -->');
title('phase plot')
```

OUTPUT:

b = 0.8605 -2.6915 3.7535 -2.6915 0.8605

a =1.0000 -2.8722 3.7396 -2.5053 0.7404

OUTPUT WAVEFORMS:



RESULT: Output waveform is observed by using MATLAB.

10.SINUSOIDAL SIGNAL THROUGH FILTERING

AIM: -Generation of Sine Wave & Illustration of the Sampling Process in the Time Domain.

SOFTWARE REQURIED:-

1. MATLAB R2010a.

2. Windows XP SP2.

THEORY:-

Sinusoidal Signal Generation

The sine wave or sinusoid is a mathematical function that describes a smooth repetitive oscillation. It occurs often in pure mathematics, as well as physics, signal processing, electrical engineering and many other fields. Its most basic form as a function of time (t) where:

• A, the amplitude, is the peak deviation of the function from its center position.

• ω, the angular frequency, specifies how many oscillations occur in a unit time interval, in radians per second

• φ, the phase, specifies where in its cycle the oscillation begins at t = 0.

A sampled sinusoid may be written as:

$$x(n) = A\sin(2\pi \frac{f}{f_s} n + \vartheta)$$

PROCEDURE:-

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

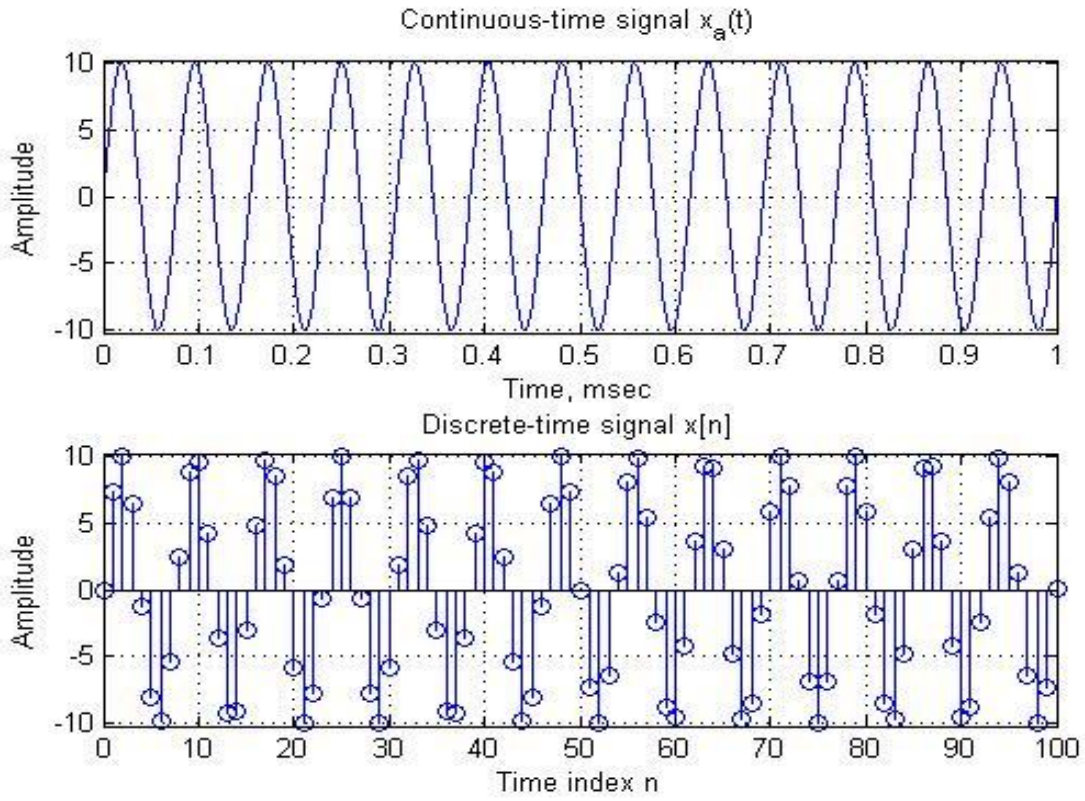
PROGRAM:-

% Generation of Sine Wave & Illustration of the Sampling Process in the Time Domain

```
clc;

t =
0:0.0005:1; a
= 10

f = 13;

xa =
a*sin(2*pi*f*t);
subplot(2,1,1)
plot(t,xa);grid
xlabel('Time,
msec');
ylabel('Amplitude'
);

title('Continuous-time signal
x_{a}(t)'); axis([0 1 -10.2 10.2])

subplot(2,1,2
); T = 0.01;

n = 0:T:1;

xs =
a*sin(2*pi*f*n); k
= 0:length(n)-1;
stem(k,xs);

grid

xlabel('Time index n');
ylabel('Amplitude');
title('Discrete-time signal
x[n]'); axis([0 (length(n)-1) -
```

10.2 10.2])

OUTPUT:-



RESULT:-

Sinusoidal signal is generated by using MATLAB.

EXERCISE PROGRAM:-

1. Write program to get Discrete time Sinusoidal Signal?
2. Write program to get Fourier Transform of Sinusoidal Signal?
3. Write program to get Inverse Fourier Transform of Sinusoidal Signal?
4. Write a matlab program for generating u(n)-u(n-1)?
5. Write program to get Discrete time co-Sinusoidal Signal?
6. Write program to get Discrete time saw tooth Signal?
7. Write program to get Discrete time triangular Signal?
8. Write program to get addition of two sinusoidal sequences?
9. Write program to get exponential sequence?
10. Write program to get Fourier Transform of Co-Sinusoidal Signal?
11. Write program to get Inverse Fourier Transform of Co-Sinusoidal Signal?
12. Write program to get exponential decaying sequence?
13. Write program to get exponential growing sequence?
14. Write program to get addition of two Co-sinusoidal sequences?
15. Write program to get Discrete time Square Signal?

VIVA QUESTIONS:-

1. Define sinusoidal signal?
2. Define C.T.S?
3. Define D.T.S?
4. Compare C.T.S & D.T.S?
5. . Draw the C.T.S & D.T.S diagrams?

11. DUAL TONE MULTI FREQUENCY GENERATION

AIM: To generate Dual tone multi frequency signals

APPARATUS:

1. Software:-Matlab 7.0v
2. PC

THEOREY

DTMF signaling is the basis for voice communication control and is widely used for worldwide in modern telephony to dial telephone numbers and configure switch boards. It is also used in systems such as in voice mails, e-mails and tele banking.

Generating DTMF Tones: A DTMF signal consists of sum of two sinusoidal or tones with frequency's taken from 2 mutually exclusive groups.

These frequencies were chosen to prevent any harmonics from being in corrected detected by receiver as some other DTMF. Each pair of tones contains one frequency of the low frequency group(697hz,770hz,852hz,941hz) and one frequency of the high frequency group(1209hz,1336hz,1477hz) and represents a unique symbol. The frequency allocated to the push buttons of allocated pad.

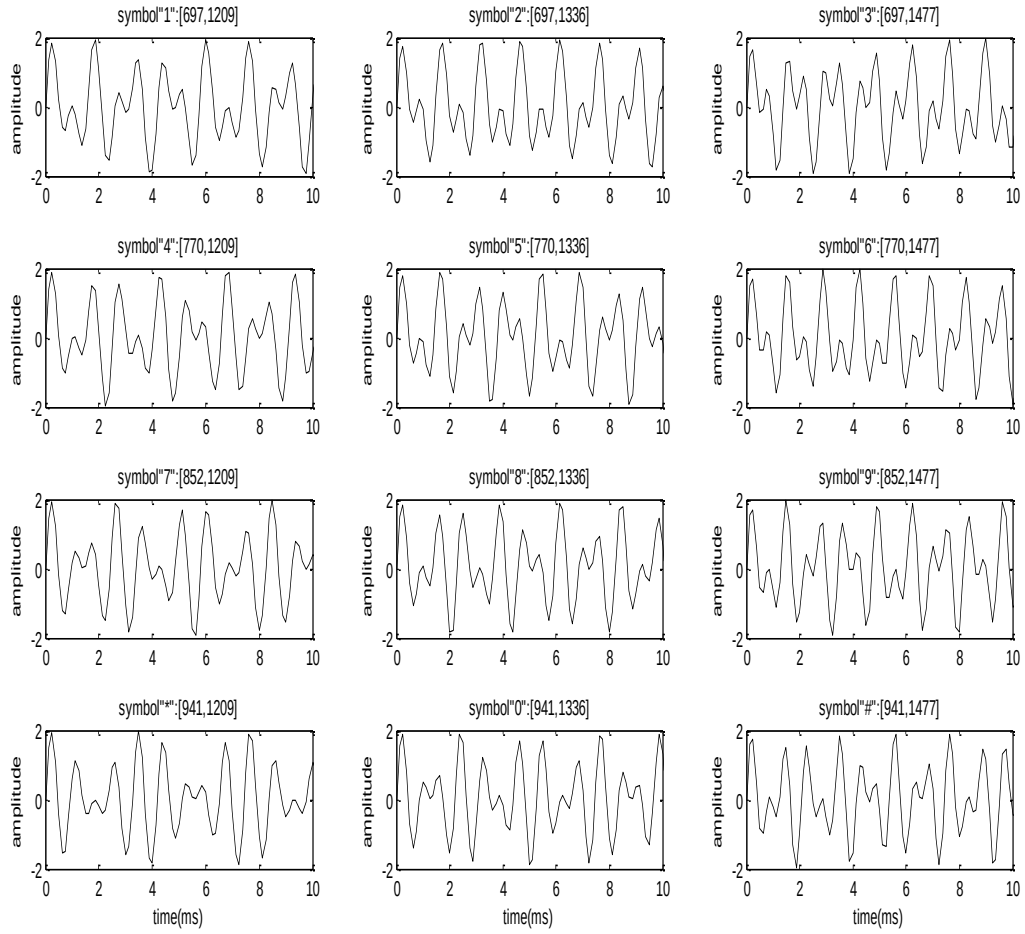
PROGRAM:

```
%file name:dtmf1

clc;

close all;

clear all;

%file name:DTMF 19-jan-2011

%first, let's generate the twelve frequency path

symbol={'1','2','3','4','5','6','7','8','9','*','0','#'}
```

```
lfg=[697 770 852 941]; %low frequency group
hfg=[1209 1336 1477]; %high freuqency group
f=[];
for c=1:4
 for r=1:3
  f=[f [lfg(c);hfg(r)]];
 end
end
%%a=f(:,12)
%%next let's generate and visualize the dtmf
Fs=8000 %sampling frequency 8K Hz
N=800 %tones of 100ms
t=(0:N-1)/Fs %800 samples at Fs
pit=2*pi*t
tones=zeros(N,size(f,2))
for tonechoice=1:12
 %generate tone
  a=f(:,tonechoice)
 b=a*pit
 sinb=sin(b)
 c=sum(sinb)
tones(:,tonechoice)= c'
%%plot tone
```

```
subplot(4,3,tonechoice)
plot(t*1e3,tones( :,tonechoice));

title(['symbol'",symbol{tonechoice},"':[',num2str(f(1,tonechoice)),',',num2str(f(2,to
nechoice)),']'])
set(gca,'xlim',[0 10]);
ylabel('amplitude')
 if tonechoice>9,

 xlabel('time(ms)')
  end
end
```

OUTPUT WAVE FORMS:



RESULT:

Output waveform is observed by using MATLAB.

12. DECIMATION OF A GIVEN SEQUENCE

AIM: To verify decimation of a give sequence

APPARATUS:

1.Software:-Matlab 7.0v
2. PC

THEORY:

The DSP systems must be employed to work with different sampling rates. Such a system with an ability to work with multiple sampling rates are known as multiple DSP systems.

The multirate signal processing is done by increasing or decreasing the sampling rate. Increasing the sampling rate by a factor 'I' is known as "Interpolation" and and decreasing the sampling rate by a factor D is known as "Decimation".

The Decimation factof is a factor by witch the signal is decimated .the following equation depicts the relation between input and output with a decimation factor 'P'

$$Y(m)=W(m_p)$$

Where $W(m)=W(m_p)$=filterd sequence

$$Y(m)=\sum_{-\infty}^{\infty} bk\, X(mp-k)$$

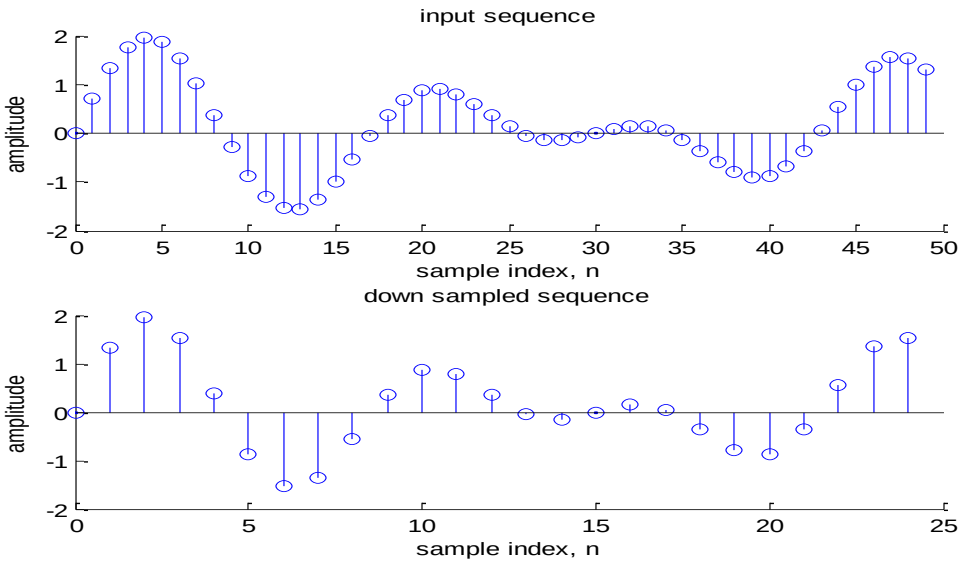
In figure we can see that at the output,sampling frequency 'fs' is decimated by a factor of 'P' thus making the sampling frequency equal to $\frac{fs}{p}$.

PROGRAM:

```
% file name : down sampler

clear all;

close all;

clc;

N=50
```

```
n=0:N-1
X=sin(2*pi*n/20)+sin(2*pi*n/15) %input sequence
D=2  %down sampling factor
X1=X(1:D:N) %down sampled output sequence
n1=1:N/D
subplot(2,1,1)
stem(n,X)
xlabel('sample index, n')
ylabel('amplitude')
title('input sequence')
subplot(2,1,2)
stem(n1-1,X1)
xlabel('sample index, n')
ylabel('amplitude')
title('down sampled sequence')
```

OUTPUT WAVE FORMS :



RESULT:

Output waveform is observed by using MATLAB.

13. INTERPOLATION OF A GIVEN SEQUENCE

AIM: To verify interpolation of a give sequence

APPARATUS:

1. Software:-Matlab 7.0v

THEORY:

The DSP systems must be employed to work with different sampling rates. Such a system with an ability to work with multiple sampling rates are known as multiple DSP systems.

The multirate signal processing is done by increasing or decreasing the sampling rate. Increasing the sampling rate by a factor 'I' is known as "Interpolation" and and decreasing the sampling rate by a factor D is known as "Decimation".

Interpolation is done by introdusing additional samples between successive samples. The below figure shows the interpolation by afactor I=3.

The response of interpolating filter is

$B(w)= I$ for $0 \le w \le w_c$

0 for $w_c \le w \le \pi$

here $w_c = \frac{\pi}{I}$ or $f_c = \frac{fs}{2I}$

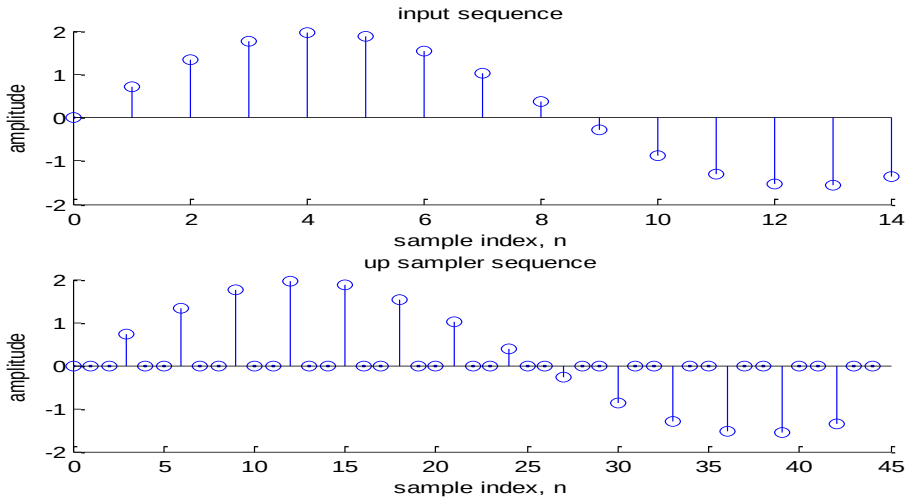
$f_c=f_s/2$

PROGRAM:

```
% file name : up sampler

clear all;

close all;

clc;

N=15 % sequence length
```

```
n=0:N-1
X=sin(2*pi*n/20)+sin(2*pi*n/15) %input sequence
I=3  %up sampling factor
X1=[zeros(1,I*N)] %up sampler output sequence
n1=1:I*N, j=1:I:I*N
X1(j)=X
subplot(2,1,1)
stem(n,X)
xlabel('sample index, n')
ylabel('amplitude')
title('input sequence')
subplot(2,1,2)
stem(n1-1,X1)
xlabel('sample index, n')
ylabel('amplitude')
title('up sampler sequence')
```

OUTPUT WAVEFORM :



RESULT:

Output waveform is observed by using MATLAB.

14. IMPLEMENTATION OF I/D SAMPLING RATE CONVERSION

AIM: To Implement the I/D sampling rate conversion.

APPARATUS:

1. Software:-Matlab 7. 0v
2. PC

THEORY:

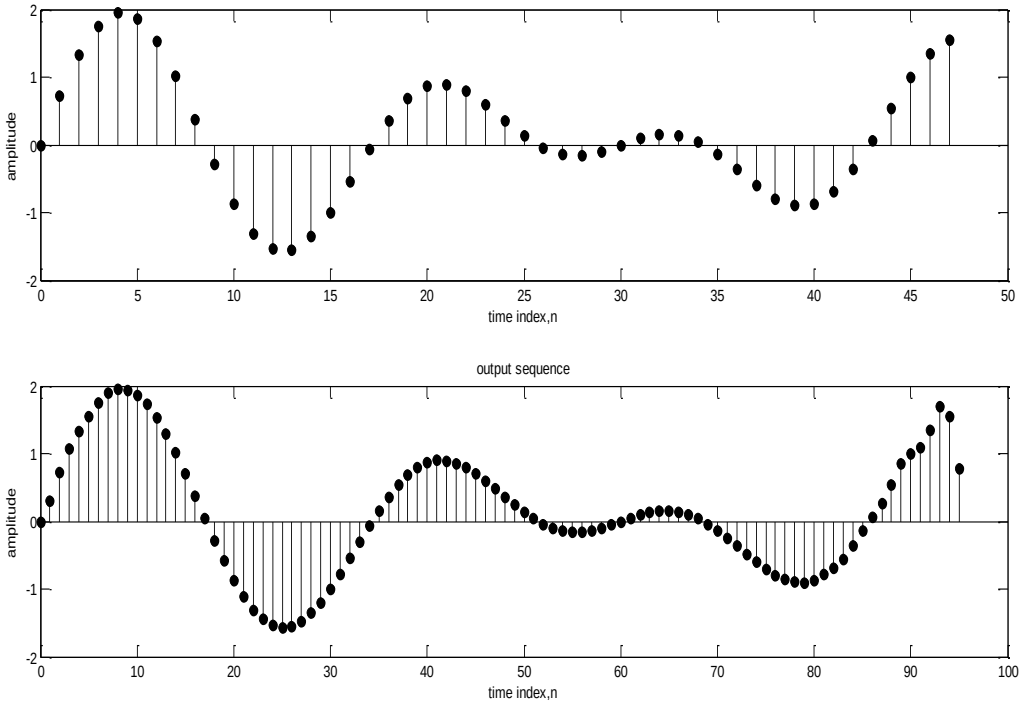
The sampling rate conversion by the factor I/D is done by cascading the interpolation with decimation. Here the interpolation is doing first and decimation later because the interpolation presers the spectral shape of the given input signal.

PROGRAM:

```
% Illustration of Sampling Rate Alteration by
% a Ratio of Two Integers
%
clc:
close all;
clear all;
clf;
N = input('Length of input signal = ');
L = input('Up-sampling factor = ');
M = input('Down-sampling factor = ');
f1 = input('Frequency of first sinusoid = ');
f2 = input('Frequency of second sinusoid = ');
%  Generate the input sequence
n = 0:N-1;
```

```
x = sin(2*pi*f1*n) + sin(2*pi*f2*n);
%  Generate the resampled output sequence
y = resample(x,L,M);
%  Plot the input and the output sequences
subplot(2,1,1)
stem(n,x(1:N));
title('Input sequence');
xlabel('Time index n'); ylabel('Amplitude');
subplot(2,1,2)
m=0:N*L/M-1;
stem(m,y(uint8(1:N*L/M)));
title('Output sequence');
xlabel('Time index n'); ylabel('Amplitude');
```

OUTPUT WAVE FORMS:



RESULT:

Output waveform is observed by using MATLAB.

15. AUDIO APPLICATIONS

Aim: - To Perform Audio applications such as to plot a time and frequency display of
microphone plus a cosine using DSP. Read a wav file and match with their respective
spectrograms.

Equipments:-

1) Operating System - Windows XP
2) Software - CC STUDIO 3.1
3) Software Matlab 7.0
4) DSK 6713 DSP Trainer kit.
5) USB Cable
6) Power supply

Theory:

Spectrogram with RTDX using MATLAB

This version of project makes use of RTDX with MATLAB for transferring data from the DSK to the PC host. This section introduces configuration file(.CDB) file and RTDX with MATLAB.

This project uses source program spectrogram_rtdx_mtl.c that runs on the DSK which computes 256 point FFT and enables an RTDX output channel to write/send the resulting FFT data to the PC running MATLAB for finding the spectrogram. A total of N/2 (128 points)are sent. The (.CDB) configuration file is used to set interrupt INT11. From this configuration file select Input/Output ◊ RTDX. Right click on properties and change the RTDX buffer size to 8200.

Within CCS, select tools ◊ RTDX ◊ Configure to set the host buffer size to 2048(from
1024).

An input signal is read in blocks of 256 samples. Each block of data is then multiplied with a hamming window of length 256 points. The FFT of the windowed data is calculated and squared. Half of the resulting FFT of each block of 256 points is then transferred to the PC running MATLAB to find the specrtrogram.

Spectrogram_rtdx_mtl.c Time-Frequency analysis of signals Using RTDX-MATLAB

```
#include "dsk6713_aic23.h"      //codec-DSK support file
Uint32 fs=DSK6713_AIC23_FREQ_8KHZ;     //set sampling rate
#include <rtdx.h>           //RTDX support file
#include <math.h>
#include "hamming.cof"              //Hamming window coefficients
#define PTS 256          //# of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n);          //FFT prototype
float iobuffer[PTS],iobuffer1[PTS],a[PTS];     //input and output buffer
float x1[PTS];                    //intermediate buffer
short i;                 //general purpose index variable
int j, k,l, curr_block = 0;              //index variables
short buffercount = 0;              //number of new samples in iobuffer
short flag = 0;                //set to 1 by ISR when iobuffer full
COMPLEX w[PTS];                   //twiddle constants stored in w
```

```
COMPLEX samples[PTS];                  //primary working buffer
RTDX_CreateOutputChannel(ochan);    //create output channel C6x->PC


main()
{
 for (i = 0 ; i<PTS ; i++)          //set up twiddle constants in w
  {
w[i].real = cos(2*PI*i/512.0);     //Re component of twiddle constants
   w[i].imag =-sin(2*PI*i/512.0);     //Im component of twiddle constants
  }
 comm_intr();                    //init DSK, codec, McBSP

 while(!RTDX_isOutputEnabled(&ochan))      //wait for PC to enable RTDX
   puts("\n\n Waiting . . . ");           //while waiting
 for(l=0;l<256;l++)
  a[l]=cos(2*3.14*1500*l/8000);
 for(k=0;k<5000;k++)             //infinite loop
 {
  while (flag == 0) ;                 //wait until iobuffer is full
  flag = 0;                          //reset flag
  for (i = 0 ; i < PTS ; i++)         //swap buffers
   { iobuffer1[i]=iobuffer[i]+a[i];
     samples[i].real=h[i]*iobuffer1[i];  //multiply by Hamming window coeffs
     iobuffer1[i] = x1[i];               //process frame to iobuffer
   }
  for (i = 0 ; i < PTS ; i++)
```

```
    samples[i].imag = 0.0;              //imag components = 0
  FFT(samples,PTS);                     //call C-coded FFT function
  for (i = 0 ; i < PTS ; i++)        //compute square of FFT magnitude
   {
     x1[i] = (samples[i].real*samples[i].real
        + samples[i].imag*samples[i].imag)/16;   //FFT data scaling
 }

  RTDX_write(&ochan, x1, sizeof(x1)/2);     //send 128 samples to PC
 }                          //end of infinite loop
}                    //end of main

interrupt void c_int11()              //ISR
{
 output_sample((short)(iobuffer[buffercount]));   //out from iobuffer
 iobuffer[buffercount++]=(short)(input_sample()); //input to iobuffer
 if (buffercount >= PTS)          //if iobuffer full
 {
 buffercount = 0;          /reinit buffercount
 flag = 1;            //reset flag
 }
}

FFT.c  C callable FFT function in C
#define PTS 256          //# of points for FFT
typedef struct {float real,imag;} COMPLEX;
extern COMPLEX w[PTS];             //twiddle constants stored in w
```

```
void FFT(COMPLEX *Y, int N)      //input sample array, # of points
{
 COMPLEX temp1,temp2;            //temporary storage variables
 int i,j,k;                      //loop counter variables

int upper_leg, lower_leg;        //index of upper/lower butterfly leg
 int leg_diff;                   //difference between upper/lower leg
int num_stages = 0;              //number of FFT stages (iterations)
 int index, step;                //index/step through twiddle constant
 i = 1;                          //log(base2) of N points= # of stages
 do
  {
   num_stages +=1;
   i = i*2;
  }while (i!=N);
 leg_diff = N/2;                 //difference between upper&lower legs
 step = 512/N;                   //step between values in twiddle.h
 for (i = 0;i < num_stages; i++)  //for N-point FFT
  {
   index = 0;
   for (j = 0; j < leg_diff; j++)
    {
     for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
      {
       lower_leg = upper_leg+leg_diff;
   temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
```

```
    temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
    temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
    temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
    (Y[lower_leg]).real = temp2.real*(w[index]).real

      -temp2.imag*(w[index]).imag;
    (Y[lower_leg]).imag = temp2.real*(w[index]).imag
         +temp2.imag*(w[index]).real;
    (Y[upper_leg]).real = temp1.real;
    (Y[upper_leg]).imag = temp1.imag;
   }
index += step;
 }
  leg_diff = leg_diff/2;
  step *= 2;
   }
    j = 0;
    for (i = 1; i < (N-1); i++)     //bit reversal for resequencing data
   {
  k = N/2;
  while (k <= j)
   {
    j = j - k;
    k = k/2;
   }
  j = j + k;
  if (i<j)
```

```
      {
       temp1.real = (Y[j]).real;
       temp1.imag = (Y[j]).imag;

  (Y[j]).real = (Y[i]).real;
       (Y[j]).imag = (Y[i]).imag;
       (Y[i]).real = temp1.real;
       (Y[i]).imag = temp1.imag;
      }
    }
   return;
}
```

Spectrogram_RTDX.m For spectrogram plot using RTDX with MATLAB

```
clc;
ccsboardinfo          %board info
cc=ccsdsp('boardnum',0);      %set up CCS object
reset(cc);            %reset board
visible(cc,1);          %for CCS window
enable(cc.rtdx);          %enable RTDX
if ~isenabled(cc.rtdx);
  error('RTDX is not enabled')
end
cc.rtdx.set('timeout',50);      %set 50sec timeout for RTDX
open(cc,'spectrogram1.pjt');      %open CCS project
load(cc,'./debug/spectrogram1.out');    %load executable file
```

```
run(cc);          %run program
configure(cc.rtdx,2048,1);      %configure one RTDX channel
open(cc.rtdx,'ochan','r');    %open output channel

pause(3)          %wait for RTDX channel to open

enable(cc.rtdx,'ochan');      %enable channel from DSK
isenabled(cc.rtdx,'ochan');
M = 256;              %window size
N = round(M/2);
B = 128;            %No. of blocks (128)
fs = 8000;              %sampling rate
t=(1:B)*(M/fs);         %spectrogram axes generation
f=((0:(M-1)/2)/(M-1))*fs;
set(gcf,'DoubleBuffer','on');
y = ones(N,B);
column = 1;
set(gca,'NextPlot','add');
axes_handle = get(gcf,'CurrentAxes');
set(get(axes_handle,'XLabel'),'String','Time (s)');
set(get(axes_handle,'YLabel'),'String','Frequency (Hz)');
set(get(axes_handle,'Title'),'String','\fontname{times}\bf Real-Time Spectrogram');
set(gca,'XLim', [0 4.096]);
set(gca,'YLim', [0 4000]);
set(gca,'XLimMode','manual');
set(gca,'YLimMode','manual');
for i = 1:32768
  w=readmsg(cc.rtdx,'ochan','single');    %read FFT data from DSK
```

```
  w=double(w(1:N));
  y(:, column) = w';

imagesc(t,f,dB(y));        %plot spectrogram
  column = mod(column, B) + 1;
end
halt(cc);            %halt processor


close(cc.rtdx,'ochan');                %close channel
clear cc           %clear object
```

Procedure:

1. Create a new project with name spectrogram.pjt.
2. Open "spectrogram.cdb" from given CD and save it in your new project folder.
3. Copy the following files from the CD to your new project folder

1) c6713dskinit . c
2) FFT.c
3) spectrogram_rtdx_mtl.c
4) c6713dskinit . h
5) hamming.cof
6) spectrogram_RTDX.m

4. Add "spectrogram.cdb", "c6713dskinit.c" and "spectrogram_rtdx_mtl.c" to the current project.

Path ◊ □C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib□□

5. Set the following compiler options.

Select Project ◊ Build options.

Select the following for compiler option with Basic (for category):

(1) c671x{mv6710} (for target version)

(2) Full symbolic debug (for Generate Debug info)

(3) Speed most critical(for Opt Speed vs. Size)

(4) None (for Opt Level and Program Level Opt)

Select The Preprocessor Category and Type for Define Symbols{d}: CHIP_6713, and from Feedback category, select for Interlisting: OPT / C and ASM{-s}

6. Build project.

7. Close CCS

8. Open MATLAB and Run spectrogram_RTDX.m . within MATLAB ,CCS will enable RTDX and will load and run the COFF(.out) executable file. Then MATLAB will plot the spectrogram of an input signal .

Result:- Thus Audio application is performed and spectrogram of an input signal is plotted using Matlab.

16. NOISE REMOVAL

Aim:- To

1) Add noise above 3kHz and then remove (using adaptive filters)
2) Interference suppression using 400 Hz tone

Equipments:-

1) Operating System - Windows XP
2) Software - CC STUDIO 3
3) DSK 6713 DSP Trainer kit.
4) USB Cable
5) Power supply
6) CRO
7) Function Generator

Theory:- Adaptive filters are best used in cases where signal conditions or system parameters are slowly changing and the filter is to be adjusted to compensate for this change. The least mean squares (LMS) criterion is a search algorithm that can be used to provide the strategy for adjusting the filter coefficients.

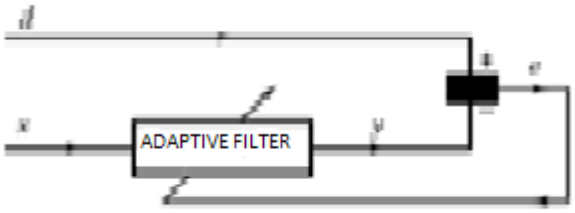


Fig 1 Basic adaptive filter structure.

In conventional FIR and IIR digital filters, it is assumed that the process parameters to determine the filter characteristics are known. They may vary with time, but the nature of the variation is assumed to be known. In many practical

problems, there may be a large uncertainty in some parameters because of inadequate prior test data about the process. Some parameters might be expected to change with time, but the exact nature of the change is not predictable. In such cases it is highly desirable to design the filter to be self-learning, so that it can adapt itself to the situation at hand. The coefficients of an adaptive filter are adjusted to compensate for changes in input signal, output signal, or system parameters. Instead of being rigid, an adaptive system can learn the signal characteristics and track slow changes. An adaptive filter can be very useful when there is uncertainty about the characteristics of a signal or when these characteristics change.

Program:-

```
#include "NCcfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
#define beta 1E-13                    //rate of convergence
#define N 30                          //adaptive FIR filter length-vary this parameter & observe

float delay[N];
float w[N];

DSK6713_AIC23_Config config = { \
   0x0017,           /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,           /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
```

```
   0x00d8,          /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone
volume */  \
   0x00d8,         /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone
volume */ \
   0x0011,        /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */
\
   0x0000,        /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */
\
   0x0000,        /* 6 DSK6713_AIC23_POWERDOWN  Power down control */
\
   0x0043,     /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x0081,        /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */
\
   0x0001      /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};
                /* main() - Main code routine, initializes BSL and generates tone*/
void main()
{

  DSK6713_AIC23_CodecHandle hCodec;

     int l_input, r_input,l_output, r_output,T;
                                    /* Initialize the board support library, must be
called first */
     DSK6713_init();
     hCodec = DSK6713_AIC23_openCodec(0, &config);          /* Start the codec
*/
     DSK6713_AIC23_setFreq(hCodec, 1);
```

```
  for (T = 0; T < 30; T++)              //Initialize the adaptive FIR coeffs=0
  {     w[T] = 0;          //init buffer for weights
    delay[T] = 0;             //init buffer for delay samples
  }

  while(1)
      { /* Read a sample to the left channel */
     while (!DSK6713_AIC23_read(hCodec,&l_input));
     /* Read a sample to the right channel */
     while (!DSK6713_AIC23_read(hCodec, &r_input));
     l_output=(short int)adaptive_filter(l_input,r_input);
                                    r_output=l_output;
     while (!DSK6713_AIC23_write(hCodec, l_output));          /* Send o/p to the
left channel */
    while (!DSK6713_AIC23_write(hCodec, r_output));          /* Send o/p to the
right channel* /
       }
   DSK6713_AIC23_closeCodec(hCodec);                         /* Close the codec
*/
}
 signed int adaptive_filter(int l_input,int r_input)        //ISR

 {
  short i,output;
  float yn=0, E=0, dplusn=0,desired,noise;
    desired = l_input;
```

```
 noise = r_input;
 dplusn = (short)(desired + noise);        //desired+noise
 delay[0] = noise;           //noise as input to adapt FIR

 for (i = 0; i < N; i++)          //to calculate out of adapt FIR
 yn += (w[i] * delay[i]);       //output of adaptive filter
 E = (desired + noise) - yn;      //"error" signal=(d+n)-yn
 for (i = N-1; i >= 0; i--)       //to update weights and delays
  {
 w[i] = w[i] + beta*E*delay[i]; //update weights
    delay[i] = delay[i-1];       //update delay samples
  }

   //output=((short)E);          //error signal as overall output
     output=((short)dplusn);          //output (desired+noise)
         //overall output result
 return(output);
}
```

Procedure:-

1. Switch on the DSP Board.
2. Open the code composer studio
3. Create a new project
4. Initialise on board codec
5. Add the above c source file to the project

6. Input a desired sinusoidal signal into the left channel and noise signal of 3KHz into the

right channel

7. Build the project
8. Load the generated object file onto the target board
9. Run the program
10. Observe the waveform that appears on the CRO screen

Verify that the 3KHz noise signal is being cancelled gradually.

Result:- Thus noise signal cancellation using adaptive filters is verified.

17. IMPULSE RESPONSE OF FIRST AND SECOND ORDER SYSTEM

AIM: To find the impulse response of a first and second order systems

APPARATUS:

1. Software:-Matlab 7.0v
2. PC

THEORY:

The impulse response of a system is given by the transfer function.

If the transfer function of a system is given by H(s), then the impulse response of a system is given by h(t)

where h(t) is the inverse Laplace Transform of H(s).

$h(t) = \mathfrak{L}^{-1}\{H(s)\}$

PROGRAM:

```
clc;

close all;

clear all;

x=input('type the x vector');

y=input('type the y vector');

N=input('type the desired length of the output sequence N');

n=0:N-1;

imp=[1,zeros(1,N-1)];

h=filter(x,y,imp);

disp('the impulse response of LTI system is');

disp(h);
```

```
stem(n,h);
xlabel('time index n');
ylabel('h(n)');
title('impulse response of LTI system');
```

OUTPUT:

type the x vector[1 -2.4 2.88]

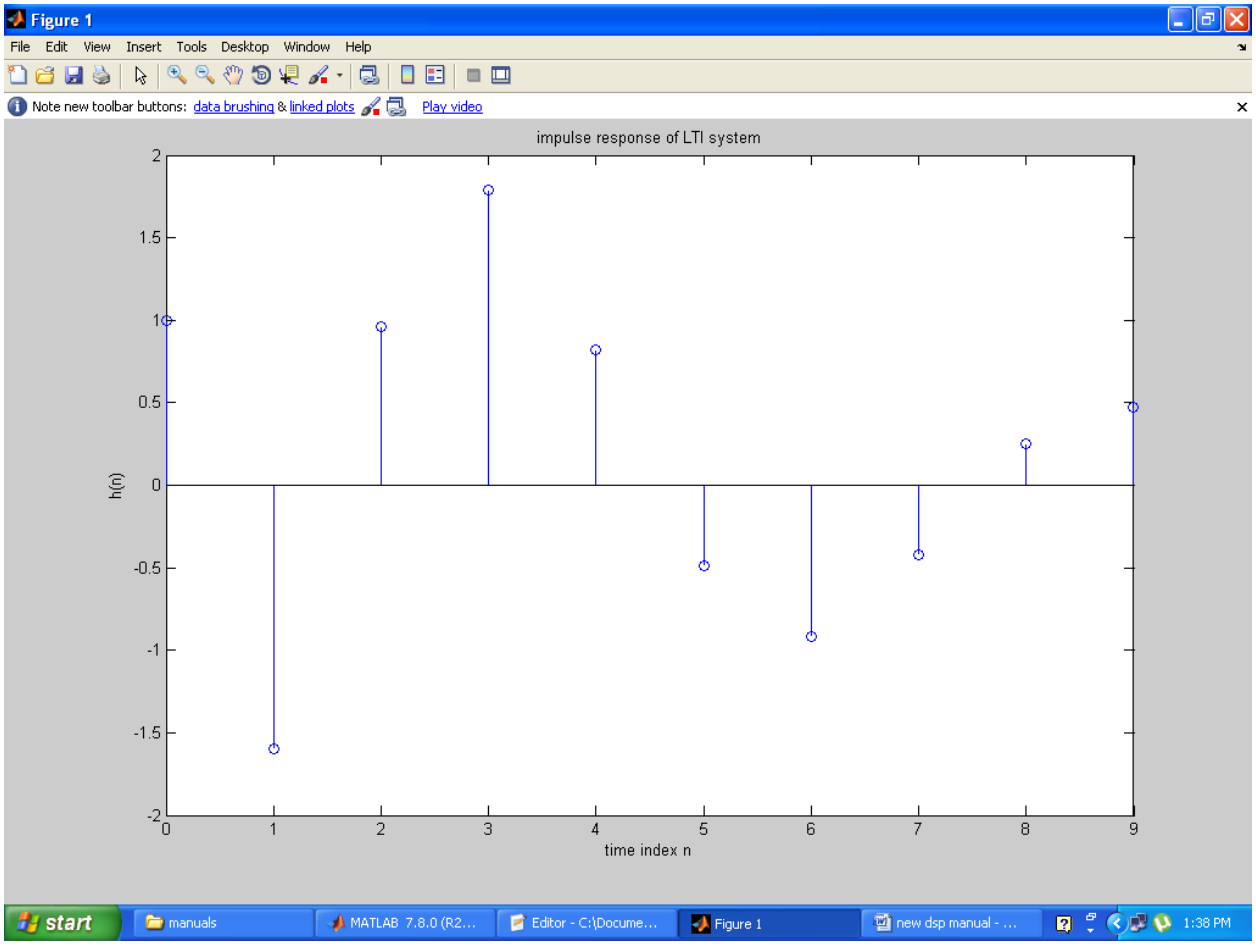
type the y vector[1 -.8 0.64]

type the desired length of the output sequence N10

the impulse response of LTI system is

1.0000 -1.6000 0.9600 1.7920 0.8192 -0.4915 -0.9175 -0.4194 0.2517 0.4698

OUTPUT WAVEFORMS:



RESULT:

Output waveform is observed by using MATLAB.

HARDWARE EXPERIMENTS

1. GENERATION OF SINE WAVE AND SQUARE WAVE

AIM:- To generate a sine wave and square wave using C6713 simulator

EQUIPMENT:

- Operating System - Windows XP
- Software – CODE COMPOSER STUDIO
- DSK 6713 DSP Trainer kit.
- USB Cable
- Power supply
- PC

PROCEDURE:-

1. Open Code Composer Setup and select C6713 simulator, click save and quit
2. Start a new project using „Project□→New‟ pull down menu, save it in a separate directory (C:\My projects) with file name **sinewave.pjt**
3. Create a new source file using File□→ New→Source file menu and save it in the project folder(sinewave.c)
4. Add the source file (sinewave.c) to the project Project□ →Add files to Project→ Select sinewave.c
5. Add the linker command file hello.cmd Project□→Add files to Project (path: C:\CCstudio\tutorial\dsk6713\hello\hello.cmd)
6. Add the run time support library file rts6700.lib Project□→Add files to Project (path: C\CCStudio\cgtools\lib\rts6700.lib)
7. Compile the program using „project →Compile‟ menu or by Ctrl+F7
8. Build the program using „project□ →Build‟ menu or by F7

9. Load the sinewave.out file (from project folder lcconv\Debug) using File→Load Program
10. Run the program using „Debug →Run or F5
11. To view the output graphically Select →View Graph →Time and Frequency 12. Repeat the steps 2 to 11 for square wave

PROGRAM :

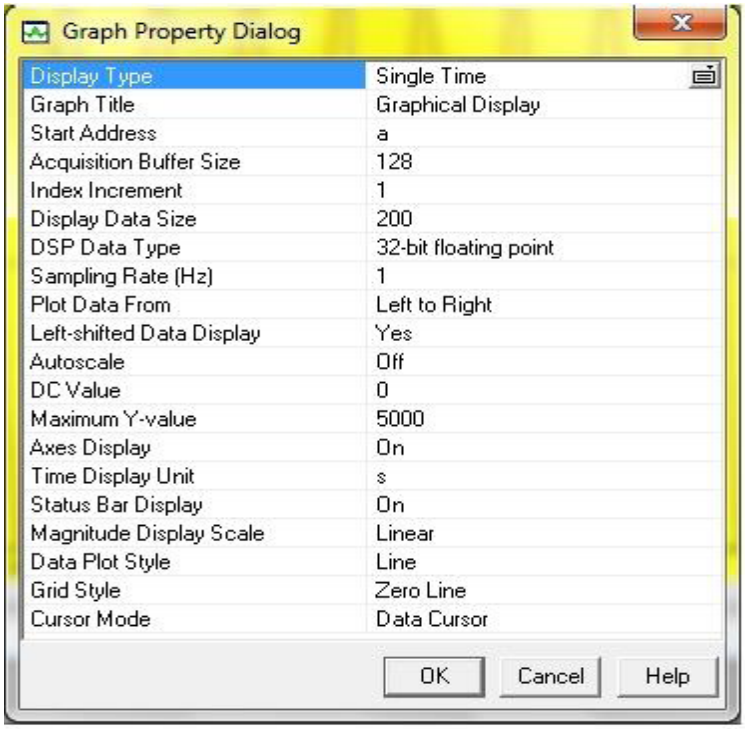
```
#include <stdio.h>
#include <math.h>
float a[500];
void main()
{
int i=0;
for(i=0;i<500;i++)
{
a[i]=sin(2*3.14*10000*i);
}
}
```

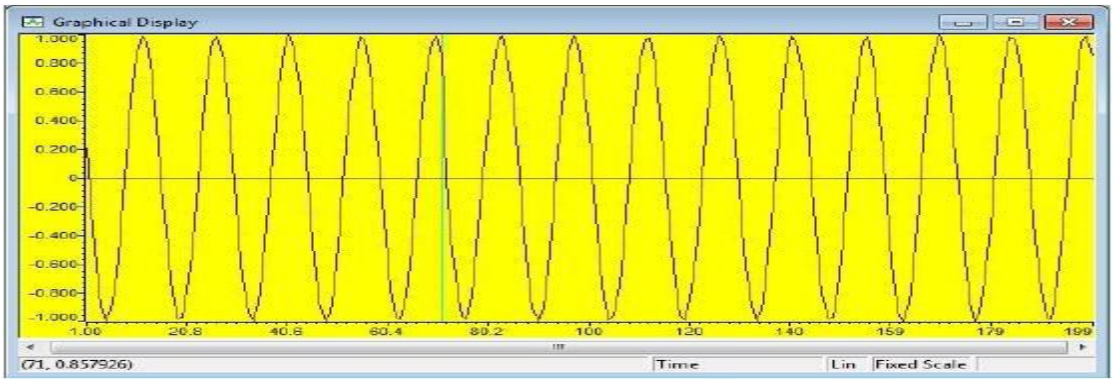
Square wave

```
#include <stdio.h>
#include <math.h> int a[1000];
void main()
{
```

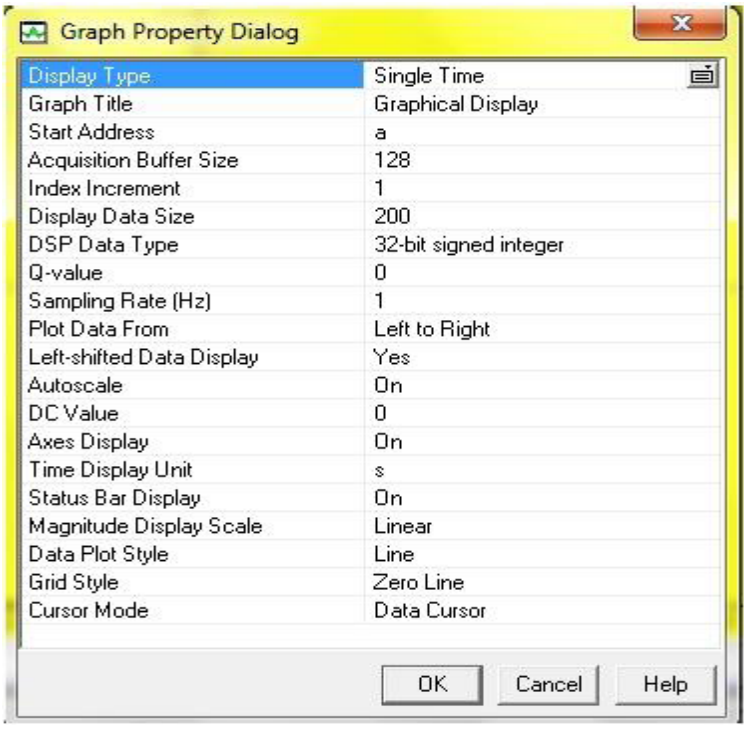
```
int i,j=0;
int b=5;
for(i=0;i<10;i++)
 {
for (j=0;j<=50;j++)
{
a[(50*i)+j]=b;
}
b=b*(-1) ;
}
}
```

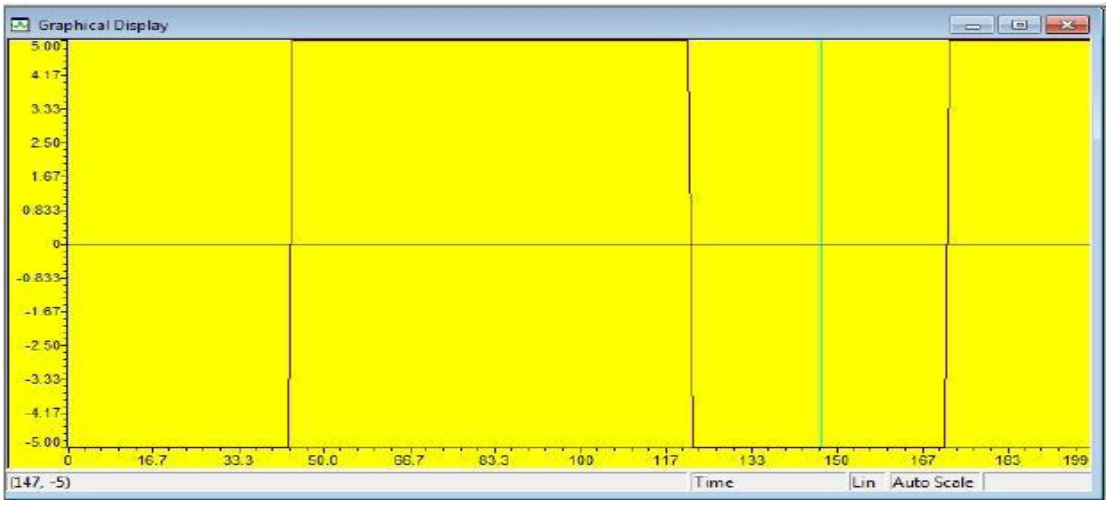
OUTPUT:- Sine wave





OUTPUT SQUARE WAVE:-





RESULT:- Output waveform is observed by using MATLAB.

2. LINEAR CONVOLUTION

AIM: To verify the linear convolution using DSK6713 processor

SOFTWARE REQUIRED:

- Operating System - Windows XP
- Software – CODE COMPOSER STUDIO

HARDWARE REQUIRED:

- DSK 6713 DSP Trainer kit.
- USB Cable
- Power supply
- PC

THEORY:

Linear Convolution Involves the following operations.

1. Folding
2. Multiplication
3. Addition
4. Shifting

These operations can be represented by a Mathematical Expression as follows:

$$y[n] = \sum_{k=-\infty} x[k]h[n-k]$$

x[ ]= Input signal Samples

h[ ]= Impulse response co-efficient.

y[ ]= Convolution output.

n = No. of Input samples

h = No. of Impulse response co-efficient.

'C' PROGRAM TO IMPLEMENT LINEAR CONVOLUTION

PROGRAM:

```
// Linear convolution program in c language using CC Studio
#include<stdio.h> int x[15],h[15],y[15];
main()
{
 int i,j,m,n;
printf("\n enter value for m");
scanf("%d",&m);
 printf("\n enter value for n");
 scanf("%d",&n);
printf("Enter values for i/p x(n):\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("Enter Values for i/p h(n) \n");
 for(i=0;i<n; i++)
scanf("%d",&h[i]);
// padding of zeros
for(i=m;i<=m+n-1;i++)
x[i]=0;
for(i=n;i<=m+n-1;
i++) h[i]=0;
/* convolution operation */
```

```
for(i=0;i<m+n-1;i++)
 {
y[i]=0;
for(j=0;j<=i;j++)
{
 y[i]=y[i]+(x[j]*h[i-j]);
}
}
//displaying the o/p
for(i=0;i<m+n-1;i++)
printf("\n The Value of output y[%d]=%d",i,y[i]);
}
```

INPUT

x[n] = [1 2 3 4]

h[k] =[1 2 3 4]

OUTPUT:

y[i] =

Output:- enter value for m 4

enter value for n 4

Enter values for i/p 1 2 3 4

Enter Values for n 1 2 3 4

The Value of output y[0]=1

The Value of output y[1]=4

The Value of output y[2]=10

The Value of output y[3]=20

The Value of output y[4]=25

The Value of output y[5]=24

The Value of output y[6]=16

RESULT:- Thus linear convolution of two sequences is verified using CC Studio

`

3. CIRCULAR CONVOLUTION

AIM: To verify the Circular Convolution using DSK6713 Processor.

SOFTWARE REQUIRED:

- Operating System - Windows XP

  Software – CODE COMPOSER STUDIO

HARDWARE REQUIRED:

- DSK 6713 DSP Trainer kit.
- USB Cable
- Power supply
- PC

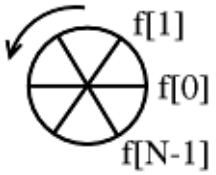
THEORY:

STEPS FOR CYCLIC CONVOLUTION:

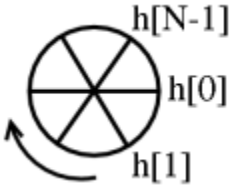
Steps for cyclic convolution are the same as the usual convolution, except all index calculations are done "mod N" = "on the wheel"

Steps for Cyclic Convolution:

Step1: "Plot f[m] and h[−m]



Subfigure 1.1



Subfigure 1.2

Step 2: "Spin" *h*[−m] *n* times Anti Clock Wise (counter-clockwise) to get h[n-m]

(i.e. Simply rotate the sequence, *h*[*n*], clockwise by *n* steps)

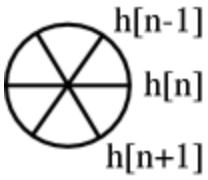


Figure 2: Step 2

Step 3: Pointwise multiply the $f[m]$ wheel and the h[n−m] wheel. sum=$y[n]$

Step 4: Repeat for all $0 \leq n \leq N-1$

PROGRAM :

```
#include<stdio.h>
  int m,n,x[30],h[30],y[30],i,j,temp[30],k,x2[30],a[30];
void main()
{
  printf("  enter the length of the first sequence\n");
  scanf("%d",&m);
  printf("  enter the length of the second sequence\n");
  scanf("%d",&n);
  printf("  enter the first sequence\n");
  for(i=0;i<m;i++)
  scanf("%d",&x[i]);
  printf("  enter the second sequence\n");
  for(j=0;j<n;j++)
  scanf("%d",&h[j]);
```

```
if(m-n!=0)                /*If length of both sequences are not equal*/
{
    if(m>n)               /* Pad the smaller sequence with zero*/
    {
    for(i=n;i<m;i++)
    h[i]=0;
    n=m;
    }
    for(i=m;i<n;i++)
    x[i]=0;
    m=n;
}
y[0]=0;
a[0]=h[0];
for(j=1;j<n;j++)                    /*folding h(n) to h(-n)*/
a[j]=h[n-j];
  /*Circular convolution*/
for(i=0;i<n;i++)
y[0]+=x[i]*a[i];
for(k=1;k<n;k++)
{
```

```
    y[k]=0;
    /*circular shift*/
    for(j=1;j<n;j++)
    x2[j]=a[j-1];
    x2[0]=a[n-1];
    for(i=0;i<n;i++)
     {
          a[i]=x2[i];
          y[k]+=x[i]*x2[i];
     }
   }
    /*displaying the result*/
    printf("  the circular convolution is\n");
    for(i=0;i<n;i++)
    printf("%d \t",y[i]);
    }
```

INPUT:

Eg: x[4]={3, 2, 1,0}

h[4]={1, 1, 0,0}

OUTPUT: y[4]={3, 5, 3,0}

RESULT:

Verified the Circular Convolution using DSK6713 Processor.

4. FINITE IMPULSE RESPONSE (FIR) FILTER IMPLEMENTATION

AIM: To design FIR filter (LP/HP) using windowing technique and verify using the DSP Processor

a) Using rectangular window

b) Using triangular window

c) Using Kaiser Window

SOFTWARE REQUIRED: Code Composer Studio, Matlab

HARDWARE REQUIRED:

- DSP Processor - DSK320C6713
- Computer
- CRO
- Function **g**enerator

THEORY:

A **finite impulse response (FIR)** filter is a type of a discrete-time filter. The impulse response, the filter's response to a Kronecker delta input, is *finite* because it settles to zero in a finite number of sample intervals. This is in contrast to infinite impulse response (IIR) filters, which have internal feedback and may continue to respond indefinitely. The impulse response of an Nth-order FIR filter lasts for N+1 samples, and then dies to zero.

The difference equation that defines the output of an FIR filter in terms of its input is:

$$y[n] = b_0x[n] + b_1x[n-1] + \cdots + b_Nx[n-N]$$

where:

- $x[n]$ is the input signal,

- $y[n]$ is the output signal,
- b_i are the filter coefficients, and
- N is the filter order – an Nth-order filter has $(N + 1)$ terms on the right-hand side; these are commonly referred to as *taps*.

This equation can also be expressed as a convolution of the coefficient sequence b_i with the input signal:

$$y[n] = \sum_{i=0}^{N} b_i x[n-i].$$

That is, the filter output is a weighted sum of the current and a finite number of previous values of the input.

IMPLEMENTATION OF AN FIR FILTER :

Using MATLAB To Determine Filter Coefficients:

A. MATLAB Program to generate 'FIR Filter-Low Pass' Coefficients:

```
% FIR Low  pass filters using rectangular, triangular and kaiser windows

% sampling rate - 8000

order = 30;

cf=[500/4000,1000/4000,1500/4000];          % cf--> contains set of cut-off
                                              frequencies[Wc ]

% cutoff frequency - 500

b_rect1=fir1(order,cf(1),boxcar(31));       %% Rectangular
```

```
b_tri1=fir1(order,cf(1),bartlett(31));          %%Triangular

b_kai1=fir1(order,cf(1),kaiser(31,8));          %%Kaisar [where 8-->Beta Co-efficient]

% cutoff frequency - 1000

b_rect2=fir1(order,cf(2),boxcar(31));

b_tri2=fir1(order,cf(2),bartlett(31));

b_kai2=fir1(order,cf(2),kaiser(31,8));

% cutoff frequency - 1500

b_rect3=fir1(order,cf(3),boxcar(31));

b_tri3=fir1(order,cf(3),bartlett(31));

b_kai3=fir1(order,cf(3),kaiser(31,8));
```

T.1 :MATLAB generated Coefficients for FIR Low Pass Rectangular filter

Cutoff -500Hz

float b_rect1[31]={-0.008982,-0.017782,-0.025020,-0.029339,-0.029569,-0.024895,

-0.014970,0.000000,0.019247,0.041491,0.065053,0.088016,0.108421,0.124473,0.134729,

0.138255,0.134729,0.124473,0.108421,0.088016,0.065053,0.041491,0.019247,0.000000,

-0.014970,-0.024895,-0.029569,-0.029339,-0.025020,-0.017782,-0.008982};

Cutoff -1000Hz

float b_rect2[31]={-0.015752,-0.023869,-0.018176,0.000000,0.021481,0.033416,0.026254,-0.000000,-0.033755,-0.055693,-0.047257,0.000000,0.078762,0.167080,0.236286,0.262448,

0.236286,0.167080,0.078762,0.000000,-0.047257,-0.055693,-0.033755,-0.000000,0.026254,

0.033416,0.021481,0.000000,-0.018176,-0.023869,-0.015752};

Cutoff -1500Hz

float b_rect2[31]={-0.020203,-0.016567,0.009656,0.027335,0.011411,-0.023194,-0.033672,

0.000000,0.043293,0.038657,-0.025105,-0.082004,-0.041842,0.115971,0.303048,0.386435,

0.303048,0.115971,-0.041842,-0.082004,-0.025105,0.038657,0.043293,0.000000,-0.033672,

-0.023194,0.011411,0.027335,0.009656,-0.016567,-0.020203};

T.2 : MATLAB generated Coefficients for FIR Low Pass Triangular filter

Cutoff -500Hz

float b_tri1[31]={0.000000,-0.001185,-0.003336,-0.005868,-0.007885,-0.008298,-0.005988,
0.000000,0.010265,0.024895,0.043368,0.064545,0.086737,0.107877,0.125747,0.138255,
0.125747,0.107877,0.086737,0.064545,0.043368,0.024895,0.010265,0.000000,-0.005988,
-0.008298,-0.007885,-0.005868,-0.003336,-0.001185,0.000000};

Cutoff -1000Hz

float b_tri2[31]={0.000000,-0.001591,-0.002423,0.000000,0.005728,0.011139,0.010502,
-0.000000,-0.018003,-0.033416,-0.031505,0.000000,0.063010,0.144802,0.220534,0.262448,
0.220534,0.144802,0.063010,0.000000,-0.031505,-0.033416,-0.018003,-0.000000,0.010502,
0.011139,0.005728,0.000000,-0.002423,-0.001591,0.000000};

Cutoff -1500Hz

float b_tri3[31]={0.000000,-0.001104,0.001287,0.005467,0.003043,-0.007731,-0.013469,
0.000000,0.023089,0.023194,-0.016737,-0.060136,-0.033474,0.100508,0.282844,0.386435,
0.282844,0.100508,-0.033474,-0.060136,-0.016737,0.023194,0.023089,0.000000,-0.013469,
-0.007731,0.003043,0.005467,0.001287,-0.001104,0.000000};

T.3 : MATLAB generated Coefficients for FIR Low Pass Kaiser filter

Cutoff -500Hz

float b_kai1[31]={-0.000019,-0.000170,-0.000609,-0.001451,-0.002593,-0.003511,-0.003150,0.000000,0.007551,0.020655,0.039383,0.062306,0.086494,0.108031,0.122944,0.128279,0.122944,0.108031,0.086494,0.062306,0.039383,0.020655,0.007551,0.000000,-0.003150,-0.003511,-0.002593,-0.001451,-0.000609,-0.000170,-0.000019};

Cutoff -1000Hz

float b_kai2[31]={-0.000035,-0.000234,-0.000454,0.000000,0.001933,0.004838,0.005671,-0.000000,-0.013596,-0.028462,-0.029370,0.000000,0.064504,0.148863,0.221349,0.249983,0.221349,0.148863,0.064504,0.000000,-0.029370,-0.028462,-0.013596,-0.000000,0.005671,0.004838,0.001933,0.000000,-0.000454,-0.000234, -0.000035};

Cutoff -1500Hz

float b_kai3[31]={-0.000046,-0.000166,0.000246,0.001414,0.001046,-0.003421,-0.007410,0.000000,0.017764,0.020126,-0.015895,-0.060710,-0.034909,0.105263,0.289209,0.374978,0.289209,0.105263,-0.034909,-0.060710,-0.015895,0.020126,0.017764,0.000000,-0.007410,-0.003421,0.001046,0.001414,0.000246,-0.000166, -0.000046};

% FIR HIGH PASS FILTERS USING RECTANGULAR, TRIANGULAR AND KAISER WINDOWS

```
% sampling rate - 8000
order = 30;

cf=[400/4000,800/4000,1200/4000]; %% cf --> contains set of cut-off frequencies[Wc]

% cutoff frequency - 400
b_rect1=fir1(order,cf(1),'high',boxcar(31))
b_tri1=fir1(order,cf(1),'high',bartlett(31))
b_kai1=fir1(order,cf(1),'high',kaiser(31,8))    %%e Kaiser(31,8)--> 'beta' is '8'

% cutoff frequency - 800
b_rect2=fir1(order,cf(2),'high',boxcar(31))
b_tri2=fir1(order,cf(2),'high',bartlett(31))
b_kai2=fir1(order,cf(2),'high',kaiser(31,8))

% cutoff frequency - 1200
b_rect3=fir1(order,cf(3),'high',boxcar(31))
b_tri3=fir1(order,cf(3),'high',bartlett(31))
b_kai3=fir1(order,cf(3),'high',kaiser(31,8))
```

T.4 : MATLAB generated Coefficients for FIR High Pass Rectangular filter

Cutoff -400Hz

float b_rect1[31]={0.021665,0.022076,0.020224,0.015918,0.009129,-0.000000,-0.011158,
-0.023877,-0.037558,-0.051511,-0.064994,-0.077266,-0.087636,-0.095507,-.100422,0.918834,
-0.100422,-0.095507,-0.087636,-0.077266,-0.064994,-0.051511,-0.037558,-0.023877,
-0.011158,-0.000000,0.009129,0.015918,0.020224,0.022076,0.021665};

Cutoff -800Hz

float b_rect2[31]={0.000000,-0.013457,-0.023448,-0.025402,-0.017127,-0.000000,0.020933,
0.038103,0.043547,0.031399,0.000000,-0.047098,-0.101609,-0.152414,-0.188394,0.805541,
-0.188394,-0.152414,-0.101609,-0.047098,0.000000,0.031399,0.043547,0.038103,0.020933,
-0.000000,-0.017127,-0.025402,-0.023448,-0.013457,0.000000};

Cutoff -1200Hz

float b_rect3[31]={-0.020798,-0.013098,0.007416,0.024725,0.022944,-0.000000,-0.028043,
-0.037087,-0.013772,0.030562,0.062393,0.045842,-0.032134,-0.148349,-0.252386,0.686050,
-0.252386,-0.148349,-0.032134,0.045842,0.062393,0.030562,-0.013772,-0.037087,-0.028043,
-0.000000,0.022944,0.024725,0.007416,-0.013098,-0.020798};

T.5: MATLAB generated Coefficients for FIR High Pass Triangular filter

Cutoff -400Hz

float b_tri1[31]={0.000000,0.001445,0.002648,0.003127,0.002391,-0.000000,-0.004383,
-0.010943,-0.019672,-0.030353,-0.042554,-0.055647,-0.068853,-0.081290,-0.092048,
0.902380,-0.092048,-0.081290,-0.068853,-0.055647,-0.042554,-0.030353,-0.019672,
-0.010943,-0.004383,-0.000000,0.002391,0.003127,0.002648,0.001445,0.000000};

Cutoff -800Hz

float b_tri2[31]={0.000000,-0.000897,-0.003126,-0.005080,-0.004567,-0.000000,0.008373,
0.017782,0.023225,0.018839,0.000000,-0.034539,-0.081287,-0.132092,-0.175834,0.805541,
-0.175834,-0.132092,-0.081287,-0.034539,0.000000,0.018839,0.023225,0.017782,0.008373,
-0.000000,-0.004567,-0.005080,-0.003126,-0.000897,0.000000};

Cutoff -1200Hz

float b_tri3[31]={0.000000,-0.000901,0.001021,0.005105,0.006317,-0.000000,-0.011581,
-0.017868,-0.007583,0.018931,0.042944,0.034707,-0.026541,-0.132736,-0.243196,0.708287,
-0.243196,-0.132736,-0.026541,0.034707,0.042944,0.018931,-0.007583,-0.017868,-0.011581,
-0.000000,0.006317,0.005105,0.001021,-0.000901,0.000000};

T.6: MATLAB generated Coefficients for FIR High Pass Kaiser filter

Cutoff -400Hz

float b_kai1[31]={0.000050,0.000223,0.000520,0.000831,0.000845,-0.000000,-0.002478,
-0.007437,-0.015556,-0.027071,-0.041538,-0.057742,-0.073805,-0.087505,-0.096739,
0.899998,-0.096739,-0.087505,-0.073805,-0.057742,-0.041538,-0.027071,-0.015556,
-0.007437,-0.002478,-0.000000,0.000845,0.000831,0.000520,0.000223,0.000050};

Cutoff -800Hz

float b_kai2[31]={0.000000,-0.000138,-0.000611,-0.001345,-0.001607,-0.000000,0.004714,
0.012033,0.018287,0.016731,0.000000,-0.035687,-0.086763,-0.141588,-0.184011,0.800005,
-0.184011,-0.141588,-0.086763,-0.035687,0.000000,0.016731,0.018287,0.012033,0.004714,
-0.000000,-0.001607,-0.001345,-0.000611,-0.000138,0.000000};

Cutoff -1200Hz

float b_kai3[31]={-0.000050,-0.000138,0.000198,0.001345,0.002212,-0.000000,-0.006489,
-0.012033,-0.005942,0.016731,0.041539,0.035687,-0.028191,-0.141589,-0.253270,0.700008,
-0.253270,-0.141589,-0.028191,0.035687,0.041539,0.016731,-0.005942,-0.012033,-0.006489,
-0.000000,0.002212,0.001345,0.000198,-0.000138,-0.000050};

C Program To Implement FIR Filter:

fir.c

```
#include "filtercfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
float filter_Coeff[] ={0.000000,-0.001591,-0.002423,0.000000,0.005728,
0.011139,0.010502,-0.000000,-0.018003,-0.033416,-0.031505,0.000000,
```

```
0.063010,0.144802,0.220534,0.262448,0.220534,0.144802,0.063010,0.000000,
-0.031505,-0.033416,-0.018003,-0.000000,0.010502,0.011139,0.005728,
0.000000,-0.002423,-0.001591,0.000000 };
static short in_buffer[100];
DSK6713_AIC23_Config config = {\
 0x0017, /* 0 DSK6713_AIC23_LEFTINVOL Leftline input channel volume */\
 0x0017, /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel
volume*/\
 0x00d8, /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume
*/\
0x00d8, /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume
*/\
   0x0011,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */\
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */\
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */\
   0x0043,  /* 7 DSK6713_AIC23_DIGIF    Digital audio interface format */\
   0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */\
   0x0001   /* 9 DSK6713_AIC23_DIGACT   Digital interface activation */   \
};
/* *  main() - Main code routine, initializes BSL and generates tone * */
void main()
{
   DSK6713_AIC23_CodecHandle hCodec;
```

```
Uint32 l_input, r_input,l_output, r_output;


/* Initialize the board support library, must be called first */
DSK6713_init();
/* Start the codec */
hCodec = DSK6713_AIC23_openCodec(0, &config);
DSK6713_AIC23_setFreq(hCodec, 1);
  while(1)
   { /* Read a sample to the left channel */
                while (!DSK6713_AIC23_read(hCodec, &l_input));


                /* Read a sample to the right channel */
                while (!DSK6713_AIC23_read(hCodec, &r_input));


                l_output=(Int16)FIR_FILTER(&filter_Coeff ,l_input);
                   r_output=l_output;
                /* Send a sample to the left channel */
      while (!DSK6713_AIC23_write(hCodec, l_output));
      /* Send a sample to the right channel */
      while (!DSK6713_AIC23_write(hCodec, r_output));
   }
/* Close the codec */
DSK6713_AIC23_closeCodec(hCodec);
```

```
}
signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x;  /* new input at buffer[0]  */
for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer   */
for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

Code Configuration Procedure:

```
#include "dsk6713.h"
#include "dsk6713_aic23.h"
/* Codec configuration settings */
DSK6713_AIC23_Config config = { \
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00d8,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
```

```
    0x00d8,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone
volume */ \
    0x0011,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */     \
    0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */    \
    0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */
\
    0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
    0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */          \
    0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};
/* main() - Main code routine, initializes BSL and generates tone */
void main()
{
    DSK6713_AIC23_CodecHandle hCodec;
    int l_input, r_input,l_output, r_output;
    /* Initialize the board support library, must be called first */
    DSK6713_init();
    /* Start the codec */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
     /*set codec sampling frequency*/
    DSK6713_AIC23_setFreq(hCodec, 3);
     while(1)
        {
```

```
        /* Read a sample to the left channel */
        while (!DSK6713_AIC23_read(hCodec, &l_input));

        /* Read a sample to the right channel */
        while (!DSK6713_AIC23_read(hCodec, &r_input));
        /* Send a sample to the left channel */
         while (!DSK6713_AIC23_write(hCodec, l_input));
         /* Send a sample to the right channel */
         while (!DSK6713_AIC23_write(hCodec, l_input));
      }
   /* Close the codec */
   DSK6713_AIC23_closeCodec(hCodec);
}
```

RESULT:

FIR filter (LP/HP) using windowing technique and verified using the DSP Processor .

5. IMPLEMENTATION OF INFINITE IMPULSE RESPONSE FILTER

AIM: To design IIR filter (LP/HP) using Low pass Butterworth and Chebyshev filters technique and verify using the DSP processor

SOFTWARE REQUIRED: 1. CODE COMPOSER STUDIO

2.MATLAB 7.0V

EQUIPMENT REQUIRED:

1.CRO

2. Function generator

3. BNC Probs and connecting wires

THEORY:

The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

$$H(z)\frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=1}^{N} a_k Z^{-k}}$$

The difference equation for such a system is described by the following:

$$y(n)=\sum_{k=0}^{M} b_k x(n-k) \; + \; \sum_{k=1}^{N} a_k y(n-k)$$

M and N are order of the two polynomials bk and ak are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That's why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

MATLAB PROGRAM TO GENRATE FILTER CO-EFFICIENTS

```
% IIR Low pass Butterworth and Chebyshev filters
% sampling rate - 24000
order = 2;
cf=[2500/12000,8000/12000,1600/12000];

% cutoff frequency - 2500
[num_bw1,den_bw1]=butter(order,cf(1));
[num_cb1,den_cb1]=cheby1(order,3,cf(1));
% cutoff frequency - 8000
[num_bw2,den_bw2]=butter(order,cf(2));
[num_cb2,den_cb2]=cheby1(order,3,cf(2));
fid=fopen('IIR_LP_BW.txt','wt');
fprintf(fid,'\t\t----------Pass band range: 0-2500Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Monotonic-----\n\n\');
```

```
fprintf(fid,'\n float num_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw1);
fprintf(fid,'\nfloat den_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw1);
fprintf(fid,'\n\n\n\t\t----------Pass band range: 0-8000Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Monotonic-----\n\n');
fprintf(fid,'\nfloat num_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw2);
fprintf(fid,'\nfloat den_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw2);
fclose(fid);
winopen('IIR_LP_BW.txt');
fid=fopen('IIR_LP_CHEB Type1.txt','wt');
fprintf(fid,'\t\t----------Pass band range: 2500Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Rippled (3dB) -----\n\n\');
fprintf(fid,'\nfloat num_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb1);
fprintf(fid,'\nfloat den_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb1);
fprintf(fid,'\n\n\n\t\t----------Pass band range: 8000Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Rippled (3dB)-----\n\n');
fprintf(fid,'\nfloat num_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb2);
```

```
fprintf(fid,'\nfloat den_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb2);
fclose(fid);
winopen('IIR_LP_CHEB Type1.txt');

%%%%%%%%%%%%%%%%%
figure(1);
[h,w]=freqz(num_bw1,den_bw1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb1,den_cb1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response of Low pass IIR filters (Fc=2500Hz)');
figure(2);
[h,w]=freqz(num_bw2,den_bw2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
```

```
hold on
[h,w]=freqz(num_cb2,den_cb2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1 (Ripple: 3dB)');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response in the passband');
axis([0 12000 -20 20]);
```

IIR_CHEB_LP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=800Hz		Fc=8000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.044408	1455	0.005147	168	0.354544	11617
B1	0.088815	1455[B1/2]	0.010295	168[B1/2]	0.709088	11617[B1/2]
B2	0.044408	1455	0.005147	168	0.354544	11617
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-	-23140[A1/	-1.844881	-30225[A1	0.530009	8683[A1/2

	1.412427	2]		/2]		]
A2	0.663336	21735	0.873965	28637	0.473218	15506

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_BUTTERWORTH_LP FILTER CO-EFFICIENTS:

Co-Efficients	**Fc=2500Hz**		**Fc=800Hz**		**Fc=8000Hz**	
	Floating Point Values	**Fixed Point Values(Q15)**	**Floating Point Values**	**Fixed Point Values(Q15)**	**Floating Point Values**	**Fixed Point Values(Q15)**
B0	0.072231	2366	0.009526	312	0.465153	15241
B1	0.144462	2366[B1/2]	0.019052	312[B1/2]	0.930306	15241[B1/2]
B2	0.072231	2366	0.009526	312	0.465153	15241
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18179[A1/2]	-1.705552,	-27943[A1/2]	0.620204	10161[A1/2]
A2	0.398152	13046	0.743655	24367	0.240408	7877

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_CHEB_HP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.388513	12730	0.282850	9268	0.117279	3842
B1	-0.777027	-12730[B1/2]	-0.565700	-9268[B1/2]	-0.234557	-3842[B1/2]
B2	0.388513	12730	0.282850	9268	0.117279	3842
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.118450	-18324[A1/2]	-0.451410	-7395[A1/2]	0.754476	12360[A1/2]
A2	0.645091	21137	0.560534	18367	0.588691	19289

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_BUTTERWORTH_HP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)	Floating Point Values	Fixed Point Values(Q15)
B0	0.626845	20539	0.465153	15241	0.220195	7215
B1	-1.253691	-20539[B1/	-0.930306	-15241[B1	-0.440389	-7215[B1/2

		2]		/2]		]
B2	0.626845	20539	0.465153	15241	0.220195	7215
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18173[A1/2]	-0.620204	-10161[A1/2]	0.307566	5039[A1/2}
A2	0.398152	13046	0.240408	7877	0.188345	6171

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

'C' PROGRAM TO IMPLEMENT IIR FILTER

```
#include "filtercfg.h"
#include "dsk6713.h"
#include "dsk6713_aic23.h"
const signed int filter_Coeff[] =
{
    //12730,-12730,12730,2767,-18324,21137 /*HP 2500 */
    //312,312,312,32767,-27943,24367       /*LP 800 */
    //1455,1455,1455,32767,-23140,21735    /*LP 2500 */
    //9268,-9268,9268,32767,-7395,18367    /*HP 4000*/
      7215,-7215,7215,32767,5039,6171,              /*HP 7000*/
} ;
/* Codec configuration settings */
DSK6713_AIC23_Config config = { \
```

```
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume
*/ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel
volume */\
   0x00d8,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume
*/  \
   0x00d8,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone
volume */ \
   0x0011,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */     \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */    \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */
\
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */          \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};


/**  main() - Main code routine, initializes BSL and generates tone */
void main()
{
   DSK6713_AIC23_CodecHandle hCodec
   int l_input, r_input, l_output, r_output;
   /* Initialize the board support library, must be called first */
   DSK6713_init();
```

```
    /* Start the codec */
    hCodec = DSK6713_AIC23_openCodec(0, &config);
    DSK6713_AIC23_setFreq(hCodec, 3);
        while(1)
    {       /* Read a sample to the left channel */
            while (!DSK6713_AIC23_read(hCodec, &l_input));
            /* Read a sample to the right channel */
            while (!DSK6713_AIC23_read(hCodec, &r_input));
                l_output=IIR_FILTER(&filter_Coeff ,l_input);
                r_output=l_output;
            /* Send a sample to the left channel */
            while (!DSK6713_AIC23_write(hCodec, l_output));
            /* Send a sample to the right channel */
            while (!DSK6713_AIC23_write(hCodec, r_output));
    }

    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}
signed int IIR_FILTER(const signed int * h, signed int x1)
{
        static signed int x[6] = { 0, 0, 0, 0, 0, 0 };  /* x(n), x(n-1), x(n-2). Must be
static */
```

```
        static signed int y[6] = { 0, 0, 0, 0, 0, 0 };  /* y(n), y(n-1), y(n-2). Must be
static */

         int temp=0;


         temp = (short int)x1; /* Copy input to temp */

        x[0] = (signed int) temp; /* Copy input to x[stages][0] */

        temp =  ( (int)h[0] * x[0]) ;   /* B0 * x(n)   */

        temp += ( (int)h[1] * x[1]);    /* B1/2 * x(n-1) */

        temp += ( (int)h[1] * x[1]);    /* B1/2 * x(n-1) */

        temp += ( (int)h[2] * x[2]);    /* B2 * x(n-2) */

        temp -= ( (int)h[4] * y[1]);    /* A1/2 * y(n-1) */

        temp -= ( (int)h[4] * y[1]);    /* A1/2 * y(n-1) */

        temp -= ( (int)h[5] * y[2]);   /* A2 * y(n-2) */
   /* Divide temp by coefficients[A0] */

         temp >>= 15;

   if ( temp > 32767 )

     {

      temp = 32767;

     }

    else if ( temp < -32767)

     {

      temp = -32767;

     }
```

```
 y[0] =  temp ;
    /* Shuffle values along one place for next time */
    y[2] = y[1];   /* y(n-2) = y(n-1) */
    y[1] = y[0];   /* y(n-1) = y(n)   */

    x[2] = x[1];   /* x(n-2) = x(n-1) */
    x[1] = x[0];   /* x(n-1) = x(n)   */

    /* temp is used as input next time through */
  return (temp<<2);
}
```

RESULT:

Design of FIR filter (LP/HP) using windowing technique and verified using the DSP Processor

6. FAST FOURIER TRANSFORMS (FFT)

AIM: To verify the FFT using DSP processor

SOFTWARE REQUIRED: CODE COMPOSER STUDIO

HARDWARE REQUIRED:

- DSK 6713 DSP Trainer kit.
- USB Cable
- Power supply
- PC

THEORY:

The DFT Equation

$$X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$where \quad W_N^{nk} = e^{-j\frac{2\pi nk}{N}} \text{ [TWIDDLE FACTOR]}$$

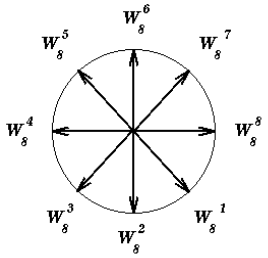
Twiddle Factor

In the Definition of the DFT, there is a factor called the *Twiddle Factor*

$$W_N^{nk} = e^{-j\frac{2\pi nk}{N}}$$

where N = number of samples.

If we take an 8 bit sample sequence we can represent the twiddle factor as a vector in the unit circle. e.g.



'C' Program

Main.c (fft 256.c):

```
#include <math.h>
#define PTS 64                          //# of points for FFT
#define PI 3.14159265358979

typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n);            //FFT prototype
float iobuffer[PTS];                    //as input and output buffer
float x1[PTS];                          //intermediate buffer
short i;                        //general purpose index variable
short buffercount = 0;          //number of new samples in iobuffer
short flag = 0;                 //set to 1 by ISR when iobuffer full
COMPLEX w[PTS];                         //twiddle constants stored in w
COMPLEX samples[PTS];                   //primary working buffer

main()
{
  for (i = 0 ; i<PTS ; i++)    // set up twiddle constants in w
```

```
  {
  w[i].real = cos(2*PI*i/(PTS*2.0)); //Re component of twiddle constants
  w[i].imag =-sin(2*PI*i/(PTS*2.0)); //Im component of twiddle constants
  }
      for (i = 0 ; i < PTS ; i++)    //swap buffers
   {
    iobuffer[i] = sin(2*PI*10*i/64.0);/*10- > freq,
                                                    64 -> sampling freq*/
    samples[i].real=0.0;
    samples[i].imag=0.0;
   }

for (i = 0 ; i < PTS ; i++)    //swap buffers
   {
    samples[i].real=iobuffer[i]; //buffer with new data
   }
  for (i = 0 ; i < PTS ; i++)
    samples[i].imag = 0.0;    //imag components = 0
FFT(samples,PTS);             //call function FFT.c

   for (i = 0 ; i < PTS ; i++)    //compute magnitude
   {
    x1[i] = sqrt(samples[i].real*samples[i].real
```

```
            + samples[i].imag*samples[i].imag);
   }

}                                   //end of main
```

fft.c:

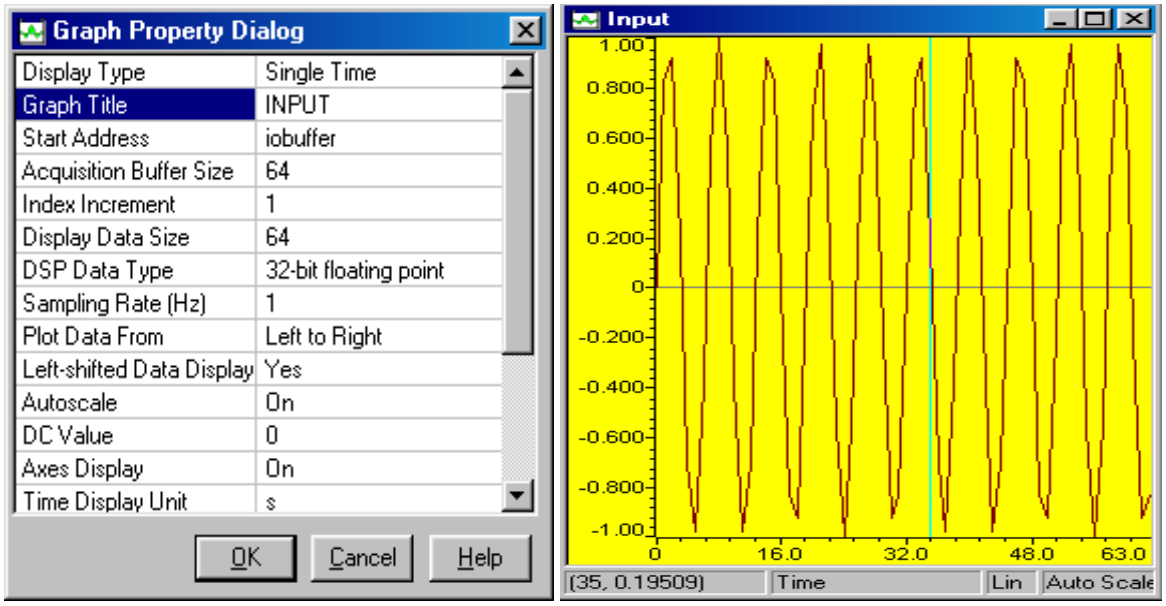
```
#define PTS 64                      //# of points for FFT
typedef struct {float real,imag;} COMPLEX;
extern COMPLEX w[PTS];              //twiddle constants stored in w

void FFT(COMPLEX *Y, int N)   //input sample array, # of points
{
 COMPLEX temp1,temp2;        //temporary storage variables
 int i,j,k;                  //loop counter variables
 int upper_leg, lower_leg;         //index of upper/lower butterfly leg
 int leg_diff;               //difference between upper/lower leg
 int num_stages = 0;         //number of FFT stages (iterations)
 int index, step;            //index/step through twiddle constant
 i = 1;                  //log(base2) of N points= # of stages
 do
 {
  num_stages +=1;
  i = i*2;
```

```
 }while (i!=N);
leg_diff = N/2;                 //difference between upper&lower legs
step = (PTS*2)/N;           //step between values in twiddle.h
for (i = 0;i < num_stages; i++)  //for N-point FFT
 {
  index = 0;
  for (j = 0; j < leg_diff; j++)
   {
 for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
  {
     lower_leg = upper_leg+leg_diff;
         temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
     temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
     temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
     temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
     (Y[lower_leg]).real = temp2.real*(w[index]).real
           -temp2.imag*(w[index]).imag;
     (Y[lower_leg]).imag = temp2.real*(w[index]).imag
           +temp2.imag*(w[index]).real;
     (Y[upper_leg]).real = temp1.real;
     (Y[upper_leg]).imag = temp1.imag;
  }
  index += step;
```

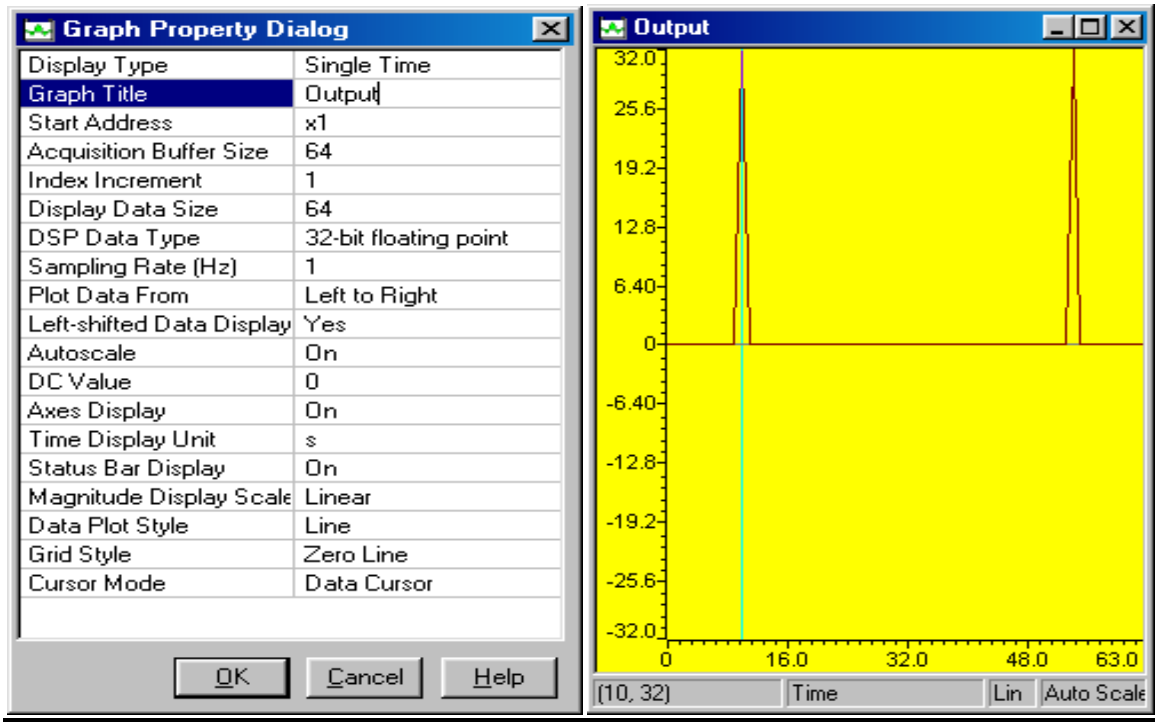
```
}
 leg_diff = leg_diff/2;
 step *= 2;
  }
   j = 0;
   for (i = 1; i < (N-1); i++)     //bit reversal for resequencing data
  {
 k = N/2;
 while (k <= j)
  {
   j = j - k;
   k = k/2;
  }
 j = j + k;
    if (i<j)
  {
   temp1.real = (Y[j]).real;
   temp1.imag = (Y[j]).imag;
   (Y[j]).real = (Y[i]).real;
   (Y[j]).imag = (Y[i]).imag;
   (Y[i]).real = temp1.real;
   (Y[i]).imag = temp1.imag;
  }
```

```
    }
  return;
}
```

INPUT:



OUTPUT:



RESULT:

FET is verified using DSP Processor.

7. IMPULSE RESPONSE OF FIRST ORDER AND SECOND ORDER SYSTEMS

AIM:- To find Impulse response of a first order and second order system.

EQUIPMENTS:-

- Operating System - Windows XP
- Software - CC STUDIO
- DSK 6713 DSP Trainer kit.
- USB Cable
- Power supply

THOERY:

First Order Systems - Impulse Response

You might be tempted to think that you can't get a transfer function from an impulse response. That's not true. Imagine that this is the response to a unit impulse.

Since you know that this is the response to a unit impulse, you can get the transfer function from this response. Consider a first order system transfer function.

$G(s) = G_{dc}/(st + 1)$

The response of this system to a unit impulse is just the inverse Laplace transform of the transfer function. That means the impulse response is given by this expression.

$g(t) = (G_{dc}/\mathbf{t})e^{-t/t}$

If the input is a unit impulse, then we can measure the starting value of the transient.

Starting Value of Transient = $G_{dc}/\mathbf{t}$

If the impulse is not a unit impulse, we can still get the same information as long as we know the size of the impulse. Imagine that you have an impulse, A**d(t)**. Then, the response would be $A(G_{dc}/\mathbf{t})e^{-t/t}$. Then, we would have:

Starting Value of Transient = $AG_{dc}/\mathbf{t}$

We can always get the ratio of the DC gain to the time constant. Then, you can use any technique you want to get the time constant. That will give you both the DC gain and the time constant, and you'll have the transfer function.

% For first order difference equation.%

PROGRAM:

```
 #include<stdio.h>
 #define Order 1
 #define Len 5
 float h[Len] = {0.0,0.0,0.0,0.0,0.0},
sum;
 void main()
{
int j,
k;
float a[Order+1] = {0.1311, 0.2622};
float b[Order+1] = {1, -0.7478};
for(j=0; j<Len; j++)
 {
sum = 0.0;
for(k=1; k<=Order; k++)
 {
if((j-k)>=0) sum = sum+(b[k]*h[j-k]);
}
if(j<=Order) h[j] = a[j]-sum;
else h[j] = -sum;
printf("%f", j, h[j]);
```

```
 }

}
```

OUTPUT: 0.131100 0.360237 0.269385 0.201446 0.150641.

%Find out the impulse response of second order difference equation.%

PROGRAM:

```
 #include<stdio.h>
 #define Order 2
 #define Len 5
 float h[Len] = {0.0,0.0,0.0,0.0,0.0},sum;
void main()
{
int j, k;
float a[Order+1] = {0.1311, 0.2622, 0.1311};
 float b[Order+1] = {1, -0.7478, 0.2722};
for(j=0; j<Len; j++)
{
sum = 0.0;
 for(k=1; k<=Order; k++)
{

if ((j-k) >= 0) sum = sum+(b[k]*h[j-k]);

 }
if (j <= Order) h[j] = a[j]-sum;
else
 h[j] = -sum; printf (" %f “,h[j]);

 }

}
```

OUTPUT: 0.131100 0.360237 0.364799 0.174741 0.031373

RESULT:- Impulse response of a first order and second order system is performed using CC Studio