DIGITAL SIGNAL PROCESSING LAB MANUAL

G.DEEPIKA M.E.,

REGULATION 2021

9 789334 001556

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

REGULATION – 2021

EC3492 - DIGITAL SIGNAL PROCESSING LABORATORY

Mrs.G.DEEPIKA.,M.E.,

Assistant Professor/ Electronics and Communication Engineering

Annai Mira College of Engineering and Technology

Ranipet – 632 517

GENERAL GUIDELINES AND SAFETY NSTRUCTIONS

1. Sign in the log register as soon as you enter the lab and strictly observe your lab timings.
2. Strictly follow the written and verbal instructions given by the teacher / Lab Instructor. If you do not understand the instructions, the handouts and the procedures, ask the instructor or teacher.
3. **Never work alone!** You should be accompanied by your laboratory partner and / or the instructors / teaching assistants all the time.
4. It is mandatory to come to lab in a formal dress and wear your ID cards.
5. Do not wear loose-fitting clothing or jewels in the lab. Rings and necklaces are usual excellent conductors of electricity.
6. Mobile phones should be switched off in the lab. Keep bags in the bag rack.
7. Keep the labs clean at all times, no food and drinks allowed inside the lab.
8. Intentional misconduct will lead to expulsion from the lab.
9. Do not handle any equipment without reading the safety instructions. Read the handout and procedures in the Lab Manual before starting the experiments.
10. Do your wiring, setup, and a careful circuit checkout before applying power. Do not make circuit changes or perform any wiring when power is on.
11. Avoid contact with energized electrical circuits.
12. Do not insert connectors forcefully into the sockets.
13. **NEVER** try to experiment with the power from the wall plug.
14. Immediately report dangerous or exceptional conditions to the Lab instructor / teacher: Equipment that is not working as expected, wires or connectors are broken, the equipment that smells or "smokes". If you are not sure what the problem is or what's going on, switch off the Emergency shutdown.
15. Never use damaged instruments, wires or connectors. Hand over these parts to the Lab instructor/Teacher.
16. Be sure of location of fire extinguishers and first aid kits in the laboratory.
17. After completion of Experiment, return the bread board, trainer kits, wires, CRO probes and other components to lab staff. Do not take any item from the lab without permission.
18. Observation book and lab record should be carried to each lab. Readings of current lab experiment are to be entered in Observation book and previous lab experiment should be written in Lab record book. Both the books should be corrected by the faculty in each lab.
19. Special Precautions during soldering practice

    a. Hold the soldering iron away from your body. Don't point the iron towards you. b. Don't use a spread solder on the board as it may cause short circuit.

    c. Do not overheat the components as excess heat may damage the components/board.

    d. In case of burn or injury seek first aid available in the lab or at the college dispensary.

PREFACE

This book on "**DIGITAL SIGNAL PROCESSING LABORATORY MANUAL (Electronics and communication Engineering)**" covers the complete syllabus prescribed by the Anna University, Chennai for the fourth semester **B.E/ B.Tech**. Degree course under **Outcome Based Education Credit System with the new regulation 2021**.

This book covers Discrete time sequences,Linear and circular convolution, Design of FIR((LPF/HPF/BPF/BSF)) and IIR filters((LPF/HPF/BPF/BSF)).

We hope that this book will be useful to both teachers and students. Finally we would request the readers to kindly send their valuable comments and suggestions towards the improvement of the manual and the same will be gratefully acknowledge.

Any suggestion from the reader for the betterment of this book can be dropped into flytodeepi@gmail.com.

Mrs.G.DEEPIKA.,M.E.,

ACKNOWLEDGEMENT

We are thankful to and fortunate enough to get constant encouragement, support and guideline from Chairman **Thiru.S.Ramadoss** , Secretary & Treasurer **Mr.G.Thamotharan** for his blessings to complete the book successfully.

We would not forget to remember our Principal **Dr.T.K.Gopinathan** and Vice-Principal **Dr.D.Saravanan** for his constant assistance in preparing this book.

ANNAI MIRA COLLEGE OF ENGINEERINGAND TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

LAB MANUAL
(Regulation - 2021)

Subject Code / Name : EC3492 /DIGITAL SIGNAL PROCESSING LAB
Semester/Year : IV/II – ECE

PREPARED BY
Mrs.G.DEEPIKA.,M.E
Assistant Professor / ECE

APPROVED BY
Dr.V.SRIVIDHYA.M.E.,Ph.D
HOD / ECE

Department of Electronics and

Communication Engineering

EC 3492 DIGITAL SIGNAL PROCESSING

LABORATORY

List of Experiments

MATLAB / EQUIVALENT SOFTWARE PACKAGE/ DSP PROCESSOR BASED

IMPLEMENTATION

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
7. Study of architecture of Digital Signal Processor
8. Perform MAC operation using various addressing modes
9. Generation of various signals and random noise
10. Design and demonstration of FIR Filter for Low pass, High pass, Band pass and Band stop filtering
11. Design and demonstration of Butter worth and Chebyshev IIR Filters for Low pass, High pass, Band pass and Band stop filtering
12. Implement an Up-sampling and Down-sampling operation in DSP Processor

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

AIM:

To write a program to generate the elementary discrete time sequences using MATLAB.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

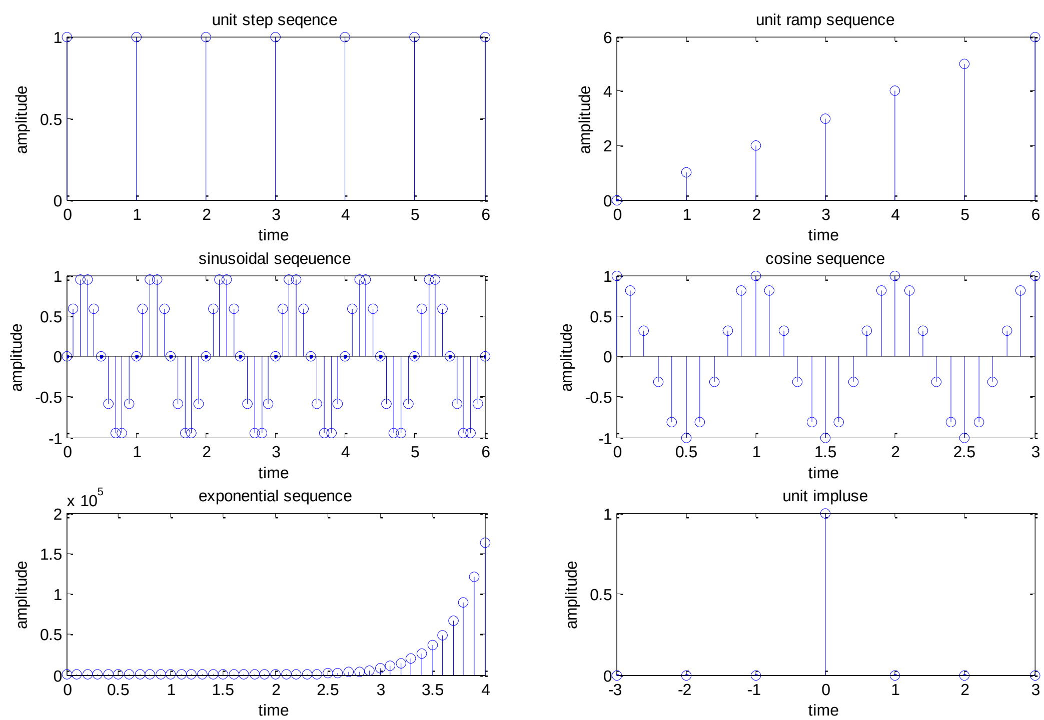
- Get the input for the required sequences.
- Generate the sequence.
- Plot the corresponding sequences.

PROGRAM:

```
%Program for sine wave

Clc;

Clearall;
Closeall;
N = 7;
n = 0:1:N-1;
y = ones(1,N);
subplot(3,2,1);
stem(n,y);
xlabel('time');
ylabel('amplitude');
title('unit step seqence');
N1 = 5;
n1 = 0:1:N-1;
y1 = n1;
subplot(3,2,2);
stem(n1,y1);
xlabel('time');
ylabel('amplitude');
title('unit ramp sequence');
N2 = 6;
n2 = 0:0.1:N-1;
```

```
y2 = sin(2*pi*n2);
subplot(3,2,3);
stem(n2,y2);
xlabel('time');
ylabel('amplitude');
title('sinusoidal seqeuence');
N3 = 4;
n3 = 0:0.1:N3-1;
y3 = cos(2*pi*n3);
subplot(3,2,4);
stem(n3,y3);
xlabel('time');
ylabel('amplitude');
title('cosine sequence');
N4 = 5;
n4 = 0:0.1:N4-1;
a = 3;
y4 = exp(a*n4);
subplot(3,2,5);
stem(n4,y4);
xlabel('time');
ylabel('amplitude');
title('exponential sequence');
n5 = -3:1:3;
y5 = [zeros(1,3),ones(1,1),zeros(1,3)];
subplot(3,2,6);
stem(n5,y5);
xlabel('time');
ylabel('amplitude');
title('unit impluse');
```

OUTPUT:



RESULT:

Thus the elementary discrete time sequences are generated and plottedusing MATLAB.

EXP.NO: 2 **LINEAR AND CIRCULAR CONVOLUTIONS**

AIM:

To write a program to performLinear and Circular Convolution of two sequences using MATLAB.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the input sequence x (n) and impulse sequence h (n).
- Perform the convolution of two sequences.
- Plot the convoluted sequences.

PROGRAM:

LINEAR CONVOLUTION:

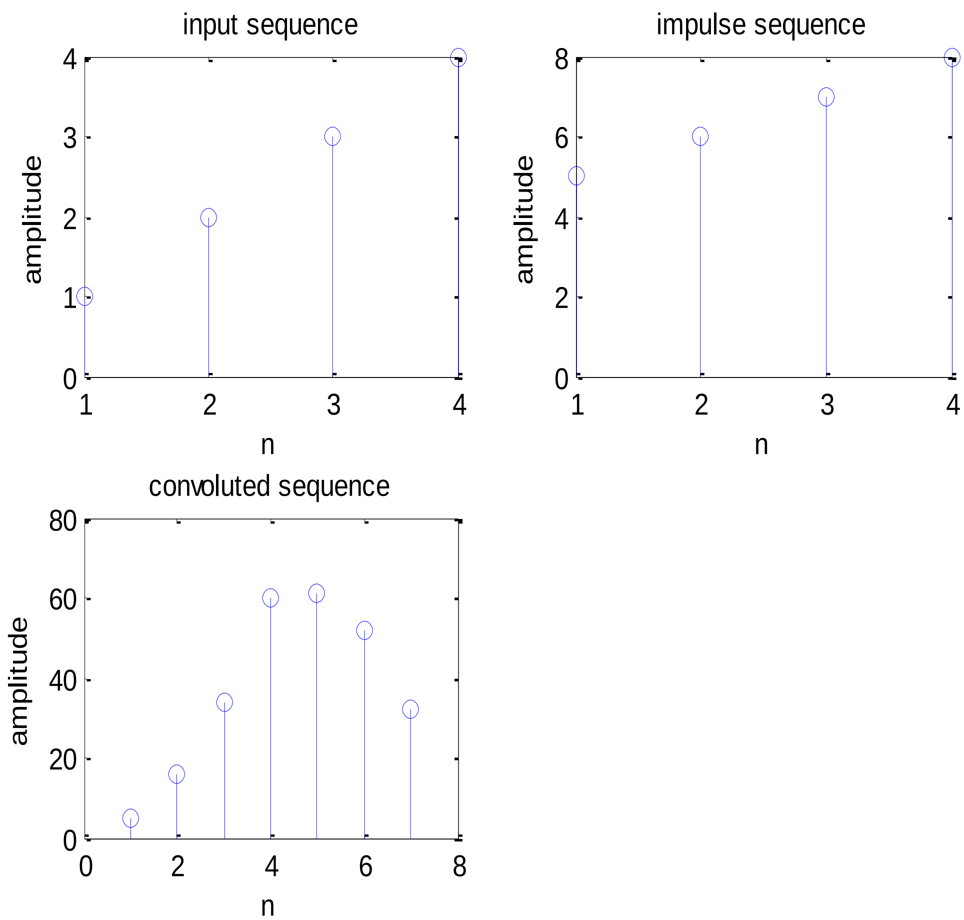
```
clc;
clearall;
closeall;
x=input('Enter the input sequence');
h=input('Enter the impulse sequence');
y=conv(x,h);
subplot(2,2,1);
stem(x);
xlabel('n');
ylabel('amplitude');
title('input sequence');
subplot(2,2,2);
stem(h);
xlabel('n');
ylabel('amplitude');
title('impulse sequence');
subplot(2,2,3);
stem(y);
xlabel('n');
ylabel('amplitude');
title('convoluted sequence');
disp('Convoluted sequence');y
```

INPUT :

Enter the input sequence[1 2 3 4]
Enter the impulsesequence[5 6 7 8]
Convoluted sequence

y =   5   16   34   60   61   52   32

OUTPUT:



CIRCULAR CONVOLUTION

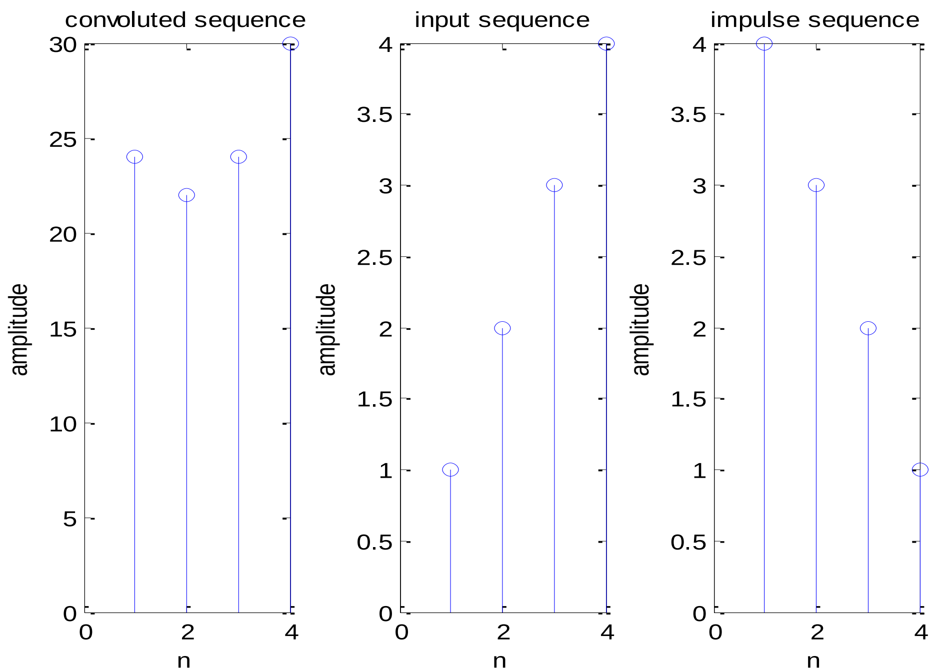
```
clc;
clearall;
closeall;
x=input('Enter the input sequence');
h=input('Enter the impulse sequence');
N1=length(x);
N2=length(h);
N=max(N1,N2);
N3=N1-N2;
if (N3>=0);
    h=[h,zeros(1,N3)];
else
    x=[x,zeros(1,N3)];
end
for n=1:N;
y(n)=0;
for i=1:N;
            j=n-i+1;
if (j<=0)
                j=N+j;
end
y(n)=y(n)+[x(i)*h(j)];
end
end
subplot(1,3,1);
stem(y);
xlabel('n');
ylabel('amplitude');
title('convoluted sequence');
disp('Convoluted sequence');y
subplot(1,3,2);
stem(x);
xlabel('n');
ylabel('amplitude');
title('input sequence');
subplot(1,3,3);
stem(h);
xlabel('n');
ylabel('amplitude');
title('impulse sequence');
```

INPUT:

Enter the inputsequence[1 2 3 4]
Enter the impulsesequence[4 3 2 1]
Convoluted sequence

y =   24   22   24   30

OUTPUT :



RESULT:

Thus the linear and circular convolutions of two sequences wereperformed using MATLAB.

EXP NO: 3 **AUTO CORRELATION AND CROSS CORRELATION**

AIM:

To write a program to perform the Autocorrelation and cross correlation of two sequences using MATLAB.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the required input sequences.
- Perform the correlation of two sequences.
- Plot the correlated sequences.

PROGRAM:

AUTOCORRELATION:

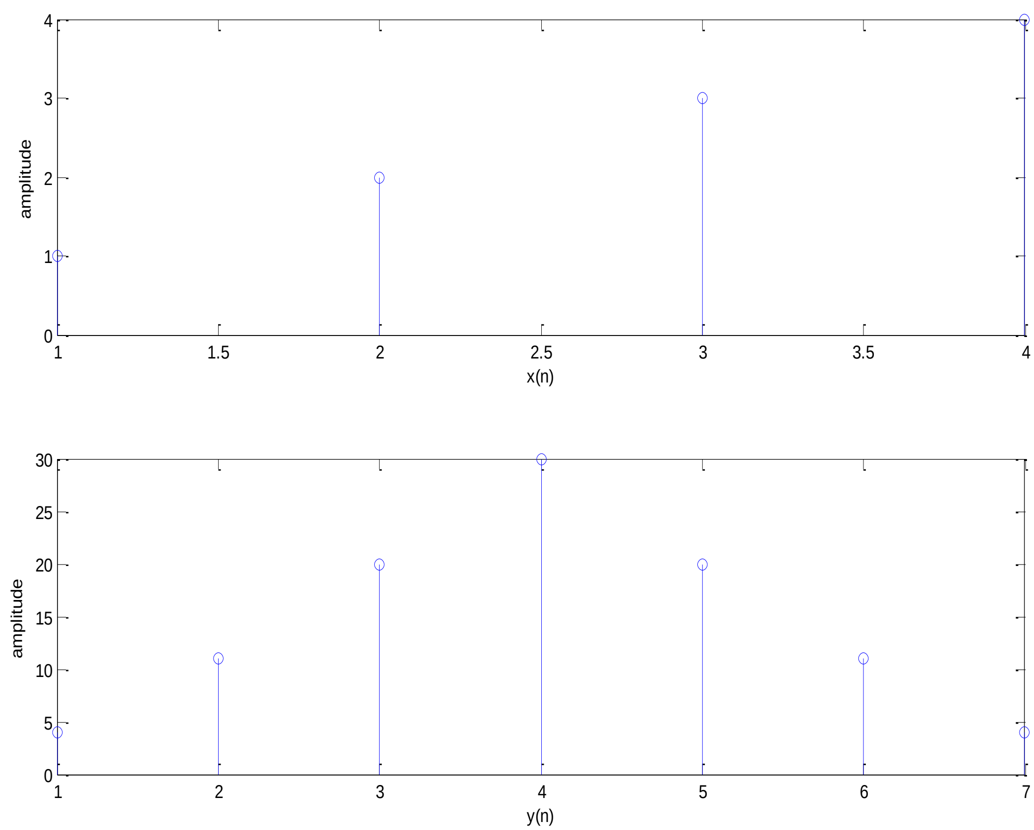
```
clc;
clearall;
closeall;
x=input('Enter the input sequence');
y=xcorr(x,x);
subplot(2,1,1);
stem(x);
ylabel('amplitude');
xlabel('x(n)');
subplot(2,1,2);
stem(y);
ylabel('amplitude');
xlabel('y(n)');
disp('The resultant signal is');y
```

INPUT:

Enter the inputsequence[1 2 3 4]
The resultant signal is

y =   4.0000   11.0000   20.0000   30.0000   20.0000   11.0000   4.0000

OUTPUT:



CROSS CORRELATION:

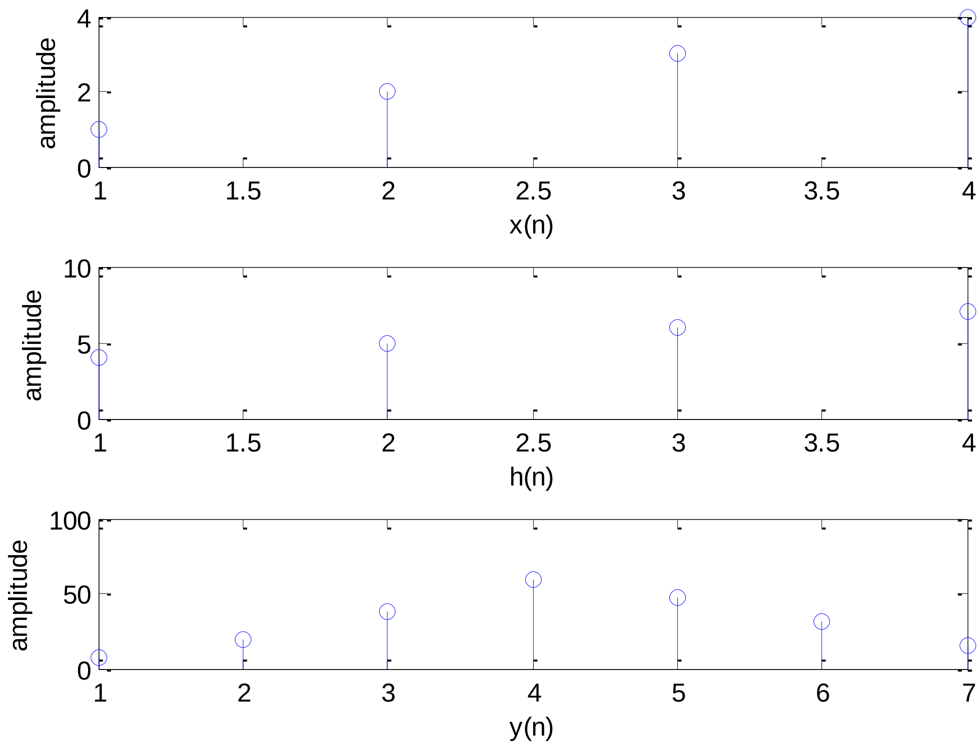
```
clc;
clearall;
closeall;
x=input('Enter the first sequence');
h=input('Enter the second sequence');
y=xcorr(x,h);
subplot(3,1,1);
stem(x);
ylabel('amplitude');
xlabel('x(n)');
subplot(3,1,2);
stem(h);
ylabel('amplitude');
xlabel('h(n)');
subplot(3,1,3);
stem(y);
ylabel('amplitude');
xlabel('y(n)');
disp('The resultant signal is:');y
```

INPUT:

Enter the first sequence [1 2 3 4]
Enter the second sequence[4 5 6 7]
The resultant signal is:

y = 7.0000 20.0000 38.0000 60.0000 47.0000 32.0000 16.0000

OUTPUT:



RESULT:

Thus the autocorrelation and cross correlation of two sequences were performed using MATLAB.

EXP.NO:4 **FREQUENCY ANALYSIS USING DFT**

AIM:

To writea MATLAB program for frequency analysis using DFT.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the input sequence x (n).
- Obtain DFT of the input sequence(resultant sequence) using FFT.
- Plot the resultant sequence.
- Calculate the magnitude and phase values of resultant signal.
- Plot the magnitude and phase plots.

PROGRAM:

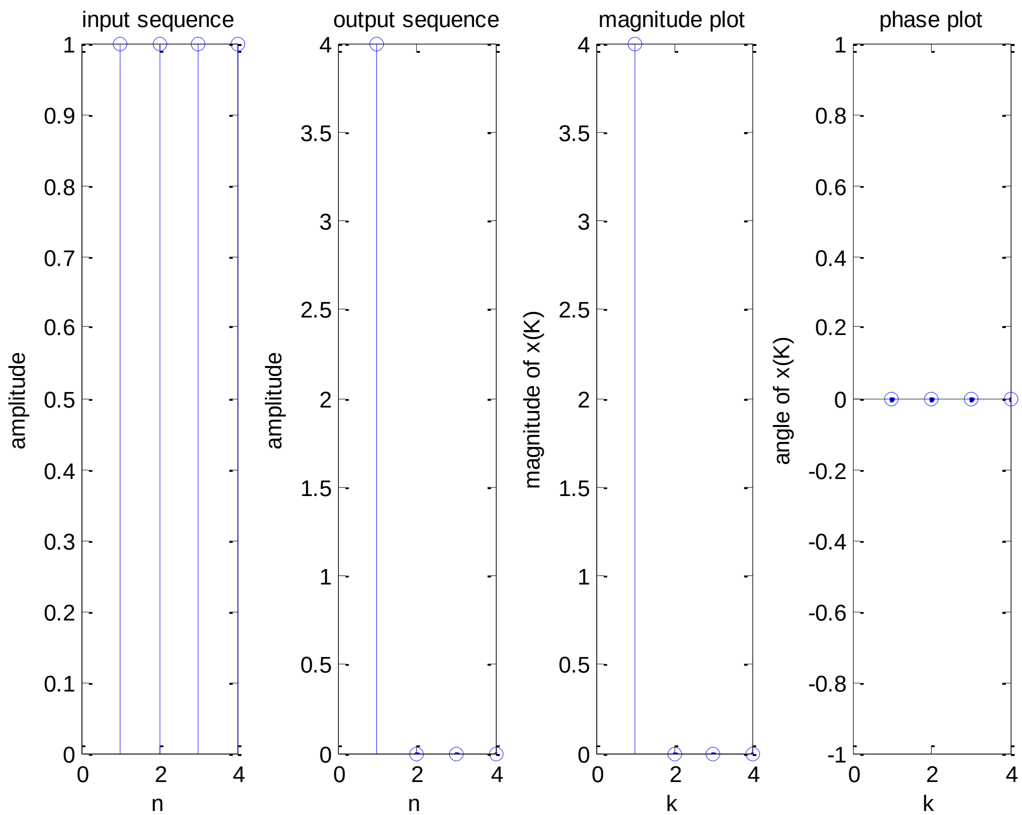
```
clc;
clearall;
closeall;
xn=input('enter the input sequence');
XK = fft (xn);
subplot(1,4,1);
stem(xn);
xlabel('n');
ylabel('amplitude');
title('input sequence');
subplot(1,4,2);
stem(XK)
xlabel('n');
ylabel('amplitude');
title('output sequence');
disp('resultant sequence XK');XK
subplot(1,4,3);
stem(abs(XK));
xlabel('k');
ylabel('magnitude of x(K)');
title('magnitude plot');
subplot(1,4,4);
stem(angle(XK));
xlabel('k');
ylabel('angle of x(K)');
title('phase plot');
```

INPUT:

Enter the input sequence[1 1 1 1]
Resultant sequence XK

XK =   4   0   0   0

OUTPUT:



RESULT:

Thus the frequency analysis using DFT was performed using MATLAB.

EXP.NO:5(a) **DESIGN OF FIR FILTER USING HAMMING WINDOW**

AIM:

To write a MATLAB program to design a FIR filters(LPF/HPF/BPF/BSF) and demonstrates the filtering operation using hamming window.

SOFTWARE REQUIRED

MATLAB R2014a

ALGORITHM:

- Get the FIR filter specifications.
- Obtain the filter coefficients using window function.
- Plot the frequency response of the filters.

PROGRAM:

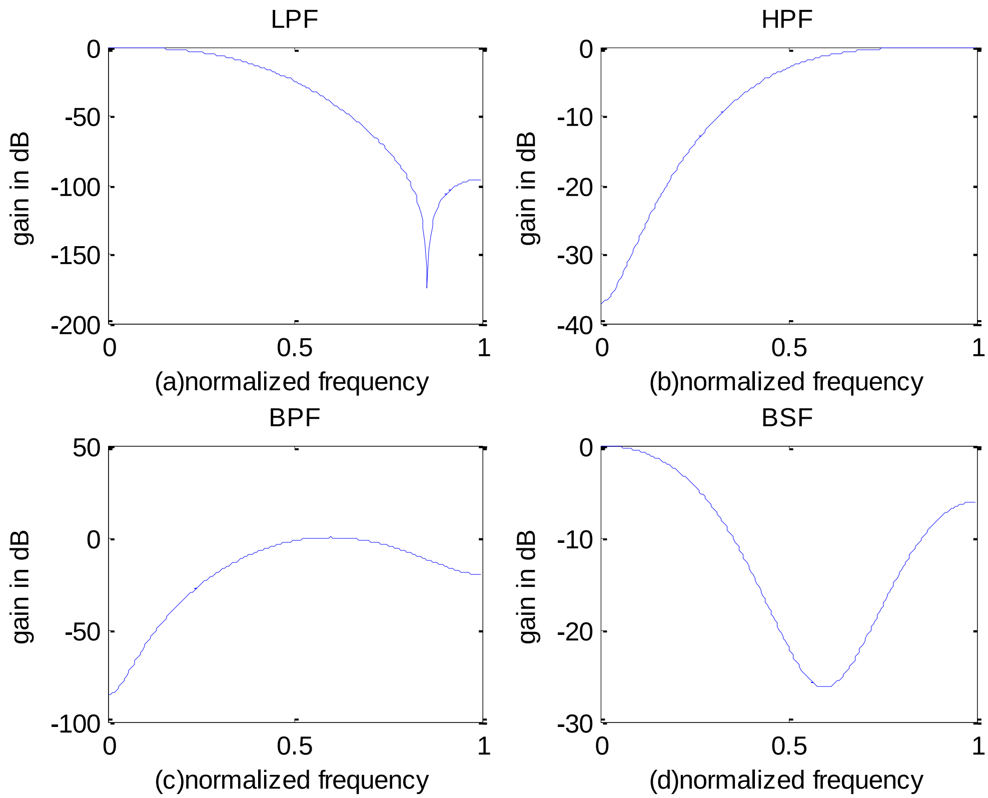
```
clc;
clearall;
closeall;
rp=input('Enter the passband ripple');
rs=input('Enter the stopband ripple');
fp=input('Enter the passband frequency');
fs=input('Enter the stopband frequency');
f=input('Enter the sampling frequency');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end
y=hamming(n1);
disp('The window coefficient are as follows');y
b= fir1(n,wp,y);
disp('unit sample response of fir filter is h(n)=');b
disp(b);b
[h,o]=freqz(b,1,256);
```

```
m=20*log(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(a)normalized frequency');
title('LPF');
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');
title('HPF');
wn=[wpws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(c)normalized frequency');
title('BPF');
wn=[wpws];
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(d)normalized frequency');
title('BSF');
```

INPUT:

```
Enter the passband ripple0.01
Enter the stopband ripple0.02
Enter the passband frequency1000
Enter the stopband frequency2000
Enter the sampling frequency5000
The window coefficient are as follows

y =
    0.0800
    0.2147
    0.5400
    0.8653
    1.0000
    0.8653
    0.5400
    0.2147
    0.0800
unit sample response of fir filter is h(n)=
b =

  Columns 1 through 8

   -0.0061   -0.0136    0.0512    0.2657    0.4057    0.2657    0.0512   -0.0136

  Column 9

   -0.0061

  Columns 1 through 8

   -0.0061   -0.0136    0.0512    0.2657    0.4057    0.2657    0.0512   -0.0136

  Column 9

   -0.0061

b =

  Columns 1 through 8

   -0.0061   -0.0136    0.0512    0.2657    0.4057    0.2657    0.0512   -0.0136

  Column 9

   -0.0061
```

OUTPUT:



RESULT:

Thus the FIR filter using hamming window is designed using MATLAB.

EXP.NO:5(b) DESIGN OF FIR FILTER USING HANNING WINDOW

AIM:

To write a MATLAB program for design a FIR filters(LPF/HPF/BPF/BSF) and demonstrates the filtering operation usinghanning window.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the FIR filter specifications.
- Obtain the filter coefficients using window function.
- Plot the frequency response of the filters.

PROGRAM:

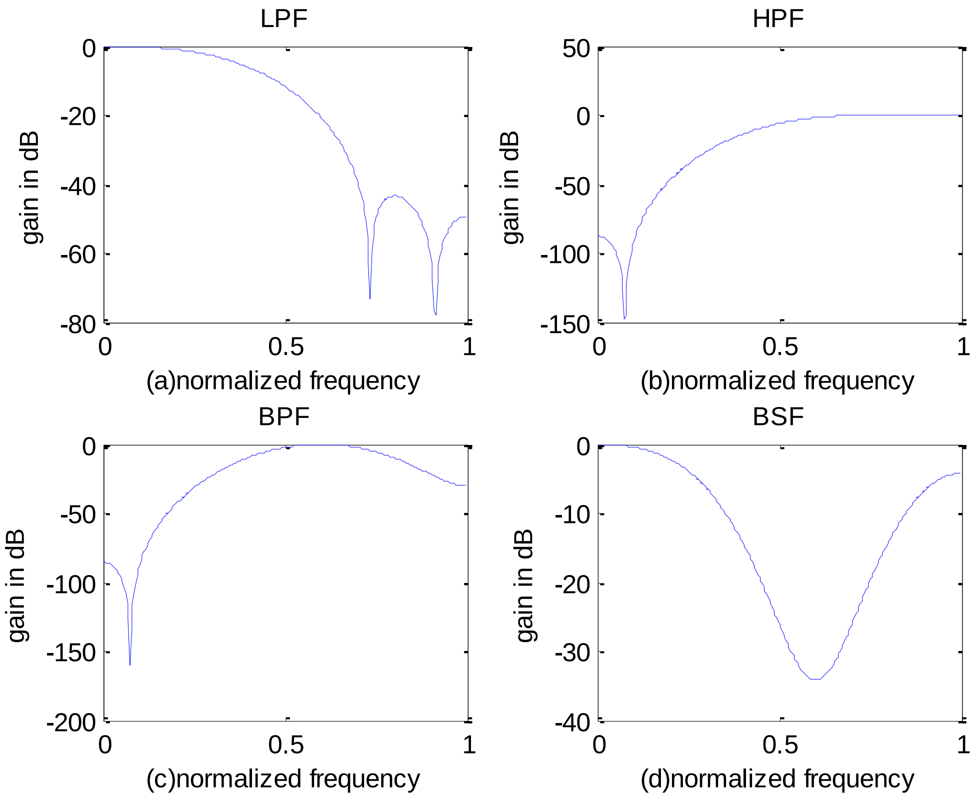
```
clc;
clearall;
closeall;
rp=input('Enter the passband ripple');
rs=input('Enter the stopband ripple');
fp=input('Enter the passband frequency');
fs=input('Enter the stopband frequency');
f=input('Enter the sampling frequency');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end
y=hanning(n1);
disp('the window coefficient are as follows');y
b= fir1(n,wp,y);
disp('unit sample response of fir filter is h(n)=');b
disp(b);b
```

```
[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');
title('LPF');
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(b)normalized frequency');
title('HPF');
wn=[wpws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(c)normalized frequency');
title('BPF');
wn=[wpws];
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(d)normalized frequency');
title('BSF');
```

INPUT:

```
Enter the passband ripple 0.01
Enter the stopband ripple0.02
Enter the passband frequency1000
Enter the stopband frequency2000
Enter the sampling frequency5000
Ehe window coefficient are as follows
y =
    0.0955
    0.3455
    0.6545
    0.9045
    1.0000
    0.9045
    0.6545
    0.3455
    0.0955
unit sample response of fir filter is h(n)=

b =
  Columns 1 through 8

   -0.0071   -0.0213    0.0605    0.2704    0.3950    0.2704    0.0605   -0.0213

  Column 9

   -0.0071

  Columns 1 through 8

   -0.0071   -0.0213    0.0605    0.2704    0.3950    0.2704    0.0605   -0.0213

  Column 9

   -0.0071

b =

  Columns 1 through 8

   -0.0071   -0.0213    0.0605    0.2704    0.3950    0.2704    0.0605   -0.0213

  Column 9

   -0.0071
```

OUTPUT:



RESULT:

Thus the FIR filter using hanning window is designed using MATLAB.

EXP.NO:5(c)　　DESIGN OF FIR FILTER USING KAISER WINDOW

AIM:

To write a MATLAB program for design a FIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operation using Kaiser Window.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the FIR filter specifications.
- Obtain the filter coefficients using window function.
- Plot the frequency response of the filters.

PROGRAM:

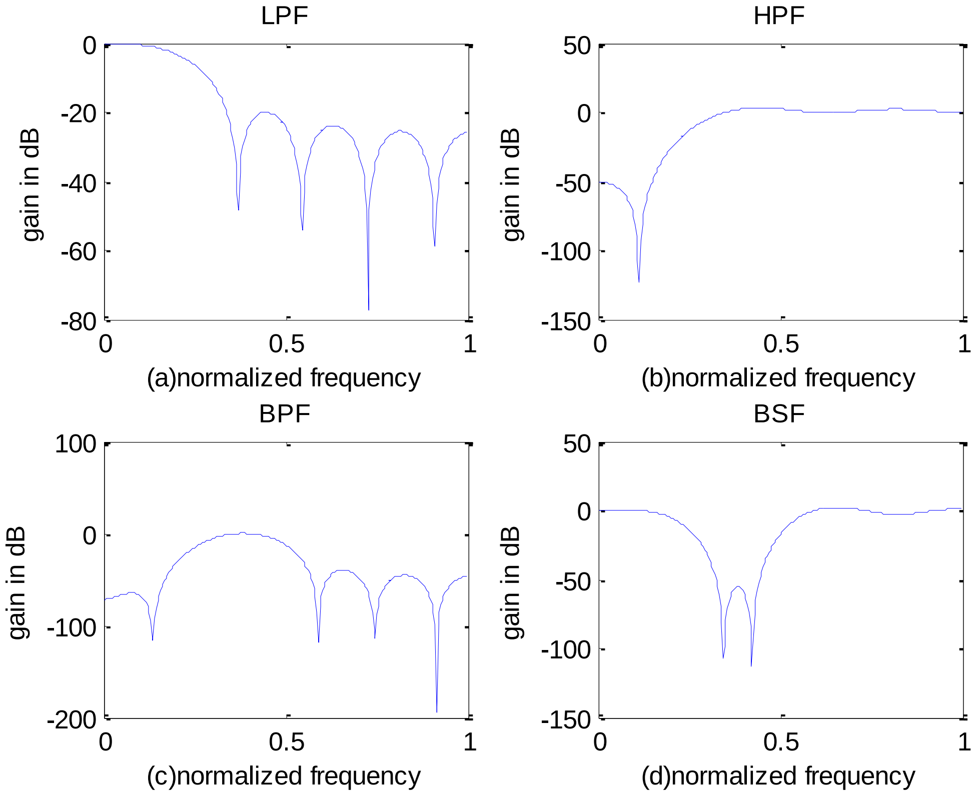
```
clc;
clearall;
closeall;
rp=input('Enter the passband ripple');
rs=input('Enter the stopband ripple');
fp=input('Enter the passband frequency');
fs=input('Enter the stopband frequency');
f=input('Enter the sampling frequency');
beta=input(‘enter the beta value’);
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end
y=kaiser(n1,beta);
disp('The window coefficient are as follows');y
b= fir1(n,wp,y);
disp('unit sample response of fir filter is h(n)=');b
disp(b);b
```

```
[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');
title('LPF');
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(b)normalized frequency');
title('HPF');
wn=[wpws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(c)normalized frequency');
title('BPF');
wn=[wpws];
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('gain in dB');
xlabel('(d)normalized frequency');
title('BSF');
```

INPUT:

```
Enter the passband ripple 0.02
Enter the stopband ripple0.04
Enter the passband frequency1000
Enter the stopband frequency2000
Enter the sampling frequency8000
Enter the beta value 2
The window coefficient are as follows
y =
    0.9403
    0.9616
    0.9783
    0.9903
    0.9976
    1.0000
    0.9976
    0.9903
    0.9783
    0.9616
    0.9403

unit sample response of fir filter is h(n)=
b =  Columns 1 through 8

   -0.0393    0.0000    0.0682    0.1464    0.2086    0.2322    0.2086    0.1464

  Columns 9 through 11

    0.0682    0.0000   -0.0393

  Columns 1 through 8

   -0.0393    0.0000    0.0682    0.1464    0.2086    0.2322    0.2086    0.1464

  Columns 9 through 11

    0.0682    0.0000   -0.0393
b =  Columns 1 through 8

   -0.0393    0.0000    0.0682    0.1464    0.2086    0.2322    0.2086    0.1464

  Columns 9 through 11

    0.0682    0.0000   -0.0393
```

OUTPUT:



RESULT:

Thus the FIR filter using Kaiser Window was designed using MATLAB.

EXP.NO:5(d) DESIGN OF FIR FILTER USING RECTANGULAR WINDOW

AIM:

To write a MATLAB program for design a FIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation using Rectangular Window.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the FIR filter specifications.
- Obtain the filter coefficients using window function.
- Plot the frequency response of the filters.

PROGRAM:

```
clc;
clear all;
close all;
rp=input('enter the pass band ripple');
rs=input('enter the stop band ripple');
fp=input('enter the pass band frequency');
fs=input('enter the stop band 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);
% computation for odd or even
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end
% window function
y=boxcar(n1);
disp('the window coefficient are as follows');
```

```
% low pass filter design
b=fir1(n,wp,y);
disp('unit sample response of fir filter is h(n)=');b
% frequency response
[h,o]=freqz(b,1,256);
% to find gain
m=20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
ylabel('gain in db');
xlabel('(a)normalised frequency');
title('LPF');

% high pass filter design
% fir filter design
b=fir1(n,wp,'high',y);
% frequency response
[h,o]=freqz(b,1,256);
% to find gain
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
ylabel('gain in db');
xlabel('(b)normalised frequency');
title('HPF');

% band pass filter design
% fir filter design
wn=[wpws];
b=fir1(n,wn,y);
% frequency response
[h,o]=freqz(b,1,256);
% to find gain
m=20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
ylabel('gain in db');
xlabel('(c)normalised frequency');
title('BPF');
```

```
% band stop filter design
% fir filter design
wn=[wpws];
b=fir1(n,wn,'stop',y);
% frequency response
[h,o]=freqz(b,1,256);
% to find gain
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
ylabel('gain in db');
xlabel('(d)normalised frequency');
title('BSF');
```

INPUT:

Enter the pass band ripple: 0.07

Enter the stop band ripple: 0.05

Enter the pass band freq: 1300

Enter the stop band freq: 2000

Enter the sampling freq: 7000

The window co-efficient are follows

Y = 1

1

1

1

1

1

1

1

1

Unit sample response of fir filter is h (n)=

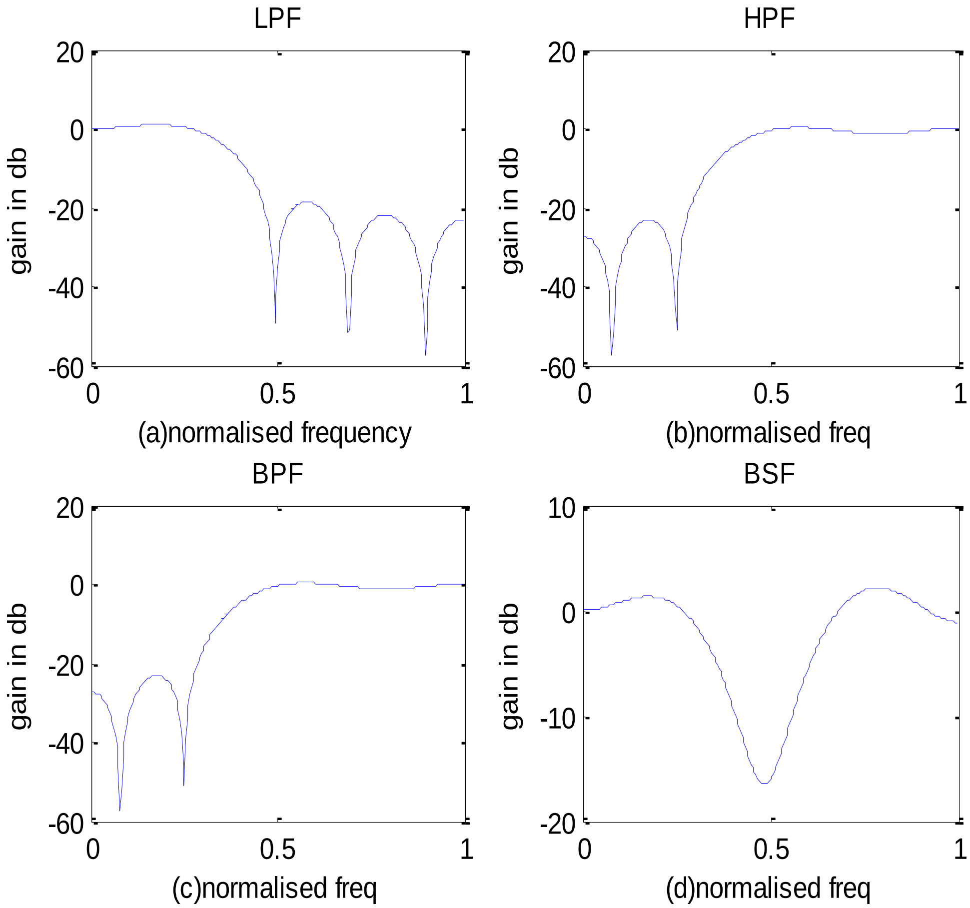
Columns 1 through 7

-0.0834 -0.0391 0.1207 0.3070 0.3896 0.3070 0.1207

Columns 8 through 9

-0.0391 -0.0834

Output Waveform:



RESULT:

Thus the FIR filter using Rectangular Window is designed using MATLAB.

EXP.NO:6 DESIGN OF BUTTERWORTHIIR FILTER

AIM:

To design a Butterworth IIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation using MATLAB program.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the IIR filter specifications.
- Obtain the filter coefficients
- Plot the frequency response of the filters.

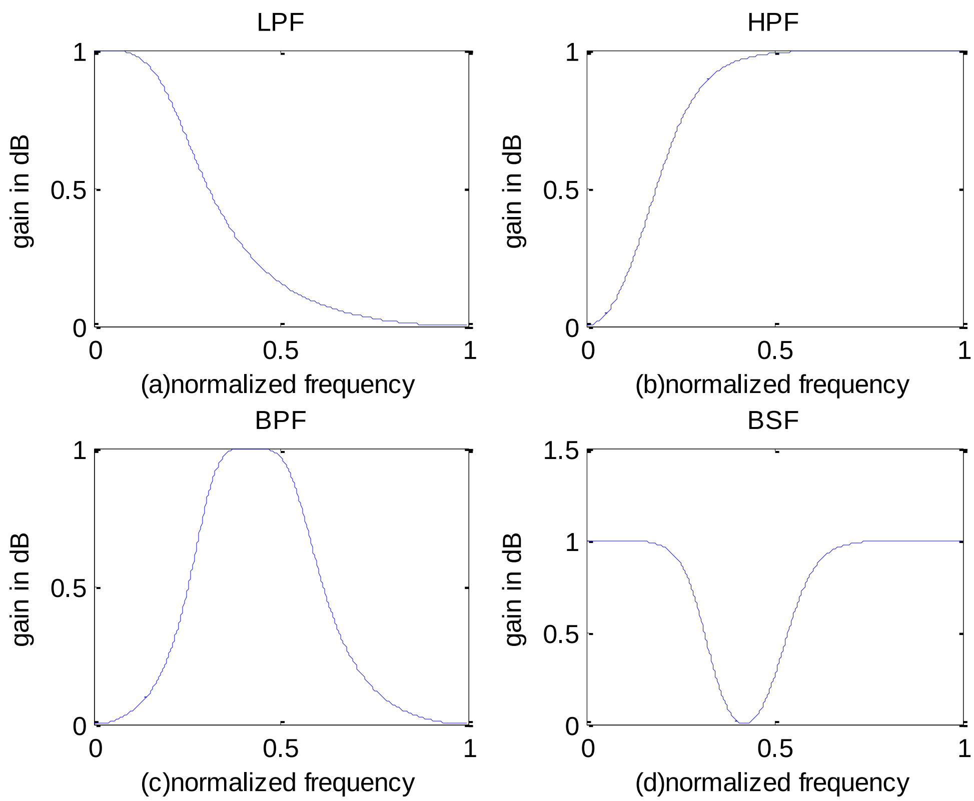
PROGRAM:

```
clc;
clearall;
closeall;
rp=input('Enter the passband ripple');
rs=input('Enter the stopband ripple');
wp=input('Enter the passband frequency');
ws=input('Enter the stopband frequency');
f=input('Enter the sampling frequency');
w1=2*wp/f;
w2=2*ws/f;
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn,'low');
[h,w]=freqz(b,a,512);
subplot(2,2,1);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(a)normalized frequency');
title('LPF');
[b,a]=butter(n,wn,'high');
[h,w]=freqz(b,a,512);
subplot(2,2,2);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(b)normalized frequency');
```

```
title('HPF');
wn1=[w1 w2];
[b,a]=butter(n,wn1);
[h,w]=freqz(b,a,512);
subplot(2,2,3);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(c)normalized frequency');
title('BPF');
wn2=[w1 w2];
[b,a]=butter(n,wn2,'stop');
[h,w]=freqz(b,a,512);
subplot(2,2,4);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(d)normalized frequency');
title('BSF');
```

INPUT
Enter the passbandripple6
Enter the stopbandripple20
Enter the passband frequency1000
Enter the stopband frequency2000
Enter the sampling frequency7000

OUTPUT:



RESULT:

Thus the Butterworth IIR filter was designed using MATLAB.

EXP.NO:6(b) **DESIGN OF CHEBYSHEV-I IIR FILTER**

AIM:

To design a Chebyshev-I IIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation using MATLAB program.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the IIR filter specifications.
- Obtain the filter coefficients.
- Plot the frequency response of the filters.

PROGRAM:

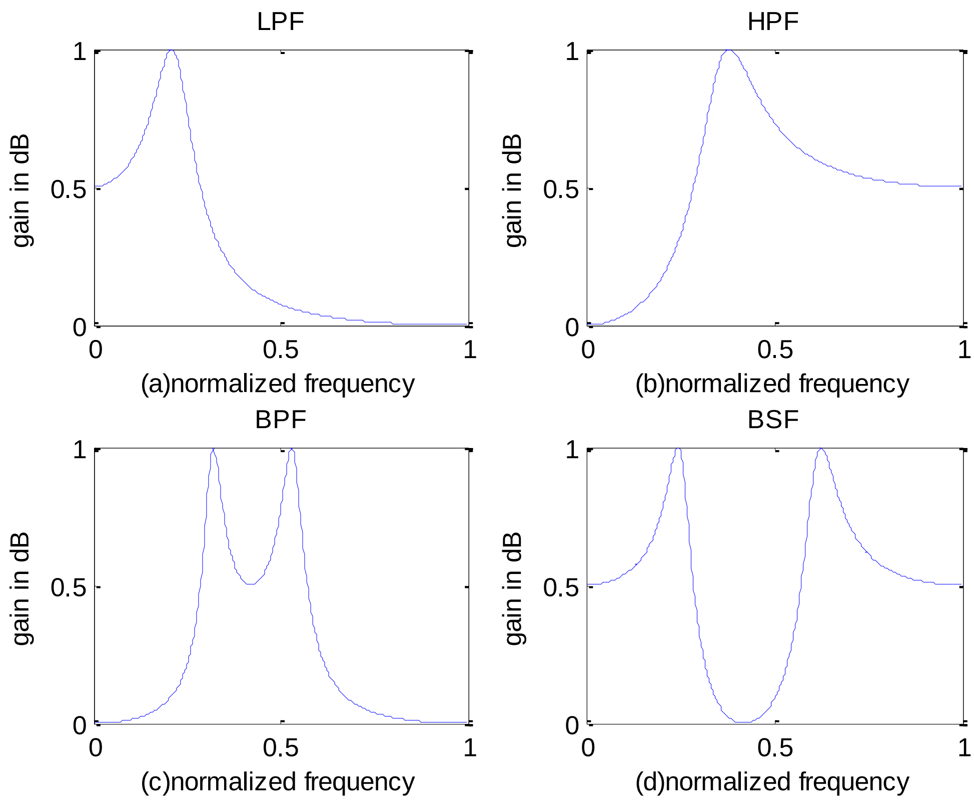
```
clc;
clearall;
closeall;
rp=input('Enter the passband ripple');
rs=input('Enter the stopband ripple');
wp=input('Enter the passband frequency');
ws=input('Enter the stopband frequency');
f=input('Enter the sampling frequency');
w1=2*wp/f;
w2=2*ws/f;
[n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rp,wn,'low');
[h,w]=freqz(b,a,512);
subplot(2,2,1);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(a)normalized frequency');
title('LPF');
[b,a]=cheby1(n,rp,wn,'high');
[h,w]=freqz(b,a,512);
subplot(2,2,2);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(b)normalized frequency');
```

```
title('HPF');
wn1=[w1 w2];
[b,a]=cheby1(n,rp,wn1);
[h,w]=freqz(b,a,512);
subplot(2,2,3);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(c)normalized frequency');
title('BPF');
wn2=[w1 w2];
[b,a]=cheby1(n,rp,wn2,'stop');
[h,w]=freqz(b,a,512);
subplot(2,2,4);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(d)normalized frequency');
title('BSF');
```

INPUT

Enter the passband ripple 6
Enter the stopband ripple 20
Enter the passband frequency 1000
Enter the stopband frequency 2000
Enter the sampling frequency 7000

OUTPUT



RESULT:

Thus the Chebyshev-I IIR filter was designed using MATLAB.

EXP.NO:6(c) DESIGN OF CHEBYSHEV-II IIR FILTER

AIM:

To design a Chebyshev-I IIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation using MATLAB program.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

- Get the IIR filter specifications.
- Obtain the filter coefficients using .
- Plot the frequency response of the filters.

PROGRAM:

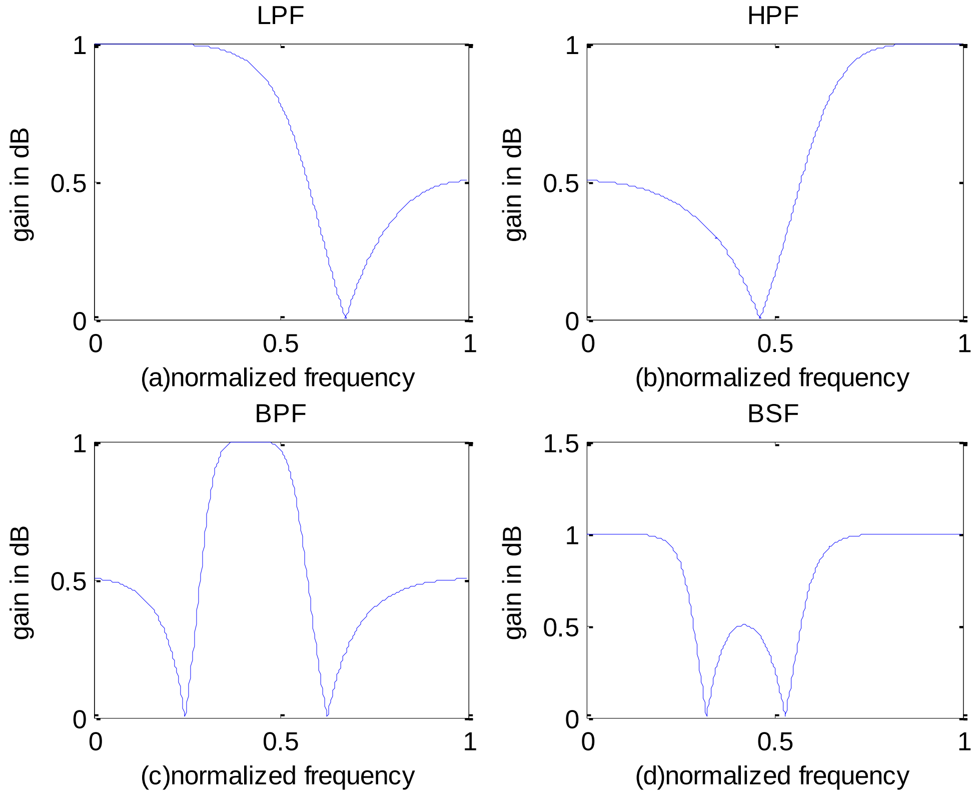
```
clc;
clearall;
closeall;
rp=input('Enter the passband ripple');
rs=input('Enter the stopband ripple');
wp=input('Enter the passband frequency');
ws=input('Enter the stopband frequency');
f=input('Enter the sampling frequency');
w1=2*wp/f;
w2=2*ws/f;
[n,wn]=cheb2ord(w1,w2,rp,rs);
[b,a]=cheby2(n,rp,wn,'low');
[h,w]=freqz(b,a,512);
subplot(2,2,1);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(a)normalized frequency');
title('LPF');
[b,a]=cheby2(n,rp,wn,'high');
[h,w]=freqz(b,a,512);
subplot(2,2,2);
plot(w/pi,abs(h));
ylabel('gain in dB');
```

```
xlabel('(b)normalized frequency');
title('HPF');
wn1=[w1 w2];
[b,a]=cheby2(n,rp,wn1);
[h,w]=freqz(b,a,512);
subplot(2,2,3);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(c)normalized frequency');
title('BPF');
wn2=[w1 w2];
[b,a]=cheby2(n,rp,wn2,'stop');
[h,w]=freqz(b,a,512);
subplot(2,2,4);
plot(w/pi,abs(h));
ylabel('gain in dB');
xlabel('(d)normalized frequency');
title('BSF');
```

INPUT

Enter the passband ripple 6
Enter the stopband ripple 20
Enter the passband frequency 1000
Enter the stopband frequency 2000
Enter the sampling frequency 7000

OUTPUT



RESULT:

Thus the Chebyshev-II IIR filter was designed using MATLAB

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

ARCHITECTURE:

The 54x DSP 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 DSPs families also provide a highly specialized instruction set which is the basis of the operational flexibility and the speed of these DSPs. Separate program and the 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 be performed in a single machine cycle. Also included are the control mechanisms to manage interrupts, repeated operations and function calls.

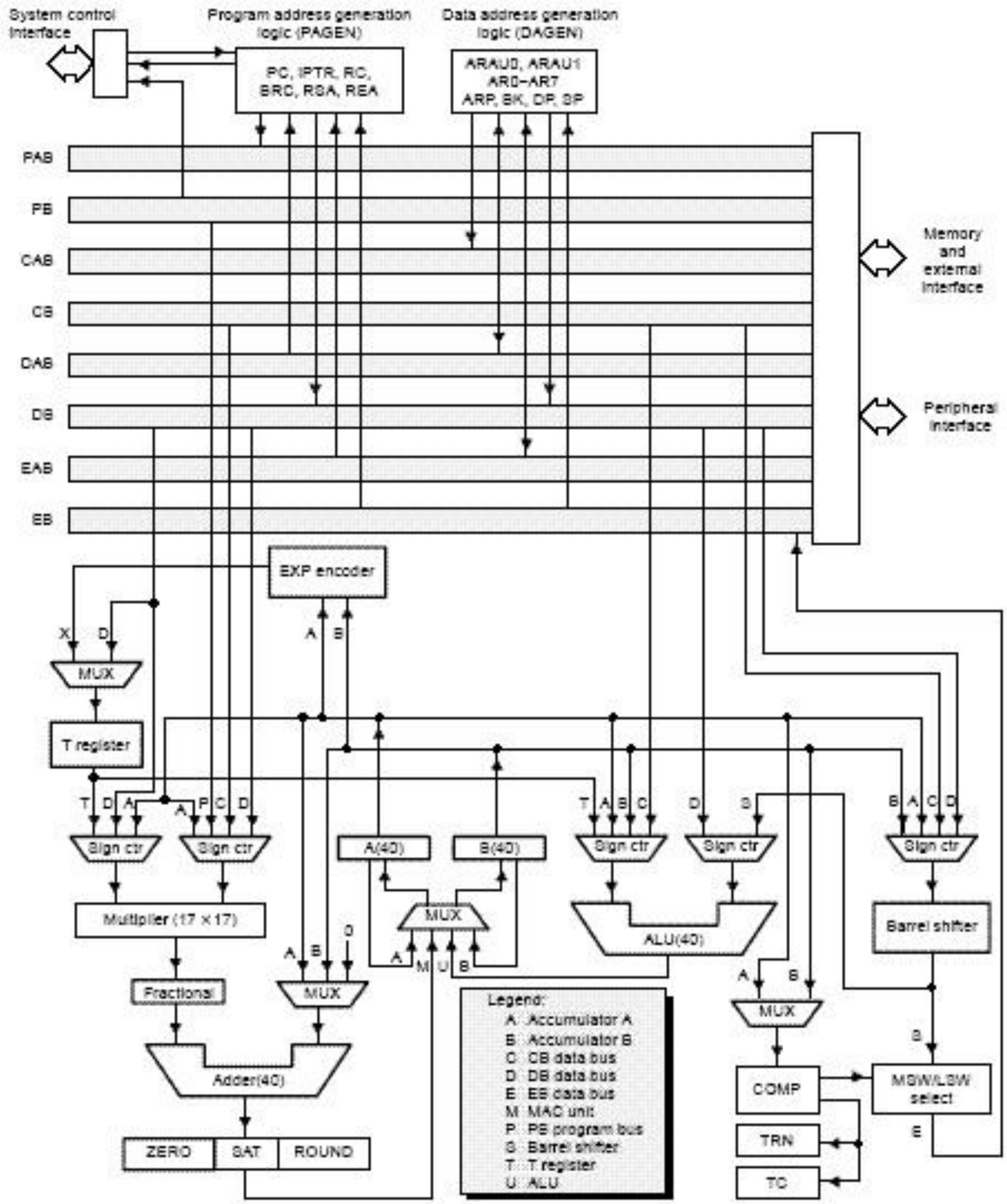
1. CENTRAL PROCESSING UNIT (CPU):

The CPU of the 54x devises contains:

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

2. ARITHMETIC LOGIC UNIT (ALU):

The 54x devises perform 2's complement arithmetic using 40-bit ALU and two 40-bit accumulators (ASSU and ACCB). The ALU also can perform Boolean operations. The ALU can function as a 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 provide a second input to the ALU or the multiplier /adder. The bit in each accumulator is grouped as follows:

- Guard bits (bits 32-39)
- A high – order word (bits 16-31)
- A low order word (bits 0-15)
- 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 designed as a shift-count register. This shifter and the exponent detector normalize the values in the accumulator in a single cycle. The least significant bits (LSBs) of the output are filled with 0s and the most significant bits (MSBs) can neither be zero filled or sign extended, depending on the state of the sign-extended mode bit (SXM) of ST1.addtional shift capabilities enable the processor to perform numerical scaling, bit extraction, extended arithmetic and overflow prevention operation.

5. MULTIPLIER/ADDER:

The multiplier /adder perform 17- bit 2's complement multiplication with the 40-bit accumulation in a single instruction cycle. The multiplier /adder block consists of several elements such as multiplier, adder, signed /unsigned input control, fractional control, a zero detector, a rounder (2's 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, 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 Veterbi type butterfly computation with optimized on – chip hardware.

7. PROGRAM CONTROL IS PROVIDED BY SEVERAL HARDWARE AND SOFTWARE MECHANISMS:

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

The 54x supports both the use of hardware and software interrupts for the program control. The interrupts service routine is vectored through a reloadable interrupt vector table. The interrupts can be globally enabled/disable and can be individually masked through the interrupt, mask register (IMR).

8. STATUS REGISTER (ST0, ST1):

The status register ST0, ST1 contain the status of the various conditions and the modes for the 54x devises. The ST0 contains the flags (OV,C, and TC) produced by the arithmetic operations and bit manipulations in addition to the data pointer (DP) and the auxiliary register pointer (ARP fields). ST1 contains the various modes and the instructions that the processor operates on and executes.

9. AUXILLARY 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. TEMPORAY REGISTERS(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 the 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:

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 the program memory 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) indicated the current status of the interrupts.

16. PROCESSOR-MODE STATUS REGISTER:

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.

EXP.NO: 8(a)PERFORM MAC OPERATION USING VARIOUS ADDRESSING MODES

AIM:

To write an assembly language program for the study of direct, indirect and immediate addressing modes using TMS320C5X.

TOOLS REQUIRED:

DSP HARDWARE:

- TMS320C5X- Starter Kit
- RS 232 Cable
- Power Supply unit

DSP SOFTWARE:

- Assembler
- Loader
- Debugger

ALGORITHM:

- Initialize all memory mapped register.
- Initialize the processor.
- Initialize the analog interface chip.
- Enable receiver interrupt.
- Store the sample length and buffer starting address.
- Initialize analog interface chip register.

PROGRAM:

DIRECT ADDRESSING MODE:

```
        .mmregs                          ; includes memory mapped registers
        .ds 0f00h                        ; set data segment to 0f00ah
        .ps 0a00h                        ; origin of the program 0a00h
rint    b getdata                        ; receive interrupt
xint     b xint                          ; transmit interrupt
         .ps 0a00h                       ; program entry point
         .entry                          ; initialize the program counter
         .include "c:/c5xinz.asm"
```

```
            lap #20h                  ; the data page number 20h(32) is loaded into
                                      accumulator
            lacc 10h                  ; content of 20h(32) page 10h location
            lac 5h,2
            lar Aro,#15h              ; AR0 loaded with content of dma 1115h
            sacl 15h
            sacl20h,3                 ; accumulator low byte is left shifted by 3 bits and
                                        stored in into dma 1120h

getdata     samm ART                  ;accumulator low byte stored into ART in page0 DP
                                       remains unaffected
            ldp #12h                  ;the data page number 12h is loaded in DP
            add 25h
            add 7h,2
            sub 10h                   ;the content of dma 0910h is subtracted from the
                                       content of accumulator
            sub 12h,2
            splk  #10h,TREGO          ;constant 10h is stored into TREGO
            mpy 15h
REG1        .set 010ch
REG2        .set 020ah
REG4        .set 0415h
REG5        .set 0505h
            .include "c:/ac0 1inz.asm"
            .end                      ; program end
```

INDIRECT ADDRESSING MODE

```
            .mmregs              ; includes memory mapped registers
            .ds 0f00h            ; set data segment to 0f00ah
            .ps 0a00h            ; origin of the program 0a00h
    rint     b getdata           ; receive interrupt
    xint     b xint              ; transmit interrupt
             .ps 0a00h           ; program entry point
             .entry              ; initialize the program counter
             .include "c:/c5xinz.asm"
            lar ARO,#1000h

            lacc *               ;content of dma pointed by ARO is loaded in
                          accumulator
```

```
            lacc *,4,AR1                    ;content of dma 1000h left shifted by 4 bits and loaded
                                       into accumulator. ARP points to auxiliary register 1
            lar AR1,#,1010h
            sacl *                          ;accumulator low byte is stored into the dma pointed by
                                       AR1
            sacl*+,2,AR0                    ;accumulator low byte is shifted by 2 bits and stored
                                       into the dma pointed in AR1
            lacc*-2,AR1
getdata     lacc*0+
            lacc *BR0+                      ;accumulator loaded with content of dma pointed by
                                       AR1 and the content of INDEX register added to AR1
                                  with the reverse carry propagation
            add #+.0.ar0
            sub *.-2                        ;content of dma pointed by AR1 is added from the
                                       content of accumulator. The result is stored into the
                                  accumulator AR0is decremented by 1.
            splk #10h,TREGO                 ;constant 10h is stored into TREGO
            mpy *                           ;content of 0915h is multiplied with the content of
                                       TREGO and the result is stored into PREG
REG1        .set 010ch
REG2        .set 020ah
 REG4       .set 0415h
REG5        .set 0505h
            .include "c:/ac0 1inz.asm"
            .end                            ; program end
```

IMMEDIATE ADDRESSING MODE

```
            .mmregs                         ; includes memory mapped registers
            .ds 0f00h                       ; set data segment to 0f00af
            .ps 0a00h                       ; origin of the program 0a00h
      rint  b getdata                       ;receive interrupt
      xint  b xint                          ;transmit interrupt
            .ps 0a00h                       ; program entry point
            .entry                          ; initialize the program counter
            .include "c:/c5xinz.asm"
            lacc#1000h                      ;value 1000h is loaded into accumulator
            lacc#1111h,3                    ;constant 1111h is left shifted by 3 bits and loaded in
                                            accumulator. The accumulator after execution is 8888h
   getdata  lar AR0,#1000h                  ;AR0 is loaded the content of 1000h
            lar AR1,#1100h                  ;1100f is loaded in AR1
            add#00ffh                       ;ffhis added to the content of accumulator
            sp1k #10h,TREGO
```

```
      mpy#0010h                    ;0010h is multiplied with the content of TREGO
      sub #0022h
      sub #0011h,3                 ;0011h is left shifted by 3 bits subtracted from the
                                   content of accumulator
REG1    .set 010ch
REG2    .set 020ah
REG4    .set 0415h
REG5    .set 0505h
        .include “c:/ac0 1inz.asm”
        .end                              ; end of program
```

RESULT:

Thus the assembly language program for the study of direct, indirect and immediate addressing modes was written and executed successfullyusing TMS320C5X.

EXP.NO:9 GENERATION OF VARIOUS SIGNALS AND RANDOM NOISE

AIM:

To write an assembly language program for the generation of sine wave using TMS320C5X.

TOOLS REQUIRED:

DSP HARDWARE:

- TMS320C5X- Starter Kit
- RS 232 Cable
- Power Supply unit

DSP SOFTWARE:

- Assembler
- Loader
- Debugger

ALGORITHM:

- Initialize all memory mapped register.
- Initialize the processor.
- Initialize the analog interface chip.
- Enable receiver interrupt.
- Store the sample length and buffer starting address.
- Initialize analog interface chip register.

PROGRAM:

```
        .mmregs                 ;initialise all registers.

        .ds     1000h
        .include "sinetbl.dat"  ;load 800 point wavetable
                                ;f1= fs/D = 8000/800 = 10hz.
        .ds 0f00h
temp    .word   0
mod     .word   100         ;Required freq. = mod * f1 = 100*10 = 1000hz.
scale   .q15    0.5
;---------------------------------
;Interrupt vectors
;---------------------------------
        .ps     080ah
```

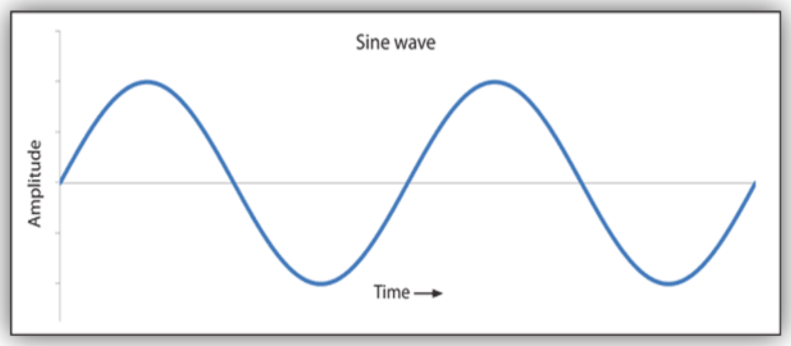
```
rint        b    getdata        ;receive interrupt
xint        b    xint                   ;transmit interrupt

            .ps    0a00h                ;program entry point
            .entry

;--------------------------------
;Processor initialization
;--------------------------------

            .include "c5xinz.asm"

            splk   #012h,IMR            ;enable RINT & INT2.
            call   ac01_init            ;call to initialize serial port & AC01.
            clrc   INTM                 ;enable all interrupts.

      wait:  nop                        ;wait for interrupt.
            b    wait

;--------------------------------
;Receive interrupt handler
;--------------------------------
getdata     splk   #1,gotflag            ;set a flag to indicate data available.
            lamm   DRR
            ldp    #mod                  ;set data page pointer.
            lacc   mod                  ;load modifier
            samm   INDX                 ;store modifier in INDX register.
      callwavgen                ;calculate current sample output from
                         wavetable.
      and    #0FFFCh,0                  ;only 14 MSBs are used in ADC &DAC,So
                                        ; mask unused two LSBs.
            samm   DXR                  ;send digital data to DAC to produce analog
                                o/p.
            clrc   INTM                  ;enable interrupt.
            rete             ;return back to main from interrupt routine.

offset      .set 1320h                  ;table length = 800 + table start address.

            .include "wavgen.asm"       ;includewavgen module.
```

```
;---------------------------------
;AC01 register initialization.
;---------------------------------
            REG1    .set 0124h
            REG2    .set 0212h
            REG4    .set 0417h
            REG5    .set 0505h
;---------------------------------
;Serial port and AC01 initialization
;---------------------------------
                    .include "ac01inz.asm"

                    .END                        ;end of program.
```

TABULATION:

Amplitude (v)	Time (ms)

OUTPUT:



RESULT:

Thus the assembly language program for the generation of sine waveform was written and executed successfullyusing TMS320C5X.

EXP.NO:10(a) IMPLEMENTATION OF FIR LOW PASS FILTER

AIM:

To write an assemble language program for the implementation of FIR low pass filter using TMS320C5X.

TOOLS REQUIRED:

DSP HARDWARE:

- TMS320C5X- Starter Kit
- RS 232 Cable
- Power Supply unit

DSP SOFTWARE:

- Assembler
- Loader
- Debugger

ALGORITHM:

- Initialize all memory mapped register.
- Initialize the processor.
- Initialize the analog interface chip.
- Enable receiver interrupt.
- Store the sample length and buffer starting address.
- Initialize analog interface chip register.

PROGRAM:

```
        .mmregs          ;initialize all memory mapped registers

        .ds 0f00h                ;set data segment to 0f00h.
;-------------------------------
;201 coefficients table.
;-------------------------------
        .include   "firlpf.cof"

rbuf    .word 0                  ;Temp buffer allocation.

;---------------------------------
;Interrupt vectors
;---------------------------------
```

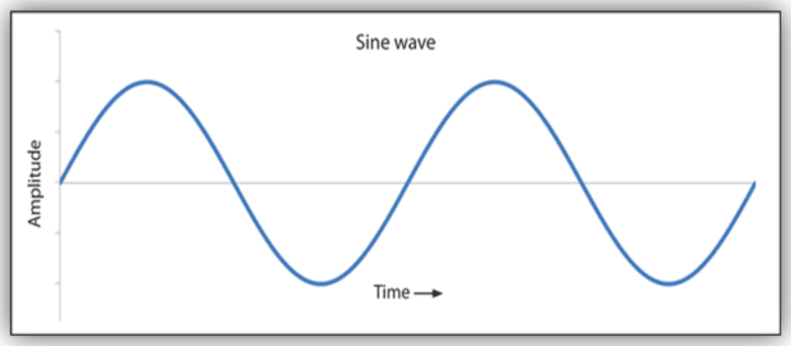
```
            .ps     080ah

rint        b       getdata         ;receive interrupt
xint        b       xint            ;transmit interrupt


            .ps     0a00h                   ;program entry point
            .entry

;--------------------------------
;Processor initialization
;--------------------------------

            .include "c5xinz.asm"

;--------------------------------
;Internal memory initialization
;--------------------------------

            mar     *,AR7                   ;ARP = AR7
            lacl    #0                      ;ACC = 0
            lar     AR7,#300h               ;clear 300 to 3ffh(data array).
            rpt     #255
            sacl    *+

            mar     *,AR0
            lar     AR0,#0200h              ;copy 201 co-efficients
            rpt     #200                    ;to address 200h-2C8h(B0).
            bldd    #FIR_COEFFS,*+,AR0

            splk    #012h,IMR               ;enable RINT & INT2.
            call    ac01_init               ;call to initialize serial port & AC01.
            clrc    INTM                    ;enable all interrupts.

wait:       nop                             ;wait for interrupt.
            b       wait

;--------------------------------
;Receive interrupt handler
;--------------------------------
```

```
getdata     splk    #1,gotflag          ;set a flag to indicate data available.
            lamm    DRR                 ;read ADC data from DRR register.
            and     #0fffch             ;mask LSB two bits.
            ldp     #rbuf
            saclrbuf
            lacc    rbuf,13             ;load accu-high with ADC data.
            ldp     #06h                ;set page pointer = 6.
            sach    0                   ;store ADC data(address=300h).
            mar     *,AR1
            lar     AR1,#3C8h           ;load AR1 with data buffer end addr.
                                        ;(data memory).
;---------------------------------
;FILTERING.
;---------------------------------
            setc    CNF                 ;convert B0 to program memory.
            mpy     #0                  ;clear product reg.
            lacl    #0                  ;clear accumulator.
            rpt     #200                ;repeat MACD insru. 201 times.
            macd    #0ff00h,*-          ;convolution process.
            apac                        ;get result in accumulator.
            sach    0           ;store result in data buffer.
            lacc    0,4
            ldp     0
            samm    DXR                 ;send digital ADC data to DAC.
            clrc    CNF                 ;convert B0 to data memory.
            rete        ;return from interrupt.
;---------------------------------
;AC01 register initialization.
;---------------------------------
REG1    .set 0124h
REG2    .set 0212h
REG4    .set 0415h
REG5    .set 0505h
;---------------------------------
;Serial port and AC01 initialization
;---------------------------------
            .include "ac01inz.asm"

            .END                        ;end of program.
```

TABULATION:

Amplitude (v)	Time (ms)

OUTPUT



RESULT:

Thus the assembly language program for the implementation of FIR low pass filter was written and executed successfullyusing TMS320C5X.

EXP.NO:10 (b) **IMPLEMENTATION OF FIR HIGH PASS FILTER**

AIM:

To write an assembly language program for the implementation of FIR high pass filter using TMS320C5X.

TOOLS REQUIRED:

DSP HARDWARE:

- TMS320C5X- Starter Kit
- RS 232 Cable
- Power Supply unit

DSP SOFTWARE:

- Assembler
- Loader
- Debugger

ALGORITHM:

- Initialize all memory mapped register.
- Initialize the processor.
- Initialize the analog interface chip.
- Enable receiver interrupt.
- Store the sample length and buffer starting address.
- Initialize analog interface chip register.

PROGRAM:

```
            .mmregs                 ;initialize all memory mapped registers

            .ds 0f00h               ;set data segment to 0f00h.
;-------------------------------
;201 coefficients table.
;-------------------------------
            .include   "firhpf.cof"

rbuf        .word 0                 ;Temp buffer allocation.
;---------------------------------
;Interrupt vectors
;---------------------------------
```

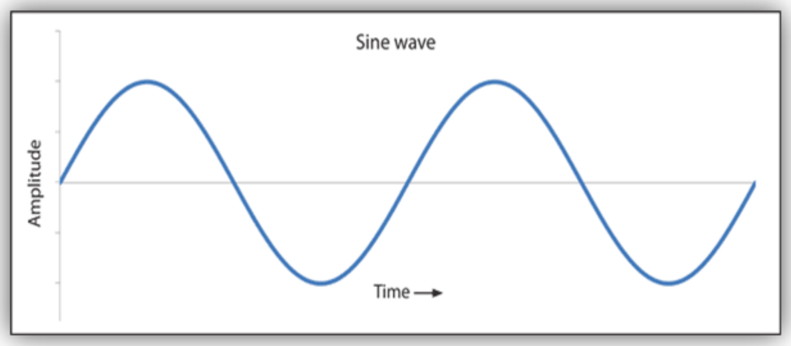
```
            .ps    080ah

rint        b      getdata          ;receive interrupt
xint        b      xint             ;transmit interrupt

            .ps    0a00h                  ;program entry point
            .entry
;--------------------------------
;Processor initialization
;--------------------------------

            .include "c:\fepl\c5xinz.asm"
;--------------------------------
;Internal memory initialization
;--------------------------------

            mar     *,AR7
            lacl    #0
            lar     AR7,#300h              ;clear 300 to 3ffh(data array).
            rpt     #255
            sacl    *+

            mar     *,AR0
            lar     AR0,#0200h             ;copy 201 co-efficients
            rpt     #200                   ;to address 200h-2C8h(B0).
            blkd    COEFFS,*+,AR0

            splk    #012h,IMR
            call    ac01_init              ;call to initialize serial port & AC01.
            clrc    INTM

wait:       nop                            ;wait for interrupt.
            b       wait
;--------------------------------
;Receive interrupt handler
;--------------------------------
Getdata     splk    #1,gotflag             ;set a flag to indicate data available.
            lamm    DRR
            and     #0fffch
            ldp     #rbuf
```

```
          saclrbuf
          lacc    rbuf,13
          ldp     #06h
          sach    0
          mar     *,AR1
          lar     AR1,#3C8h               ;load AR1 with data buffer end addr.


          setc    CNF
          mpy     #0
          lacl    #0
          rpt     #200
          macd    #0ff00h,*-              ;convolution process.
          apac
          sach    0
          lacc    0,4
          ldp     0
          samm    DXR
          clrc    CNF
          rete
;---------------------------------
;AC01 register initialization.
;---------------------------------
     REG1    .set 0124h
     REG2    .set 0212h
     REG4    .set 0415h
     REG5    .set 0505h
;---------------------------------
;Serial port and AC01 initialization
;---------------------------------
          .include "c:\fepl\ac01inz.asm"

          .end                            ;end of program.
```

TABULATION:

Amplitude (v)	Time (ms)

OUTPUT:



RESULT:

Thus the assembly language program for the implementation of FIR high low pass filter was written and executed successfully using TMS320C5X.

EXP.NO:11 **IMPLEMENTATION OF IIR FILTER**

AIM:

To write an assembly language program for the implementation of IIR filter using TMS320C5X.

TOOLS REQUIRED:

DSP HARDWARE:

- TMS320C5X- Starter Kit
- RS 232 Cable
- Power Supply unit

DSP SOFTWARE:

- Assembler
- Loader
- Debugger

ALGORITHM:

- Initialize all memory mapped register.
- Initialize the processor.
- Initialize the analog interface chip.
- Enable receiver interrupt.
- Store the sample length and buffer starting address.
- Initialize analog interface chip register.

PROGRAM:

```
            .mmregs             ;initialize all memory mapped registers

            .ds 0f00h           ;set data segment to 0f00h.

DN          .word   0           ;Input data delay line
DNM1        .word   0
DNM2        .word   0

YN          .word   0           ;output buffer
XN          .word   0           ;input buffer
```

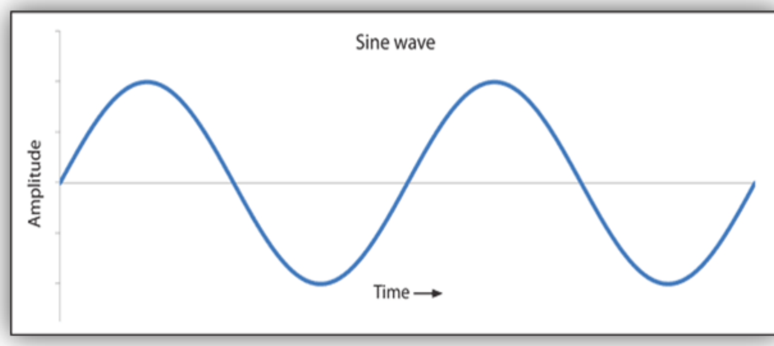
```
                .include "bilinear.cof"       ;bilinear IIR filter coefficients
                .include "invarian.cof"       ;invariance IIR filter coefficients

                .ps     080ah

rint            b       getdata         ;receive interrupt
xint            b       xint            ;transmit interrupt
                .ps     0a00h                 ;program entry point
                .entry
                .include "c5xinz.asm"

                splk   #012h,IMR
                call   ac01_init              ;call to initialize serial port & AC01.
                lacl   #0
                ldp    #DN
                sacl   DN                     ;clear input data delay line.
                sacl   DNM1
                sacl   DNM2
                clrc   INTM

wait:           nop                           ;wait for interrupt.
                b      wait

getdata         splk   #1,gotflag             ;set a flag to indicate data available.
                lamm   DRR                    ;read input from AIC
                and    #0FFFCh,0              ;mask unwanted bits
                ldp    #XN
                sacl   XN                     ;store input sample

                lacc   XN,15
                lt     DNM1
                mpy    A1
                lta    DNM2
                mpy    A2
                apac                          ;DN = x(n) + d(n-1)*a1 + d(n-2)*a2
                sach   DN,0                   ;store pole result
                lacl   #0
                mpy    B2
                ltd    DNM1
                mpy    B1
```

```
            ltd    DN
            mpy    B0
            apac
            sach   YN,3                 ;Y(n) = d(n)*b0 + d(n-1)*b1 + d(n-2)*b2
            lacc   YN,0                 ;store y(n) result
            and    #0FFFCh,0
            samm   DXR                  ;output the filter response y(n) to AIC.
            rete

REG1        .set 010ch
REG2        .set 0212h
REG4        .set 0415h
REG5        .set 0505h
            .include "ac01inz.asm"

            .END                        ;end of program.
```

TABULATION:

Amplitude (v)	Time (ms)

OUTPUT:



RESULT:

Thus the assembly language program for the implementation of IIR filter was written and executedsuccessfully using TMS320C5X.

EX.No: 12 IMPLEMENT AN UP-SAMPLING AND DOWN-SAMPLING OPERATION IN DSP PROCESSOR

AIM:

To write an assembly language program for sampling the given input signal using TMS320C5X.

TOOLS REQUIRED:

DSP HARDWARE:

- TMS320C5X- Starter Kit
- RS 232 Cable
- Power Supply unit

DSP SOFTWARE:

- Assembler
- Loader
- Debugger

ALGORITHM:

- Initialize all memory mapped register.
- Initialize the processor.
- Initialize the analog interface chip.
- Enable receiver interrupt.
- Store the sample length and buffer starting address.
- Initialize analog interface chip register.

PROGRAMM

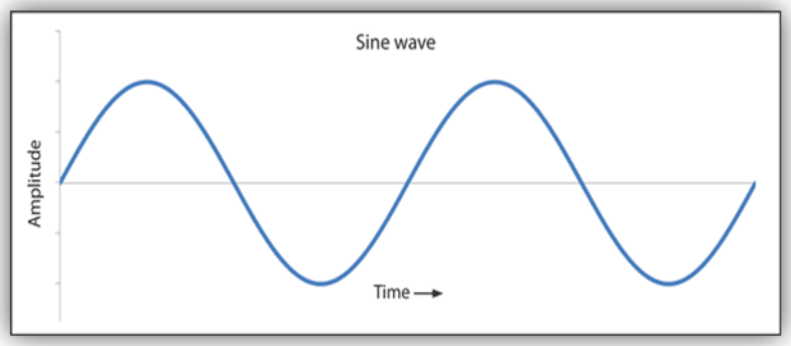
```
                .mmregs                 ;initialize all memory mapped registers

                .ps     080ah

rint            b       getdata          ;receive interrupt

xint            b       xint            ;transmit interrupt

                .ps     0a00h                   ;program entry point

                .entry

                .include "c5xinz.asm"

                lar     AR2,#1000h

                splk    #012h,IMR               ;enable RINT & INT2.
```

```
                call    ac01_init              ;call to initialize serial port & AC01.
                clrc    INTM                   ;enable all interrupts.
                lar     AR1,#2048
                lar     AR2,#1000h
wait:           idle                           ;wait for interrupt.
                mar     *,AR1
                banz    wait,*-,AR2
                nop
                nop
                setc    INTM
                splk    #02h,IMR
                clrc    INTM
hlt:            b       hlt
getdata splk    #1,gotflag             ;set a flag to indicate data available.
                lamm    DRR                    ;read ADC data from DRR register.
                and     #0fffch                ;mask LSB two bits.
                samm    DXR                    ;send digital ADC data to DAC.
                mar     *,AR2
                sacl    *+
                rete            ;return from interrupt.

REG1            .set 010ch
REG2            .set 020ah
REG4            .set 0415h
REG5            .set 0505h
                .include "ac01inz.asm"
                .END                           ;end of program.
```

TABULATION:

Amplitude (v)	Time (ms)

OUTPUT:



RESULT:

Thus the assembly language program for the sampling operation was written and executed successfully using TMS320C5X.

EXP.NO: **LINEAR CONVOLUTION**

AIM:

To write an assembly language program for linear convolution using TMS320C5X.

TOOLS REQUIRED:

DSP HARDWARE:

- TMS320C5X- Starter Kit
- RS 232 Cable
- Power Supply unit

DSP SOFTWARE:

- Assembler
- Loader
- Debugger

ALGORITHM:

- Include memory mapped register and set pointer program memory and data memory.
- Append 0's buffer and after impulse response no of zero in length of input sequence.
- Zero accumulator and product register.
- Multiply accumulator program memory with data memory.
- Each time program memory is incremented by one and data memory decremented by one.
- Repeat step 4 for n+1 timer where n is length of largest sequence.
- Decrement count value ARZ if ARZ≠0 go to step3.

PROGRAM:

```
        .mmregs                          ;initialize all registers
        .ps  0a00h
        .word 1h,2h,3h,2h,1h             ;x(n) stored form pma 0a00h
        .ds  1000h
        .word0h,0h,0h,0h
        .word 3h,4h,5h,0h
        .word oh,0h,0h,0h
        .entry
        LAR AR0.#1004H                   ;actual data starts only at 1004h
        LAR AR1,#1020H                   ;starting address for result
        LAR AR2,#07H                     ;length for output sequence
Loop    ZAP                              ;zero accumulator and product reg
        MAR*,AR0
        RPT#5H                           ; execute instructions followed by RPT
                                   instructions 5 times
        MAC 0a00h,*-
        MAR *.AR1
```

```
SACL*+.0.AR0          ; one result is stored
ADRK#7H
MAR*,AR2
MAR*
BANZ loop
.end                  ; end of program
```

INPUT:

1000	01
1001	02
1002	03
1003	02
1004	01
1005	03
1006	04
1007	05
1008	00

OUTPUT:

1020	03
1021	10
1022	22
1023	28
1024	26
1025	10
1026	05
1027	00

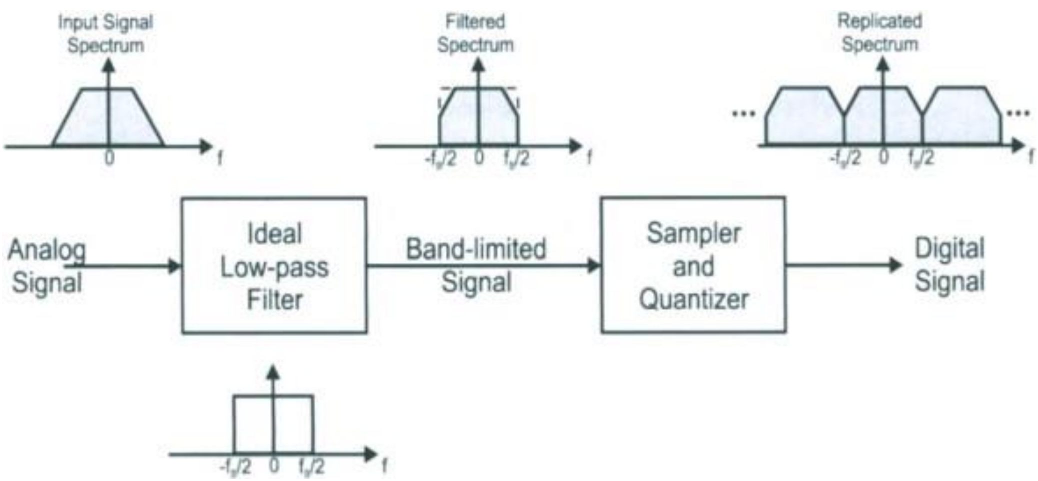
RESULT:

Thus the assembly language program for linear convolution was written and executed successfully using TMS320C5X.

EXP.NO: 13 STUDY OF ANTI- ALIASING FILTER

Antialiasing filters:

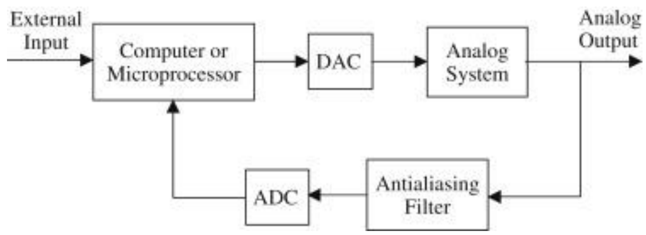
Anti-aliasing filters are always analog filters as they process the signal before it is sampled. In most cases, they are also low-pass filters unless band-pass sampling techniques are used.The sampling process incorporating an ideal low-pass filter as the anti-alias filter is shown below. The ideal filter has a flat passband and the cut-off is very sharp. Since the cut-off frequency of this filter is half of that of the sampling frequency, the resulting replicated spectrum of the sampled signal do not overlap each other. Thus no aliasing occurs.



Analog to Digital conversion process using Anti – aliasing filter

If the sampling frequency does not satisfy the sampling theorem (i.e., the sampled signal has frequency components greater than half the sampling frequency), then the sampling process creates new frequency components .This phenomenon is called aliasing and must obviously be avoided in a digital control system. Hence, the continuous signal to be sampled must not include significant frequency components greater than the Nyquist frequency ωs/2.

For this purpose, it is recommended to low-pass filter the continuous signal before sampling, especially in the presence of high-frequency noise. The analog low-pass filter used for this purpose is known as the antialiasing filter. The antialiasing filter is typically a simple first-order RC filter, but some applications require a higher-order filter such as a Butterworth or a Bessel filter. The overall control scheme is shown below.



Control scheme with an antialiasing filter.

Because a low-pass filter can slow down the system by attenuating high-frequency dynamics, the cutoff frequency of the low-pass filter must be higher than the bandwidth of the closed-loop system so as not to degrade the transient response. A rule of thumb is to choose the filter bandwidth equal to a constant time the bandwidth of the closed-loop system. The value of the constant varies depending on economic and practical considerations. For a conservative but more expensive design, the cutoff frequency of the low-pass filter can be chosen as 10 times the bandwidth of the closed-loop system to minimize its effect on the control system dynamics, and then the sampling frequency can be chosen 10 times higher than the filter cutoff frequency so there is a sufficient attenuation above the Nyquist frequency. Thus, the sampling frequency is 100 times the bandwidth of the closed-loop system. To reduce the sampling frequency, and the associated hardware costs, it is possible to reduce the antialiasing filter cutoff frequency. In the extreme case, we select the cutoff frequency slightly higher than the closed-loop bandwidth. For a low-pass filter with a high roll-off (i.e., a high-order filter), the sampling frequency is chosen as five times the closed-loop bandwidth. In summary, the sampling period T can be chosen in general as $5\omega b \leq 2\pi T \leq 100\omega b$ where ωb is the bandwidth of the closed-loop system.

EX.No: 14 **CONVERSION OF ANALOG TO DIGITAL FILTERS**

AIM:

To write a program for the conversion of analog to digital filters using MATLAB.

SOFTWARE REQUIRED:

MATLAB R2014a

ALGORITHM:

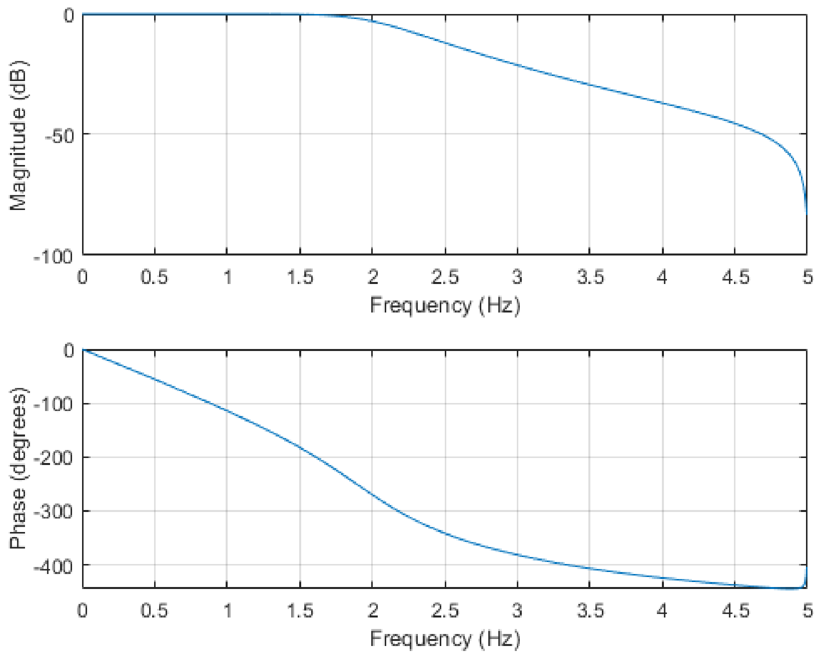
- Get the required analog input specifications.
- Convert the analog specifications to digital specifications .
- Plot the digital filter specifications.

PROGRAM:

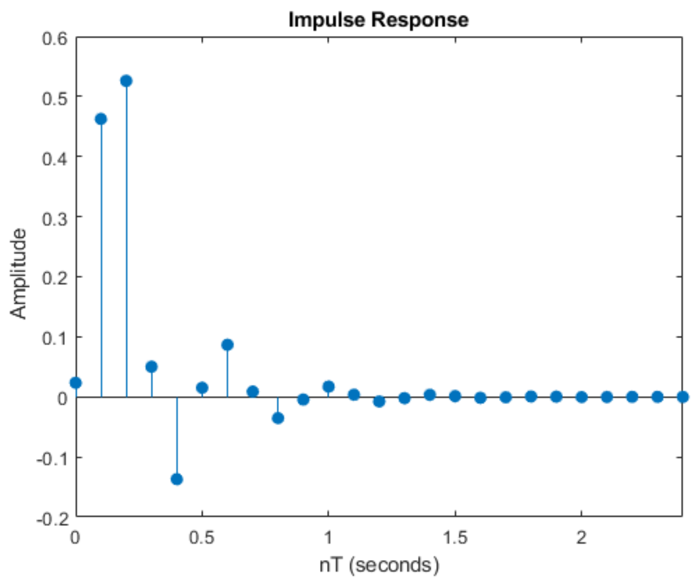
```
alpha = 0.2;
fs = 200; % Sample Frequency [Hz]
% Laplace Domain
B = 1;
A = [1, alpha];
w = 0:0.2:(fs / 2);
h = freqs(B, A, w);
figure;
plot(w, abs(h .* conj(h)));
% Digital Filter
[b, a] = bilinear(B, A, fs);
figure;
freqz(b, a, 1000);
% Frequency Response of the filter
f = 2;
fs = 10;
[b,a] = butter(6,2*pi*f,'s');
[bz,az] = impinvar(b,a,fs);
freqz(bz,az,1024,fs)
% Impulse Response of the Digital filter
fs = 10;
[b,a] = ellip(3,1,60,2*pi*2.5,'s');
[bz,az] = impinvar(b,a,fs);
impz(bz,az,[],fs)
```

OUTPUT:

Frequency Response of the filter



Impulse Response of the Digital filter



RESULT:

Thus the analog filter was converted to digital filter using MATLAB.