ST. ANNE'S

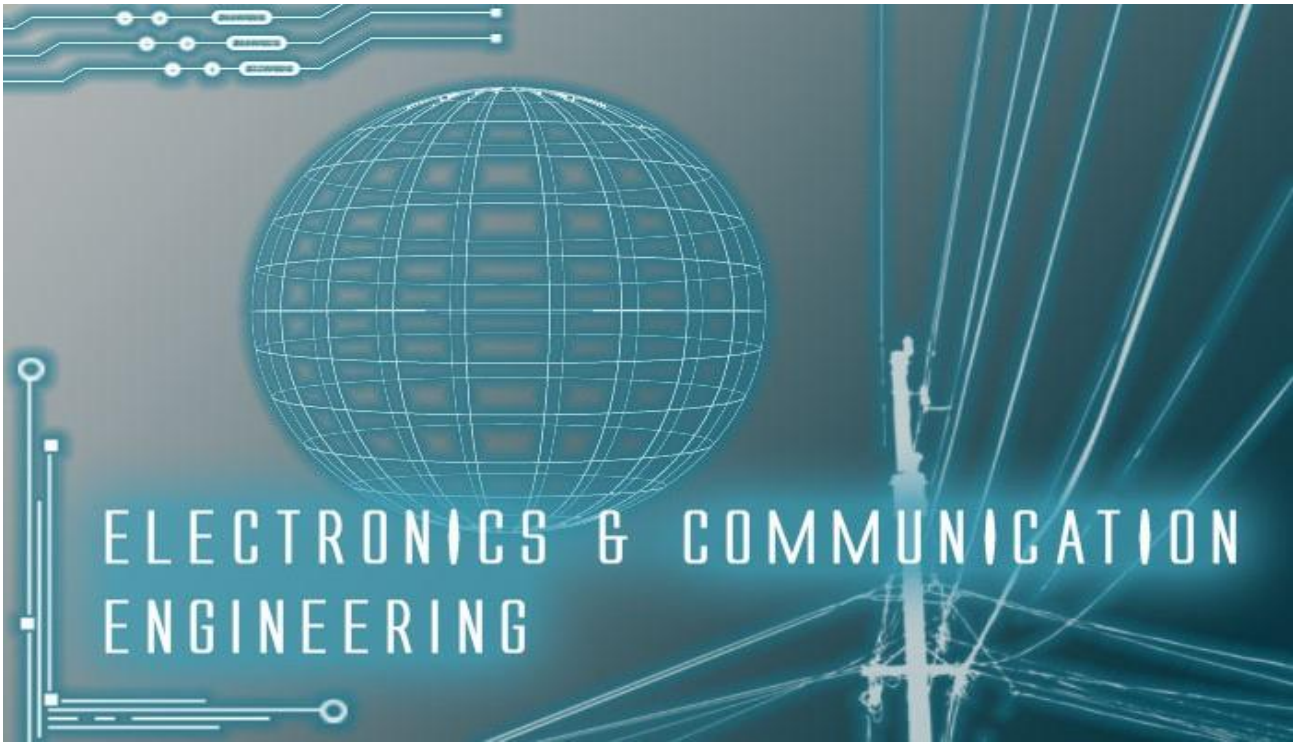
COLLEGE OF ENGINEERING AND TECHNOLOGY

(Approved by AICTE, New Delhi. Affiliated to Anna University, Chennai)
(An ISO 9001: 2015 Certified Institution)
ANGUCHETTYPALAYAM, PANRUTI – 607 106

LAB MANUAL

Regulation	: 2017
Branch	: *B.E.* – ECE
Year & Semester	: III Year / V Semester

EC8562-DIGITAL SIGNAL PROCESSING LABORATORY



ANNA UNIVERSITY CHENNAI

Regulation 2017

EC8562-DIGITAL SIGNAL PROCESSING LABORATORY

LIST OF EXPERIMENTS:

MATLAB / EQUIVALENT SOFTWARE PACKAGE

1. Generation of elementary Discrete-Time sequences
2. Linear and Circular convolutions
3. Auto correlation and Cross Correlation
4. Frequency Analysis using DFT
5. Design of FIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation
6. Design of Butterworth and Chebyshev IIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operations

DSP PROCESSOR BASED IMPLEMENTATION

1. Study of architecture of Digital Signal Processor
2. Perform MAC operation using various addressing modes
3. Generation of various signals and random noise
4. Design and demonstration of FIR Filter for Low pass, High pass, Band pass and Band stop filtering
5. Design and demonstration of Butter worth and Chebyshev IIR Filters for Low pass, High pass, Band pass and Band stop filtering
6. Implement an Up-sampling and Down-sampling operation in DSP Processor

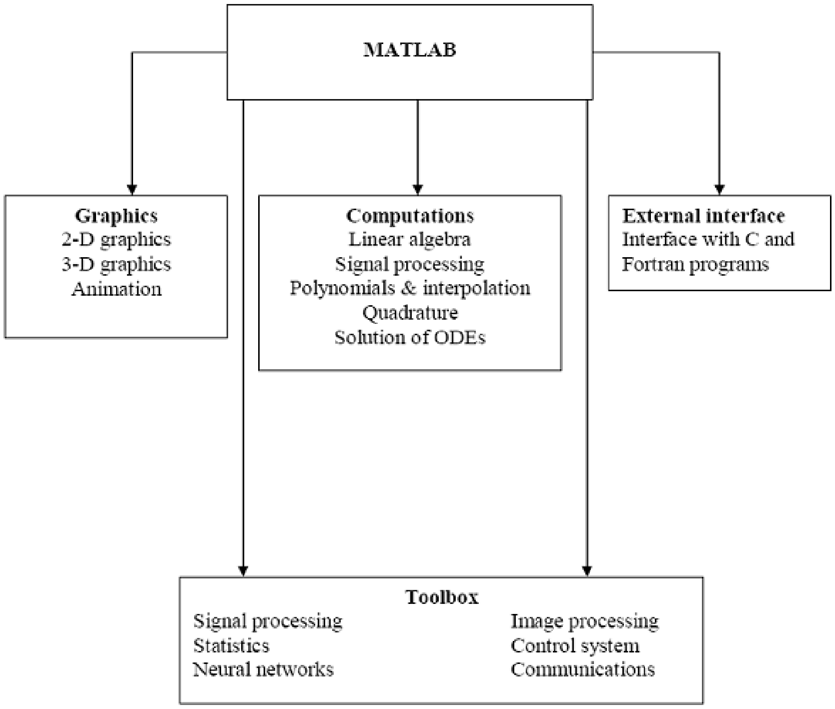
TOTAL: 60 PERIODS

INDEX

LIST OF EXPERIMENTS

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 96kHz. 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 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: 1

Date :

GENERATION OF DISCRETE TIME SIGNALS

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

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.

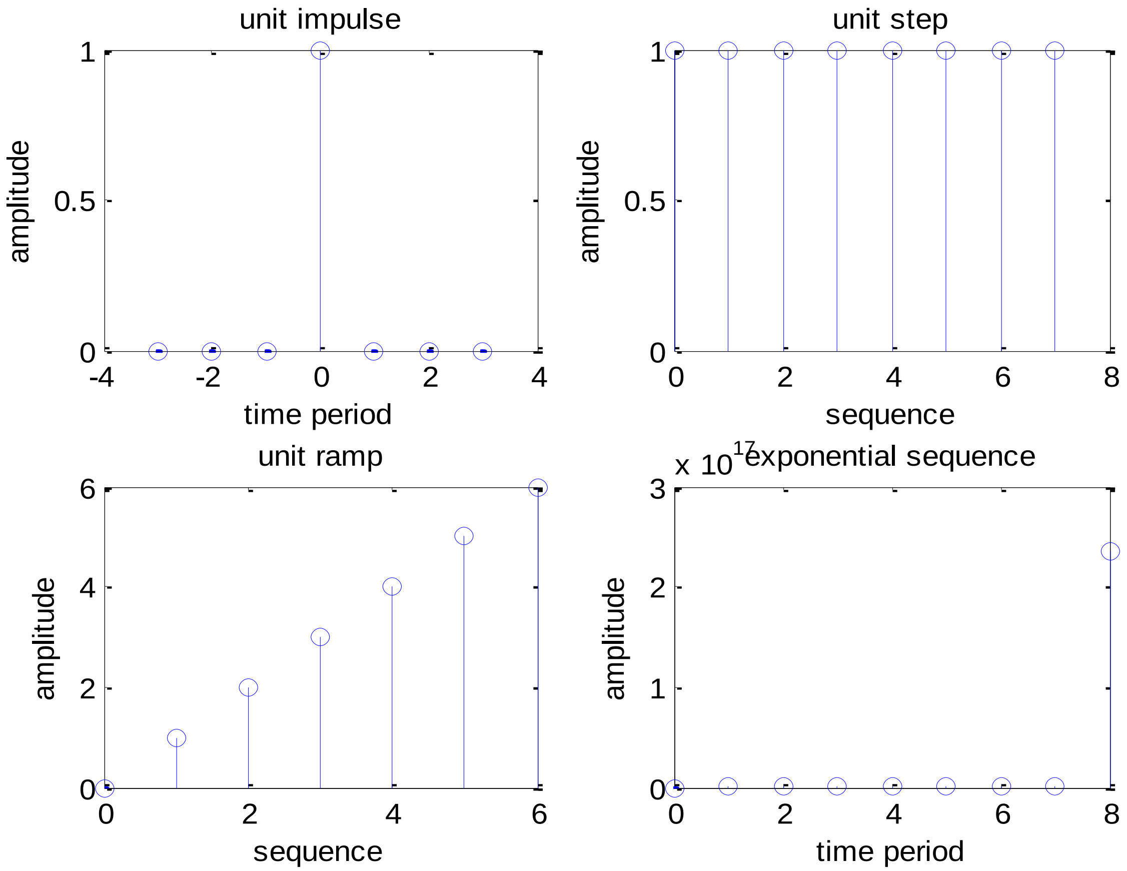
PROGRAMS: (GENERATION OF BASIC SIGNALS)

```
%Program for generation of unit impulse signal
t=-3:1:3;
y=[zeros(1,3),ones(1,1),zeros(1,3)];
Subplot (2, 2,1);
stem (t,y);
ylabel('amplitude');
xlabel('time period');
title('unit impulse')

%Program for generation of unit step signal
n=input('enter the sample length of unit step sequence');
t=0:1:n-1;
y=ones(1,n);
subplot(2,2,2);
stem(t,y);
ylabel('amplitude');
xlabel('sequence');
title('unit step')
```

```
%Program for generation of unit ramp  signal
n1=input('enter the sample length of unit  ramp sequence');
t=0:n1;
subplot(2,2,3);
stem(t,t);
ylabel('amplitude');
xlabel('sequence');
title('unit ramp')

%Program for generation of discrete exponential signal:
n2=input('enter the  length of the exponential sequence');
t=0:n2;
a=input('enter the a value');
y2=exp(a*t);
subplot(2,2,4);
stem(t,y2);
ylabel('amplitude');
xlabel('time period');
title('exponential sequence')


%Program for generation of continuous exponential signal:
n3=input('enter the  length of the exponential sequence');
t=0:2:n3-1;
a=input('enter the a value');
y3=exp(a*t);
subplot(3,1,1);
stem(t,y3);
ylabel('amplitude');
xlabel('time period');
title('continuous exponential sequence')

%Program for generation of sine wave
t=0:0.01: pi;
y=sin(2*pi*t);
subplot(3,1,2);
stem(t,y);
ylabel('amplitude');
xlabel('time period');
title('sine wave')

%Program for generation of cosine wave
t=0:0.01: pi;
y=cos(2*pi*t);
subplot(3,1,3);
stem(t,y);
ylabel('amplitude');
xlabel('time period');
title('cosine wave')
```

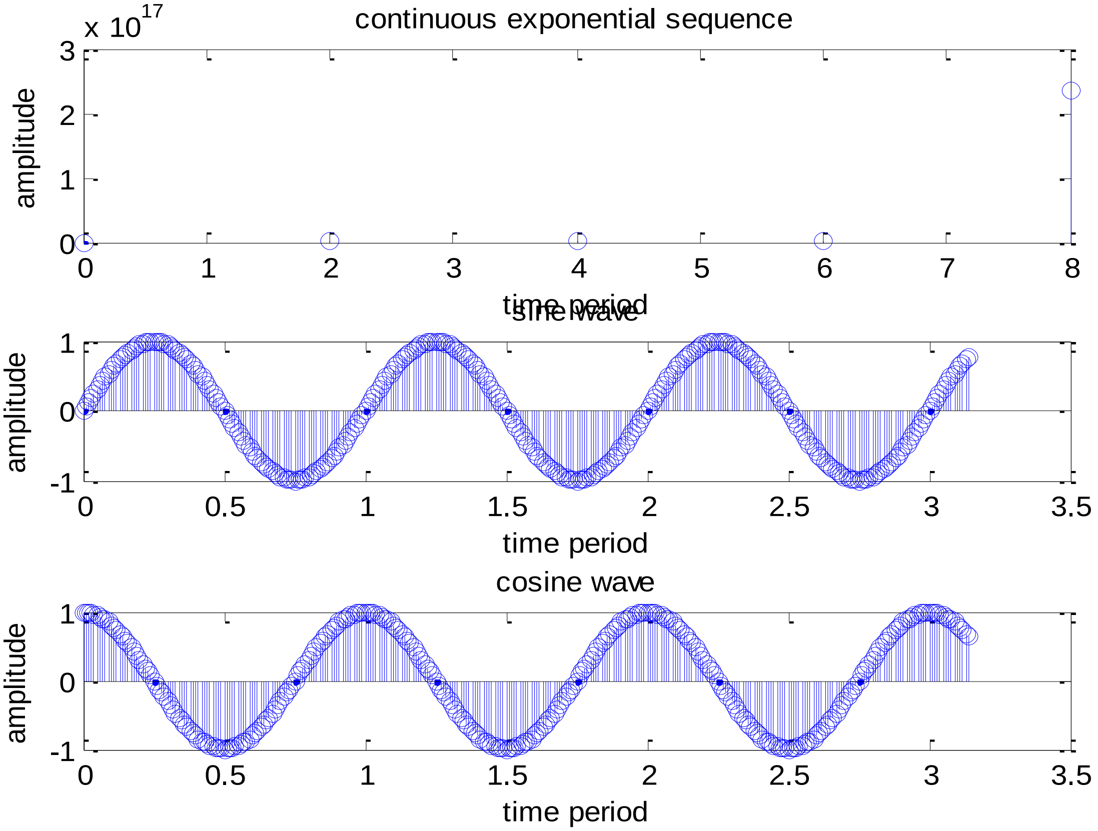
OUTPUT: (DISCRETE SIGNALS)

```
Enter the sample length of unit step sequence 8
Enter the length of ramp sequence    6
Enter the length of the exponential sequence 8
Enter the a value 5
```



OUTPUT: (CONTINUOUS SIGNALS)

```
Enter the length of continuous exponential sequence 10
Enter the a value 5
```



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 R2014a

PROCEDURE:

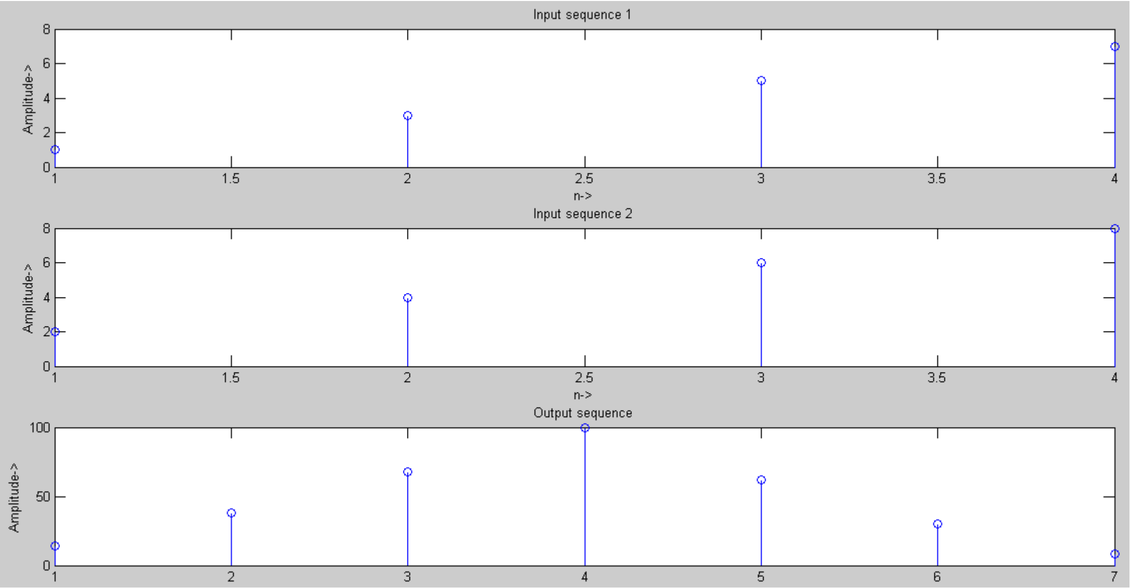
1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (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]
```

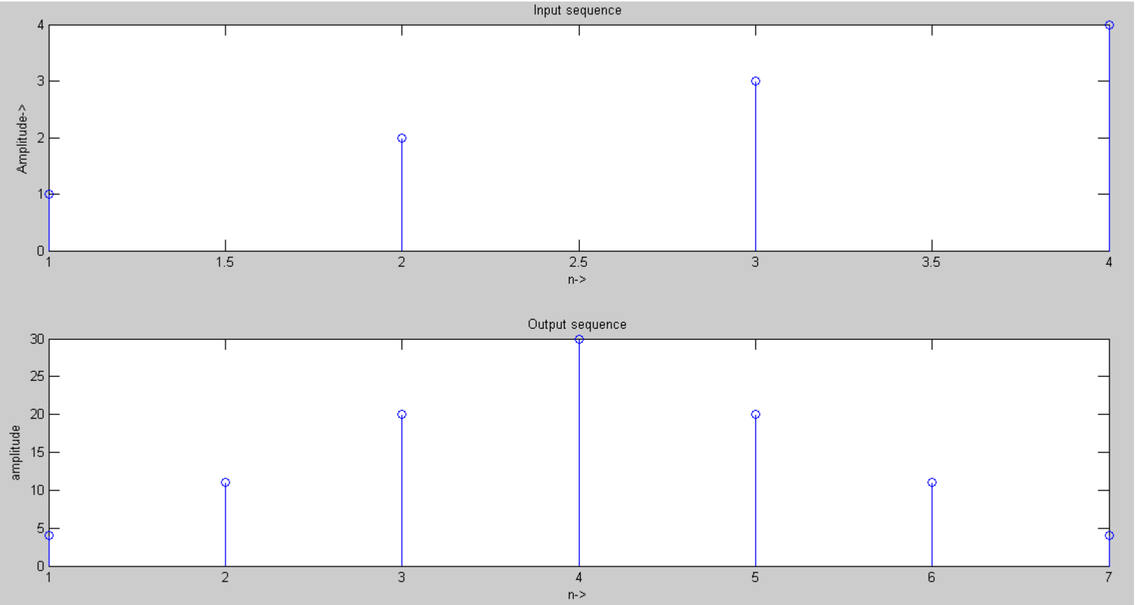


PROGRAM: (Auto Correlation Function)

```
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 AND CIRCULAR CONVOLUTIONS

AIM:

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

APPARATUS REQUIRED:

HARDWARE	: Personal Computer
SOFTWARE	: MATLAB R2014a

PROCEDURE:

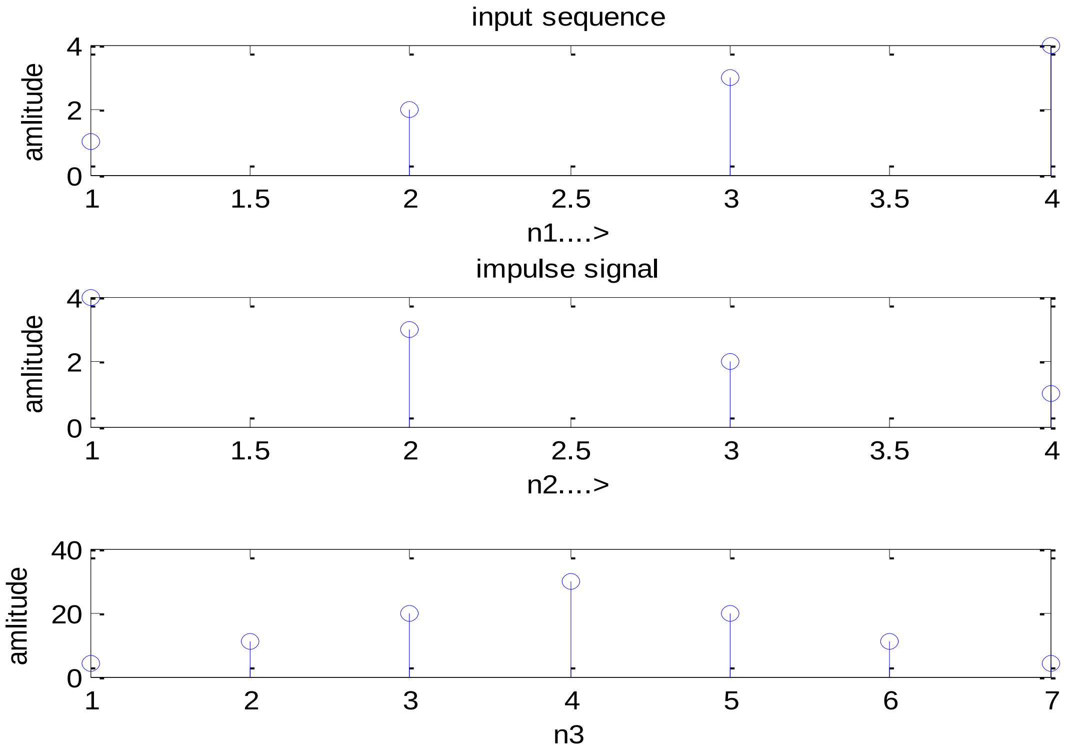
1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

```
%Program for linear convolution
%to get the input sequence
n1=input('enter the length of input sequence');
n2=input('enter the length of impulse sequence');
x=input('enter the input sequence');
h=input('enter the impulse sequence');

%convolution operation
y=conv(x,h);
%to plot the signal
subplot(3,1,1);
stem(x);
ylabel('amplitude');
xlabel('n1....>');
title('input sequence')
subplot(3,1,2);
stem(h);
ylabel('amplitude');
xlabel('n2....>');
title('impulse signal')
subplot(3,1,3);
stem(y);
ylabel('amplitude');
xlabel('n3');
disp('the resultant signal is');y
```

```
%circular convolution
clc;
clear all;
close all;
%to get the input sequence
g=input('enter the input sequence');
h=input('enter the impulse sequence');
N1=length(g);
N2=length(h);
N=max(N1,N2);
N3=N1-N2
%loop for getting equal length sequence
if(N3>=0)
h=[h,zeros(1,N3)];
else
g=[g,zeros(1,-N3)];
end
%computation of circular convoluted sequence
for n=1:N;
y(n)=0;
for i=1:N;
j=n-i+1;
if(j<=0)
  j=N+j;
end
y(n)=y(n)+g(i)*h(j);
end
end
figure;
subplot(3,1,1);
stem(g);
ylabel('amplitude');
xlabel('n1..>');
title('input sequence')
subplot(3,1,2);
stem(h);
ylabel('amplitude');
xlabel('n2');
title('impulse sequence')
subplot(3,1,3);
stem(y);
ylabel('amplitude');
xlabel('n3');
disp('the resultant signal is');
```

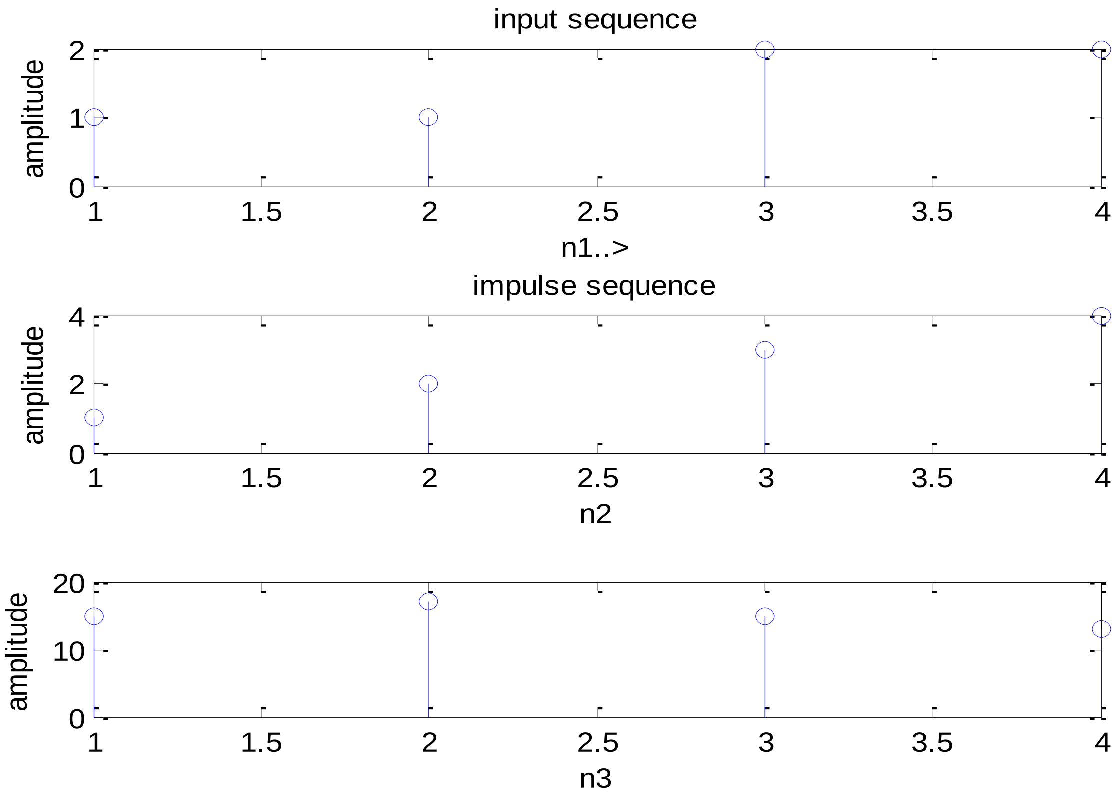
OUTPUT :   LINEAR CONVOLUTION

```
Enter the length of input sequence 4
Enter the length of impulse sequence 4
Enter the input sequence   [1 2 3 4]
Enter the impulse sequence   [4   3 2 1]

The resultant signal is
y=  4    11    20     30     20     11     4
```



OUTPUT : CIRCULAR CONVOLUTION

```
Enter the input sequence   [1   2    2   1]
Enter the impulse sequence   [4    3   2   1]

The resultant signal is
y=   15      17       15       13
```



RESULT:

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

Ex. No: 4
Date:

FREQUENCY ANALYSIS USING DFT

AIM:

To write MATLAB program for Frequency analyzing signal using DFT.

APPARATUS REQUIRED:

HARDWARE	: Personal Computer
SOFTWARE	: MATLAB

PROCEDURE:

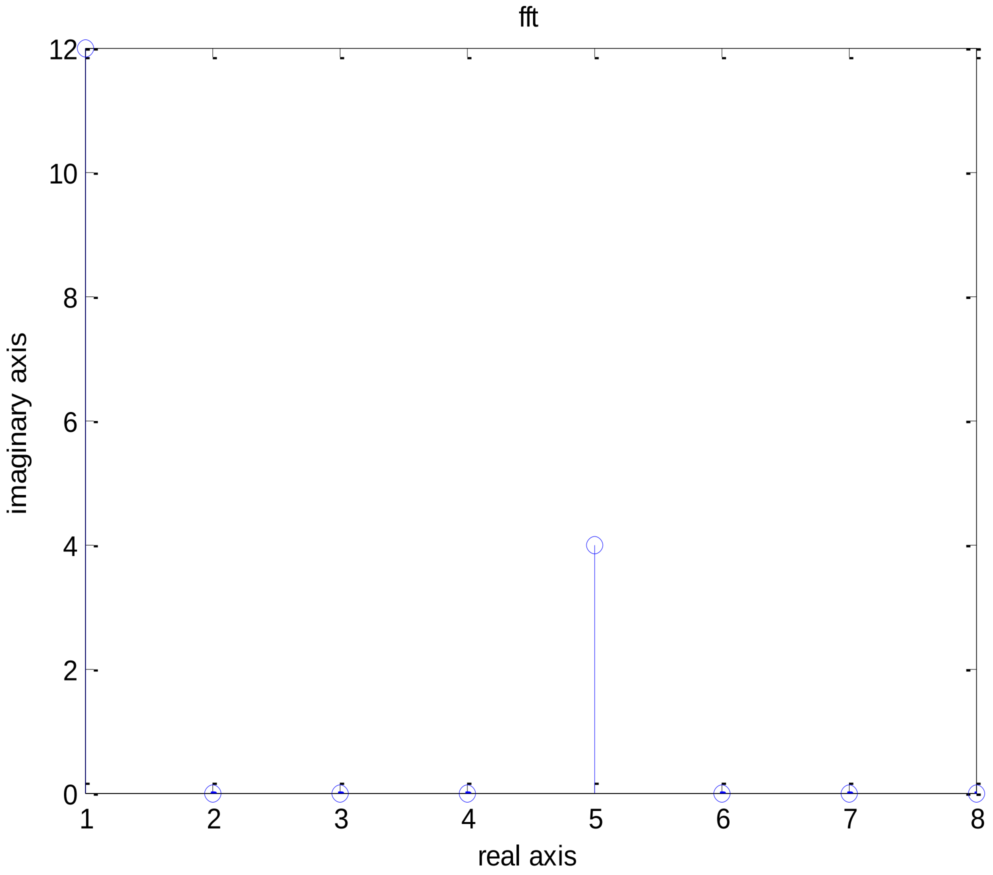
9. Start the MATLAB program.
10. Open new M-file
11. Type the program
12. Save in current directory
13. Compile and Run the program
14. If any error occurs in the program correct the error and run it again
15. For the output see command window\ Figure window
16. Stop the program.

PROGRAM: (COMPUTATION OF FFT OF A SIGNAL)

```
%program for computation of fft
Clear all;
Close all;
xn=input('enter the input sequence');
n=input ('enter the number of points in fft');
l=length (xn);
if (n<1)
   disp (n>=1);
end
xk=fft(xn,n);
stem(xk);
xlabel('real axis');
ylabel('imaginary axis');
title('fft');
disp('the values....');xk
```

OUTPUT: (FFT)

```
Enter the input sequence [2 1 2 1 2 1 2 1]
Enter the number of points in fft  8
The values ….
Xk= 12    0    0    0    4    0    0    0
```



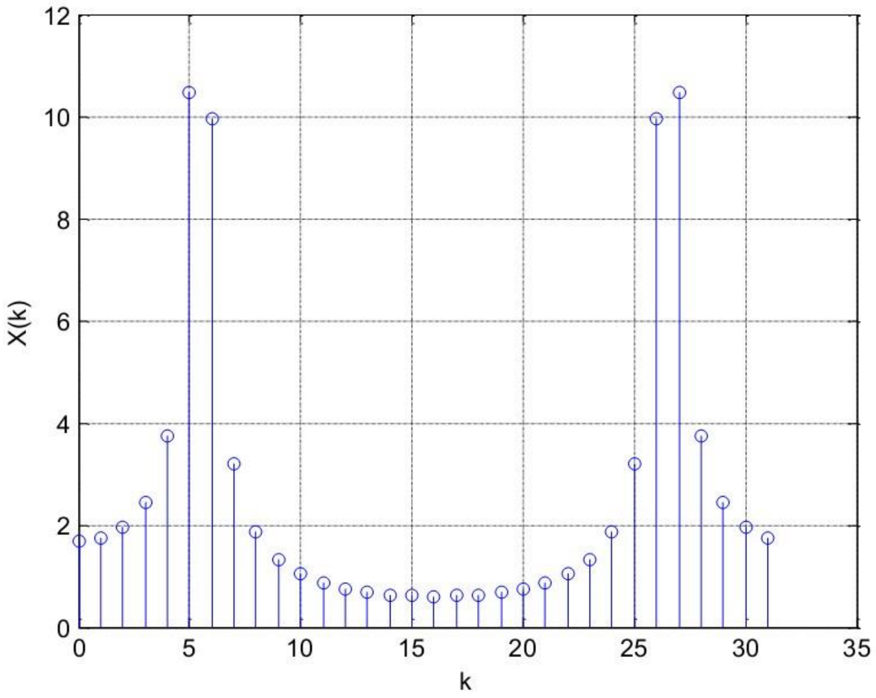
PROGRAM: (Spectrum Analysis Using DFT)

```
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 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.

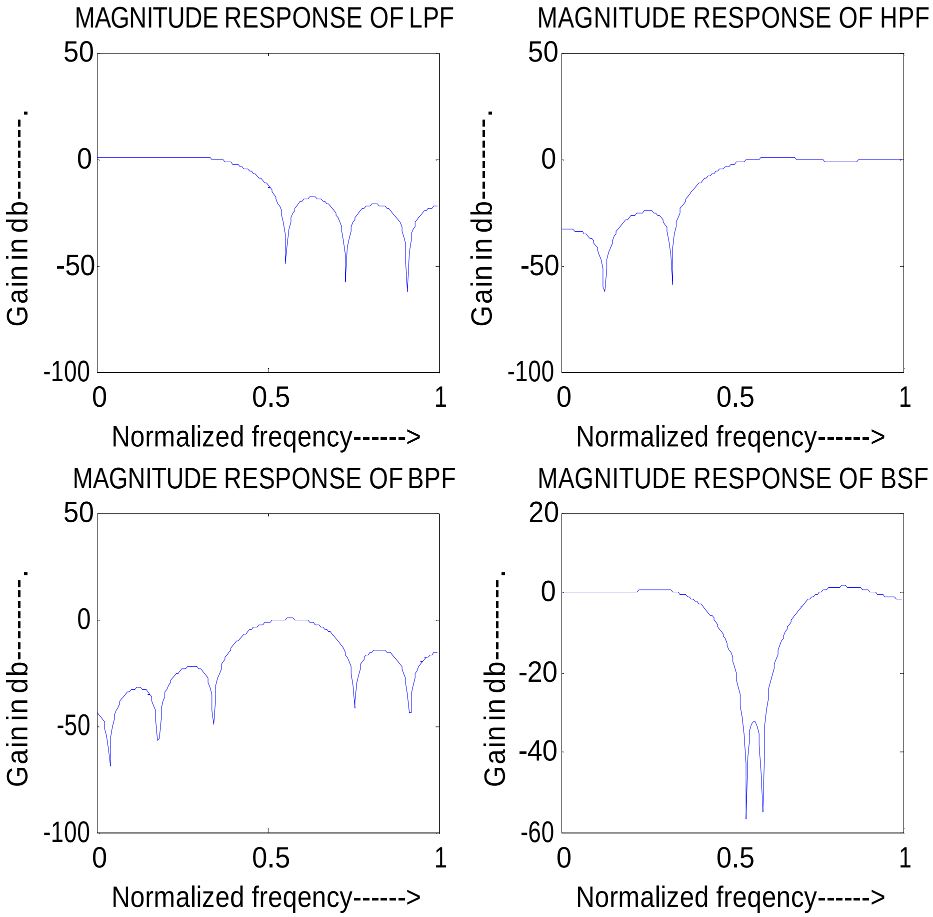
PROGRAM: (Rectangular Window)

```
%program for the design of FIR low pass, high pass, band pass and band stop filter using rectangular window
clc;
clear all;
close all;
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter the passband frequency');
fs=input('enter the stopband frequency');
f=input('enter the sampling frequency');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
    n1=n;
    n=n-1;
end;
y=boxcar(n1);
%lowpass filter
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('gain in db....>');
xlabel('(a)normalized frequency.....>');

%highpass filter
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);
ylabel('gain in db......>');
xlabel('(b)normalized frequency......>');
%bandpass filter
wn=[wp ws];0
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('gain in db....>');
xlabel('(c)normalized frequency....>');
%bandstop filter
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);
ylabel('gain in db....>');
xlabel('(d)normalized frequency.....>');
```

OUTPUT: (Rectangular Window)

```
Enter the pass band ripple 0.03
Enter the stop band ripple 0.01
Enter the pass band frequency 1400
Enter the stop band frequency 2000
Enter the sampling frequency 8000
```



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 R2014a

PROCEDURE:

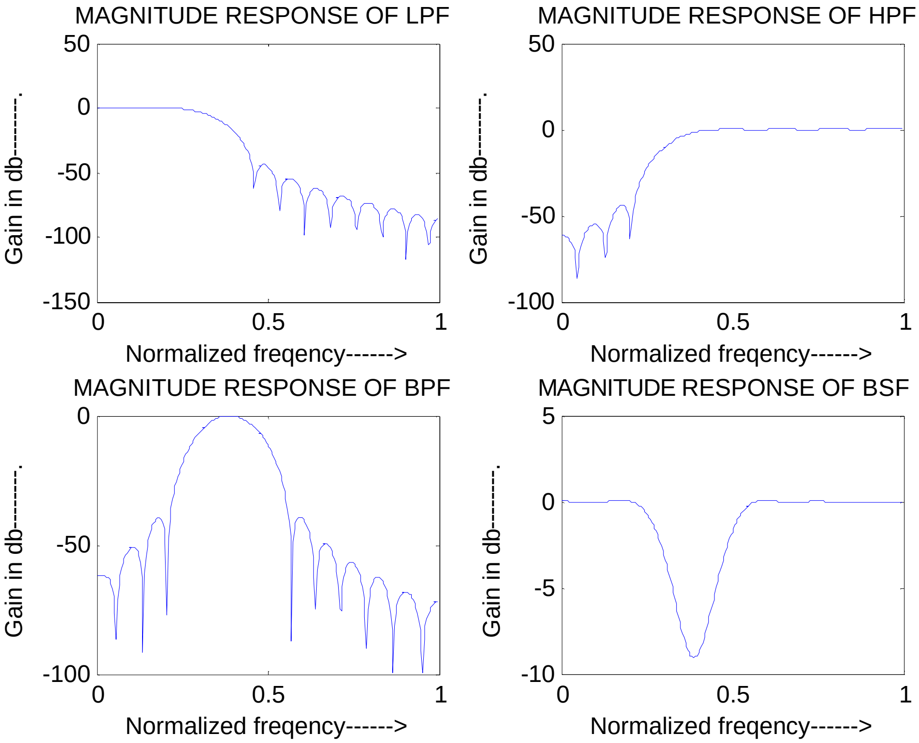
1. Start the MATLAB program.
2. Open new M-file
3. Type the program
4. Save in current directory
5. Compile and Run the program
6. If any error occurs in the program correct the error and run it again
7. For the output see command window\ Figure window
8. Stop the program.

PROGRAM: (Hanning Window)

```
%program for the design of FIR lowpass, high pass, band pass, band stop filter
using hanning window
clc;
clear all;
close all;
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter the passband frequency');
fs=input('enter the stopband frequency');
f=input('enter the sampling frequency');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
    n1=n;
    n=n-1;
end;
Y=hanning(n1);

%lowpass filter
b=fir1(n,wp,Y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('gain in db....>');
xlabel('(a)normalized frequency');

%highpass filter
 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);
ylabel('gain in db...>');
xlabel('(b)normalized frequency...>');

%bandpass filter
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);
ylabel('gain in db.....>');
xlabel('(c)normalized frequency....>');
```

```
%bandstop filter
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);
ylabel('gain in db...>');
xlabel('(d)normalized frequency....>')
```

```
FIR : ( HAMMING  WINDOW)

Enter the pass band ripple 0.03
Enter the stop band ripple 0.01
Enter the pass band frequency 1400
Enter the stop band frequency 2000
Enter the sampling frequency 8000
```



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 Butterworth & Chebyshew Filter (LPF/HPF/BPF/BSF) 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.

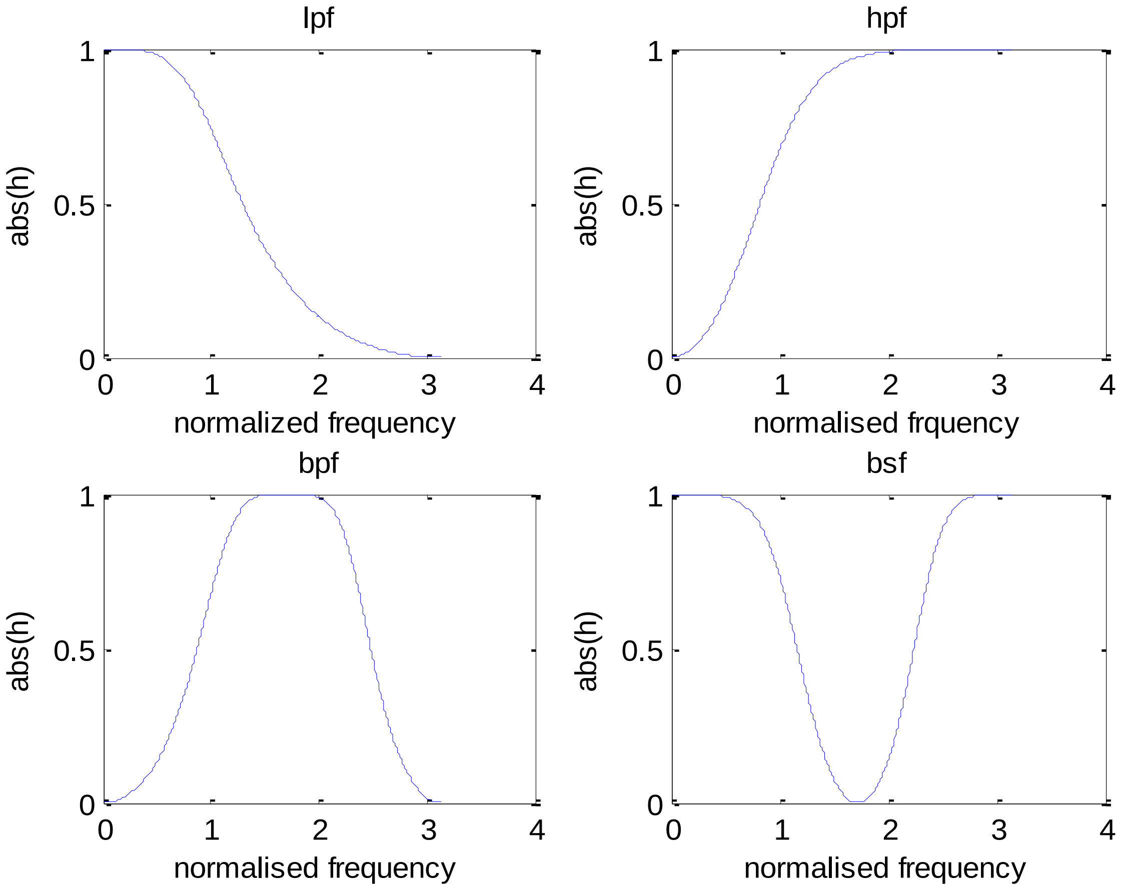
PROGRAM: (IIR Butterworth Filter)

```
PROGRAMS:   IIR (BUTTERWORTH FILTER)
% Butterworth filter
% get  the input values
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband frequency');
ws=input('enter the stopband frequency');
fs=input('enter the sampling frequency');
w1=2*wp/fs;
w2=2*ws/fs;
%filter order
[n,wn]=buttord(w1,w2,rp,rs);
%lowpass filter
%either coefficient
[b,a]=butter(n,wn);
%frequency response
```

```
[h,w]=freqz(b,a,512);
subplot(2,2,1);
plot(w,abs(h));
xlabel('normalized frequency');
ylabel('abs(h)');
title('lpf')

%high pass filter
%filter coefficient
[b,a]=butter (n,wn,'high');
%frequency response
[h,w]=freqz(b,a,512);
subplot(2,2,2);
plot(w,abs(h));
xlabel('normalised frquency');
ylabel('abs(h)');
title('hpf')

%band pass filter
%filter coefficient
wn1=[w1 w2];
[b,a]=butter(n,wn1);
%frequency response
[h,w]=freqz(b,a,512);
subplot(2,2,3);
plot(w,abs(h));
xlabel('normalised frequency');
ylabel('abs(h)');
title('bpf')

%band pass filter
%filter coefficient
wn2=[w1 w2];
[b,a]=butter(n,wn2,'stop');
%frequency response
[h,w]=freqz(b,a,512);
subplot(2,2,4);
plot(w,abs(h));
xlabel('normalised frequency');
ylabel('abs(h)');
title('bsf')
```

IIR : (BUTTERWORTH FILTER)

```
Enter the pass band ripple 6
Enter the stop band ripple 25
Enter the pass band frequency 1300
Enter the stop band frequency 3000
Enter the sampling frequency 8000
```

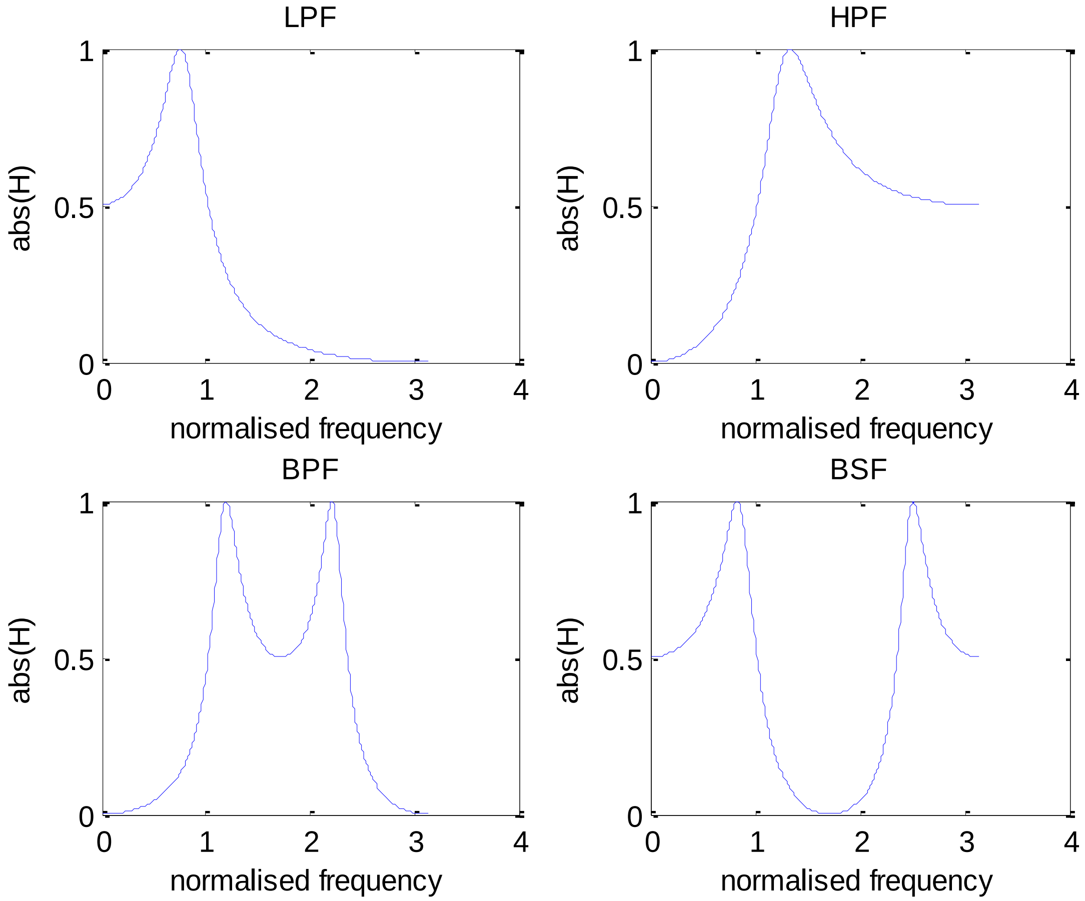


PROGRAMS: IIR (CHEBYSHEW FILTER)

```
%  chebyshew   filter
% get  the input values
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband frequency');
ws=input('enter the stopband frequency');
fs=input('enter the sampling frequency');
w1=2*wp/fs;
w2=2*ws/fs;
%filter order
 [n,wn]=cheb1ord(w1,w2,rp,rs);
%lowpass filter
%either coefficient
[b,a]=cheby1(n,rp,wn);
%frequency response
[H,w]=freqz(b,a,512);
subplot(2,2,1);
plot(w,abs(H));
xlabel('normalised frequency');
ylabel('abs(H)');
title('LPF')
%high pass filter
%filter coefficient
[b,a]=cheby1(n,rp,wn,'High');
%frequency response
[H,w]=freqz(b,a,512);
subplot(2,2,2);
plot(w,abs(H));
xlabel('normalised frequency');
ylabel('abs(H)');
title('HPF')
%band pass filter
%filter coefficient
wn1=[w1,w2];
[b,a]=cheby1(n,rp,wn1);
%frequency response
[H,w]=freqz(b,a,512);
subplot(2,2,3);
plot(w,abs(H));
xlabel('normalised frequency');
ylabel('abs(H)');
title('BPF')
%band stop filter
%filter coefficient
wn2= [w1, w2];
%frequency response
[b,a]=cheby1(n,rp,wn2,'stop');
[H,w]=freqz(b,a,512);
subplot(2,2,4);
```

```
plot(w,abs(H));
xlabel('normalised frequency');
ylabel('abs(H)');
title('BSF')
```

```
 IIR : (CHEBYSHEW  FILTER)
Enter the pass band ripple 6
Enter the stop band ripple 25
Enter the pass band frequency 1300
Enter the stop band frequency 3000
Enter the sampling frequency 8000
```



RESULT:

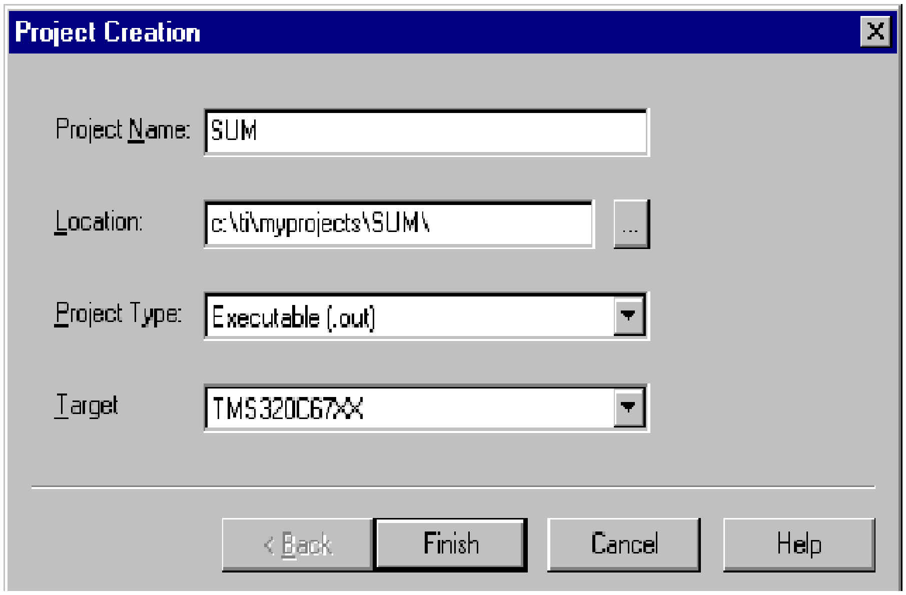
Thus the program to design IIR Butterworth & Chebyshew Filter (LPF/HPF/BPF/BSF) by using MATLAB was executed.

DSP PROCESSOR EXPERIMENTS

PROCEDURE TO WORK ON CODE COMPOSER STUDIO

1. To create a New Project

Project →New (SUM.pjt)



Project Creation

Project Name: SUM

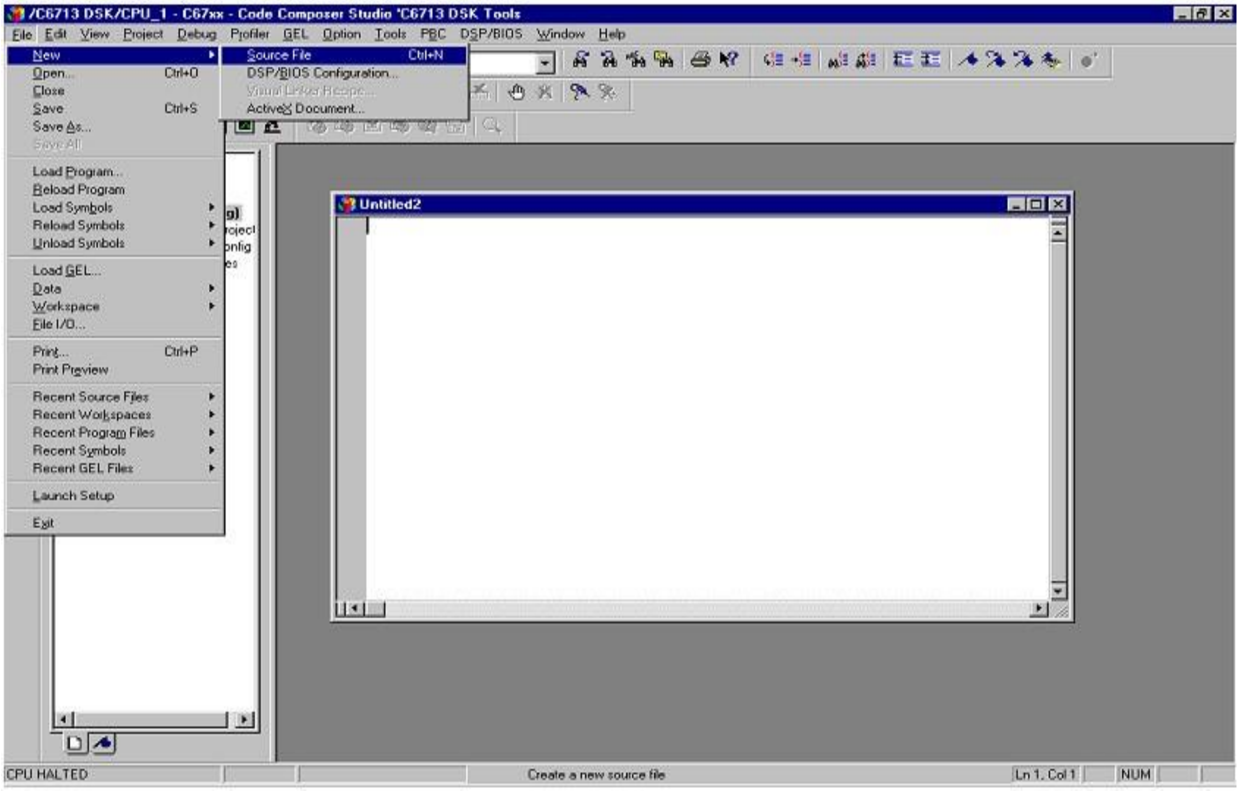
Location: c:\ti\myprojects\SUM\

Project Type: Executable (.out)

Target: TMS320C67XX

2. To Create a Source file

File → New



/C6713 DSK/CPU_1 - C67xx - Code Composer Studio 'C6713 DSK Tools

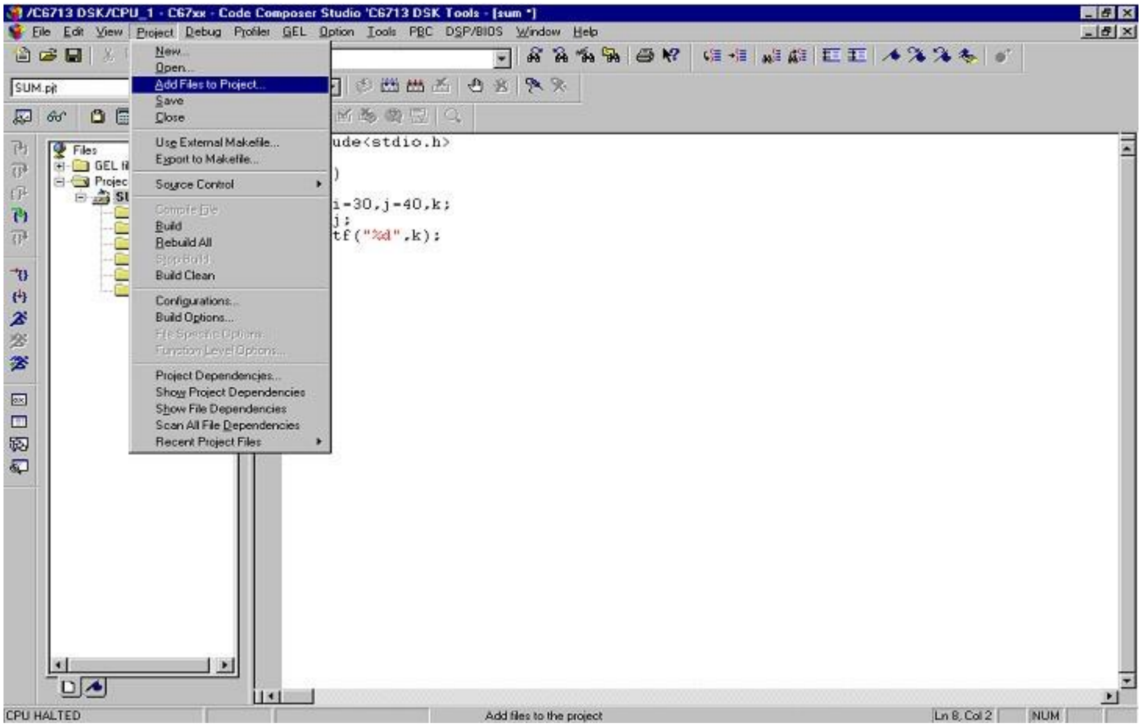
Untitled2

CPU HALTED | Create a new source file | Ln 1, Col 1 | NUM

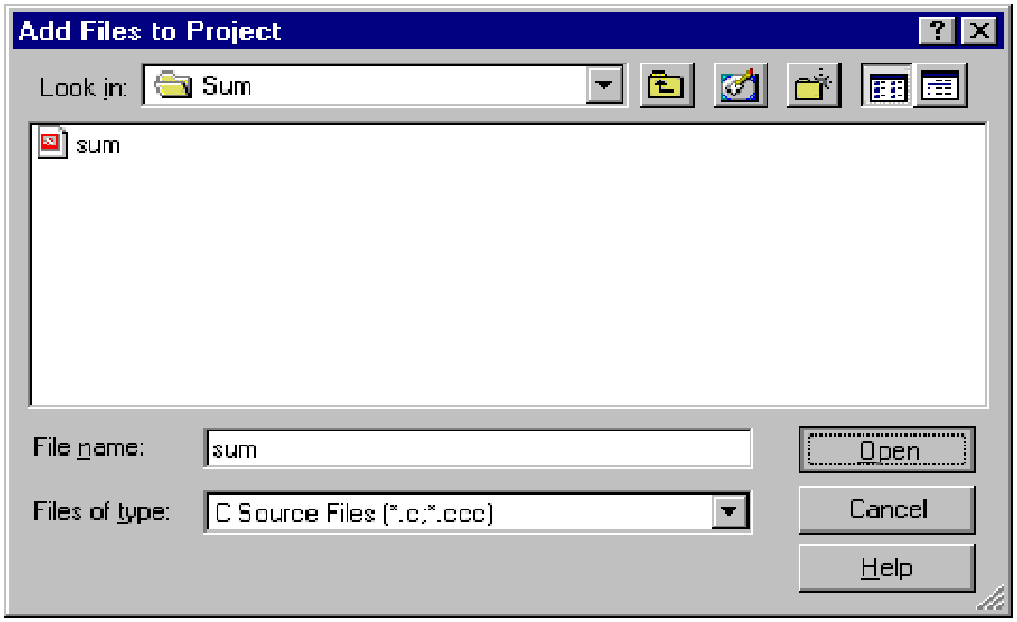
Type the code (Save & give a name to file, Eg: sum.c).

3. To Add Source files to Project

Project → Add files to Project → sum.c



```
ude<stdio.h>

)

i=30,j=40,k;
j;
tf("%d",k);
```



Look in: Sum

File name: sum

Files of type: C Source Files (*.c;*.ccc)

4. To Add rts6700.lib file & hello.cmd:

Project →Add files to Project →rts6700.lib

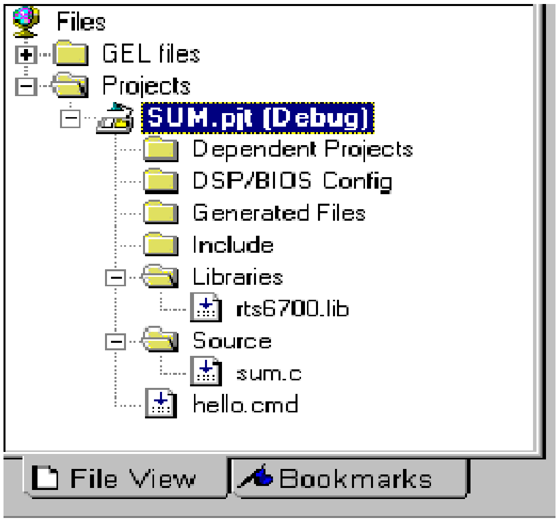
Path: c:\CCStudio\c6000\cgtools\lib\rts6700.lib

Note: Select Object & Library in(*.o,*.l) in Type of files

Project →Add files to Project →hello.cmd

Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd

Note: Select Linker Command file(*.cmd) in Type of files



5. To Compile:

Project → Compile File

6. To build or Link:

Project → build,

Which will create the final executable (.out) file.(Eg. sum.out).

7. Procedure to Load and Run program:

Load program to DSK:

File →Load program →sum. Out

8. To execute project:

Debug →Run.

Ex. No: 7

Date:

STUDY OF ARCHITECTURE OF DIGITAL SIGNAL PROCESSOR

AIM:

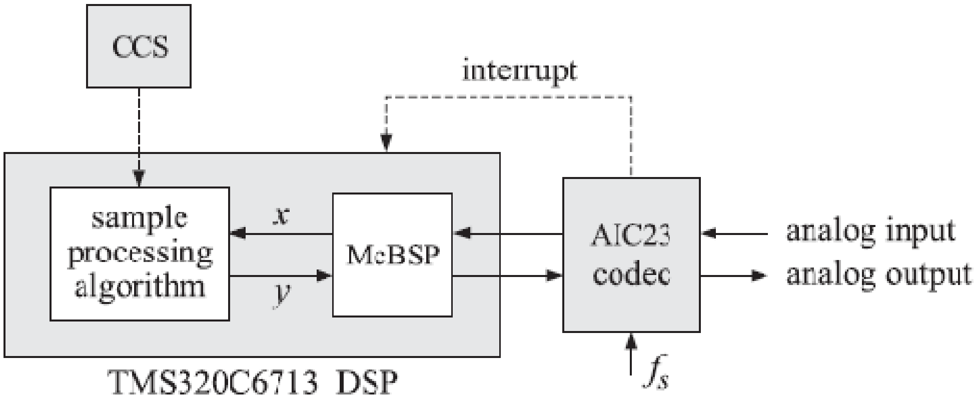
To study the Architecture of TMS320VC67XX DSP processor.

INTRODUCTION

The hardware experiments in the DSP lab are carried out on the Texas Instruments TMS320C6713 DSP Starter Kit (DSK), based on the TMS320C6713 floating point DSP running at 225MHz. The basic clock cycle instruction time is 1/(225 MHz)= 4.44 nanoseconds. During each clock cycle, up to eight instructions can be carried out in parallel, achieving up to 8×225 = 1800 million instructions per second (MIPS). The DSK board includes a 16MB SDRAM memory and a 512KB Flash ROM. It has an on-board 16-bit audio stereo codec (the Texas Instruments AIC23B) that serves both as an A/D and a D/A converter. There are four 3.5 mm audio jacks for microphone and stereo line input, and speaker and headphone outputs. The AIC23 codec can be programmed to sample audio inputs at the following sampling rates: f_s = 8, 16, 24, 32, 44.1, 48, 96 kHz

The ADC part of the codec is implemented as a multi-bit third-order noise-shaping delta-sigma converter) that allows a variety of oversampling ratios that can realize the above choices of fs. The corresponding oversampling decimation filters act as anti-aliasing pre-filters that limit the spectrum of the input analog signals effectively to the Nyquist interval [−fs/2,fs/2]. The DAC part is similarly implemented as a multi-bit second-order noise-shaping delta-sigma converter whose oversampling interpolation filters act as almost ideal reconstruction filters with the Nyquist interval as their pass band.

The DSK also has four user-programmable DIP switches and four LEDs that can be used to control and monitor programs running on the DSP. All features of the DSK are managed by the Code Composer Studio (CCS). The CCS is a complete integrated development environment (IDE) that includes an optimizing C/C++ compiler, assembler, linker, debugger, and program loader. The CCS communicates with the DSK via a USB connection to a PC. In addition to facilitating all programming aspects of the C6713 DSP, the CCS can also read signals stored on the DSP‐‐s memory, or the SDRAM, and plot them in the time or frequency domains. The following block diagram depicts the overall operations involved in all of the hardware experiments in the DSP lab. Processing is interrupt-driven at the sampling rate fs, as explained below.

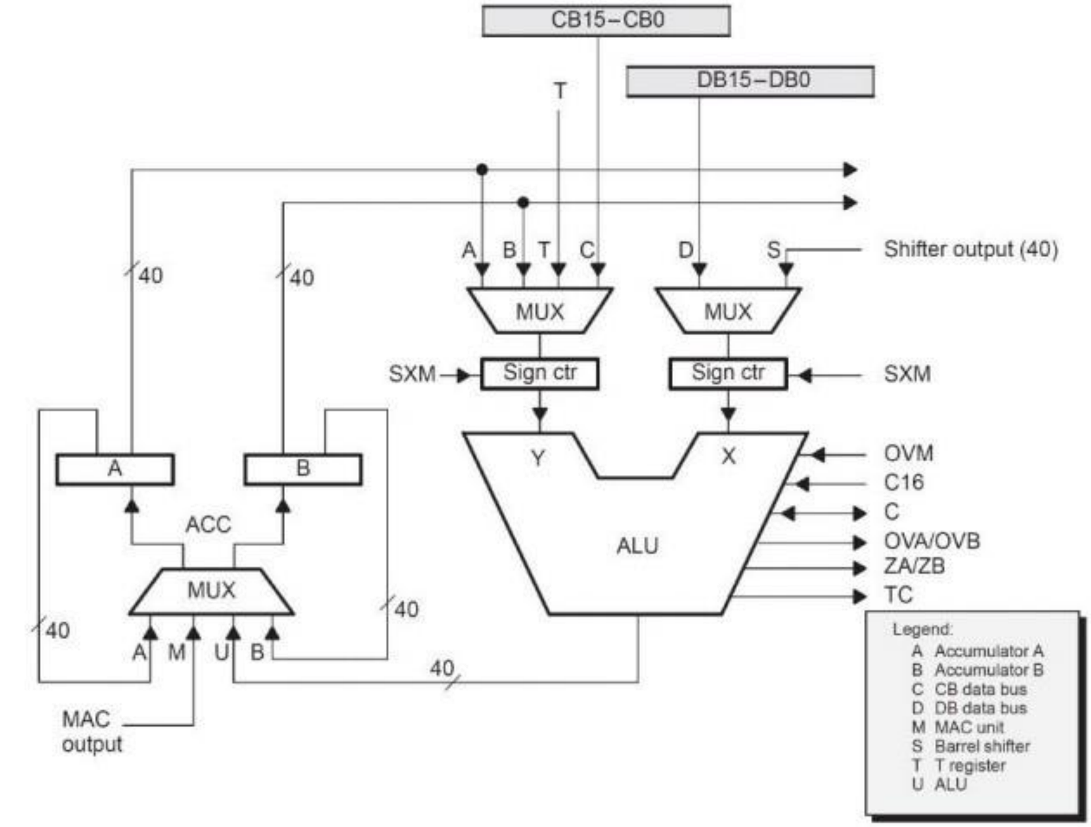


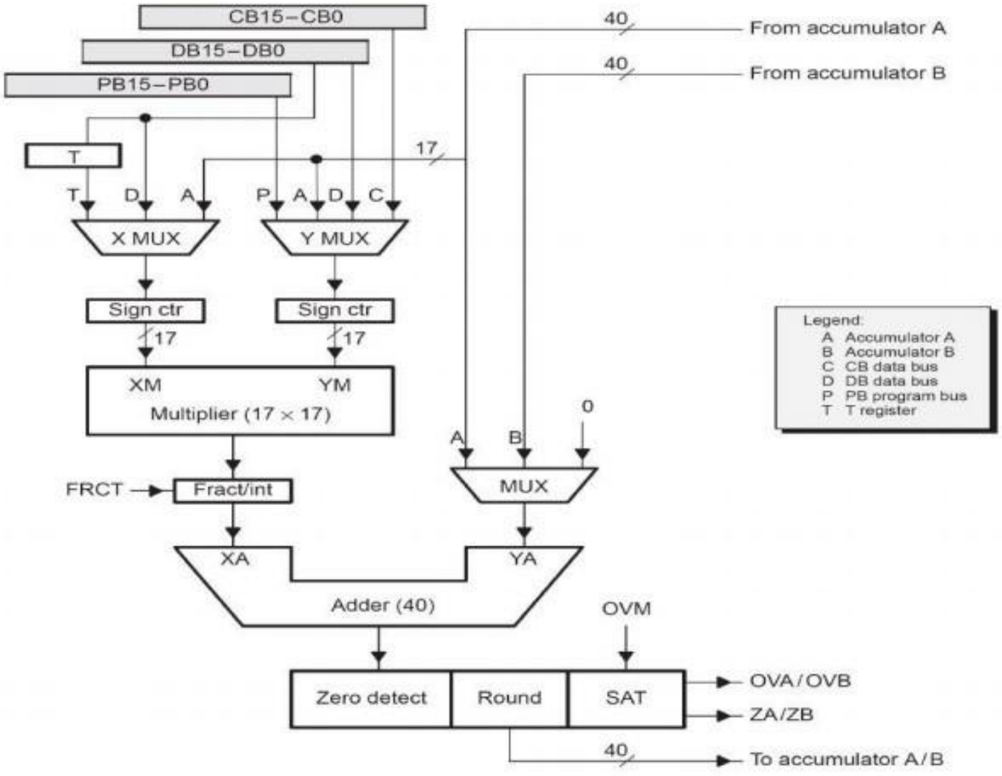
TMS320C6713 floating point DSP

The AIC23 codec is configured (through CCS) to operate at one of the above sampling rates fs. Each collected sample is converted to a 16-bit two's complement integer (a **short** data type in C). The codec actually samples the audio input in stereo, that is, it collects two samples for the left and right channels

ARCHITECTURE

The 67XX DSPs use an advanced, modified Harvard architecture that maximizes processing power by maintaining one program memory bus and three data memory buses. These processors also provide an arithmetic logic unit (ALU) that has a high degree of parallelism, application-specific hardware logic, on-chip memory, and additional on-chip peripherals. These DSP families also provide a highly specialized instruction set, which is the basis of the operational flexibility and speed of these DSPs. Separate program and data spaces allow simultaneous access to program instructions and data, providing the high degree of parallelism. Two reads and one write operation can be performed in a single cycle. Instructions with parallel store and application-specific instructions can fully utilize this architecture. In addition, data can be transferred between data and program spaces. Such parallelism supports a powerful set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. Also included are the control mechanisms to manage interrupts, repeated operations, and function calls.


CB15–CB0
T
DB15–DB0
40
40
A
B
T
C
D
S
Shifter output (40)
MUX
MUX
SXM
Sign ctr
Sign ctr
SXM
Y
X
OVM
C16
C
OVA/OVB
ZA/ZB
TC
ALU
A
B
ACC
MUX
40
40
A
M
U
B
40
MAC output
Legend:
A Accumulator A
B Accumulator B
C CB data bus
D DB data bus
M MAC unit
S Barrel shifter
T T register
U ALU


CB15–CB0
DB15–DB0
PB15–PB0
40
From accumulator A
40
From accumulator B
T
17
T
D
A
P
A
D
C
X MUX
Y MUX
Sign ctr
Sign ctr
17
17
XM
YM
Multiplier (17 × 17)
0
A
B
FRCT
Fract/int
MUX
XA
YA
Adder (40)
OVM
Zero detect
Round
SAT
OVA/OVB
ZA/ZB
40
To accumulator A/B
Legend:
A Accumulator A
B Accumulator B
C CB data bus
D DB data bus
P PB program bus
T T register

1. **Central Processing Unit (CPU)**

The CPU of the 67XX devices contains:

- A 40-bit arithmetic logic unit (ALU)
- Two 40-bit accumulators
- A barrel shifter
- A 17 -bit multiplier/adder
- A compare, select, and store unit (CSSU)

2. **Arithmetic Logic Unit (ALU)**

The 67XX devices perform 2s-complement arithmetic using a 40-bit ALU and two 40-bit accumulators (ACCA and ACCB). The ALU also can perform Boolean operations. The ALU can function as two 16-bit ALUs and perform two 16-bit operations simultaneously when the C16 bit in status register 1 (ST1) is set.

3. **Accumulators**

The accumulators, ACCA and ACCB, store the output from the ALU or the multiplier / adder block; the accumulators can also provide a second input to the ALU or the multiplier / adder. The bits in each accumulator are grouped as follows:

- Guard bits (bits 32–39)
- A high-order word (bits 16–31)
- A low-order word (bits 0–15)

Instructions are provided for storing the guard bits, the high-order and the low-order accumulator words in data memory, and for manipulating 32-bit accumulator words in or out of data memory. Also, any of the accumulators can be used as temporary storage for the other.

4. **Barrel Shifter**

The 67XXs barrel shifter has a 40-bit input connected to the accumulator or data memory (CB, DB) and a 40-bit output connected to the ALU or data memory (EB). The barrel shifter produces a left shift of 0 to 31 bits and a right shift of 0 to 16 bits on the input data. The shift requirements are defined in the shift-count field (ASM) of ST1 or defined in the temporary register (TREG), which is designated as a shift-count register. This shifter and the exponent detector normalize the values in an accumulator in a single cycle. The least significant bits (LSBs) of the output are filled with 0s and the most significant bits (MSBs) can be either zero-filled or sign-extended, depending on the state of the sign-extended mode bit (SXM) of ST1.

Additional shift capabilities enable the processor to perform numerical scaling, bit extraction, extended arithmetic, and overflow prevention operations

5. **Multiplier/Adder**

   The multiplier / adder perform 17-bit 2s-complement multiplication with a 40-bit accumulation in a single instruction cycle. The multiplier / adder block consists of several elements: a multiplier, adder, signed/unsigned input control, fractional control, a zero detector, a rounder (2s-complement), overflow/saturation logic, and TREG. The multiplier has two inputs: one input is selected from the TREG, a data memory operand, or an accumulator; the other is selected from the program memory, the data memory, an accumulator, or an immediate value. The fast on-chip multiplier allows the C67XX to perform operations such as convolution, correlation, and filtering efficiently. In addition, the multiplier and ALU together execute multiply/accumulate (MAC) computations and ALU operations in parallel in a single instruction cycle. This function is used in determining the Euclid distance, and in implementing symmetrical and least mean square (LMS) filters, which are required for complex DSP algorithms.

6. **Compare, Select, and Store Unit (CSSU)**

   The compare, select, and store unit (CSSU) performs maximum comparisons between the accumulator's high and low words, allows the test/control (TC) flag bit of status register 0 (ST0) and the transition (TRN) register to keep their transition histories, and selects the larger word in the accumulator to be stored in data memory. The CSSU also accelerates Viterbi-type butterfly computation with optimized on-chip hardware.

7. **Program Control**

   <u>Program control is provided by several hardware and software mechanisms:</u>

   The program controller decodes instructions, manages the pipeline, stores the status of operations, and decodes conditional operations. Some of the hardware elements included in the program controller are the program counter, the status and control register, the stack, and the address-generation logic.

   Some of the software mechanisms used for program control includes branches, calls, and conditional instructions, are peat instruction, reset, and interrupt.

   The C67XX supports both the use of hardware and software interrupts for program control. Interrupt service routines are vectored through a re-locatable interrupt vector table. Interrupts can be globally enabled / disabled and can be individually masked through the interrupt mask register (IMR). Pending interrupts are indicated in the interrupt flag register (IFR). For detailed information on the structure of the interrupt vector table, the IMR and the IFR, see the device-specific data sheets.

8. **Status Registers (ST0, ST1)**

   The status registers, ST0 and ST1, contain the status of the various conditions and modes for the 67XX devices. ST0 contains the flags (OV, C, and TC) produced by arithmetic operations and bit manipulations in addition to the data page pointer (DP) and the auxiliary register pointer (ARP) fields. ST1 contains the various modes and instructions that the processor operates on and executes.

9. **Auxiliary Registers (AR0–AR7)**

   The eight 16-bit auxiliary registers (AR0–AR7) can be accessed by the central arithmetic logic unit (CALU) and modified by the auxiliary register arithmetic units (ARAUs). The primary function of the auxiliary registers is generating 16-bit addresses for data space. However, these registers also can act as general-purpose registers or counters.

10. **Temporary Register (TREG)**

    The TREG is used to hold one of the multiplicands for multiply and multiply/accumulate instructions. It can hold a dynamic (execution-time programmable) shift count for instructions with a shift operation such as ADD, LD, and SUB. It also can hold a dynamic bit address for the BITT instruction. The EXP instruction stores the exponent value computed into the TREG, while the NORM instruction uses the TREG value to normalize the number. For ACS operation of Viterbi decoding, TREG holds branch metrics used by the DADST and DSADT instructions.

11. **Transition Register (TRN)**

    The TRN is a 16-bit register that is used to hold the transition decision for the path to new metrics to perform the Viterbi algorithm. The CMPS (compare, select, max, and store) instruction updates the contents of the TRN based on the comparison between the accumulator high word and the accumulator low word.

12. **Stack-Pointer Register (SP)**

    The SP is a 16-bit register that contains the address at the top of the system stack. The SP always points to the last element pushed onto the stack. The stack is manipulated by interrupts, traps, calls, returns, and the PUSHD, PSHM, POPD, and POPM instructions. Pushes and pops of the stack pre decrement and post increment, respectively, all 16 bits of the SP.

13. **Circular-Buffer-Size Register (BK)**

The 16-bit BK is used by the ARAUs in circular addressing to specify the data block size.

14. **Block-Repeat Registers (BRC, RSA, REA)**

The block-repeat counter (BRC) is a 16-bit register used to specify the number of times a block of code is to be repeated when performing a block repeat. The block-repeat start address (RSA) is a 16-bit register containing the starting address of the block of program memory to be repeated when operating in the repeat mode. The 16-bit block-repeat end address (REA) contains the ending address if the block of program memory is to be repeated when operating in the repeat mode.

15. **Interrupt Registers (IMR, IFR)**

The interrupt-mask register (IMR) is used to mask off specific interrupts individually at required times. The interrupt-flag register (IFR) indicates the current status of the interrupts.

16. **Processor-Mode Status Register (PMST)**

The processor-mode status registers (PMST) controls memory configurations of the 67XX devices.

17. **Power-Down Modes**

There are three power-down modes, activated by the IDLE1, IDLE2, and IDLE3 instructions. In these modes, the 67XX devices enter a dormant state and dissipate considerably less power than in normal operation. The IDLE1instruction is used to shut down the CPU. The IDLE2 instruction is used to shut down the CPU and on-chip peripherals. The IDLE3 instruction is used to shut down the 67XX processor completely. This instruction stops the PLL circuitry as well as the CPU and peripherals.

RESULT

Thus the study of architecture TMS320VC67XX and its functionalities has been identified.

Ex. No: 8

Date:

MAC OPERATION USING VARIOUS ADDRESSING MODES

AIM:

To Study the various addressing modes of TMS320C67XX DSP processor.

THEORY:

Addressing Modes The TMS320C67XX DSP supports three types of addressing modes that enable flexible access to data memory, to memory-mapped registers, to register bits, and to I/O space:

The absolute addressing mode allows you to reference a location by supplying all or part of an address as a constant in an instruction.

The direct addressing mode allows you to reference a location using an address offset.

The indirect addressing mode allows you to reference a location using a pointer.

Each addressing mode provides one or more types of operands. An instruction that supports an addressing-mode operand has one of the following syntax elements listed below.

Baddr

When an instruction contains Baddr, that instruction can access one or two bits in an accumulator (AC0–AC3), an auxiliary register (AR0–AR7), or a temporary register (T0–T3). Only the register bit test/set/clear/complement instructions support Baddr. As you write one of these instructions, replace Baddr with a compatible operand.

Cmem

When an instruction contains Cmem, that instruction can access a single word (16 bits)of data from data memory. As you write the instruction, replace Cmem with a compatible operand.

Lmem

When an instruction contains Lmem, that instruction can access a long word (32 bits) of data from data memory or from a memory-mapped registers. As you write the instruction, replace Lmem with a compatible operand.

Smem

When an instruction contains Smem, that instruction can access a single word (16 bits) of data from data memory, from I/O space, or from a memory-mapped register. As you write the instruction, replace Smem with a compatible operand.

Xmem and Ymem

When an instruction contains Xmem and Ymem, that instruction can perform two simultaneous 16-bit accesses to data memory. As you write the instruction, replace Xmem and Ymem with compatible operands.

Absolute Addressing Modes k16 absolute

This mode uses the 7-bit register called DPH (high part of the extended data page register) and a 16-bit unsigned constant to form a 23-bit data space address. This mode is used to access a memory location or a memory-mapped register.

k23 absolute

This mode enables you to specify a full address as a 23-bit unsigned constant. This mode is used to access a memory location or a memory-mapped register.

I/O absolute

This mode enables you to specify an I/O address as a 16-bit unsigned constant. This mode is used to access a location in I/O space.

Direct Addressing Modes DP direct

This mode uses the main data page specified by DPH (high part of the extended data page register) in conjunction with the data page register (DP).This mode is used to access a memory location or a memory-mapped register.

SP direct

This mode uses the main data page specified by SPH (high part of the extended stack pointers) in conjunction with the data stack pointer (SP). This mode is used to access stack values in data memory.

Register-bit direct

This mode uses an offset to specify a bit address. This mode is used to access one register bit or two adjacent register bits.

PDP direct

This mode uses the peripheral data page register (PDP) and an offset to specify an I/O address. This mode is used to access a location in I/O space. The DP direct and SP direct addressing modes are mutually exclusive. The mode selected depends on the CPL bit in status register ST1_67: 0 DP direct addressing mode 1 SP direct addressing mode The register-bit and PDP direct addressing modes are independent of the CPL bit.

Indirect Addressing Modes

You may use these modes for linear addressing or circular addressing.

AR indirect

This mode uses one of eight auxiliary registers (AR0–AR7) to point to data. The way the CPU uses the auxiliary register to generate an address depends on whether you are accessing data space (memory or memory-mapped registers), individual register bits, or I/O space.

Dual AR indirect

This mode uses the same address-generation process as the AR indirect addressing mode. This mode is used with instructions that access two or more data-memory locations.

CDP indirect

This mode uses the coefficient data pointer (CDP) to point to data. The way the CPU uses CDP to generate an address depends on whether you are accessing data space (memory or memory-mapped registers), individual register bits, or I/O space.

Coefficient indirect

This mode uses the same address-generation process as the CDP indirect addressing mode. This mode is available to support instructions that can access a coefficient in data memory at the same time they access two other data-memory values using the dual AR indirect addressing mode.

Circular Addressing

Circular addressing can be used with any of the indirect addressing modes. Each of the eight auxiliary registers (AR0–AR7) and the coefficient data pointer (CDP) can be independently configured to be linearly or circularly modified as they act as pointers to data or to register bits, see Table 3−10. This configuration is done with a bit (ARnLC) in status register ST2_67. To choose circular modification, set the bit. Each auxiliary register ARn has its own linear/circular configuration bit in ST2_67: 0 Linear addressing 1 Circular addressing The CDPLC bit in status register ST2_67 configures the DSP to use CDP for linear addressing or circular addressing: 0 Linear addressing 1 Circular addressing You can use the circular addressing instruction qualifier, .CR, if you want every pointer used by the instruction to be modified circularly, just add .CR to the end of the instruction mnemonic (for example, ADD.CR). The circular addressing instruction qualifier overrides the linear/circular configuration in ST2_67.

ADDITION:

```
INP1 .SET 0H
INP2 .SET 1H
OUT .SET 2H
     .mmregs
     .text
START:
     LD #140H,DP
     RSBX CPL
     NOP
     NOP
     NOP
     NOP
     LD INP1,A
     ADD INP2,A
     STL A,OUT
HLT: B HLT
```

INPUT:

Data Memory:

```
A000h 0004h
A001h 0004h
```

OUTPUT:

Data Memory:

```
A002h 0008h
```

SUBTRACTION:

```
INP1 .SET 0H
INP2 .SET 1H
OUT .SET 2H
     .mmregs
     .text
START:
     LD #140H,DP
     RSBX CPL
     NOP
     NOP
     NOP
     NOP
     LD INP1,A
     SUB INP2,A
     STL A,OUT
HLT: B HLT
```

INPUT:

Data Memory:

A000h 0004h
A001h 0002h

OUTPUT:

Data Memory:

A002h   0002h

MULTIPLICATION

```
 .mmregs
        .text
START:
        STM     #0140H,ST0
        STM     #40H,PMST
        STM     #0A000H,AR0
        ST      #1H,*AR0
        LD       *AR0+,T
        ST      #2H,*AR0
        MPY      *AR0+,A
        STL     A,*AR0
HLT:    B       HLT
        .END
```

OUTPUT

```
A002H  2H
```

DIVISION

```
DIVID           .SET    0H
DIVIS           .SET    1H
OUT             .SET    2H
                .mmregs
                .text
START:
        STM     #140H,ST0
        RSBX    CPL
        RSBX    FRCT
        NOP
        NOP
        NOP
        NOP
        LD      DIVID,A            ;dividend to acc
        RPT     #0FH
        SUBC    DIVIS,A            ;A / DIVIS -> A
        STL     A,OUT          ;result in 9002h
HLT:    B       HLT
        .END
```

INPUT

DATA MEMORY

```
       A000H          000AH
       A001H          0002H
```

OUTPUT

```
      A002H          0005H
```

RESULT:

Thus, the various addressing mode of DSP processor TMS320C67XX was studied

Ex. No: 9
Date:

WAVEFORM GENERATION

AIM:

To generate a sine wave and square wave using TMS320C67XX DSP KIT.

APPARATUS REQUIRED:

HARDWARE	: Personal Computer & TMS320C67XX kit
SOFTWARE	: VSK5416

PROCEDURE:

1. Open Code Composer Studio v4.
2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)
3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.asm).
4. To Add Source files to Project
   Project→ Add files to Project
   sum.asm

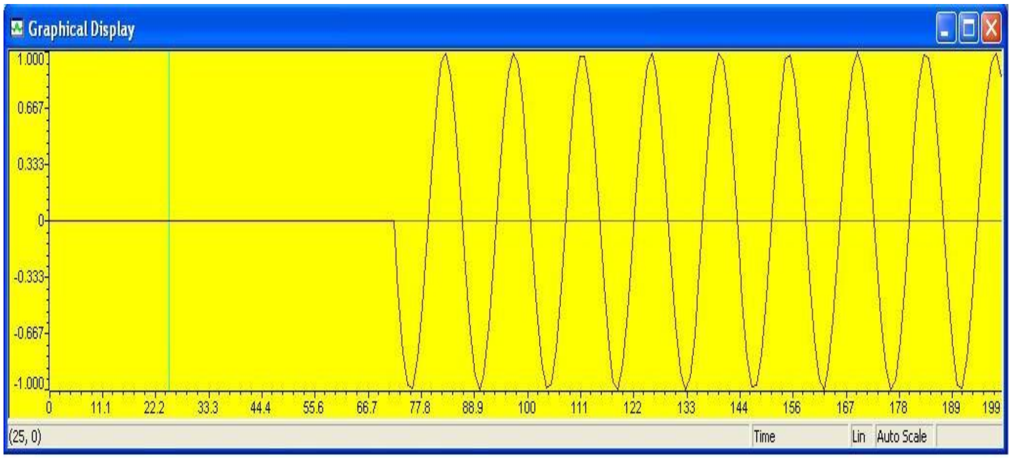
COMPILE:

1. To Compile: Project→ Compile
2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).
3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program
   →vvit.out To Execute project: Debug → Run

PROGRAM: (Sine waveform)

```
mmreg
   .text
START:
        STM     #140H,ST0
        RSBX    CPL
       NOP
        NOP
        NOP
        NOP
REP:
        LD      #TABLE,A
        STM     #372,AR1
LOOP:
        READA   DATA
        PORTW   DATA,04H
        ADD     #1H,A
      B  REP
                             }
                      }
```

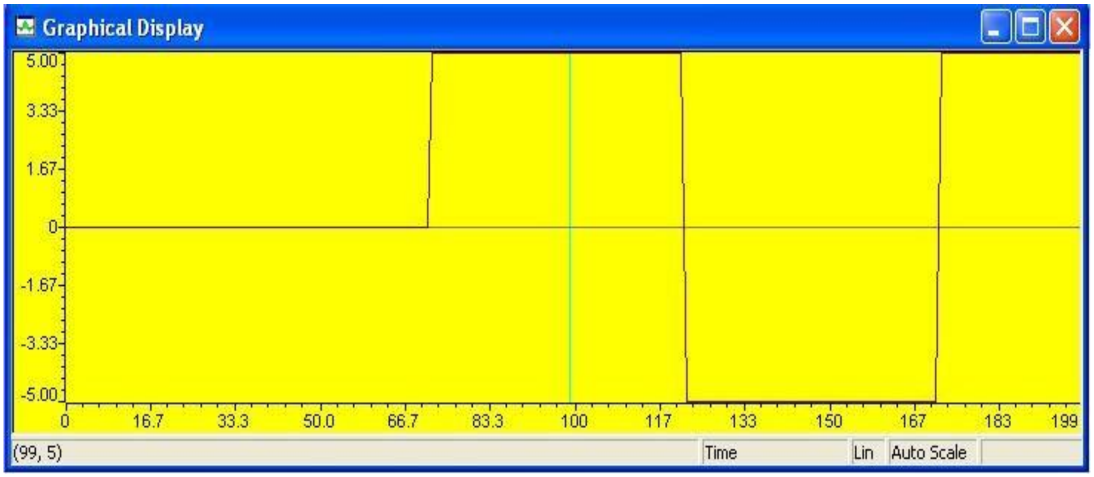
OUTPUT: (Sine waveform)



PROGRAM: (Square waveform)

```
DATA          .SET    0H
        .mmregs
        .text
START:
        STM     #140H,ST0
        RSBX    CPL           ;       NOP
        NOP
        NOP
        NOP
REP:
        ST      #0H,DATA
        CALL    DELAY
        ST      #0FFFH,DATA
        CALL    DELAY
        B       REP
DELAY:
        STM     #0FFFH,AR1
DEL1:
        PORTW   DATA,04H
        BANZ    DEL1,*AR1-
        RET
```

Output



TRIANGULAR WAVE GENERATION

```
DATA           .SET    0H
        .mmregs
        .text
START:
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP
REP:
        ST      #0H,DATA
INC:
        LD      DATA,A
        ADD     #1H,A
        STL     A,DATA
        PORTW   DATA,04H
        CMPM    DATA,#0FFFH
        BC      INC,NTC
DEC:
        LD      DATA,A
        SUB     #1H,A
        STL     A,DATA
        PORTW   DATA,04H
        CMPM    DATA,#0H
        BC      DEC,NTC
        B       REP
```

SAWTOOTH WAVE GENERATION

```
DATA          .SET    0H
        .mmregs
        .text
START:
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP
REP:
        ST      #0H,DATA
INC:
        LD      DATA,A
        ADD     #1H,A
        STL     A,DATA
        PORTW   DATA,04H
        CMPM    DATA,#0FFFH
        BC      INC,NTC
        B       REP
```

RESULT:

Thus, the sine wave ,square ,triangular and saw tooth waveform was generated displayed at graph.

Ex. No: 10
Date:

DESIGN OF FIR FILTERS

AIM:

To write a C program for the design of FIR Filter, also plots the magnitude responses for the same.

APPARATUS REQUIRED:

HARDWARE	: Personal Computer & TMS320C67XX kit
SOFTWARE	: Code Composer Studio version4

PROCEDURE:

1. Open Code Composer Studio v4.
2. To create the New Project
   Project→ New (File Name.)
3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.asm).
4. To Add Source files to Project
   Project→ Add files to
   Projectsum.asm

COMPILE:

1. To Compile: Project→ Compile
2. To Rebuild: project → rebuild,
   Which will create the final .out executable file. (Eg. vvit.out).
3. Procedure to Lode and Run program:
   Load the Program to DSK: File→ Load program
   →vvit.out To Execute project: Debug → Run

FIR-LOW-PASS FILTER

Filter type : FIR-LPF
Window type : Rectangular window
Sampling frequency : 41khz
Cut-off frequency : 4khz
No. of taps : 52

```
B3          .SET          0F000H
B2          .SET          0F00H
B1          .SET          00F0H
B0          .SET          000FH
DATA          .SET          50H
TXD          .SET          51H

          .mmregs
          .text
START:
      STM    #01h,ST0
      RSBX   CPL
      RSBX   FRCT
      NOP
      NOP
      STM    #150H,AR1
      LD     #0H,A
      RPT    #34H
      STL    A,*AR1+

REPFIRL:
      STM    #0A200H,AR4
      STM    #359,AR5
LOOP:
      PORTR  06,0
CHK_BUSY:
      PORTR  07,0
      BITF   0,#20H
      BC     CHK_BUSY,TC
      PORTR  04,0
      LD     0,A
      AND    #0FFFH,A
      XOR    #0800H,A
      SUB    #800H,A
      STM    #150H,AR1
      STL    A,*AR1
      STM    #183H,AR2
      LD     #0H,A
      RPT    #33H
      MACD   *AR2-,TABLE,A
      STH    A,1,0H
```

```
        LD      0H,A
        ADD     #800H,A
      STL     A,1H
       PORTW   1H,04H
        STL     A,*AR4+
        BANZ    LOOP,*AR5-

        STM     #0A200H,AR2
        STM     #359,AR3
REPSER:
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP
        LD      *AR2+,A

        SUB  #7FFH,A
        STL     A,DATA
        CALL    SERIAL
        BANZ    REPSER,*AR3-
        STM     #01h,ST0
        RSBX    CPL
        RSBX    FRCT
        NOP
        NOP

        B       REPFIRL

SERIAL:
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP

        LD      #25H,A
        CALL    TXDATA
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP

        LD      DATA,A
        AND     #B3,A           ;1st digit (from msb)
        SFTL    A,-12
        CALL    HEXASC
        CALL    TXDATA
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP

        LD      DATA,A
        AND     #B2,A           ;1st digit (from msb)
        SFTL    A,-8
        CALL    HEXASC
        CALL    TXDATA
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP

        LD      DATA,A
        AND     #B1,A           ;1st digit (from msb)
        SFTL    A,-4
        CALL    HEXASC
        CALL    TXDATA
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP

        LD      DATA,A
        AND     #B0,A           ;1st digit (from msb)
        CALL    HEXASC
        CALL    TXDATA
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP
      LD      #24H,A
        CALL    TXDATA
        STM     #140H,ST0
        RSBX    CPL
        NOP
        NOP
        NOP
        NOP
        RET
```

```
HEXASC:
        ADD     #30H,A
        LD      A,B
        SUB     #3AH,B
        BC      LESS9,BLT
        ADD     #7H,A
LESS9:
        RET
TXDATA:
        CALL    8C69H              ;8C38H for 5416 mode 1
        SSBX    INTM
        rpt     #2ffh           ;delay
        nop
        RET
;fs = 41khz ; fc = 4khz ; N = 52
TABLE:
        .word   01FH
        .word   010EH
        .word   01ABH
        .word   01B4H
        .word   0117H
        .word   0H
        .word   0FECDH
        .word   0FDEEH
        .word   0FDC2H
        .word   0FE6EH
        .word   0FFCDH
        .word   016FH
        .word   02C0H
        .word   0333H
        .word   0274H
        .word   097H
        .word   0FE19H
        .word   0FBCBH
        .word   0FA9BH
        .word   0FB53H
        .word   0FE50H
        .word   0362H
        .word   09C5H
        .word   01048H
        .word   01599H
        .word   01895H
        .word   01895H
        .word   01599H
        .word   01048H
        .word   09C5H
        .word   0362H
        .word   0FE50H
        .word   0FB53H
        .word   0FA9BH
        .word   0FBCBH
        .word   0FE19H
        .word   097H
        .word   0274H
        .word   0333H
        .word   02C0H
        .word   016FH
        .word   0FFCDH
        .word   0FE6EH
        .word   0FDC2H
        .word   0FDEEH
        .word   0FECDH
        .word   0H
        .word   0117H
        .word   01B4H
        .word   01ABH
        .word   010EH
        .word   01FH
```

FIR-HIGH PASS FIR FILTER

Sampling freq : 43khz
Cut-off freq : 2khz
N : 52
Filter type : High pass filter
Window type : Rectangular

Program Description:
1. Make all the x(n) zero initially
2. Read the data from the adc.
3. Store the adc data in x(0)
4. Make the pointer to point the x(n_end)
5. Perform the convolution of x(n) and the coefficients h(n) using MACD instruction.
6. Send the convolution output to the dac
7. Repeat from step 2.

```
             .mmregs
             .text
START:
        STM     #01h,ST0        ;intialize the data page pointer
        RSBX    CPL             ;Make the processor to work using DP
        RSBX    FRCT            ;reset the fractional mode bit
        NOP
        NOP
;*****loop to make all x(n) zero initially*****
        STM     #150H,AR1       ;initialize ar1 to point to x(n)
        LD      #0H,A           ;make acc zero
        RPT     #34H
        STL     A,*AR1+         ;make all x(n) zero
```

```
;*****to read the adc data and store it in x(0)*****
LOOP:
        PORTR   06,0            ;start of conversion
CHK_BUSY:
       ; PORTR   07,0            ;check for busy
       ; BITF    0,#20H
       ; BC      CHK_BUSY,TC
        PORTR   04,0            ;read the adc data
        LD      0,A
        AND     #0FFFH,A        ;AND adc data with 0fffh for 12 bit adc
        XOR     #0800H,A        ;recorrect the 2's complement adc data
        SUB     #800H,A         ;remove the dc shift
        STM     #150H,AR1       ;initialize ar1 with x(0)
        STL     A,*AR1          ;store adc data in x(0)
        STM     #183H,AR2       ;initialize ar2 with x(n_end)
;*****start of convolution*****
        LD      #0H,A           ;sum is 0 initially
        RPT     #33H
        MACD    *AR2-,TABLE,A   ;convolution process
        STH     A,1,0H
        LD      0H,A
        ADD     #800H,A         ;add the dc shift to the convolution output
        STL     A,1H
        PORTW   1H,04H          ;send the output to the dac
        B       LOOP

TABLE:
        .word   0FCEFH
        .word   62H
        .word   0FD50H
        .word   14AH
        .word   0FE1BH
        .word   28FH
        .word   0FF11H
        .word   3E5H
        .word   0FFD1H
        .word   4ECH
        .word   0FFF5H
        .word   54FH
        .word   0FF28H
        .word   4DAH
        .word   0FD38H
        .word   398H
        .word   0FA2EH
        .word   1DDH
        .word   0F627H
        .word   55H
        .word   0F131H
        .word   4BH
        .word   0EA6DH
        .word   568H
        .word   0D950H
        .word   459EH
        .word   459EH
        .word   0D950H
        .word   568H
        .word   0EA6DH
        .word   4BH
        .word   0F131H
        .word   55H
        .word   0F627H
        .word   1DDH
        .word   0FA2EH
        .word   398H
        .word   0FD38H
        .word   4DAH
        .word   0FF28H
        .word   54FH
        .word   0FFF5H
        .word   4ECH
        .word   0FFD1H
        .word   3E5H
        .word   0FF11H
        .word   28FH
        .word   0FE1BH
        .word   14AH
        .word   0FD50H
        .word   62H
        .word   0FCEFH
```

FIR-BAND PASS FIR FILTER

Sampling freq : 43khz
Cut-off freq1 : 2khz
Cut-off freq2 : 4khz
N : 52
Filter type : Band pass filter
Window type : Rectangular

```
;Program Description:
;1. Make all the x(n) zero initially
;2. Read the data from the adc.
;3. Store the adc data in x(0)
;4. Make the pointer to point the x(n_end)
;5. Perform the convolution of x(n) and the coefficients h(n) using
;   MACD instruction.
;6. Send the convolution output to the dac
;7. Repeat from step 2.
```

```
          .mmregs
          .text
START:
      STM    #01h,ST0       ;intialize the data page pointer
      RSBX   CPL            ;Make the processor to work using DP
      RSBX   FRCT           ;reset the fractional mode bit
      NOP
      NOP
;*****loop to make all x(n) zero initially*****
      STM    #150H,AR1      ;initialize ar1 to point to x(n)
      LD     #0H,A          ;make acc zero
      RPT    #34H
      STL    A,*AR1+        ;make all x(n) zero
;*****to read the adc data and store it in x(0)*****
LOOP:
      PORTR  06,0           ;start of conversion
CHK_BUSY:
     ;PORTR  07,0           ;check for busy
     ; BITF   0,#20H
     ; BC     CHK_BUSY,TC
      PORTR  04,0           ;read the adc data
      LD     0,A
      AND    #0FFFH,A       ;AND adc data with 0fffh for 12 bit adc
      XOR    #0800H,A       ;recorrect the 2's complement adc data
      SUB    #800H,A        ;remove the dc shift
      STM    #150H,AR1      ;initialize ar1 with x(0)
      STL    A,*AR1         ;store adc data in x(0)
      STM    #183H,AR2      ;initialize ar2 with x(n_end)
;*****start of convolution*****
      LD     #0H,A          ;sum is 0 initially
      RPT    #33H
      MACD   *AR2-,TABLE,A  ;convolution process
      STH    A,1,0H
      LD     0H,A
      ADD    #800H,A        ;add the dc shift to the convolution output
      STL    A,1H
      PORTW  1H,04H         ;send the output to the dac
      B      LOOP

TABLE:
      .word  208H
      .word  257H
      .word  218H
      .word  143H
      .word  0H
      .word  0FE9EH
      .word  0FD7AH
      .word  0FCE7H
      .word  0FD08H
      .word  0FDD1H
```

```
.word   0FEECH
.word   0FFE4H
.word   3DH
.word   0FFA1H
.word   0FDFCH
.word   0FB8FH
.word   0F8ECH
.word   0F6D4H
.word   0F608H
.word   0F713H
.word   0FA21H
.word   0FEE6H
.word   4A7H
.word   0A60H
.word   0EF8H
.word   1187H
.word   1187H
.word   0EF8H
.word   0A60H
.word   4A7H
.word   0FEE6H
.word   0FA21H
.word   0F713H
.word   0F608H
.word   0F6D4H
.word   0F8ECH
.word   0FB8FH
.word   0FDFCH
.word   0FFA1H
.word   3DH
.word   0FFE4H
.word   0FEECH
.word   0FDD1H
.word   0FD08H
.word   0FCE7H
.word   0FD7AH
.word   0FE9EH
.word   0H
.word   143H
.word   218H
.word   257H
.word   208H
```

FIR- BAND REJECT FIR FILTER

Sampling freq	: 43khz
Cut-off freq1	: 2khz
Cut-off freq2	: 4khz
N	: 52
Filter type	: Band reject filter
Window type	: Rectangular

Program Description:

1. Make all the x(n) zero initially
2. Read the data from the adc.
3. Store the adc data in x(0)
4. Make the pointer to point the x(n_end)
5. Perform the convolution of x(n) and the coefficients h(n) using MACD instruction.
6. Send the convolution output to the dac
7. Repeat from step 2.

```
            .mmregs
            .text
START:
        STM     #01h,ST0        ;intialize the data page pointer
        RSBX    CPL             ;Make the processor to work using DP
        RSBX    FRCT            ;reset the fractional mode bit
        NOP
        NOP
;*****loop to make all x(n) zero initially*****
        STM     #150H,AR1       ;initialize ar1 to point to x(n)
        LD      #0H,A           ;make acc zero
        RPT     #34H
        STL     A,*AR1+         ;make all x(n) zero
;*****to read the adc data and store it in x(0)*****
LOOP:
        PORTR   06,0            ;start of conversion
CHK_BUSY:
       ; PORTR   07,0           ;check for busy
       ; BITF    0,#20H
       ; BC      CHK_BUSY,TC
        PORTR   04,0            ;read the adc data
        LD      0,A
        AND     #0FFFH,A        ;AND adc data with 0fffh for 12 bit adc
        XOR     #0800H,A        ;recorrect the 2's complement adc data
        SUB     #800H,A         ;remove the dc shift
        STM     #150H,AR1       ;initialize ar1 with x(0)
        STL     A,*AR1          ;store adc data in x(0)
        STM     #183H,AR2       ;initialize ar2 with x(n_end)
;*****start of convolution*****
```

```
        LD      #0H,A           ;sum is 0 initially
        RPT     #33H
        MACD    *AR2-,TABLE,A   ;convolution process
        STH     A,1,0H
        LD      0H,A
        ADD     #800H,A         ;add the dc shift to the convolution output
        STL     A,1H
        PORTW   1H,04H          ;send the output to the dac
        B       LOOP

TABLE:
        .word   0FEB9H
        .word   14EH
        .word   0FDA1H
        .word   155H
        .word   0FE1BH
        .word   282H
        .word   0FEAFH
        .word   2ACH
        .word   0FD35H
        .word   8DH
        .word   0F9D9H
        .word   0FE07H
        .word   0F7CCH
        .word   0FEE2H
        .word   0FA2FH
        .word   4BAH
        .word   1AH
        .word   25CH
        .word   420H
        .word   1008H
        .word   89H
        .word   0D61H
        .word   0F3F2H
        .word   0AF9H
        .word   0DB7EH
        .word   045DFH
        .word   045DFH
        .word   0DB7EH
        .word   0AF9H
        .word   0F3F2H
        .word   0D61H
        .word   89H
        .word   1008H
        .word   420H
        .word   25CH
        .word   1AH
        .word   4BAH
        .word   0FA2FH
        .word   0FEE2H
        .word   0F7CCH
```

```
.word   0FE07H
.word   0F9D9H
.word   8DH
.word   0FD35H
.word   2ACH
.word   0FEAFH
.word   282H
.word   0FE1BH
.word   155H
.word   0FDA1H
.word   14EH
.word   0FEB9H
```

RESULT:

Thus the asm program for the design of FIR filter was plotted successfully.

Ex. No: 11
Date:

DESIGN OF IIR FILTERS

AIM:

To write a C program for the design of IIR Filter, also plots the magnitude responses for the same.

APPARATUS REQUIRED:

HARDWARE	: Personal Computer & TMS320C67XX
kit SOFTWARE	: Code Composer Studio version4

PROCEDURE:

1. Open Code Composer Studio v4.
2. To create the New Project
   Project→ New (File Name. pjt, Eg: vvits.pjt)
3. To create a Source file
   File →New→ Type the code (Save & give file name, Eg: sum.c).
4. To Add Source files to Project
   Project→ Add files to Project→
   sum.c
5. To Add rts.lib file & Hello.cmd:
   Project→ Add files to Project→ rts6700.lib
   Library files: rts6700.lib (Path: c:\ti\c6000\cgtools\lib\
   rts6700.lib) Note: Select Object& Library in (*.o,*.l) in Type of
   files
6. Project→ Add files to Project →hello.cmd
   CMD file - Which is common for all non real time
   programs. (Path: c:\ti \
   tutorial\dsk6713\hello1\hello.cmd)
   Note: Select Linker Command file (*.cmd) in Type of files

COMPILE:

1. To Compile: Project→ Compile
2. To Rebuild: project → rebuild,

Which will create the final .out executable file. (Eg. vvit.out).

3. Procedure to Lode and Run program:

   Load the Program to DSK: File→ Load program

   →vvit.out To Execute project: Debug → Run

IIR-LOWPASS FILTER

Filter type : Low pass filter
Filter order : 2
Filter design type : Butterworth
Pass band attenuation : 3db
First corner freq : 0.2
Second corner freq : 0.24
Sampling freq : 50Khz
Cut-off freq : 10Khz

FROM PCDSP COEFFICIENTS

```
XN          .SET    0H
XNM1          .SET    1H
XNM2          .SET    2H
YN          .SET    3H
YNM1          .SET    4H
YNM2          .SET    5H
XN1          .SET    6H
XN1M1          .SET    7H
XN1M2          .SET    8H
YN1          .SET    9H
YN1M1          .SET    0AH
YN1M2          .SET    0BH
TEMP          .SET    0CH
A10          .SET    0100H
A11          .SET    0FFA2H
A12          .SET    0032H
B10          .SET    0100H
B11          .SET    0200H
B12          .SET    0100H
      .mmregs
      .text
START:
      STM     #40H,PMST
      RSBX    CPL
      STM     #01H,ST0
      RSBX    FRCT
      NOP
      NOP
      NOP
;initialize xn,x(n-1),x(n-2),yn,y(n-1),y(n-2)
      ST     #0H,XN
      ST     #0H,XNM1
      ST     #0H,XNM2
      ST     #0H,YN
      ST     #0H,YNM1
```

```
        ST     #0H,YNM2
        ST     #0H,XN1
        ST     #0H,XN1M1
        ST     #0H,XN1M2
        ST     #0H,YN1
        ST     #0H,YN1M1
        ST     #0H,YN1M2
REPEAT:
;to read data from ADC
        PORTR   06,20                 ;start of conversion
CHK_BUSY:                               ;check status
       ; PORTR   07,20
       ; BITF    20,#20H
       ; BC      CHK_BUSY,TC
        PORTR   04,20                 ;read ADC data
        LD      20,A
        AND     #0FFFH,A
        XOR     #0800H,A              ;to correct 2's complement
        SUB     #800H,A
        STL     A,XN                 ;xn
        STL     A,TEMP
;
        LD      #0H,B               ;sum = B = 0
        LD      #B10,A              ;b0 = T
        STLM    A,T
        MPY     XN,A                  ;b0*xn = A
        SFTL    A,-8
        ADD     A,B                   ;b0*xn =B

        LD      #B11,A              ;b0 = T
        STLM    A,T
        MPY     XNM1,A                    ;b0*xn = A
        SFTL    A,-8
        ADD     A,B                   ;b0*xn =B

        LD      #B12,A              ;b0 = T
        STLM    A,T
        MPY     XNM2,A                    ;b0*xn = A
        SFTL    A,-8
        ADD     A,B                   ;b0*xn =B

        LD      #A11,A              ;b0 = T
        STLM    A,T
        MPY     YNM1,A                    ;b0*xn = A
        SFTL    A,-8
        SUB     A,B                  ;b0*xn =B

        LD      #A12,A              ;b0 = T
        STLM    A,T
        MPY     YNM2,A                    ;b0*xn = A
        SFTL    A,-8
        SUB     A,B                 ;b0*xn =B
        STL     B,YN
        STL     B,XN1

        LD      YNM1,A
        STL     A,YNM2
        LD      YN,A
        STL     A,YNM1
        LD      XNM1,A
        STL     A,XNM2
        LD      XN,A
        STL     A,XNM1

        LD      YN,A
        ADD     #800H,A
        STL     A,YN
        PORTW   YN,04H
        B       REPEAT
```

HIGH PASS FILTER

Filter type : High pass filter
Filter order : 2
Filter design type : Butterworth
Pass band attenuation : 3db
First corner freq : 0.2
Second corner freq : 0.24
Sampling freq : 50Khz
Cut-off freq : 10Khz

FROM PCDSP COEFFICIENTS

```
XN          .SET    0H
XNM1        .SET    1H
XNM2        .SET    2H
YN          .SET    3H
YNM1        .SET    4H
YNM2        .SET    5H
XN1         .SET    6H
XN1M1       .SET    7H
XN1M2       .SET    8H
YN1         .SET    9H
YN1M1       .SET    0AH
YN1M2       .SET    0BH
TEMP        .SET    0CH
A10         .SET    0100H
A11         .SET    0FFEDH
```

```
A12           .SET    002CH
B10           .SET    0100H
B11           .SET    0FE00H
B12           .SET    0100H
        .mmregs
        .text
START:
        STM     #40H,PMST
        RSBX    CPL
        STM     #01H,ST0
        RSBX    FRCT
        NOP
        NOP
        NOP
;initialize xn,x(n-1),x(n-2),yn,y(n-1),y(n-2)
        ST      #0H,XN
        ST      #0H,XNM1
        ST      #0H,XNM2
        ST      #0H,YN
        ST      #0H,YNM1
        ST      #0H,YNM2
        ST      #0H,XN1
        ST      #0H,XN1M1
        ST      #0H,XN1M2
        ST      #0H,YN1
        ST      #0H,YN1M1
        ST      #0H,YN1M2
REPEAT:
;to read data from ADC
        PORTR   06,20               ;start of conversion
CHK_BUSY:                             ;check status
      ;  PORTR   07,20
      ;  BITF    20,#20H
      ;  BC      CHK_BUSY,TC
        PORTR   04,20               ;read ADC data
        LD      20,A
        AND     #0FFFH,A
        XOR     #0800H,A              ;to correct 2's complement
        SUB     #800H,A
        STL     A,XN                ;xn
        STL     A,TEMP
;
        LD      #0H,B               ;sum = B = 0
        LD      #B10,A              ;b0 = T
        STLM    A,T
        MPY     XN,A                  ;b0*xn = A
        SFTL    A,-8
        ADD     A,B                   ;b0*xn =B

        LD      #B11,A              ;b0 = T
```

```
STLM    A,T
MPY     XNM1,A                  ;b0*xn = A
SFTL    A,-8
ADD     A,B                     ;b0*xn =B

LD      #B12,A              ;b0 = T
STLM    A,T
MPY     XNM2,A                  ;b0*xn = A
SFTL    A,-8
ADD     A,B                     ;b0*xn =B

LD      #A11,A              ;b0 = T
STLM    A,T
MPY     YNM1,A                  ;b0*xn = A
SFTL    A,-8
SUB     A,B                     ;b0*xn =B

LD      #A12,A              ;b0 = T
STLM    A,T
MPY     YNM2,A                  ;b0*xn = A
SFTL    A,-8
SUB     A,B                     ;b0*xn =B
STL     B,YN
STL     B,XN1

LD      YNM1,A
STL     A,YNM2
LD      YN,A
STL     A,YNM1
LD      XNM1,A
STL     A,XNM2
LD      XN,A
STL     A,XNM1

LD      YN,A
ADD     #800H,A
STL     A,YN
PORTW   YN,04H
B       REPEAT
```

BAND PASS FILTER

Filter type : Band pass filter
Filter order : 2
Filter design type : Chebyshev-I
Pass band attenuation : 3db
Edge frequencies:
f1 : 0.1
f2 : 0.125
f3 : 0.15
f4 : 0.175
Sampling freq : 50Khz
Cut-off freq1 : 5Khz
Cut-off freq2 : 7.5Khz

FROM PCDSP COEFFICIENTS

```
XN        .SET    0H
XNM1      .SET    1H
XNM2      .SET    2H
YN        .SET    3H
YNM1      .SET    4H
YNM2      .SET    5H
XN1       .SET    6H
XN1M1     .SET    7H
XN1M2     .SET    8H
YN1       .SET    9H
YN1M1     .SET    0AH
YN1M2     .SET    0BH
TEMP      .SET    0CH
A10       .SET    0100H
A11       .SET    0FECBH
A12       .SET    00DAH
B10       .SET    0100H
B11       .SET    0000H
B12       .SET    0FF00H
      .mmregs
      .text
START:
      STM     #40H,PMST
      RSBX    CPL
```

```
        STM     #01H,ST0
        RSBX    FRCT
        NOP
        NOP
        NOP
;initialize xn,x(n-1),x(n-2),yn,y(n-1),y(n-2)
        ST      #0H,XN
        ST      #0H,XNM1
        ST      #0H,XNM2
        ST      #0H,YN
        ST      #0H,YNM1
        ST      #0H,YNM2
        ST      #0H,XN1
        ST      #0H,XN1M1
        ST      #0H,XN1M2
        ST      #0H,YN1
        ST      #0H,YN1M1
        ST      #0H,YN1M2
REPEAT:
;to read data from ADC
        PORTR   06,20                ;start of conversion
CHK_BUSY:                              ;check status
       ; PORTR   07,20
       ; BITF    20,#20H
       ; BC      CHK_BUSY,TC
        PORTR   04,20                ;read ADC data
        LD      20,A
        AND     #0FFFH,A
        XOR     #0800H,A              ;to correct 2's complement
        SUB     #800H,A
        STL     A,XN                ;xn
        STL     A,TEMP
;
        LD      #0H,B               ;sum = B = 0
        LD      #B10,A              ;b0 = T
        STLM    A,T
        MPY     XN,A                  ;b0*xn = A
        SFTL    A,-8
        ADD     A,B                   ;b0*xn =B

        LD      #B11,A              ;b0 = T
        STLM    A,T
        MPY     XNM1,A                  ;b0*xn = A
        SFTL    A,-8
        ADD     A,B                   ;b0*xn =B

        LD      #B12,A              ;b0 = T
        STLM    A,T
        MPY     XNM2,A                  ;b0*xn = A
        SFTL    A,-8
```

```
ADD     A,B                    ;b0*xn =B

LD      #A11,A             ;b0 = T
STLM    A,T
MPY     YNM1,A                 ;b0*xn = A
SFTL    A,-8
SUB     A,B                    ;b0*xn =B

LD      #A12,A             ;b0 = T
STLM    A,T
MPY     YNM2,A                 ;b0*xn = A
SFTL    A,-8
SUB     A,B                    ;b0*xn =B
STL     B,YN
STL     B,XN1

LD      YNM1,A
STL     A,YNM2
LD      YN,A
STL     A,YNM1
LD      XNM1,A
STL     A,XNM2
LD      XN,A
STL     A,XNM1

LD      YN,A
ADD     #800H,A
STL     A,YN
PORTW   YN,04H
B       REPEAT
```

RESULT:

Thus the asm program for the design of IIR filter were plotted successfully.