

DIGITAL SIGNAL PROCESING LABORATORY

(AY-2021-22)

Prepared by

S.Chandramohan,

Assistant Professor- ECE-SCSVMV

DIGITAL SIGNAL PROCESSING **LABORATORY**

LIST OF THE EXPERIMENTS PRESCRIBED BY THE UNIVERSITY

1. Generation of sequences & Correlation.
2. Linear and Circular Convolution.
3. Spectrum Analysis using DFT.
4. FIR Filter Design
5. IIR Filter Design.
6. Multirate Filters.
7. Equalization.

DSP PROCESSOR BASED IMPLEMENTATION USING KIT – TMS 320C6713

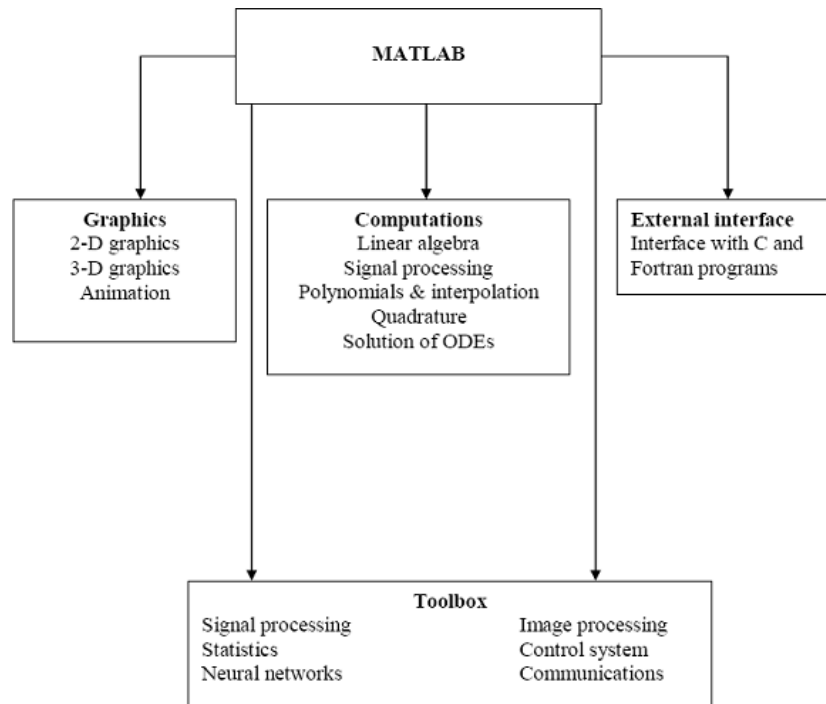
8. Study of architecture of Digital Signal Processor.
9. MAC operation using various addressing modes.
10. Linear convolution.
11. Circular Convolution.
12. FFT Implementation.
13. Waveform Generation
14. IIR & FIR Implementation
15. Finite Word Length Effect

INDEX
LIST OF EXPERIMENTS

S. No	Date	Name of the Experiment	Page no	Marks	Signature
1a		Generation of Continuous Time Signals			
1b		Generation of Discrete Time Signals			
2		Correlation of Sequences			
3		Linear and Circular Convolutions			
4		Spectrum Analysis using DFT			
5a		Design of FIR Filters (rectangular window design)			
5b		Design of FIR Filters (Hanning window design)			
6		Design of IIR Filters			
7		Multirate Filters			
8		Equalization			
9		Study of Architecture of Digital Signal Processor			
10		MAC Operation using Various Addressing Modes			
11		Linear Convolution			
12		Circular Convolution			
13		FFT Implementation			
14		Waveform Generation			
15a		Design of FIR Filters			
15b		Design of IIR Filters			
16		Analysis of Finite Word Length Effect in Fixed Point DSP Systems			

INTRODUCTION

MATLAB is a software package for high performance numerical computation and visualization provides an interactive environment with hundreds of a built in functions for technical computation, graphics and animation. The MATLAB name stands for Matrix laboratory.



At its core, MATLAB is essentially a set (a “toolbox”) of routines (called “m files” or “mex files”) that sit on your computer and a window that allows you to create new variables with names (e.g. voltage and time) and process those variables with any of those routines (e.g. plot voltage against time, find the largest voltage, etc).

It also allows you to put a list of your processing requests together in a file and save that combined list with a name so that you can run all of those commands in the same order at some later time. Furthermore, it allows you to run such lists of commands such that you pass in data and/or get data back out (i.e. the list of commands is like a function in most programming languages). Once you save a function, it becomes part of your toolbox. For those with computer programming backgrounds: Note that MATLAB runs as an interpretive language (like the old BASIC). That is, it does not need to be compiled. It simply reads through each line of the function, executes it, and then goes on to the next line.

DSP Development System

- Testing the software and hardware tools with Code Composer Studio
- Use of the TMS320C6713 DSK
- Programming examples to test the tools

Digital signal processors such as the TMS320C6x (C6x) family of processors are like fast special-purpose microprocessors with a specialized type of architecture and an instruction set appropriate for signal processing. The C6x notation is used to designate a member of Texas Instruments' (TI) TMS320C6000 family of digital signal processors. The architecture of the C6x digital signal processor is very well suited for numerically intensive calculations. Based on a very-long-instruction-word (VLIW) architecture, the C6x is considered to be TI's most powerful processor. Digital signal processors are used for a wide range of applications, from communications and controls to speech and image processing. The general-purpose digital signal processor is dominated by applications in communications (cellular). Applications embedded digital signal processors are dominated by consumer products. They are found in cellular phones, fax/modems, disk drives, radio, printers, hearing aids, MP3 players, high-definition television (HDTV), digital cameras, and so on. These processors have become the products of choice for a number of consumer applications, since they have become very cost-effective. They can handle different tasks, since they can be reprogrammed readily for a different application.

DSP techniques have been very successful because of the development of low-cost software and hardware support. For example, modems and speech recognition can be less expensive using DSP techniques. DSP processors are concerned primarily with real-time signal processing. Real-time processing requires the processing to keep pace with some external event, whereas non-real-time processing has no such timing constraint. The external event to keep pace with is usually the analog input. Whereas analog-based systems with discrete electronic components such as resistors can be more sensitive to temperature changes, DSP-based systems are less affected by environmental conditions.

DSP processors enjoy the advantages of microprocessors. They are easy to use, flexible, and economical. A number of books and articles address the importance of digital signal processors for a number of applications. Various technologies have been used for real-time processing, from fiber optics for very high frequency to DSPs very suitable for the audio-frequency range. Common applications using these processors have been for frequencies from 0 to 96 kHz. Speech can be sampled at 8 kHz (the rate at which samples are acquired), which implies that each value sampled is acquired at a rate of $1/(8 \text{ kHz})$ or 0.125ms. A commonly used sample rate of a compact disk is 44.1 kHz. Analog/digital (A/D)- based boards in the megahertz sampling rate range are currently available.

Ex. No: 1a

Date:

GENERATION OF CONTINUOUS TIME SIGNALS

AIM:

To generate a functional sequence of a signal (Sine, Cosine, triangular, Square, Saw tooth and sinc) using MATLAB function.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2013a

PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (Generation of Continuous Time Signals) _

```
%Program for sine wave
t=0:0.1:10;
y=sin(2*pi*t);
subplot(3,3,1);
plot(t,y,'k');
xlabel('Time');
ylabel('Amplitude');
title('Sine wave');

%Program for cosine wave
t=0:0.1:10;
y=cos(2*pi*t);
subplot(3,3,2);
plot(t,y,'k');
xlabel('Time');
ylabel('Amplitude');
title('Cosine wave');

%Program for square wave
t=0:0.001:10;
y=square(t);
subplot(3,3,3);
```

```

    plot(t,y,'k');
    xlabel('Time');
    ylabel('Amplitude');
    title('Square wave');

%Program for sawtooth wave
    t=0:0.1:10;
    y=sawtooth(t);
    subplot(3,3,4);
    plot(t,y,'k');
    xlabel('Time');
    ylabel('Amplitude');
    title('Sawtooth wave');

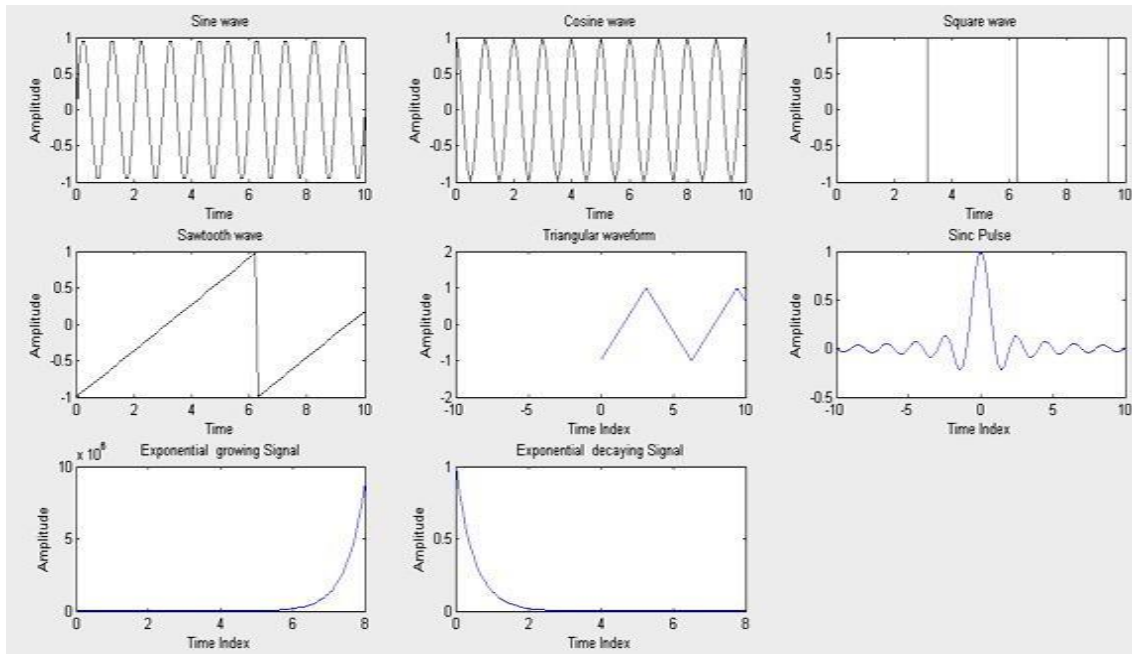
%Program for Triangular wave
    t=0:.0001:20;
    y=sawtooth(t,.5); % sawtooth with 50% duty cycle
    (triangular)
    subplot(3,3,5);
    plot(t,y);
    ylabel ('Amplitude'); xlabel
    ('Time Index');
    title('Triangular waveform');
%Program for Sinc Pulse
    t=-10:.01:10;
    y=sinc(t);
    axis([-10 10 -2 2]);
    subplot(3,3,6)
    plot(t,y)
    ylabel ('Amplitude');
    xlabel ('Time Index');
    title('Sinc Pulse');

% Program for Exponential Growing signal
    t=0:.1:8;
    a=2;
    y=exp(a*t);
    subplot(3,3,7);
    plot(t,y);
    ylabel ('Amplitude'); xlabel ('Time
    Index'); title('Exponential growing
    Signal');

% Program for Exponential Growing signal
    t=0:.1:8;
    a=2;
    y=exp(-a*t);
    subplot(3,3,8);
    plot(t,y);
    ylabel ('Amplitude'); xlabel ('Time
    Index'); title('Exponential decaying
    Signal');

```

OUTPUT: (Generation of Continuous Time Signals)



RESULT:

Thus the MATLAB programs for functional sequence of a signal (Sine, Cosine, triangular, Square, Saw tooth and sinc) using MATLAB function written and the results were plotted.

Ex. No: 1b

GENERATION OF DISCRETE TIME SIGNALS

Date:

AIM:

To generate a discrete time signal sequence (Unit step, Unit ramp, Sine, Cosine, Exponential, Unit impulse) using MATLAB function.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2013a

PROCEDURE:

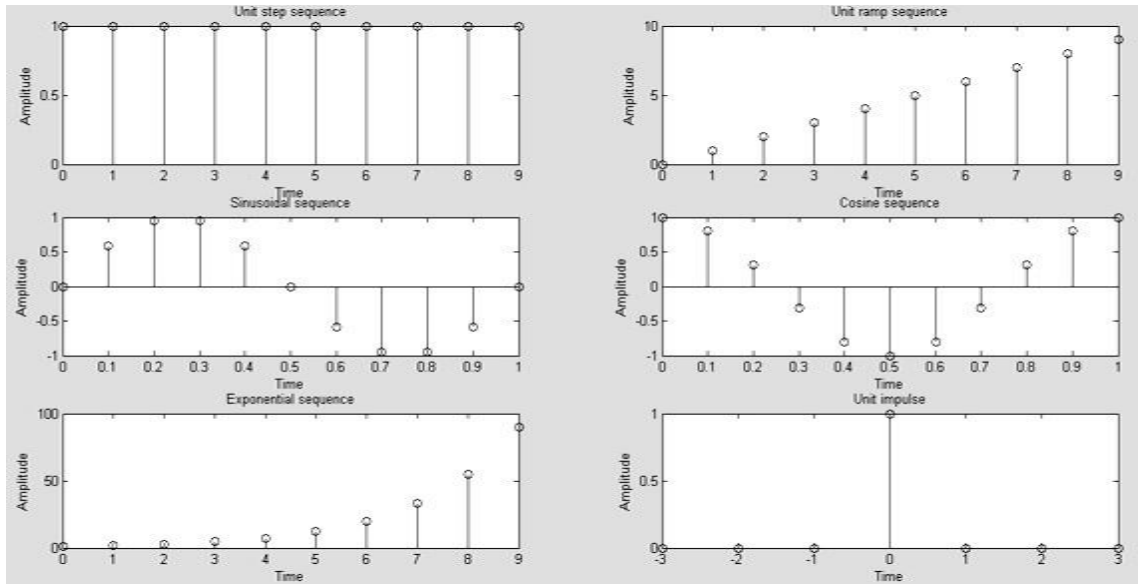
1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (Generation of Discrete Time Signals)

```
%Program for unit step
sequence clc;
N=input('Enter the length of unit step sequence(N)= ');
n=0:1:N-1;
y=ones(1,N);
subplot(3,2,1);
stem(n,y,'k');
xlabel('Time')
ylabel('Amplitude'
)
title('Unit step sequence');
%Program for unit ramp sequence
N1=input('Enter the length of unit ramp sequence(N1)= ');
n1=0:1:N1-1; y1=n1;
subplot(3,2,2);
stem(n1,y1,'k');
xlabel('Time');
ylabel('Amplitude')
;
title('Unit ramp sequence');
%Program for sinusoidal sequence
N2=input('Enter the length of sinusoidal sequence(N2)=
');
n2=0:0.1:N2-1;
y2=sin(2*pi*n2);
subplot(3,2,3);
stem(n2,y2,'k');
xlabel('Time');
ylabel('Amplitude');
title('Sinusoidal
sequence');
%Program for cosine sequence
N3=input('Enter the length of the cosine sequence(N3)=');
n3=0:0.1:N3-1;
y3=cos(2*pi*n3);
subplot(3,2,4);
stem(n3,y3,'k');
xlabel('Time');
ylabel('Amplitude');
title('Cosine
sequence');
%Program for exponential sequence
N4=input('Enter the length of the
exponential sequence(N4)= ');
n4=0:1:N4-1;
a=input('Enter the value of the exponential sequence(a)=
'); y4=exp(a*n4);
subplot(3,2,5);
stem(n4,y4,'k');
xlabel('Time');
ylabel('Amplitude')
;
title('Exponential sequence');
%Program for unit impulse
n=-3:1:3;
y=[zeros(1,3),ones(1,1),zeros(1,3)]
;
```

```
subplot(3,2,6);  
stem(n,y,'k');  
xlabel('Time');  
ylabel('Amplitude');  
title('Unit impulse');
```

OUTPUT: (Generation of Discrete Time Signals)



RESULT:

Thus the MATLAB programs for discrete time signal sequence (Unit step, Unit ramp, Sine, Cosine, Exponential, Unit impulse) using MATLAB function written and the results were plotted.

Ex. No: 2

Date:

CORRELATION OF SEQUENCES

AIM:

To write MATLAB programs for auto correlation and cross correlation.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2013a

PROCEDURE:

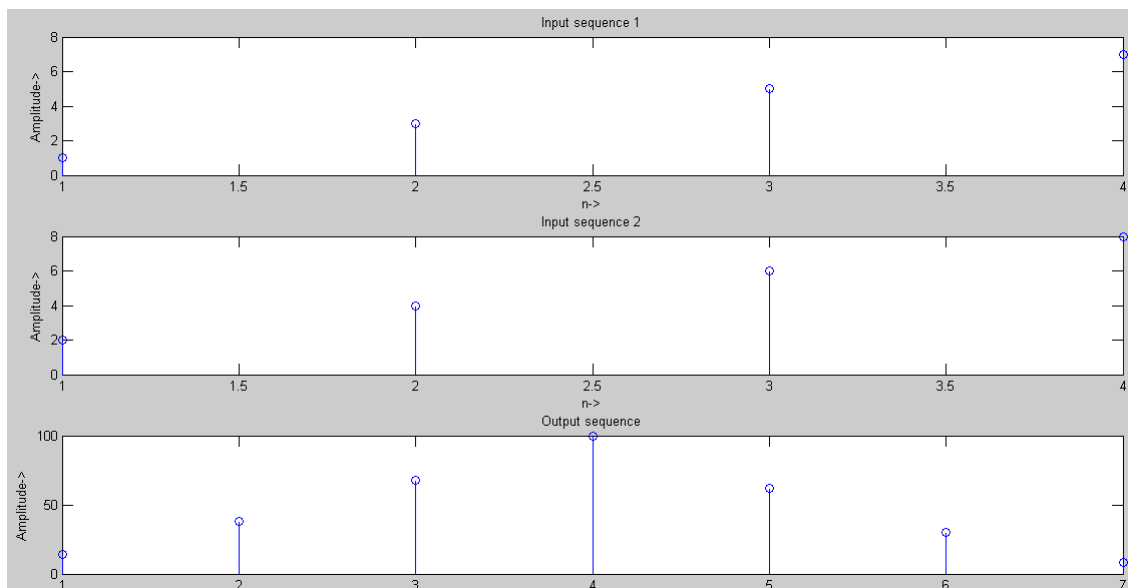
1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (Cross-Correlation of the Sequences)

```
clc;
clear all;
close all;
x=input('Enter the sequence 1: ');
h=input('Enter the sequence 2: ');
y=xcorr(x,h);
figure;
subplot(3,1,1);
stem(x);
xlabel('n->');
ylabel('Amplitude->');
title('Input sequence 1');
subplot(3,1,2);
stem(fliplr(y));
stem(h);
xlabel('n->');
ylabel('Amplitude->');
title('Input sequence 2');
subplot(3,1,3);
stem(fliplr(y));
xlabel('n->');
ylabel('Amplitude->');
title('Output sequence');
disp('The resultant is');
fliplr(y);
```

OUTPUT: (Cross-Correlation of the Sequences)

```
Enter the sequence 1: [1 3 5 7]
Enter the sequence 2: [2 4 6 8]
```

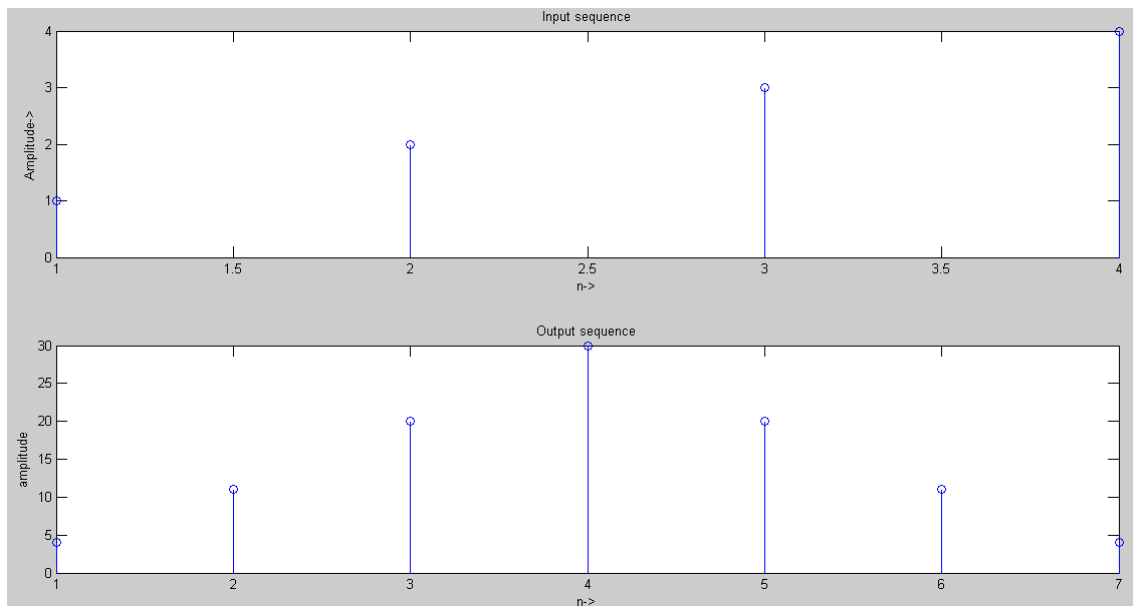


PROGRAM: (Auto Correlation Function)

```
clc;
close all;
clear all;
x=input('Enter the sequence 1: ');
y=xcorr(x,x);
figure;
subplot(2,1,1)
; stem(x);
ylabel('Amplitude->');
xlabel('n->');
title('Input sequence');
subplot(2,1,2);
stem(fliplr(y));
ylabel('amplitude');
xlabel('n->');
title('Output sequence');
disp('the resultant is
'); fliplr(y);
```

OUTPUT: (Auto Correlation Function)

Enter the sequence [1 2 3 4]



RESULT:

Thus the MATLAB programs for auto correlation and cross correlation written and the results were plotted.

Ex.No:3

Date:

Linear & Circular Convolution

AIM:

To write MATLAB programs to find out the linear convolution and Circular convolution of two sequences.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2013a

PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

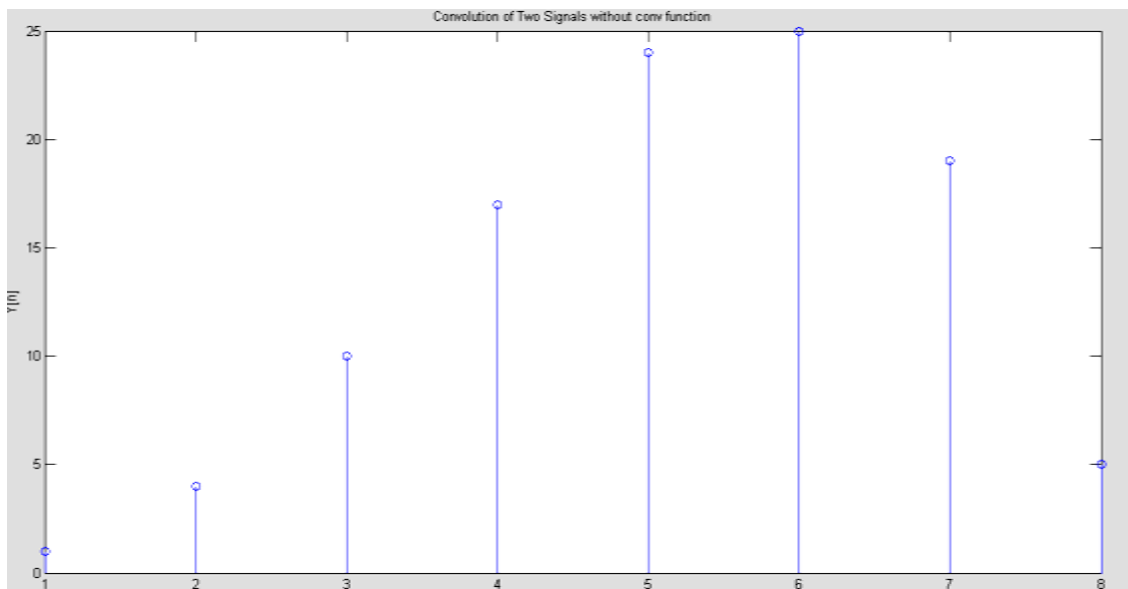
PROGRAM: (Linear Convolution)

```
% linear convolution
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:i
        Y(i)=Y(i)+X(j)*H(i-j+1);
    end
end
Y
stem(Y);
ylabel('Y[n]');
xlabel(' ---->n');
title('Convolution of Two Signals without conv
function');
```

INPUT:

```
Enter x:  [1 2 3 4 5]
x = 1 2 3 4 5
Enter h:  [1 2 3 1]
h = 1 2 3 1
Y =      1 4 10 17 24 25 19 5
```

OUTPUT: (Linear Convolution)



PROGRAM: (Circular Convolution)

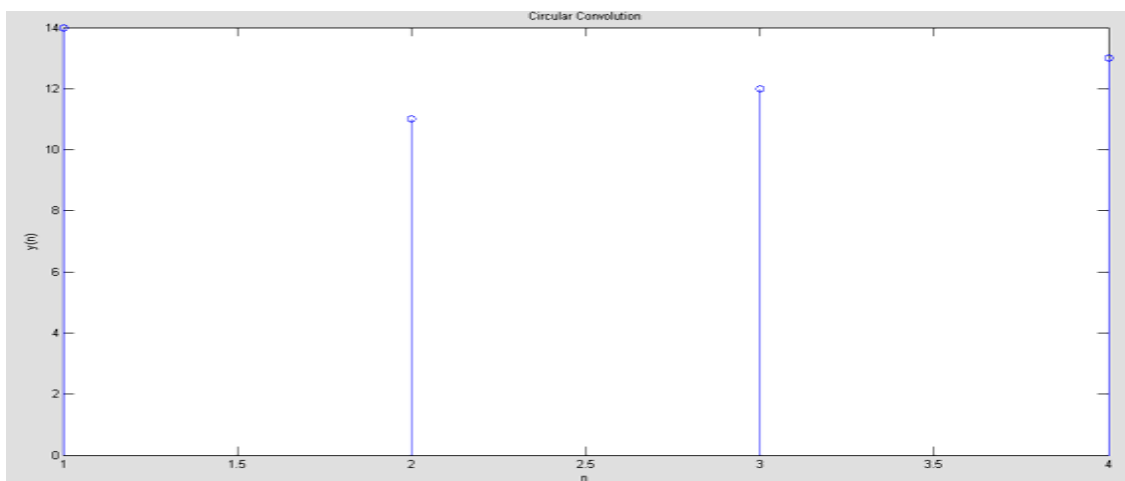
```
clc; clear;
a = input('enter the sequence x(n) = ');
b = input('enter the sequence h(n) = ');
n1=length(a);
n2=length(b);
N=max(n1,n2);
x = [a zeros(1,(N-n1))];
for i = 1:N
k = i;
for j = 1:n2
H(i,j)=x(k)* b(j);
k = k-1;
if (k == 0)
k = N;
end
end
end
y=zeros(1,N);
M=H';
for j = 1:N
for i = 1:n2
y(j)=M(i,j)+y(j);
end end
disp('The output sequence is y(n)= ');
disp(y);
stem(y);
title('Circular Convolution');
xlabel('n');
ylabel('y(n)');
```

OUTPUT:(Circular Convolution)

Enter the sequence x(n) = [1 2 3 4]

Enter the sequence h(n) = [1 2 1 1]

The output sequence is y(n)= 14 11 12 13



RESULT:

Thus the MATLAB programs for linear convolution and circular convolution written and the results were plotted.

Ex. No: 4

Date:

SPECTRUM ANALYSIS USING DFT

AIM:

To write MATLAB program for spectrum analyzing signal using DFT.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2014a

PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

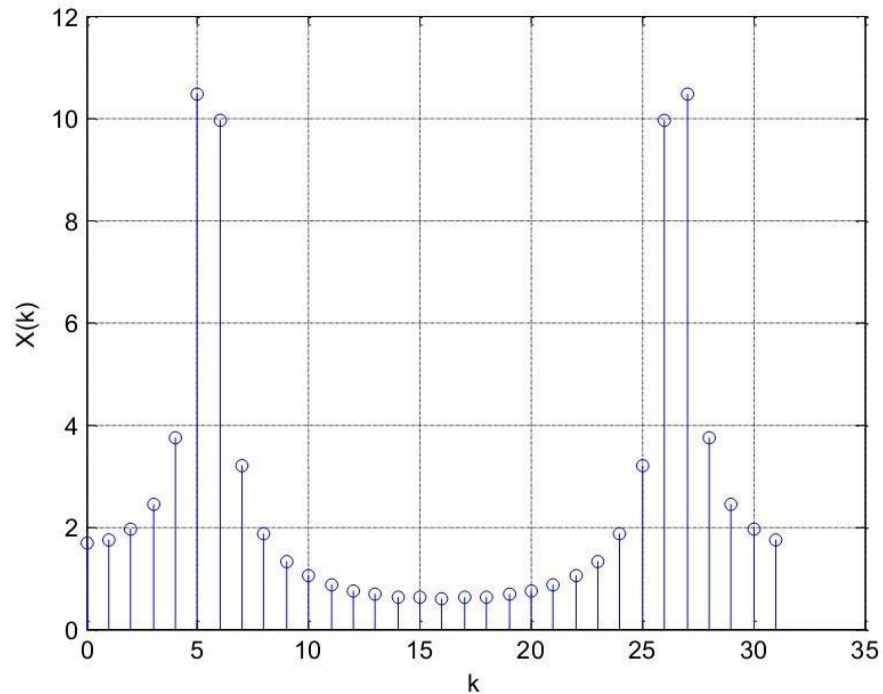
PROGRAM: (Spectrum Analysis Using DFT)

```
N=input('type length of DFT= ');
T=input('type sampling period= ');
freq=input('type the sinusoidal freq= ');
k=0:N-1;
f=sin(2*pi*freq*1/T*k);
F=fft(f);
stem(k,abs(F));
grid on;
xlabel('k');
ylabel('X(k)');
```

INPUT:

```
type length of DFT=32
type sampling period=64
type the sinusoidal freq=11
```

OUTPUT: (Spectrum Analysis Using DFT)



RESULT:

Thus the Spectrum Analysis of the signal using DFT is obtained using MATLAB.

Ex. No: 5a

Date:

**DESIGN OF FIR FILTERS
(RECTANGULAR WINDOW DESIGN)**

AIM:

To write a program to design the FIR low pass, High pass, Band pass and Band stop filters using RECTANGULAR window and find out the response of the filter by using MATLAB.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2013a

PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (Rectangular Window)

```
clear all;
rp=input('Enter the PB ripple rp =');
rs=input('Enter the SB ripple rs =');
fp=input('Enter the PB ripple fp =');
fs=input('Enter the SB ripple fs =');
f=input('Enter the sampling frequency f =');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
den=14.6*(fs-fp)/f;
n=ceil(num/den);
n1=n+1;
if(rem(n,2)~=0)
    n=n1;
    n=n-1;
end;
y=boxcar(n1);
%LPF
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
xlabel('Normalized frequency----->');
ylabel('Gain in db----- --. ');
title('MAGNITUDE RESPONSE OF LPF');
%HPF
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
xlabel('Normalized frequency----->');
ylabel('Gain in db----- --. ');
title('MAGNITUDE RESPONSE OF HPF');
%BPF
wn=[wp ws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
xlabel('Normalized frequency----->');
ylabel('Gain in db----- --. ');
title('MAGNITUDE RESPONSE OF BPF');
%BSF
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
```

```

xlabel('Normalized frequency----->');
ylabel('Gain in db----- --. ');
title('MAGNITUDE RESPONSE OF BSF');

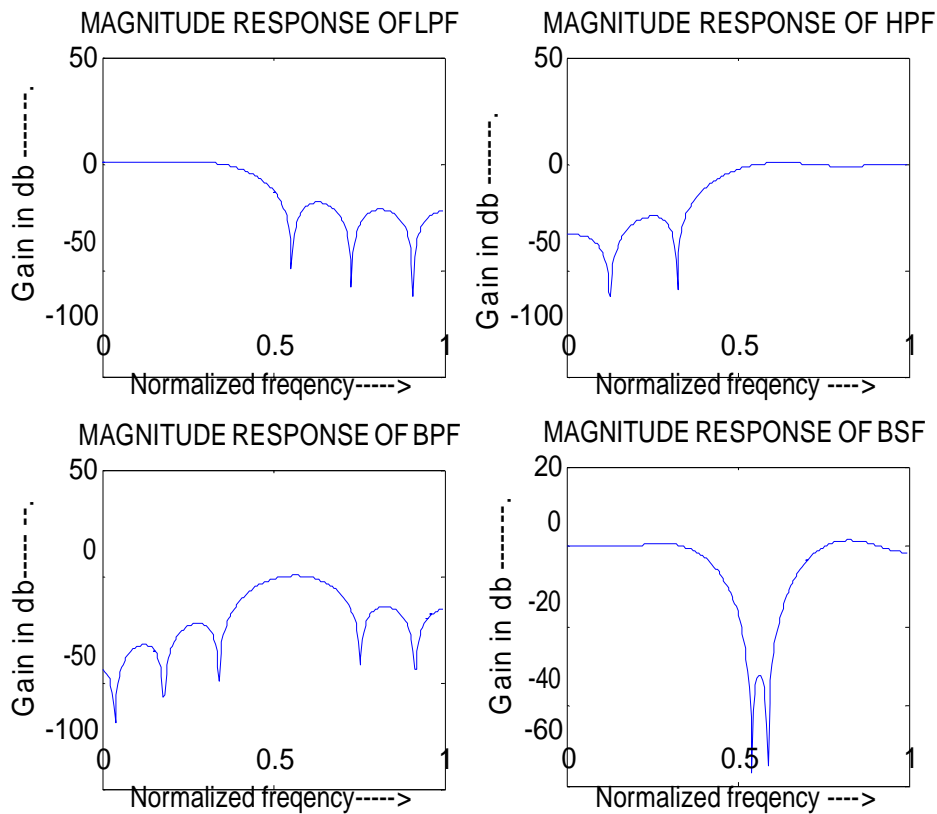
```

OUTPUT: (Rectangular Window)

```

Enter the PB ripple  $r_p = .03$ 
Enter the SB ripple  $r_s = .05$ 
Enter the PB ripple  $f_p = 2000$ 
Enter the SB ripple  $f_s = 3000$ 
Enter the sampling frequency  $f = 9000$ 

```



RESULT:

Thus the program to design FIR low pass, high pass, band pass and band stop Filters using RECTANGULAR Window was written and response of the filter using MATLAB was executed.

Ex. No: 5b

Date:

**DESIGN OF FIR FILTERS
(HANNING WINDOW DESIGN)**

AIM:

To write a program to design the FIR low pass, High pass, Band pass and Band stop filters using HANNING window and find out the response of the filter by using MATLAB.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2013a

PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (Hanning Window)

```
clear all;
rp=input('Enter the PB ripple rp =');
rs=input('Enter the SB ripple rs =');
fp=input('Enter the PB ripple fp =');
fs=input('Enter the SB ripple fs =');
f=input('Enter the sampling frequency f =');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
den=14.6*(fs-fp)/f;
n=ceil(num/den);
n1=n+1;
if(rem(n,2)~=0)
    n=n1;
    n=n-1;
end;
y=hanning(n1);
%LPF b=fir1(n,wp,y);
[h,O]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(O/pi,m);
xlabel('Normalized frequency----->');
ylabel('Gain in db----- --. ');
title('MAGNITUDE RESPONSE OF LPF');

%HPF
b=fir1(n,wp,'high',y);
[h,O]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(O/pi,m);
xlabel('Normalized frequency----->');
ylabel('Gain in db----- --. ');
title('MAGNITUDE RESPONSE OF HPF');

%BPF
wn=[wp ws];
b=fir1(n,wn,y);
[h,O]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(O/pi,m);
xlabel('Normalized frequency----->');
ylabel('Gain in db----- --. ');
title('MAGNITUDE RESPONSE OF BPF');

%BSF
b=fir1(n,wn,'stop',y);
[h,O]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(O/pi,m);
```



```

xlabel('Normalized frequency----->');
ylabel('Gain in db-----');
title('MAGNITUDE RESPONSE OF BSF');

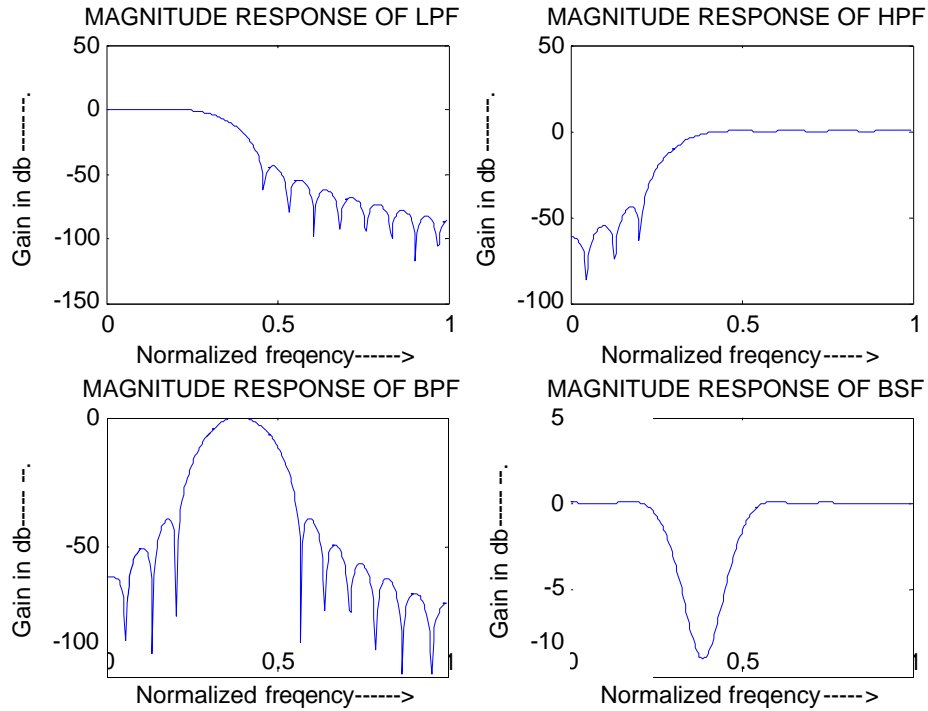
```

OUTPUT: (Hanning Window)

```

Enter the PB ripple rp =.03
Enter the SB ripple rs =.02
Enter the PB ripple fp =1500
Enter the SB ripple fs =2000
Enter the sampling frequency f =9000

```



RESULT:

Thus the program to design FIR low pass, high pass, band pass and band stop Filters using **HANNING** Window was written and response of the filter using **MATLAB** was executed.

Ex. No: 6

Date:

DESIGN OF IIR FILTERS

AIM:

To write a program to design the IIR Filter using Impulse Invariant Transformation method and find out the Magnitude response and Pole Zero Plot by using MATLAB.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2014a

PROCEDURE:

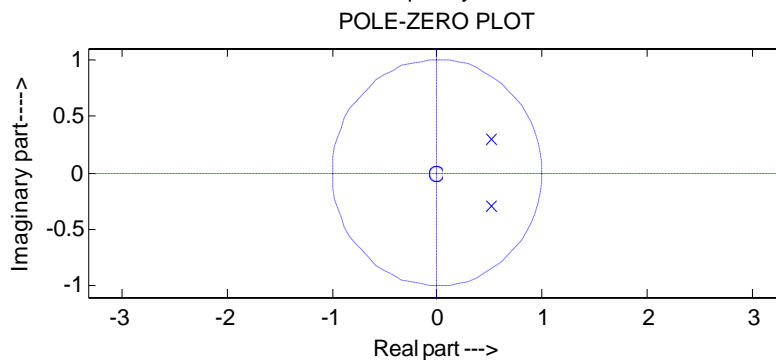
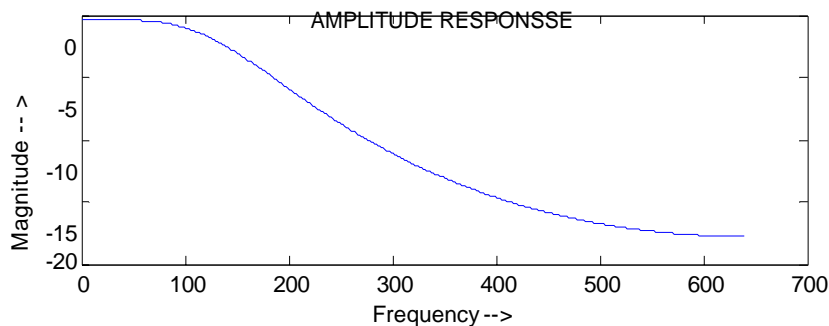
1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (IIR Butterworth Filter using Impulse Method)

```
N=input('ENTER THE FILTER ORDER N = ');
fs=input('ENTER THE SAMPLING FREQUENCY fs = ');
fc=input('ENTER THE CUT-OFF FREQUENCY fc = ');
wc=2*pi*fc;
[na,da]=butter(N,wc,'s');
[n,d]=impinvar(na,da,fs);
[h,f]=freqz(n,d,512,fs);
gain=20*log10(abs(h));
subplot(2,1,1);
plot(f,gain);
xlabel('Frequency --- >');
ylabel('Magnitude --- >');
title('AMPLITUDE RESPONSSSE');
subplot(2,1,2);
zplane(n,d);
z=roots(n); p=roots(d);
xlabel('Real part --- >');
ylabel('Imaginary part --- >');
title('POLE-ZERO PLOT');
```

OUTPUT: (IIR Butterworth Filter using Impulse Method)

```
ENTER THE FILTER ORDER N = 2
ENTER THE SAMPLING FREQUENCY fs = 1280
ENTER THE CUT-OFF FREQUENCY fc = 150
```



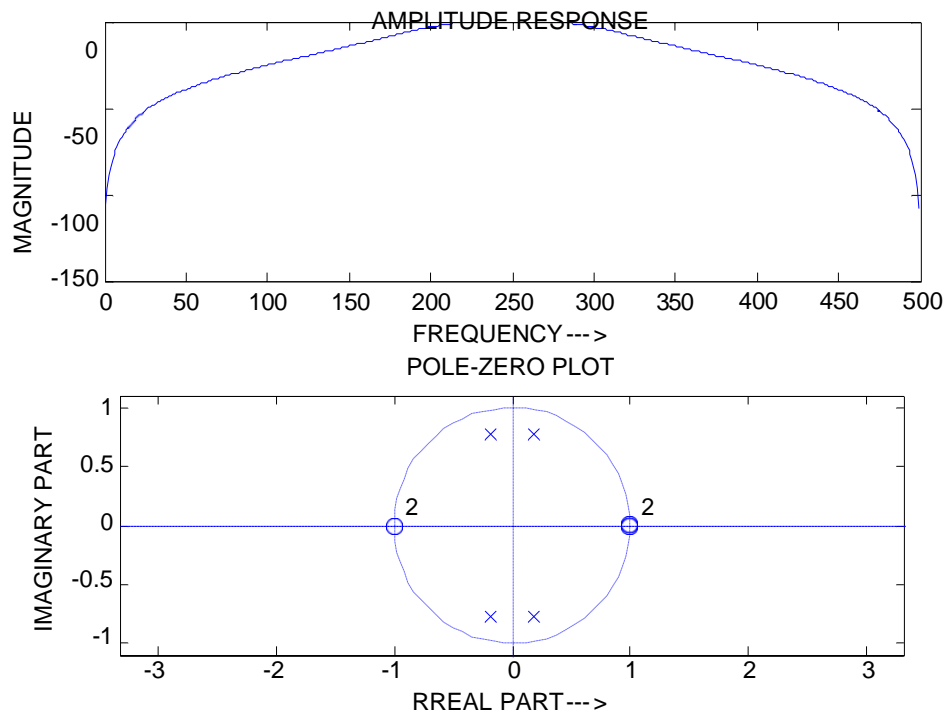
PROGRAM: (IIR Butterworth Using Bilinear Transformation)

```
wp=input('ENTER THE PASSBAND EDGE FREQUENCIES wp= ');
ws=input('ENTER THE STOPBAND EDGE FREQUENCIES ws= ');
rp=input('ENTER THE PASSBAND RIPPLE rp= ');
rs=input('ENTER THE STOPBAND RIPPLE rs= ');
fs=input('ENTER THE SAMPLING FREQUENCY fs= ');
wpn=wp/(fs/2);
wsn=ws/(fs/2);
[N,fc]=buttord(wpn,wsn,rp,rs);
disp('ORDER OF THE FILTER');
disp(N);
[n,d]=butter(N,wpn);
[h,f]=freqz(n,d,512,fs);
gain=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);
plot(f,gain);
xlabel('FREQUENCY --- >');
ylabel('MAGNITUDE');
title('AMPLITUDE RESPONSE');
subplot(2,1,2);
zplane(n,d);
z=roots(n);
p=roots(d);
xlabel('RREAL PART --- >');
ylabel('IMAGINARY PART');
title('POLE-ZERO PLOT');
```

INPUT: (IIR Butterworth Using Bilinear Transformation)

```
Enter the passband edge frequencies wp= [200 300]
Enter the stopband edge frequencies ws= [50 450]
Enter the passband ripple rp= 3
Enter the stopband ripple rs= 20
Enter the sampling frequency fs= 1000
Order of the filter      2
```

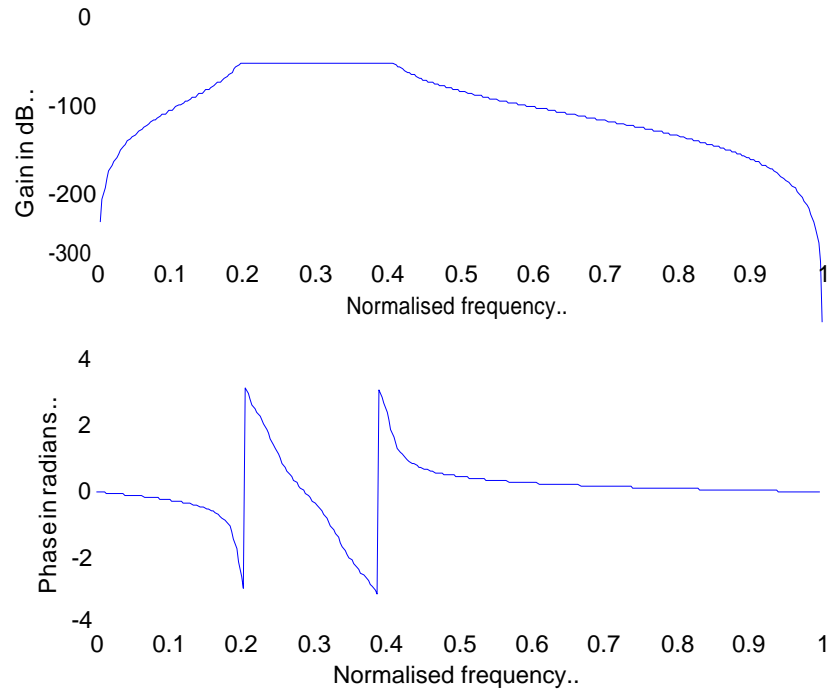
OUTPUT: (IIR Butterworth Using Bilinear Transformation)



PROGRAM: (Chebyshev Type 1 Band pass Filter)

```
clear all;
alphap=2; %pass band attenuation in dB
alphas=20; %stop band attenuation in dB
wp=[.2*pi,.4*pi];
ws=[.1*pi,.5*pi];
%To find cutoff frequency and order of the filter
[n,wn]=buttord(wp/pi,ws/pi,alphap,alphas);
%system function of the filter
[b,a]=cheby1(n,alphap,wn);
w=0:.01:pi;
[h,ph]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);
plot(ph/pi,m);
grid;
ylabel('Gain in dB..');
xlabel('Normalised frequency..');
subplot(2,1,2);
plot(ph/pi,an);
grid;
ylabel('Phase in radians..');
xlabel('Normalised frequency..');
```

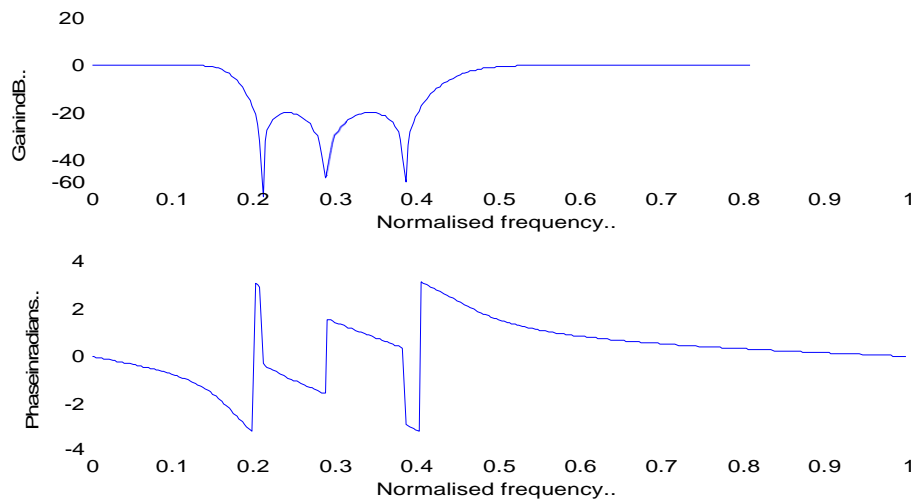
OUTPUT: (Chebyshev Type 1 Band pass Filter)



PROGRAM: (Chebyshev II Band Reject Filter)

```
clear all;
alphap=2; %pass band attenuation in dB
alphas=20; %stop band attenuation in dB
ws=[.2*pi,.4*pi];
wp=[.1*pi,.5*pi];
%To find cutoff frequency and order of the filter
[n,wn]=cheb2ord(wp/pi,ws/pi,alphap,alphas);
%system function of the filter
[b,a]=cheby2(n,alphas,wn,'stop');
w=0:.01:pi;
[h,ph]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);
plot(ph/pi,m);
grid;
ylabel('Gain in dB..');
xlabel('Normalised frequency..');
subplot(2,1,2);
plot(ph/pi,an);
grid;
ylabel('Phase in radians..');
xlabel('Normalised frequency..');
```

OUTPUT: (Chebyshev II Band Reject Filter)



RESULT:

Thus the program to design IIR BUTTERWORTH Low Pass Filter using Impulse Invariant Transformation method and find out the Magnitude response and Pole Zero Plot by using MATLAB was executed.

Ex. No: 7

Date:

MULTIRATE FILTERS

AIM:

To design linear-phase FIR L^{th} -band filters of the length $N = 31$, with $L = 3$ and with the roll-off factors: $\rho = 0.2, 0.4$, and 0.6 . Plot the impulse responses and the magnitude responses for all designs.

APPARATUS REQUIRED:

HARDWARE : Personal Computer

SOFTWARE : MATLAB R2014a

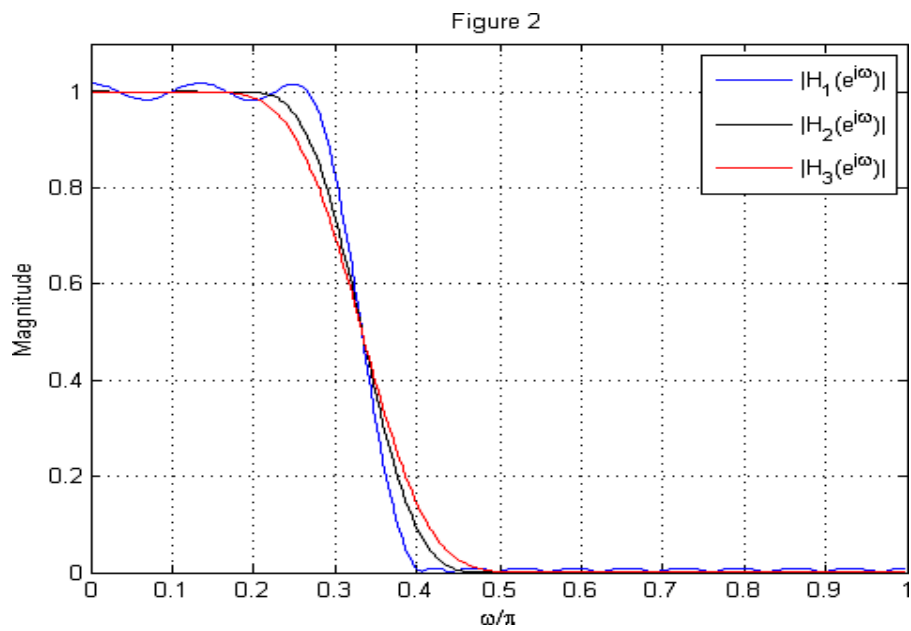
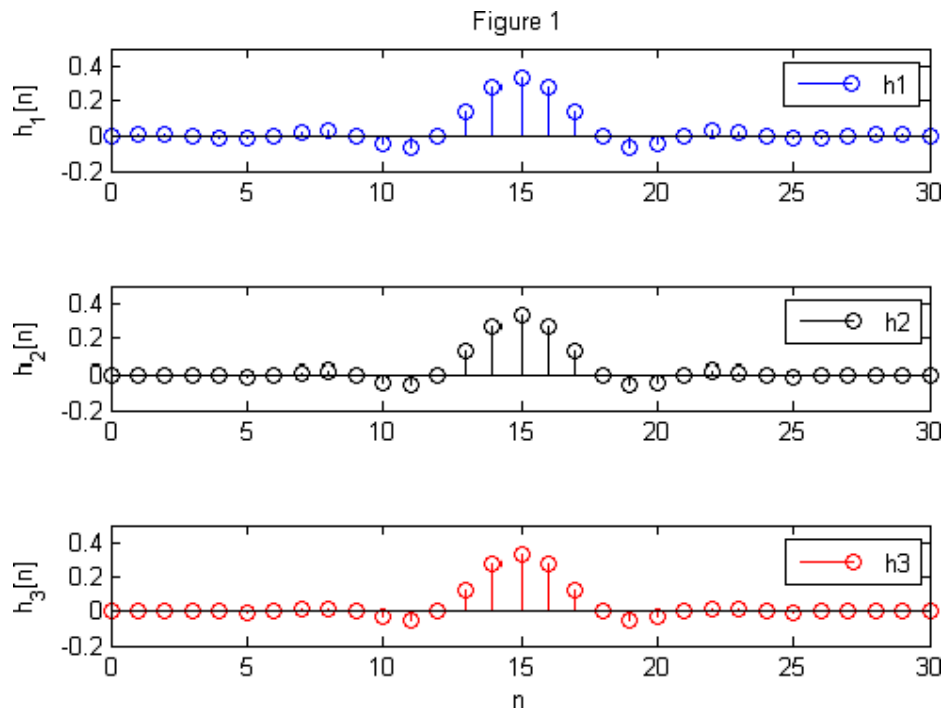
PROCEDURE:

1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (Multirate Filters)

```
close all, clear all
N =31; % Filter length
Nord = N-1; % Filter order
L = 3;
ro1 = 0.2; % Roll-off
factor
h1 = firnyquist(Nord,L,ro1); % Filter design
ro2 = 0.4; % Roll-off
factor
h2 = firnyquist(Nord,L,ro2); % Filter design
ro3 = 0.6; % Roll-off
factor
h3 = firnyquist(Nord,L,ro3); % filter design
figure (1)
subplot(3,1,1)
stem(0:N-1,h1,'b')
axis([0,30,-0.2,0.5])
ylabel('h_1[n]')
title('Figure 1')
legend('h1')
subplot(3,1,2)
stem(0:N-1,h2,'k')
axis([0,30,-0.2,0.5])
ylabel('h_2[n]')
legend('h2')
subplot(3,1,3)
stem(0:N-1,h3,'r')
axis([0,30,-0.2,0.5])
xlabel('n')
ylabel('h_3[n]')
legend('h3')
% Computing frequency responses
[H1,f] = freqz(h1,1,256,2);
[H2,f] = freqz(h2,1,256,2);
[H3,f] = freqz(h3,1,256,2);
figure (2)
plot(f,abs(H1),'b',f,abs(H2),'k',f,abs(H3),'r'), grid
title ('Figure 2')
axis([0,1,0,1.1])
xlabel('\omega/\pi')
ylabel('Magnitude')
legend(' |H_1(e^j\omega)| ', ' |H_2(e^j\omega)| ', ' |H_3(e^j\omega)| ')
```

OUTPUT: (Multirate Filters)



RESULT:

Thus the linear phase L^{th} band filter is designed and the magnitude response of the filter is obtained using MATLAB.

PART B - TMS320C6713 DSP Processor Based Experiments

- ❖ **Study of TMS320C6713 DSP Processor**
 - To study the architecture of TMS320C6713 DSP Processor.
- ❖ **Linear & Circular Convolution**
 - To write the assembly program to implement linear convolution operation
Using DSK Code composer
studio
- ❖ **FIR Filter Design Techniques (Using Windowing Method)**
 - To design and implement a Digital FIR Filter & observe its frequency response.
- ❖ **N-Point Fast Fourier Transform (FFT) Algorithm**
 - To find the DFT of a sequence using FFT algorithm.
- ❖ **Power Density Spectrum of a Sequence**
 - To compute power density spectrum of a sequence
- ❖ **Image Computation Using Discrete Cosine Transform**
 - To find the Discrete Cosine Transform of Image
- ❖ **LED Flash System Design Using DSP Processor**
 - To design and implement a LED FLASH System

PART C - VSK 6713 DSP Processor Based Lab Experiments

- ❖ **Arithmetic/Logic operation**
 - Write assembly language program to implement Arithmetic/Logic operation using VI Universal debugger.
- ❖ **Linear and Circular convolution.**
 - Write assembly language program to perform the operation of Linear and Circular convolution.
- ❖ **Auto and Cross correlation**
 - Write assembly language program to perform the operation of Cross correlation.
- ❖ **Fourier Transform**
 - Write assembly language program to perform the Fourier Transform (4,8,N-Point)
- ❖ **LED Interfacing**
 - Write assembly language program to interface LED display program.

PART B

TMS320C6713 DSP PROCESSOR

Based Experiments

(Both Kit/Simulator/Debugger)

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

Linear Convolution involves the following operations.

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

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

$x[n]$ = Input signal Samples

$h[k]$ = Impulse response co-efficient.

$y[r]$ = Convolution output.

n = No. of Input samples

h = No. of Impulse response co-efficient.

ALGORITHM TO IMPLEMENT 'C' OR ASSEMBLY PROGRAM FOR CONVOLUTION:

Eg: $x[n] = \{1, 2, 3, 4\}$

$h[k] = \{1, 2, 3, 4\}$

Where: $n=4, k=4$. ; Values of n & k should be a multiple of 4.

If n & k are not multiples of 4, pad with zero's to make multiples of 4

$r = n+k-1$; Size of output sequence.

$= 4+4-1$

$= 7$.

$r = 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6$

$n = 0 \ x[0]h[0] \ x[0]h[1] \ x[0]h[2] \ x[0]h[3]$

$1 \ x[1]h[0] \ x[1]h[1] \ x[1]h[2] \ x[1]h[3]$

$2 \ x[2]h[0] \ x[2]h[1] \ x[2]h[2] \ x[2]h[3]$

$3 \ x[3]h[0] \ x[3]h[1] \ x[3]h[2] \ x[3]h[3]$

Output: $y[r] = \{1, 4, 10, 20, 25, 24, 16\}$.

NOTE: At the end of input sequences pad 'n' and 'k' no. of zero's

ASSEMBLY PROGRAM TO IMPLEMENT LINEAR CONVOLUTION



conv.asm:

```
.global _main
X .half 1,2,3,4,0,0,0,0 ;input1, M=4
H .half 1,2,3,4,0,0,0,0 ;input2, N=4
.bss Y,14,2 ;OUTPUT, R=M+N-1
;At the end of input sequences pad 'M' and 'N' no. of zero's
_main:
MVKL .S1 X,A4
MVKH.S1 X,A4 ;POINTER TO X
MVKL .S2 H,B4
MVKH.S2 H,B4 ;POINTER TO H
MVKL .S1 Y,A5
MVKH.S1 Y,A5 ;POINTER TO Y
MVK .S2 7,B2 ;R=M+N-1
;MOVE THE VALUE OF 'R' TO B2 FOR DIFFERENT LENGTH OF I/P SEQUENCES
ZERO .L1 A7
ZERO .L1 A3 ;I=0
LL2:
ZERO .L1 A2
ZERO .L1 A8 ;J=0, for(i=0;i<m+n-1;i++)
LL1:
LDH .D1 *A4[A8],A6 ; for(j=0;j<=i;j++)
MV .S2X A8,B5 ; y[i]+=x[j]*h[i-j];
SUB .L2 A3,B5,B7
LDH .D2 *B4[B7],B6
NOP 4
MPY .M1X A6,B6,A7
ADD .L1 A8,1,A8
ADD .L1 A2,A7,A2
CMPLT .L2X B5,A3,B0
[B0] B .S2 LL1
NOP 5
STH .D1 A2,*A5[A3]
ADD .L1 A3,1,A3
CMPLT .L1X A3,B2,A2
[A2] B .S1 LL2
NOP 5
B B3
NOP 5
```

'C' PROGRAM TO IMPLEMENT LINEAR CONVOLUTION

```
#include<stdio.h>
main()
{ int m=4; /*Lenght of i/p samples sequence*/
int n=4; /*Lenght of impulse response Co-efficients */
int i=0,j;
int x[10]={ 1,2,3,4,0,0,0,0}; /*Input Signal Samples*/
int h[10]={ 1,2,3,4,0,0,0,0}; /*Impulse Response Co-efficients*/
/*At the end of input sequences pad 'M' and 'N' no. of zero's*/
int y[10];
for(i=0;i<m+n-1;i++)
{ y[i]=0;
for(j=0;j<=i;j++)
y[i]+=x[j]*h[i-j];
}
for(i=0;i<m+n-1;i++)
printf("%d\n",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.

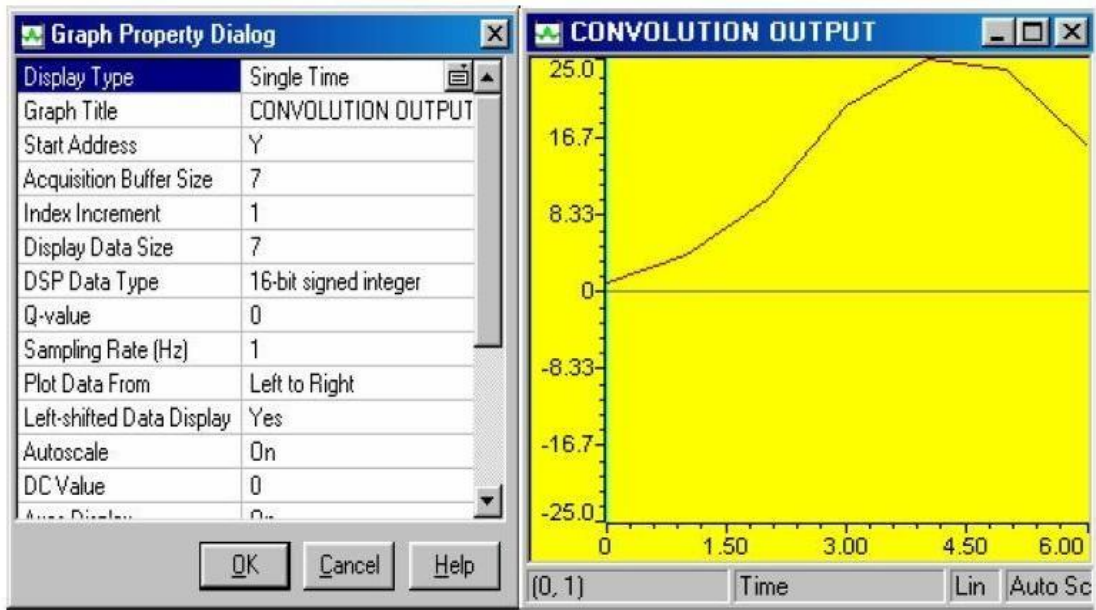
RESULT:

Configure the graphical window as shown below

INPUT

$x[n] = \{1, 2, 3, 4, 0, 0, 0, 0\}$

$h[k] = \{1, 2, 3, 4, 0, 0, 0, 0\}$

OUTPUT:**Note:**

1. To execute the above program follow “ procedure to work on code composer studio”
2. To view graphical output follow the above procedure.

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

C Program to Implement Circular Convolution

```
#include<stdio.h>
int m,n,x[30],h[30],y[30],i,j,temp[30],k,x2[30],a[30];
void main()
{
printf(" enter the length of the first sequence\n");
scanf("%d",&m);
printf(" enter the length of the second sequence\n");
scanf("%d",&n);
printf(" enter the first sequence\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf(" enter the second sequence\n");
for(j=0;j<n;j++)
scanf("%d",&h[j]);
if(m-n!=0) /*If length of both sequences are not equal*/
{
if(m>n) /* Pad the smaller sequence with zero*/
{
for(i=n;i<m;i++)
h[i]=0;
n=m;
}
for(i=m;i<n;i++)
x[i]=0;
m=n;
} y[0]=0;
a[0]=h[0];
```

```

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

```

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.

INPUT:

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

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

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

RESULT:

The C program was written and verified successfully for linear & circular convolution operation Using DSK Code composer studio.

EXP. NO: 10	FIR FILTER (WINDOWING TECHNIQUES) DESIGN USING TMS320C6713 DSP PROCESSOR
DATE:	

AIM:

Design and implement a Digital FIR Filter & observe its frequency response.

C Program for Digital FIR Filter


```
#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()
{
```

```

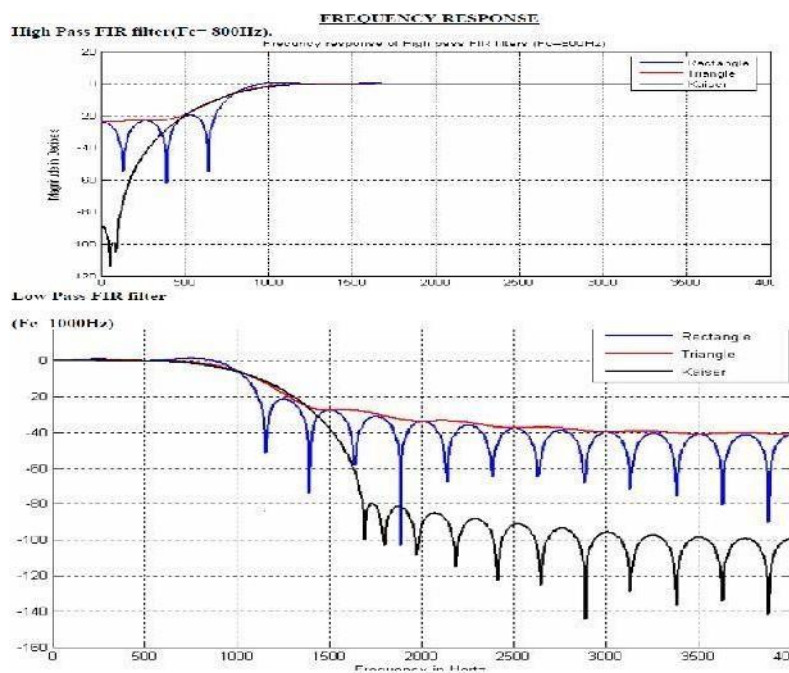
DSK6713_AIC23_CodecHandle hCodec;
Uint32 l_input, r_input, l_output, r_output;
/* Initialize the board support library, must be called first */
DSK6713_init();
/* Start the codec */
hCodec = DSK6713_AIC23_openCodec(0, &config);
DSK6713_AIC23_setFreq(hCodec, 1);
while(1)
{ /* Read a sample to the left 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 :

- Switch on the DSP board.
- Open the Code Composer Studio.
- Create a new project
- Project -  New (File Name. pj1 , Eg: FIR.pj1)
- 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.
- Observe the waveform that appears on the CRO screen.

RESULT: FREQUENCY RESPONSE High Pass FIR filter($F_c = 800\text{Hz}$).



RESULT:

The C program was written and verified successfully for Digital FIR Filter operation Using DSK Code composer studio.

EXP. NO: 11	IMPLEMENTAION OF FAST FOURIER TRANSFORM (FFT) ALGORITHM
DATE:	

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

C PROGRAM TO IMPLEMENT 4 POINT FFT :

Main.c (fft 256.c):

```
#include <math.h>
#define PTS 64 //# of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n); //FFT prototype
float iobuffer[PTS]; //as input and output buffer
float x1[PTS]; //intermediate buffer
short i; //general purpose index variable
short buffercount = 0; //number of new samples in iobuffer
short flag = 0; //set to 1 by ISR when iobuffer full
COMPLEX w[PTS]; //twiddle constants stored in w
COMPLEX samples[PTS]; //primary working buffer
main()
{
for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/(PTS*2.0)); //Re component of twiddle constants
w[i].imag = -sin(2*PI*i/(PTS*2.0)); //Im component of twiddle constants
}
for (i = 0 ; i < PTS ; i++) //swap buffers
{
iobuffer[i] = sin(2*PI*10*i/64.0); /*10- > freq,
64 -> sampling freq*/
samples[i].real=0.0;
samples[i].imag=0.0;
}
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i] = sqrt(samples[i].real*samples[i].real
+ samples[i].imag*samples[i].imag);
}
} //end of main
```

C PROGRAM TO IMPLEMENT 8 POINT FFT :

```
#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;
}

```

C PROGRAM TO IMPLEMENT N POINT FFT :

```

#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;

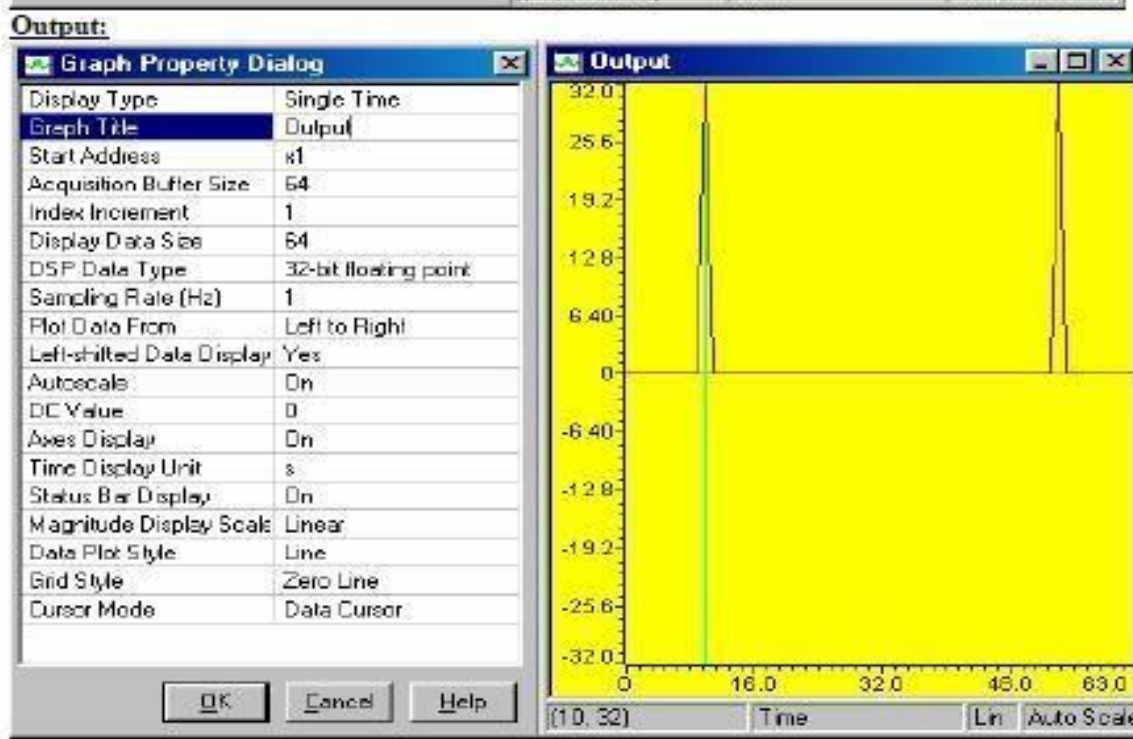
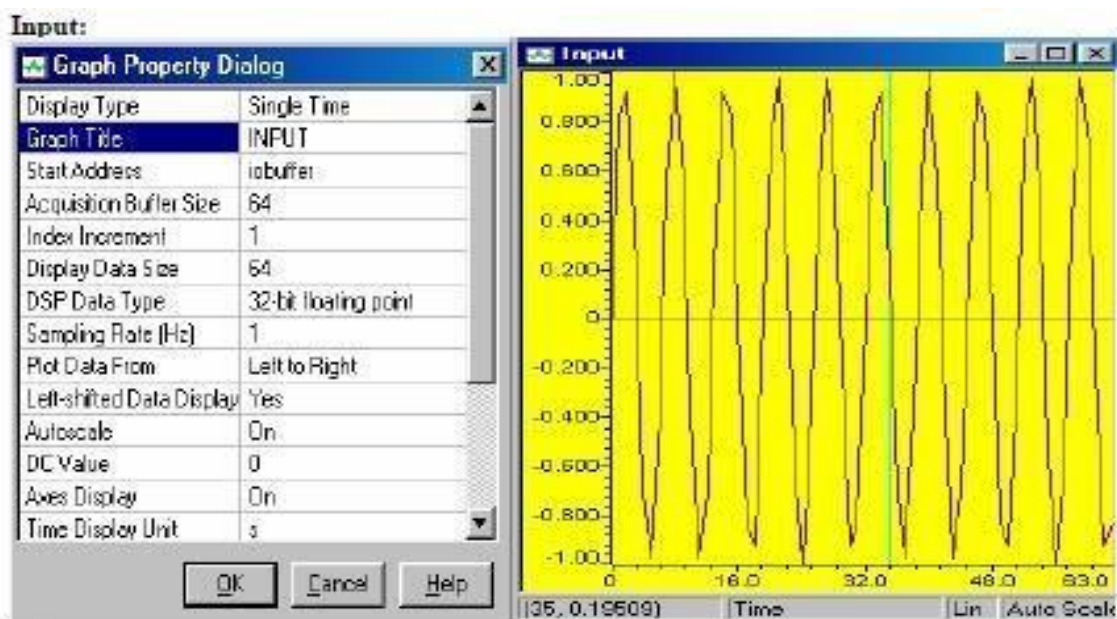
```

```

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;
}

```

RESULT:



RESULT:

The C program was written and verified successfully for 4,8,N-Point FFT Algorithm Using DSK Code composer studio.

EXP. NO:12	POWER DENSITY SPECTRUM OF A SEQUENCE
DATE:	

AIM: To compute power density spectrum of a sequence [using TMS320c6713 DSP processor]

‘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*/
}
}

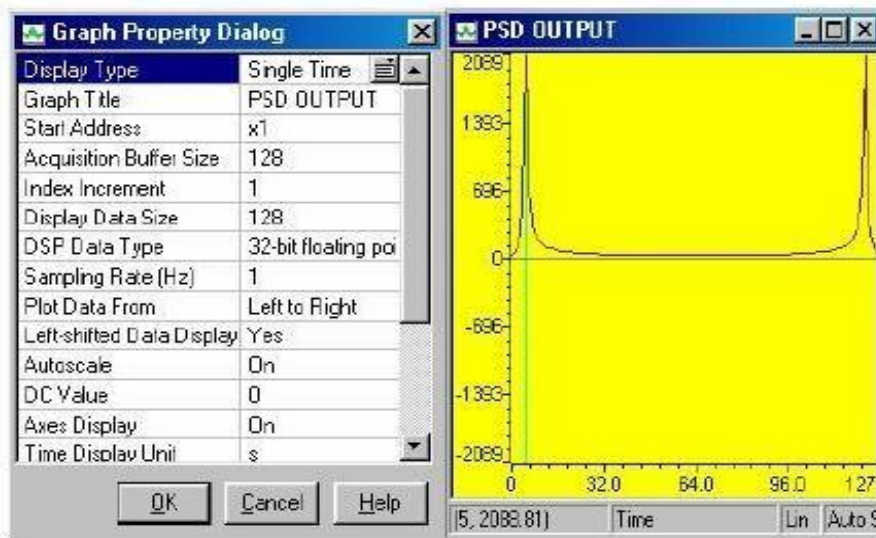
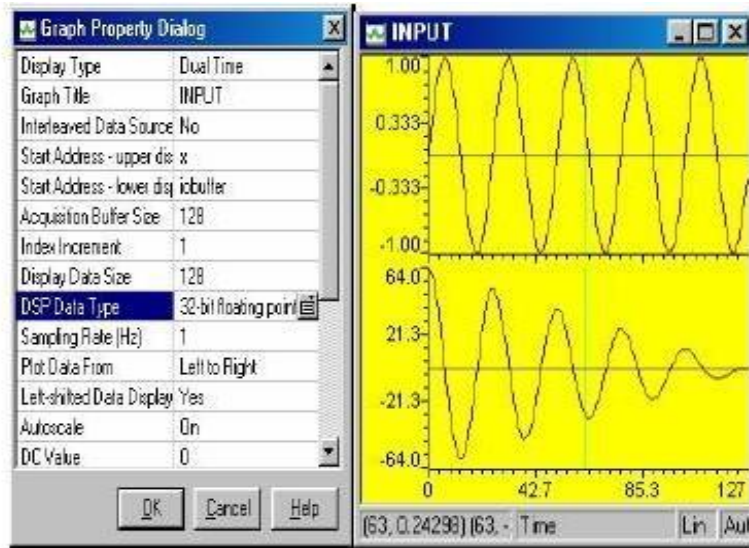
```

```

}
/*****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

```

OUTPUT:



RESULT:

The C program was written and verified successfully for Power density Spectram for the given sequence Using DSK Code composer studio.

PART C

**VSK 6713 DSP PROCESSOR Based
Lab Experiments**

(Both Kit/VI Debugger Simulator)

Step by step Procedure for VI Debugger Software

- 1) User can enter the debugger for C6713 icon, the corresponding page is opened immediately.
- 2) Now a new window is opened without work space.
- 3) Select menu bar - **View > Workspace..**
- 4) Select **serial** and click **port settings**
- 5) Click **Auto Detect** for communication VSK - C6713 trainer kit and PC.

Note :

- i. Connect PC & kit by serial port connector (PC to PC)
 - ii. Reset the kit and set the Baudrate at 19200 in communication port setting window.
- 6) Select the **Project** menu and click **New Project**, for creating new project window
 - 7) In the **file name** block type project name Eg: ADDITION and **save** it.
 - 8) Type ADDITION Program in Assembling Language and **Save**
 - 9) While saving change in **Save As type** as **Assembly Files** and type file name eg: ADD.ASM inside the **My Project** Folder
 - 10) Select **Project -> Add File to Project**, for adding the assembly file eg: ADD.ASM to above created project eg: ADDITION.
 - 11) Select the File name and **Open** it eg: ADD.ASM
 - 12) Select **Project -> Add File to Project** for adding the CMD file eg:

MICRO6713.CMD Now assembling and CMD files are added to the created project (eg: ADDITION) Select **Project -> Build**, for compiling the project
After compilation, if the program have no error the following view will appear

Note :

Now only ADD.ASC file is created for the project

- 13) Select **Serial -> Load Program**, for downloading the file eg: ADD.ASC to VSKC6713kit
Now browse the ADD.ASC file from **My Project** folder.

Now click **OK** in **Download File** window, then **successfully downloaded** window will appear. Select **Serial -> Communication window** for executing and checking the result

- 14) Now type, (Words in caps)

#GO 00006000

<ENTER>

After Getting execution Reset the VSK-C6713 Trainer kit.

- 15) Check the Result by type, (words in caps)
#SP 00008000 (This is ON chip memory location)

Now Result will appear in the window.

EXP. NO:13	LED FLASH SYSTEM DESIGN USING DSP PROCESSOR
DATE:	

To design and implement a LED FLASH System using DSP Processor TMS320C6713

ASSEMBLY LANGUAGE PROGRAM FOR THE LED FLASH SYSTEM
USING VLDEBUGGER

```
;***** LED PROGRAM*****
        .sect
        .text
start:
    mvkl  .s1    0x000000AA,a4
    mvkl  .s1    0x00000055,a6
    mvkl  .s1    0x90040016,a3
    mvkh  .s1    0x90040016,a3
    stb   .d1    a4,*a3
    nop
    mvkl          RET,b11
    mvkh          RET,b11
    b      delay
    nop
    nop
RET:
    stb   .d1    a6,*a3
    nop
    mvkl  start,b11
    mvkh  start,b11
    b      delay
    nop
    nop    6
delay:
    mvkl  0x0005ffff,b2
    mvkh  0x0005ffff,b2
rep:
    sub          b2,1,b2
    nop
    nop    3
[b2]  b      rep
    nop
    nop
    b      b11
    nop
    nop    6
        .end
```

PROCEDURE:

- Start VSI6713 (ICON on the desktop/start-program)
- In the Window of VI, Select workspace view (Menu → VIEW →Workspace)
- Open new project (Menu → Project → New Project)
- In the edit window of the workspace, type the assembly language programming and save it as ASM File.
- In Root Window of workspace, select Assembly → Go to Menu → Project → Add File to Project – Browse to the newly saved ASM File.
- Check the tree for the correct file and content by double clicking it (view the content on the edit window of the work space).
- In Root Window of workspace, select Cmd Files → Go to Menu → Project → Add File to Project – Browse to the MICRO167.cmd file.
- Check the tree for the correct file and content by double clicking it (view the content on the edit window of the work space).
- Go to Menu → Project → Build (compile/interpret will be completed without error)
- Power on the DSK
- Go to Menu → Serial → Port Settings; in the pop up window, set baud rate = 19200;
- Reset the DSK using the Button switch.
- In Port Setting Window, click on “Autodetect” (Connection to DSK with PC is acknowledged thro COM port)
- Go to Menu → Serial → Load program → browse to the new file saved with extension .ASC
- Go to Menu → Serial → Communication Window.
- In the pop window, type GO <space> <Starting address> <Press Enter>
- Verify the program output in the DSK through LEDs.

RESULT:

The ASM program was written and verified successfully for LED Display Interfacing operation Using DSK Code composer studio.

BASIC DSP OPERATION IN C6713

CONVOLUTION

4.1 LINEAR CONVOLUTION

```
input    .set 80001000h    ; 00009000h ;
coeff    .set 80001100h    ; 00009050h ;
output   .set 80001200h    ; 00009100h ;
buff     .set 80001300h    ; 00009200h ;
```

```
.sect "00006000h"
```

```
.text
```

```
    mvkl    input,a4
    mvkh    input,a4
    mvkl    coeff,a5
    mvkh    coeff,a5
    add     a4,10h,a4
    nop     2
    add     a5,10h,a5
    nop     2
    mvkl    buff,a3
    mvkh    buff,a3
    mvkl    output,a6
    mvkh    output,a6
    mvkl    8,b2
    mvkh    8,b2

zer:  mvkl    00000000h,a2
      mvkh    00000000h,a2
      stw     a2,*a3++
      nop     7
      stw     a2,*a4++
      nop     7
      stw     a2,*a5++
      nop     7
      stw     a2,*a6++
      nop     7
      sub     b2,1h,b2
      nop     2
[b2] b      zer
      nop     6
      mvkl    input,a4
```

```

    mvkh    input,a4
    mvkl    7h,b1
    mvkh    7h,b1
    mvkl    output,a9
    mvkh    output,a9
start1:
    mvkl    coeff,a1
    mvkh    coeff,a1
    mvkl    buff,a3
    mvkh    buff,a3
    ldw     *a4++[1],a8
    nop     6
    stw     a8,*a3
    nop     6
    mvkl    4,b0
    mvkh    4,b0
    nop     3
    mvkl    00000000H,a7
    mvkh    00000000H,a7
loop1:
    ldw     *a1++,a5
    nop     6
    ldw     *a3++,a6
    nop     6
    mpy     a5,a6,a6
    nop     4
    add     a7,a6,a7
    nop     2
    sub     b0,1,b0
    nop     2
[b0] b     loop1
    nop     7
    stw     a7,*a9++[1]
    nop     6 mvkl
    4,b0 mvkh
    4,b0 mvkl
    buff,b3 mvkh
    buff,b3 ldw
    *b3,b4 nop
    6
loop2:
    ldw     *+b3(4),b5
    nop     6
    stw     b4,*++b3
    nop     6
    mv      b5,b4
    nop     2
; loop to copy x(n) to x(n-1)

```

```

        sub    b0,1,b0
        nop    2
[b0]   b      loop2
        nop    6
        sub    b1,1,b1
        nop    2
[b1]   b      start1
        nop    7
halt:
        b      halt
        nop    7

```

INPUT : x(n)

```

;; Addr      Data
;; 80001000   00000001
;; 80001004   00000001
;; 80001008   00000001
;; 8000100C   00000001

```

IMPULSE : h(n)

```

Addr      Data
80001100   00000001
80001104   00000002
80001108   00000003
8000110C   00000004

```

OUTPUT : y(n)

```

;; 80001200   00000000
;; 80001204   00000004
;; 80001208   00000008
;; 8000120C   0000000C

```

```

80001210   00000019
80001210   00000018
80001210   00000010

```

4.2 CIRCULAR CONVOLUTION

```
.sect "00006000h"
.text

value1 .SET 80001000H ; input value1
value2 .SET 80001100H ; input value2
OMEMEM .SET 80001200H ; output values

IMEMEM .SET 80001500H ; intermediate

start:
    mvkl    value1,a12
    mvkh    value1,a12
    mvkl    value2,a13
    mvkh    value2,a13
    add     a12,10h,a12
    add     a13,10h,a12
    nop     2
    mvkl    8h,b0
    mvkh    8h,b0
    zero    a5

filzer:
    stw     a5,*a12++[1]
    nop     5
    stw     a5,*a12++[1]
    nop     5
    sub     b0,1,b0
    nop     2
[b0] b     filzer
    nop     7
    mvkl    IMEMEM,a12
    mvkh    IMEMEM,a12
    nop
    mvkl    value2,a13
    mvkh    value2,a13
    nop
    mvkl    4,b0
    mvkh    4,b0
    nop

another:
    ldw     *a13++[1],a4
    nop     6
```

```

        stw      a4,*a12++[1]
        nop      5
        sub      b0,1,b0
        nop
[b0]    b        another
        nop
        nop      5
        mvkl    IMEM,a14
        mvkh    IMEM,a14
        nop
        ldw     *++a14[1],a4
        nop      5
        ldw     *++a14[2],a5
        nop      5
        stw     a4,*a14--[2]
        nop
        nop      5
        stw     a5,*a14
        nop
        nop      5
        mvkl    OMEM,a10
        mvkh    OMEM,a10
        nop
        mvkl    4,b2
        mvkh    4,b2
        nop
nextdata:
        mvkl    IMEM,a11
        mvkh    IMEM,a11
        nop
        mvkl    value1,a12
        mvkh    value1,a12
        nop
        mvkl    4,b0
        mvkh    4,b0
        nop
        mvkl    0,a9
        mvkh    0,a9
        nop
next:
        ldw     *a11++[1],a4
        nop      6
        ldw     *a12++[1],a5
        nop      6
        mpy     a4,a5,a6
        nop      3
        add     a6,a9,a9
        nop      3
        sub     b0,1,b0
        nop      3

```

```

[b0]  b      next
      nop
      nop      5
      stw     a9,*a10++[1]
      nop
      nop      5
      mvkl   back,b11
      mvkh   back,b11
      nop
      b      shift
      nop
      nop      5
back:  sub     b2,1,b2
      nop      3
[b2]  b      nextdata
      nop
      nop      5
halt:  b      halt
      nop
      nop      5
shift: mvkl   IMEM,a5
      mvkh   IMEM,a5
      nop
      mvkl   3,b1
      mvkh   3,b1
      nop
      ldw    *++a5[3],a4
      nop      6
      mvkl   IMEM,a5
      mvkh   IMEM,a5
      nop
      add    a5,8H,a5
      nop      2
shloop: ldw    *a5++[1],a6
      nop      6
      stw    a6,*a5--[2]
      nop      6
      sub    b1,1,b1
      nop

```

```
[b1] b      shloop
      nop
      nop      6
      mvkl     IMEM,a5
      mvkh     IMEM,a5
      nop
      stw      a4,*a5
      nop      6
      b        b11
      nop
      nop      6
```

```
;  
; Sample Inputs and Outputs:
```

```
;  
; Location      Data  
;  
; x1(n) Input  
;  
; 80001000h     00000004h  
; 80001004h     00000003h  
; 80001008h     00000002h  
; 8000100ch     00000001h  
;  
; x2(n) Input  
;  
; 80001100h     00000001h  
; 80001104h     00000002h  
; 80001108h     00000003h  
; 8000110ch     00000004h  
;  
; y(n) Output  
;  
; 80001200h     00000018h  
; 80001204h     00000016h  
; 80001208h     00000018h  
; 8000120ch     0000001eh  
;  
;  
;  
;  
;
```

CORRELATION

4.3 CROSS CORRELATION

```
INPUT1 .SET 80001000H
INPUT2 .SET 80001100H
OUTPUT .SET 80001200H
```

```
BUFFER .SET 80001500H
```

```
.sect "00006000h"
.text
    mvkl    BUFFER,a4           ;BUFFER
    mvkh    BUFFER,a4
    mvkl    INPUT1,a6
    mvkh    INPUT1,a6
    mvkl    INPUT2,a7
    mvkh    INPUT2,a7
    zero    a5
    mvkl    00000010h,b0
    mvkh    00000010h,b0
    add     a6,b0,a6
    nop     2
    add     a7,b0,a7
    nop     2
filz:
    stw     a5,*a4++[1]
    nop     6
    stw     a5,*a6++[1]
    nop     6
    stw     a5,*a7++[1]
    nop     6
    sub     b0,1,b0
    nop     2
[b0] b     filz
    nop     6
    mvkl    INPUT2,a3           ; x2
    mvkh    INPUT2,a3
    mvkl    BUFFER,a4
    mvkh    BUFFER,a4
    mvkl    00000004h,b0
    mvkh    00000004h,b0
buff:
    ldw     *a3++[1],a5
```

```

        nop        6
        stw        a5,*a4++[1]
        nop        6
        sub        b0,1,b0
        nop        2
[b0]    b          buff
        nop
        nop        6
        mvkl       INPUT1,a0          ; x1
        mvkh       INPUT1,a0
        mvkl       BUFFER,a3          ; x2 transferred to buffer
        mvkh       BUFFER,a3
        mvkl       OUTPUT,a1         ; y
        mvkh       OUTPUT,a1
        mvkl       0004h,b0
        mvkh       0000h,b0
loopg:
        mvkl       INPUT1,a0
        mvkh       INPUT1,a0
        mvkl       BUFFER,a3
        mvkh       BUFFER,a3
        mvkl       0004h,b2
        mvkh       0000h,b2
        mvkl       0000h,b7
        mvkh       0000h,b7
        nop        4
corlp:
        ldw        *a0++,b4
        nop        6
        ldw        *a3++,b5
        nop        6
        mpy        b4,b5,b6
        nop        6
        add        b6,b7,b7
        nop        6
        sub        b2,1h,b2
        nop        6
[b2]    b          corlp
        nop        6
        nop        5
        mv         b7,a8
        nop        2
        mvkl       0004h,a10
        mvkl       0000h,b9
        nop
again:
        nop        4
        sub        a8,a10,a8

```

```

        nop
        nop      6
        mv       a8,b1
        nop
        nop      4
        cmpgt   a8,0h,b1
        nop      4
[b1]    b       sum
        nop      6
        cmplt  a8,0h,b1
        nop      4
[b1]    b       store
        nop      6
        b       sum
        nop      6
sum:
        add    b9,1h,b9
        nop    4
[b1]    b      again
        nop    6
store:
        nop
        stw   b9,*a1++
        nop   4
        nop   5
        mvkl BUFFER,a7
        mvkh BUFFER,a7
        add  a7,4,a6
        mvkl 0004h,b2
        nop
cpylp:
        ldw   *a6++,a9
        nop   5
        stw  a9,*a7++
        nop   5
        sub  b2,1h,b2
        nop   5
[b2]    b    cpylp
        nop   6
        sub  b0,1h,b0
        nop   5
[b0]    b    loopg
        nop   5
        nop   4
halt:
        b     halt
        nop
        nop   5

```

```
;
;
;
;
; Sample Inputs and Outputs:
;-----
; Location      Data
;-----
; x1(n) Input Sequence
;
; 80001000h     00000001h
; 80001004h     00000002h
; 80001008h     00000003h
; 8000100ch     00000004h
;
; x2(n) Input Sequence
;
; 80001100h     00000001h
; 80001104h     00000002h
; 80001108h     00000003h
; 8000110ch     00000004h
;
; y(n) Output Sequence
;
; 80001200h     00000007h
; 80001204h     00000005h
; 80001208h     00000002h
; 8000120ch     00000001h
;
;
;
```



EVALUATION SHEET

Department of Electronics and Communication Engineering

SCSVMV UNIVERSITY, Enathur, Kanchipuram

Title of the Experiment	
Name of the Candidate	
Register Number	
Date of Submission	

S.No	Marks Split up	Maximum Marks (50)	Marks Earned
1	Attendance	5	
2	Pre lab viva questions	5	
3	Execution of Experiments	20	
4	Calculation/Evaluation of Result	10	
5	Post lab viva questions	10	
6	Grand Total	50	

Signature of the Faculty Handling the Lab