TAGORE INSTITUTE OF ENGINEERING AND TECHNOLOGY

Deviyakurichi-636112, Attur (TK), Salem (DT). Website: www.tagoreiet.ac.in

(Approved by AICTE, New Delhi and Affiliated to Anna University, Chennai)

Accredited by NAAC

Department of Electronics and Communication Engineering

III Year- V Semester - Electronics and Communication Engineering

EC 8562 Digital Signal Processing Laboratory

LAB MANUAL

Academic Year 2020-2021

(2017 Regulation)

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING V SEM - E.C.E.

EC 8562 – DIGITAL SIGNAL PROCESSING LABORATORY

S.NO	NAME OF EXPERIMENTS
1.	Generation of Elementary Discrete-Time Sequences
2.	Linear Convolution and Circular Convolution
3.	Auto correlation and Cross correlation.
4.	Frequency Analysis using DFT.
5.	Design of FIR filters for LPF, HPF, BPF, BSF
6.	Design of IIR filters for LPF, HPF, BPF, BSF.
7.	Study of architecture of Digital Signal Processor.
8.	Perform MAC operations using various addressing modes
9.	Design of FIR filters for LPF, HPF, BPF, BSF using DSP Processor.
10.	Design of IIR filters for LPF, HPF, BPF, BSF using DSP Processor
11.	Implement Up-sampling and Down-sampling operations

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

V SEM – E.C.E.

EC 8562- DIGITAL SIGNAL PROCESSING LABORATORY

CYCLE I

1. Generation of Elementary Discrete-Time Sequences.
2. Linear Convolution and Circular Convolution.
3. Auto correlation and Cross correlation.
4. Frequency Analysis using DFT.
5. Design of FIR filters for LPF, HPF, BPF, BSF.
6. Design of IIR filters for LPF,HPF, BPF, BSF.

CYCLE II

7. Study of architecture of Digital Signal Processor.
8. Perform MAC operations using various addressing modes.
9. Design and demonstration of FIR filters for LPF, HPF, BPF, BSF.
10. Design and demonstration of IIR filters for LPF, HPF, BPF, BSF.
11. Implement Up-sampling and Down-sampling operations

EXP. NO: 1 GENERATION OF ELEMENTARY DISCRETE-TIME SEQUENCES

DATE:

AIM:

To generate basic sequences such as Unit impulse, Unit step, Ramp, Exponential, Sine sequence & Cosine Sequence using MATLAB Programs.

APPARATUS REQUIRED:

1.Personal Computer

2.MATLAB Software

ALGORITHM:

1. Start the program.
2. Get the N point values from the user.
3. Assign the range of the time axis.
4. Give the title for the x axis and y axis for the program.
5. Plot the data sequence as a discrete values or continuous as per our requirements.

THEORY:

i. Unit Step Sequence:

The Unit Step Sequence is designed as unit step means that the amplitude of U(t) = 1

$U(n) = 1 \; ; n \geq 0$

$= 0 \; ; n < 0$

ii. Ramp Sequence:

The ramp sequence is defined as

$Ur(n) = n \; ; n \geq 0$

$= 0 \; ; n < 0$

iii .Exponential Sequence:

Exponential sequence is defined as

$g(n) = a^n \; ; n \geq 0$

$0 \; ; n < 0$

When the values of a > 1 the sequence grows exponentially and when the value is 0 < a < 1 the sequence

decay's exponentially. Note also that a < 0 the discrete time exponential signal takes attenuating signal.

iv. Cosine signal:

A discrete cosine signal is given by

$$X(n) = A \cos (\omega_o n + \phi)$$

Where ωo is the frequency and ϕ is the phase using bulers identity

we can write $A\cos (\omega_o n + \phi) = A/2\ e^{j\phi}\ e^{j\omega on} + A/2\ e^{-j\phi}\ e^{-j\omega on}$ ·Since $| e^{j\omega on}|^2 = 1^{st}$ the energy signal is infinite and the average power of the signal is 1

v. Sinusoidal signal:

A continuous time sinusoidal signal is given by

$$x(t) = A \sin (rt + \theta)$$

Where 'A' is amplitude & r is the frequency in rad/sec and θ are the phase angle radians. The analog sinusoidal signal has the following properties

i) The signal is periodic satisfy the condition

$$x(t+T) = x(t)$$

ii) For different value of frequencies the continuous time sinusoidal signals are themselves different.

PROCEDURE:

- Start the MATLAB software and create new M-file.
- Type the program in the file.
- Save and compile the program.
- Give the input data.
- Observe the output waveform.
- Thus the graph is to be plotted.

PROGRAM:

Discrete Time Signals:

%Unit step sequence:

```
n=input ('Enter the n value');
t=0:1: n-1;
```

```
y=ones (1,n);
subplot (2,2,1);
stem (t,y);
xlabel ('n-->');
ylabel ('Amplitude');
title ('Unit step signal');
```

%Generation of sine sequence:

```
n=0:0.01:pi;
y=sin (2*pi*n);
subplot (2,2,2)
stem (n,y);
xlabel ('n->');
ylabel ('Amplitude');
title ('Sine sequence');
```

%Generation of cosine sequence:

```
n=0:0.01: pi;
y=cos (2*pi*n);
subplot (2,2,3);
stem (n,y);
xlabel ('x(n)');
ylabel ('Amplitude');
title ('Cosine sequence');
```

%Exponential sequence:

```
n=input ('Enter the length');
t=0: n;
a=input ('Enter the a value');
y=exp (a*t);
subplot (2,2,4);
stem (t,y);
xlabel ('Time');
ylabel('Amplitude');
title('Exponential sequence');
```

%Ramp signal:

```
clc;
l=input('Enter the length of the sequence');
n=0:l;
r=n;
stem(n,r);
disp(r);
title('Ramp sequence');
xlabel('Time');
ylabel('Amplitude');
```

%Impulse signal:

```
n=input ('Enter the n value');
t=0:1: n-1;
y= (1, zeros(1,n-1));
subplot (2,2,5);
stem (t,y);
xlabel ('n-->');
ylabel ('Amplitude');
title ('Impulse signal');
```

Continuous Time Signals:

%Unit step sequence:

```
n=input ('Enter the n value');
t=0:1: n-1;
y=ones (1,n);
subplot (2,2,1);
plot (t,y);
xlabel ('n-->');
ylabel ('Amplitude');
title ('Unit step signal');
```

%Generation of sine sequence:

```
t=0:0.01:pi;
y=sin (2*pi*t);
subplot (2,2,2)
```

```
plot (t,y);
xlabel ('n->');
ylabel ('Amplitude');
title ('Sine sequence');
```

%Generation of cosine sequence:

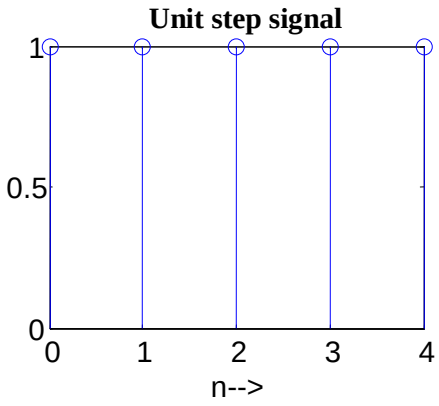
```
t=0:0.01: pi;
y=cos (2*pi*t);
subplot (2,2,3);
plot (t,y);
xlabel ('x(n)');
ylabel ('Amplitude');
title ('Cosine sequence');
```

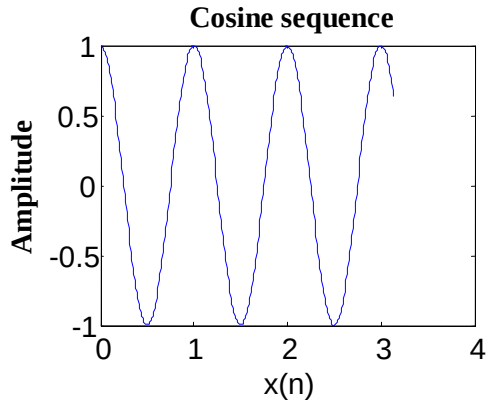
%Exponential sequence:

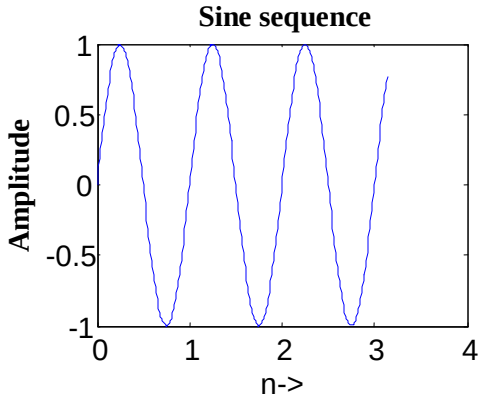
```
n=input ('Enter the length');
t=0: n;
a=input ('Enter the a value');
y=exp (a*t);
subplot (2,2,4);
plot (t,y);
xlabel ('Time');
ylabel('Amplitude');
title('Exponential sequence');
```

OUTPUT:

Enter the n value = 5



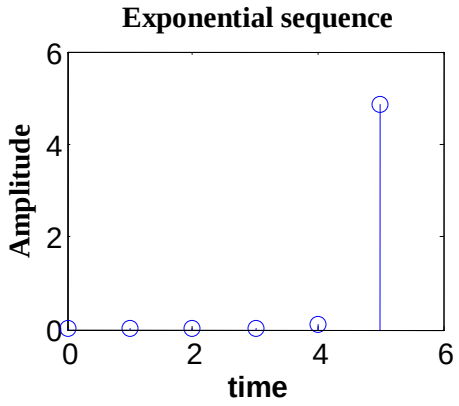


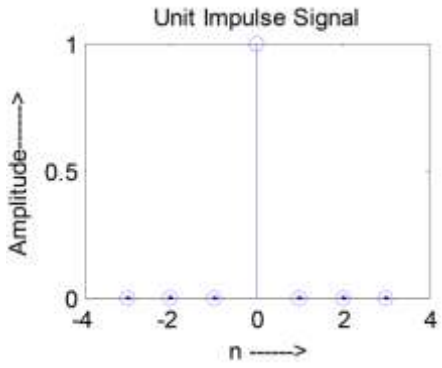


OUTPUT:

Enter the length 5

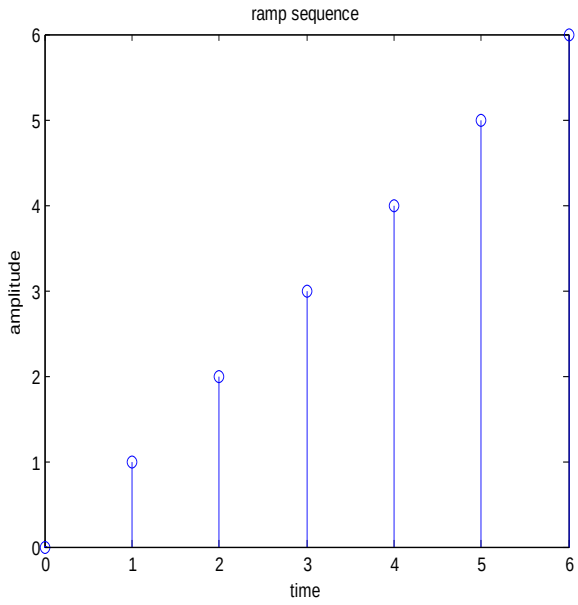
Enter the a value 5





OUTPUT:

Enter the length of the sequence 6



RESULT:

Thus the basic sequences of Unit impulse, Unit step, Ramp, Exponential and Sine Sequence & Cosine Sequence are generated using MATLAB Programs.

EXP. NO: 2 LINEAR CONVOLUTION AND CIRCULAR CONVOLUTION
DATE:

AIM:

To perform Linear Convolution and Circular Convolution of two discrete sequences using MATLAB.

APPARATUS REQUIRED:

Personal Computer

MATLAB Software

ALGORITHM:

1. Start the program.
2. Get the input values from the user.
3. Assign the range of the time axis.
4. Give the title for the x axis and y axis for the program.
5. Plot the data sequence as a discrete values or continuous as per our requirements.

THEORY:

The linear convolution of two sequence x(n) of L no of samples and h(n) of M no of samples produce a result y(n) which contains N= L + M – 1 . If is a sequence which is periodic with N samples. Linear convolution can be used to find the response of a filter.

In the case of circular convolution if x(n) contains L no of samples and h(n) has N no of samples and that L > M, then we perform circular convolution between the two using N = Max (L, M) by adding L, M no of zero samples to the sequence h(n) . So that both sequence are periodic with N. Circular cannot be used to find the response of a linear filter without zero padding.

PROCEDURE:

- Start the MATLAB software and create new M-file.
- Type the program in the file.
- Save & compile the program.
- Give the input data.
- Observe the output waveform.
- Thus the graph is to be plotted.

PROGRAM:

%Linear convolution:

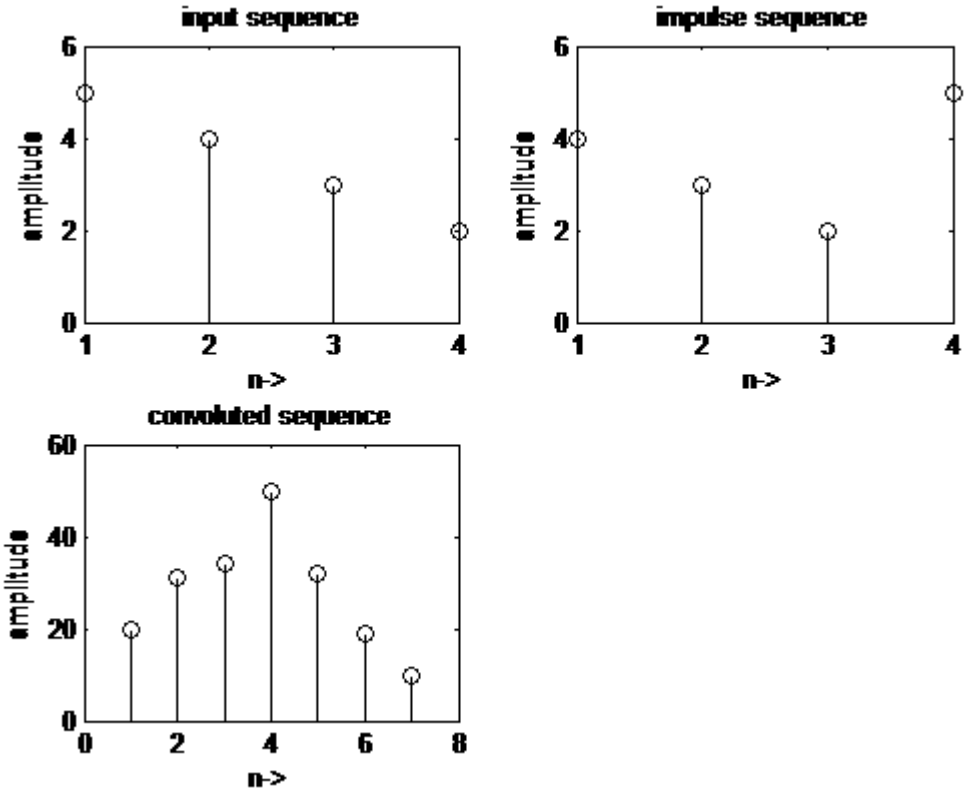
```
clc;
x=input ('Enter the input sequence');
n1=length(x);
subplot (2,2,1);
Stem(x);
title ( ' Input sequence');
xlabel ('n->');
ylabel ('Amplitude');
h=input ('Enter the impulse sequence');
n2=length (h);
subplot (2,2,2);
stem (h);
title (' Impulse sequence');
xlabel ('n->');
ylabel ('Amplitude');
y=conv(x, h);
n=1:n1+n2-1;
subplot (2,2,3);
stem (n,y);
Disp(y);
title ('Convoluted sequence');
xlabel ('n->');
ylabel ('Amplitude');
```

OUTPUT

Enter the input sequence [5 4 3 2]

Enter the impulse sequence [4 3 2 5]

Output is 20 31 34 50 32 19 10



PROGRAM

%Circular convolution

```
clc;
X1=input ('Enter the first sample');
X2=input ('Enter the second sample');
l1=length(x1);
l2=length(x2);
x1s=fft(x1);
Disp (x1s);
x2s=fft(x2);
Disp (x2s);
x3s=x1s.*x2s;
y=ifft (x3s);
Disp(y);
Subplot (3, 1, 1);
n=0:1:l1-1;
Stem (n, x1);
Title ('First input sample');
```

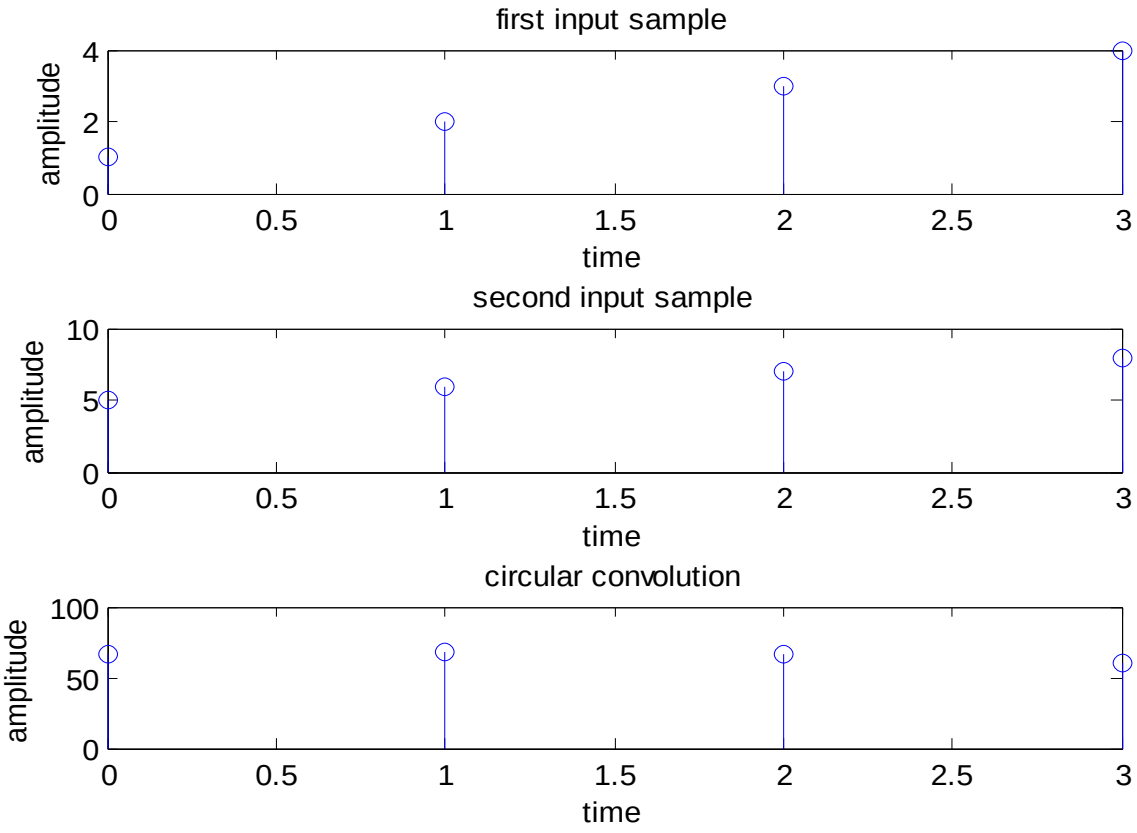
```
Xlabel ('Time');
Ylabel ('Amplitude');
Subplot (3, 1, 2);
n=0:1:l2-1;
Stem (n, x2);
Title (Second input sample');
Xlabel ('Time');
Ylabel ('Amplitude');
Subplot (3, 1, 3);
Stem (n, y);
Title ('Circular convolution');
Xlabel ('Time');
Ylabel ('Amplitude');
```

OUTPUT:

Enter the first sample [1 2 3 4]

Enter the second sample [5 6 7 8]

Output is 66 68 66 60



RESULT:

Thus the linear convolution and circular convolution programs were performed and verified using MATLAB.

EXP.NO:3 AUTO CORRELATION AND CROSS CORRELATION

DATE:

AIM:

To implement auto-correlation and cross-correlation functions using MATLAB.

APPARATUS REQUIRED:

Personal Computer

MATLAB Software

ALGORITHM:

1. Start the program.
2. Get the input values from the user.
3. Assign the range of the time axis.
4. Give the title for the x axis and y axis for the program.
5. Plot the data sequence as a discrete values or continuous as per our requirements.

PROCEDURE:

- Start the MATLAB software and create new M-file.
- Type the program in the file.
- Save and compile the program.
- Give the input data.
- Observe the output waveform.
- Thus the graph is to be plotted.

PROGRAM:

% Cross correlation

```
clc;
clear all; close all;
x=input ('Enter the first sequence');
h=input('Enter the second sequence');
y=xcorr(x,h);
subplot(3,1,1);
stem(x);
xlabel('x');
ylabel('Amplitude');
title('x sequence');
subplot(3,1,2);
```

```
stem(h);
xlabel('h');
ylabel('Amplitude');
title('h sequence');
subplot(3,1,3);
stem(y);
xlabel('y');
ylabel('amplitude');
title(' y sequence');
```

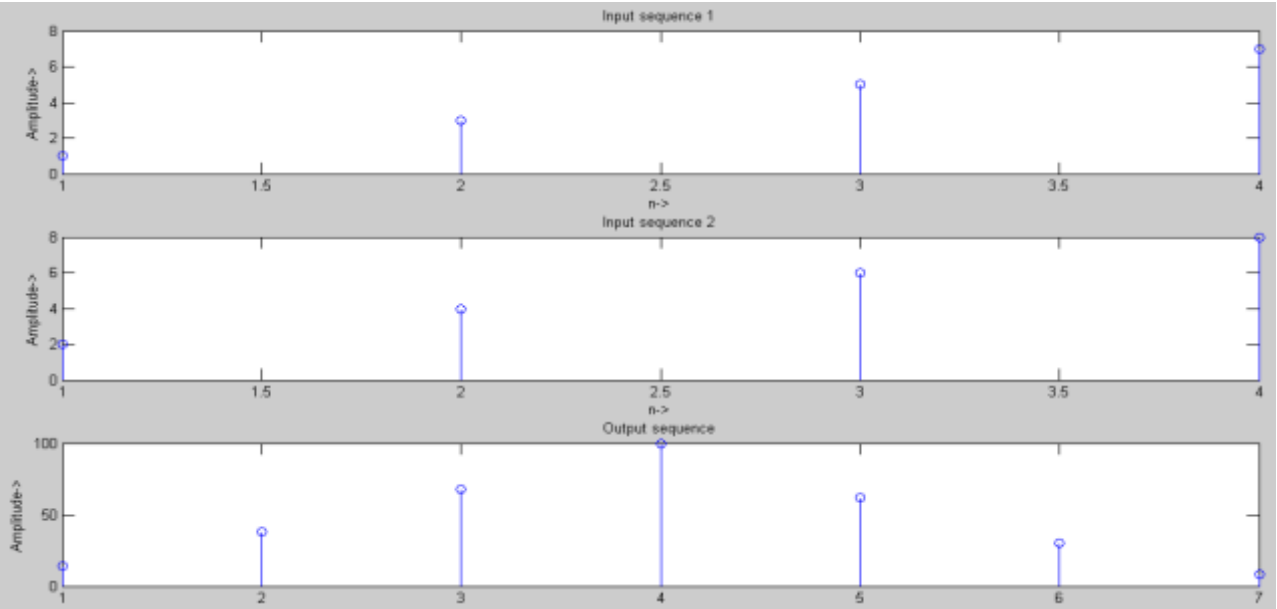
OUTPUT (Cross Correlation):

Enter the first sequence [1 2 3 4]

Enter the second sequence [4 3 2 1]

The resultant signal is

Y= 1.0 4.0 10.0 20.0 25.0 24.00 16.00



%Auto correlation

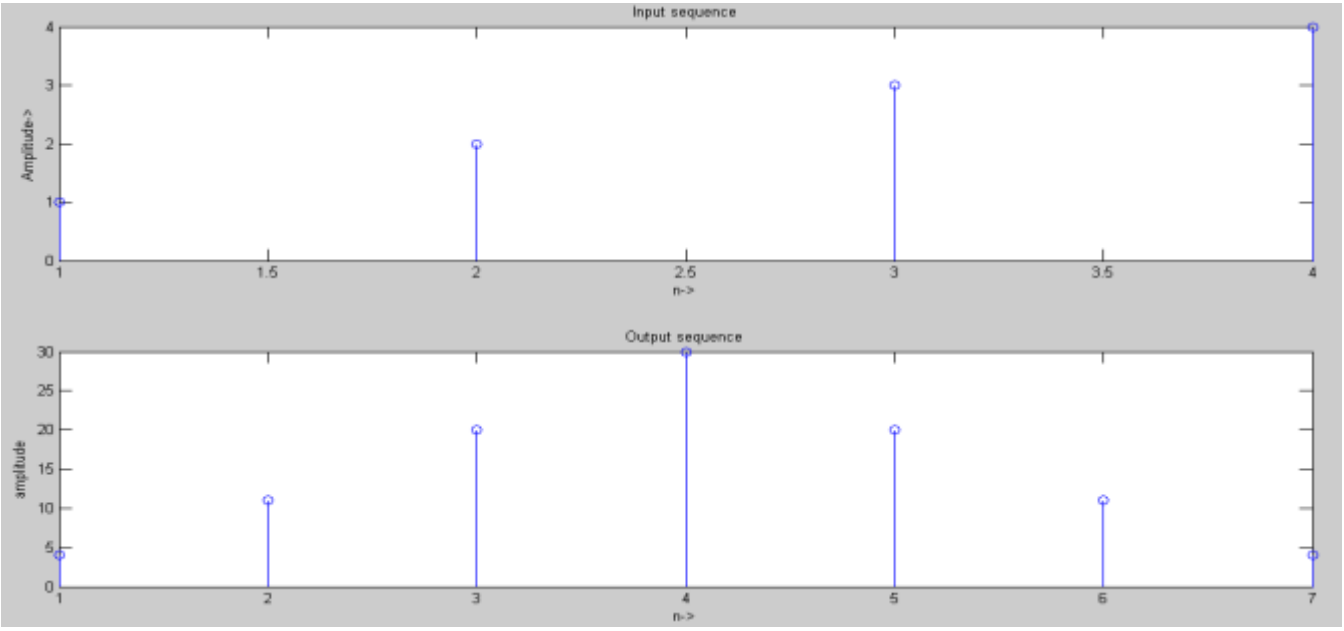
```
clc;
clear all; close all;
x=input ('Enter the t sequence');
y=xcorr(x,x);
```

```
subplot(2,1,1);
stem(x);
xlabel('x');
ylabel('Amplitude');
title('x sequence');
subplot(2,1,2);
stem(y);
xlabel('y');
ylabel('Amplitude');
title(' y sequence');
disp('The resultant signal is');
```

OUTPUT (Auto Correlation):

Enter the sequence = [1 2 3 4]

The resultant signal is Y= 4 11 20 30 20 11 4



RESULT:

Thus the auto-correlation and cross-correlation functions are generated using MATLAB.

EXP. NO: 4 FREQUENCY ANALYSIS USING DFT
DATE:

AIM

To write a program for frequency analysis using DFT.

APPARATUS REQUIRED:

Personal Computer

MATLAB Software

ALGORITHM:

1. Start the program.
2. Get the input values from the user.
3. Assign the range of the time axis.
4. Give the title for the x axis and y axis for the program.
5. Plot the data sequence as a discrete values or continuous as per our requirements.

THEORY:

DFT is used for analyzing discrete-time finite-duration signals in the frequency domain

Let x[n]be a finite-duration sequence of length N such that x[n]=0, 0<n<N-1outside.

The DFT pair of is:

$$X[k] = \left\{ \frac{1}{N} \sum_{n=0}^{N-1} x(n) \mathrm{W}_N^{kn}, 0 \le k \le N-1 \right.$$

$$X[k] = \left\{ \frac{1}{N} \sum_{n=0}^{N-1} x(n) \mathrm{W}_N^{kn}, 0 \le k \le N-1 \right.$$

PROGRAM:

```
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)');
```

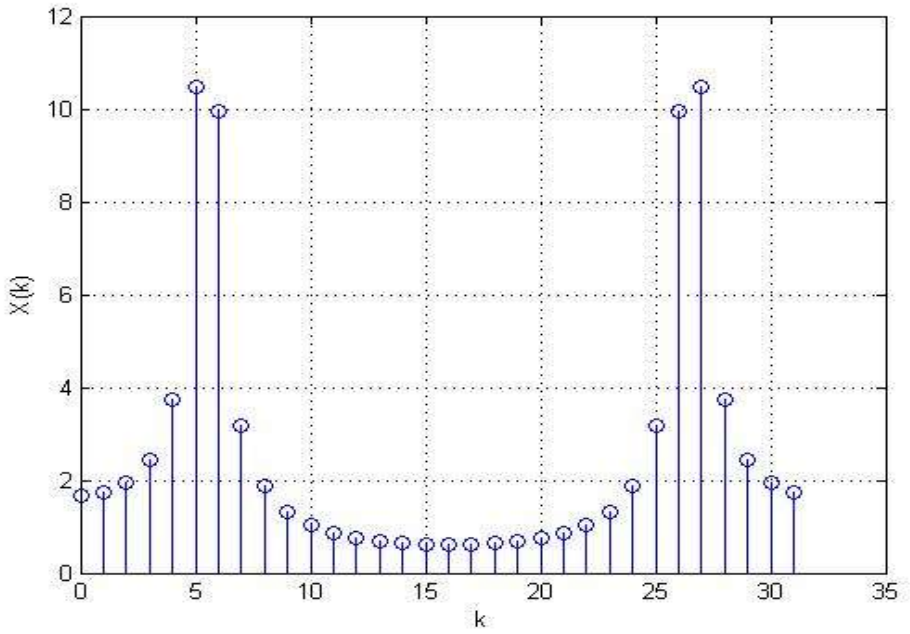
INPUT:

type length of DFT=32

type sampling period=64

type the sinusoidal freq=11

OUTPUT:



RESULT

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

EXP.NO: 5A DESIGN OF FIR FILTER USING RECTANGULAR WINDOW
DATE:

AIM

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

Personal Computer

MATLAB Software

ALGORITHM

1. Start the program.
2. Get pass band ripple and stop band ripple.
3. Get Pass band and stop band frequency.
4. Get sampling frequency.
5. Calculate the order of the filter.
6. Find the window Coefficient.
7. Plot the magnitude and phase response.

THEORY

The rectangular window (sometimes known as the **boxcar** or **Dirichlet window**) is the simplest window, equivalent to replacing all but *N* values of a data sequence by zeros, making it appear as though the waveform suddenly turns on and off:

W(n)= 1.

Other windows are designed to moderate these sudden changes because discontinuities have undesirable effects on the discrete-time Fourier transform (DTFT) and/or the algorithms that produce samples of the DTFT.

The rectangular window is the 1st order *B*-spline window as well as the 0th power cosine window.

PROCEDURE

- Start the MATLAB software and create new M-file.
- Type the program in the file.

- Save & compile the program.
- Give the input data.
- Observe the output waveform.
- Thus the graph is to be plotted.

PROGRAM:

```
clc;
clear all;
close all;
rp=input('Pass band ripple=');
rs=input('Stop band ripple=');
fs=input('Stop band frequency in rad/sec=');
fp=input('Pass band frequency in rad/sec=');
f=input('Sampling frequency in rad/sec=');
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);
                                %LOW PASS FILTER
b=fir1(n,wp,'low',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('Normalized frequency---->');
```

```
title('LOW PASS FILTER')
                                        %HIGH PASS 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('Normalized frequency---->');
title('HIGH PASS FILTER')
                                        %BAND PASS FILTER
wn=[wp,ws];
b=fir1(n,wp,'band',y);
[h,o]=freqz(b,1,256);
subplot(2,2,3);

plot(o/pi,m);
ylabel('Gain in db---->');
xlabel('Normalized frequency---->');
title('BAND PASS FILTER')
                                        %BAND STOP 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('Normalized frequency---->');
title('BAND STOP FILTER')
```

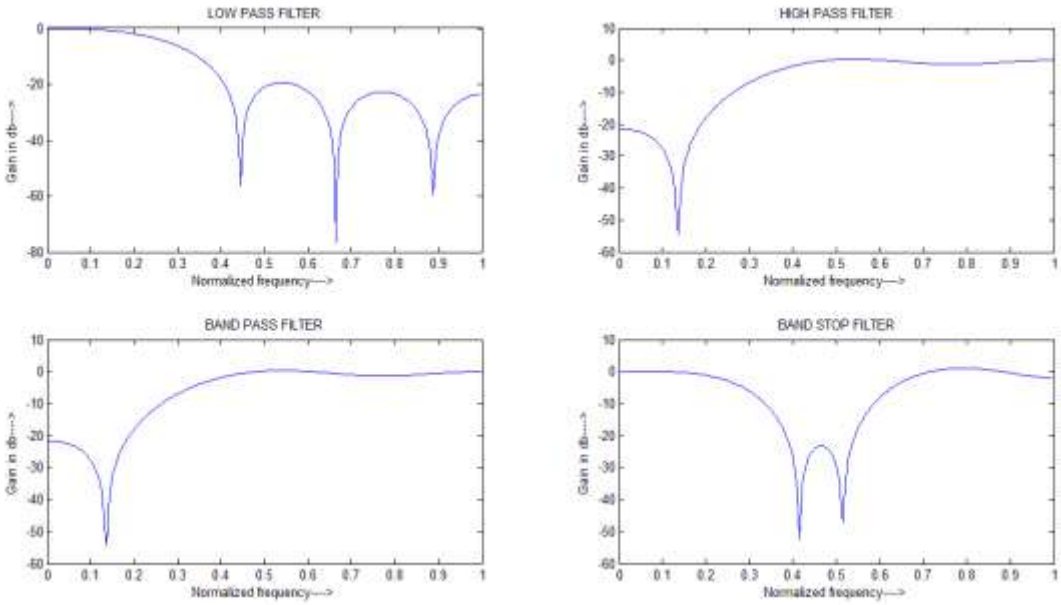
OUTPUT

```
Pass band ripple=0.03
Stop band ripple=0.04
```

Stop band frequency in rad/sec=1200

Pass band frequency in rad/sec=600

Sampling frequency in rad/sec=4000



RESULT

Thus the design of FIR low pass, high pass, band pass and band stop filters were obtained for Rectangular Window using MATLAB.

EXP.NO: 5B DESIGN OF FIR FILTER USING HAMMING WINDOW
DATE:

AIM:

To design the FIR low pass, High pass, Band pass and Band stop filters for Hamming window by using MATLAB.

APPARATUS REQUIRED:

Personal Computer

MATLAB Software

ALGORITHM:

1. Start the program.
2. Get pass band ripple and stop band ripple.
3. Get Pass band and stop band frequency.
4. Get sampling frequency.
5. Calculate the order of the filter.
6. Find the window Coefficient.
7. Plot the magnitude and phase response.

THEORY:

HAMMING WINDOW:

The filters response can be obtained by

$$W_H(n) = 0.54+0.46\cos 2\pi n/N-1 \quad ; \quad -(N-1)/2 \le n \le N-1/2$$

$$= 0 \quad ; \quad 0$$

The frequency response is

$$W_H(e^{jw}) = 0.54\sin wn/2/\sin w/2+0.23\sin(wn/2-\pi n/n-1)/\sin(w/2-\pi/n-1)+0.23\sin(wn/2+n/n-1)/\sin(w/2-\pi/n-1)$$

PROCEDURE:

- Start the MATLAB software and create new M-file.
- Type the program in the file.
- Save & compile the program.
- Give the input data.
- Observe the output waveform.

- Thus the graph is to be plotted.

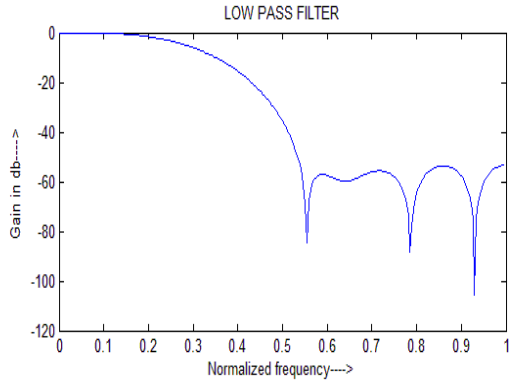
PROGRAM:

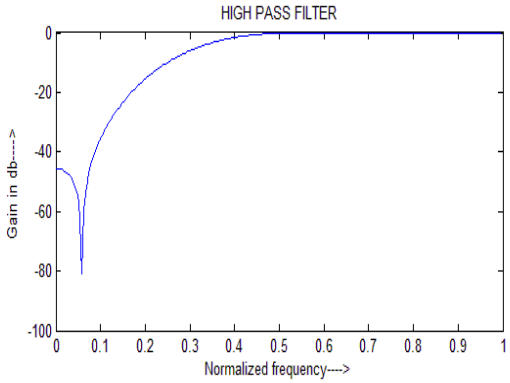
```
clc;
clear all;
close all;
rp=input('Pass band ripple=');
rs=input('Stop band ripple=');
fs=input('Stop band frequency in rad/sec=');
fp=input('Pass band frequency in rad/sec=');

f=input('Sampling frequency in rad/sec=');
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=hamming(n1);
                                %LOW PASS FILTER
b=fir1(n,wp,'low',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('Normalized frequency---->');
title('LOW PASS FILTER')
                                %HIGH PASS FILTER
b=fir1(n,wp,'high',y);
```

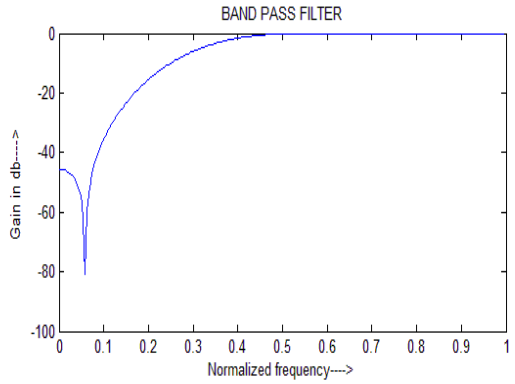
```
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('Gain in db---->');
xlabel('Normalized frequency---->');
title('HIGH PASS FILTER')
                                %BAND PASS FILTER
wn=[wp,ws];
b=fir1(n,wp,'band',y);
[h,o]=freqz(b,1,256);
subplot(2,2,3);
plot(o/pi,m);
ylabel('Gain in db---->');
xlabel('Normalized frequency---->');
title('BAND PASS FILTER')

                                %BAND STOP 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('Normalized frequency---->');
title('BAND STOP FILTER')
```

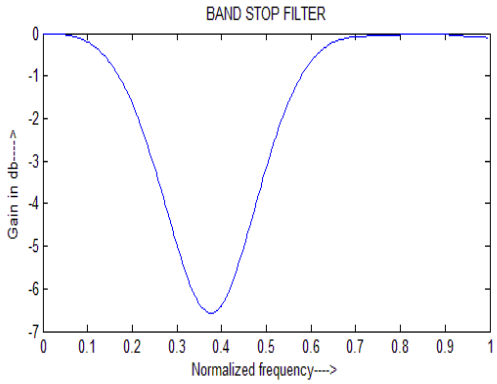
OUTPUT:

Pass band ripple	=0.03
Stop band ripple	=0.04
Stop band frequency in rad/sec	=1200
Pass band frequency in rad/sec	=600
Sampling frequency in rad/sec	=4000









RESULT:

Thus the design of FIR low pass, high pass, bands pass and band stop filters for Hamming Window was obtained using MATLAB.

EXP. NO: 6A IIR FILTER USING BUTTERWORTH FILTER APPROXIMATION DATE:

AIM:

To design the IIR low pass, High pass, Band pass and Band stop filters using Butterworth approximation and find out the response of the filter by using MATLAB.

APPARATUS REQUIRED:

Personal Computer

MATLAB Software

ALGORITHM:

1. Start the program.
2. Get pass band ripple and stop band ripple.
3. Get Pass band and stop band frequency.
4. Calculate the order of the filter.
5. Plot the band pass and band stop filter.

THEORY:

Butterworth Filter:

- ✓ The magnitude response of butter worth filter decreases maintain as the frequency increases 0 to $\propto$.
- ✓ The filter transition band is more in butter worth filter.
- ✓ The poles on the butter worth filter are lie on a circle.
- ✓ For the same specification the no. of files create disadvantage.

PROCEDURE:

- ➢ Start the MATLAB software and create new M-file.
- ➢ Type the program in the file.
- ➢ Save & compile the program.
- ➢ Give the input data.
- ➢ Observe the output waveform.
- ➢ Thus the graph is to be plotted.

PROGRAM:

```
clear all;
clc;
close all;
format long
rp=input('enter the pass band ripple');
rs=input('enter the stop band ripple');
wp=input('enter the pass band frequency ');
ws=input('enter the stop band frequency ');
[n1,w1]=buttord(wp,ws,rp,rs);
[num,den]=butter(n1,w1);
[num1,den1]=butter(n,w1,'stop');

[g,w]=freqz(num,den);
[g1,w1]=freqz(num1,den1);
m=20*log10(abs(g));
m1=20*log10(abs(g1));
title('Gain response of Butterworth filter');
Subplot(2,1,1);
Plot(w/pi,m);
Ylabel('gain in db?>');
Xlabel('omega/pi');
title('Bandpass filter');
Subplot(2,1,2);
Plot(w/pi,m1);
Ylabel('gain in db?>');
Xlabel('omega/pi');
title('Bandreject filter');
```

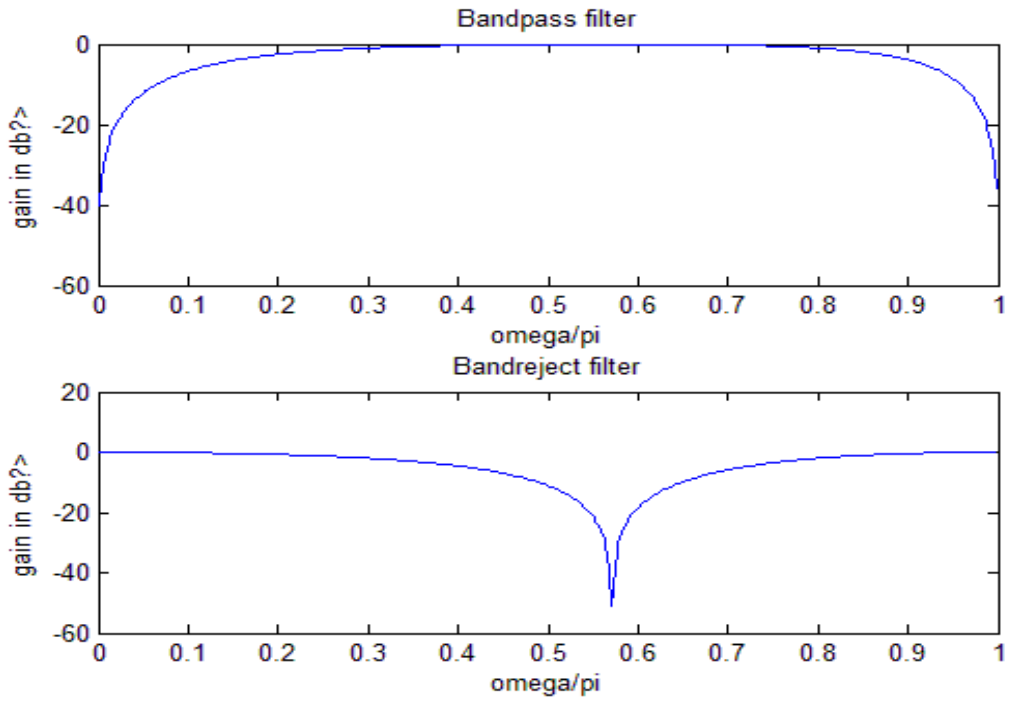
OUTPUT:

enter the pass band ripple 10

enter the stop band ripple 30

enter the pass band frequency [0.2 0.8]

enter the stop band frequency [0.4 0.7]



RESULT:

Thus the design of IIR band pass and band stop filters using Butterworth method was executed using MATLAB.

EXP. NO: 6B IIR FILTER USING CHEBYSHEV FILTER APPROXIMATION
DATE:

AIM:

To design the IIR Band pass and Band stop filters using Chebyshev approximation and find out the response of the filter by using MATLAB.

APPARATUS REQUIRED:

Personal Computer

MATLAB Software

ALGORITHM:

1. Start the program.
2. Get pass band ripple and stop band ripple.
3. Get Pass band and stop band frequency.
4. Calculate the order of the filter.
5. Plot the band pass and band stop filter.

THEORY:

Chebyshev Filter:

- ✓ The magnitude response of the chebyshev filter exhibits ripple in the pass band or stop band according to the type.
- ✓ The transition band is less as compare to butter worth filter.
- ✓ The poles on a chebyshev filter are lie on ellipse.
- ✓ The order of chebyshev filter is less than that of butter worth.

PROCEDURE:

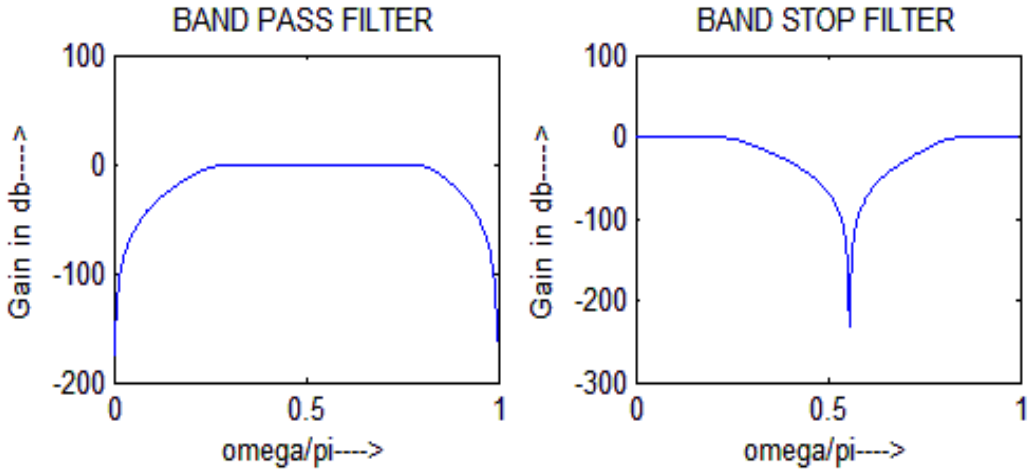
- ➢ Start the MATLAB software and create new M-file.
- ➢ Type the program in the file.
- ➢ Save & compile the program.
- ➢ Give the input data.
- ➢ Observe the output waveform.
- ➢ Thus the graph is to be plotted.

PROGRAM:

```
clear all;
clc;
close all;
format long
rp=input('enter the pass band ripple');
rs=input('enter the stop band ripple');
wp=input('enter the pass band frequency ');
ws=input('enter the stop band frequency ');
[n1,w1]=cheb1ord(wp,ws,rp,rs);
[num,den]=cheby1(n1,rp,w1);
[num1,den1]=cheby1(n1,rs,w1,'stop');
[g,w]=freqz(num,den);
[g1,w1]=freqz(num1,den1);
m=20*log10(abs(g));
m1=20*log10(abs(g1));
title('Gain response of chebyshev filter');
Subplot(2,1,1);
plot(w/pi,m);
Ylabel('gain in db?>');
Xlabel('omega/pi');
title('Bandpass filter');
Subplot(2,1,2);
plot(w/pi,m1);
Ylabel('gain in db?>');
Xlabel('omega/pi');
title('Bandreject filter');
```

OUTPUT:

```
enter the pass band ripple      1
enter the stop band ripple      2
enter the pass band frequency          [0.3 0.8]
enter the stop band frequency          [0.2 0.9]
```



RESULT:

Thus the MATLAB program for IIR filter using Chebyshev approximation for the sequence was performed.

EXP.NO:7 STUDY OF ARCHITECTURE OF DIGITAL SIGNAL PROCESSOR

DATE:

AIM:

To study the various architecture of digital signal processor TMS320C50 Kit.

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 225 MHz 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 head-phone 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 memory, or the SDRAM, and plot them 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.

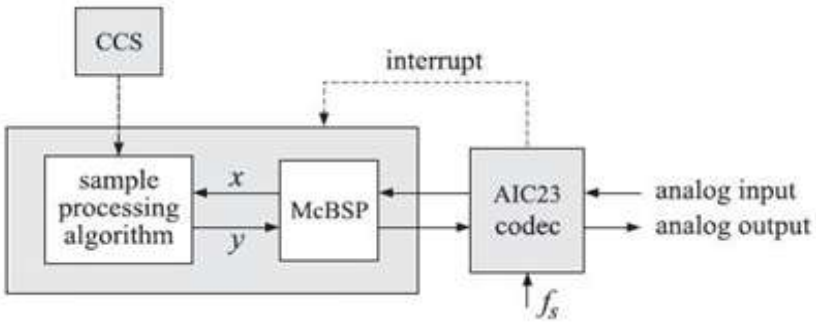


Fig: Interrupt processing

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 **short** data type in integer C). The codec actually samples the audio input in stereo, that is, it collects two samples for the left and right channels.

Architecture

The ‟54x 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.

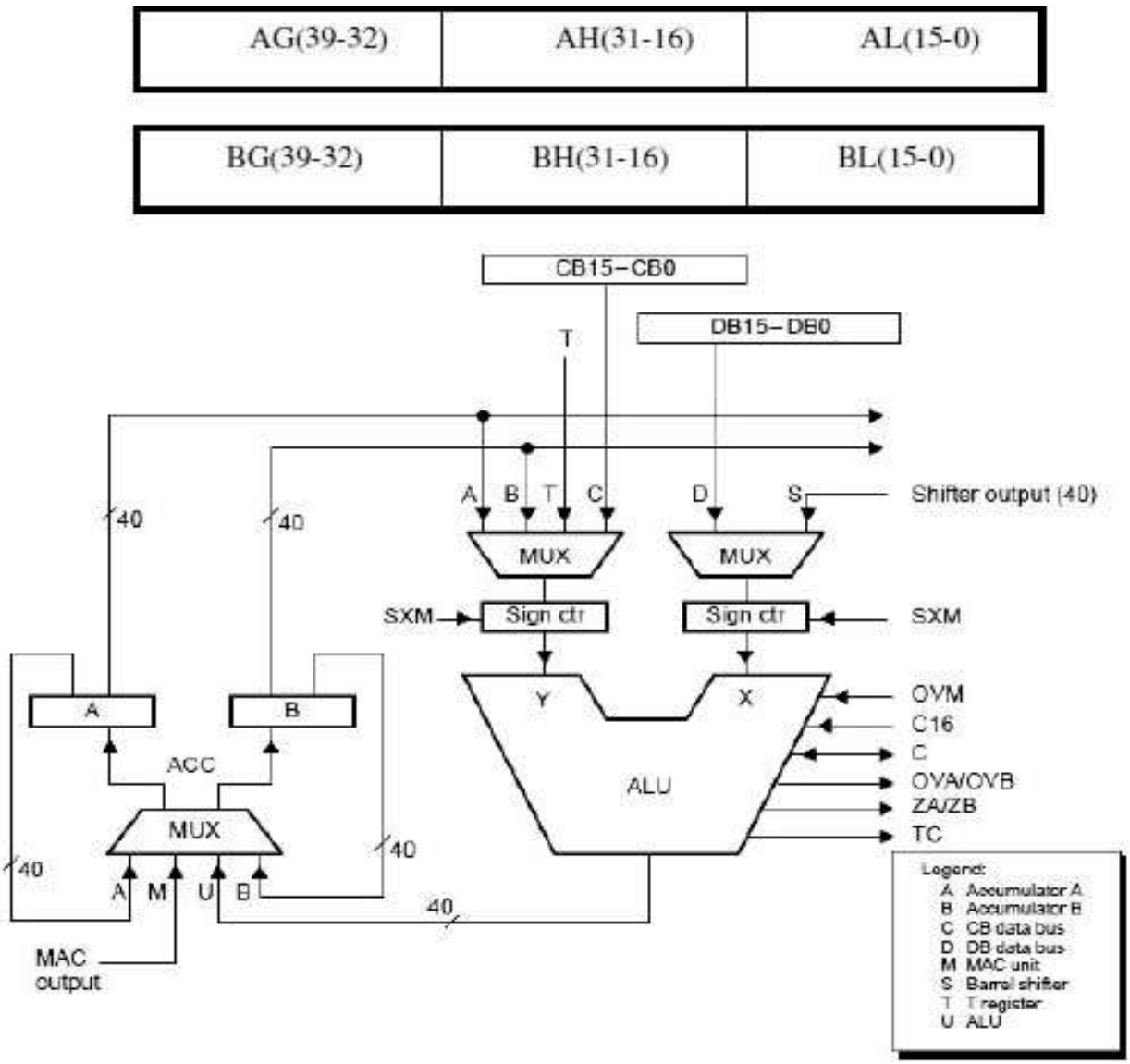


Figure Functional diagram of the central processing unit of the TMS320C54xx processors.

1.Central Processing Unit (CPU)

The CPU of the "54x 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 ‟54x 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 ‟54x‟s 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 performs 17 □ □ 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 ‟54x 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

Program control is provided by several hardware and software mechanisms: 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 include branches, calls, conditional instructions, a repeat instruction, reset, and interrupts.

The ‟54x supports both the use of hardware and software interrupts for program control. Interrupt service routines are vectored through a reloadable 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 ‟54x 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 predecrement 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 register (PMST) controls memory configurations of the ‟54x devices.

17 Power-Down Modes

There are three power-down modes, activated by the IDLE1, IDLE2, and IDLE3 instructions. In these modes, the ‟54x devices enter a dormant state and dissipate considerably less power than in normal operation. The IDLE1 instruction 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 ‟54x processor completely. This instruction stops the PLL circuitry as well as the CPU and peripherals.

RESULT:

Thus, the architecture of DSP processor TMS320C50 was studied.

EXP. NO: 8 MAC OPERATIONS USING VARIOUS ADDRESSING MODES
DATE:

AIM:

To write an assembly language program for MAC operations using various addressing modes of Processor.

TOOLS REQURIED:

- DSP hardware.
- TMS320C5X-starter kit.
- RS232 cable.

PROCEDURE:

- ✓ Start the Process.
- ✓ In the C50 debugger software.
    - Project-> New project → Save project (.dbj)
    - File → New file
    - Type the program → Save file (.asm)
    - Project → add file to project (asm file)
    - Project → add file to project (micro 50)
    - Project → build
    - Serial → Port settings → Auto detect
    - Serial →Load program → filename.asc
    - Serial→Communication window
    - Reset kit.
    - SD input address→ender
    - Give the input data
    - Reset kit
    - Go C000→ enter
    - Reset kit
    - SD output address→ enter
- ✓ Stop the Process.

PROGRAM

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
H:     B      H
```

INPUT:

A000H 0004H

A001H 0004H

OUTPUT:

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
H:      B    H
```

INPUT:

A000H 0004H

A001H 0002H

OUTPUT:

A002H 0002H

MULTIPLICATION

```
INP1  .SET  0H
INP2  .SET  1H
OUT   .SET  2H
      .MMREGS
      .TEXT
START:
      LD    #140H, DP
      RSBX  CPL
      RSBX  FRCT
      NOP
      NOP
      NOP
      NOP
      LD    #00H, A
      LD    00H, T
      MPY   01H, A
```

```
        STL     A, 02H
        STH     A, 03H
H:      B       H
```

INPUT:

A000H 0004H

A001H 0002H

OUTPUT:

A002H 0008H

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
        RPT     #0FH
        SUBC    DIVIS, A
        STL     A, OUT
H:      B       H
```

INPUT:

A000H 000AH

A001H 0002H

OUTPUT:

A002H 0005H

RESULT

Thus, the MAC operations of various addressing modes were performed by using DSP processor.

EXP. NO: 9 FIR FILTER IMPLEMENTATION

DATE:

AIM:

To implement the FIR Filter using DSP Processor.

TOOLS REQURIED:

- DSP hardware.
- TMS320C5X-starter kit.
- RS232 cable.

ALGORITHM:

- ✓ Get the sum of terms to design the filter.
- ✓ Specify the value of angular frequency from 0 to π and plots divided into 0.01 division.
- ✓ Using FIR function get the design of filter.
- ✓ Finally adjust the value and execute the program.

THEORY:

- ➢ In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time.
- ➢ Infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).
- ➢ The impulse response of N^{th}-order discrete-time FIR filter lasts for $N + 1$ samples, and then settles to zero.
    - o FIR is non recursive structure without response of FIR filter depends only on present and past input samples.

PROCEDURE:

- ✓ Start the program by clicking view click the workspace.
- ✓ Click the serial go to the port settings.
- ✓ Before auto detect reset the kit& click ok to continue.
- ✓ Click project ⟶ new. Save the file &all files will be DSP project.
- ✓ Click assembly file & save the file as ".asm".
- ✓ In left hand side, right click project & add file to project.
- ✓ In left hand side, right click command file & add file to project.
- ✓ Click built in project for the compile of program. Click ok to continue.

- ✓ In serial select the load program, download file will open and browse it. Click ok to continue.
- ✓ In serial communication window type '*sd'* space starting address, enter the input value.
- ✓ After entering the data execute the program.
- ✓ Click enter to verify the output

%FIR LOW PASS FILTER:

```
.mmregs
        .text
        B       START
CTABLE:
        .word   0196H
        .word   017EH
        .word   0EBH
        .word   00H
        .word   0FEFFH
        .word   0FE37H
        .word   0FDEDH
        .word   0FE44H
        .word   0FF35H
        .word   083H
        .word   01D3H
        .word   02B9H
        .word   02DEH
        .word   0218H
        .word   07EH
        .word   0FE6CH
        .word   0FC72H
        .word   0FB36H
        .word   0FB4CH
        .word   0FD0FH
        .word   084H
        .word   0552H
        .word   0ACBH
```

```
        .word   0100CH
        .word   0142FH
        .word   01675H
        .word   01675H
        .word   0142FH
        .word   0100CH
        .word   0ACBH
        .word   0552H
        .word   084H
        .word   0FD0FH
        .word   0FB4CH
        .word   0FB36H
        .word   0FC72H
        .word   0FE6CH
        .word   07EH
        .word   0218H
        .word   02DEH
        .word   02B9H
        .word   01D3H
        .word   083H
        .word   0FF35H
        .word   0FE44H
        .word   0FDEDH
        .word   0FE37H
        .word   0FEFFH
        .word   00H
        .word   0EBH
        .word   017EH
        .word   0196H
        Move the Filter coefficients
        from program memory to data memory
START:
        LAR    AR0,#0200H
```

```
        MAR    *,AR0
        RPT    #33H
        BLKP  CTABLE,*+
        SETC  CNF
 Input data and perform convolution
ISR:    LDP    #0AH
        IN      0,6H
      IN     0,4H
        NOP
        NOP
        NOP
        NOP
        LAR    AR1,#0300H
        LACC 0
        AND   #0FFFH
        SUB    #800H
      MAR    *,AR1
        SACL  *
        LAR    AR1,#333H
        ZAP
        RPT    #33H
        MACD          0FF00H,*-
        APAC
        LAR    AR1,#0300H
        SACH *              ;give as sach *,1 incase of overflow
        LACC *
        ADD   #800H
        SFR                       ;remove if o/p is less amplitude
        SACL  *
        OUT    *,4
        NOP
        B       ISR
        .end
```

%FIR HIGH PASS FILTER:

* Filter type: high pass filter

* Filter Order: 52

* Cutoff frequency in KHz = 3.000000

```
        .mmregs
        .text
        B       START
TABLE:
        .word   0FCD3H
        .word   05H
        .word   0FCB9H
        .word   087H
        .word   0FD3FH
        .word   01ADH
        .word   0FE3DH
        .word   0333H
        .word   0FF52H
        .word   04ABH
        .word   0FFF8H
        .word   0595H
        .word   0FFACH
        .word   0590H
        .word   0FE11H
        .word   047CH
        .word   0FB0BH
        .word   029DH
        .word   0F6BAH
        .word   0AEH
        .word   0F147H
        .word   01CH
        .word   0E9FDH
        .word   04C5H
        .word   0D882H
```

```
        .word   044BCH
        .word   044BCH
        .word   0D882H
        .word   04C5H
        .word   0E9FDH
        .word   01CH
        .word   0F147H
        .word   0AEH
        .word   0F6BAH
        .word   029DH
        .word   0FB0BH
        .word   047CH
        .word   0FE11H
        .word   0590H
        .word   0FFACH
        .word   0595H
        .word   0FFF8H
        .word   04ABH
        .word   0FF52H
        .word   0333H
        .word   0FE3DH
        .word   01ADH
        .word   0FD3FH
        .word   087H
        .word   0FCB9H
        .word   05H
        .word   0FCD3H
*       Move the Filter coefficients
*       from program memory to data memory
START:
        MAR   *,AR0
        LAR    AR0,#0200H
        RPT    #33H
```

```
        BLKP  CTABLE,*+
        SETC  CNF
* Input data and perform convolution
ISR:    LDP   #0AH
        LACC  #0
        SACL  0
        OUT   0,05              ;pulse to find sampling frequency
        IN    0,06H
        LAR   AR7,#0            ;change value to modify sampling freq.
        MAR   *,AR7
BACK:           BANZ BACK,*-
        IN    0,4
        NOP
        NOP
        NOP
        NOP
        MAR   *,AR1
        LAR   AR1,#0300H
        LACC  0
        AND   #0FFFH
        SUB   #800H
        SACL  *
        LAR   AR1,#333H
        MPY   #0
        ZAC
        RPT   #33H
        MACD          0FF00H,*-
        APAC
        LAR   AR1,#0300H
        SACH  *             ;give as sach *,1 incase of overflow
        LACC  *
        ADD   #800H
        SACL  *
```

```
        OUT   *,4

        LACC #0FFH
        SACL  0
        OUT   0,05
        NOP
        B       ISR
.end
```

% FIR BAND PASS FILTER:

```
* Filter type: bandpass filter
* Filter Order: 52
*  lower Cutoff frequency in KHz = 3.000000Hz
*  upper Cutoff frequency in KHz = 5.000000Hz
.mmregs
.text
B       START
CTABLE:
.word  024AH
        .word  010FH
        .word  0FH
        .word  0FFECH
        .word  0C6H
        .word  0220H
        .word  0312H

.word  02D3H
        .word  012FH
        .word  0FEBDH
        .word  0FC97H
        .word  0FBCBH
        .word  0FCB0H
        .word  0FE9EH
```

```
.word  029H
.word  0FFDCH
.word  0FD11H
.word  0F884H
.word  0F436H
.word  0F2A0H
.word  0F58AH
.word  0FD12H
.word  075FH
.word  01135H
.word  01732H
.word  01732H
.word  01135H
.word  075FH
.word  0FD12H
.word  0F58AH
.word  0F2A0H
.word  0F436H
.word  0F884H
.word  0FD11H
.word  0FFDCH
.word  029H
.word  0FE9EH
.word  0FCB0H
.word  0FBCBH
.word  0FC97H
.word  0FEBDH
.word  012FH
.word  02D3H
.word  0312H
.word  0220H
.word  0C6H
.word  0FFECH
```

```
        .word   0FH
        .word   010FH
        .word   024AH
*       Move the Filter coefficients
*       from program memory to data memory
START:
MAR   *,AR0
LAR   AR0,#0200H
RPT   #33H
BLKP  CTABLE,*+
SETC  CNF
ISR:  LDP   #0AH
LACC  #0
SACL  0
OUT   0,05          ;pulse to find sampling frequency
IN    0,06H
LAR   AR7,#0        ;change value to modify sampling freq.
MAR   *,AR7
BACK:       BANZ BACK,*-
IN    0,4
NOP
NOP
NOP
NOP
MAR   *,AR1
LAR   AR1,#0300H
LACC  0
AND   #0FFFH
SUB   #800H
SACL  *
LAR   AR1,#333H
MPY   #0
ZAC
RPT   #33H
```

```
MACD    0FF00H,*-
APAC
LAR   AR1,#0300H
SACH *          ;give as sach *,1 incase of overflow
LACC *
ADD   #800H
SACL  *
OUT   *,4
LACC #0FFH
SACL  0
OUT   0,05
NOP
B     ISR
.end
```

% FIR BAND REJECT FILTER:

```
* Filter Order: 52
*  lower Cutoff frequency in KHz = .000000Hz
*  upper Cutoff frequency in KHz = .000000Hz
        .mmregs
        .text
        B       START
CTABLE:
        .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
START:
        MAR   *,AR0
        LAR   AR0,#0200H
        RPT   #33H
        BLKP  CTABLE,*+
        SETC  CNF
ISR:    LDP   #0AH
        LACC  #0
        SACL  0
        OUT   0,05              ;pulse to find sampling frequency
        IN    0,06H
        LAR   AR7,#0            ;change value to modify sampling freq.
        MAR   *,AR7
BACK:       BANZ BACK,*-
        IN    0,4
        NOP
        NOP
        NOP
        NOP
        MAR   *,AR1
        LAR   AR1,#0300H
        LACC  0
        AND   #0FFFH
        SUB   #800H
```

```
SACL  *
LAR    AR1,#333H
MPY   #0
ZAC
RPT    #33H
MACD          0FF00H,*-
APAC
LAR    AR1,#0300H
SACH *              ;give as sach *,1 incase of overflow
LACC *
ADD   #800H
SACL  *
OUT    *,4
LACC #0FFH
SACL  0
OUT    0,05
NOP
B        ISR
.end
```

RESULT:

Thus the FIR Filter was implemented using DSP Processor.

EXP. NO: 10 IIR FILTER IMPLEMENTATION

DATE:

AIM:

To implement the IIR Filter using DSP Processor.

TOOLS REQURIED:

- DSP hardware.
- TMS320C5X-starter kit.
- RS232 cable.

ALGORITHM:

- ✓ Get the sum of terms to design the filter.
- ✓ Specify the value of angular frequency from 0 to π and plots divided into 0.01 division.
- ✓ Using IIR function gets the design of filter.
- ✓ Finally adjust the value and execute the program.

THEORY:

- In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time.
- Infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying).
- The impulse response of N^{th}-order discrete-time FIR filter lasts for $N + 1$ samples, and then settles to zero.
    - FIR is non recursive structure without response of FIR filter depends only on present and past input samples.

PROCEDURE:

- ✓ Start the program by clicking view click the workspace.
- ✓ Click the serial go to the port settings.
- ✓ Before auto detect reset the kit& click ok to continue.
- ✓ Click project ⟶ new. Save the file &all files will be DSP project.
- ✓ Click assembly file & save the file as ".asm".
- ✓ In left hand side, right click project & add file to project.
- ✓ In left hand side, right click command file & add file to project.

- ✓ Click built in project for the compile of program. Click ok to continue.
- ✓ In serial select the load program, download file will open and browse it. Click ok to continue.

- ✓ In serial communication window type '*sd'* space starting address, enter the input value.
- ✓ After entering the data execute the program.
- ✓ Click -enter- to verify the output.

PROGRAM FOR IIR FILTER:

%Low Pass Filter

```
.MMREGS
      .TEXT
TEMP   .SET   0
INPUT  .SET   1
T1     .SET   2
T2     .SET   3
T3     .SET   4
;
K      .SET   315eh
M      .SET   4e9fh

;cut-off freq is 1Khz. = Fc
;sampling frequency is 100 æs (ie) 0.1ms.
; a = 2 * (355/113) * 1000 = 6283.18/1000 = 6.28 ;; divide by 1000 for secs
; K = aT/(1+aT) = 6.28*0.1 / (6.28*0.1+1) = 0.3857
; M = 1/(1+aT) = 1 / (6.28*0.1+1) = 0.61425
;convert to Q15 format
; K = K * 32767 = 12638.23 = 315Eh
; M = M * 32767 = 20127.12 = 4E9Fh
;Sampling Rate is 100 æs & Cut off Frequency is 1 Khz
      LDP    #100H
      LACC   #0
      SACL   T1
      SACL   T2
```

```
      SACL    TEMP
      OUT     TEMP,4          ;CLEAR DAC BEFORE START TO WORK
LOOP:
      LACC    #0
      SACL    TEMP
      OUT     TEMP,5          ;OUTPUT LOW TO DAC2 TO CALCULATE TIMING;
      IN      TEMP,06           ;SOC;
      LAR     AR7,#30h        ;CHANGE VALUE TO MODIFY SAMPLING FREQ
                              ;sampling rate 100ms.
       MAR     *,AR7
         BACK:          BANZ BACK,*-
;
      IN      INPUT,4         ;INPUT DATA FROM ADC1
         NOP
         NOP
;
      LACC    INPUT
      AND     #0FFFH
      SUB     #800h
      SACL    INPUT
;
      LT      INPUT
      MPY     #K
      PAC
      SACH    T1,1
;;;CALL MULT ----MULTIPLICATION TO BE DONE WITH K
;;RESULT OF MULT IN T1
;
      LT      T2      ;PREVIOUS RESULT IN T2
      MPY     #M
      PAC
      SACH    T3 ,1
;;;CALL MULT ----MULTIPLICATION TO BE DONE WITH M
```

```
;;RESULT OF MULT IN T3+
      LACC    T1
      ADD     T3

      SACL    T2
      ADD     #800h
      SACL    TEMP
      OUT     TEMP,4          ;OUTPUT FILTER DATA TO DAC1
      LACC    #0FFH
      SACL    TEMP
      OUT     TEMP,5          ;OUTPUT HIGH TO DAC2 TO CALCULATE TIMING
      B       LOOP
```

%High Pass Filter:

```
        .MMREGS
        .TEXT
START:
        LDP #100H
        LACC #00H
        SACL 00H
        SACL 01H
        SACL 02H
        SACL 03H
        SACL 04H
        SACL 05H
LOOP:
        LACC #00H
        SACL 00H
        IN 0,06H
        LAR AR7,#30H
        MAR *,AR7
BACK:       BANZ BACK,*-
;       LT 01H
;       MPY #0FFFFB5DEH
```

```
;       PAC
;       SACH 05H,1
        IN 0,04H
        NOPS

        NOP
        NOP
        NOP
        LT 01H
;       MPY #0FFFFB5DEH
        MPY #4A22H
;       MPY #315EH
        PAC
        SACH 05H,1
        LACC 00H
        AND #0FFFH
        XOR #800H
        SUB #800H
        SACL 00H
        SACL 01H
        ZAP
        LT 00H
;       DMOV 00H
;       LTD 00H
        MPY #4A22H
;       MPY #315EH
        PAC
        SACH 02H,1
        LT 03H
        MPY #1446H
;       MPY #4E9FH
        PAC
        SACH 04H,1
```

```
        LACC 02H
        ADD 04H
        SUB 05H
        SACL 03H
        ADD #800H

        SACL 00H
;       OUT 00H,1AH
        OUT 0,04H
        B LOOP
        NOP
        NOP
H:      B H
```

%Band Pass Filter:

```
        .MMREGS
.TEXT
START:
        LDP #100H
        NOP
        NOP
        NOP
        LACC #00H
        LAR AR0,#00FFH
        LAR AR1,#8000H
        MAR *,AR1
LOOP:
        SACL *+,AR0
        BANZ LOOP,AR1

LOOP1:
        LACC #00H
        SACL 00H
```

```
        IN 0,06H
        LAR AR7,#30H
        MAR *,AR7
BACK:   BANZ BACK,*-
        IN 0,04H
        NOP

        NOP
        NOP
        NOP

        LT 04H
        MPY #2FC4H
;       MPY #05F8H
        PAC
        SACH 24H,0
;       SACH 24H,4

        LT 03H
        MPY #99B2H
;       MPY #1336H
        PAC
        SACH 23H,0
;       SACH 23H,4
        LT 02H
        MPY #0DB29H
;       MPY #1B65H
        PAC
        SACH 22H,0
;       SACH 22H,4
        LT 01H
        MPY #99B2H
;       MPY #1336H
```

```
        PAC
        SACH 21H,0
;       SACH 21H,4
        LACC 03H
        SACL 04H
        LACC 02H
        SACL 03H

        LACC 01H
        LACC 02H

        LACC 00H
        AND #0FFFH
        XOR #800H
        SUB #800H
        SACL 00H
        SACL 01H
        ZAP
;       DMOV 03H
;       DMOV 02H
;       DMOV 01H
        LT 00H
        MPY #2FC4H
;       MPY #05F8H
        PAC
        SACH 20H,0
;       SACH 20H,4
;
        LT 73H
        MPY #2A22H
;       MPY #0544H
        PAC
        SACH 63H,0
```

```
;       SACH 63H,4
        LT 72H
        MPY #6761H
;       MPY #0CECH
        PAC
        SACH 62H,0
;       SACH 62H,4
        LT 71H

        MPY #0B6E8H
;       MPY #16DDH
        PAC
        SACH 61H,0
;       SACH 61H,4

        LACC 72H
        SACL 73H
        LACC 71H
        SACL 72H
        LACC 70H
        SACL 71H
;       DMOV 72H
;       DMOV 71H
;       LTD 70H
        LT 70H
        MPY #0F184H
;       MPY #1E30H
        PAC
        SACH 60H,0
;       SACH 60H,4
        LACC 20H
        SUB  21H
        ADD  22H
```

```
        SUB  23H
        ADD  24H
        ADD  60H
        SUB  61H
        ADD  62H
        SUB  63H
        SACL 70H
        ADD #800H
        SACL 00H

        OUT 00H,1AH
        IN 0,04H
        B  LOOP1
        NOP
        NOP
H:      B       H
```

% Band Reject Filter:

```
;IIR BANDREJECT FILTER
;.MMREGS
.TEXT
START:
        LDP #100H
        LACC #00H
        LAR AR0,#00FFH
        LAR AR1,#8000H
        MAR *,AR1
LOOP:
        SACL *+,AR0
        BANZ LOOP,AR1
LOOP1:
        LACC #00H
        SACL 00H
```

```
        IN 0,06H
        LAR AR7,#30H
        MAR *,AR7
BACK:   BANZ BACK,*-
        IN 0,04H
        NOP
        NOP
        NOP
        NOP
        LT 04H

        MPY #003BH

        PAC
;       SACH 24H,0
;       RPT #0BH
;       SFR
        SACH 24H,4
        LT 03H
        MPY #0000H
        PAC
;       RPT #0BH
;       SFR
;       SACH 23H,0
        SACH 23H,4
        LT 02H
        MPY #0077H
        PAC
;       RPT #0BH
;       SFR
;       SACH 22H,0
        SACH 22H,4
        LT 01H
```

```
        MPY #0000H
        PAC
;       RPT #0BH
;       SFR
;       SACH 21H,0
        SACH 21H,4
        LACC 03H
        SACL 04H
        LACC 02H
        SACL 03H

        LACC 01H
        LACC 02H

        LACC 00H
        AND #0FFFH
        XOR #800H
        SUB #800H
        SACL 00H
        SACL 01H
        ZAP
        LT 00H
        MPY #003BH
        PAC
;       RPT #0BH
;       SFR
;       SACH 20H,0
        SACH 20H,4
        LT 73H
        MPY #0B04H
        PAC
;       RPT #0BH
;       SFR
```

```
;       SACH 63H,0
        SACH 63H,4
        LT 72H
        MPY #1226H
        PAC
;       RPT #0BH
;       SFR
;       SACH 62H,0
        SACH 62H,4

LT 71H
        MPY #21A3H
        PAC
;       RPT #0BH
;       SFR
;       SACH 61H,0
        SACH 61H,4
        LACC 72H
        SACL 73H
        LACC 71H
        SACL 72H
        LACC 70H
        SACL 71H
        LT 70H
        MPY #15E9H
        PAC
;       RPT #0BH
;       SFR
;       SACH 60H,0
        SACH 60H,4
        LACC 20H
        ADD 21H
        SUB 22H
```

```
        ADD 23H
        ADD 24H
        ADD  60H
        SUB  61H
        ADD  62H
        SUB  63H
        SACL 70H
        ADD #800H
        SACL 00H
;       OUT 00H,1AH

        OUT 0,04H
        B  LOOP1
        NOP
        NOP
H:      B  H
```

RESULT:

Thus the IIR Filter was implemented using DSP Processor.

EXP. NO: 11 IMPLEMENT UP-SAMPLING AND DOWN-SAMPLING

DATE:

AIM

To write a program to implement Up-sampling and Down-sampling operation using MATLAB.

APPARATUS REQUIRED:

Personal Computer

MATLAB Software

ALGORITHM:

1. Start the program.
2. Get the number of samples.
3. Calculate the output using the interpolator and decimator formula.
4. Stop the process.

THEORY:

Decimation is the process of reducing the sampling rate. In practice, this usually implies low pass-filtering a signal, then throwing away some of its samples. "Down sampling" is a more specific term which refers to just the process of throwing away samples, without the low pass filtering operation.

The decimation factor is simply the ratio of the input rate to the output rate. It is usually symbolized by "D", so input rate / output rate D.A signal can be down sampled (without doing any filtering) whenever it is "oversampled", that is, when a sampling rate was used that was greater than the Nyquist criteria required.

Specifically, the signal's highest frequency must be less than half the post-decimation sampling rate.

Interpolation is the process of increasing the sampling rate. In practice, this usually implies low pass-filtering a signal, then throwing away some of its samples. "Up sampling" is a more specific term which refers to just the process of throwing away samples, without the low pass filtering operation. The interpolation factor is simply the ratio of the input rate to the output rate. It is usually symbolized by "M", so input rate / output rate=M.

A signal can be up sampled (without doing any filtering) whenever it is "under sampled", that is, when a sampling rate was used that was greater than the Nyquist criteria required. Specifically, the signal's highest frequency must be double the post-interpolation sampling rate.

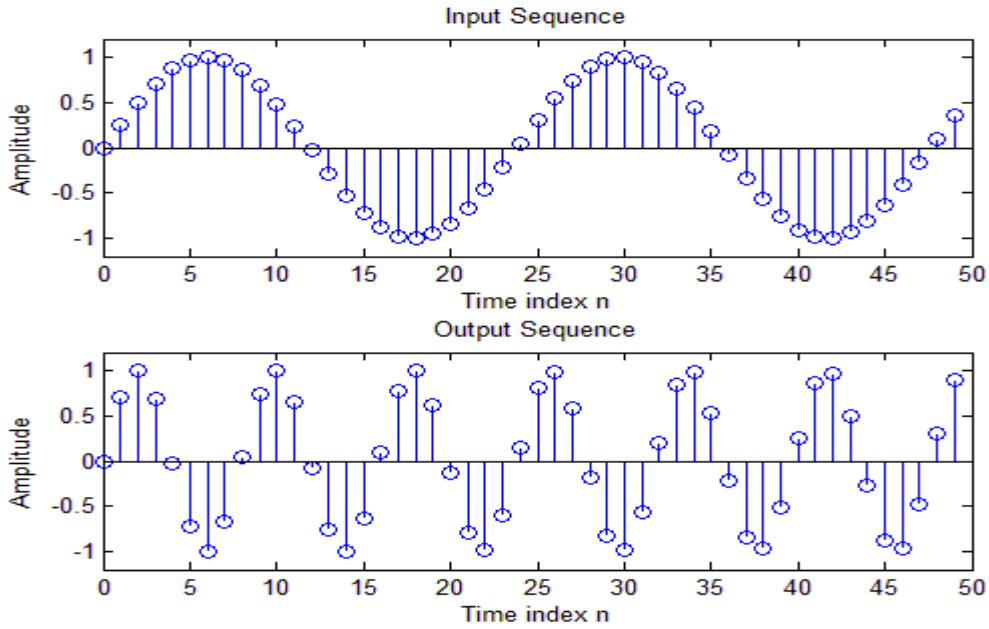
PROCEDURE:

- Start the MATLAB software and create new M-file.
- Type the program in the file.
- Save & compile the program.
- Give the input data.
- Observe the output waveform.
- Thus the graph is to be plotted

PROGRAM:

```
%down sampler
    clf;
    n = 0: 49;
    m = 0: 50*3 - 1;
    x = sin(2*pi*0.042*m);
    y = x([1: 3: length(x)]);
    subplot(2,1,1)
    stem(n, x(1:50)); axis([0 50 -1.2 1.2]);
    title('Input Sequence');
    xlabel('Time index n');
    ylabel('Amplitude');
    subplot(2,1,2)
    stem(n, y); axis([0 50 -1.2 1.2]);
    title('Output Sequence');
    xlabel('Time index n');
    ylabel('Amplitude');
```

OUTPUT:



PROGRAM

%UP SAMPLING

```
clc;
clear all;
close all;
N=125;
N= 0 : 1: N -1 ;
X= sin (2* pi * n/15);
L=2;
figure (1)
stem (n,x);
grid on;
xlabel (´No. of samples´);
ylabel (´Amplitude´);
title (´ Original sequence´);
x1 =[zeros (1, L*N)];
n1 = 1:1: L*N;
j= 1:L: L*N;
x1 (j) =x;
figure(2)
stem (n1-1, x1);
```

grid on;

xlabel (´No. of samples´);

ylabel (´Amplitude´);

title (´ Unsampled sequence´);

a=1;

b=fir1(5, 0.5, ´low´);

y=filter (b,a, x_1);

figure (3)

stem (n_1-1, y);

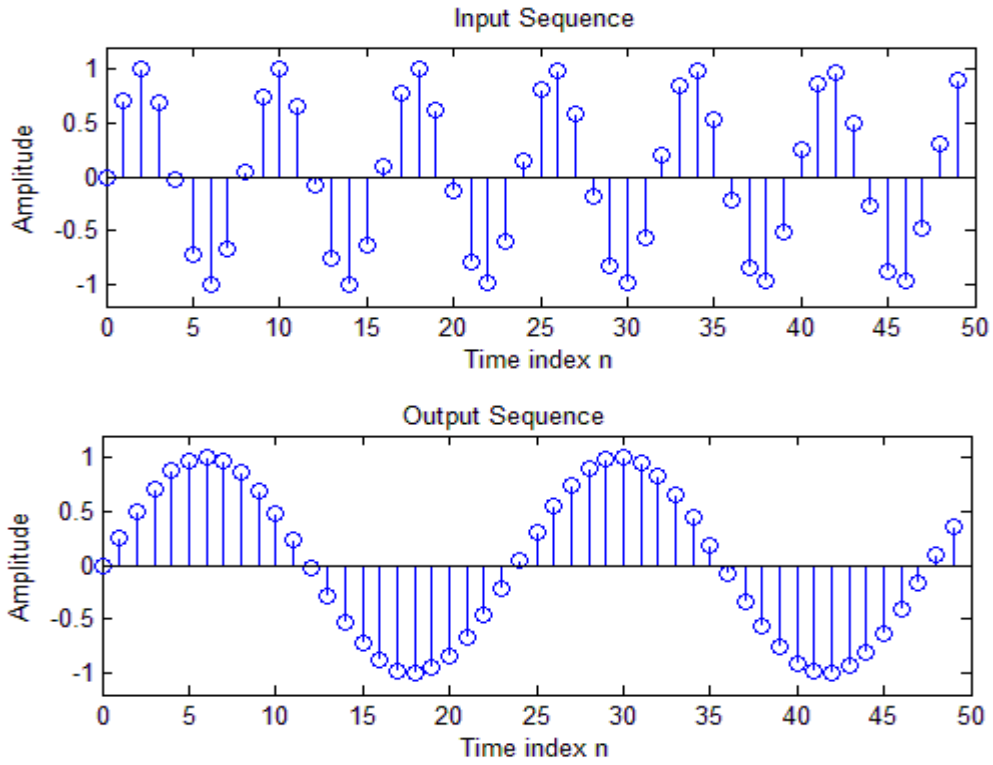
grid on;

xlabel(´ No. of samples´);

ylabel (´Amplitude´);

title (´ Interpolated sequence´);

OUTPUT:



RESULT:

Thus the Up sampling and down-sampling operations were implemented using MATLAB.