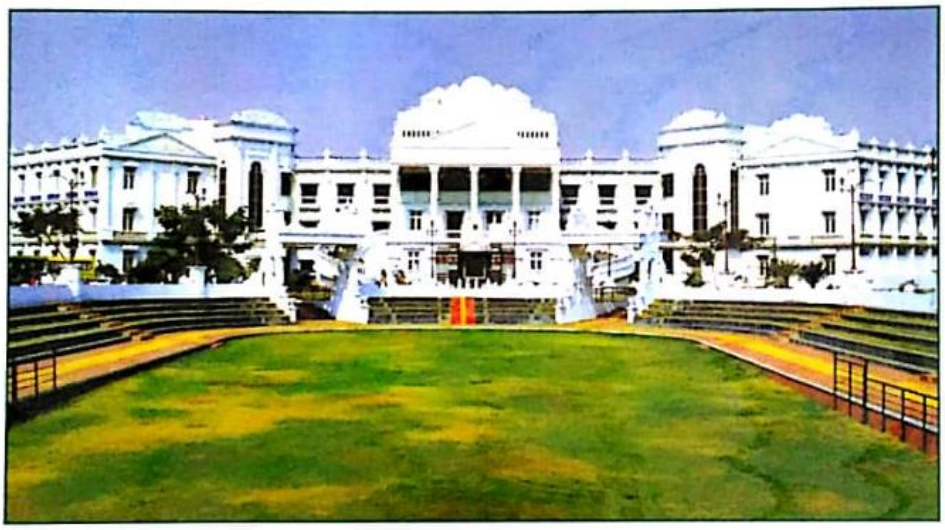
AVN
Institute of Engineering and Technology
Approved by AICTE Affiliated to JNTUH
College Code : AVNI - Category - I Institution
Cor. Office : Sagar Road, Karmanghat 'X' Road, Karmanghat, Hyderabad. Ph : 2409 2929
Campus : Koheda Road, Ibrahimpatnam (M), Ranga Reddy Dist.
Ph : 08415-201345
www.avniet.ac.in

JNTU CODE: (EC604PC**)**

Program: UG



DEPARTMENTOFELECTRONICS AND COMMUNICATIONENGINEERING

LABORATORY MANUAL

DIGITAL SIGNAL PROCESSING LAB

III Year B.Tech. ECE – II Sem

LAB INCHARGES **HOD**

Note: Minimum of 12 experiments has to be conducted (six from each part)

Vision and Mission of ECE Department:

Vision:

To impart quality technical education in Electronics and Communication Engineering emphasizing analysis, design/synthesis and evaluation of hardware/embedded software usingvarious Electronic Design Automation (EDA) tools with accent on creativity, innovation and research thereby producing competent engineers who can meet global challenges with societal commitment.

Mission:

i. To impart quality education in fundamentals of basic sciences, mathematics, electronics and communication engineering through innovative teaching-learning processes.
ii. To facilitate Graduates define, design, and solve engineering problems in the field of Electronics and Communication Engineering using various Electronic Design Automation (EDA) tools.
iii. To encourage research culture among faculty and students thereby facilitating them to be creative and innovative through constant interaction with R & D organizations and Industry.
iv. To inculcate teamwork, imbibe leadership qualities, professional ethics and social responsibilities in students and faculty.

Program Educational Objectives of B. Tech (ECE) Program :

I. To prepare students with excellent comprehension of basic sciences, mathematics and engineering subjects facilitating them to gain employment or pursue postgraduate studies with an appreciation for lifelong learning.
II. To train students with problem solving capabilities such as analysis and design with adequate practical skills wherein they demonstrate creativity and innovation that would enable them to develop state of the art equipment and technologies of multidisciplinary nature for societal development.
III. To inculcate positive attitude, professional ethics, effective communication and interpersonal skills which would facilitate them to succeed in the chosen profession exhibiting creativity and innovation through research and development both as team member and as well as leader.

Program Outcomes of B.Tech ECE Program:

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

DIGITAL SIGNAL PROCESSING LAB

B.Tech. III Year II Sem. **L T P C**

` **0 0 3 2**

Note:

- The Programs shall be implemented in Software (Using MATLAB / Lab View / C Programming/ Equivalent) and Hardware (Using TI / Analog Devices / Motorola / Equivalent DSP processors).
- Minimum of 12 experiments to be conducted.

List of Experiments:

1. Generation of Sinusoidal Waveform / Signal based on Recursive Difference Equations
2. Histogram of White Gaussian Noise and Uniformly Distributed Noise.
3. To find DFT / IDFT of given DT Signal
4. To find Frequency Response of a given System given in Transfer Function/ Differential equation form.
5. Obtain Fourier series coefficients by formula and using FET and compare for half sine wave.
6. Implementation of FFT of given Sequence
7. Determination of Power Spectrum of a given Signal(s).
8. Implementation of LP FIR Filter for a given Sequence/Signal.
9. Implementation of HP IIR Filter for a given Sequence/Signal
10. Generation of Narrow Band Signal through Filtering
11. Generation of DTMF Signals
12. Implementation of Decimation Process
13. Implementation of Interpolation Process
14. Implementation of I/D Sampling Rate Converters
15. Impulse Response of First order and Second Order Systems.

* Additional Experiments

16. To find the Discrete Cosine Transform of image.
17. Verification of Linear and Circular Convolution of signals/ Sequences.
18. Develop an algorithm to implement A) Histogram equalization, B) Histogram specification. Of image
19. Read a 256x256 image. Do the following operations. A) Filtering using simple averaging masks, B) Median filtering C) Gaussian filtering D) Triangular filtering (Pyramidal filter, cone filter) E) Compare your results

EXP.NO: 1

GENERATE SINUSOIDAL WAVEFORM BASED ON RECURSIVE DIFFERENCE EQUATIONS

Aim: Generation of Sinusoidal waveform / signal based on recursive difference equations

EQUIPMENT: PC with windows (95/98/XP/NT/2000).

MATLAB Software

THEORY: For the given difference equation, a sinusoidal signal/sequence is applied as the input. Using the given initial conditions, the sinusoidal response of the given discrete system is to be computed.

1. The difference equation is y(n)=x(n)+y(n-1), which is a first order system, with the initial condition y(-1)=4.

Program:

```
clc;
clear all;
close all;
a=input('enter the amplitude of the signal');
f=input('enter the frequency of the signal');
N=input('enter the number of cycles');
l=input('enter the intial condition y(-1)');
T=1/f;
t=0:1/100*T:N*T;
x=a*sin(2*pi*f*t);
y=zeros(1,length(t));
for i=1:1:length(x)
    y(i)=x(i)+l;
    l=y(i);
end;
subplot(2,1,1);
plot(t,x);
xlabel('timet');
ylabel('amplitude');
title('sine signal ');
subplot(2,1,2);
plot(t,y);
xlabel('timet');
ylabel('amplitude');
title(' sine signal generated by recursive equation');
```

OUTPUT

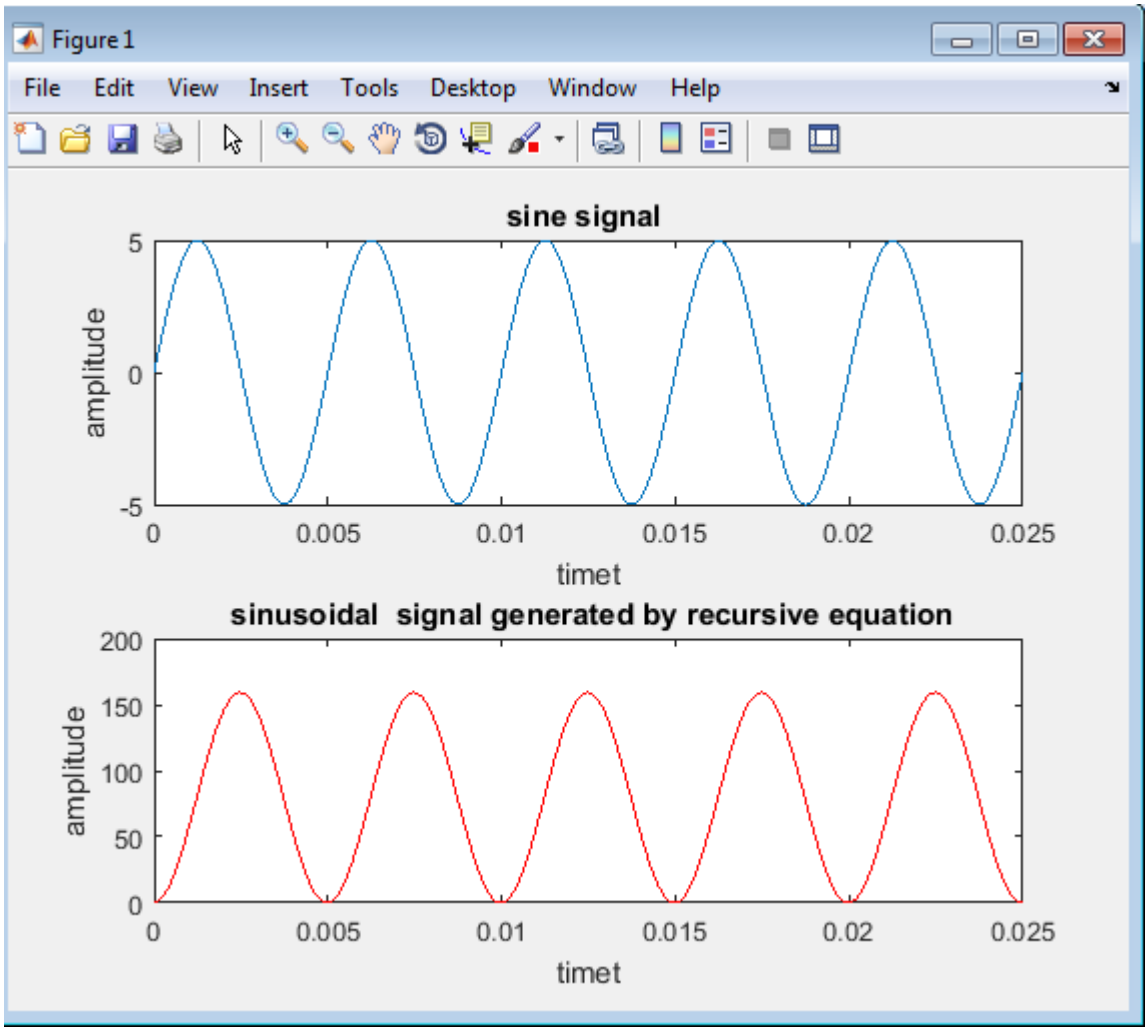
enter the amplitude of the signal5

enter the frequency of the signal200

enter the number of cycles5

enter the intial condition y(-1)0

RESULT



EXP.NO: 2

HISTOGRAM OF WHITE GAUSSIAN NOISE AND UNIFORMLY DISTRIBUTED NOISE

Aim: To generate histogram of white Gaussian noise and uniformly distributed noise using matlab software

EQUIPMENT: PC with windows (95/98/XP/NT/2000).

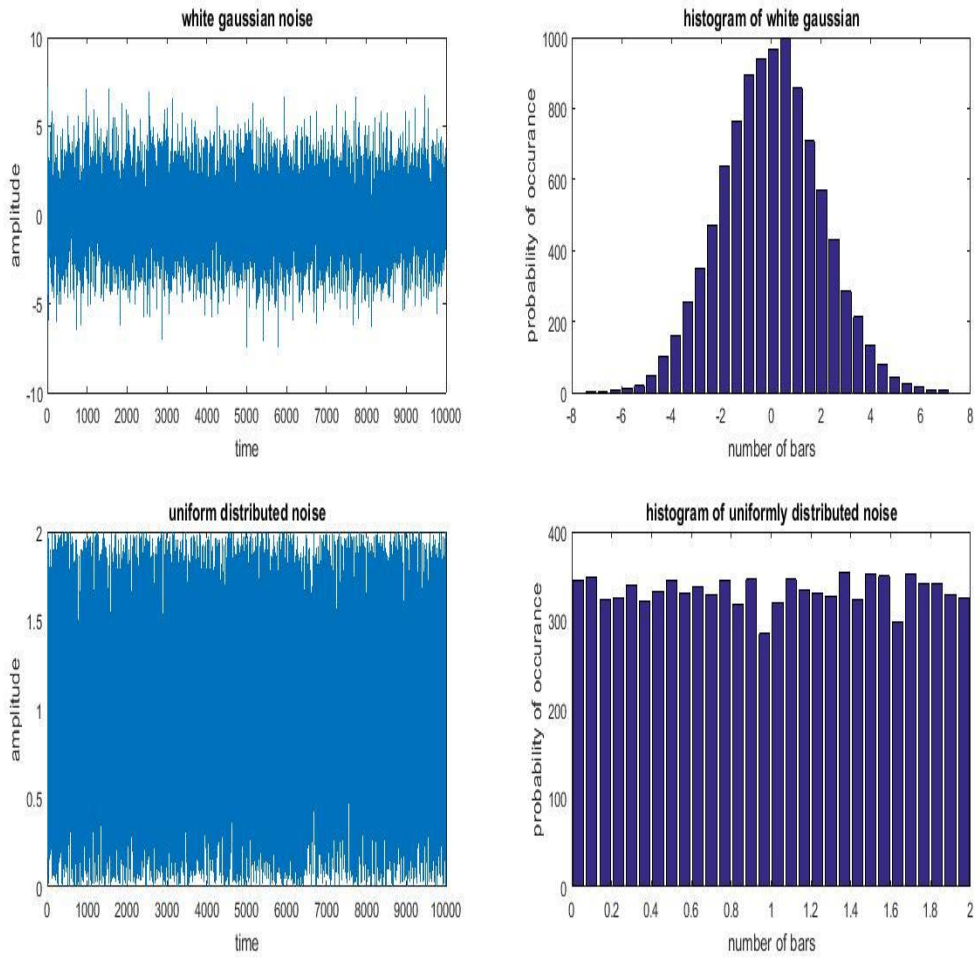
MATLAB Software

PROGRAM:

```
clc;
clear all;
close all;
l=10000;
mu=input('enter the mean value');
sigma=input('enter the standard deviation');
n=input('enter the number of bins');
x=sigma*randn(1,l)+mu;
y=sigma*rand(1,l)+mu;
[f1,x1]=hist(x,n);
[f2,y2]=hist(y,n);
subplot(2,2,1);
plot(x);
xlabel('time');
ylabel('amplitude');
title('white gaussian noise');
subplot(2,2,2);
bar(x1,f1);
xlabel('number of bars');
ylabel('probability of occurance');
title('histogram of white gaussian');
subplot(2,2,3);
plot(y);
xlabel('time');
ylabel('amplitude');
title('uniform distributed noise');
subplot(2,2,4);
bar(y2,f2);
xlabel('number of bars');
ylabel('probability of occurance');
title('histogram of uniformly distributed noise');
```

INPUTS:
Enter the mean value0
Enter the standard deviation2
Enter the number of bins30

OUTPUT



EXP.NO: 3

DFT/IDFT OF GIVEN DISCRETE TIME SIGNAL

AIM: To find DFT/IDFT of a given Discrete Time signal

EQUIPMENT: PC with windows (95/98/XP/NT/2000). MATLAB Software

THEORY: The discrete Fourier Transform of sequence is Periodic and we are interested in frequency range 0 to 2π there are infinitely many ω in this range. If we use a digital computer to compute N equally spaced points over the interval $0 \leq \omega < 2\pi$ then the N points should be located at $\omega_k = (2\pi/N)k$; k=0,1,2……,N-1;

These N equally space frequency samples of the DTFT are known as DFT denoted by X(k) is

$X(k) = X(e^{j\omega})|_{\omega k = (2\pi/N)k}$; $0 \leq k \leq N-1$.

The formulas for DFT and IDFT are

DFT $X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}$ $0 \leq k \leq N-1$.

IDFT $x(n) = (1/N) \sum_{k=0}^{N-1} X(k) e^{j2\pi kn/N}$ $0 \leq n \leq N-1$.

For notation purpose $X(k) =$ DFT $[x(n)]$

$x(n) =$ IDFT $[X(k)]$

PROGRAM:

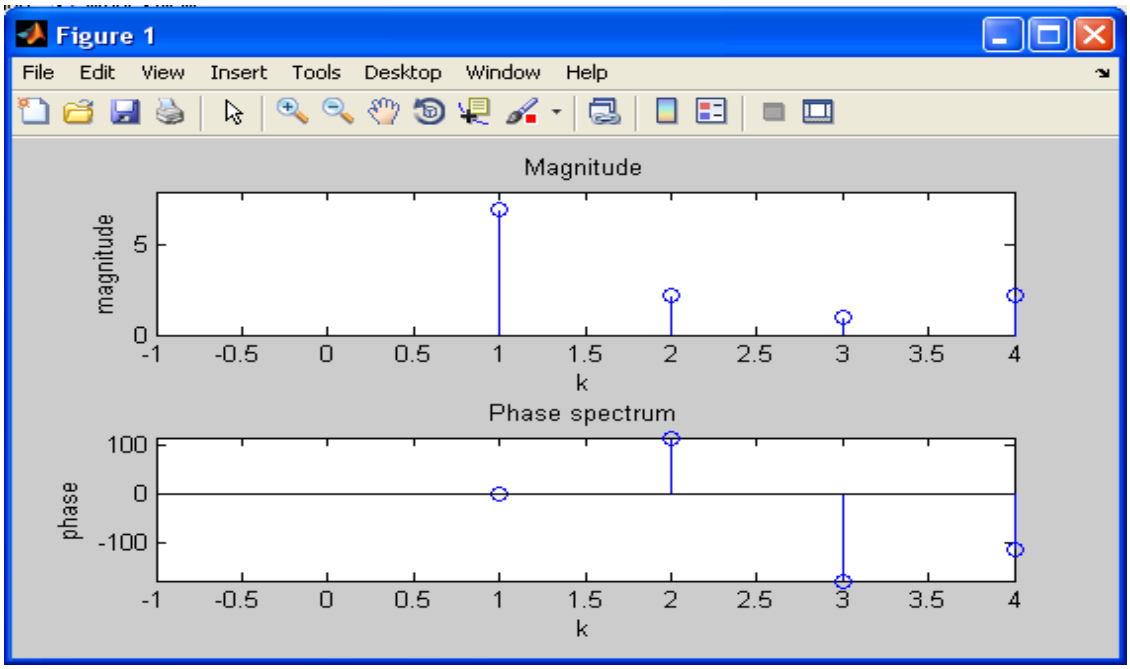
%DFT function

```
clc;
clear all;
close all;
x=input('enter the values of input sequence');
N=input('enter the npoint dft values');
l=length(x);
if(N<l)
    error('N should be >=l');
end;
x1=[x zeros(1,(N-l))]
for k=0:1:N-1;
    for n=0:1:N-1;
        p=exp(-j*2*pi*n*k/N)
        x2(n+1,k+1)=p;
    end;
end;
y=(x1*x2);
psd=abs(y.*y)/N;
disp('the input sequence is:');
```

```
disp(x1);
disp('the thiddle factor matrix is:');
disp(x2);
disp('the dft of x(n) is');
disp(y);
magnitude=abs(y);
phase=angle(y);
subplot(2,1,1);
stem(magnitude);
xlabel('frequency');
ylabel('magnitude');
title('magnitude plot');
subplot(2,1,2);
stem(phase);
xlabel('frequency');
ylabel('phase');
title('phase plot');
```

OUTPUT: Enter the sequence:[1 1 2 3]
Enter the value of N:4
The DFT of the given sequence is
7.0000 -1.0000 + 2.0000i -1.0000 - 0.0000i -1.0000 - 2.0000i
The corresponding magnitude vector is
7.0000 2.2361 1.0000 2.2361
The corresponding phase vector is
0 116.5651 -180.0000 -116.5651

RESULT:



PROGRAM:

%IDFT function

```
clc;
clear all;
close all;
x=input('enter the values of input sequence');
N=input('enter the npoint dft values');
l=length(x);
if(N<l)
    error('N should be >=l');
end;
x1=[x zeros(1,(N-l))]
for k=0:1:N-1;
    for n=0:1:N-1;
        p=exp(j*2*pi*n*k/N)
        x2(n+1,k+1)=p;
    end;
end;
y=(1/N)*(x1*x2);
disp('the input sequence is:');
disp(x1);
disp('the thiddle factor matrix is:');
disp(x2);
disp('the idft of x(n) is');
disp(y);
magnitude=abs(y);
phase=angle(y);
subplot(2,1,1);
stem(magnitude);
xlabel('frequency');
ylabel('magnitude');
title('magnitude plot');
subplot(2,1,2);
stem(phase);
xlabel('frequency');
ylabel('phase');
title('phase plot');
```

OUTPUT:

Enter the sequence:[7.0000 -1.0000 + 2.0000i -1.0000 - 0.0000i -1.0000 - 2.0000i]
Enter the value of N:4
The DFT of the given sequence is
1.0000 1.0000 + 0.0000i 2.0000 - 0.0000i 3.0000 - 0.0000i
The corresponding magnitude vector is
1 1 2 3
The corresponding phase vector is
1.0e-014 * 0 0.3836 -0.1754 -0.8607

RESULT:

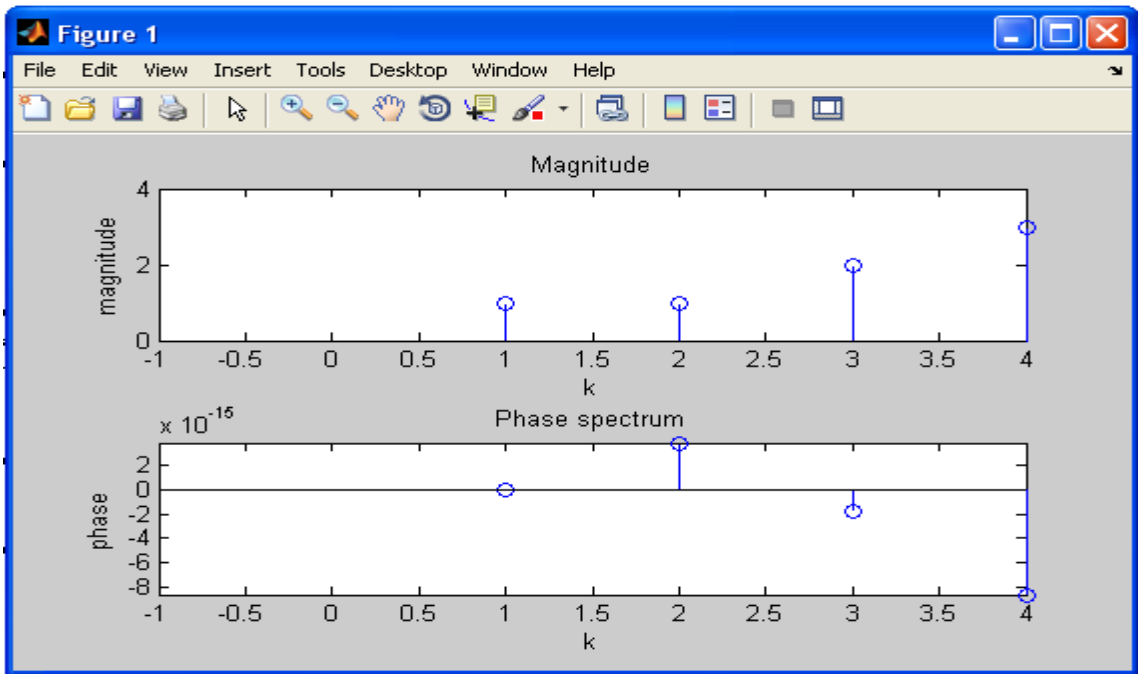


Figure 1
File Edit View Insert Tools Desktop Window Help
Magnitude
magnitude
4
2
0
-1 -0.5 0 0.5 1 1.5 2 2.5 3 3.5 4
k
x 10^-15
Phase spectrum
phase
2
0
-2
-4
-6
-8
-1 -0.5 0 0.5 1 1.5 2 2.5 3 3.5 4
k

EXP.NO: 4

FREQUENCY RESPONSE OF A GIVEN SYSTEM

AIM: To find frequency response of a given system given in (Transfer function/Difference equation form)

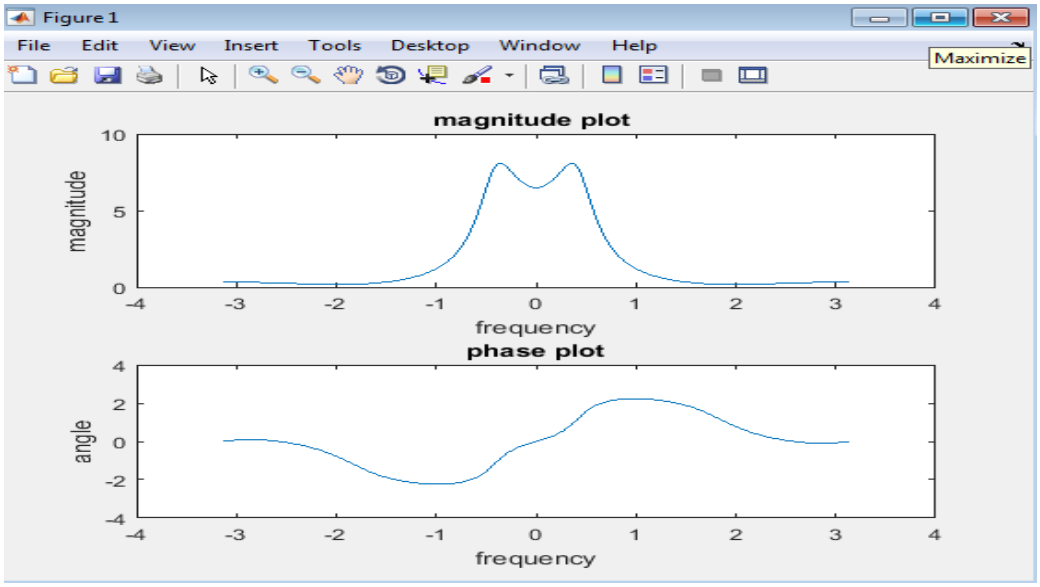
EQUIPMENT: PC with windows (95/98/XP/NT/2000). MATLAB Software

PROGRAM:

```
clc;
clear all;
close all;
w=-pi:0.01:pi;
num=input('enter the numerator coefficient');
den=input('enter the denominator coefficient');
h=freqz(num,den,w);
magnitude=abs(h);
phase=angle(h);
subplot(2,1,1);
plot(w,magnitude);
xlabel('frequency');
ylabel('magnitude');
title('magnitude plot ');
subplot(2,1,2);
plot(w,phase);
xlabel('frequency');
ylabel('angle');
title('phase plot');
```

OUTPUT:

enter the numerator coefficient[1 3 4 5]
enter the denominator coefficient[3 -2 -6 7]
RESULT:



EXP.NO: 6

IMPLEMENTATION OF FFT FOR A GIVEN SEQUENCE

AIM: To implement FFT of a given sequence.

EQUIPMENT: PC with windows (95/98/XP/NT/2000),MATLAB Software

THEORY:DFT of a sequence

$$X[K] = \sum_{K=0}^{N-1} x[n] e^{\frac{-j2\Pi Kn}{N}}$$

Where N= Length of sequence.

K= Frequency Coefficient.

n = Samples in time domain.

FFT : -Fast Fourier transformer .

There are Two methods.

1.Decimation in time **(DIT FFT).**

2. Decimation in Frequency **(DIF FFT).**

Why we need FFT ?

The no of multiplications in **DFT = N^2.**

The no of Additions in **DFT = N(N-1).**

For **FFT.**

The no of multiplication = **N/2 $\log_2$N.**

The no of additions = **N $\log_2$ N.**

Program:

```
clc;
clear all;
close all;
x=input('enter the values of input sequence ');
N=input('enter the npoint fft values');
l=length(x);
if(N<l)
error('N should be >=l');
end;
x1=[x,zeros(1,(N-l))]
y=fft(x1,N)
magnitude=abs(y);
phase=angle(y);
disp('the input sequence x(n) is');
disp(x1);
disp('the fft of x1 is y');
disp(y);
subplot(2,1,1);
stem(magnitude);grid;
xlabel('frequency');
ylabel('magnitude');
title('magnitude plot');
subplot(2,1,2);
stem(phase);grid;
xlabel('frequency');
ylabel('phase');
title('phase plot');
```

FFT INPUTS

enter the values of input sequence [1 2 3 4 5 6 7 8]
enter the npoint fft values8
the input sequence x(n) is
1 2 3 4 5 6 7 8

the fft of x1 is y
Columns 1 through 6

36.0000 + 0.0000i -4.0000 + 9.6569i -4.0000 + 4.0000i -4.0000 + 1.6569i -4.0000 + 0.0000i -4.0000 - 1.6569i

Columns 7 through 8

-4.0000 - 4.0000i -4.0000 - 9.6569i

RESULT:

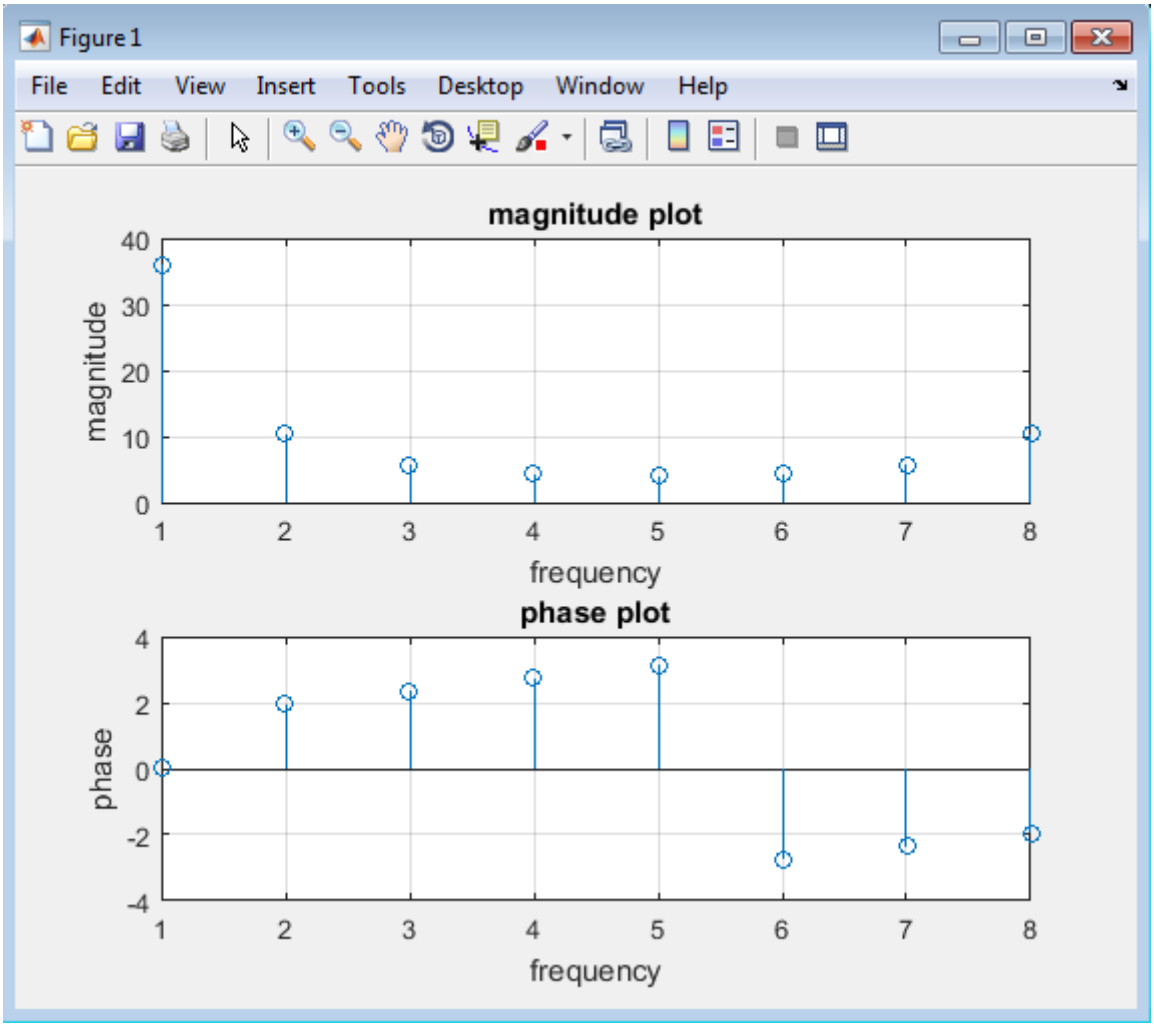


Figure 1
File Edit View Insert Tools Desktop Window Help
magnitude plot
magnitude
frequency
phase plot
phase
frequency

EXP.NO: 7

DETERMINATION OF POWER SPECTRUM OF A SIGNAL

AIM: To determine the power density spectrum of a given signal.

EQUIPMENT: PC with windows (95/98/XP/NT/2000), MATLAB Software

THEORY:

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.

$$S(\omega) = FT(R(\tau)) = \lim_{N\to\infty} \sum_{\tau=-N+1}^{N-1} \hat{R}(\tau)e^{-j\omega\tau}$$

$$R(\tau) = FT^{-1}(S(\omega)) = \frac{1}{2\pi}\int S(\omega)e^{j\omega\tau}d\omega$$

PROGRAM:

```
clc;
close all;
clear all;
x=input('enter the sequence');
```

```
N=length(x);
n=0:1:N-1;
y=xcorr(x,x);
subplot(3,1,1);
stem(n,x);
xlabel(' n----->');ylabel('Amplitude--->');
title('input seq');
subplot(3,1,2);
N=length(y);
n=0:1:N-1;
stem(n,y);
xlabel('n---->');ylabel('Amplitude----.');
title('autocorr seq for input');
disp('autocorr seq for input');
disp(y)
P=fft(y,N);
subplot(3,1,3);
stem(n,P);
xlabel('K----->');ylabel('Amplitude--->');
title('psd of input');
disp('the psd function:');
disp(P)
```

OUTPUT:

```
enter the values of input sequence [1 2 3 4 5 6 7 8]
enter the npoint fft values8
the input sequence x(n) is
    1     2     3     4     5     6     7     8

the psd of x1 is
  162.0000   13.6569    4.0000    2.3431    2.0000    2.3431    4.0000   13.6569
```

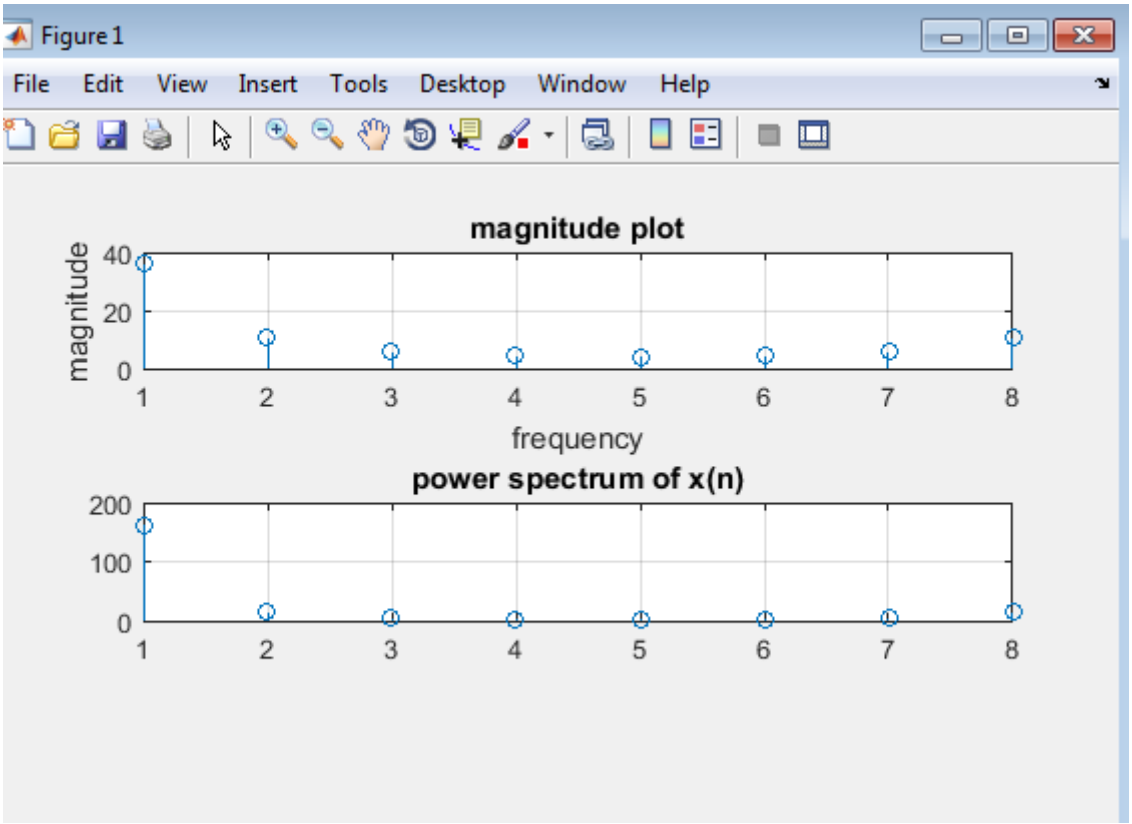


Figure 1
File Edit View Insert Tools Desktop Window Help
magnitude plot
magnitude
40
20
0
1 2 3 4 5 6 7 8
frequency
power spectrum of x(n)
200
100
0
1 2 3 4 5 6 7 8

EXP.NO: 8

IMPLEMENTATION OF LOW PASS FIR FILTER FOR A GIVEN SEQUENCE/SIGNAL

AIM: To implement LP FIR for a given sequence.

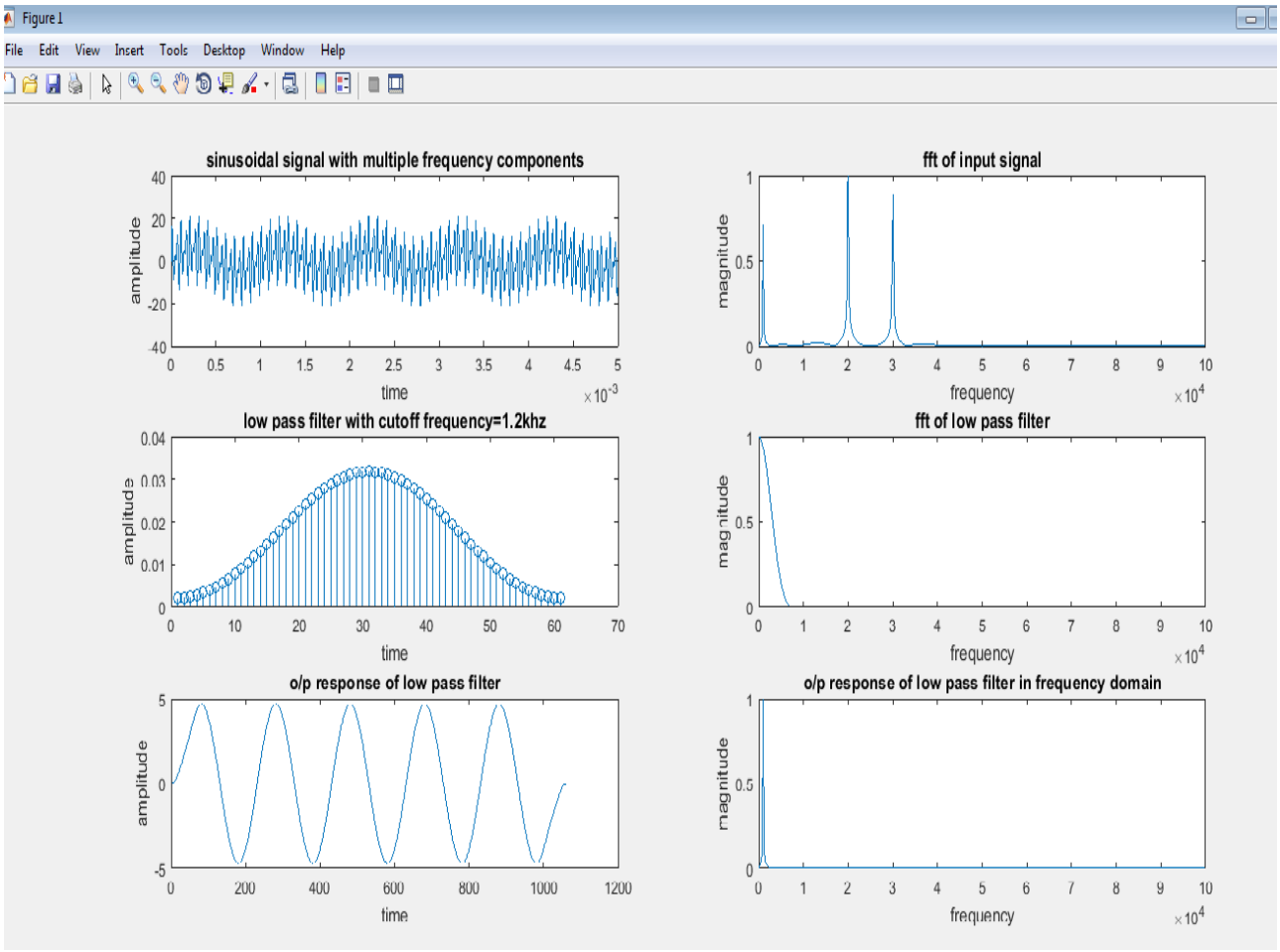
EQUIPMENT:PC with windows (95/98/XP/NT/2000). MATLAB Software

PROGRAM:

```
clc;
clear all;
close all;
fs=200e3;
ts=1/fs;
t=0:ts:5e-3-ts;
f1=1e3;
f2=20e3;
f3=30e3;
y=5*sin(2*pi*f1*t)+9*sin(2*pi*f2*t)+8*sin(2*pi*f3*t);
nfft=length(y);
nfft2=2.^nextpow2(nfft);
fy=fft(y,nfft2);
fy1=fy(1:nfft2/2);
xfft=fs.*(0:nfft2/2-1)/nfft2;
cutoff=(1.2e3)/(fs/2);
order=60;
h=fir1(order,cutoff);
fh=fft(h,nfft2);
fh1=fh(1:nfft2/2);
fc=fy1.*fh1;
convolution=conv(y,h);
subplot(3,2,1);
plot(t,y);
xlabel('time');
ylabel('amplitude')
title('sinusoidal signal with multiple frequency components');
subplot(3,2,2);
plot(xfft,abs(fy1/max(fy1)));
xlabel('frequency');
ylabel('magnitude')
title('fft of input signal');
subplot(3,2,3);
stem(h);
xlabel('time');
ylabel('amplitude')
title('low pass filter with cutoff frequency=1.2khz');
subplot(3,2,4);
plot(xfft,abs(fh1/max(fh1)));
xlabel('frequency');
ylabel('magnitude')
title('fft of low pass filter');
subplot(3,2,5);
plot(convolution);
xlabel('time');
ylabel('amplitude')
title('o/p response of low pass filter');
subplot(3,2,6);
plot(xfft,abs(fc/max(fc)));
```

```
xlabel('frequency');
ylabel('magnitude')
title('o/p response of low pass filter in frequency domain');
```

RESULT:



EXP.NO: 9

IMPLEMENTATION OF HIGH PASS FIR FILTER

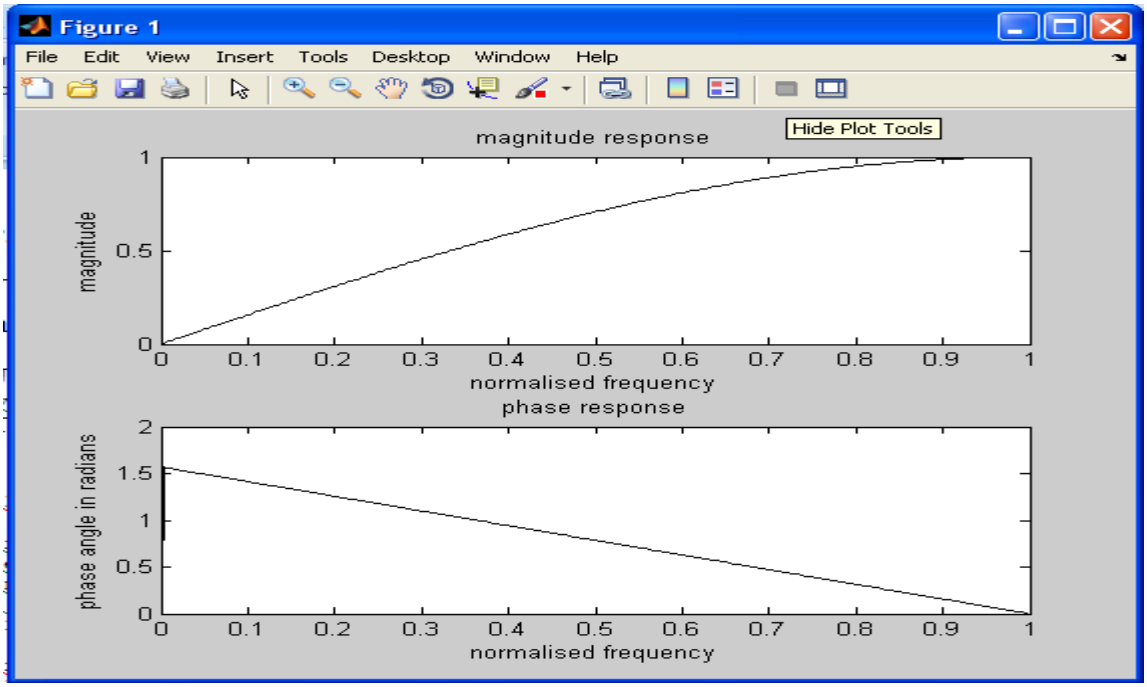
AIM: To implement HP FIR for a given sequence.

EQUIPMENT: PC with windows (95/98/XP/NT/2000). MATLAB Software

PROGRAM:

```
%Implimentation of HP Fir filter for given sequence
clc
clear all
close all
b1=input('enter the sequence b1');
a1=[1];
w=0:.01:pi;
h1=freqz(b1,a1,w);
subplot(2,1,1);
plot(w/pi,abs(h1),'k');
xlabel('normalised frequency'), ylabel('magnitude');
title('magnitude response')
subplot(2,1,2);
plot(w/pi,angle(h1),'k');
xlabel('normalised frequency'), ylabel('phase angle in radians');
title('phase response')
```

RESULT: enter the sequence b1[.5 -.5]



EXP.NO: 10

GENERATION OF NARROW BAND SIGNAL THROUGH FILTERING

AIM: To implement LP IIR for a given sequence.

EQUIPMENT: PC with windows (95/98/XP/NT/2000). MATLAB Software

THEORY: An Infinite impulse response (IIR) filter possesses an output response to an impulse which is of an infinite duration. The impulse response is "infinite" since there is feedback in the filter, that is if you put in an impulse ,then its output must produced for infinite duration of time. 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:

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

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

$$y(n)=\sum_{k=0}^{M} b_k x(n-k) \; + \; \sum_{k=1}^{N} a_k y(n-k) \qquad (2)$$

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

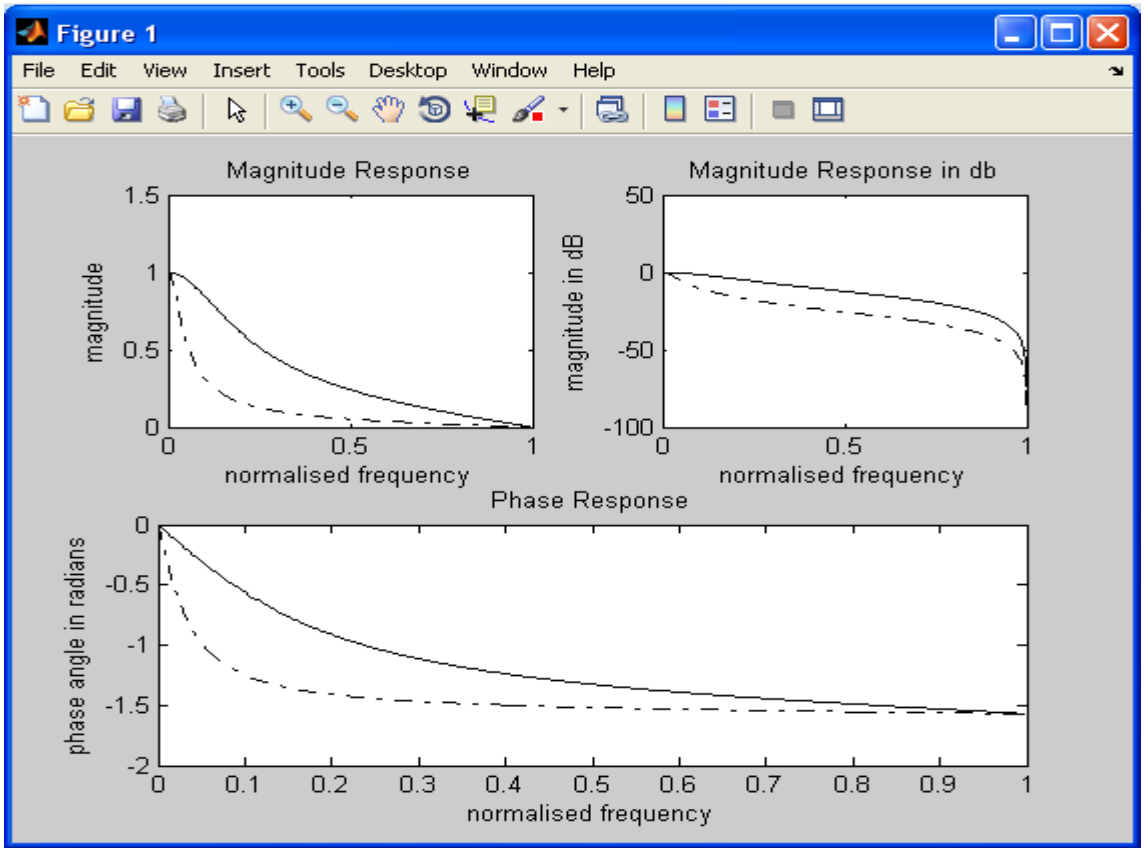
PROGRAM:

```
clc;
clear all
close all
b1=input('enter the sequence');
%b1=[.2 .2];
a1=[1 -.6];
w=0:.01:pi;
h1=freqz(b1,a1,w);
subplot(2,2,1);
plot(w/pi,abs(h1),'k');
hold on
subplot(2,2,2);
plot(w/pi,20*log10(abs(h1)),'k');
hold on
subplot(2,1,2);
plot(w/pi,angle(h1),'k');
hold on
b2=[.05 .05];
a2=[1 -0.9];
w=0:.01:pi;
```

```
h2=freqz(b2,a2,w);
subplot(2,2,1);
plot(w/pi,abs(h2),'k-.');
xlabel('normalised frequency'), ylabel('magnitude');
title('Magnitude Response')
subplot(2,2,2);
plot(w/pi,20*log10(abs(h2)),'k-.');
xlabel('normalised frequency'), ylabel('magnitude in dB');
title('Magnitude Response in db')
subplot(2,1,2);
plot(w/pi,angle(h2),'k-.');
xlabel('normalised frequency'), ylabel('phase angle in radians');
title('Phase Response')
```

OUTPUT: enter the sequence [.2 .2]

RESULT:



EXP.NO: 11

GENERATION OF DTMF SIGNALS

AIM: To generate DTMF signals

EQUIPMENT: PC with windows (95/98/XP/NT/2000). MATLAB Software

THEORY: Analog DTMF telephone signaling is based on encoding standard telephone Keypad digits and symbols in two audible sinusoidal signals of frequencies FL and FH.

Thus the scheme gets its name as dual tone multi frequency (DTMF).

Hz	12 09	1336	1477	1633
697	**1**	**2**	**3**	**A**
770	**4**	**5**	**6**	**B**
852	**7**	**8**	**9**	**C**
941	*****	**0**	#	**D**

Each digit or symbol represented in figure 1 has 2 distinct high and low frequency components. Thus each high-low frequency pair uniquely identifies the corresponding telephone keypad digit or symbol. Each key pressed can be represented as a discrete time signal of form

DTMF digit = row tone +columntone

$d_t[n] = \sin[\omega_L n] + \sin[\omega_L n]$, $0 \le n \le N-1$ (1)

Where N is defined as number of samples taken. Typically in the sampling frequency used is 8khz. Thus if the two individual frequency components of the signal can be identified then the number dialed can be decoded.
In this report we have used (dual tone and digit/symbols) interchangeably but both mean the same. Dual tone means the encoded samples of the corresponding DTMF digits/symbols.
The DTMF encoder is implemented in MATLAB function dtmfe.m. The implementation is based on a digital oscillator, that will generate sinusoidal tones at frequencies F_o in response to an input signal $x[n] = \delta[n]$.
Note :Implementation of DTMF Encoder

x[n] → H[n] → y[n] {y[n] = x[n]*H[n]

Consider a causal filter with
y(n) - 2*cos(2*pi * f *Ts)y(n-1) + y(n-2) =0*x(n) sin(f)x(n-1) + 0*x(n-2).
The impulse response of this system tells us that this indeed is a digital oscillator.

The H[n] is plotted and is sinusoidal and hence any input to this system will oscillate .

PROGRAM:

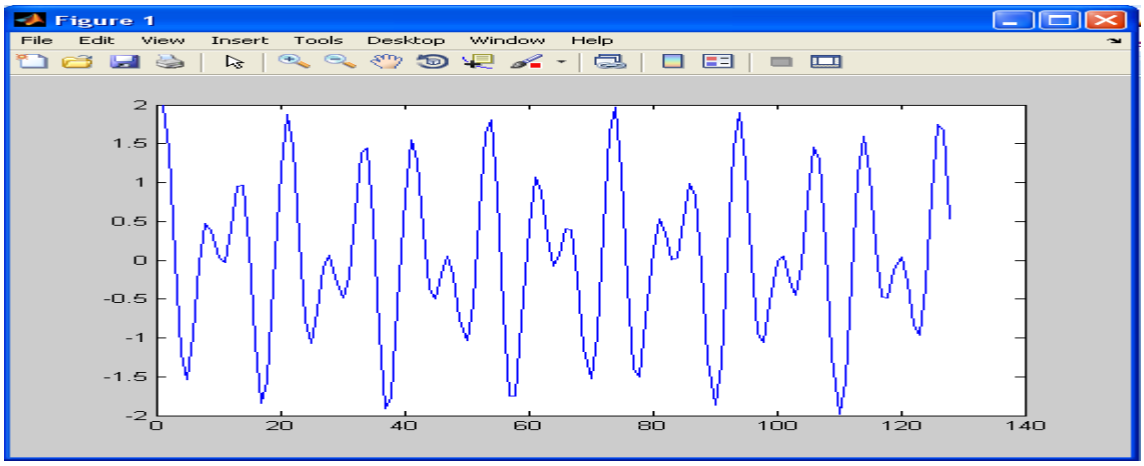
```
%Generation of DTMF signals
Fs = 8000;           % Sampling frequency
Ts = 1/Fs;
Numof_samples =input('enter the no of samples');
dialnumber=input('enter the dial number');
T = Ts*(0:Numof_samples-1)';


switch dialnumber
case 0
   F1 = 941;
   F2 = 1336;
case 1
   F1 = 697;
   F2 = 1209;
case 2
   F1 = 697;
   F2 = 1336;
case 3
   F1 = 697;
   F2 = 1477;
case 'A'
   F1 = 697;
   F2 = 1633;
case 4
   F1 = 770;
   F2 = 1209;
case 5
   F1 = 770;
   F2 = 1336;
case 6
   F1 = 770;
   F2 = 1477;
case 'B'
   F1 = 770;
   F2 = 1633;
case 7
   F1 = 852;
   F2 = 1209;
case 8
   F1 = 852;
   F2 = 1336;
case 9
   F1 = 852;
   F2 = 1477;
case 'C'
   F1 = 852;
   F2 = 1633;
```

```
case '*'
   F1 = 941;
   F2 = 1209;
case '#'
   F1 = 941;
   F2 = 1477;
otherwise
   F1 = 941;
   F2 = 1633;
end;
firstsine = cos(2*pi*F1*T);          % first sinusoidal signal
secondsine = cos(2*pi*F2*T);         % second sinusoidal signal
d = firstsine + secondsine;

dtmfoutput = d ;
figure(1);
 title('THE DTMF OUTPUT');
 plot(dtmfoutput);
```

OUTPUT: enter the no of samples 128

enter the dial number 3

RESULT:



EXP.NO: 12

IMPLEMENTATION OF DECIMATION PROCESS

AIM: To implement decimation process.

EQUIPMENT:PC with windows (95/98/XP/NT/2000).MATLAB Software

THEORY: The sampling rate of a discrete time signal x(n) can be reduced by a factor M by taking every M_th value of the signal the block diagram representation of the down sampler is shown in figure below. The quadratic symbol in below figure with arrow pointing down words is called a down sampler. The output signal y(n) is a down sampled signal of the input signal x(n) and can be represented by

$$y(n) = x(Mn)$$

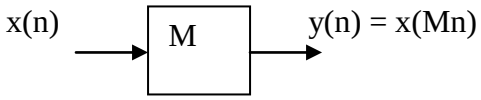


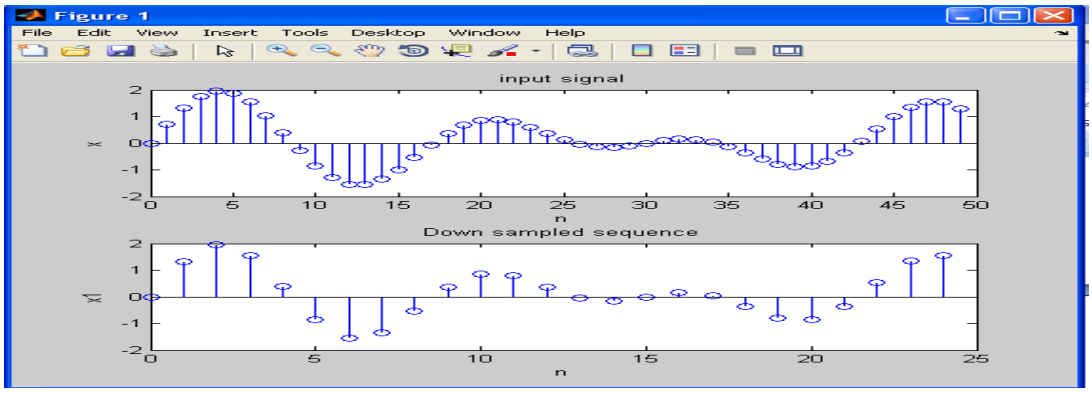
fig: down sampler

Let us assume the signal x(n) as shown in output figure. The down sampled signal x(n) can be obtained by simply keeping every M_th sample and removing (M-1) in between samples. This process is equal to reducing the sampling rate by a factor M.

PROGRAM:

```
%illustration of down sampling(decimation)
clear all;
N=50;
n=0:1:N-1;
x=sin(2*pi*n/20)+sin(2*pi*n/15);
M=2;
x1=x(1:M:N);
n1=1:1:N/M;
subplot(2,1,1);
stem(n,x);
xlabel('n');ylabel('x');title('input signal');
subplot(2,1,2);
stem(n1-1,x1);
xlabel('n');ylabel('x1');title('Down sampled sequence');
```

RESULT:



EXP.NO: 13

IMPLEMENTATION OF INTERPOLATION PROCESS

AIM: To implement Interpolation process.

EQUIPMENT: PC with windows (95/98/XP/NT/2000). MATLAB Software

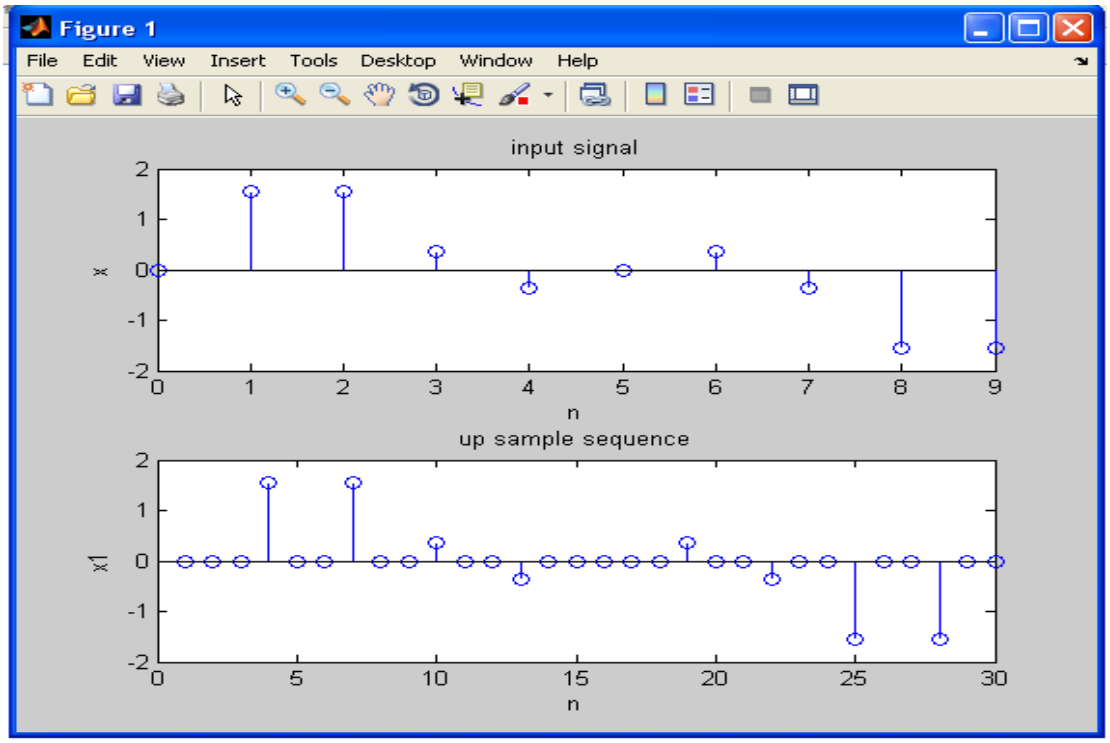
THEORY: The sampling rate of a discrete time signal can be increased by a factor L by placing L-1 equally spaced zeros between each pair of samples. Mathematically up sampling is represented by

$$Y(n) = x(n/L) \quad n= 0, +L, +2L,\ldots\ldots..$$

PROGRAM:

```
%illustration of up sampling(interpolation)
clear all;
N=10;
n=0:1:N-1;
x=sin(2*pi*n/10)+sin(2*pi*n/5);
L=3;
x1=[zeros(1,L*N)]
n1=1:1:L*N;
j=1:L:L*N;
x1(j)=x;
subplot(2,1,1);
stem(n,x);
xlabel('n');ylabel('x');title('input signal');
subplot(2,1,2);
stem(n1,x1);
xlabel('n');ylabel('x1');title('up sample sequence');
```

RESULT:



EXP.NO: 14

Implementation of I/D Sampling Rate Converters

AIM: To Implement I/D Sampling Rate Converter

EQUIPMENT: PC with windows (95/98/XP/NT/2000). MATLAB Software

PROGRAM:

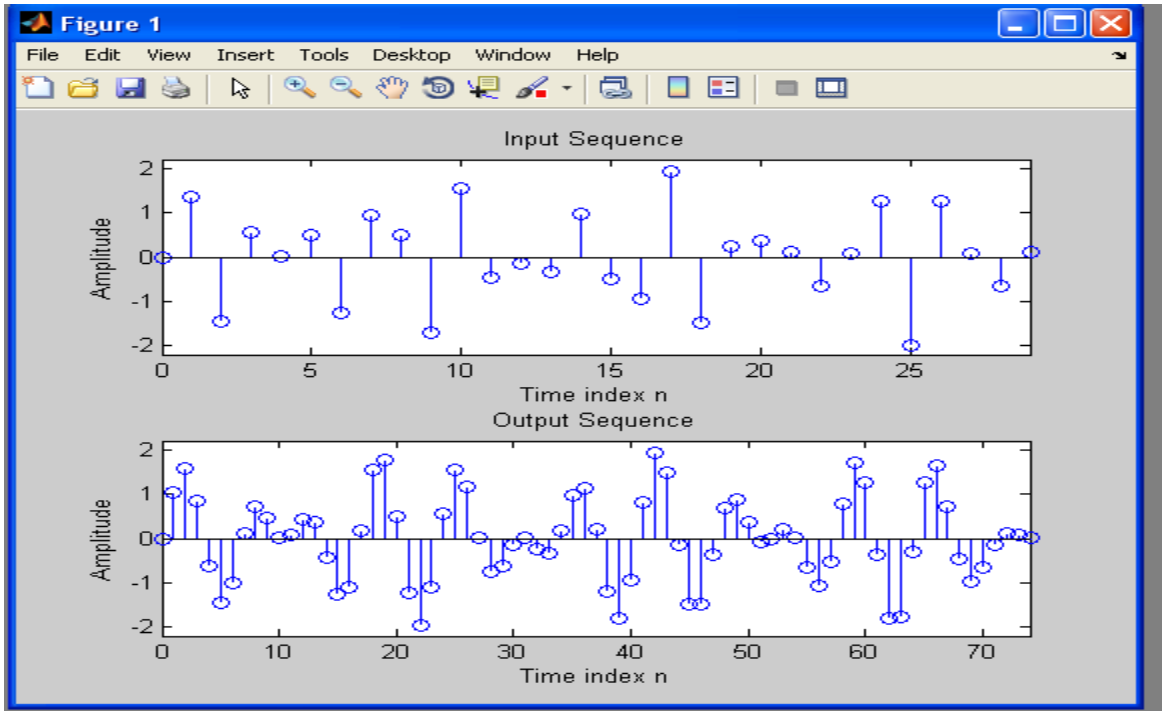
```
% Illustration of Sampling Rate Alteration by a Ratio of Two Integers
clf;clear all; clc;
L = input('Up-sampling factor = ');
M = input('Down-sampling factor = ');
n = 0:29;
x = sin(2*pi*0.43*n) + sin(2*pi*0.31*n);
y = resample(x,L,M);
subplot(2,1,1);
stem(n,x(1:30));axis([0 29 -2.2 2.2]);
title('Input Sequence');
xlabel('Time index n'); ylabel('Amplitude');
subplot(2,1,2);
m = 0:(30*L/M)-1;
stem(m,y(1:30*L/M));axis([0 (30*L/M)-1 -2.2 2.2]);
title('Output Sequence');
xlabel('Time index n'); ylabel('Amplitude');
```

OUTPUT:

Up-sampling factor = 5

Down-sampling factor = 2

RESULT:



EXP.NO: 15

IMPULSE RESPONSE OF FIRST ORDER AND SECOND ORDER SYSTEM

AIM: To determine the response of first order and second order systems.

EQUIPMENT: PC with windows (95/98/XP/NT/2000).MATLAB Software

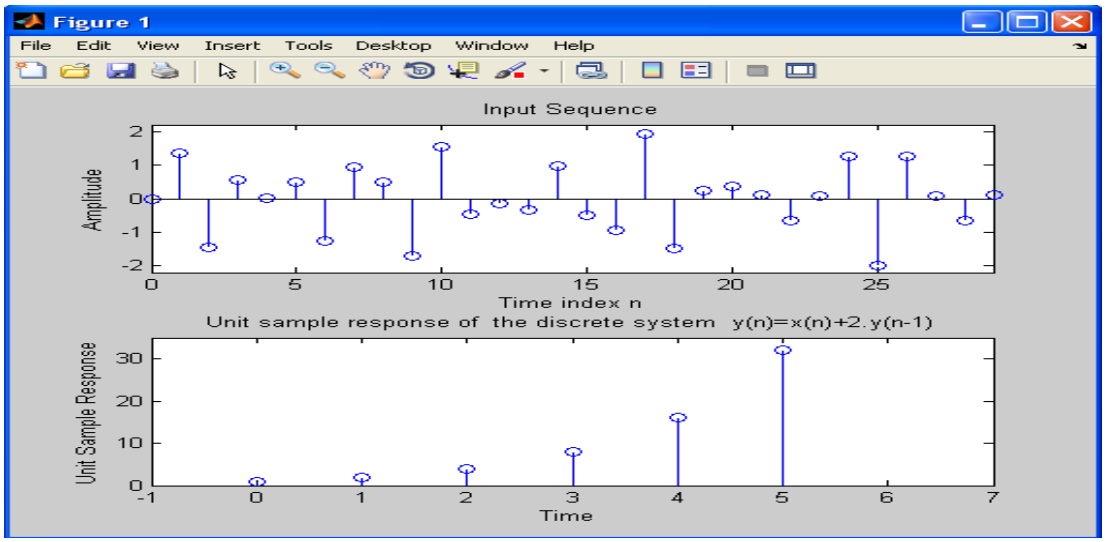
PROGRAM:

```
%This program finds the unit sample response of the  first order discrete system, expressed
%by its difference equation as  y(n)= x(n)+2.y(n-1)
a=input('enter the coefficient vector of input  starting from the coefficient of x(n) term')
b=input('enter the  coefficient  vector of  output starting from the coefficient of y(n) term')
n1=input('enter the lower limit of the range of impulse response')
n2=input('enter the upper limit of the range of impulse response')
n=[n1:n2];
x=zeros(1,length(n));
for i=1:length(n)
   if n(i)==0
      x(i)=1;
   end
end
h=filter(a,b,x);
stem(n,h)
title('Unit sample response of  the discrete system  y(n)=x(n)+2.y(n-1)');
xlabel('Time ')
ylabel('Unit Sample Response')
axis([-1 7 0 35])
```

OUTPUT:

enter the coefficient vector of input starting from the coefficient of x(n) term1
a =1
enter the coefficient vector of output starting from the coefficient of y(n) term[1 -2] b =1 -2
enter the lower limit of the range of impulse response0 n1 = 0
enter the upper limit of the range of impulse response5 n2 = 5

RESULT:



PROGRAM:

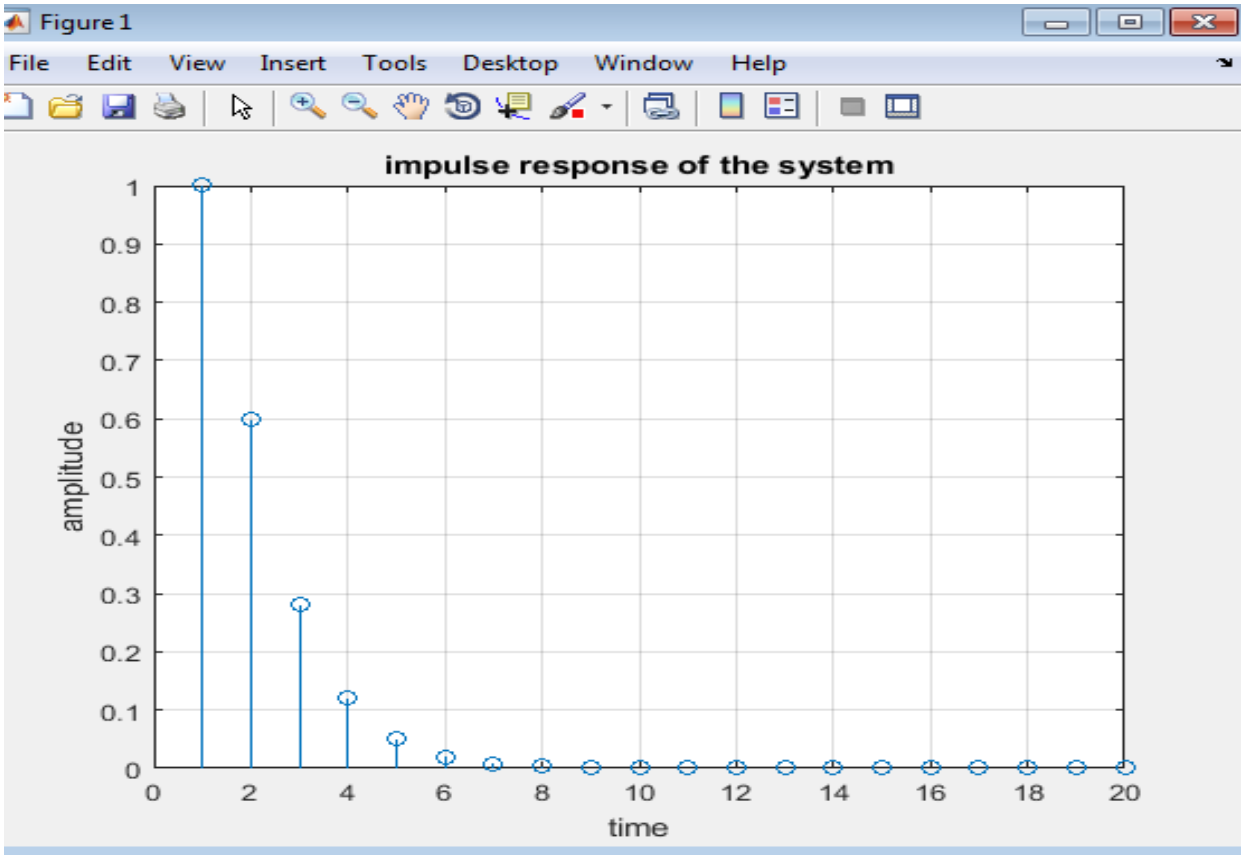
% This program find the Unit Sample response of the **second order** discrete system represented

```
clc;
clear all;
close all;
num=input('enter the numerator coefficient');
den=input('enter the denominator coefficient');
N=input('enter the no of samples');
h=impz(num,den,N);
disp('impulse response is');
disp(h);
stem(h);grid;
xlabel('time');
ylabel('amplitude');
title('impulse response of first order');
```

OUTPUT:

enter the numerator coefficient1
enter the denominator coefficient[1 -0.6 0.08]
enter the no of samples20

RESULT:



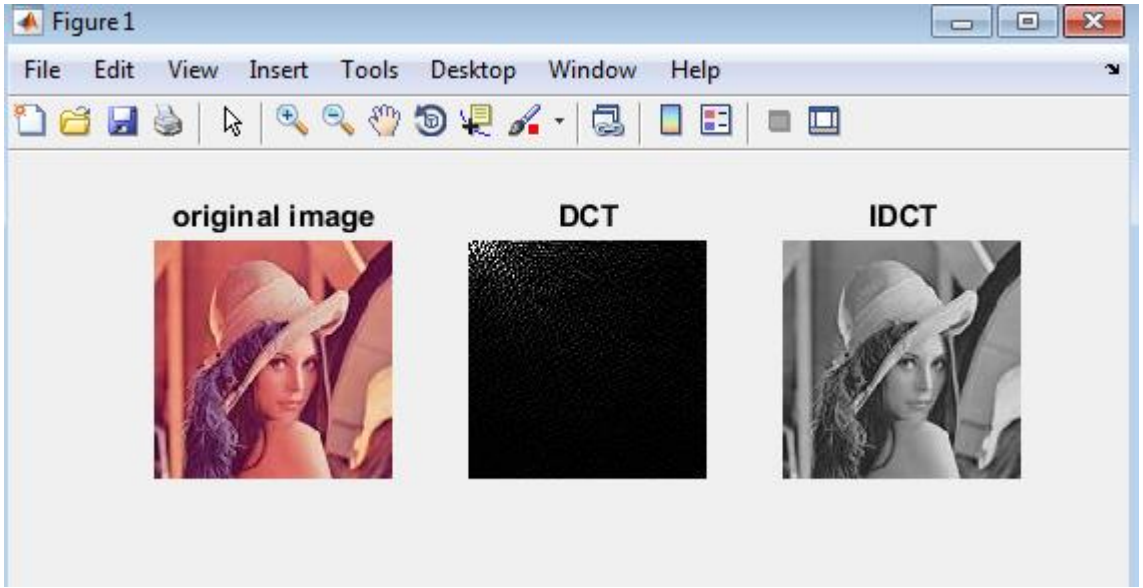
EXP.NO: 16

FIND DCT AND IDCT OF OF GIVEN IMAGE

AIM: To find DCT and IDCT of a given image
EQUIPMENT: PC with windows (95/98/XP/NT/2000).MATLAB Software

PROGRAM:

```
a=imread('C:\Users\eceavniet\Desktop\leena.jpg');
x=rgb2gray(a);
[m,n]=size(x);
y=dct2(x);
x1=idct2(y);
subplot(2,3,1);
imshow(a);
title('original image');
subplot(2,3,2);
imshow(y,[0 255]);
title('DCT');
subplot(2,3,3);
imshow(x1,[0 255]);
title('IDCT');
```



EXP.NO: 17

VERIFY LINEAR AND CORCULAR CONVOLUTION OF SIGNAL /SEQUENCE

AIM: To verify linear and circular convolution of signals /sequence
EQUIPMENT: PC with windows (95/98/XP/NT/2000).MATLAB Software

PROGRAM:

```
clc;
clear all;
close all;
x=input('Enter the input sequence x[n]= ');
lx=input('Enter the starting time index of x[n] =');
h=input('Enter the impulse response h[n]= ');
lh=input('Enter the starting time index of h[n] =');
y=conv(x,h);
n=lx+lh:length(y)+lx+lh-1;
stem(n,y);
ylabel('amplitude');
xlabel('time index');
title('Linear convolution output');
```

SAMPLE INPUTS

```
Enter the input sequencex[n]= [1 2 3 4]
Enter the starting time index of x[n] =-2
Enter the impulse response h[n]= [3 3 4]
Enter the starting time index of h[n] =-1
```

b)CONVOLUTION WITHOUT USING CONV

```
clc;
close all
clear all
x=input('Enter x: ')
h=input('Enter h: ')
m=length(x);
n=length(h);
X=[x,zeros(1,n)];
H=[h,zeros(1,m)];
for i=1:n+m-1;
Y(i)=0;
for j=1:m
if(i-j+1>0)
Y(i)=Y(i)+X(j)*H(i-j+1);
else
end
end
end
stem(Y);
ylabel('Y[n]');
xlabel('----->n');
title('Convolution of Two Signals without conv function');
```

SAMPLE INPUTS

enter first sequence[1 2 3 4]
enter second sequence[3 3 4]

CIRCULAR CONVOLUTION

MATLAB CODE

```
clc;
clear all;
close all;%Circular convolution
x=input('first seqn');
h=input('secondseqn');
subplot(3,1,1);
stem(x);
subplot(3,1,2);
stem(h);n1=length(x);
n2=length(h);
N=max(n1,n2);
x=[x,zeros(1,N-n1)];
h=[h,zeros(1,N-n2)];
for n=1:Ny(n)=0;
for k=1:N
y(n)=y(n)+x(k)*h(mod(n+N-k,N)+1);
end;
end;
subplot(3,1,3);
stem(y);
```

SAMPLE INPUTS

first seqn[1 2 3 4]second seqn[4 3 2 1]

RESULT

The circular convolution of two given sequences was performed using MATLABand the resulting sequencewas plotted.y = 24 22 24 30

EXP.NO: 18

VERIFY SPATIAL DOMAIN FILTERING OF GIVEN

AIM: Read a 256x256 image. Do the following operations.

a) Filtering using simple averaging masks.
b) Median filtering
c) Gaussian filtering
d) Triangular filtering(Pyramidal filter, cone filter)
e) Compare your results

EQUIPMENT: PC with windows (95/98/XP/NT/2000).MATLAB Software

PROGRAM:

```
clc
clear all
close all
A=imread('boat_512.tiff');
A=imresize(A,[256 256]);
%% Creating a 5x5 averaging filter
h1=fspecial('average',5);
B1=imfilter(A,h1,'replicate');
imshow(A)
title('Original image')
figure
imshow(B1)
title('Output of averaging filter')
%% Creating a 5x5 Median filter
B2=medfilt2(A, [5 5]);
figure
imshow(B2)
title('Output of median filter')
%% Creating a 5x5 Gaussian filter
h2=fspecial('gaussian',[5 5],1);
B3=imfilter(A,h2,'replicate');
figure
imshow(B3)
title('Output of Gaussian filter')
%% Creating a 5x5 Pyramidal filter
h3=(1/81).*[1 2 3 2 1 ;2 4 6 4 2 ; 3 6 9 6 3 ; 2 4 6 4 2 ; 1 2 3 2 1 ];
B4=imfilter(A,h3,'replicate');
figure
imshow(B4)
title('Output of pyramidal filter')
%% Creating a 5x5 Cone filter
h4=(1/25).*[0 0 1 0 0;0 2 2 2 0;1 2 5 2 1;0 2 2 2 0;0 0 1 0 0];
B5=imfilter(A,h4,'replicate');
figure
imshow(B5)
title('Output of cone filter')
```

Output

Original image

Output of averaging filter

Output of median filter

Output of Gaussian filter

Output of pyramidal filter

Output of cone filter

Observations

All the filters used above are defined using 5x5 masks. No noise has been added to any of the images. From the above images, we can see that the simple averaging filter produces the largest amount of blurring (smoothening) of the image. Out of all the filters above, the Gaussian filter produces the best results in terms of the visual perception.

EXP.NO: 18

DEVELOP AN ALGORITHM TO IMPLEMENT HISTOGRAM EQUALIZATION AND HISTOGRAM SPECIFICATION OF OF GIVEN IMAGE

AIM: to develop an algorithm to implement Histogram equalization and Histogram Specification of given image

EQUIPMENT: PC with windows (95/98/XP/NT/2000).MATLAB Software

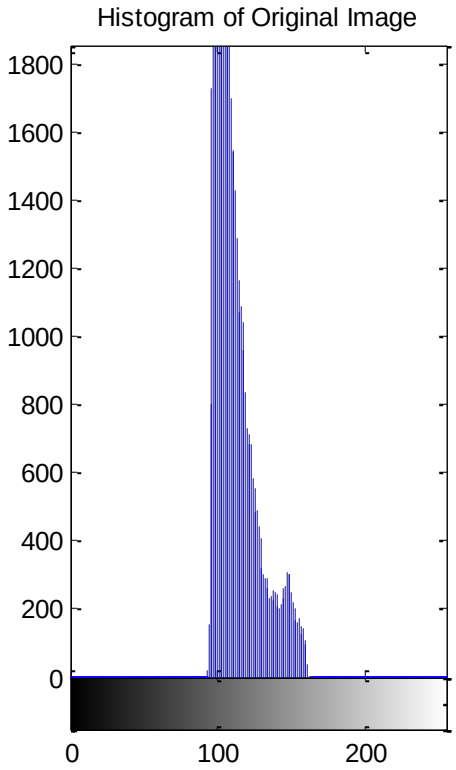
Matlab Code:

a) **Histogram equalization**

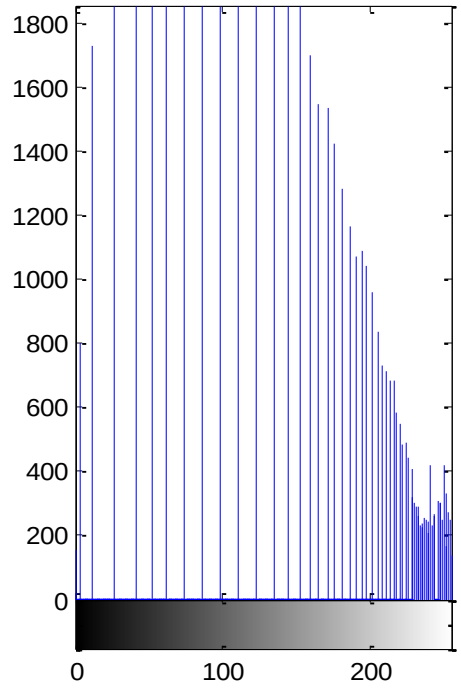
```
%% Histogram Equalization %%
% Implementation of Algorithm
clc;
clear all;
close all;
A1=imread('einstein.tif');
A=imresize(A1,[256 256]);
B=uint8(zeros(256,256));
freq=zeros(256,1);
probf=zeros(256,1);
cum_prob=zeros(256,1);
cum=zeros(256,1);
output=zeros(256,1);
for i=1:256
   for j=1:256
      value=A(i,j);
      freq(value+1)=freq(value+1)+1;
      probf(value+1)=freq(value+1)/(256*256);
   end
end
sum=0;
no_bins=255;
for i=1:size(probf)
   sum=sum+freq(i);
   cum_prob(i)=sum/(256*256);
   output(i)=round(cum_prob(i)*no_bins);
end
for i=1:256
   for j=1:256
      B(i,j)=output(A(i,j)+1);
   end
end
figure
subplot (1,2,1)
imshow(A);
title('Original Image')
subplot (1,2,2)
```

```
imhist(A)
title('Histogram of Original Image')
figure
subplot (1,2,1)
imshow(B);
title('Histogram Equalized Image');
subplot (1,2,2)
imhist(B)
title('Histogram of Equalized Image')

%% Implemenattion using built in function
Heq = histeq(A);
figure
subplot (1,2,1)
imshow(Heq);
title('Histogram Equalized Image using built-in function');
subplot (1,2,2)
imhist(Heq)
title('Histogram of Equalized Image')
```

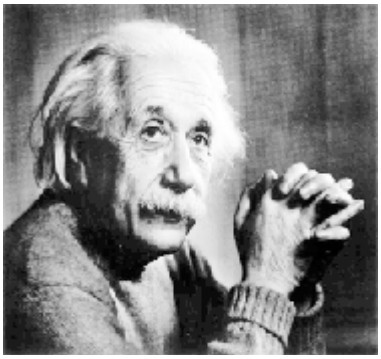
Output



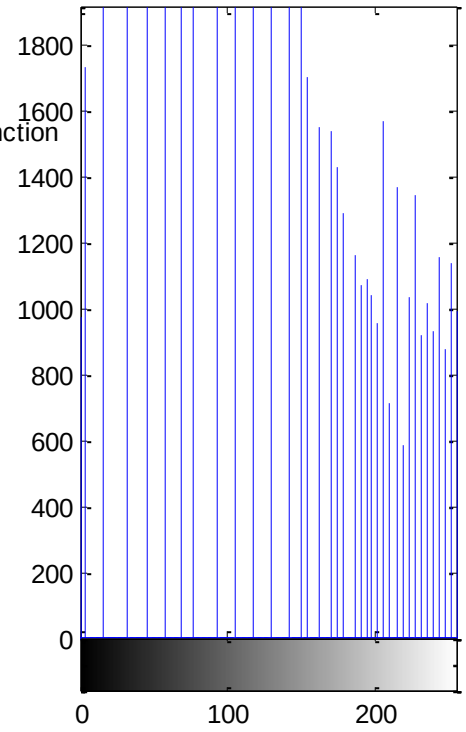
Histogram of Equalized Image



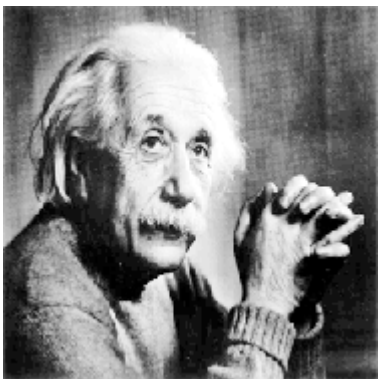
Histogram Equalized Image



Histogram of Equalized Image



Histogram Equalized Image using built-in function



b) Histogram Specification

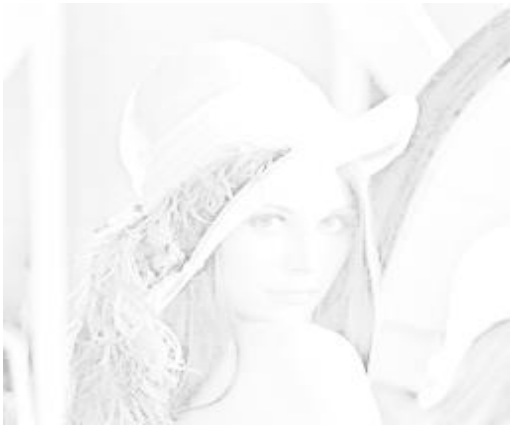
```
%% Histogram Specification
clear all;close all;clc
A1 = imread('lena512','bmp');
A  =imresize(A1,[256 256]);
%% Finding the pdf of input image
B=uint8(zeros(256,256));
freq=zeros(256,1);
probf=zeros(256,1);
cum_prob=zeros(256,1);
cum=zeros(256,1);
for i=1:256
   for j=1:256
      value=A(i,j);
      freq(value+1)=freq(value+1)+1;
      probf(value+1)=freq(value+1)/(256*256);
   end
end
sum1=0;
no_bins=255;
for i=1:size(probf)
   sum1=sum1+freq(i);
   cum(i)=sum1;
   cum_prob(i)=cum(i)/(256*256);
end
%% Histogram Matching for exponential pdf
pdfZ=1+exp((0:.1:25.5));% Specified pdf
pdfZ=pdfZ./sum(pdfZ);
zk=pdfZ*triu(ones(256));
mapping=zeros(256);
z0=zeros(256);
for q=1:256
   for p=mapping(q)+1:256
      if ((zk(p)-cum_prob(q)) >= 0)
         mapping(q) = p;
         list=find(A == q-1);
         z0(list)=p;
         break
      end
   end
end
Z1=reshape(z0,256*256,1);
Zpdf=hist(Z1,0:1:255)/100000;
figure
imshow(A);
title('Original Image')
figure
z0=uint8(z0);
imshow(z0)
```

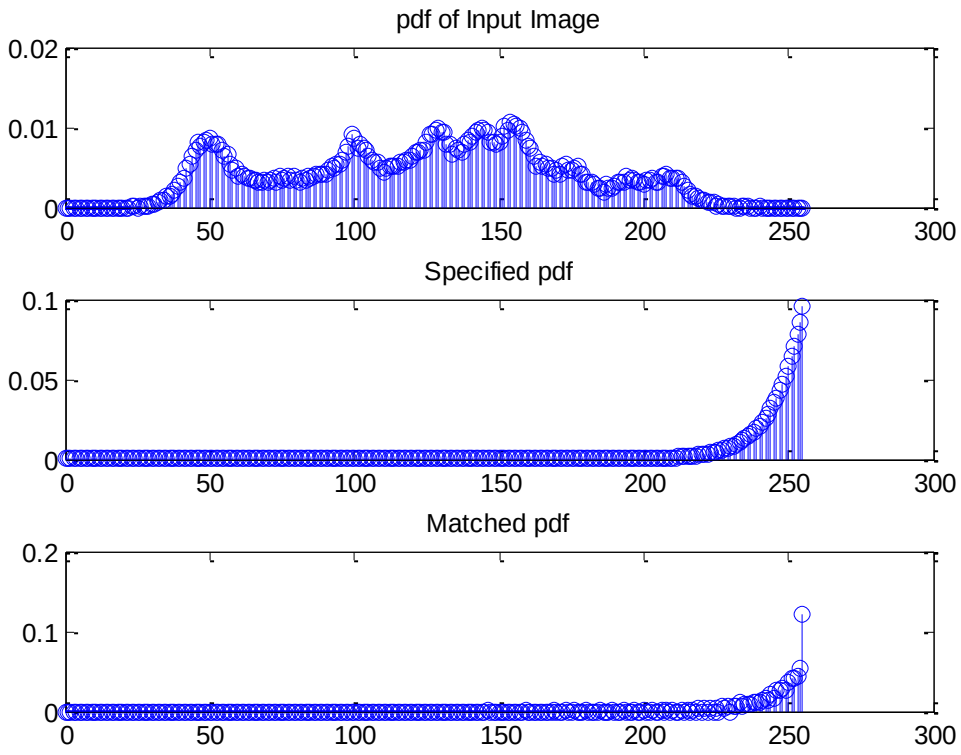
```
title(' Histogram Matched Image - Exponential pdf')
figure
subplot(3,1,1),stem(0:255,probf);
title('pdf of Input Image')
subplot(3,1,2),stem(0:255,pdfZ);
title('Specified pdf');
subplot(3,1,3),stem(0:255,Zpdf);
title('Matched pdf');
```

Output

Original Image

Histogram Matched Image - Exponential pdf




pdf of Input Image
0.02
0.01
0
0
50
100
150
200
250
300
Specified pdf
0.1
0.05
0
0
50
100
150
200
250
300
Matched pdf
0.2
0.1
0
0
50
100
150
200
250
300

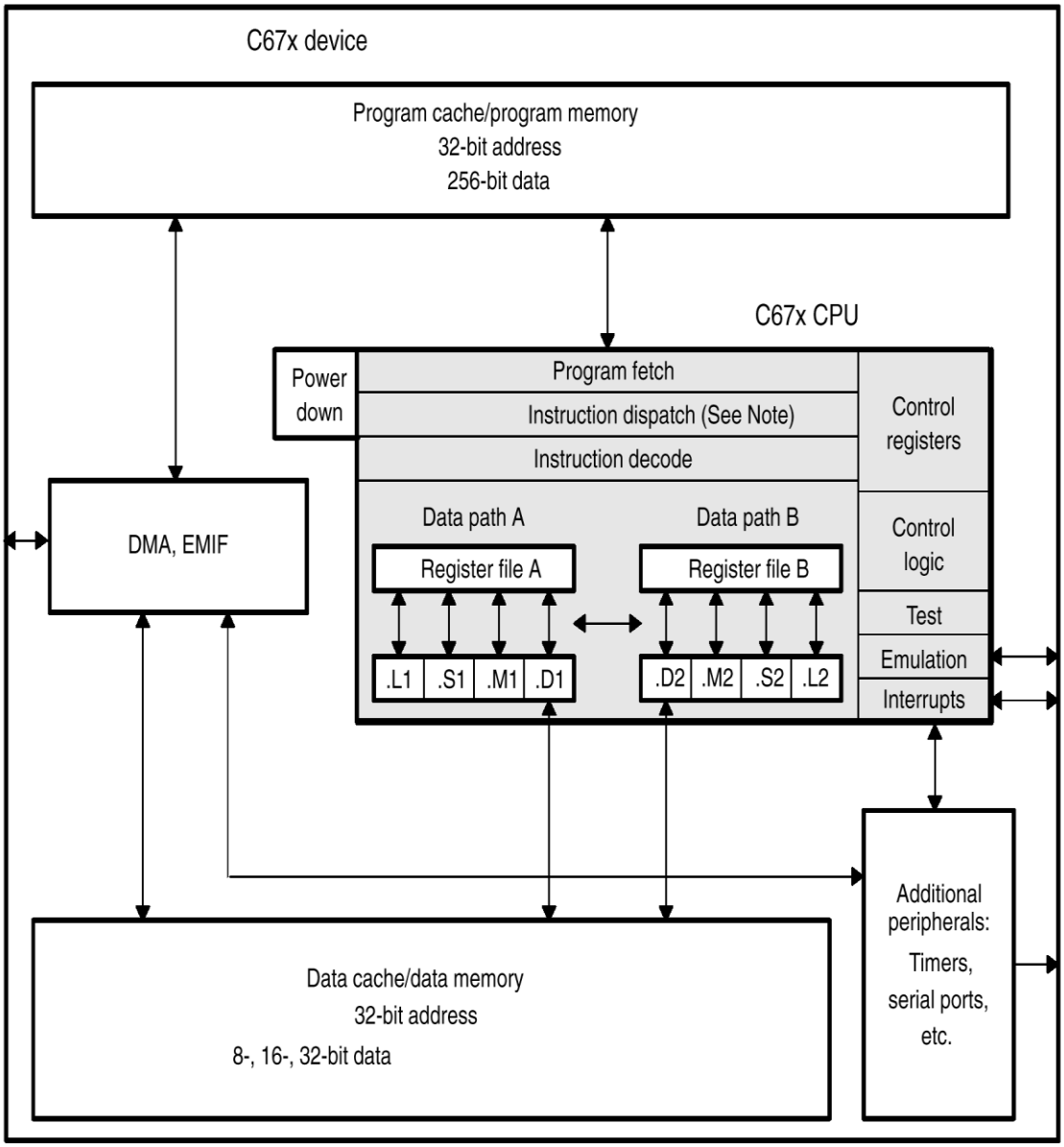
ARCHITECTURE AND INSTRUCTION **SET OF DSPCHIP-TMS320C6713**

AIM: To Study the architecture and Instruction set of DSP chips.

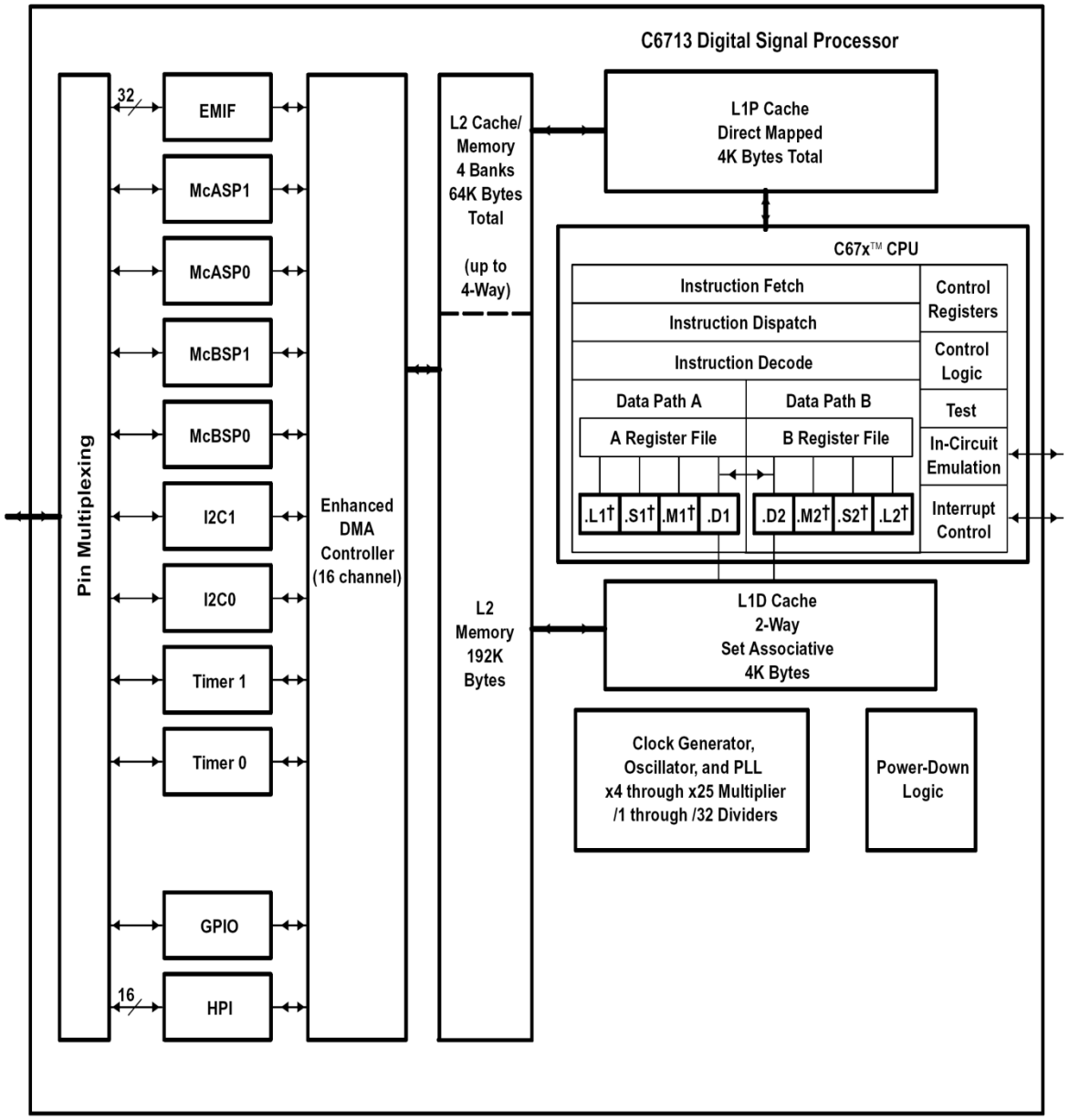
Features of Highest-Performance Floating-Point Digital Signal Processor TMS320c6713

- Enhanced Harvard Architecture
- VLIW Parallel Architecture
- Rich Addressing modes
- Two general purpose Register files (A0-A15 & B0-B15)
- 32/64- Bit Data Word
- Rich Instruction set
- Eight 32-Bit Instructions/Cycle
- 32/64-Bit Data Word
- 4.4-, 6.7-ns Instruction Cycle Time
- 1800 MIPS/1350 MFLOPS
- Rich Peripheral Set, Optimized for Audio
- Highly Optimized C/C++ Compiler

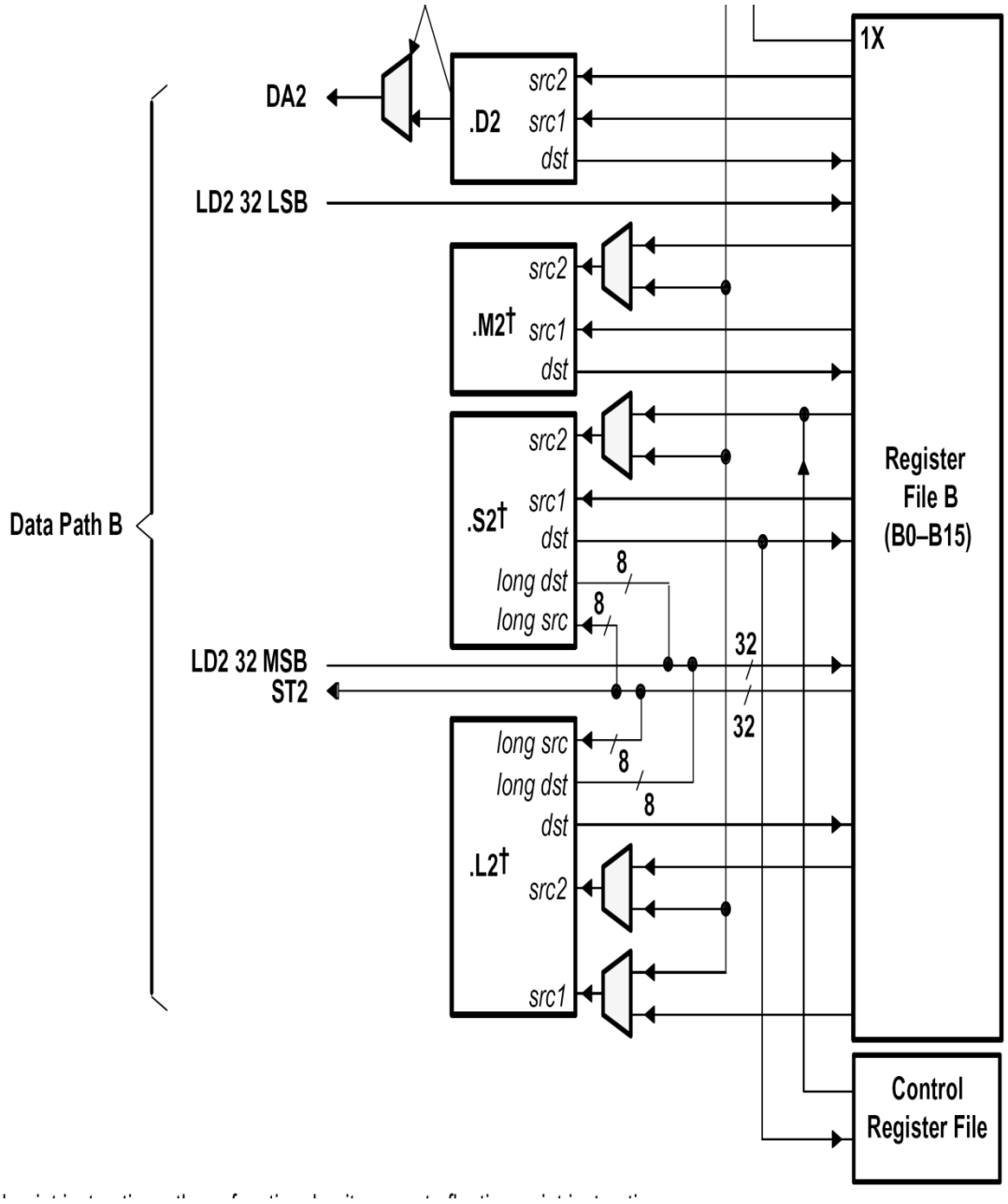
TMS320C67x Block Diagram



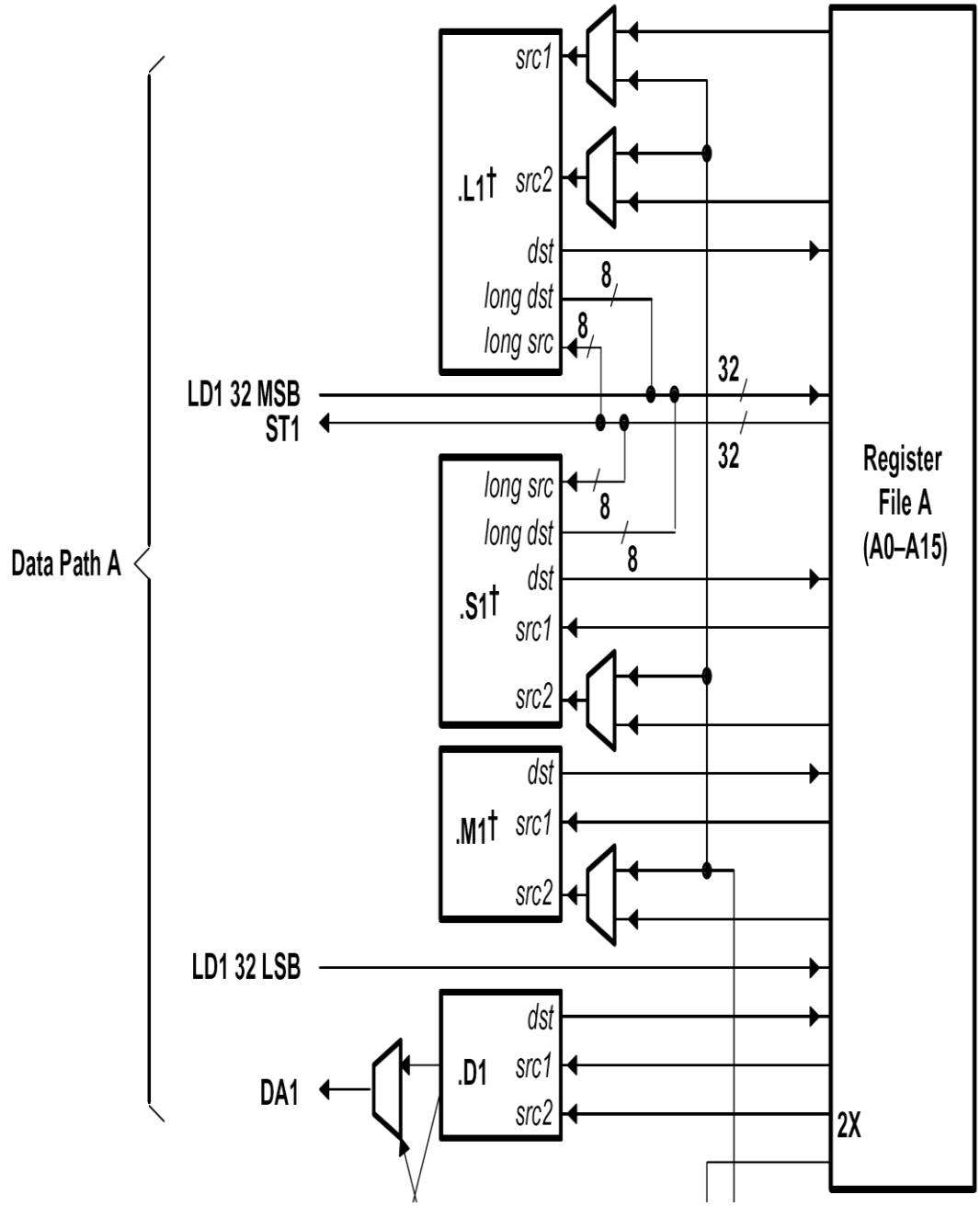
TMS320C6713 ARCHITECTURE



CPU (DSP Core)


1X
DA2
.D2
src2
src1
dst
LD2 32 LSB
.M2†
src2
src1
dst
.S2†
src2
src1
dst
long dst
long src
8
8
Data Path B
LD2 32 MSB
ST2
32
32
long src
long dst
dst
8
8
.L2†
src2
src1
Register
File B
(B0–B15)
Control
Register File

CPU (DSP CORE) CONT…



TMS C6713DSK

The 6713 DSK is a low-cost standalone development platform that enables customers to evaluate and develop applications for the TI C67XX DSP family. The DSK also serves as a hardware reference design for the TMS320C6713 DSP. Schematics, logic equations and application notes are available to ease hardware development and reduce time to market.

The DSK uses the 32-bit EMIF for the SDRAM (CE0) and daughtercard expansion interface (CE2 and CE3). The Flash is attached to CE1 of the EMIF in 8-bit mode.

An on-board AIC23 codec allows the DSP to transmit and receive analog signals. McBSP0 is used for the codec control interface and McBSP1 is used for data. Analog audio I/O is done through four 3.5mm audio jacks that correspond to microphone input, line input, line output and headphone output. The codec can select the microphone or the line input as the active input. The analog output is driven to both the line out (fixed gain) and headphone (adjustable gain) connectors. McBSP1 can be re-routed to the expansion connectors in software.

A programmable logic device called a CPLD is used to implement glue logic that ties the board components together. The CPLD has a register based user interface that lets the user configure the board by reading and writing to the CPLD registers. The registers reside at the midpoint of CE1.

The DSK includes 4 LEDs and 4 DIPswitches as a simple way to provide the user with interactive feedback. Both are accessed by reading and writing to the CPLD registers.

An included 5V external power supply is used to power the board. On-board voltage regulators provide the 1.26V DSP core voltage, 3.3V digital and 3.3V analog voltages. A voltage supervisor monitors the internally generated voltage, and will hold the board in reset until the supplies are within operating specifications and the reset button is released. If desired, JP1 and JP2 can be used as power test points for the core and I/O power supplies.

Code Composer communicates with the DSK through an embedded JTAG emulator with a USB host interface. The DSK can also be used with an external emulator through the external JTAG connector.

TMS320C6713 DSP FEATURES

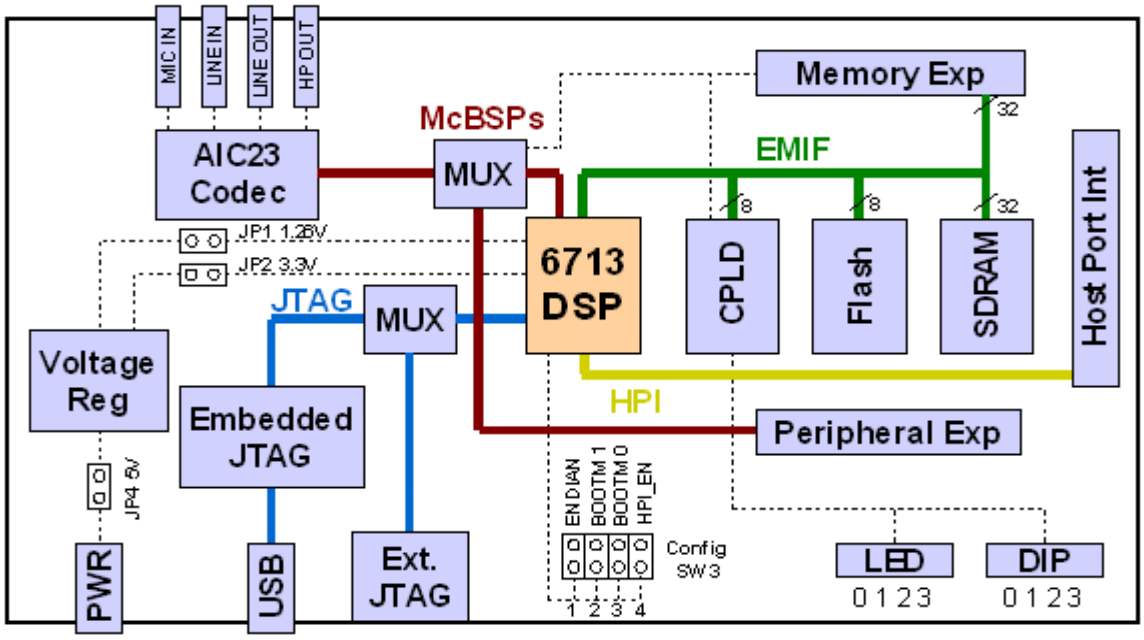
- Highest-Performance Floating-Point Digital Signal Processor (DSP):
  - Eight 32-Bit Instructions/Cycle
  - 32/64-Bit Data Word
  - 300-, **225**-, 200-MHz (GDP), and **225-**, 200-, 167-MHz (PYP) Clock Rates
  - 3.3-, 4.4-, 5-, 6-Instruction Cycle Times
  - 2400/1800, 1800/1350, 1600/1200, and 1336/1000 MIPS /MFLOPS
  - Rich Peripheral Set, Optimized for Audio
  - Highly Optimized C/C++ Compiler
  - Extended Temperature Devices Available
- Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core
  - Eight Independent Functional Units:
    - Two ALUs (Fixed-Point)
    - Four ALUs (Floating- and Fixed-Point)
    - Two Multipliers (Floating- and Fixed-Point)
  - Load-Store Architecture With 32 32-Bit General-Purpose Registers
  - Instruction Packing Reduces Code Size
  - All Instructions Conditional
- Instruction Set Features
  - Native Instructions for IEEE 754
    - Single- and Double-Precision
  - Byte-Addressable (8-, 16-, 32-Bit Data)
  - 8-Bit Overflow Protection
  - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- L1/L2 Memory Architecture
  - 4K-Byte L1P Program Cache (Direct-Mapped)
  - 4K-Byte L1D Data Cache (2-Way)
  - 256K-Byte L2 Memory Total: 64K-Byte L2 Unified Cache/Mapped RAM, and 192K-Byte Additional L2 Mapped RAM
- Device Configuration
  - Boot Mode: HPI, 8-, 16-, 32-Bit ROM Boot
  - Endianness: Little Endian, Big Endian
- 32-Bit External Memory Interface (EMIF)
  - Glueless Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
  - 512M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (16 Independent Channels)
- 16-Bit Host-Port Interface (HPI)
- Two Multichannel Audio Serial Ports (McASPs)
  - Two Independent Clock Zones Each (1 TX and 1 RX)
  - Eight Serial Data Pins Per Port:
    Individually Assignable to any of the Clock Zones
  - Each Clock Zone Includes:
    - Programmable Clock Generator
    - Programmable Frame Sync Generator
    - TDM Streams From 2-32 Time Slots

- Support for Slot Size:
  8, 12, 16, 20, 24, 28, 32 Bits
- Data Formatter for Bit Manipulation

➢ Wide Variety of I2S and Similar Bit Stream Formats
➢ Integrated Digital Audio Interface Transmitter (DIT) Supports:
- S/PDIF, IEC60958-1, AES-3, CP-430 Formats
- Up to 16 transmit pins
- Enhanced Channel Status/User Data

➢ Extensive Error Checking and Recovery

❖ Two Inter-Integrated Circuit Bus (I^2C Bus™) Multi-Master and Slave Interfaces
❖ Two Multichannel Buffered Serial Ports:
➢ Serial-Peripheral-Interface (SPI)
➢ High-Speed TDM Interface
➢ AC97 Interface
❖ Two 32-Bit General-Purpose Timers
❖ Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
❖ IEEE-1149.1 (JTAG †) Boundary-Scan-Compatible
❖ Package Options:
➢ 208-Pin PowerPAD™ Plastic (Low-Profile) Quad Flatpack (PYP)
➢ 272-BGA Packages (GDP and ZDP)
❖ 0.13-µm/6-Level Copper Metal Process
➢ CMOS Technology
❖ 3.3-V I/Os, 1.2 ‡ -V Internal (GDP & PYP)
❖ 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)



TMS320C6713 DSK Overview Block Diagram

DSK HARDWARE INSTALLATION

- Shut down and Power off the PC
- Connect the supplied USB port cable to the board
- Connect the other end of the cable to the USB port of PC
  Note: If you plan to install a Microphone, speaker, or
  Signal generator/CRO these must be plugged in properly
  Before you connect power to the DSK
- Plug the power cable into the board
- Plug the other end of the power cable into a power outlet
- The user LEDs should flash several times to indicate board is operational
- When you connect your DSK through USB for the first time on windows loaded PC the new hardware found wizard will come up. So, Install the drivers (The CCS CD contains the require drivers for C5416 DSK).
- Install the CCS software for C5416 DSK

CODE COMPOSER STUDIO

INTRODUCTION TO CODE COMPOSER STUDIO

Code Composer is the DSP industry's first fully integrated development environment (IDE) with DSP-specific functionality. With a familiar environment liked MS-based C++TM, Code Composer lets you edit, build, debug, profile and manage projects from a single unified environment. Other unique features include graphical signal analysis, injection/extraction of data signals via file I/O, multi-processor debugging, automated testing and customization via a C-interpretive scripting language and much more.

CODE COMPOSER FEATURES INCLUDE:

- IDE
- Debug IDE
- Advanced watch windows
- Integrated editor
- File I/O, Probe Points, and graphical algorithm scope probes
- Advanced graphical signal analysis
- Interactive profiling
- Automated testing and customization via scripting
- Visual project management system
- Compile in the background while editing and debugging
- Multi-processor debugging
- Help on the target DSP

To create a system configuration using a standard configuration file:

Step 1: Start CCS Setup by double clicking on the Setup CCS desktop icon.

Step 2: select Family → c67xx

Platform → simulator

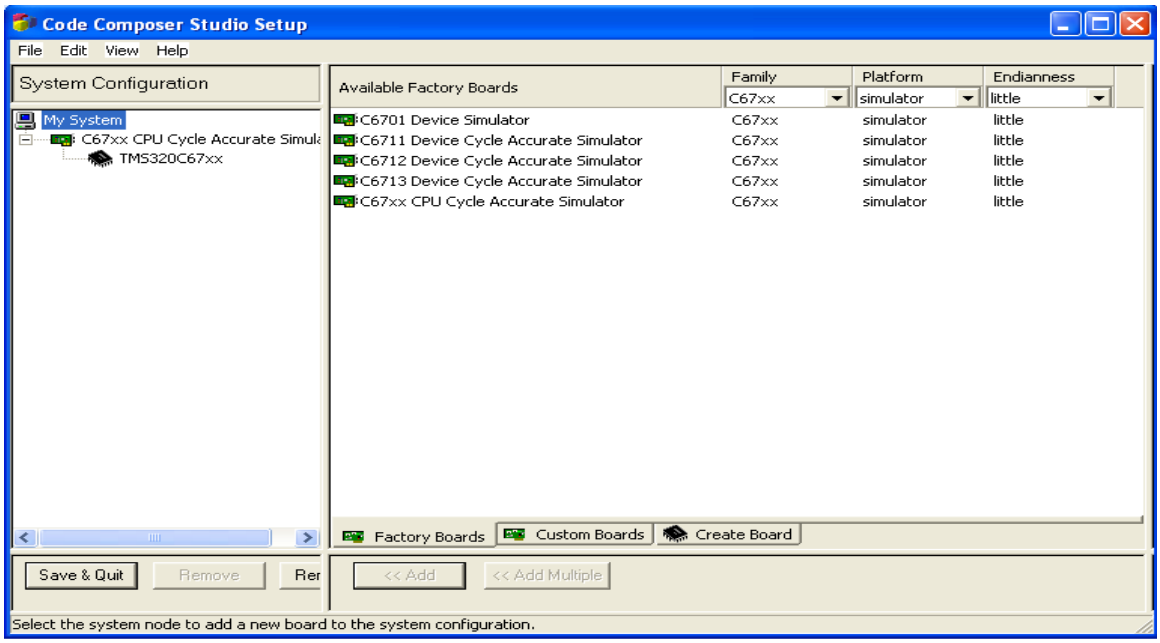
Endianness → little

.

Step 3: Click the Import button (File → import) to import our selection (c67xx_sim.ccs) to the system configuration currently being created in the CCS Setup window.

Step 4: Click the Save and Quit button to save the configuration in the System Registry.

Step 5: Click the Yes button to start the CCS IDE when we exit CCS Setup. The CCS Setup closes and the CCS IDE automatically opens using the configuration we just created.



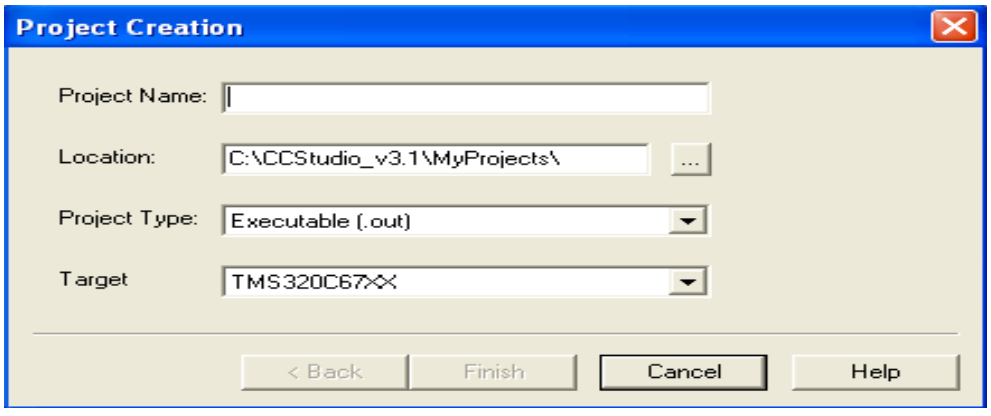
PROCEDURE TO WORK ON CODE COMPOSER STUDIO

Step 1: Creating a New Project

From the Project menu, choose New.

In the Project Name field, type the name we want for our project. Each project we create must have a unique name, and Click Finish. The CCS IDE creates a project file called projectname.pjt. This file stores our project settings and references the various files used by our project.

The Project Creation wizard window displays.

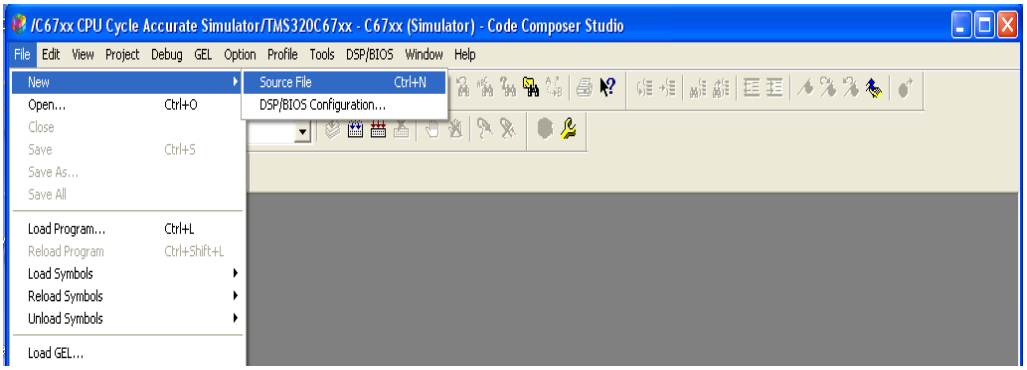


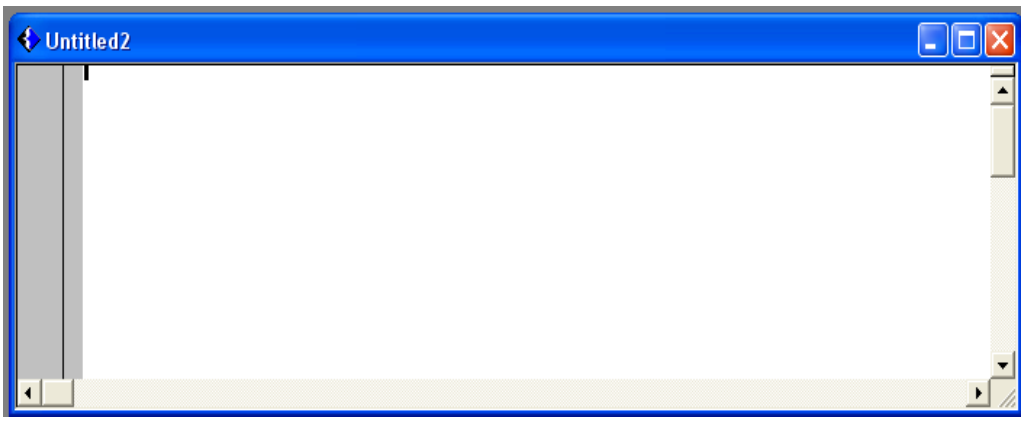
Step 2: Creating a source file

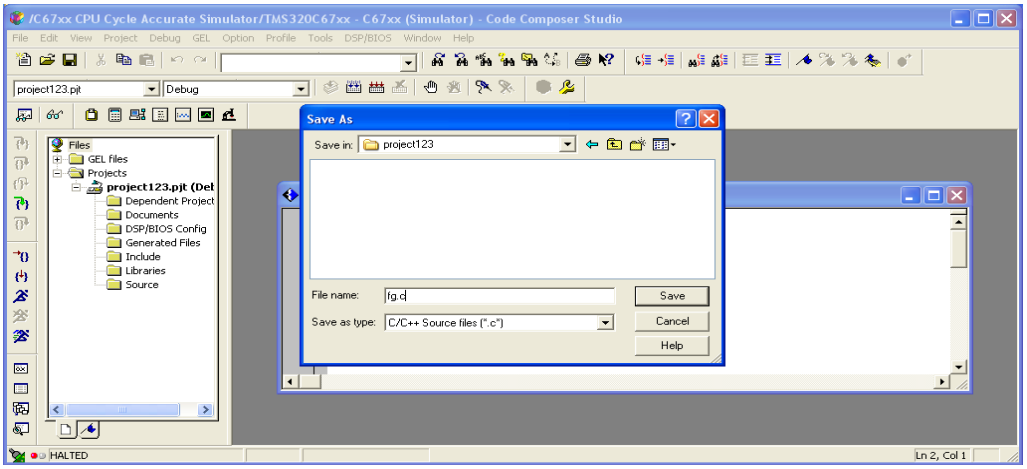
Create a new source file using 'File →new →source file ' pull down menu and save the source file with **.c** extension in the current project name directory**.**

Save as type: c/c++ source file (*.c*)

Path: C:\CCStudio_v3.1\ MyProjects\Project Name\





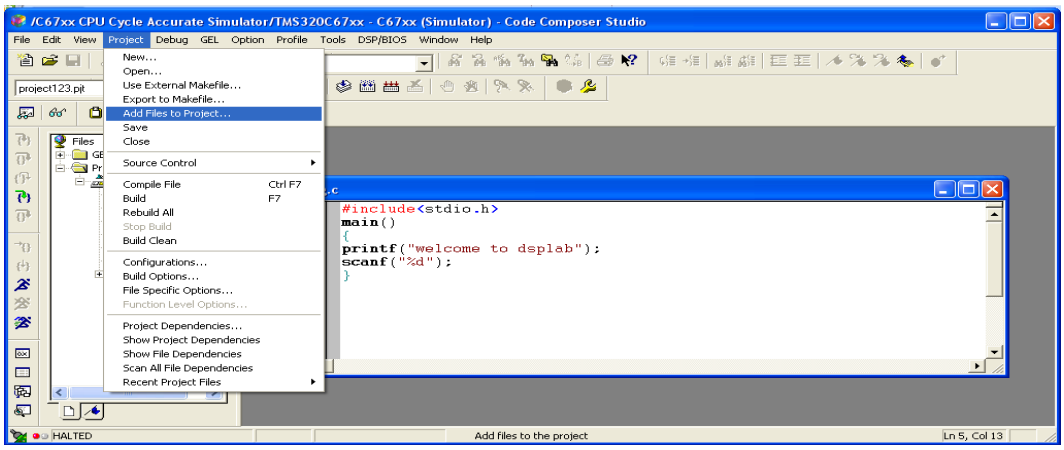


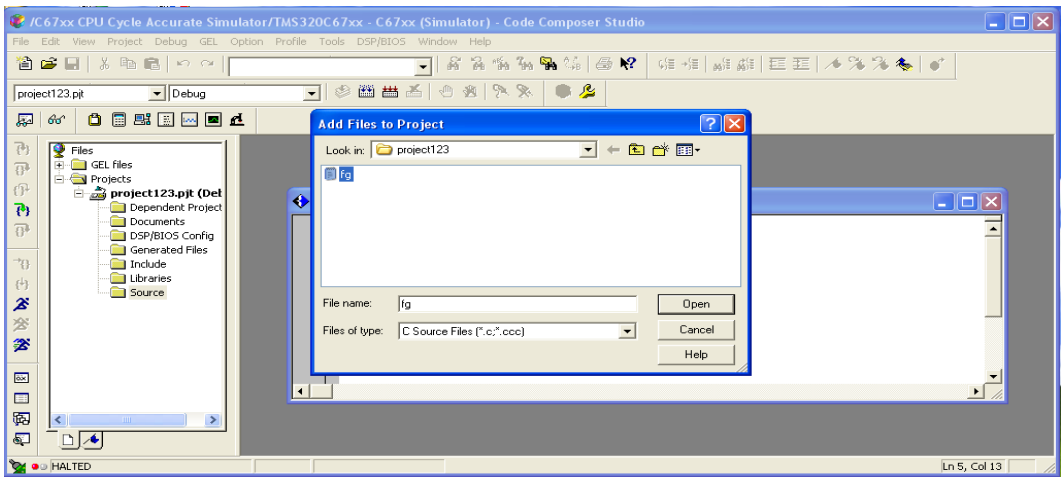
Step 3: Add files to our project (source file\ library file\ linker file)

Source file: Add the source file in the project using 'Project→add files to project' pull down menu.

Files of type: c/c++ source file (*.c*)

Path: C:\CCStudio_v3.1\ MyProjects\Project Name\file_name.c

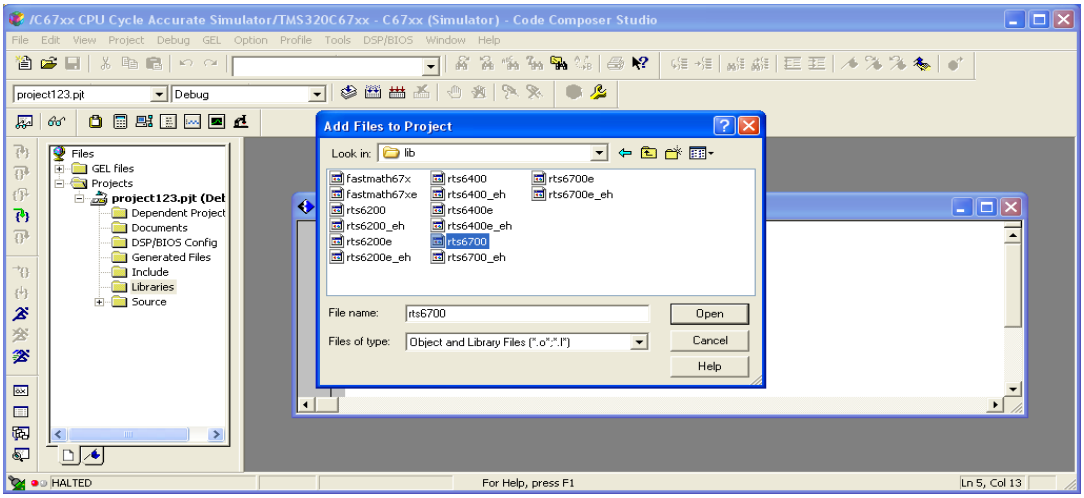




Library file: Add the library file in the project using 'Project→add files to project' pull down menu.

Files of type: Object and Library Files (*.o*,*.l*)

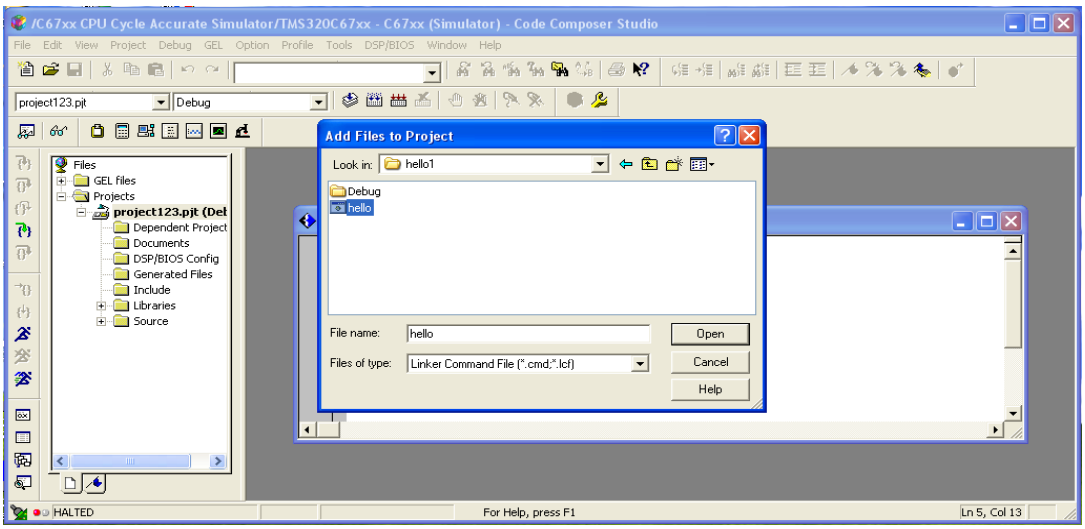
Path: C:\CCStudio_v3.1\ C6000\ cgtools\ lib \ rts6700.lib



Linker file: Add the linker file in the project using 'Project→add files to project' pull down menu.

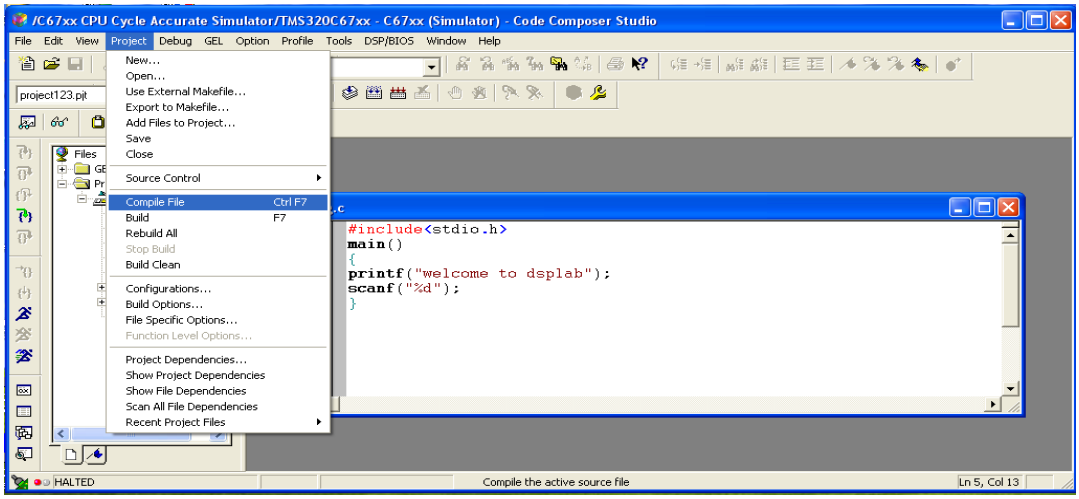
Files of type: Linker command Files (*.cmd*,*.lcf*)

Path: C:\CCStudio_v3.1\ tutorial\ dsk6711\ hello1 \ hello.cmd

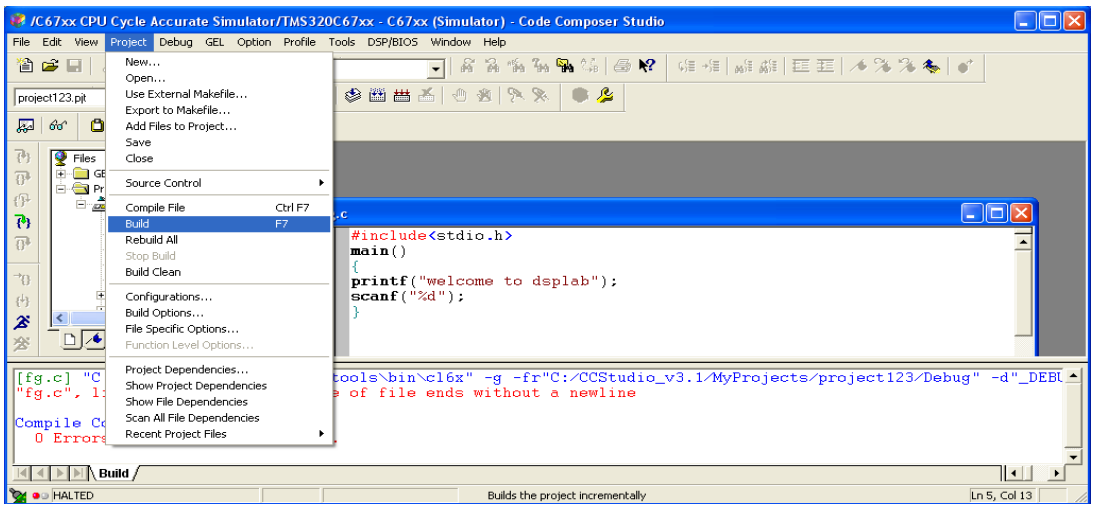


Step 4: Building and Running the Program (compile\ Build\ Load Program\ Run)

Compile: Compile the program using the 'Project-compile' pull down menu or by clicking the shortcut icon on the left side of program window.



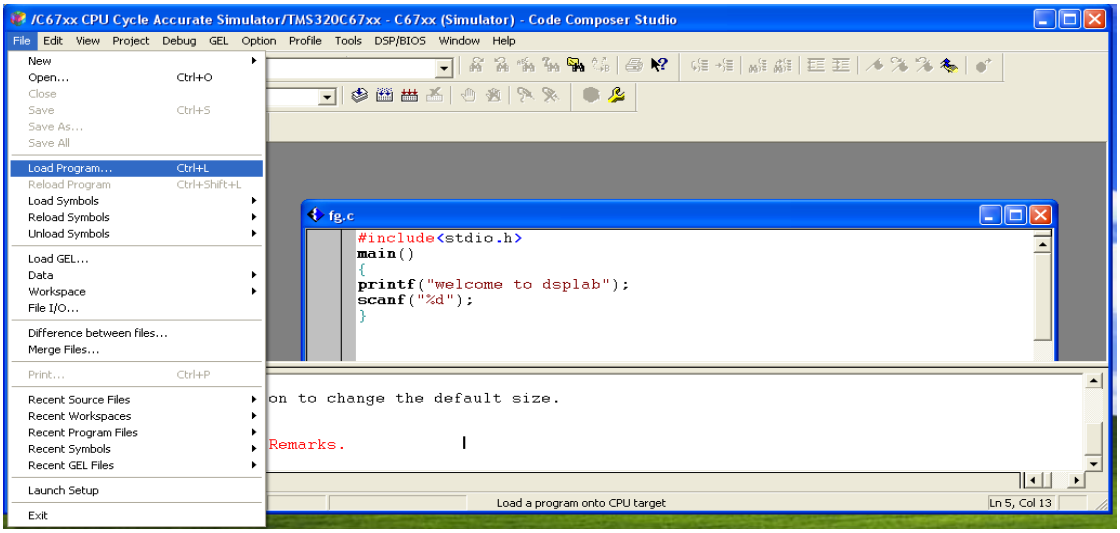
Build: Build the program using the 'Project-Build' pull down menu or by clicking the shortcut icon on the left side of program window.

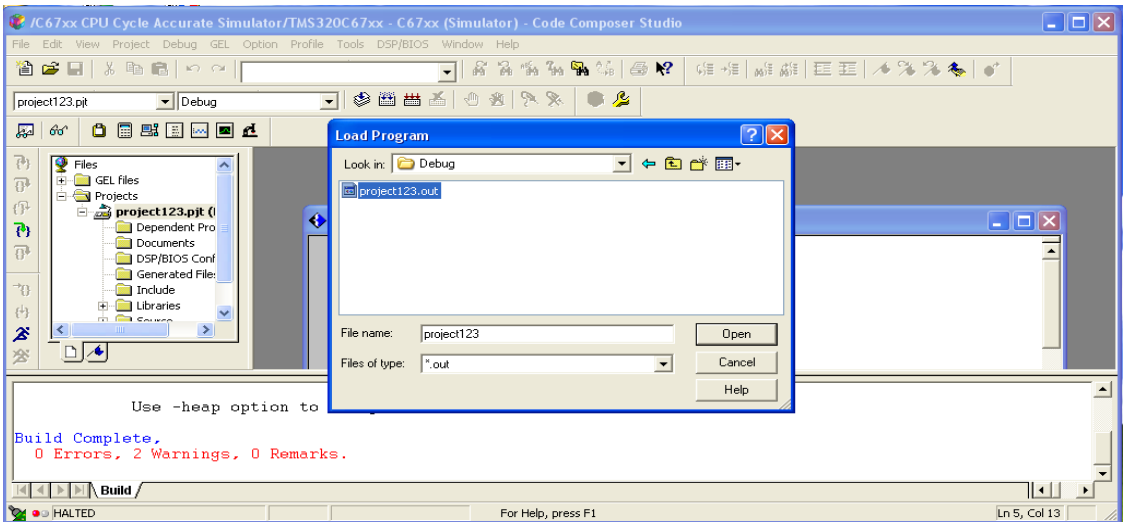


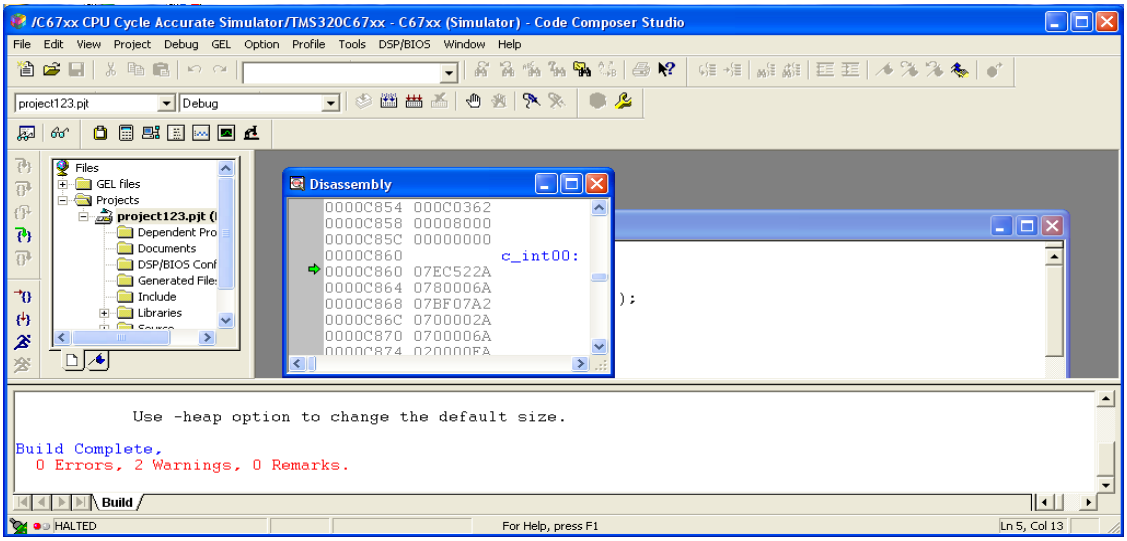
Load Program: Load the program in program memory of DSP chip using the 'File-load program' pull down menu.

Files of type:(*.out*)

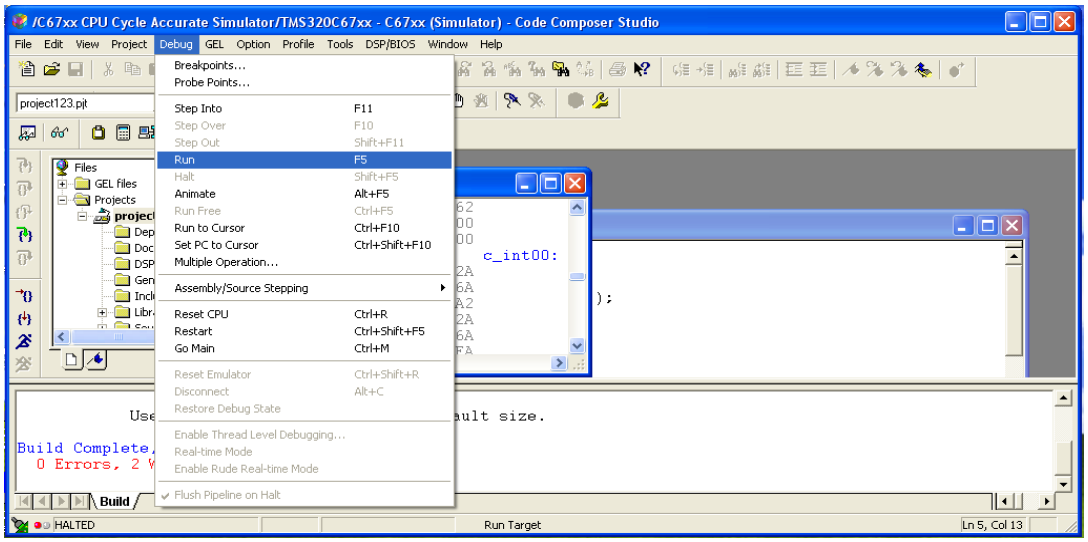
Path: C:\CCStudio_v3.1\ MyProjects\Project Name\ Debug\ Project Name.out


/C67xx CPU Cycle Accurate Simulator/TMS320C67xx - C67xx (Simulator) - Code Composer Studio
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
New
Open... Ctrl+O
Close
Save Ctrl+S
Save As...
Save All
Load Program... Ctrl+L
Reload Program Ctrl+Shift+L
Load Symbols
Reload Symbols
Unload Symbols
Load GEL...
Data
Workspace
File I/O...
Difference between files...
Merge Files...
Print... Ctrl+P
Recent Source Files
Recent Workspaces
Recent Program Files
Recent Symbols
Recent GEL Files
Launch Setup
Exit
fg.c
#include<stdio.h>
main()
{
printf("welcome to dsplab");
scanf("%d");
}
on to change the default size.
Remarks.
Load a program onto CPU target
Ln 5, Col 13


/C67xx CPU Cycle Accurate Simulator/TMS320C67xx - C67xx (Simulator) - Code Composer Studio
project123.pjt
Debug
Load Program
Look in: Debug
project123.out
File name: project123
Files of type: *.out
Open
Cancel
Help
Files
GEL files
Projects
Dependent Pro
Documents
DSP/BIOS Conf
Generated File
Include
Libraries
Use -heap option to
Build Complete,
0 Errors, 2 Warnings, 0 Remarks.
Build
HALTED
For Help, press F1
Ln 5, Col 13


/C67xx CPU Cycle Accurate Simulator/TMS320C67xx - C67xx (Simulator) - Code Composer Studio
project123.pjt
Debug
Disassembly
0000C854 000C0362
0000C858 00008000
0000C85C 00000000
0000C860 c_int00:
0000C860 07EC522A
0000C864 0780006A
0000C868 07BF07A2
0000C86C 0700002A
0000C870 0700006A
Use -heap option to change the default size.
Build Complete,
0 Errors, 2 Warnings, 0 Remarks.
Build
HALTED
For Help, press F1
Ln 5, Col 13

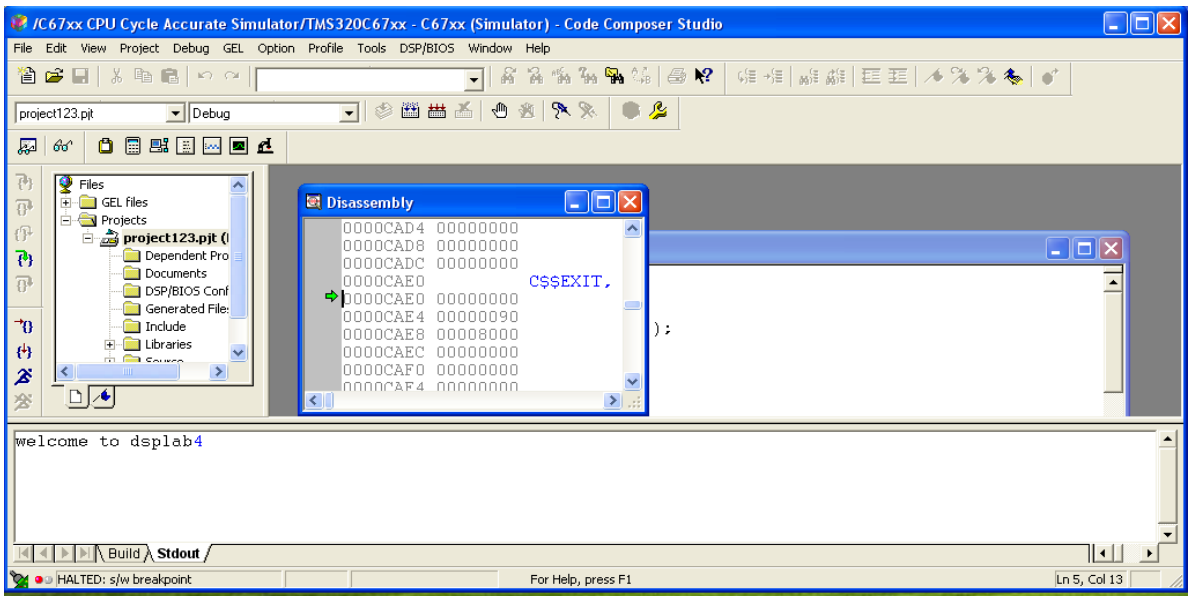
Run: Run the program using the 'Debug-Run' pull down menu or by clicking the shortcut icon on the left side of program window.



Step 5: observe output using graph

Choose View □□□□Graph□□□□□Time/Frequency.

In the Graph Property Dialog, change the Graph Title, Start Address, Acquisition Buffer Size, Display Data Size, DSP Data Type, Auto scale, and Maximum Y-Value properties to the values.



EXP.No:

LINEAR CONVOLUTION

AIM: Verify the linear convolution operation Using DSK Code composer studio

EQUIPMENT:TMS 320C6713 Kit.
RS232 Serial Cable
Power Cord
Operating System – Windows XP
Software – CCStudio_v3.1

THEORY: Convolution is a formal mathematical operation, just as multiplication, addition, and integration. Addition takes two *numbers* and produces a third *numbe*r, while convolution takes two *signals* and produces a third *signa*l. Convolution is used in the mathematics of many fields, such as probability and statistics. In linear systems, convolution is used to describe the relationship between three signals of interest: the input signal, the impulse response, and the output signal.

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

In this equation, x(k), h(n-k) and y(n) represent the input to and output from the system at time n. Here we could see that one of the inputs is shifted in time by a value every time it is multiplied with the other input signal. Linear Convolution is quite often used as a method of implementing filters of various types.

'C' PROGRAM TO IMPLEMENT LINEAR CONVOLUTION :

//Linear convolution program in c language using CCStudio

```
#include<stdio.h>
int x[15],h[15],y[15];
main()
{
int i,j,m,n;
printf("\n enter first sequence length m:");
scanf("%d",&m);
printf("\n enter second sequence length n:");
scanf("%d",&n);
printf("Enter i/p sequence for x(n):\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("Enter i/p sequence for h(n): \n");
for(i=0;i<n; i++)
scanf("%d",&h[i]);
// padding of zeors
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
printf("Output (Linear Convolution) sequence is:\n ");
for(i=0;i<m+n-1;i++)
printf("y[%d]=%d\t",i,y[i]);
}
```

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using 'Project-new ' pull down menu, save it in a separate directory(c:\ti\myprojects) with name lconv.pjt.
- Add the source files conv.asm.
- to the project using 'Project→add files to project' pull down menu.
- Add the linker command file hello.cmd.

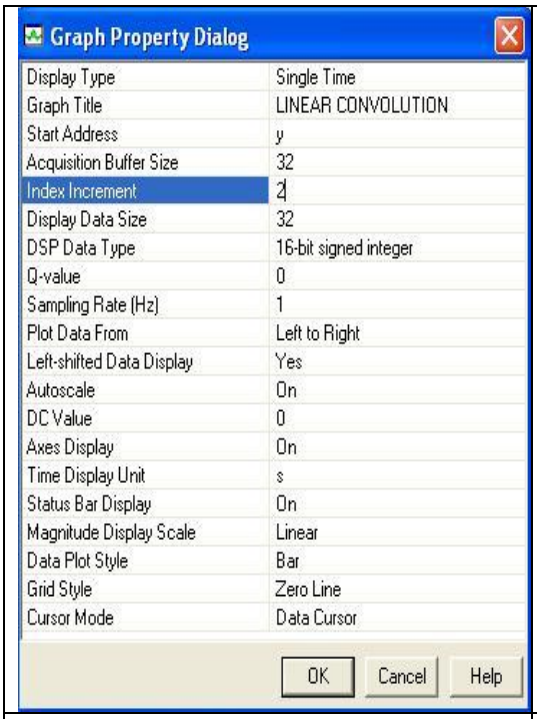
  (Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)
- Add the run time support library file rts6700.lib.

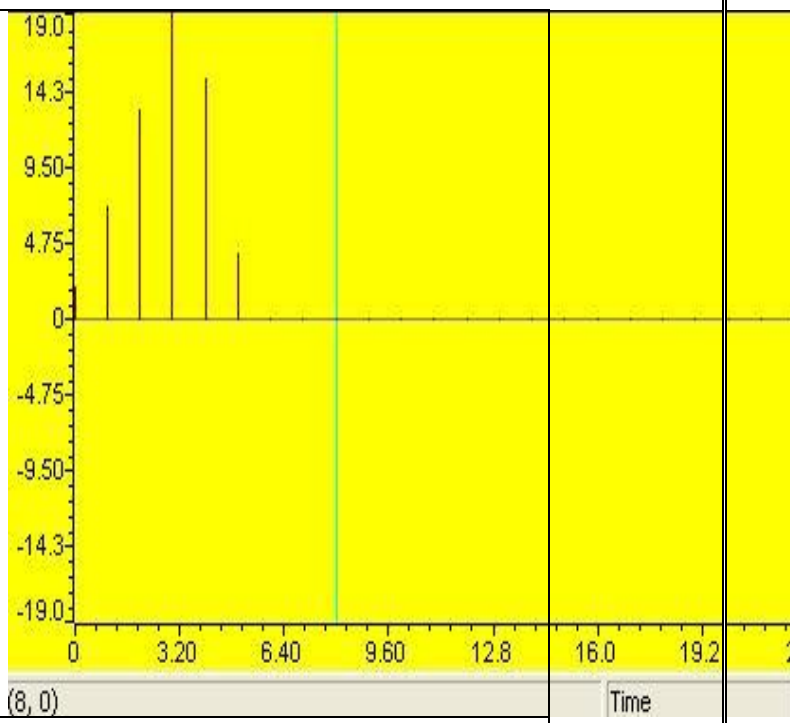
  (Path: c:\ti\c6000\cgtools\lib\rts6700.lib)
- Compile the program using the 'Project-compile' pull down menu or by clicking the shortcut icon on the left side of program window.
- Build the program using the 'Project-Build' pull down menu or by clicking the shortcut icon on the left side of program window.
- Load the program (lconv.out) in program memory of DSP chip using the 'File-load program' pull down menu.
- To View output graphically

  Select view → graph → time and frequency.

OUTPUT FOR LINEAR CONVOLUTION:

enter first sequence length m:4
enter second sequence length n:3
Enter i/p sequence for x(n):
1 2 3 4
Enter i/p sequence for h(n):
2 3 1
Output (Linear Convolution) sequence is:
y[0]=2 y[1]=7 y[2]=13 y[3]=19 y[4]=15 y[5]=4





Graph Property	Graph
LINEAR CONVOLUTION	

EXP.No:

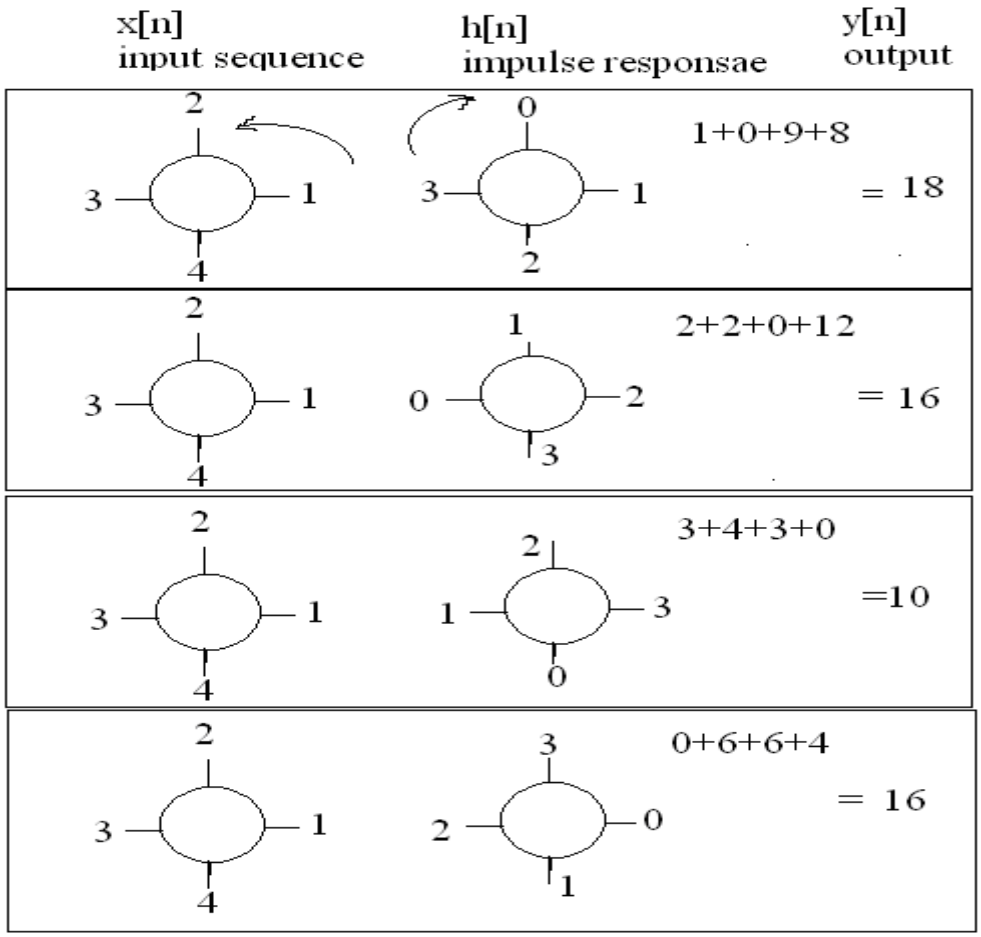
CIRCULAR CONVOLUTION

AIM: To verify the circular convolution operation Using DSK Code composer studio

EQUIPMENT:TMS 320C6713 Kit.
RS232 Serial Cable
Power Cord
Operating System – Windows XP
Software – CCStudio_v3.1

THEORY:

Circular convolution is another way of finding the convolution sum of two input signals. It resembles the linear convolution, except that the sample values of one of the input signals is folded and right shifted before the convolution sum is found. Also note that circular convolution could also be found by taking the DFT of the two input signals and finding the product of the two frequency domain signals. The Inverse DFT of the product would give the output of the signal in the time domain which is the circular convolution output. The two input signals could have been of varying sample lengths. But we take the DFT of higher point, which ever signals levels to.. This process is called circular convolution.



/*program to implement circular convolution */

```
#include<stdio.h>
 int m,n,x[30],h[30],y[30],i,j, 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]);
 }
```

PROCEDURE:

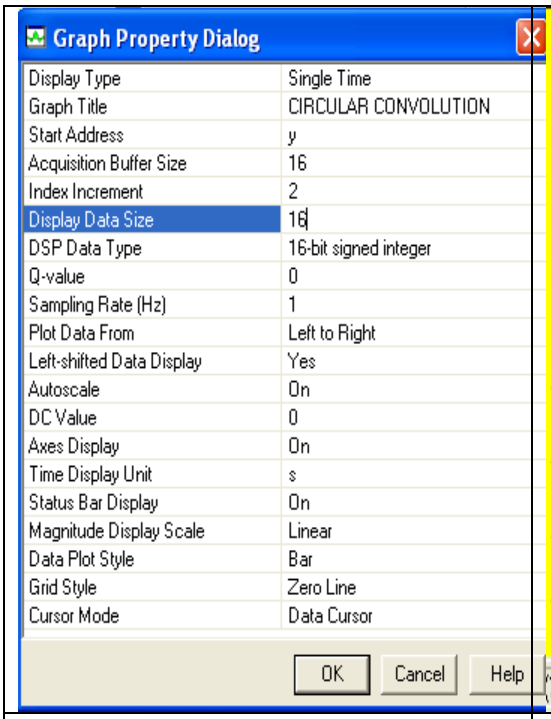
- Open Code Composer Studio; make sure the DSP kit is turned on.
- Start a new project using 'Project-new ' pull down menu, save it in a separate directory(c:\ti\myprojects) with name **cir conv.pjt.**
- Add the source files **Circular Convolution.C.**
- to the project using 'Project→add files to project' pull down menu.
- Add the linker command file **hello.cmd** .

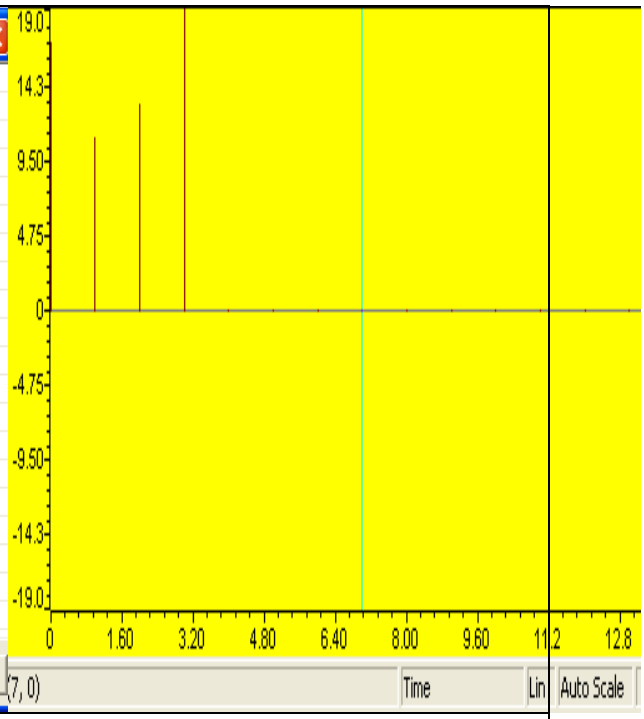
  (Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd)
- Add the run time support library file **rts6700.lib**

  (Path: c:\ti\c6000\cgtools\lib\rts6700.lib)
- Compile the program using the 'Project-compile' pull down menu or by clicking the shortcut icon on the left side of program window.
- Build the program using the 'Project-Build' pull down menu or by clicking the shortcut icon on the left side of program window.
- Load the program(lconv.out) in program memory of DSP chip using the 'File-load program' pull down menu.

OUTPUT FOR CIRCULAR CONVOLUTION:

enter the length of the first sequence
4
enter the length of the second sequence
3
enter the first sequence
1 2 3 4
enter the second sequence
1 2 3
the circular convolution is
18 16 10 16





Graph Property	Graph
CIRCULAR CONVOLUTION	

EXP.No:

N-POINT FAST FOURIER TRANSFORM (FFT)

AIM: To find the DFT of a sequence using FFT algorithm.

EQUIPMENT: TMS 320C6713 Kit.
Oscilloscope & Function Generator
RS232 Serial Cable
Power Cord
Operating System – Windows XP
Software – CCStudio_v3.1

THEORY:

The Fast Fourier Transform is useful to map the time-domain sequence into a continuous function of a frequency variable. The FFT of a sequence {x(n)} of length N is given by a
Complex-valued sequence X (k).

$$X(k) = \sum_{k=0}^{M} x(n) \quad e^{-j2\pi\frac{nk}{n}} ; 0 < k < N-1$$

The above equation is the mathematical representation of the DFT. As the number of computations involved in transforming a N point time domain signal into its corresponding frequency domain signal was found to be N^2 complex multiplications, an alternative algorithm involving lesser number of computations is opted. When the sequence x(n) is divided into 2 sequences and the DFT performed separately, the resulting number of computations would be $N^2/2$. (i.e.)

$$x(k) = \sum_{n=0}^{\frac{N^2}{2}-1} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N^2}{2}-1} x(2n+1) \quad W_N^{(2n+1)k}$$

Consider x(2n) be the even sample sequences and x(2n+1) be the odd sample sequence derived form x(n).

$$\sum_{n=0}^{\frac{N^2}{2}-1} x(2n) \quad W_N^{2nk}$$

$(N/2)^2$ multiplication's $\sum_{n=0}^{\frac{N^2}{2}-1} x(2n+1) \quad W_N^{(2n+1)k}$

an other $(N/2)^2$ multiplication's finally resulting in $(N/2)^2 + (N/2)^2$

$$= \frac{N^2}{4} + \frac{N^2}{4} = \frac{N^2}{2} \text{ Computations}$$

Further solving Eg. (2)

$$x(k) = \sum_{n=0}^{\frac{N^2}{2}-1} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \quad W_N^{(2nk)} W_N^k$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) \quad W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \quad W_N^{(2nk)}$$

Dividing the sequence x(2n) into further 2 odd and even sequences would reduce the computations.

$W_N \rightarrow$ is the twiddle factor

$$= e^{\frac{-j2\pi}{n}}$$

$$W_N^{nk} = e^{\left(\frac{-j2\pi}{n}\right)nk}$$

$$W_N^{\left(K+\frac{N}{2}\right)} = W_N \quad W_N^{\left(K+\frac{N}{2}\right)}$$

$$= e^{\frac{-j2\pi}{n}k} \quad e^{\frac{-j2\pi}{n}\frac{n}{2}}$$

$$= W_N^k \quad e^{\frac{-j2\pi}{n}k}$$

$$= W_N^k \quad (\cos\pi - j\sin\pi)$$

$$= W_N^{\left(K+\frac{N}{2}\right)} = W_N^k(-1)$$

$$= W_N^{\left(K+\frac{N}{2}\right)} = W_N^k$$

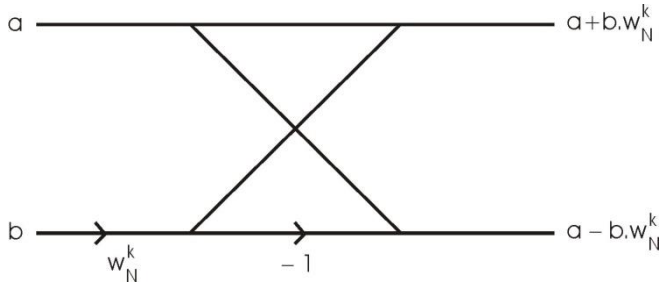
Employing this equation, we deduce

$$x(k) = \sum_{n=0}^{\frac{N^2}{2-1}} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N}{2-1}} \quad x(2n+1) \quad W_N^{(2nk)} \quad (13)$$

$$x(k+\frac{N}{2}) = \sum_{n=0}^{\frac{N}{2-1}} x(2n) \quad W_N^{2nk} - W \sum_{N}^{K} \quad x(2n+1)^{\frac{N}{2-1}} \quad W_N^{(2nk)} \quad (14)$$

The time burden created by this large number of computations limits the usefulness of DFT in many applications. Tremendous efforts devoted to develop more efficient ways of computing DFT resulted in the above explained Fast Fourier Transform algorithm. This mathematical shortcut reduces the number of calculations the DFT requires drastically. The above mentioned radix-2 decimation in time FFT is employed for domain transformation.

Dividing the DFT into smaller DFTs is the basis of the FFT. A radix-2 FFT divides the DFT into two smaller DFTs, each of which is divided into smaller DFTs and so on, resulting in a combination of two-point DFTs. The Decimation -In-Time (DIT) FFT divides the input (time) sequence into two groups, one of even samples and the other of odd samples. N/2 point DFT are performed on the these sub-sequences and their outputs are combined to form the N point DFT.



The above shown mathematical representation forms the basis of N point FFT and is called the **Butterfly Structure**.

PROGRAM
fft256.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:

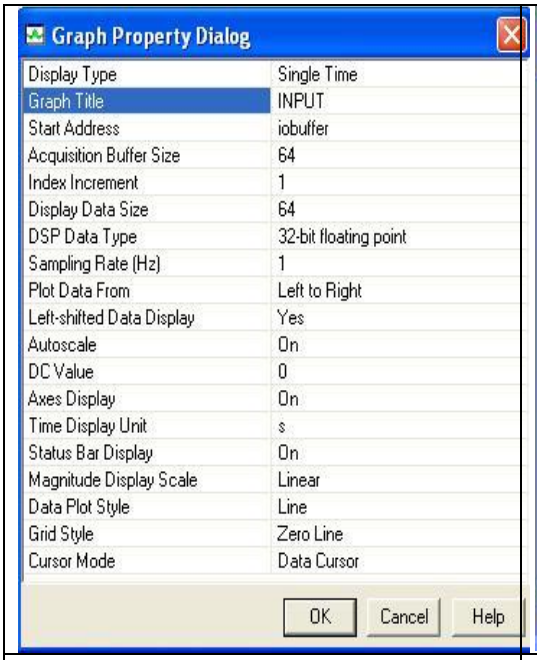
```
#define PTS 64                          //# of points for FFT
typedef struct {float real,imag;} COMPLEX;
extern COMPLEX w[PTS];                  //twiddle constants stored in w

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;
```

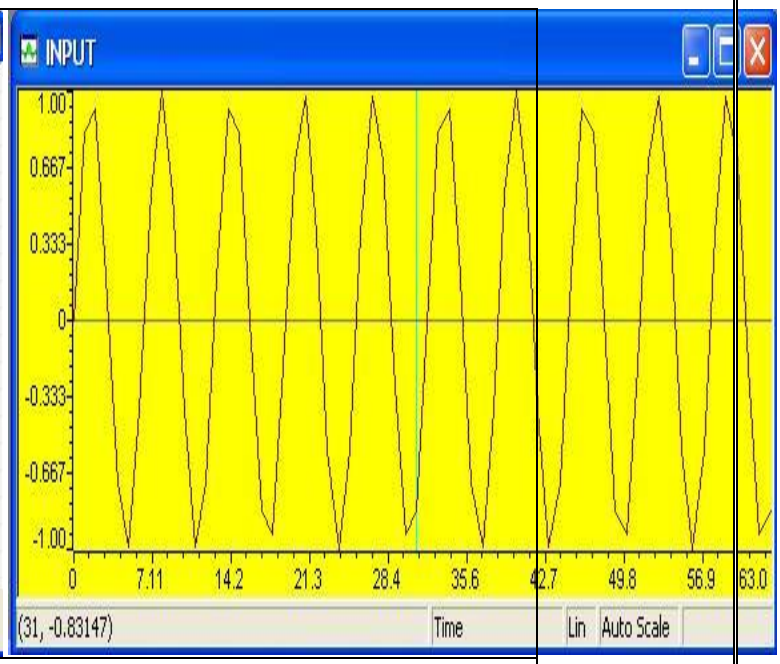
```
      for (i = 1; i < (N-1); i++)     //bit reversal for resequencing data
      {
     k = N/2;
     while (k <= j)
      {
       j = j - k;
       k = k/2;
      }
     j = j + k;
 if (i<j)
      {
       temp1.real = (Y[j]).real;
       temp1.imag = (Y[j]).imag;
       (Y[j]).real = (Y[i]).real;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

PROCEDURE:

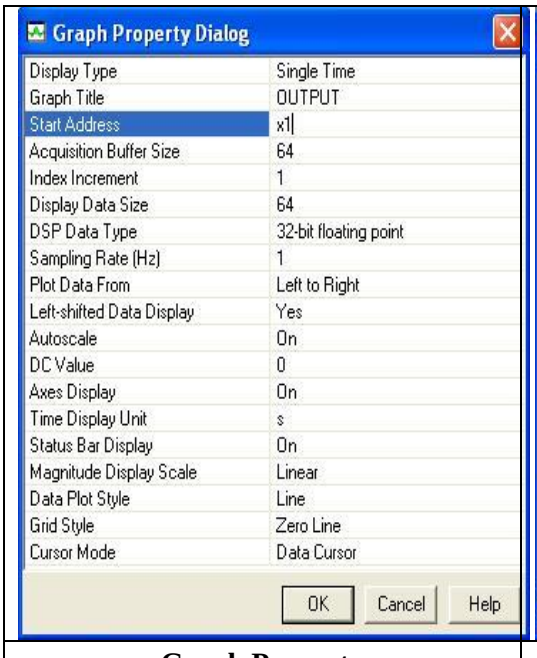
- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using 'Project-new ' pull down menu, save it in a separate directory(c:\ti\myprojects) with name "**FFT.pjt".**
- Add the source files "**fft256.c**" and **"fft.C"** in the project using 'Project→add files to project' pull down menu.
- Add the linker command file "**hello.cmd".**
- Add the rts file **"rts6700.lib" .**
- Compile the program using the 'Project-compile' pull down menu or by clicking the shortcut icon on the left side of program window.
- Load the program in program memory of DSP chip using the 'File-load program' pull down menu.
- Run the program and observe output using graph utility.

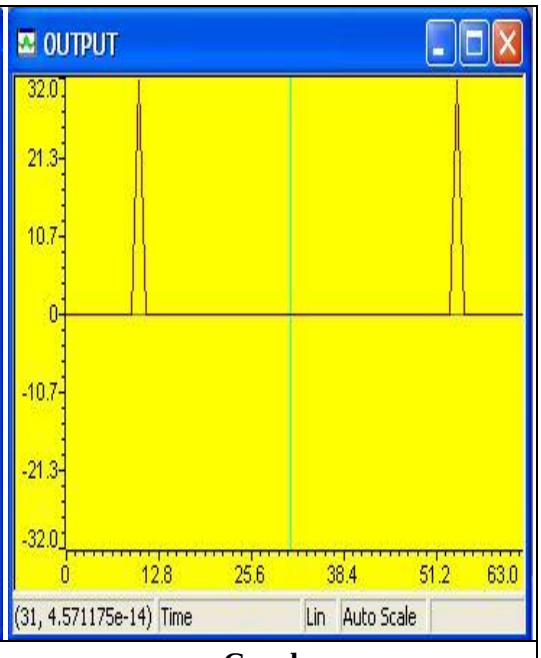
GRAPHS





Graph Property	Graph
FFT INPUT	





Graph Property	Graph
FFT OUTPUT	

EXP.No:

TO COMPUTE POWER DENSITY SPECTRUM OF A SEQUENCE [USING TMS320C6713 DSP PROCESSOR]

AIM: To find the PSD of a sequence.

EQUIPMENT:

TMS 320C6713 Kit.
Oscilloscope & Function Generator
RS232 Serial Cable
Power Cord
Operating System: Windows XP
Software: CCStudio_v3.1

INTRODUCTION: 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 avg. 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.

$$S(\omega) = FT(R(\tau)) = \lim_{N\to\infty} \sum_{\tau=-N+1}^{N-1} \hat{R}(\tau)e^{-j\omega\tau}$$

$$R(\tau) = FT^{-1}(S(\omega)) = \frac{1}{2\pi}\int S(\omega)e^{j\omega\tau}d\omega$$

ALGORITHM TO IMPLEMENT PSD:

- **Step 1 -** Select no. of points for FFT(Eg: 64).
- **Step 2 –** Generate a sine wave of frequency 'f ' (eg: 10 Hz with a sampling rate = No. of Points of FFT(eg. 64)) using **math library function**.

- **Step 3 -** Compute the Auto Correlation of Sine wave .
- **Step4** - Take output of auto correlation, apply FFT algorithm .
- **Step 4 -** Use Graph option to view the PSD.
- **Step 5 -** Repeat **Step-1 to 4** for different no. of points & frequencies.

'C' PROGRAM TO IMPLEMENT PSD:

PSD.c:

```
/*************************************************************
 * FILENAME
 * Non_real_time_PSD.c
 * DESCRIPTION
 * Program to Compute Non real time PSD
 * using the TMS320C6711 DSK.

 *************************************************************
 * DESCRIPTION
 *   Number of points for FFT (PTS)
 *   x --> Sine Wave Co-Efficients
 *   iobuffer --> Out put of Auto Correlation.
 *   x1 --> use in graph window to view PSD
/*=========================================================*/

#include <math.h>
#define PTS 128                         //# 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],x[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
float y[128];
COMPLEX w[PTS];                         //twiddle constants stored in w
COMPLEX samples[PTS];                   //primary working buffer

main()
{

 float j,sum=0.0 ;
 int n,k,i,a;
 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*/
  }

/*****************Input Signal X(n)  *************************/

     for(i=0,j=0;i<PTS;i++)
    { x[i] = sin(2*PI*5*i/PTS);
                    // Signal  x(Fs)=sin(2*pi*f*i/Fs);
      samples[i].real=0.0;
      samples[i].imag=0.0;
    }

/*********************Auto Correlation of X(n)=R(t) ***********/

    for(n=0;n<PTS;n++)
   {
         sum=0;
         for(k=0;k<PTS-n;k++)
         {
           sum=sum+(x[k]*x[n+k]);    // Auto Correlation R(t)
         }
    iobuffer[n] = sum;
   }

/*********************** FFT of R(t) ************************/

  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


/********************** PSD ********************/

  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:

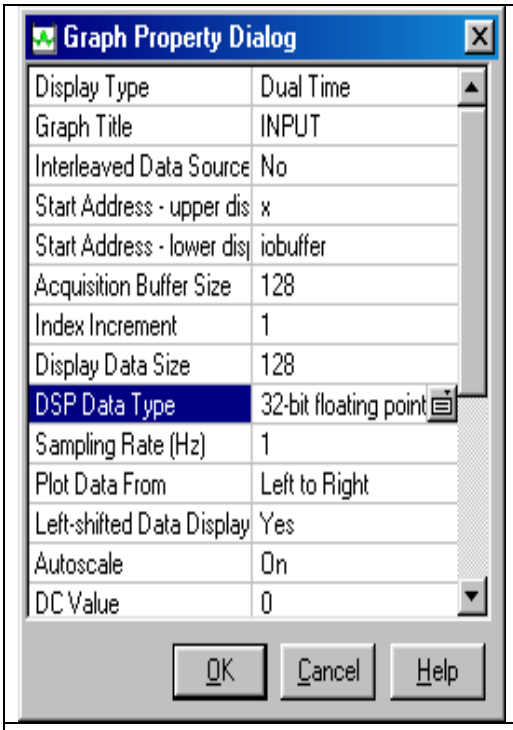
```
#define PTS 128                         //# of points for FFT

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; //indexof 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// 512
 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;
```

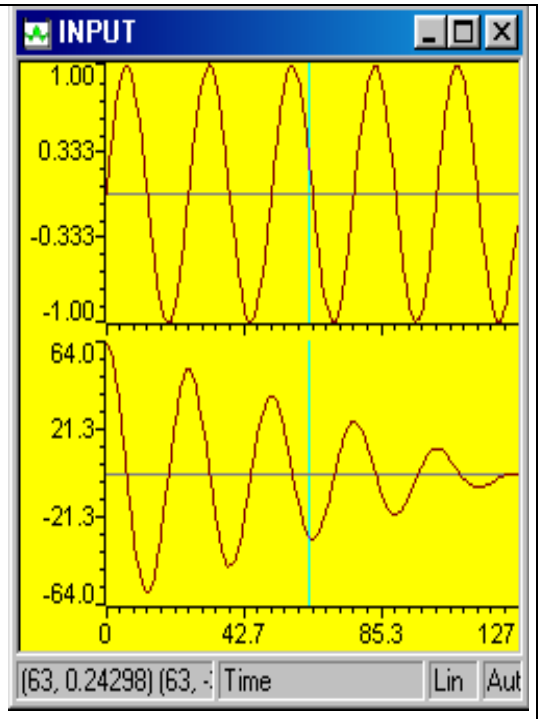
```
    }
    j = 0;
    for (i = 1; i < (N-1); i++)
                                    //bit reversal for resequencing data
    {
     k = N/2;
     while (k <= j)
      {
        j = j - k;
        k = k/2;
      }
        j = j + k;
        if (i<j)
        {
         temp1.real = (Y[j]).real;
 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

PROCEDURE:

- Open Code Composer Studio, make sure the DSP kit is turned on.
- Start a new project using 'Project-new ' pull down menu, save it in a separate directory(c:\ti\myprojects) with name "**PSD.pjt".**
- Add the source files "**PSD.c**" and **"FFT.c"** in the project using 'Project→add files to project' pull down menu.
- Add the linker command file **"hello.cmd"** .
- Add the rts file "**rts6700.lib**" .
- Compile the program using the 'Project-compile' pull down menu or by clicking the shortcut icon on the left side of program window.
- Load the program in program memory of DSP chip using the 'File-load program' pull down menu.
- Run the program and observe output using graph utility.

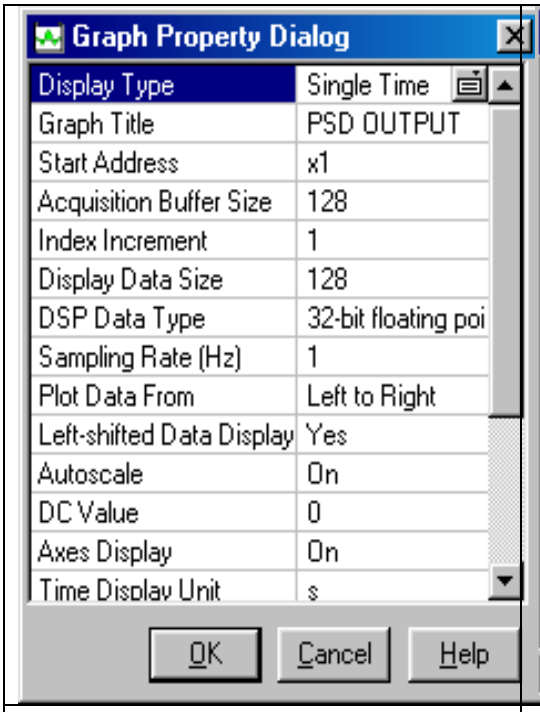
RESULTS

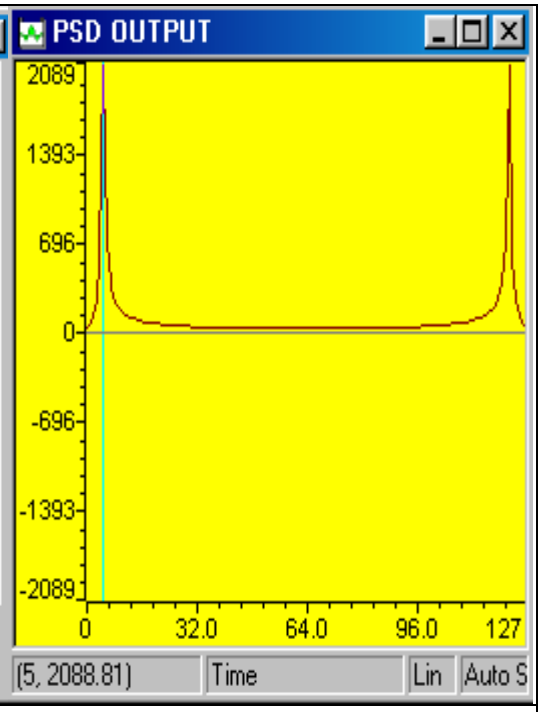




Graph Property	Graph

PSD INPUT





Graph Property	Graph

PSD OUTPUT

EXP.No:

FIR LP/HP FILTER DESIGN USING TMS320C6713 DSP PROCESSOR

AIM: The aim of this laboratory exercise is to design and implement a Digital FIR Filter & observe its frequency response. In this experiment we design a simple FIR filter so as to stop or attenuate required band of frequencies components and pass the frequency components, which are outside the required band.

EQUIPMENT:

TMS 320C6713 Kit.

Oscilloscope & Function Generator

RS232 Serial Cable

Power Cord

Operating System – Windows XP

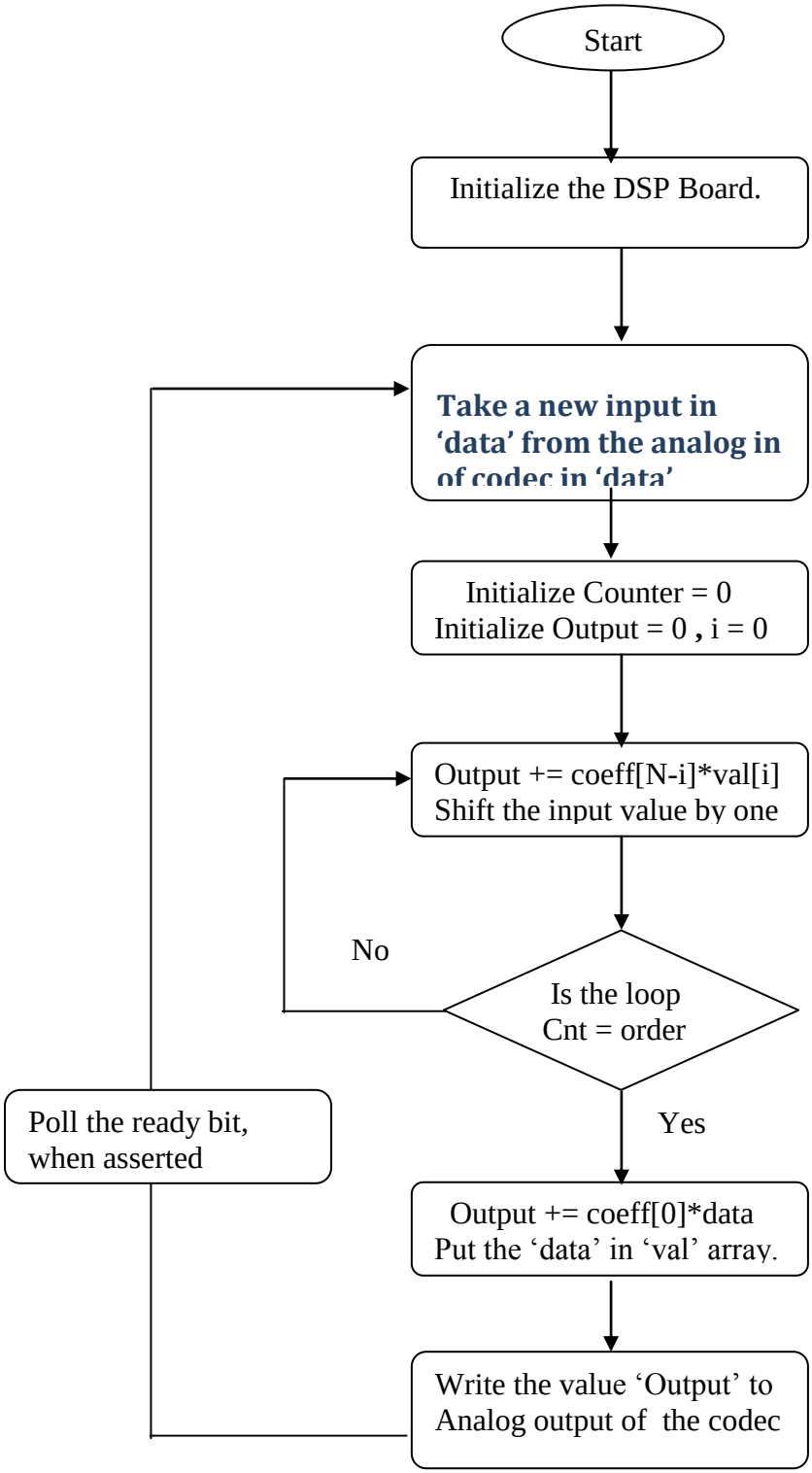
Software – CCStudio_v3.1

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.

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

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).

FLOW CHART TO IMPLEMENT FIR FILTER:



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);

}
```

PROCEDURE:

STEPS TO IMPLEMENT FILTERS

- Do diagnostic test
- Open DSKC6713 studio
- Open new project and give it a name. see whether the project is in C drive or D drive
- Go to file. Say new and select DSPBIOS configuration
- Select DSK6713 or DSK6711 whatever is available (double click)
- Click on configuration window and save
  Path: C:/ CCStudio_v3.1/myprojects/ project name
  Project→Add files to project
  Filter type as: configuration file(*.cdb)
  Configl—open
- Observe under your project .cdb file under DSP and two configure files under generated files
- Close configl window
- Go to file say new/source file/copy paste the filter program onto untitled window
  The filter program is on desktop in programs for 6713
- File/save/your project/file type is c source file and give a name to the file and say save
- After saving the file add to the project/the file name i.e the saved file and say save.
- **Add library file to the project**
  Project →Add files to project
  Path: CCStudio_v3.1\C6000\dsk6713\lib\dsk6713bsl.lib
  Files of type: Object and Library Files (*.o*,*.l*)
- **Add header file to the project**
  Open configlcfg-c.c(under generated files)/
  Copy #include configlcfg.h (close) and paste it to our program
  Remove filter.h file
- Go to project/ add files to project
  Path: CCStudio_v3.1\C6000\dsk6713
  Files of type: All files
  Select include/copy the first two header files(dsk 6713,dsk6713_aic23) and paste in our project i.e
  Path: CCStudio_v3.1/myprojects/ our project name
  Save
- **Building and Running the Program (compile\ Build\ Load Program\ Run)**
  Compile: Compile the program using the 'Project-compile' pull down menu or by clicking the shortcut icon on the left side of program window.
  Build: Build the program using the 'Project-Build' pull down menu or by clicking the shortcut icon on the left side of program window.
- **Load Program:** Load the program in program memory of DSP chip using the 'File-load program' pull down menu.
  Files of type:(*.out*)
  Path: C:\CCStudio_v3.1\ MyProjects\Project Name\ Debug\ Project Name.out
- **Run:** Run the program using the 'Debug-Run' pull down menu or by clicking the shortcut icon on the left side of program window.

EXP. No:

IIR LP/HP FILTER DESIGN USING TMS320C6713 DSP PROCESSOR

AIM: The aim of this laboratory exercise is to design and implement a Digital IIR Filter & observe its frequency response. In this experiment we design a simple IIR filter so as to stop or attenuate required band of frequencies components and pass the frequency components which are outside the required band

EQUIPMENT:

TMS 320C6713 Kit.
Oscilloscope & Function Generator
RS232 Serial Cable
Power Cord
Operating System – Windows XP
Software – CCStudio_v3.1

INTRODUCTION

GENERAL CONSIDERATIONS:

In the design of frequency – selective filters, the desired filter characteristics are specified in the frequency domain in terms of the desired magnitude and phase response of the filter. In the filter design process, we determine the coefficients of a causal IIR filter that closely approximates the desired frequency response specifications.

IMPLEMENTATION OF DISCRETE-TIME SYSTEMS:

Discrete time Linear Time-Invariant (LTI) systems can be described completely by constant coefficient linear difference equations. Representing a system in terms of constant coefficient linear difference equation is it's time domain characterization. In the design of a simple frequency–selective filter, we would take help of some basic implementation methods for realizations of LTI systems described by linear constant coefficient difference equation.

BACKGROUND CONCEPTS:

An Infinite impulse response (IIR) filter possesses an output response to an impulse which is of an infinite duration. The impulse response is "infinite" since there is feedback in the filter, that is if you put in an impulse ,then its output must produced for infinite duration of time. 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:

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

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

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) \; + \; \sum_{k=1}^{N} a_k y(n-k) \qquad (2)$$

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

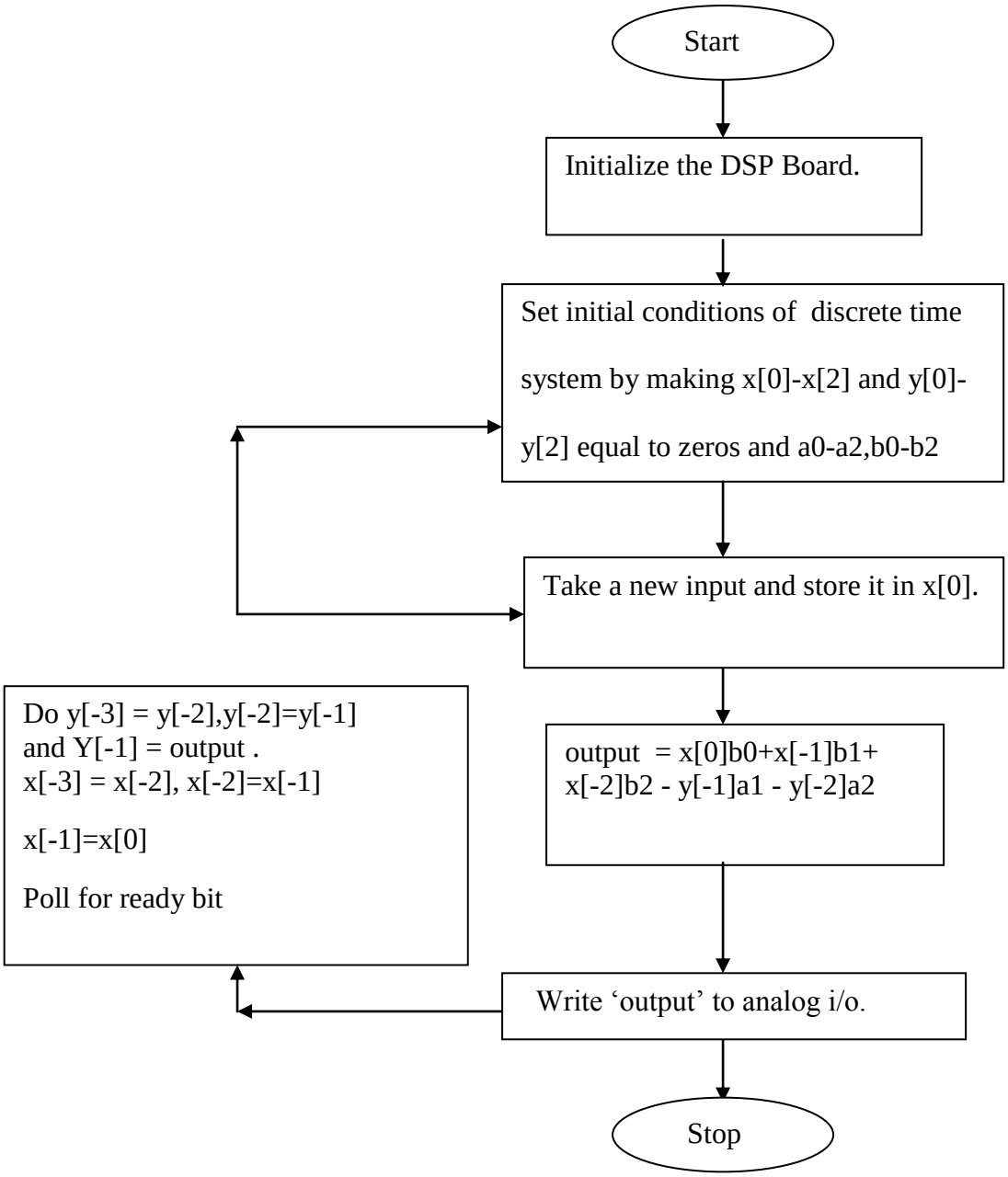
ALGORITHM TO IMPLEMENT:

We need to realize the Butter worth band pass IIR filter by implementing the difference equation $y[n] = b_0x[n] + b_1x[n-1]+b_2x[n-2]-a_1y[n-1]-a_2y[n-2]$ where $b_0 - b_2$, a_0-a_2 are feed forward and feedback word coefficients respectively [Assume 2^{nd} order of filter].These coefficients are calculated using MATLAB.A direct **form I** implementation approach is taken.

- **Step 1 -** Initialize the McBSP, the DSP board and the on board codec.
  "Kindly refer the Topic **Configuration of 6713Codec** using BSL"

- **Step 2 -** Initialize the discrete time system , that is , specify the initial conditions. Generally zero initial conditions are assumed.
- **Step 3 -** Take sampled data from codec while input is fed to DSP kit from the signal generator. Since Codec is stereo , take average of input data read from left and right channel . Store sampled data at a memory location.
- **Step 4 -** Perform filter operation using above said difference equation and store filter Output at a memory location .
- **Step 5 -** Output the value to codec (left channel and right channel) and view the output at Oscilloscope.
- **Step 6 -** Go to step 3.
-

FLOWCHART FOR IIR IMPLEMENTATION:



F.1 : Flowchart for implementing IIR filter.

'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);
}
```

PROCEDURE:

- Switch on the DSP board.
- Open the Code Composer Studio.
- Create a new project

  Project → New (File Name. pjt , Eg: **FIR.pjt**)
- Initialize on board codec.

  "Kindly refer the Topic **Configuration of 6713 Codec** using BSL"
- Add the given above 'C' source file to the current project **(remove codec.c source file from the project if you have already added).**
- Connect the speaker jack to the input of the CRO.
- Build the program.
- Load the generated object file(*.out) on to Target board.
- Run the program using F5.

RESULT:

Observe the waveform that appears on the CRO screen.

EXP.No:

DISCRETE COSINE TRANSFORM

Aim: To observe transformed and inverse transformed sequences and images using DCT

EQUIPMENT:
PC with windows(95/98/xp/NT/2000)
MATLAB software

PROGRAM:

DCT Program for Sequence

```
 function [Xk] =dctu(xn)
% Discrete Cosine Transform
% xn    = Input time domain signal
% Xk    = Output frequency domain signal
 % For ex:
 xn = randi([0 1000],[1 10]);
% Xk = dctu(xn);

N=length(xn);
Cf=[1/sqrt(2),ones(1,N-1)];
for f=0: N-1;
   for x=0: N-1;
      W(f+1, x+1)=cos((2*x+1)*pi*f/(2*N));  %matrix kernel
   end
end
Xk=W*xn';
Xk=(sqrt(2/N)*Xk.*Cf')';    % Final result

disp('DCT Result:'); Xk
subplot(3,2,1)
p=[0:N-1];
stem(p,xn); title('Input Sequence');
subplot(3,2,2)
stem(p,Xk,'fill'); title('DCT of Input sequence');
subplot(3,2,[3,4])
stem(p,abs(Xk),'fill'); title('Magnitude');
phaX = angle(Xk);
subplot(3,2,[5,6])
stem(p,Xk,'fill'); title('Phase plot');
```

RESULT

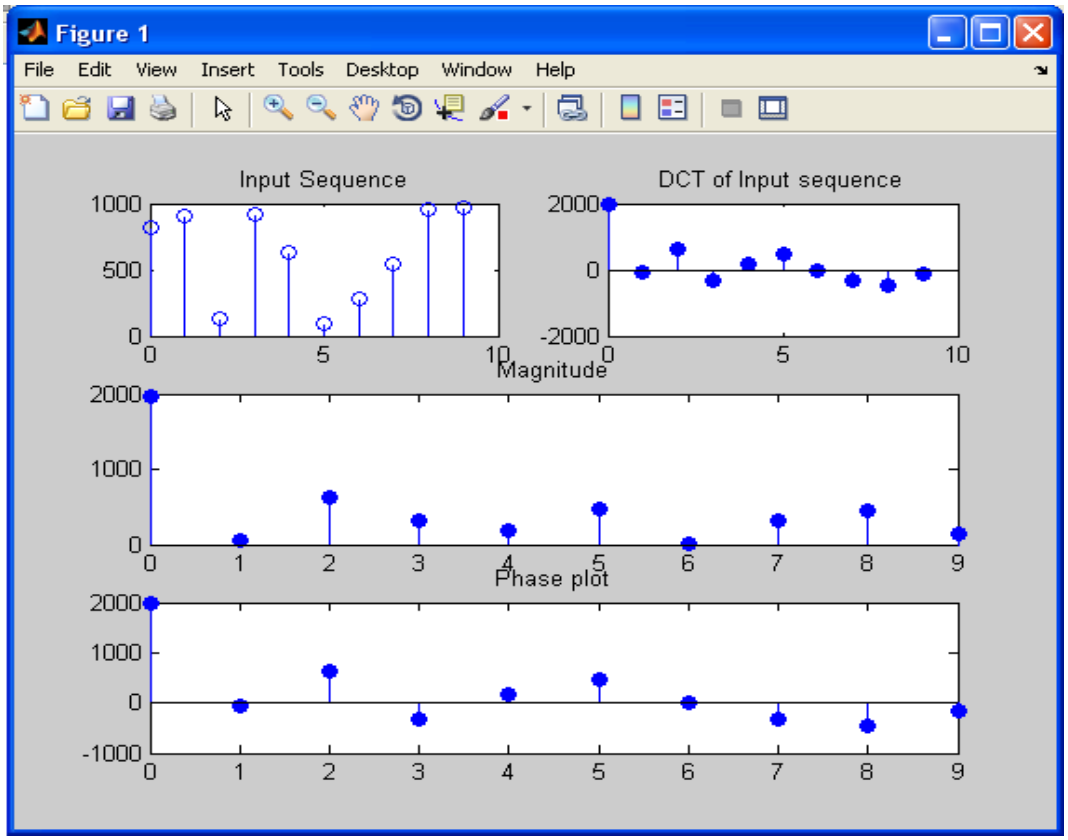


Figure 1
File Edit View Insert Tools Desktop Window Help
Input Sequence
DCT of Input sequence
Magnitude
Phase plot