DRONACHARYA
GROUP OF INSTITUTIONS

DIGITAL SIGNAL PROCESSING LAB

LABORATORY MANUAL

B.TECH. Semester -V

Subject Code: KEC-553

Session: 2023-24, Odd Semester

DRONACHARYA GROUP OF INSTITUTIONS
DEPARTMENT OF ECE
#27 KNOWLEDGE PARK 3
GREATER NOIDA

AFFILIATED TO Dr. A P J ABDUL KALAM TECHNICAL UNIVERSITY

LUCKNOW

List of Experiments mapped with COs

S. No.	Name of the Experiments	COs
1.	To verify linear convolution on DSP Processors.	CO1
2.	To verify the circular convolution on DSP Processors.	CO1
3.	To verify N-point DFT & IDFT on DSP Processors.	CO3
4.	To verify N-point FFT algorithm on DSP Processors.	CO2
5.	To design FIR filter (LP/HP/BP/BR) using windowing technique a) Using Rectangular window b) Using Triangular window	CO4
6.	To design FIR filter (LP/HP/BP/BR) using windowing technique a) Using Hamming window b) Using Hanning window c) Using Blackmann window	CO4
7.	To design FIR filter (LP/HP/BP/BR) using Kaiser window.	CO4
8.	To find the frequency response of analog Butterworth prototype filters (LP/HP/BP/BR).	CO4
9.	To find the frequency response of analog Chebyshev prototype filters (LP/HP/BP/BR).	CO4
10.	To Implement IIR Butterworth filter (LP/HP/BP/BR) using transformation techniques.	CO4
11.	To Implement IIR Chebyshev filter (LP/HP/BP/BR) using transformation techniques.	CO4
12.	Design of FIR filters using frequency sampling method.	CO2

Experiment-1 LINEAR CONVOLUTION

AIM: To verify Linear Convolution.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	- Simulator
Software	- CCStudio 4.0 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe, 5V DC supply

THEORY:

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

$$y(n) = \sum_{k=0}^{N-1} x_1(k)x_2(n-k) \quad 0 < n < N-1 \qquad (1)$$

In this equation, x1(k), x2(n-k) and y(n) represent the input to and output from the system at time n. Here we could see that one of the input is shifted in time by a value every time it is multiplied with the other input signal. Linear Convolution is quite often used as a method of implementing filters of various types.

1.1 PROGRAM:

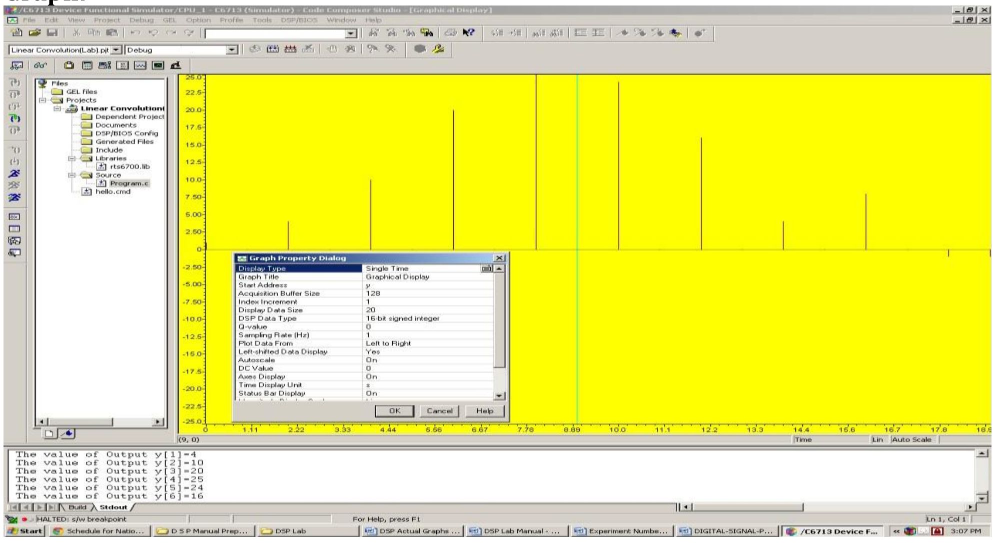
```
#include<stdio.h>
int x[15],h[15],y[15];
main()
{
inti,j,m,n;
printf("\n Enter value for m");
scanf("%d",&m);
printf("\n Enter value for n");
scanf("%d",&n);
printf("Enter values for input x(n)\n");
for(i=0;i<m;i++)
scanf("%d",&x[i]);
printf("Enter values for input h(n)\n");
for(i=0;i<n;i++)
scanf("%d",&h[i]);
//Padding of Zeros//
for(i=m;i<=m+n-1;i++)
x[i]=0;
for(i=n;i<=m+n-1;i++)
h[i]=0;
//Convolution Operation//
for(i=0;i<m+n-1;i++)
{
```

```
y[i]=0;
for(j=0;j<=i;j++)
{
y[i]=y[i]+(x[j]*h[i-j]);
}
//Displaying the Output//
}
for(i=0;i<m+n-1;i++)
printf("\n The value of Output y[%d]=%d",i,y[i]);
}
```

Result:

```
enter value for m: 4
enter value for n: 4
Enter values for input x(n):
1
2
3
4
Enter Values for input h(n):
1
2
3
4
The value of Output y[0]=1
The value of Output y[1]=4
The value of Output y[2]=10
The value of Output y[3]=20
The value of Output y[4]=25
The value of Output y[5]=24
The value of Output y[6]=16
```

Graph:



Graph Property Dialog	
Display Type	Single Time
Graph Title	Graphical Display
Start Address	y
Acquisition Buffer Size	128
Index Increment	1
Display Data Size	20
DSP Data Type	16-bit signed integer
Q-value	0
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	s
Status Bar Display	On

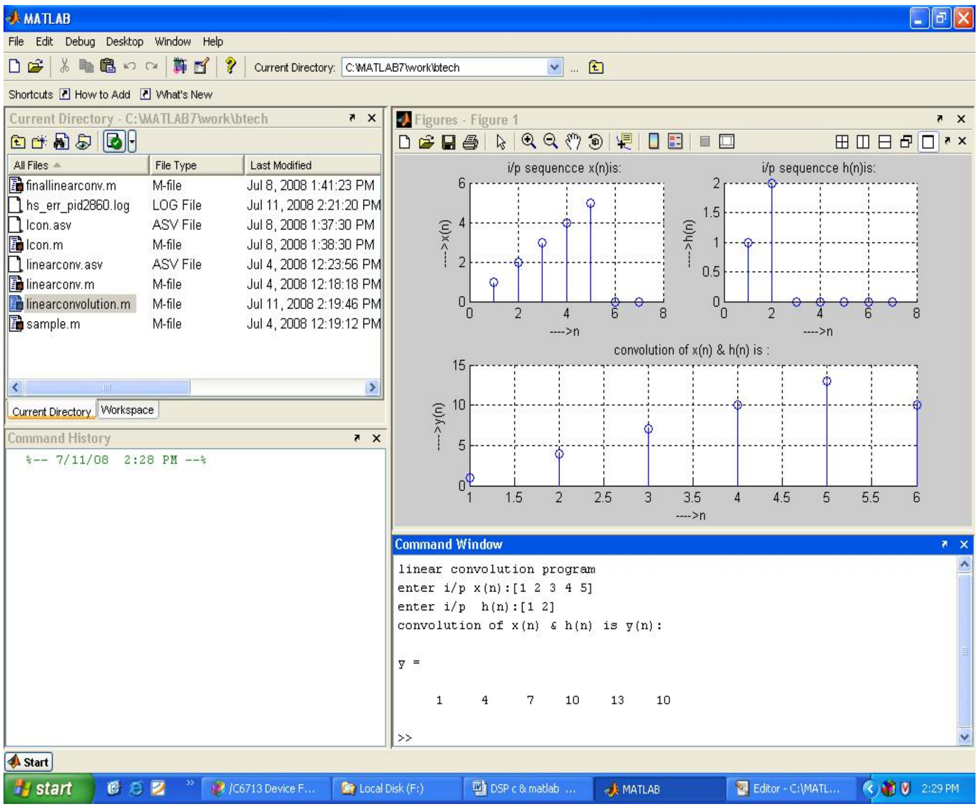
```
The value of Output y[1]=4
The value of Output y[2]=10
The value of Output y[3]=20
The value of Output y[4]=25
The value of Output y[5]=24
The value of Output y[6]=16
```

HALTED: s/w breakpoint

1.2 % MATLAB program for linear convolution

```
%linear convolution program

clc;
clear all;
close all;
disp('linear convolution program');

x=input('enter i/p x(n):');
m=length(x);
h=input('enter i/p h(n):');
n=length(h);

x=[x,zeros(1,n)];
subplot(2,2,1), stem(x);
title('i/p sequence x(n)is:');
xlabel('--- >n');
ylabel('--- >x(n)');grid;

h=[h,zeros(1,m)];
subplot(2,2,2), stem(h);
title('i/p sequence h(n)is:');
xlabel('--- >n');
ylabel('--- >h(n)');grid;
disp('convolution of x(n) & h(n) is y(n):');
y=zeros(1,m+n-1);
for i=1:m+n-1
   y(i)=0;
   for j=1:m+n-1
      if(j<i+1)
      y(i)=y(i)+x(j)*h(i-j+1);
      end
   end
end
y
subplot(2,2,[3,4]),stem(y);
title('convolution of x(n) & h(n) is :');
xlabel('--- >n');
ylabel('--- >y(n)');grid;
```

Graph:



```
linear convolution program
enter i/p x(n):[1 2 3 4 5]
enter i/p  h(n):[1 2]
convolution of x(n) & h(n) is y(n):

y =

     1     4     7    10    13    10

>>
```

Result : Hence, linear convoluion of two sequences has been verified using CCS and MATLAB.

Experiment-2 CIRCULAR CONVOLUTION

AIM: To verify Circular Convolution.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	**-** Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY

Circular convolution is another way of finding the convolution sum of two input signals. It resembles the linear convolution, except that the sample values of one of the input signals is folded and right shifted before the convolution sum is found. Also note that circular convolution could also be found by taking the DFT of the two input signals and finding the product of the two frequency domain signals. The Inverse DFT of the product would give the output of the signal in the time domain which is the circular convolution output. The two input signals could have been of varying sample lengths. But we take the DFT of higher point, which ever signals levels to. For eg. If one of the signal is of length 256 and the other spans 51 samples, then we could only take 256 point DFT. So the output of IDFT would be containing 256 samples instead of 306 samples, which follows N1+N2 – 1 where N1 & N2 are the lengths 256 and 51 respectively of the two inputs. Thus the output which should have been 306 samples long is fitted into 256 samples. The
256 points end up being a distorted version of the correct signal. This process is called circular convolution.

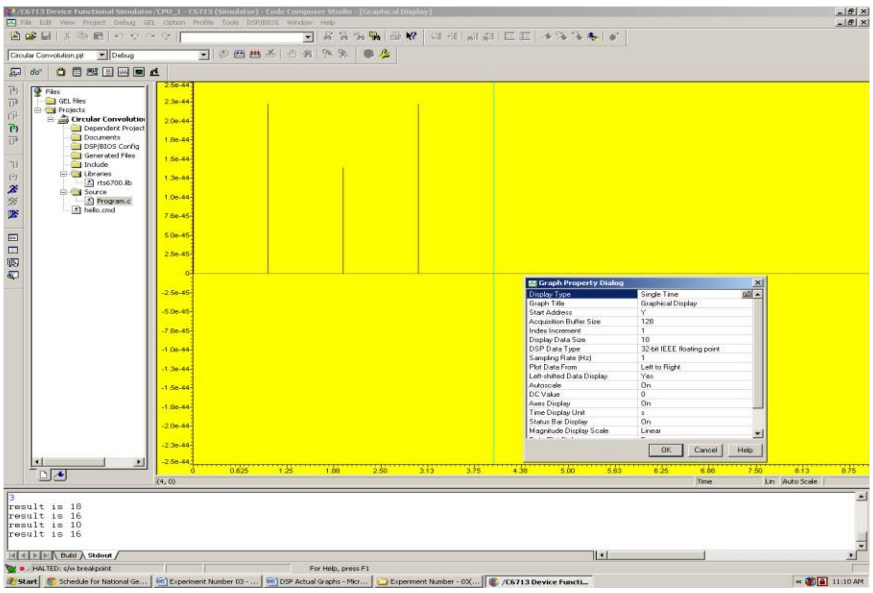
3.1 Program:

```
#include <stdio.h>
int n1,n2,i,j,n,c,x[10]=0,h[10]=0,y[10]=0,a[10]=0;
main()
{
printf("Enter the length of the sequence n1: ");
scanf("%d",&n1);
printf("Enter the length of the sequence n2: ");
scanf("%d",&n2);
printf("Enter the first sequence: ");
for(i=0;i<n1;i++)
{
scanf("%d",&x[i]);
}
printf("Enter the second sequence: ");
for(i=0;i<n2;i++)

{
scanf("%d",&h[i]);
}
if(n1>n2)
n=n1;
else
n=n2;
a[0]=h[0];
for(i=1;i<n;i++)
{
a[i]=h[n-i];
}
for(j=0;j<n;j++)
{ y[j]=
0;
c=x[0];
for(i=0;i<n;i++)
{
y[j]=(y[j]+(x[i]*a[i]));
x[i]=x[i+1];
}
x[n-1]=c;
printf("result is %d /n",y[j]);
}
}
```

Result:

```
Enter the length of the sequence n1: 4
Enter the length of the sequence n2: 3
Enter the first sequence: 1
2
3
4
Enter the second sequence: 1
2
3
The Circular Convolution is
18 16 10 16
```

Graph:



Graph Property Dialog	
Display Type	Single Time
Graph Title	Graphical Display
Start Address	Y
Acquisition Buffer Size	128
Index Increment	1
Display Data Size	10
DSP Data Type	32-bit IEEE floating point
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	s
Status Bar Display	On
Magnitude Display Scale	Linear

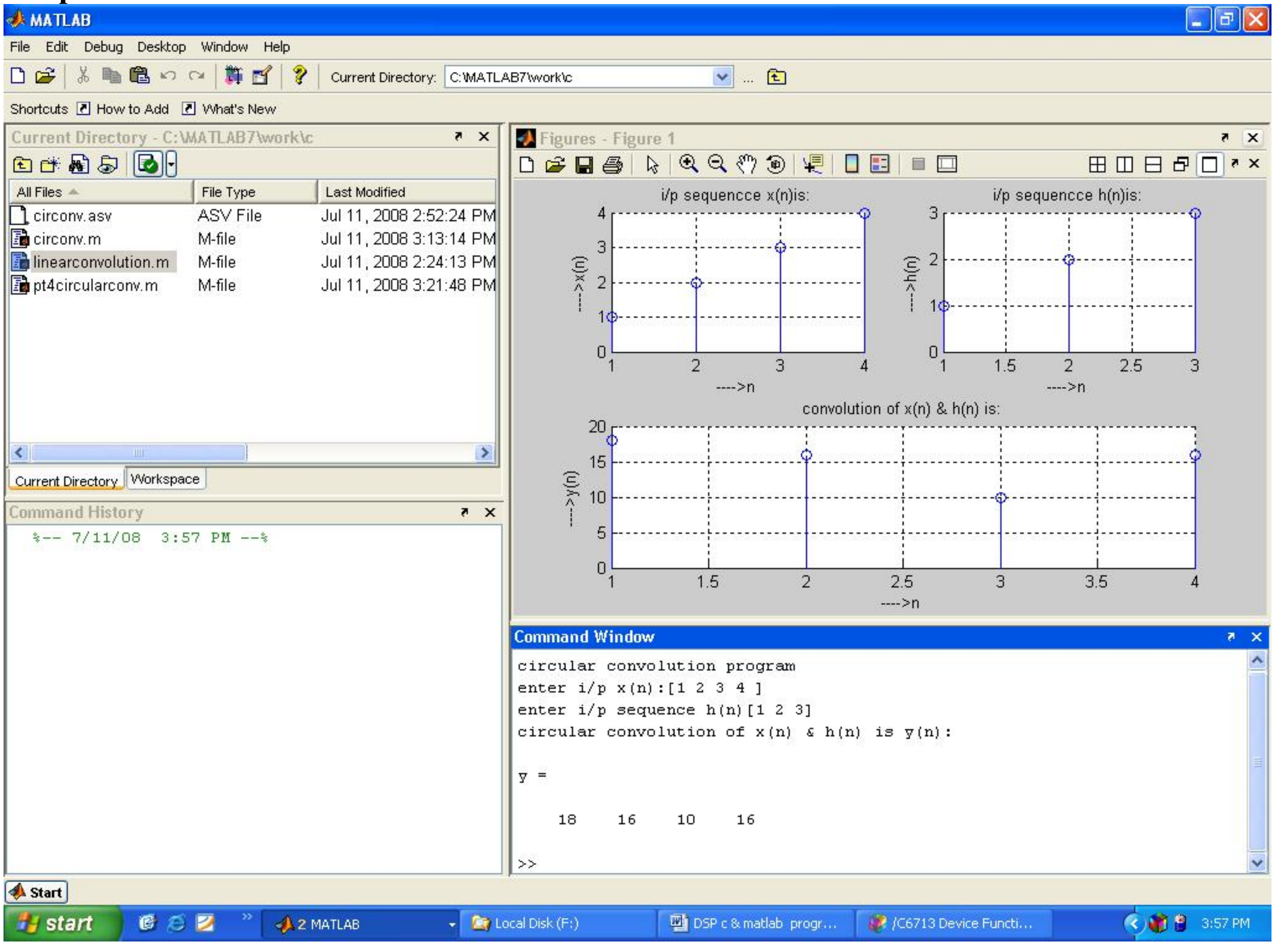
```
3
result is 18
result is 16
result is 10
result is 16
```

3.2 %Circular Convolution Program

```
clc;
clear all;
close all;
disp('circular convolution program');

x=input('enter i/p x(n):');
m=length(x);
h=input('enter i/p sequence h(n)');
n=length(h);

subplot(2,2,1), stem(x);
title('i/p sequencce x(n)is:');
xlabel('--- >n');
ylabel('--- >x(n)');grid;

subplot(2,2,2), stem(h);
title('i/p sequencce h(n)is:');
xlabel('--- >n');
ylabel('--- >h(n)');grid;

disp('circular convolution of x(n) & h(n) is y(n):');
if(m-n~=0)
   if(m>n)
      h=[h,zeros(1,m-n)];
      n=m;
   end
      x=[x,zeros(1,n-m)];
      m=n;
end

y=zeros(1,n);
y(1)=0;
a(1)=h(1);

for j=2:n
   a(j)=h(n-j+2);
end

%ciruclar conv
for i=1:n
   y(1)=y(1)+x(i)*a(i);
   end

for k=2:n
   y(k)=0;
   % circular shift
   for j=2:n
      x2(j)=a(j-1);
```

```
    end
    x2(1)=a(n);

    for i=1:n
      if(i<n+1)
        a(i)=x2(i);
      y(k)=y(k)+x(i)*a(i);
      end
    end
end
y
subplot(2,2,[3,4]),stem(y);
title('convolution of x(n) & h(n) is:');
xlabel('--- >n');
ylabel('--- >y(n)');grid;
```

Graph :


MATLAB
File Edit Debug Desktop Window Help
Current Directory: C:\MATLAB7\work\c
Current Directory - C:\MATLAB7\work\c
Figures - Figure 1
i/p sequencce x(n)is:
i/p sequencce h(n)is:
convolution of x(n) & h(n) is:
---->n
---->x(n)
---->h(n)
---->y(n)
Command History
Command Window
start

All Files	File Type	Last Modified
circonv.asv	ASV File	Jul 11, 2008 2:52:24 PM
circonv.m	M-file	Jul 11, 2008 3:13:14 PM
linearconvolution.m	M-file	Jul 11, 2008 2:24:13 PM
pt4circularconv.m	M-file	Jul 11, 2008 3:21:48 PM

```
%-- 7/11/08  3:57 PM --%
```

```
circular convolution program
enter i/p x(n):[1 2 3 4 ]
enter i/p sequence h(n)[1 2 3]
circular convolution of x(n) & h(n) is y(n):

y =

    18    16    10    16

>>
```

Result : Hence Circular convoluion of two sequences has been verified using CCS and MATLAB

Experiment-3. DISCRETE FOURIER TRANSFORM

AIM: To verify N-point DFT & IDFT of discrete time sequence.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	**-** Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY:

DFT of a discrete time signal transforms the signal in time domain into frequency domain signal.

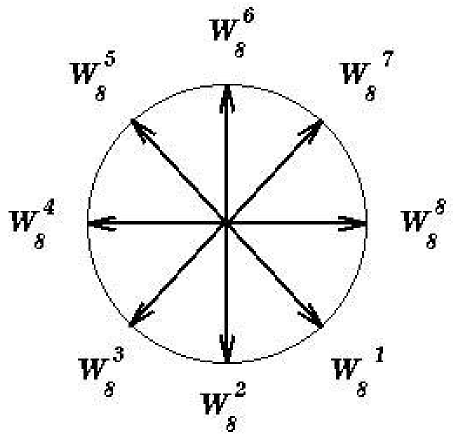
DFT EQUTION:

The sequence of *N* complex numbers $x_0, \ldots, x_{N-1}$ is transformed into another sequence of *N* complex numbers according to the DFT formula:

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

$$where\ \ W_N^{nk} = e^{-j\frac{2\pi nk}{N}}\ \text{[TWIDDLE FACTOR]}$$

where N = number of samples. If we take an 8 bit sample sequence we can represent the twiddle factor as a vector in the unit circle. e.g.



Note:

1 It is periodic. (i.e. it goes round and round the circle !!)
2 That the vectors are symmetric
3 The vectors are equally spaced around the circle

The **inverse discrete Fourier transform (IDFT)** is given by:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{+i2\pi \frac{k}{N} n}.$$

3.1 DFT

Program

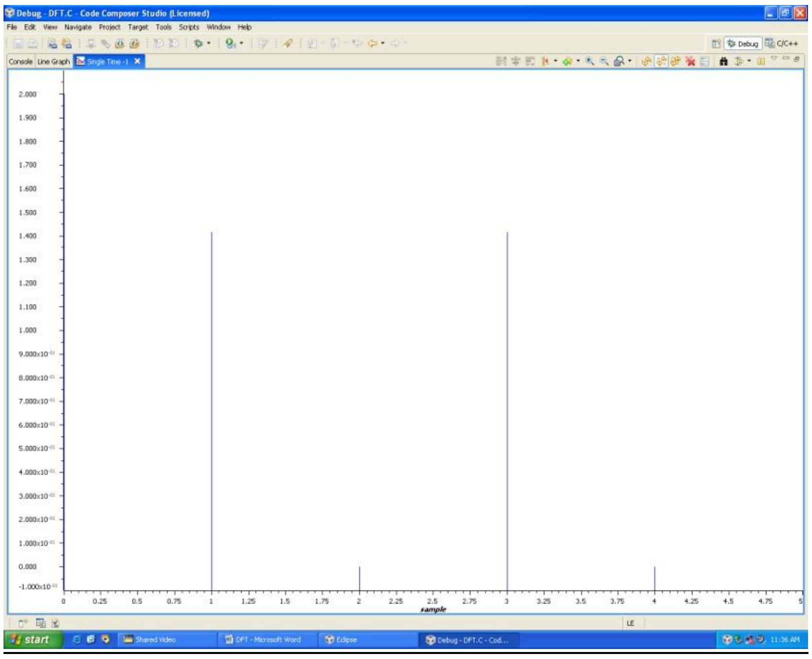
```
#include<stdio.h>
#include<math.h>
int x[32],N,k,n,i;
float
pi=3.1416,sumre=0,sumim=0,out_real[8]={0.0},out_imag[8]={0.0},m[20]={0.0},p[20]={0.0};
main()
{
        printf ("enter lenght\n");
        scanf ("%d",&N);
        printf("enter the first sequence\n");
        for(i=0;i<N;i++)
        scanf ("%d",&x[i]);
        for(k=0;k<N;k++)
        {
                 sumre=0;
                 sumim=0;
        for(n=0;n<N;n++)
        {
                sumre=sumre+x[n]*cos(2*pi*k*n/N);
                sumim=sumim-x[n]*sin(2*pi*k*n/N);
        }
        out_real[k]=sumre;
        out_imag[k]=sumim;
        printf("x[%d]=\t%f\t+j*\t%f\n",k,out_real[k],out_imag[k]);
        m[k]=sqrt(out_real[k]*out_real[k]+out_imag[k]*out_imag[k]);
        printf("magnitude =%f\n",m[k]);
        p[k]=atan(out_imag[k]/out_real[k]);
        printf("phase value =%f\n",p[k]);
        }
}
```

Console Window

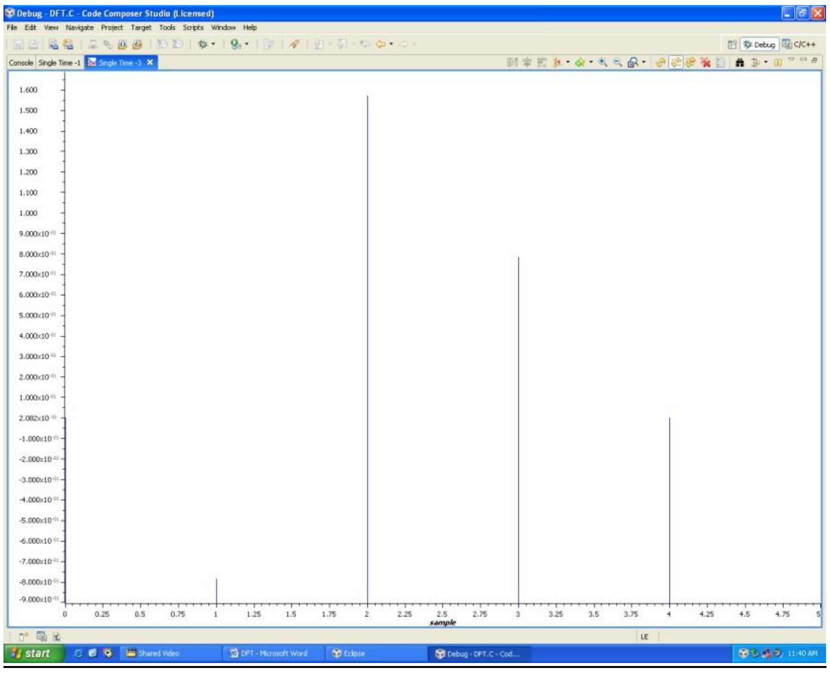
```
enter length  4
enter the first sequence
1
1
0
0
x[0]=    2.000000        +j*       0.000000
magnitude =2.000000
phase value =0.000000
x[1]=    0.999996        +j*       -1.000000
magnitude =1.414211
phase value =-0.785400
x[2]=    0.000000        +j*       0.000007
magnitude =0.000007
phase value =1.570793
x[3]=    1.000011        +j*       1.000000
magnitude =1.414221
phase value =0.785393
```

<u>Output Graph</u>

<u>Magnitude</u>



<u>Phase</u>


Debug - DFT.C - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
Console Single Time -1 Single Time -3
1.600
1.500
1.400
1.300
1.200
1.100
1.000
0
0.25
0.5
0.75
1
1.25
1.5
1.75
2
2.25
2.5
2.75
3
3.25
3.5
3.75
4
4.25
4.5
4.75
5
sample
start

3.2 IDFT

Program

```
#include<stdio.h>
#include<math.h>
int x[32],N,k,n,i;
float
pi=3.1416,sumre=0,sumim=0,out_real[8]={0.0},out_imag[8]={0.0},m[20]={0.0},p[20]={0.0};
main()
{
        printf ("enter lenght\n");
        scanf ("%d",&N);
        printf("enter the first sequence\n");
        for(i=0;i<N;i++)
        scanf ("%d",&x[i]);
        for(k=0;k<N;k++)
        {
                 sumre=0;
                sumim=0;
        for(n=0;n<N;n++)
        {
                sumre=sumre+x[n]*cos(2*pi*k*n/N);
                sumim=sumim-x[n]*sin(2*pi*k*n/N);
        }
        out_real[k]=sumre/N;
        out_imag[k]=sumim/N;
        printf("x[%d]=\t%f\t+j*\t%f\n",k,out_real[k],out_imag[k]);
        m[k]=sqrt(out_real[k]*out_real[k]+out_imag[k]*out_imag[k]);
        printf("magnitude =%f\n",m[k]);
        p[k]=atan(out_imag[k]/out_real[k]);
```

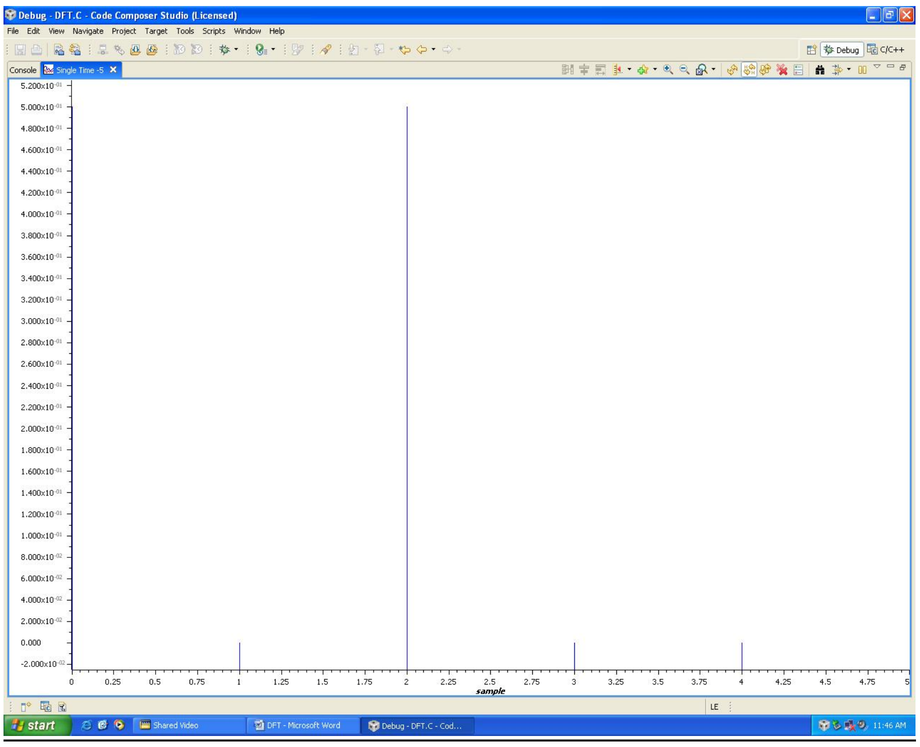
```
          printf("phase value =%f\n",p[k]);
          }
}
```

Console Window

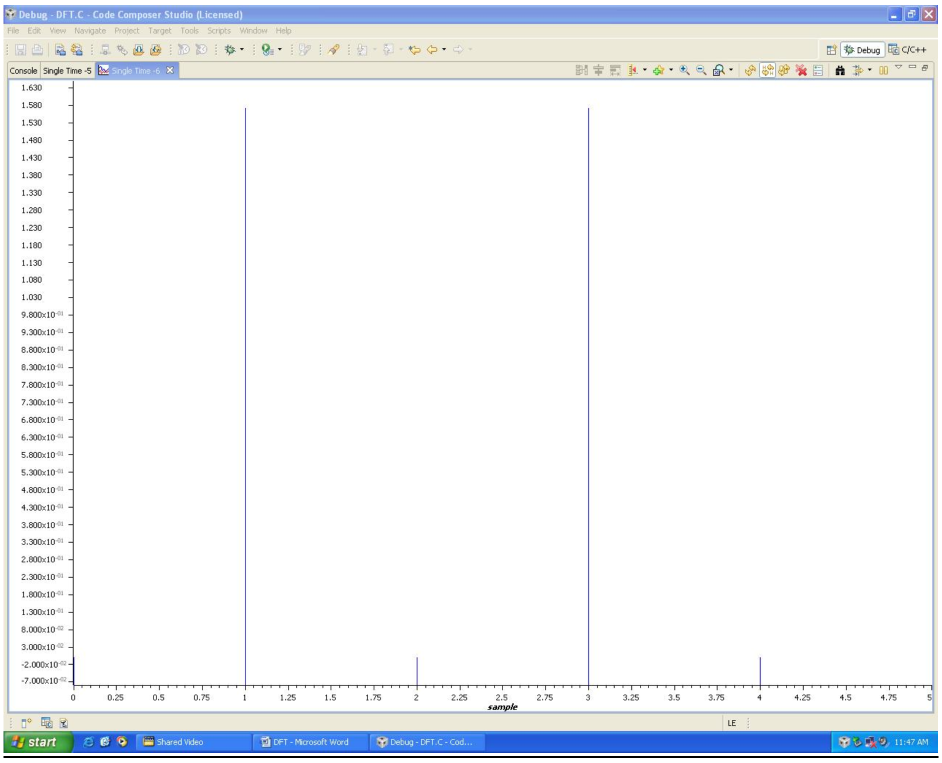
```
enter length 4
enter the first sequence
1
0
1
0
x[0]=   0.500000        +j*      0.000000
magnitude =0.500000
phase value =0.000000
x[1]=   0.000000        +j*      0.000002
magnitude =0.000002
phase value =1.570793
x[2]=   0.500000        +j*      -0.000004
magnitude =0.500000
phase value =-0.000007
x[3]=   0.000000        +j*      0.000005
magnitude =0.000005
phase value =1.570785
```

Output Graph

Magnitude


Debug - DFT.C - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
Debug
C/C++
Console
Single Time -5
sample
start
Shared Video
DFT - Microsoft Word
Debug - DFT.C - Cod...
11:46 AM

Phase



RESULTS: DFT And IDFT Of The Discret Time Sequence has been verified through CCS.

Experiment-4. N-POINT FFT ALGORITHM

AIM: To verify N-point FFT algorithm discrete Time sequence.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY:

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

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

The above equation is the mathematical representation of the DFT. As the number of computations involved in transforming a N point time domain signal into its corresponding frequency domain signal was found to be N^2 complex multiplications, an alternative algorithm involving lesser number of computations is opted.

When the sequence x(n) is divided into 2 sequences and the DFT performed separately, the resulting number of computations would be $N^2/2$

(i.e.)

$$x(k) = \sum_{n=0}^{\frac{N^2}{2-1}} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N^2}{2-1}} x(2n+1) \quad W_N^{(2n+1)k} \qquad (6)$$

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

$$\sum_{n=0}^{\frac{N^2}{2-1}} x(2n) \quad W_N^{2nk}$$ would result in (7)

$(N/2)^2$multiplication"s $\sum_{n=0}^{\frac{N^2}{2-1}} x(2n+1) \quad W_N^{(2n+1)k}$ (8)

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

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

Further solving Eg. (2)

$$x(k) = \sum_{n=0}^{\frac{N^2}{2}-1} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \quad W_N^{(2nk)} W_N^k \quad (9)$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n) \quad W_N^{2nk} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) \quad W_N^{(2nk)} \quad (10)$$

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

W_N → is the twiddle factor

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

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

$$W_N^{\left(K+\frac{N}{2}\right)} = W_N \quad W_N^{\left(K+\frac{N}{2}\right)} \quad (11)$$

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

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

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

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

$$= W_N^{\left(K+\frac{N}{2}\right)} = W_N^k \quad (12)$$

Employing this equation, we deduce

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

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

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

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

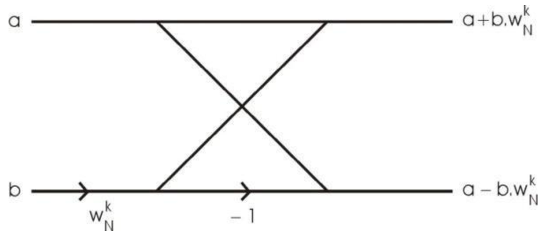
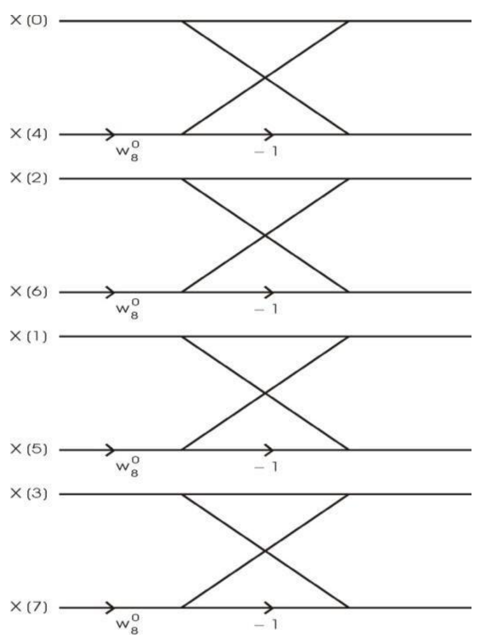
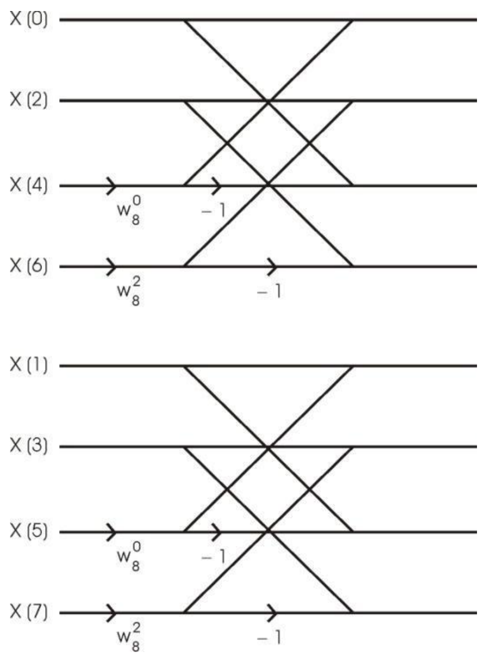


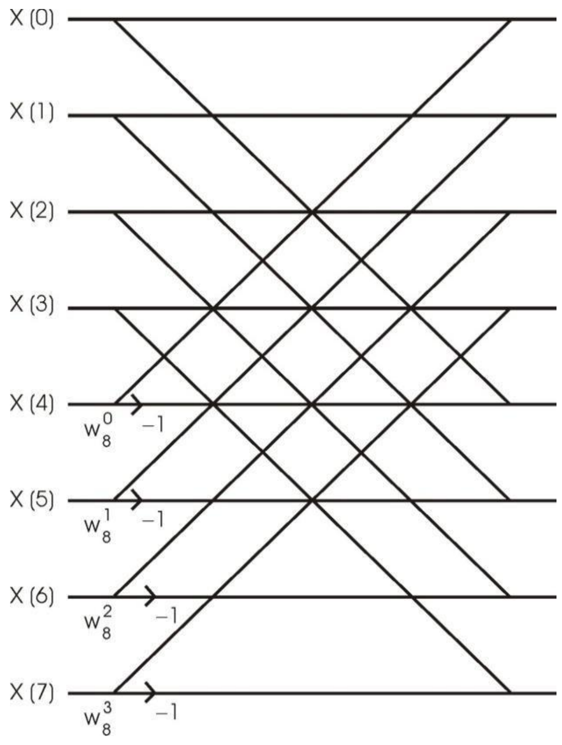
FIG. 3A.1

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





STAGE – I **STAGE - II**



STAGE – III

8 POINT DIT

4.1 PROGRAM:

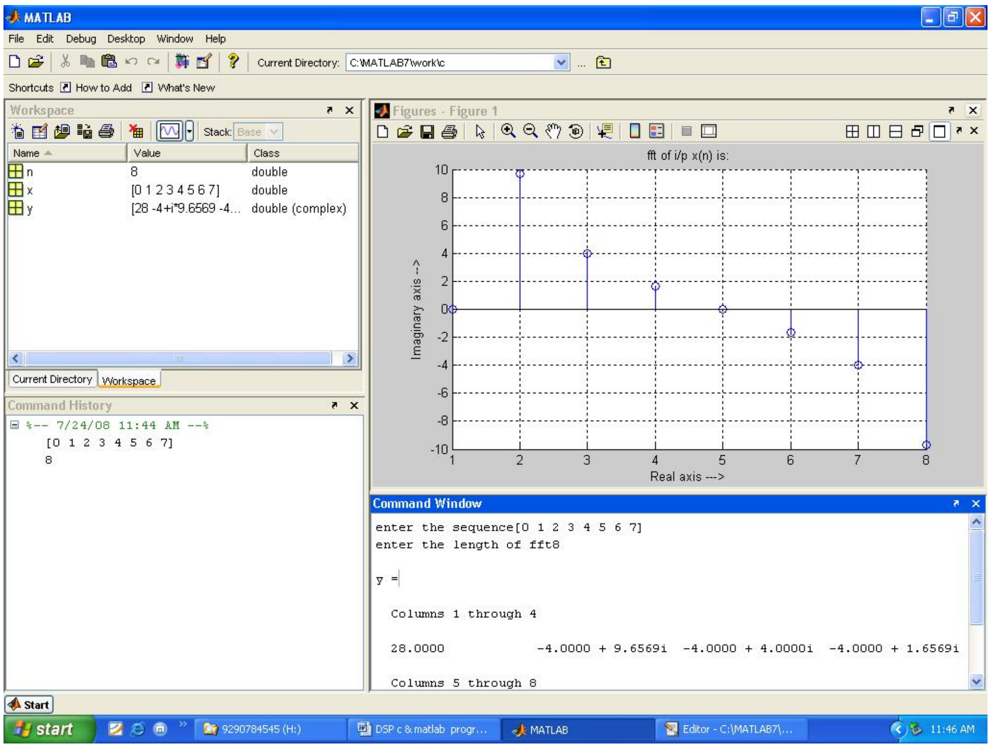
%fast fourier transform

% Matlab code for N point Fast Fourier Transform

close all; clear all;

```
x=input('enter the input sequence');
 N=length(x);
 %Code for FFT
disp('Fast Fourier Transformed Signal');
xk=fft(x,N); disp(xk);
%code for IFFT
disp('Inverse Fast Fourier Transformed Signal');
 ixk=ifft(xk,N); disp(ixk);
%code to plot all sequences
 %input sequence
 n=0:N-1; subplot(2,2,1);
stem(n,x); xlabel('Time Index'); ylabel('Amplitude'); title('input sequence');
 %FFT response sequence
k=0:N-1; subplot(2,2,2); stem(k,abs(xk)); xlabel('Frequency Index'); ylabel('Magnitude');
 title('FFT response sequence');
 %IFFT response sequence
 n=0:N-1; ixk=ixk/N; subplot(2,2,3); stem(n,abs(ixk)); xlabel('Time Index'); ylabel('Magnitude');
title('IFFT response sequence');
```

Graph:



Name	Value	Class
n	8	double
x	[0 1 2 3 4 5 6 7]	double
y	[28 -4+i*9.6569 -4...	double (complex)

```
%-- 7/24/08 11:44 AM --%
    [0 1 2 3 4 5 6 7]
    8
```

```
enter the sequence[0 1 2 3 4 5 6 7]
enter the length of fft8

y =

  Columns 1 through 4

  28.0000            -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i

  Columns 5 through 8
```

4.2 C Program to Implement FFT

```
#include <math.h>
#define PTS 128 //# of points for FFT
#define PI 3.14159265358979
typedef struct {float real,imag;} COMPLEX;
void FFT(COMPLEX *Y, int n); //FFT prototype
float iobuffer[PTS]; //as input and output buffer
float x1[PTS],x[PTS]; //intermediate buffer
short i; //general purpose index variable
short buffercount = 0; //number of new samples in iobuffer
short flag = 0; //set to 1 by ISR when iobuffer full
float y[128];
COMPLEX w[PTS]; //twiddle constants stored in w
COMPLEX samples[PTS]; //primary working buffer
main()
{
float j,sum=0.0 ;
int n,k,i,a;
for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w
{
w[i].real = cos(2*PI*i/(PTS*2.0)); /*Re component of twiddle constants*/
w[i].imag =-sin(2*PI*i/(PTS*2.0)); /*Im component of twiddle constants*/
}
/****************Input Signal X(n) *************************/
for(i=0,j=0;i<PTS;i++)
{
x[i] = sin(2*PI*5*i/PTS); // Signal x(Fs)=sin(2*pi*f*i/Fs);
samples[i].real=0.0;
samples[i].imag=0.0;
}
/********************Auto Correlation of X(n)=R(t) ***********/
for(n=0;n<PTS;n++)
{
sum=0;
for(k=0;k<PTS-n;k++)
{
sum=sum+(x[k]*x[n+k]); // Auto Correlation R(t)
}
iobuffer[n] = sum;
}
/********************** FFT of R(t) ***********************/
for (i = 0 ; i < PTS ; i++) //swap buffers
{
samples[i].real=iobuffer[i]; //buffer with new data
}
for (i = 0 ; i < PTS ; i++)
samples[i].imag = 0.0; //imag components = 0
FFT(samples,PTS); //call function FFT.c
/******************** PSD ********************/
for (i = 0 ; i < PTS ; i++) //compute magnitude
{
x1[i] = sqrt(samples[i].real*samples[i].real
+ samples[i].imag*samples[i].imag);
}
} //end of main
```

FFT.c:

```
#define PTS 128 //# of points for FFT
typedef struct {float real,imag;} COMPLEX;
extern COMPLEX w[PTS]; //twiddle constants stored in w
void FFT(COMPLEX *Y, int N) //input sample array, # of points
{
COMPLEX temp1,temp2; //temporary storage variables
int i,j,k; //loop counter variables
int upper_leg, lower_leg; //indexof upper/lower butterfly leg
int leg_diff; //difference between upper/lower leg
int num_stages = 0; //number of FFT stages (iterations)
int index, step; //index/step through twiddle constant
i = 1; //log(base2) of N points= # of stages
```

```
do
{
num_stages +=1;
i = i*2;
}while (i!=N);
leg_diff = N/2; //difference between upper&lower legs
step = (PTS*2)/N; //step between values in twiddle.h// 512
for (i = 0;i < num_stages; i++) //for N-point FFT
{
index = 0;
for (j = 0; j < leg_diff; j++)
{
for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
{
lower_leg = upper_leg+leg_diff;
temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
temp2.real = (Y[upper_leg]).real - (Y[lower_leg]).real;
temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
(Y[lower_leg]).real = temp2.real*(w[index]).real
-temp2.imag*(w[index]).imag;
(Y[lower_leg]).imag = temp2.real*(w[index]).imag
+temp2.imag*(w[index]).real;
(Y[upper_leg]).real = temp1.real;
(Y[upper_leg]).imag = temp1.imag;
}
index += step;
}
leg_diff = leg_diff/2;
step *= 2;
}
j = 0;
for (i = 1; i < (N-1); i++) //bit reversal for resequencing data
{
```

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

RESULT: FFT of a Discrete Time Sequence has been verified through MATLAB.

Experiment-5. FIR FILTER DESIGN

AIM: To design FIR Filter (LP/HP/BP/BR) Using Windowing technique

a. Rectangular window
b. Triangular window
c. Kaiser window

EQUIPMENTS:

Operating System	– Windows XP
Constructor	**-** Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY:

In this method, the desired frequency response specification $H_d(w)$, corresponding unit sample response $h_d(n)$ is determined using the following relation

$$h_d(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(w)\, e^{jwn} dw$$

$$H_d(w) = \sum_{n=-\infty}^{\infty} h_d(n)\, e^{-jwn}$$

In general, unit sample response $h_d(n)$ obtained from the above relation is infinite in duration, so it must be truncated at some point say $n= M$-1 to yield an FIR filter of length M (i.e. 0 to M-1). This truncation of $h_d(n)$ to length M-1 is same as multiplying $h_d(n)$ by the rectangular window defined as

$$\begin{aligned} w(n) = \ & 1 \qquad 0 \le n \le M\text{-}1 \\ & 0 \qquad \text{otherwise} \end{aligned}$$

Thus the unit sample response of the FIR filter becomes

$$\begin{aligned} h(n) &= h_d(n)\, w(n) \\ &= h_d(n) \qquad 0 \le n \le M\text{-}1 \\ &= 0 \qquad \text{otherwise} \end{aligned}$$

Now, the multiplication of the window function $w(n)$ with $h_d(n)$ is equivalent to convolution of $H_d(w)$ with $W(w)$, where $W(w)$ is the frequency domain representation of the window function

$$W(w) = \sum_{n=0}^{M-1} w(n)e^{-jwn}$$

Thus the convolution of $H_d(w)$ with $W(w)$ yields the frequency response of the truncated FIR filter

$$H(w) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(v)W(w-v)dw$$

The frequency response can also be obtained using the following relation

$$H(w)= \sum_{n=0}^{M-1} h(n)e^{-jwn}$$

But direct truncation of $h_d(n)$ to M terms to obtain $h(n)$ leads to the Gibbs phenomenon effect which manifests itself as a fixed percentage overshoot and ripple before and after an approximated discontinuity in the frequency response due to the non-uniform convergence of the fourier series at a discontinuity. In order to reduce the ripples, instead of multiplying $h_d(n)$ with a rectangular window $w(n)$, $h_d(n)$ is multiplied with a window function that contains a taper and decays toward zero gradually, instead of abruptly as it occurs in a rectangular window. As multiplication of sequences $h_d(n)$ and $w(n)$ in time domain is equivalent to convolution of $H_d(w)$ and $W(w)$ in the frequency domain, it has the effect of smoothing $H_d(w)$.

The several effects of windowing the Fourier coefficients of the filter on the result of the frequency response of the filter are as follows:

(i) A major effect is that discontinuities in $H(w)$ become transition bands between values on either side of the discontinuity.

(ii) The width of the transition bands depends on the width of the main lobe of the frequency response of the window function, $w(n)$ i.e. $W(w)$.

(iii) Since the filter frequency response is obtained via a convolution relation , it is clear that the resulting filters are never optimal in any sense.

(iv) As M (the length of the window function) increases, the mainlobe width of $W(w)$ is reduced which reduces the width of the transition band, but this also introduces more ripple in the frequency response.

(v) The window function eliminates the ringing effects at the bandedge and does result in lower sidelobes at the expense of an increase in the width of the transition band of the filter.

Some of the windows commonly used are as follows:

1. Bartlett triangular window:

$$W(n) = \frac{2(n+1)}{N+1} \qquad n = 0,1,2,\ldots\ldots,(N\text{-}1)/2$$
$$= 2 - \frac{2(n+1)}{N+1} \qquad n = (N\text{-}1)/2,\ldots\ldots,N\text{-}1$$
$$= 0, \qquad \text{otherwise}$$

-

2-5. Generalized cosine windows
(Rectangular, Hanning, Hamming and Blackman)

$$W(n) = a - b\cos(2p(n+1)/(N+1)) + c\cos(4p(n+1)/(N+1)) \qquad n= 0,1\ldots N\text{-}1$$
$$= 0 \qquad \text{otherwise}$$

6. Kaiser window with parameter ß :

$$W(n) = \frac{I_o(\beta\sqrt{1-(2(n+1)/(N+1))^2})}{I_o(\beta)} \qquad n= 0,1,\ldots.,N\text{-}1$$
$$= 0 \qquad \text{otherwise}$$

The Bartlett window reduces the overshoot in the designed filter but spreads the transition region considerably. The Hanning, Hamming and Blackman windows use progressively more complicated cosine functions to provide a smooth truncation of the ideal impulse response and a frequency response that looks better. The best window results probably come from using the Kaiser window, which has a parameter ß that allows adjustment of the compromise between the overshoot reduction and transition region width spreading.
The major advantages of using window method is their relative simplicity as compared to other methods and ease of use. The fact that well defined equations are often available for calculating the window coefficients has made this method successful.

There are following problems in filter design using window method:
(i) This method is applicable only if $H_d(w)$ is absolutely integrable . When $H_d(w)$ is complicated or cannot easily be put into a closed form mathematical expression, evaluation of $h_d(n)$ becomes difficult.
(ii) The use of windows offers very little design flexibility e.g. in low pass filter design, the passband edge frequency generally cannot be specified exactly since the window smears the discontinuity in frequency. Thus the ideal LPF with cut-off frequency fc, is smeared by the window to give a frequency response with passband response with passband cutoff frequency f_1 and stopband cut-off frequency f_2.
(iii) Window method is basically useful for design of prototype filters like lowpass,highpass,bandpass etc. This makes its use in speech and image processing applications very limited.

5.1 FIR Filters Using Triangular Window

Program:

```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
fp = input('Enter the passband frequency = ');
fs = input('Enter the stopband frequency = ');
f = input('Enter the sampling frequency = ');
```

```
wp = 2*fp/f;
ws = 2*fs/f;
num = -20*log10(sqrt(rp*rs))-13;
dem = 14.6*(fs-fp)/f;
n = ceil(num/dem);
n1 = n+1;
if (rem(n,2)==50)
n1 = n;
n = n-1;
end
y = bartlett(n1);
% low-pass filter
b = fir1(n,wp,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title('Magnitude Response of LPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of HPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% band pass filter
wn = [wp ws];
b = fir1(n,wn,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
title('Magnitude Response of BPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
```

```
title('Magnitude Response of BSF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.04
Enter the stopband ripple = 0.02
Enter the passband frequency = 1500
Enter the stopband frequency = 2000
Enter the sampling frequency = 8000
```

Waveform:

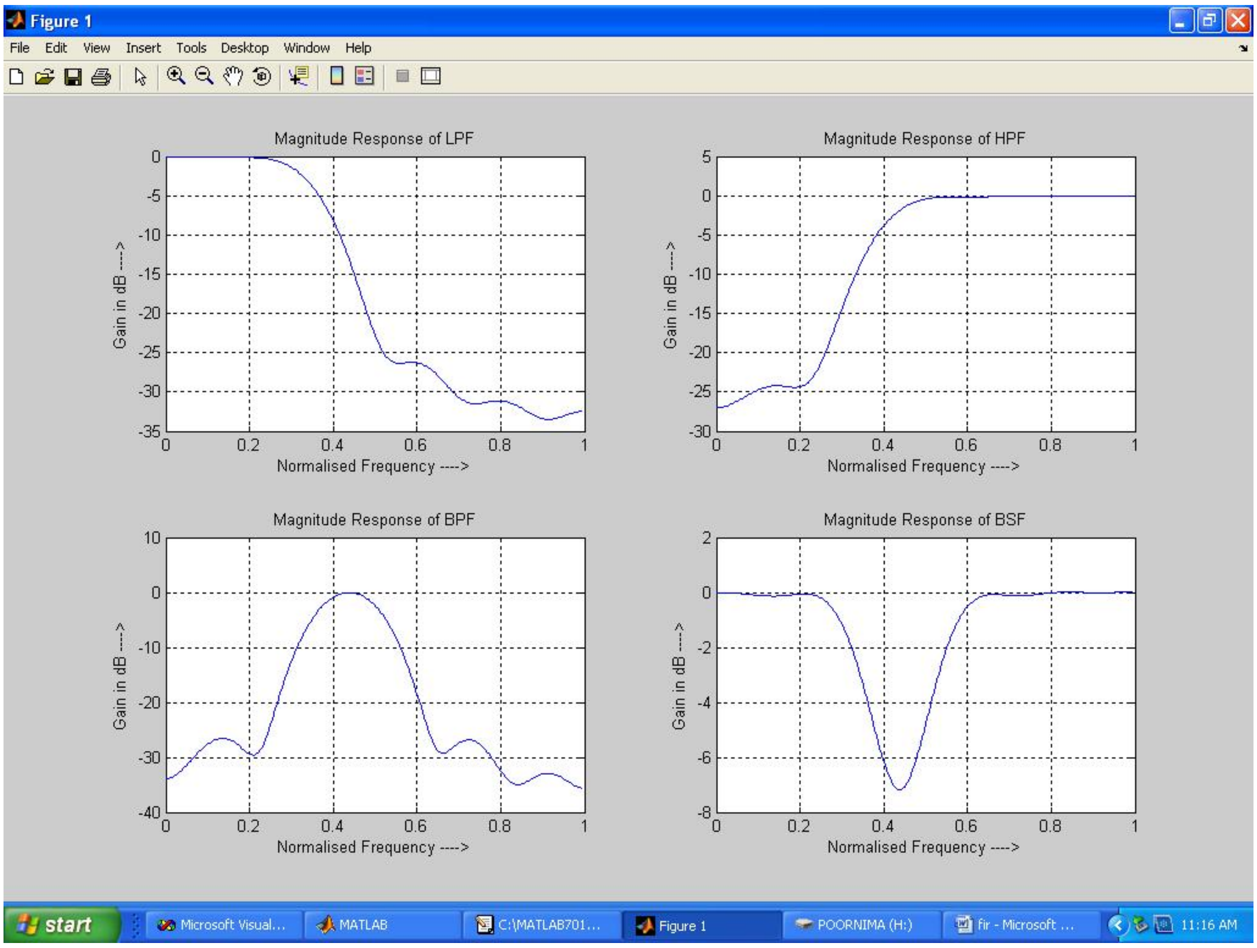


Figure 1
File Edit View Insert Tools Desktop Window Help
Magnitude Response of LPF
Magnitude Response of HPF
Magnitude Response of BPF
Magnitude Response of BSF
Gain in dB ---->
Normalised Frequency ---->
start
Microsoft Visual...
MATLAB
C:\MATLAB701...
Figure 1
POORNIMA (H:)
fir - Microsoft ...
11:16 AM

5.2 FIR Filters Using Rectangular Window

Program:

```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
fp = input('Enter the passband frequency = ');
fs = input('Enter the stopband frequency = ');
f = input('Enter the sampling frequency = ');
```

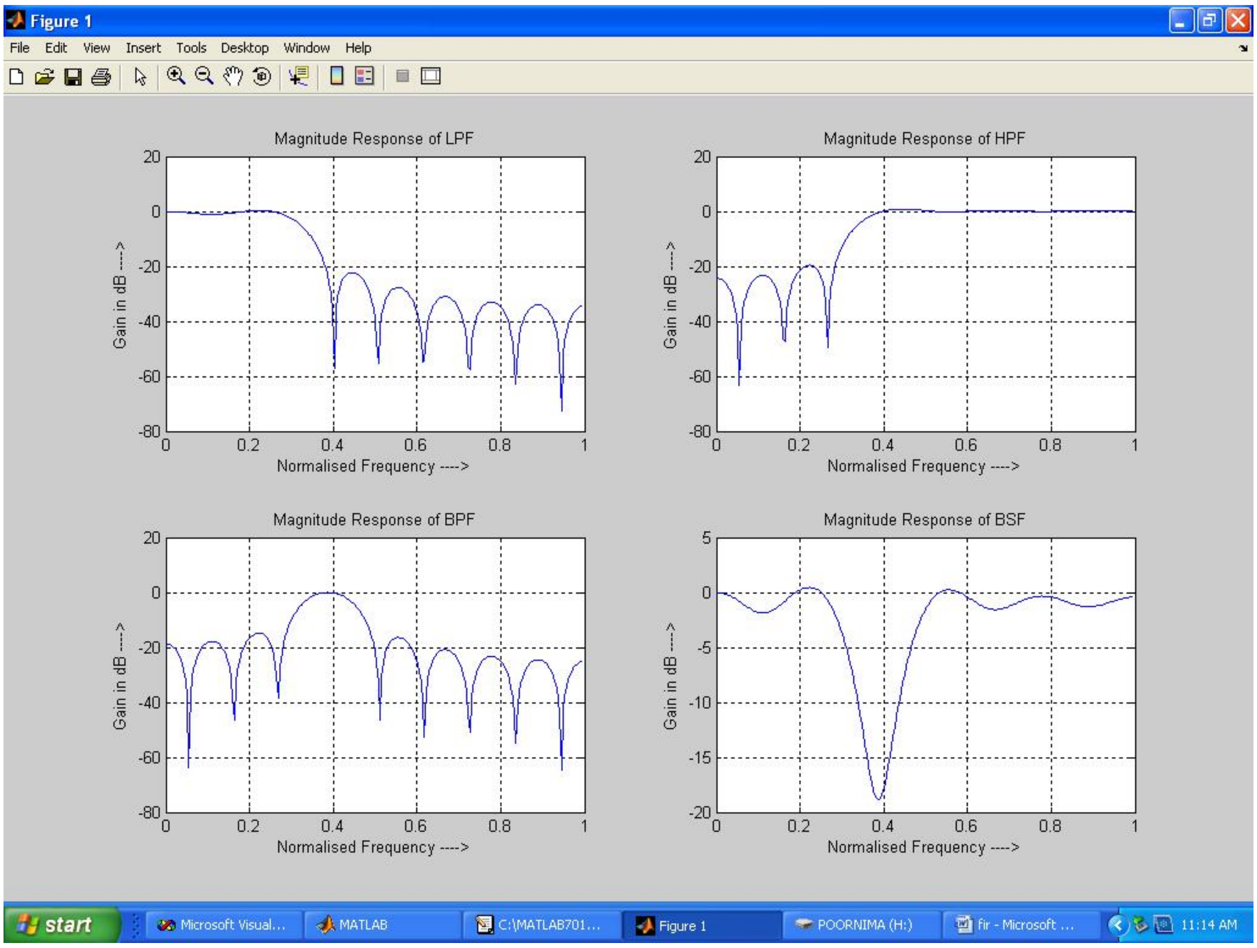
```
wp = 2*fp/f;
ws = 2*fs/f;
num = -20*log10(sqrt(rp*rs))-13;
dem = 14.6*(fs-fp)/f;
n = ceil(num/dem);
n1 = n+1;
if (rem(n,2)==50)
n1 = n;
n = n-1;
end
y = boxcar(n1);
% low-pass filter
b = fir1(n,wp,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title('Magnitude Response of LPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of HPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% band pass filter
wn = [wp ws];
b = fir1(n,wn,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
title('Magnitude Response of BPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
```

```
title('Magnitude Response of BSF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.05
Enter the stopband ripple = 0.04
Enter the passband frequency = 1500
Enter the stopband frequency = 2000
Enter the sampling frequency = 9000
```

Waveform:



6.3 FIR Filters Using Kaiser Window

Program:

```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
fp = input('Enter the passband frequency = ');
fs = input('Enter the stopband frequency = ');
f = input('Enter the sampling frequency = ');
```

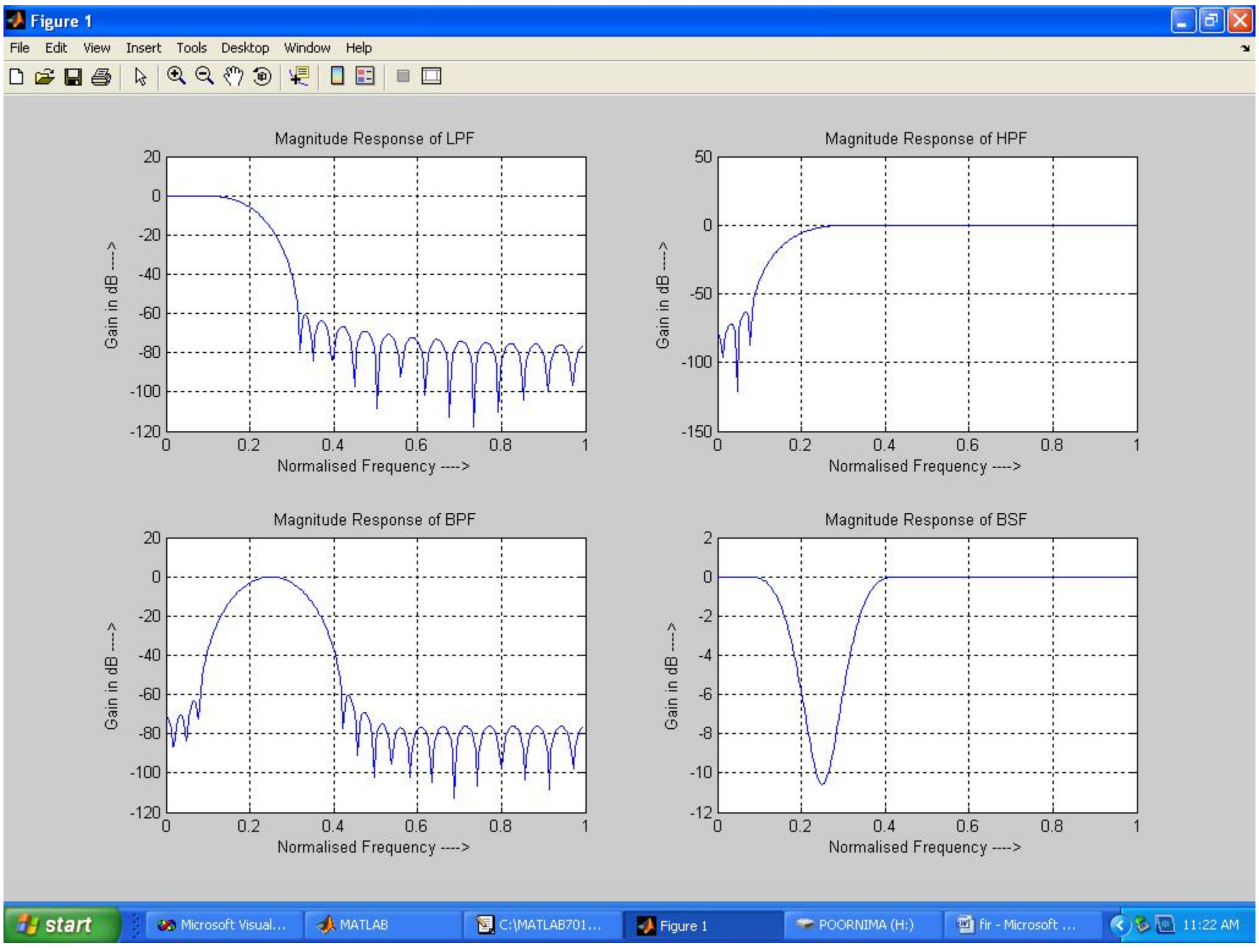
```
beta = input('Enter the beta value = ');
wp = 2*fp/f;
ws = 2*fs/f;
num = -20*log10(sqrt(rp*rs))-13;
dem = 14.6*(fs-fp)/f;
n = ceil(num/dem);
n1 = n+1;
if (rem(n,2)~=0)
n1 = n;
n = n-1;
end
y = kaiser(n1,beta);
% low-pass filter
b = fir1(n,wp,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title('Magnitude Response of LPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of HPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% band pass filter
wn = [wp ws];
b = fir1(n,wn,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
title('Magnitude Response of BPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of BSF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.02
Enter the stopband ripple = 0.01
Enter the passband frequency = 1000
Enter the stopband frequency = 1500
Enter the sampling frequency = 10000
Enter the beta value = 5.8
```

Waveform:



RESULT: FIR Filter (LP/HP/BP/BR) Using Windowing technique has been designed i) Rectangular window ii) Triangular window iii) Kaiser window and their magnitude responses are visualized.

Experiment-6. FIR FILTER DESIGN

AIM: To design FIR Filter (LP/HP/BP/BR) Using Windowing technique

a) Hamming window

b) Hanning window

c) Blackmann window

EQUIPMENTS:

Operating System	– Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY:

A Finite Impulse Response (FIR) filter is a discrete linear time-invariant system whose output is based on the weighted summation of a finite number of past inputs. An FIR transversal filter structure can be obtained directly from the equation for discrete-time convolution.

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

In this equation, x(k) and y(n) represent the input to and output from the filter at time n. h(n-k) is the transversal filter coefficients at time n. These coefficients are generated by using FDS (Filter Design Software or Digital filter design package).

FIR – filter is a finite impulse response filter. Order of the filter should be specified. Infinite response is truncated to get finite impulse response. placing a window of finite length does this. Types of windows available are Rectangular, Barlett, Hamming, Hanning, Blackmann window etc. This FIR filter is an all zero filter.

In this method, the desired frequency response specification $H_d(w)$, corresponding unit sample response $h_d(n)$ is determined using the following relation

$$h_d(n) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(w) e^{jwn} dw$$

$$H_d(w) = \sum_{n=-\infty}^{\infty} h_d(n) e^{-jwn}$$

In general, unit sample response $h_d(n)$ obtained from the above relation is infinite in duration, so it must be truncated at some point say $n= M$-1 to yield an FIR filter of length M (i.e. 0 to M-1). This truncation of $h_d(n)$ to length M-1 is same as multiplying $h_d(n)$ by the rectangular window defined as

$$w(n) = \begin{cases} 1 & 0 \le n \le M\text{-}1 \\ 0 & \text{otherwise} \end{cases}$$

Thus the unit sample response of the FIR filter becomes

$$\begin{aligned} h(n) &= h_d(n)\, w(n) \\ &= h_d(n) \qquad 0 \le n \le M\text{-}1 \\ &= 0 \qquad \text{otherwise} \end{aligned}$$

Now, the multiplication of the window function *w*(*n*) with $h_d(n)$ is equivalent to convolution of $H_d(w)$ with *W*(*w*), where *W*(*w*) is the frequency domain representation of the window function

$$W(w) = \sum_{n=0}^{M-1} w(n) e^{-jwn}$$

Thus the convolution of $H_d(w)$ with *W*(*w*) yields the frequency response of the truncated FIR filter

$$H(w) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_d(v) W(w-v) dw$$

The frequency response can also be obtained using the following relation

$$H(w) = \sum_{n=0}^{M-1} h(n) e^{-jwn}$$

But direct truncation of $h_d(n)$ to *M* terms to obtain *h*(*n*) leads to the Gibbs phenomenon effect which manifests itself as a fixed percentage overshoot and ripple before and after an approximated discontinuity in the frequency response due to the non-uniform convergence of the fourier series at a discontinuity. In order to reduce the ripples, instead of multiplying $h_d(n)$ with a rectangular window *w*(*n*), $h_d(n)$ is multiplied with a window function that contains a taper and decays toward zero gradually, instead of abruptly as it occurs in a rectangular window. As multiplication of sequences $h_d(n)$ and *w*(*n*) in time domain is equivalent to convolution of $H_d(w)$ and *W*(*w*) in the frequency domain, it has the effect of smoothing $H_d(w)$.

The several effects of windowing the Fourier coefficients of the filter on the result of the frequency response of the filter are as follows:

(i) A major effect is that discontinuities in *H*(*w*) become transition bands between values on either side of the discontinuity.

(ii) The width of the transition bands depends on the width of the main lobe of the frequency response of the window function, *w*(*n*) i.e. *W*(*w*).

(iii) Since the filter frequency response is obtained via a convolution relation , it is clear that the resulting filters are never optimal in any sense.

(iv) As *M* (the length of the window function) increases, the mainlobe width of *W*(*w*) is reduced which reduces the width of the transition band, but this also introduces more ripple in the frequency response.

(v) The window function eliminates the ringing effects at the bandedge and does result in lower sidelobes at the expense of an increase in the width of the transition band of the filter.

Some of the windows commonly used are as follows:

1. Bartlett triangular window:

$$W(n) = \frac{2(n+1)}{N+1} \qquad n = 0,1,2,\ldots\ldots,(N\text{-}1)/2$$

$$= 2 - \frac{2(n+1)}{N+1} \qquad n = (N\text{-}1)/2,\ldots\ldots,N\text{-}1$$

$$= 0\,, \qquad \text{otherwise}$$

2-5. Generalized cosine windows
(Rectangular, Hanning, Hamming and Blackman)

$$W(n) = a - b\cos(2p(n+1)/(N+1)) + c\cos(4p(n+1)/(N+1)) \qquad n= 0,1\ldots N\text{-}1$$

$$= 0 \qquad \text{otherwise}$$

6. Kaiser window with parameter ß :

$$W(n) = \frac{I_o(\beta\sqrt{1-(2(n+1)/(N+1))^2})}{I_o(\beta)} \qquad n= 0,1,\ldots,N\text{-}1$$

$$= 0 \qquad \text{otherwise}$$

The Bartlett window reduces the overshoot in the designed filter but spreads the transition region considerably. The Hanning, Hamming and Blackman windows use progressively more complicated cosine functions to provide a smooth truncation of the ideal impulse response and a frequency response that looks better. The best window results probably come from using the Kaiser window, which has a parameter ß that allows adjustment of the compromise between the overshoot reduction and transition region width spreading.

The major advantages of using window method is their relative simplicity as compared to other methods and ease of use. The fact that well defined equations are often available for calculating the window coefficients has made this method successful.

There are following problems in filter design using window method:

(i) This method is applicable only if $H_d(w)$ is absolutely integrable . When $H_d(w)$ is complicated or cannot easily be put into a closed form mathematical expression, evaluation of $h_d(n)$ becomes difficult.

(ii) The use of windows offers very little design flexibility e.g. in low pass filter design, the passband edge frequency generally cannot be specified exactly since the window smears the discontinuity in frequency. Thus the ideal LPF with cut-off frequency fc, is smeared by the window to give a frequency response with passband response with passband cutoff frequency f_1 and stopband cut-off frequency f_2.

(iii) Window method is basically useful for design of prototype filters like lowpass,highpass,bandpass etc. This makes its use in speech and image processing applications very limited.

6.1 FIR Filters Using Hamming Window

Program:

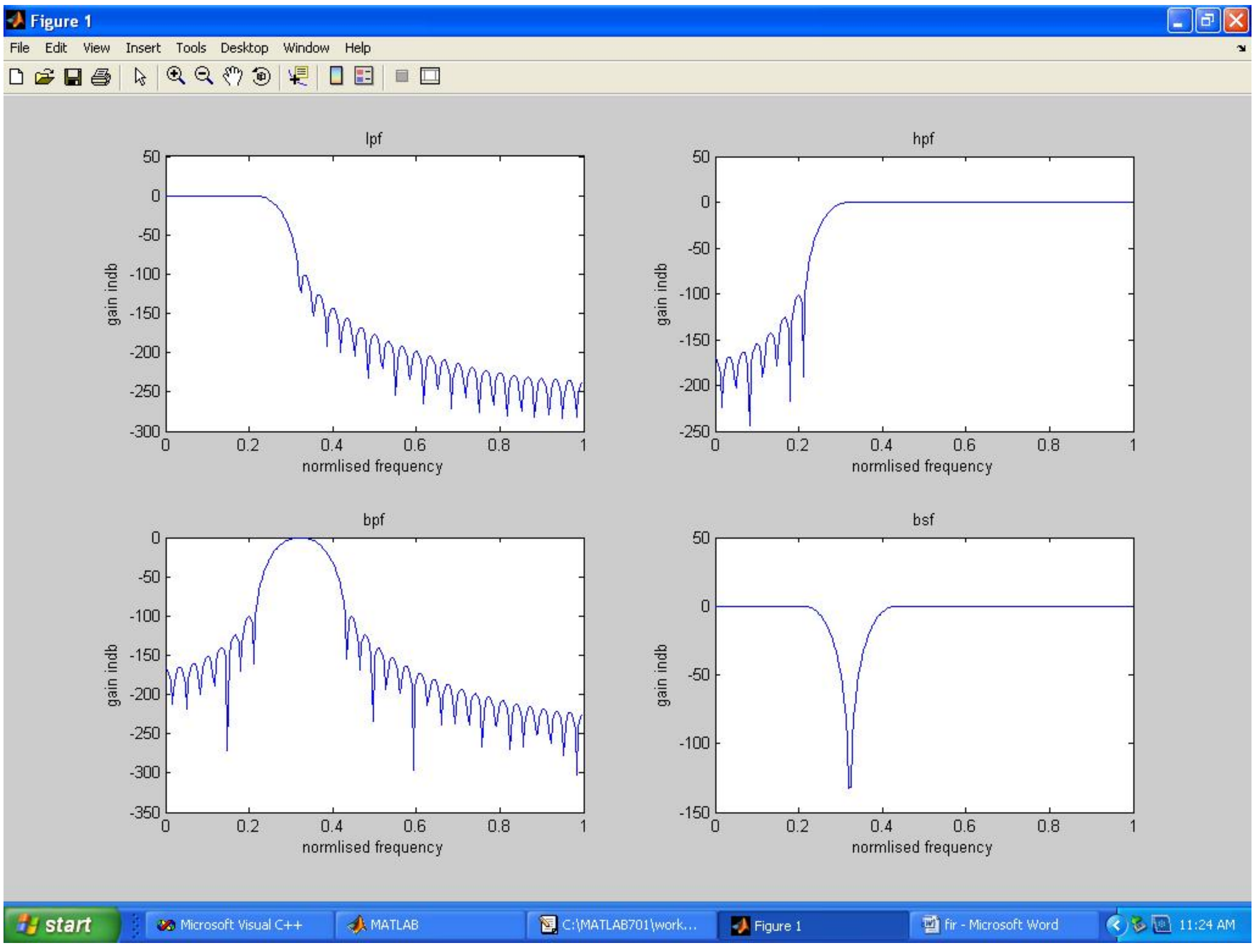
```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
fp = input('Enter the passband frequency = ');
fs = input('Enter the stopband frequency = ');
f = input('Enter the sampling frequency = ');
wp = 2*fp/f;
ws = 2*fs/f;
num = -20*log10(sqrt(rp*rs))-13;
dem = 14.6*(fs-fp)/f;
n = ceil(num/dem);
n1 = n+1;
if (rem(n,2)==50)
n1 = n;
n = n-1;
end
y = hamming(n1);
% low-pass filter
b = fir1(n,wp,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title('Magnitude Response of LPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of HPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% band pass filter
wn = [wp ws];
b = fir1(n,wn,y);
[h,o] = freqz(b,1,256);
```

```
m = 20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
title('Magnitude Response of BPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of BSF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.02
Enter the stopband ripple = 0.01
Enter the passband frequency = 1200
Enter the stopband frequency = 1700
Enter the sampling frequency = 9000
```

Waveform:



6.2 FIR Filters Using Hanning Window

Program:

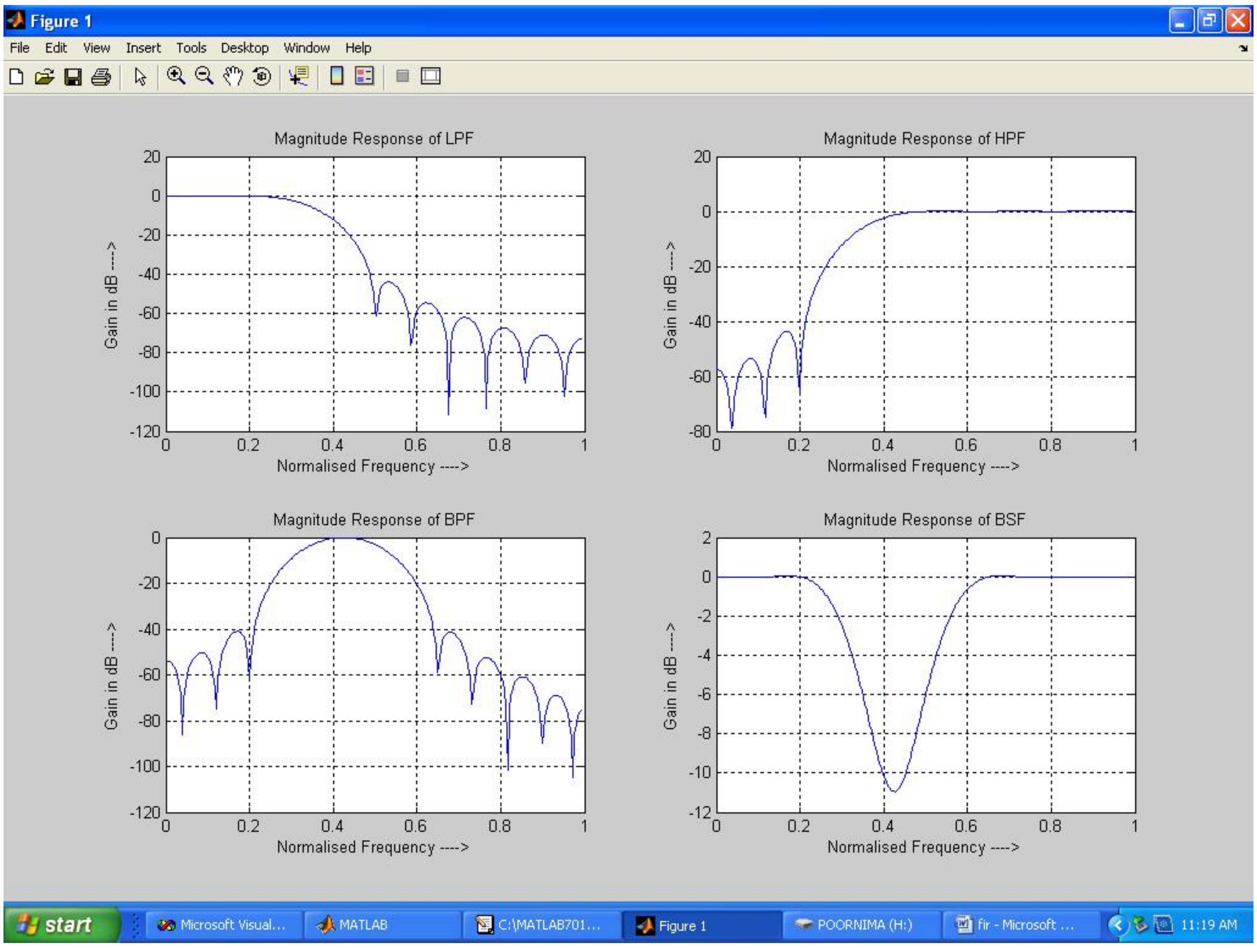
```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
fp = input('Enter the passband frequency = ');
fs = input('Enter the stopband frequency = ');
f = input('Enter the sampling frequency = ');
wp = 2*fp/f;
ws = 2*fs/f;
num = -20*log10(sqrt(rp*rs))-13;
dem = 14.6*(fs-fp)/f;
n = ceil(num/dem);
n1 = n+1;
if (rem(n,2)~=0)
n1 = n;
n = n-1;
end
y = hanning(n1);
% low-pass filter
```

```
b = fir1(n,wp,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title('Magnitude Response of LPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of HPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% band pass filter
wn = [wp ws];
b = fir1(n,wn,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
title('Magnitude Response of BPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of BSF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.03
Enter the stopband ripple = 0.01
Enter the passband frequency = 1400
Enter the stopband frequency = 2000
Enter the sampling frequency = 8000
```

Waveform:



6.3 FIR Filters Using Blackman Window

Program:

```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
fp = input('Enter the passband frequency = ');
fs = input('Enter the stopband frequency = ');
f = input('Enter the sampling frequency = ');
wp = 2*fp/f;
ws = 2*fs/f;
num = -20*log10(sqrt(rp*rs))-13;
dem = 14.6*(fs-fp)/f;
n = ceil(num/dem);
n1 = n+1;
if (rem(n,2)==50)
n1 = n;
n = n-1;
```

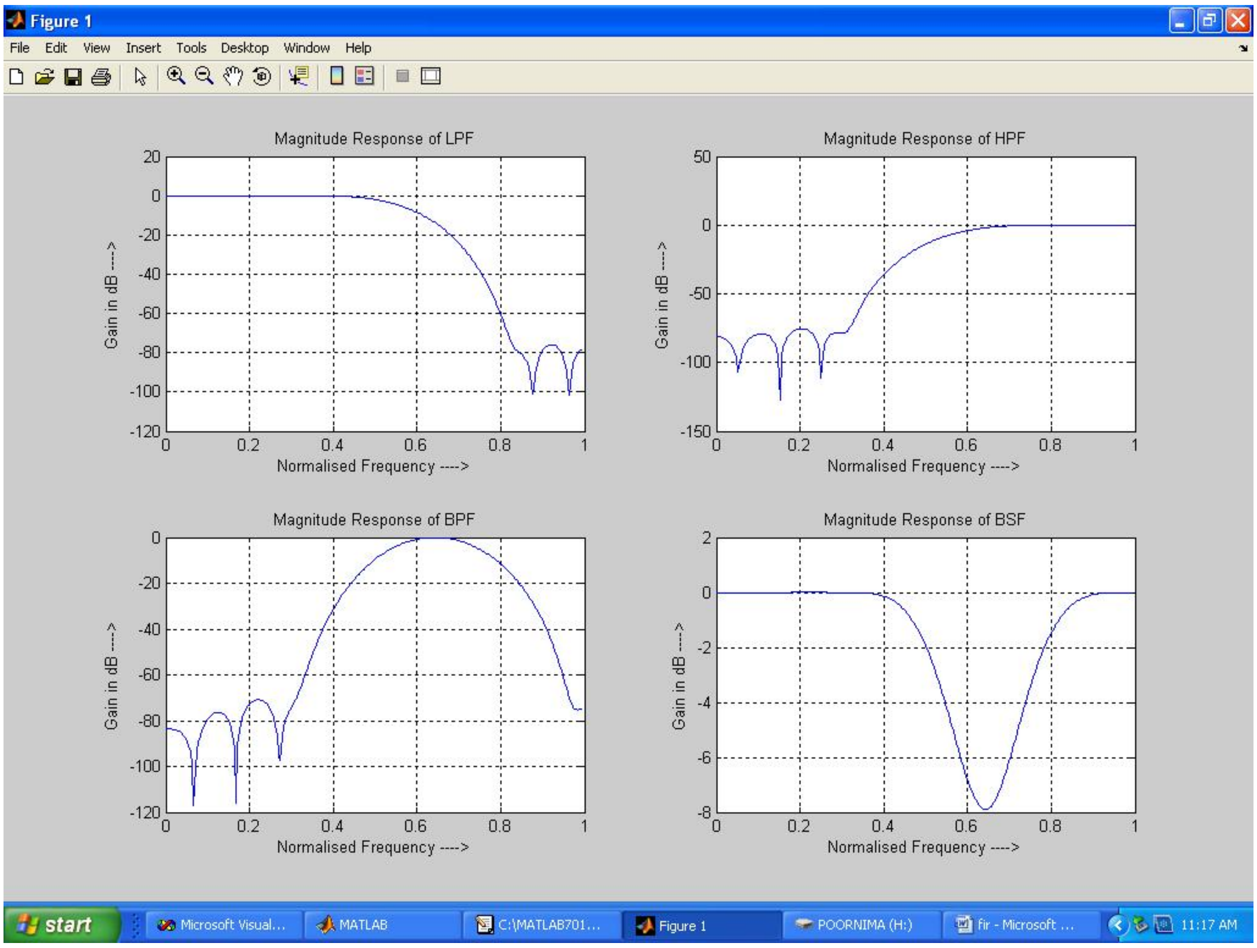
```
end
y = blackman(n1);
% low-pass filter
b = fir1(n,wp,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,1);
plot(o/pi,m);
title('Magnitude Response of LPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% high-pass filter
b = fir1(n,wp,'high',y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
title('Magnitude Response of HPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% band pass filter
wn = [wp ws];
b = fir1(n,wn,y);
[h,o] = freqz(b,1,256);
m = 20*log10(abs(h));
subplot(2,2,3);
plot(o/pi,m);
title('Magnitude Response of BPF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
% 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);
title('Magnitude Response of BSF');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.03
Enter the stopband ripple = 0.01
```

Enter the passband frequency = 2000
Enter the stopband frequency = 2500
Enter the sampling frequency = 7000

Waveform:



RESULT: FIR Filter (LP/HP/BP/BR) Using Windowing technique has been designed i) Blackmann window ii) Hamming window iii) Hanning window and their magnitude responses are visualized.

7. FREQUENCY RESPONSE OF ANALOG BUTTERWORTH PROTOTYPE LP/HP/BP/BR FILTERS

AIM: To find the frequency response of analog LP/HP/BP/BR filters using matlab.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	**-** Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY

BUTTERWORTH FILTER

The Butterworth filter has a maximally flat response that is, no passband ripple and roll-off of minus 20db per pole. It is "flat maximally magnitude" filters at the frequency of jω= 0, as the first 2N - 1 derivatives of the transfer function when jω = 0 are equal to zero [5]. The phase response of the Butterworth filter becomes more nonlinear with increasing N. This filter is completely defined mathematically by two parameters; they are cut off frequency and number of poles. The magnitude squared response of low pass Butterworth filter is given by,

$$H(j\omega) = \frac{1}{1+\left(\frac{\omega}{\omega_c}\right)^{2N}}$$

Filter Selectivity,

$$F_s = \frac{N}{2\sqrt{2}\omega_c}$$

Attenuation,

$$A=10\log(1+(\frac{\omega}{\omega_c})^{2N})$$

The frequency response of the Butterworth filter is maximally flat in the passband and rolls off towards zero in the stopband . When observed on a logarithmic bode plot the response slopes off linearly towards negative infinity. A first-order filter's response rolls off at −6 dB per octave (−20 dB per decade). A second-order filter‟s response rolls off at −12 dB per octave and a third-order at −18 dB. Butterworth filters have a monotonically varying magnitude function with ω, unlike other filter types that have non-monotonic ripple in the passband and the stopband.

Compared with a Chebyshev Type I filter or an Elliptic filter, the Butterworth filter has a slower roll-off and therefore will require a higher order to implement a particular stopband specification. Butterworth filters have a more linear phase response in the pass-band than Chebyshev Type I and Elliptic filters .

The Butterworth filter rolls off more slowly around the cut off frequency than the Chebyshev Type I and Elliptic filters without ripple. All of these filters are in fifth order. Fig.2:Phase response of the various types of IIR filters

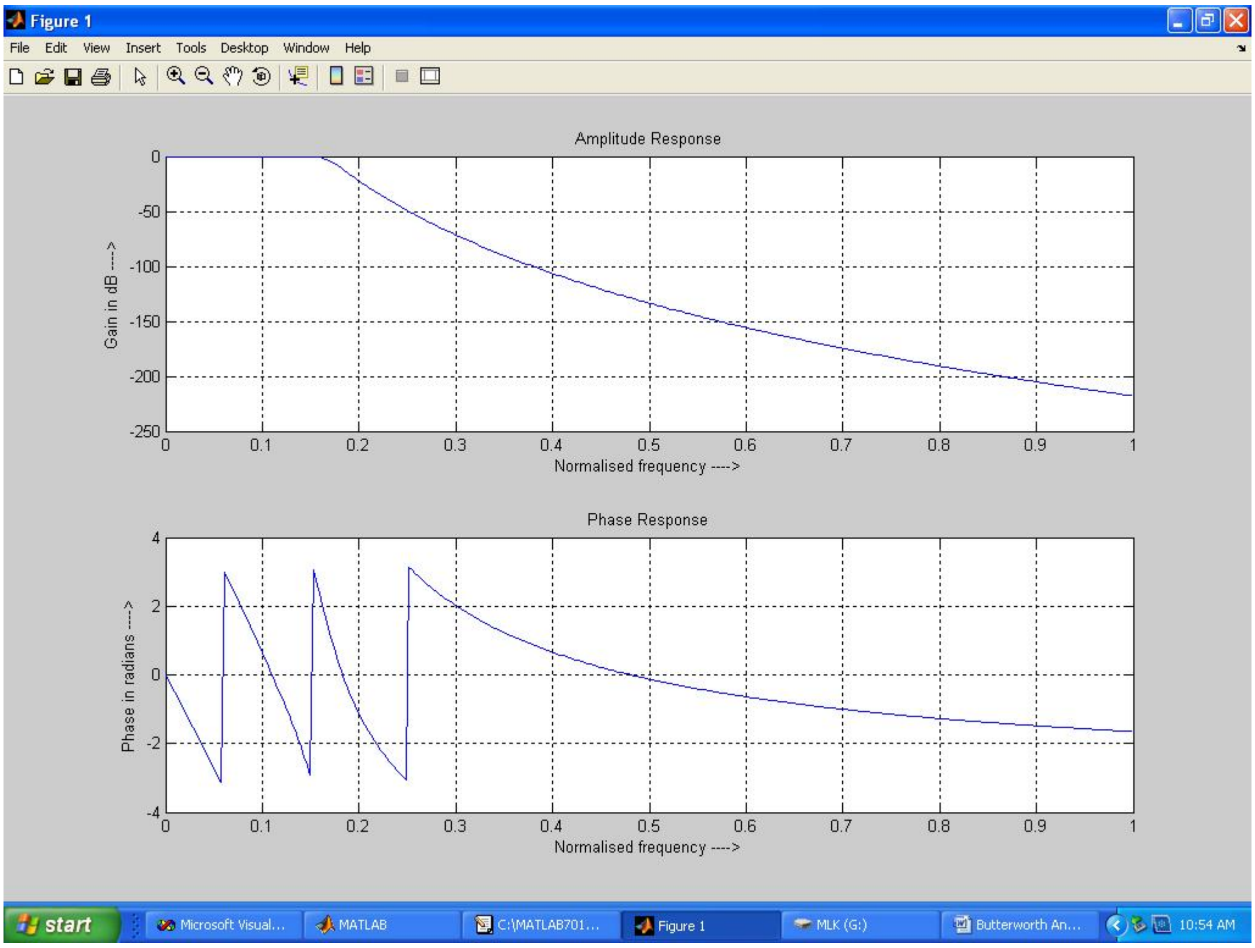
7.1 Butterworth Analog Low Pass Filter

Program

```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n,wn] = buttord(w1,w2,rp,rs,'s');
[z,p,k] = butter(n,wn);
[b,a] = zp2tf(z,p,k);
[b,a] = butter(n,wn,'s');
w = 0:0.01:pi;
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
title('Amplitude Response');
ylabel('Gain in dB---- >');
xlabel('Normalised frequency---- >');
grid on;
subplot(2,1,2);
plot(om/pi,an);
title('Phase Response');
xlabel('Normalised frequency---- >');
ylabel('Phase in radians---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.15
Enter the stopband ripple = 60
Enter the passband frequency = 1500
Enter the stopband frequency = 3000
Enter the sampling frequency = 7000
```



7.2 Butterworth Analog High Pass Filter

Program:

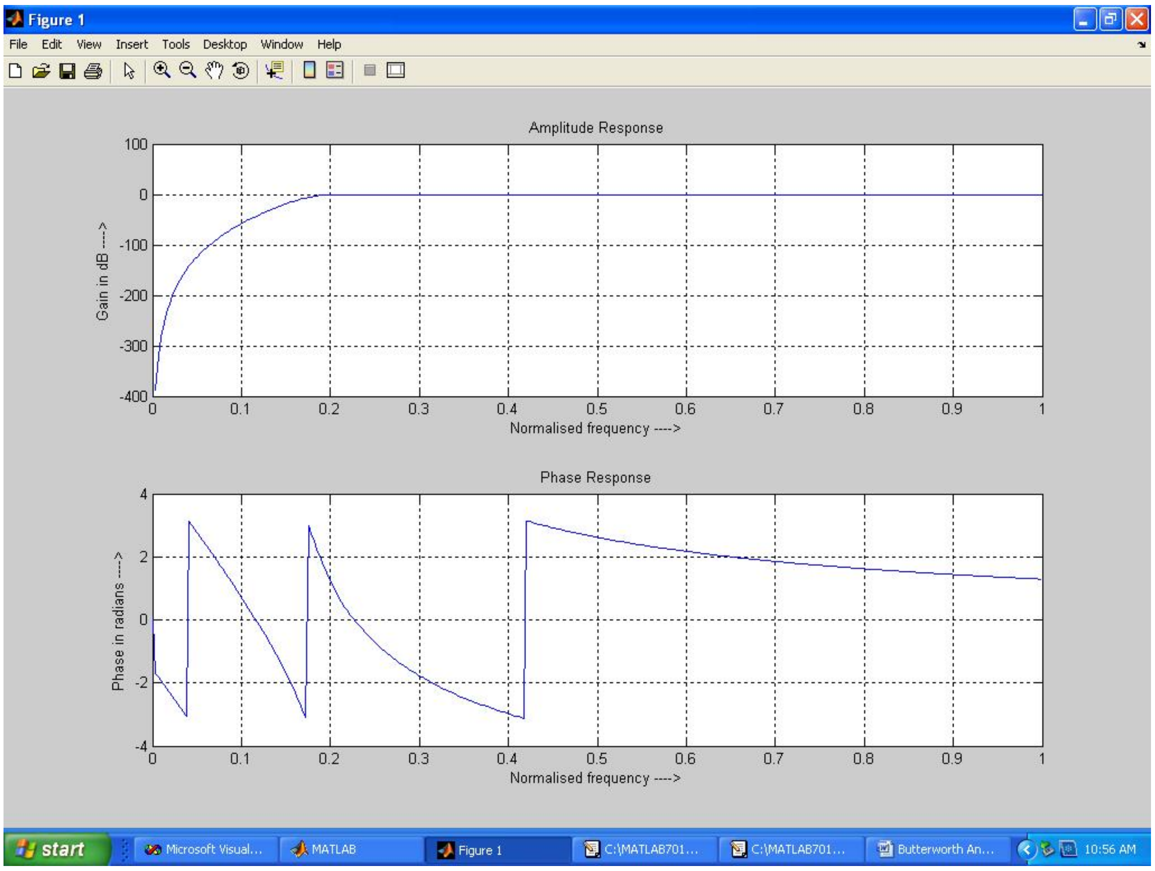
```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n,wn] = buttord(w1,w2,rp,rs,'s');
[b,a] = butter(n,wn,'high','s');
w = 0:0.01:pi;
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
ylabel('Gain in dB---- >');
```

```
xlabel('Normalised frequency---- >');
title('Amplitude Response');
grid on;
subplot(2,1,2);
plot(om/pi,an);
xlabel('Normalised frequency---- >');
ylabel('Phase in radians---- >');
title('Phase Response');
grid on;
```

Output:

```
Enter the passband ripple = 0.2
Enter the stopband ripple = 40
Enter the passband frequency = 2000
Enter the stopband frequency = 3500
Enter the sampling frequency = 8000
```

Waveform:



7.3 Butterworth Analog Band Pass Filter

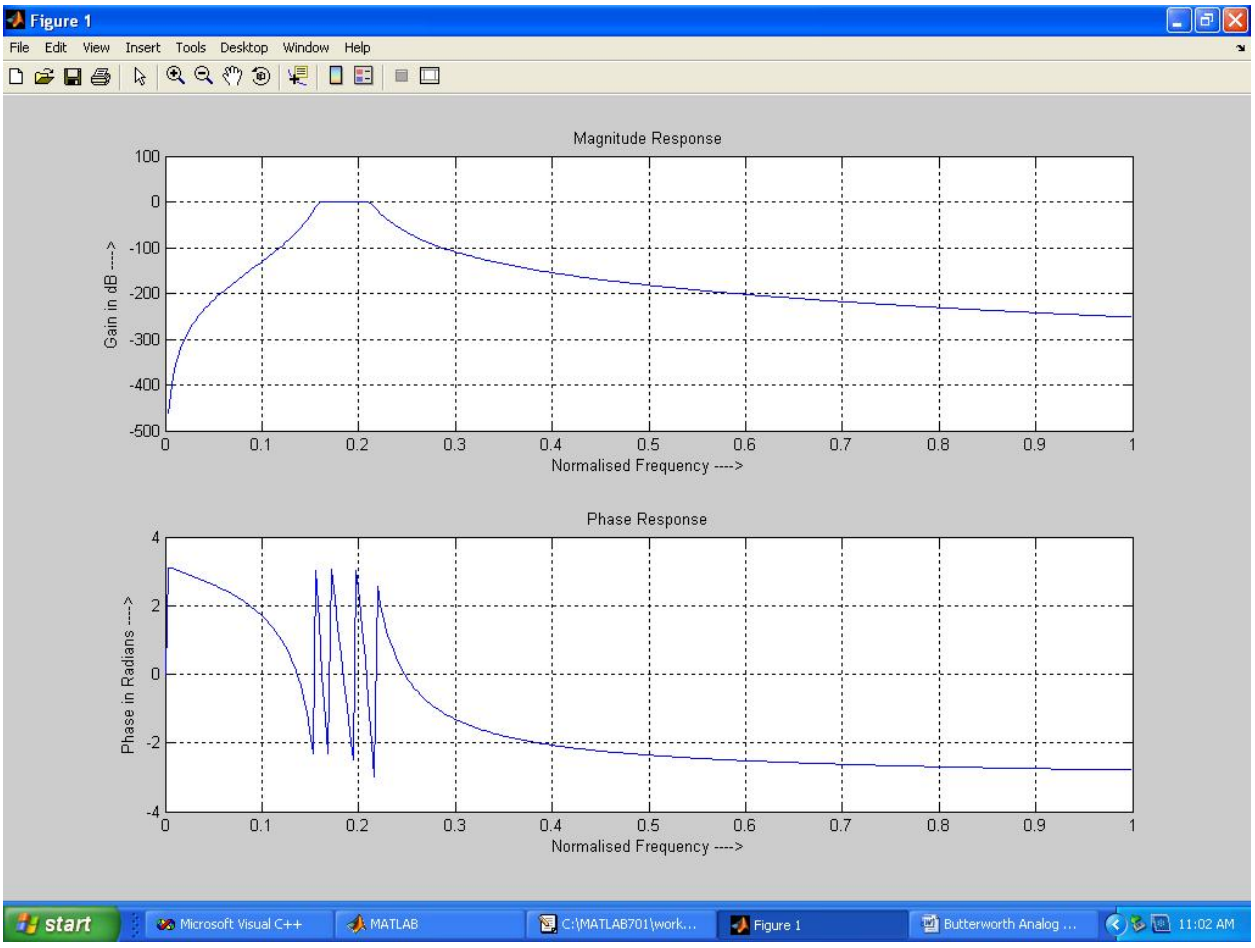
Program:

```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n] = buttord(w1,w2,rp,rs);
wn = [w1 w2];
[b,a] = butter(n,wn,'bandpass','s');
w = 0:.01:pi;
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
title('Magnitude Response');
grid on;
subplot(2,1,2);
plot(om/pi,an);
xlabel('Normalised Frequency---- >');
ylabel('Phase in Radians---- >');
title('Phase Response');
grid on;
```

Output:

```
Enter the passband ripple = 0.36
Enter the stopband ripple = 36
Enter the passband frequency = 1500
Enter the stopband frequency = 2000
Enter the sampling frequency = 6000
```

Waveform:



7.4 Butterworth Analog Band Stop Filter

Program:

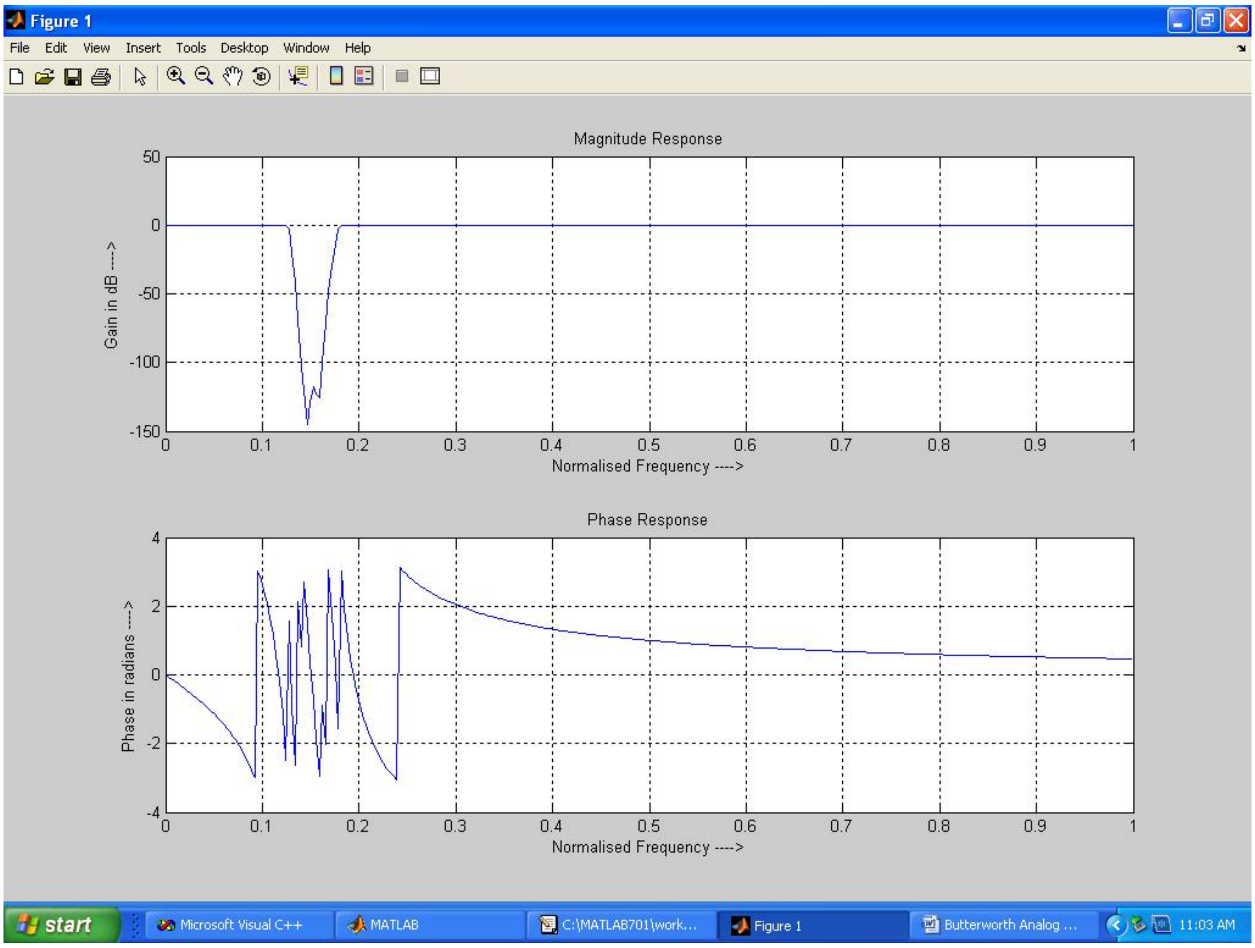
```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n] = buttord(w1,w2,rp,rs,'s');
wn = [w1 w2];
[b,a] = butter(n,wn,'stop','s');
w = 0:0.01:pi;
```

```
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
title('Magnitude Response');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
subplot(2,1,2);
plot(om/pi,an);
title('Phase Response');
xlabel('Normalised Frequency---- >');
ylabel('Phase in radians---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.28
Enter the stopband ripple = 28
Enter the passband frequency = 1000
Enter the stopband frequency = 1400
Enter the sampling frequency = 5000
```

Waveform:



RESULTS: Hence Butterworth analog (LP/HP/BP/BR) proto type filters are implemented and their magnitude responses are visualized.

Experiment-8. FREQUENCY RESPONSE OF ANALOG CHEBYSHEV LP/HP/BP/BR FILTERS

AIM: To find the frequency response of analog LP/HP filters using matlab.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY

CHEBYSHEV TYPE -I FILTER The absolute difference between the ideal and actual frequency response over the entire passband is minimized by Chebyshev Type I filter by incorporating equal ripple in the passband. Stopband response is maximally flat. The transition from passband to stopband is more rapid than for the Butterworth filter. The magnitude squared Chebyshev type I response is:

$$H(j\omega) = \frac{1}{1+\varepsilon^2 C_N^2(\frac{\omega}{\omega_p})}$$

Where

$$C_N\left(\frac{\omega}{\omega_p}\right) = \left\{\begin{matrix} \cos\left[N\cos^{-1}\left(\frac{\omega}{\omega_p}\right)\right], |\omega| \le \omega_p \\ \cosh[N\cosh^{-1}\left(\frac{\omega}{\omega_p}\right)], |\omega| \ge \omega_p \end{matrix}\right\}$$

The magnitude squared response peaks occur in the pass band when

$$C_N\left(\frac{\omega}{\omega_p}\right) = 0.$$

The ripple is often given in dB:

$$\text{Ripple}=20\log_{10}\sqrt{1+\varepsilon^2}$$

9.1 Chebyshev Type 1 Analog Low Pass Filter

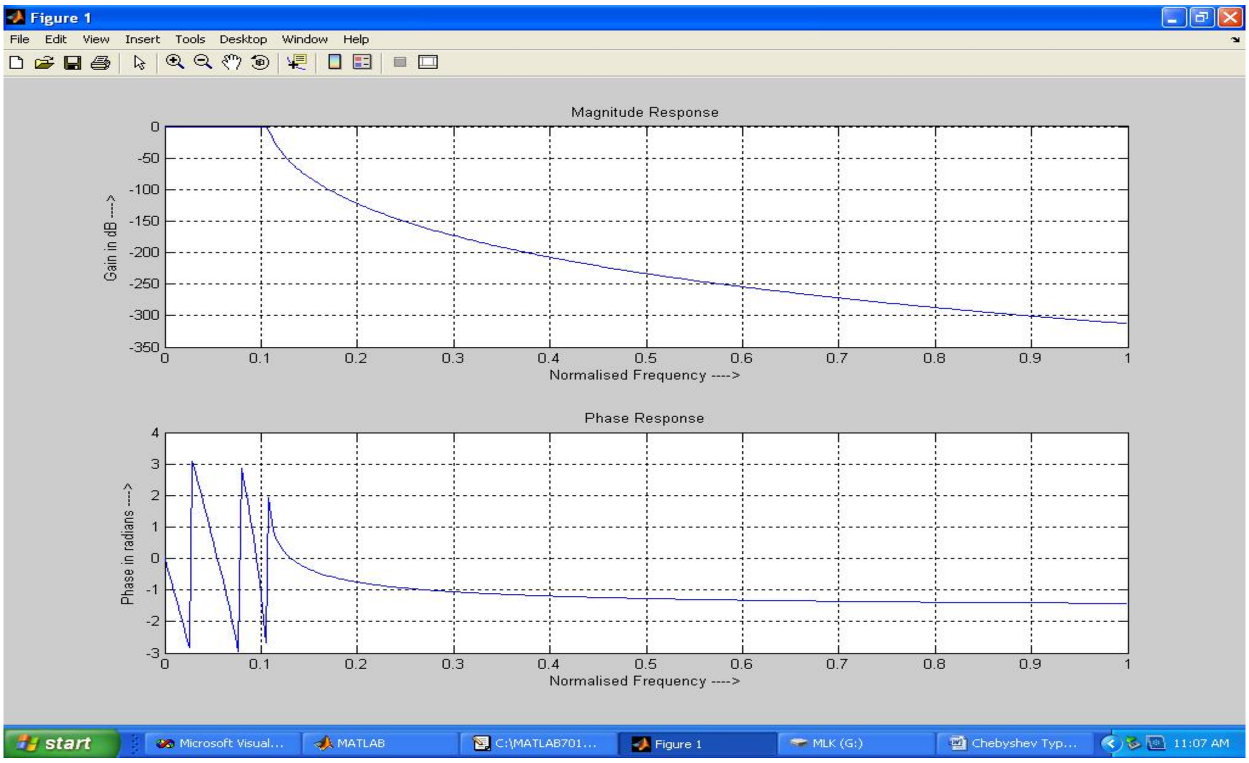
Program:
```
clc;
clear all;
rp = input('Enter the passband ripple = ');
```

```
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n,wn] = cheb1ord(w1,w2,rp,rs,'s');
[b,a] = cheby1(n,rp,wn,'s');
w = 0:0.01:pi;
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
title('Magnitude Response');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
subplot(2,1,2);
plot(om/pi,an);
title('Phase Response');
xlabel('Normalised Frequency---- >');
ylabel('Phase in radians---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.23
Enter the stopband ripple = 47
Enter the passband frequency = 1300
Enter the stopband frequency = 1550
Enter the sampling frequency = 7800
```

Waveform:



8.2 Chebyshev Type 1 Analog High Pass Filter

Program:

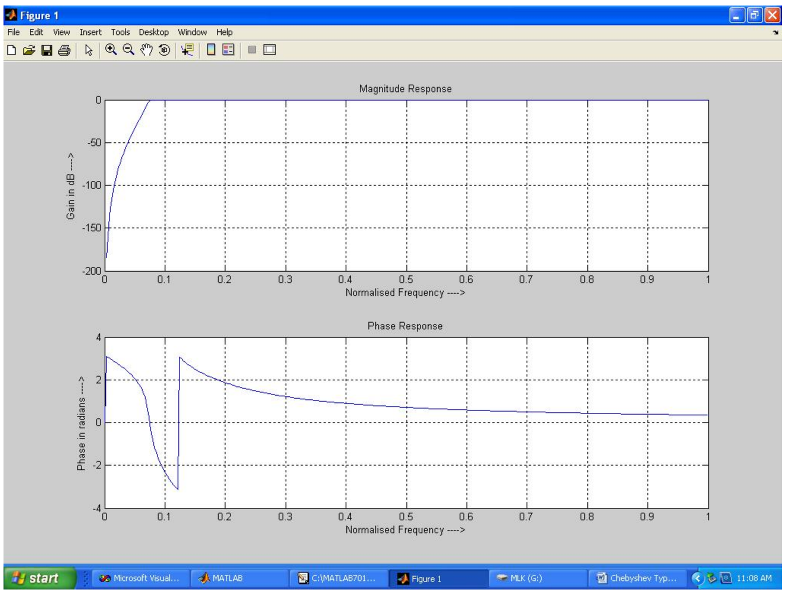
```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n,wn] = cheb1ord(w1,w2,rp,rs,'s');
[b,a] = cheby1(n,rp,wn,'high','s');
w = 0:0.01:pi;
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
subplot(2,1,1);
plot(om/pi,m);
title('Magnitude Response');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
```

```
subplot(2,1,2);
plot(om/pi,an);
title('Phase Response');
xlabel('Normalised Frequency---- >');
ylabel('Phase in radians---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.29
Enter the stopband ripple = 29
Enter the passband frequency = 900
Enter the stopband frequency = 1300
Enter the sampling frequency = 7500
```

Waveform:



8.3 Chebyshev Type 1 Analog Band Pass Filter

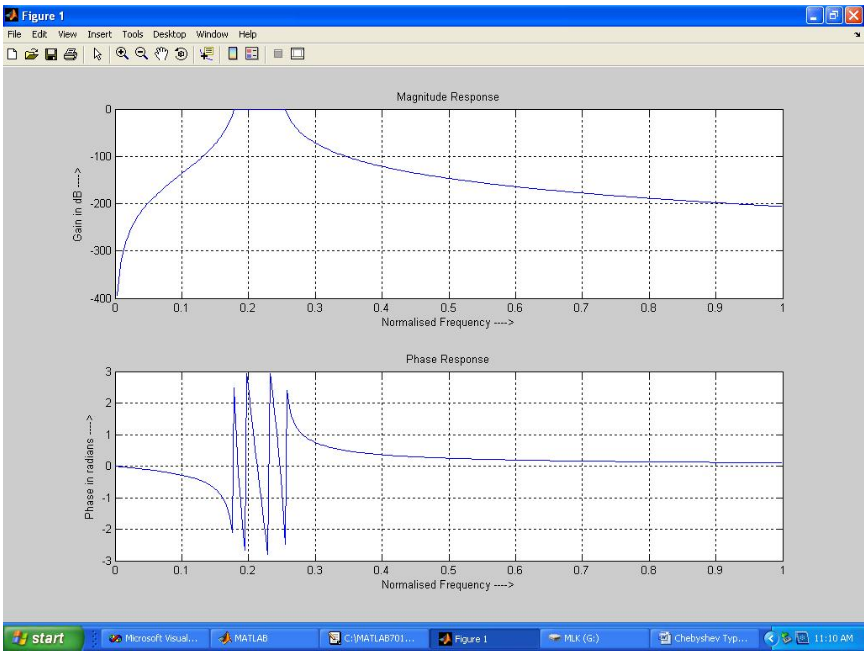
Program:

```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n] = cheb1ord(w1,w2,rp,rs,'s');
wn = [w1 w2];
[b,a] = cheby1(n,rp,wn,'bandpass','s');
w = 0:0.01:pi;
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
subplot(2,1,1);
plot(om/pi,m);
title('Magnitude Response');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
subplot(2,1,2);
plot(om/pi,an);
title('Phase Response');
xlabel('Normalised Frequency---- >');
ylabel('Phase in radians---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.3
Enter the stopband ripple = 40
Enter the passband frequency = 1400
Enter the stopband frequency = 2000
Enter the sampling frequency = 5000
```

Waveform:



8.4 Chebyshev Type 1 Analog Band Stop Filter

Program:

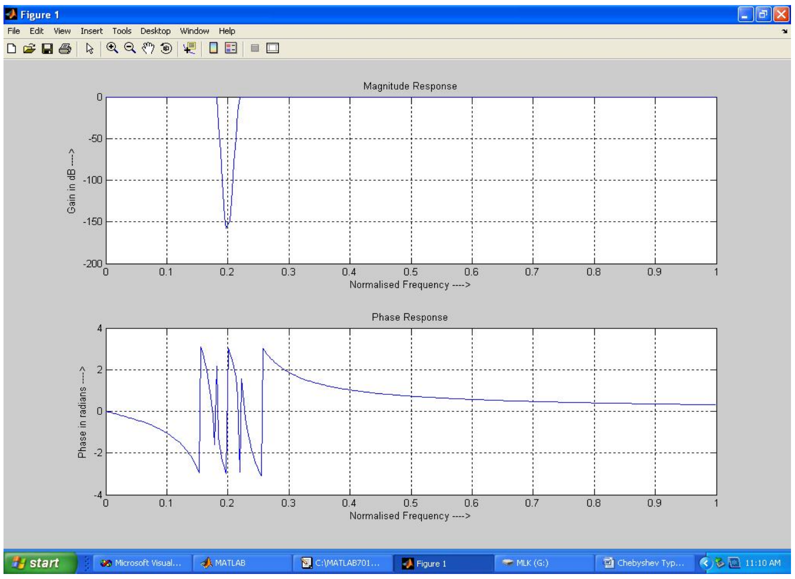
```
clc;
clear all;
rp = input('Enter the passband ripple = ');
rs = input('Enter the stopband ripple = ');
wp = input('Enter the passband frequency = ');
ws = input('Enter the stopband frequency = ');
fs = input('Enter the sampling frequency = ');
w1 = 2*wp/fs;
w2 = 2*ws/fs;
[n] = cheb1ord(w1,w2,rp,rs,'s');
wn = [w1 w2];
[b,a] = cheby1(n,rp,wn,'stop','s');
w = 0:0.01:pi;
[h,om] = freqs(b,a,w);
m = 20*log10(abs(h));
an = angle(h);
subplot(2,1,1);
plot(om/pi,m);
subplot(2,1,1);
plot(om/pi,m);
title('Magnitude Response');
ylabel('Gain in dB---- >');
xlabel('Normalised Frequency---- >');
grid on;
subplot(2,1,2);
plot(om/pi,an);
```

```
title('Phase Response');
xlabel('Normalised Frequency---- >');
ylabel('Phase in radians---- >');
grid on;
```

Output:

```
Enter the passband ripple = 0.15
Enter the stopband ripple = 30
Enter the passband frequency = 2000
Enter the stopband frequency = 2400
Enter the sampling frequency = 7000
```

Waveform:



RESULTS: Hence Butterworth and chebyshev analog (LP/HP/BP/BR) proto type filters are implemented and their magnitude responses are visualized.

Experiment-9. IIR FILTER DESIGN

AIM: To design and implement IIR (LPF/HPF) filters on DSP processors using CCS and also to implement in MATLAB.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY:

The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

$$H(z) \frac{\sum_{k=0}^{M} b_k z_{-k}}{\sum_{k=1}^{N} a_k Z^{-k}} \qquad (2)$$

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

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) \; + \; \sum_{k=1}^{N} a_k y(n-k) \qquad (3)$$

M and N are order of the two polynomials

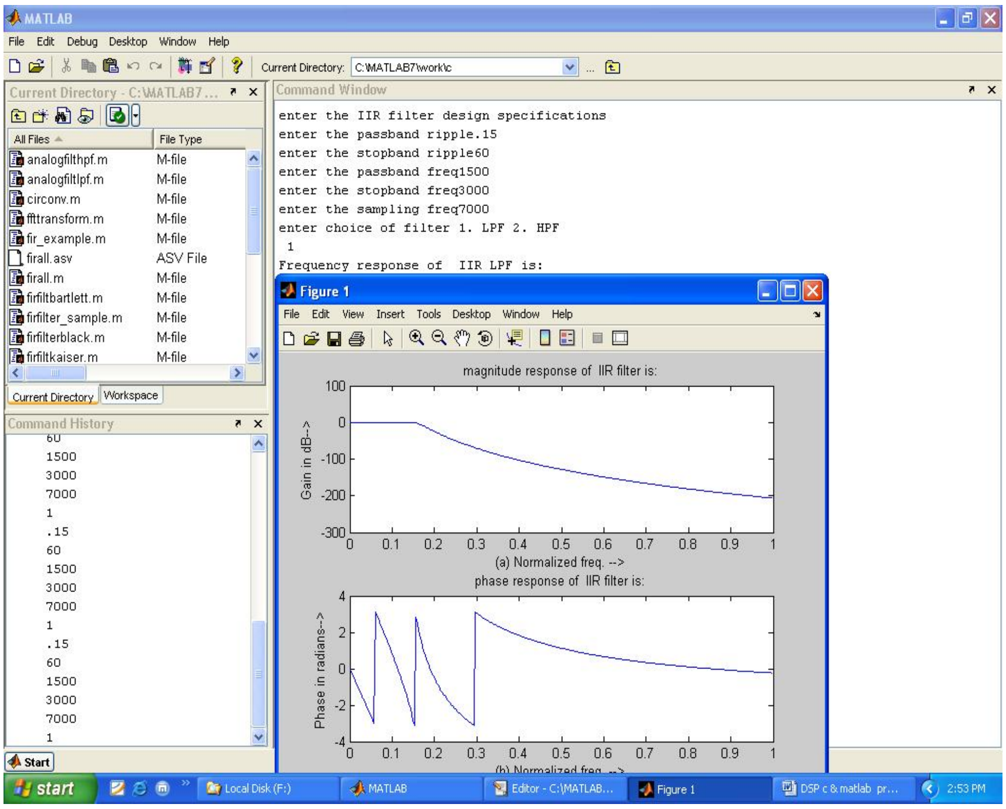
b_k and a_k are the filter coefficients. These filter coefficients are generated using FDS (Filter Design software or Digital Filter design package).

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That‟s why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

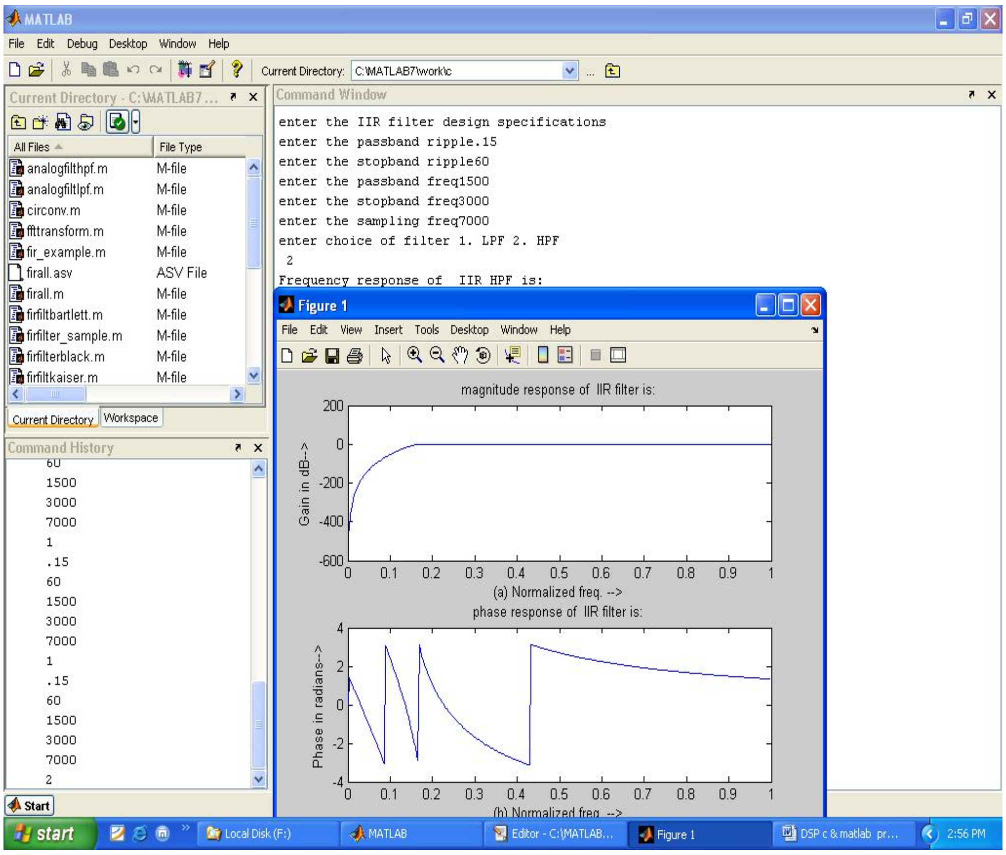
9.1 PROGRAM :

```
% IIR filters LPF & HPF
clc;
clear all;
```

```
close all;
disp('enter the IIR filter design specifications');
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband freq');
ws=input('enter the stopband freq');
fs=input('enter the sampling freq');
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs,'s');
c=input('enter choice of filter 1. LPF 2. HPF \n ');
if(c==1)
   disp('Frequency response of IIR LPF is:');
[b,a]=butter(n,wn,'low','s');
end
if(c==2)
   disp('Frequency response of IIR HPF is:');
[b,a]=butter(n,wn,'high','s');
end
w=0:.01:pi;
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure,subplot(2,1,1);plot(om/pi,m);
title('magnitude response of IIR filter is:');
xlabel('(a) Normalized freq. -->');
ylabel('Gain in dB-->');
subplot(2,1,2);plot(om/pi,an);
title('phase response of IIR filter is:');
xlabel('(b) Normalized freq. -->');
ylabel('Phase in radians-->');
```



```
enter the IIR filter design specifications
enter the passband ripple.15
enter the stopband ripple60
enter the passband freq1500
enter the stopband freq3000
enter the sampling freq7000
enter choice of filter 1. LPF 2. HPF
 1
Frequency response of  IIR LPF is:
```



```
enter the IIR filter design specifications
enter the passband ripple.15
enter the stopband ripple60
enter the passband freq1500
enter the stopband freq3000
enter the sampling freq7000
enter choice of filter 1. LPF 2. HPF
 2
Frequency response of  IIR HPF is:
```

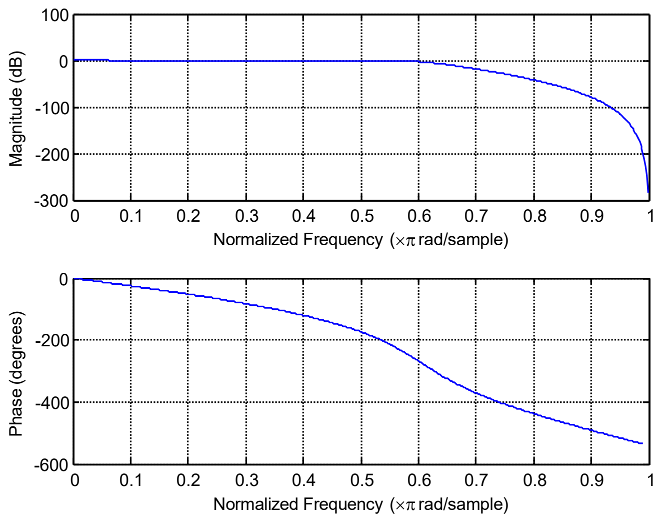
9.2 Program

```
clc;
clear;
ap=input('Enter PB attn in dB : '); as=input('Enter SB attn in dB : ');
fp=input('Enter PB freq in hz : '); fs=input('Enter SB freq in hz : ');
F=input('Enter the sampling freq : ');
wp=2*pi*fp/F;
ws=2*pi*fs/F;
wp=2*F*tan(wp/2);
ws=2*F*tan(ws/2);
[N,wc]=buttord(wp,ws,ap,as,'s')
c=input('enter choice of filter 1. LPF 2. HPF \n ');
if(c==1)
   disp('Frequency response of IIR LPF is:');
[b,a]=butter(n,wn,'low','s');
end
if(c==2)
   disp('Frequency response of IIR HPF is:');
[b,a]=butter(n,wn,'high','s');
end
%[b,a]=butter(N,wc,'s')
[bz,az]=bilinear(b,a,F)
freqz(bz,az)
```

command window

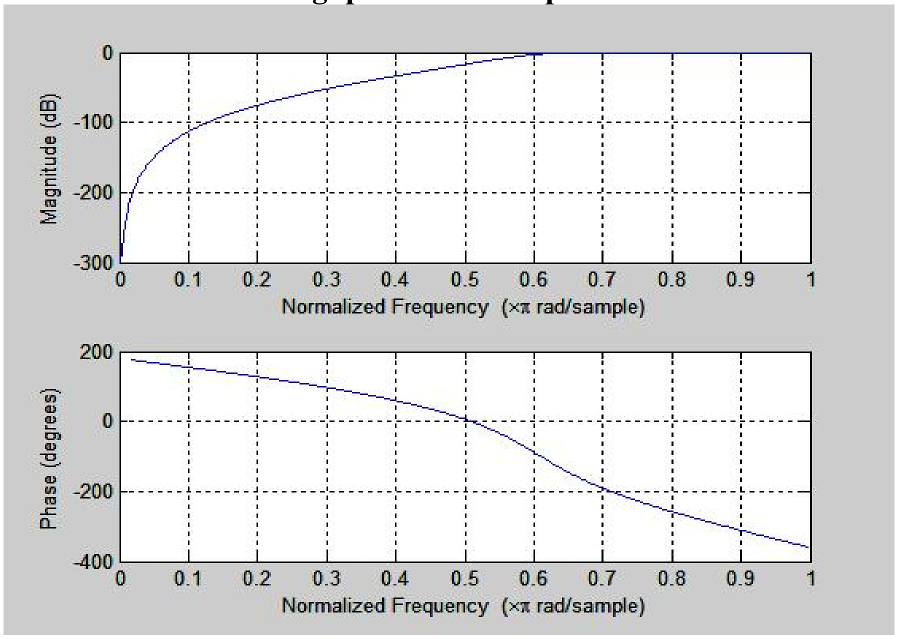
```
Enter PB attn in dB : 0.15
Enter SB attn in dB : 60
Enter PB freq in hz : 1500
Enter SB freq in hz : 3000
Enter the sampling freq : 7000
enter choice of filter 1. LPF 2. HPF : 1
```

Lowpass Filter Response



```
Enter PB attn in dB : 0.15
Enter SB attn in dB : 60
Enter PB freq in hz : 1500
Enter SB freq in hz : 3000
Enter the sampling freq : 7000
enter choice of filter 1. LPF 2. HPF : 1
```

Highpass Filter Response



9.3 IIR filter (LP/HP) Implementation on DSP Processors

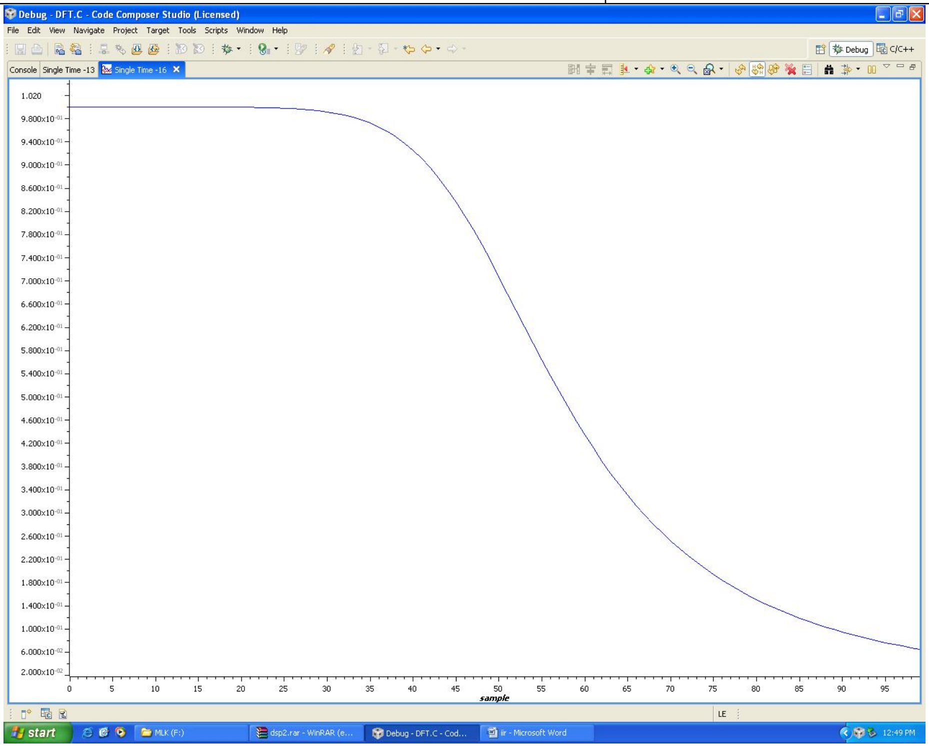
```
#include<stdio.h>
#include<math.h>
float h[100];
```

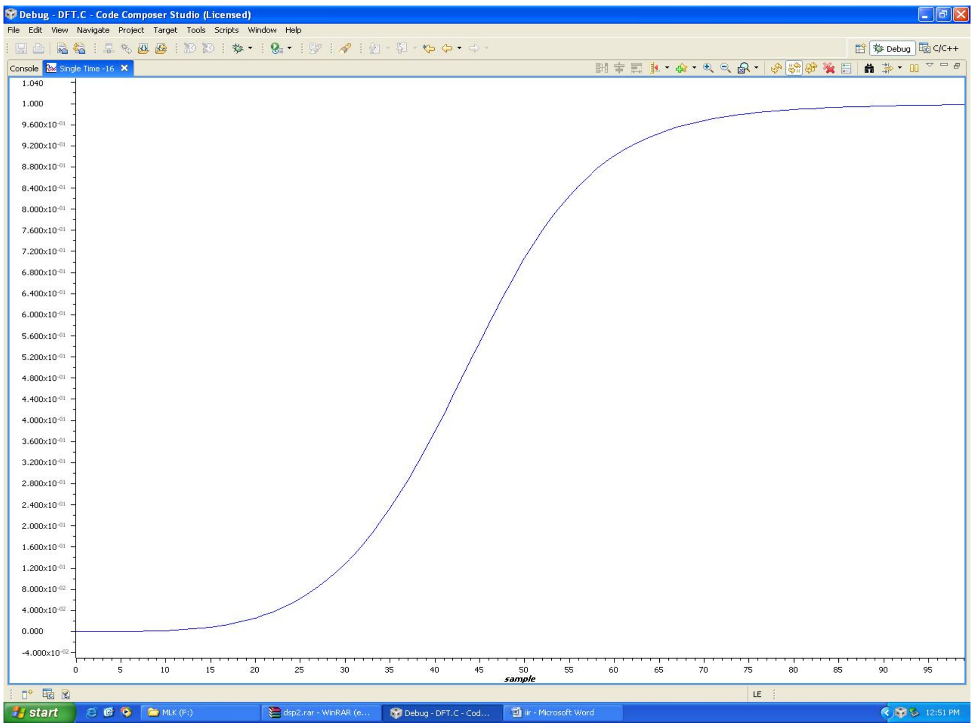
```
int n,w,c,i,wc;
float mul(float,int);
void main()
{
        printf("enter order of the filter");
        scanf("%d",&n);
        printf("enter wc");
        scanf("%d",&wc);
        printf("enter the choice 1.lpf 2.hpf");
        scanf("%d",&c);
        switch(c)
        {
                case 1:
                for(w=0;w<100;w++)
                {
                        h[w]=1/sqrt(1+mul((w/(float)wc),2*n));
                        printf("h[%d]=%f\n",w,h[w]);
                }
                break;
                case 2:
                for(w=0;w<100;w++)
                {
                                                        h[w]=1/sqrt(1+mul(((float)wc/w),2*n));
                        printf("h[%d]=%f\n",w,h[w]);
                }
                break;
        default:
                printf("enter correct choice");
                }
}
float mul(float a,int x)
{
        for(i=0;i<x-1;i++)
        {
                a*=a;
        }
        return(a);
}
```

Output

Hpf:	h[13]=0.004570
enter order of the filter 2	h[14]=0.006146
enter wc 50	h[15]=0.008100
enter the choice 1.lpf 2.hpf2	h[16]=0.010485
h[0]=0.000000	h[17]=0.013362
h[1]=0.000000	h[18]=0.016794
h[2]=0.000003	h[19]=0.020847
h[3]=0.000013	h[20]=0.025592
h[4]=0.000041	h[21]=0.031102
h[5]=0.000100	h[22]=0.037455
h[6]=0.000207	h[23]=0.044730
h[7]=0.000384	h[24]=0.053010
h[8]=0.000655	h[25]=0.062378
h[9]=0.001050	h[26]=0.072922
h[10]=0.001600	h[27]=0.084725
h[11]=0.002343	
h[12]=0.003318	
Lpf:	h[12]=0.999995

enter order of the filter2	h[13]=0.999990
enter wc50	h[14]=0.999981
enter the choice 1.lpf 2.hpf1	h[15]=0.999967
h[0]=1.000000	h[16]=0.999945
h[1]=1.000000	h[17]=0.999911
h[2]=1.000000	h[18]=0.999859
h[3]=1.000000	h[19]=0.999783
h[4]=1.000000	h[20]=0.999672
h[5]=1.000000	h[21]=0.999516
h[6]=1.000000	h[22]=0.999298
h[7]=1.000000	h[23]=0.998999
h[8]=1.000000	h[24]=0.998594
h[9]=0.999999	h[25]=0.998053
h[10]=0.999999	h[26]=0.997338
h[11]=0.999997	




Debug - DFT.C - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
Console
Single Time -16
Debug
C/C++
sample
start

RESULT: IIR Filter (LP/HP) has been implemented and their magnitude responses are visualized through DSP processor and MATLAB.

Experiment-10. IIR FILTER DESIGN

AIM: To design and implement IIR (BP/BR) filters on DSP processors using CCS and also to implement in MATLAB.

EQUIPMENTS:

Operating System	– Windows XP
Constructor	**-** Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY:

The IIR filter can realize both the poles and zeroes of a system because it has a rational transfer function, described by polynomials in z in both the numerator and the denominator:

$$H(z)\frac{\sum_{k=0}^{M} b_k z_{-k}}{\sum_{k=1}^{N} a_k Z^{-k}} \qquad (2)$$

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

$$y(n) = \sum_{k=0}^{M} b_k x(n-k) \ + \ \sum_{k=1}^{N} a_k y(n-k) \qquad (3)$$

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

IIR filters can be expanded as infinite impulse response filters. In designing IIR filters, cutoff frequencies of the filters should be mentioned. The order of the filter can be estimated using butter worth polynomial. That‟s why the filters are named as butter worth filters. Filter coefficients can be found and the response can be plotted.

10.1 IIR filter (BP/BR) Implementation on DSP Processors
Program

```
#include<stdio.h>
#include<math.h>
float h[100];
int n,w,c,i,wc,wl,wu,w1,w2;
```

```
float mul(float,int);
void main()
{
	printf("enter order of the filter");
	scanf("%d",&n);
	printf("enter wl");
	scanf("%d%d",&wl,&wu);
	printf("enter the choice 1.bpf 2.brf");
	scanf("%d",&c);
	w1=wu-wl;
	w2=wu*wl;
	switch(c)
	{
		case 1:
		for(w=0;w<100;w++)
		{
			h[w]=1/sqrt(1+mul(((w*w)-w2)/(float)(w1),2*n));
			printf("h[%d]=%f\n",w,h[w]);
		}
		break;
		case 2:
		for(w=0;w<100;w++)
		{
						h[w]=1/sqrt(1+mul((w*w1)/(float)((w*w)-
w2),2*n));
			printf("h[%d]=%f\n",w,h[w]);
		}
		break;
	default:
		printf("enter correct choice");
		}
}
float mul(float a,int x)
{
	for(i=0;i<x-1;i++)
	{
		a*=a;
	}
	return(a);
}
```

Output

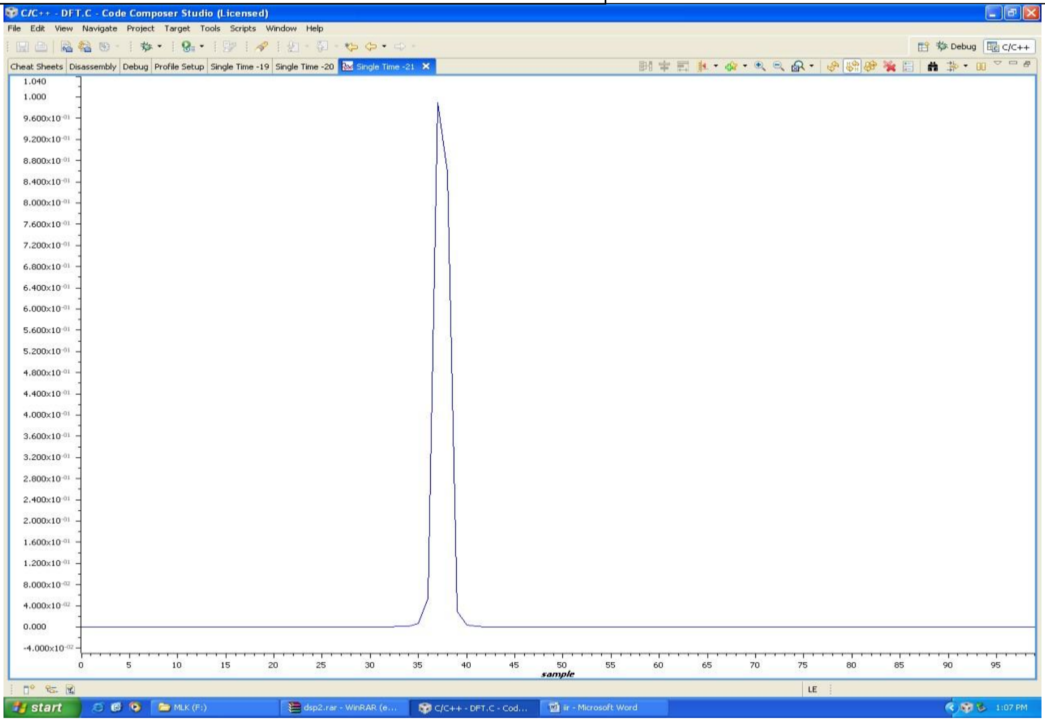
Bpf

enter order of the filter2
 	
enter wl,wu 20
 	
70
	h[25]=0.000017
	
	h[26]=0.000023
	
	h[27]=0.000031

```
enter the choice 1.bpf 2.brf

1

h[0]=0.000002

h[1]=0.000002

h[2]=0.000002

h[3]=0.000002

h[4]=0.000002

h[5]=0.000002

h[6]=0.000002

h[7]=0.000002

h[8]=0.000002

h[9]=0.000002

h[10]=0.000002

h[11]=0.000002

h[12]=0.000003

h[13]=0.000003

h[14]=0.000003

h[15]=0.000003

h[16]=0.000004

h[17]=0.000004

h[18]=0.000005

h[19]=0.000005

h[20]=0.000006

h[21]=0.000007

h[22]=0.000009

h[23]=0.000011
```

```
h[28]=0.000043

h[29]=0.000064

h[30]=0.000100

h[31]=0.000168

h[32]=0.000313

h[33]=0.000668

h[34]=0.001763

h[35]=0.006664

h[36]=0.053349

h[37]=0.989259

h[38]=0.857608

h[39]=0.029144

h[40]=0.003906

h[41]=0.001002

h[42]=0.000356

h[43]=0.000154

h[44]=0.000076

h[45]=0.000041

h[46]=0.000024

h[47]=0.000015

h[48]=0.000009

h[49]=0.000006

h[50]=0.000004

h[51]=0.000003

h[52]=0.000002
```

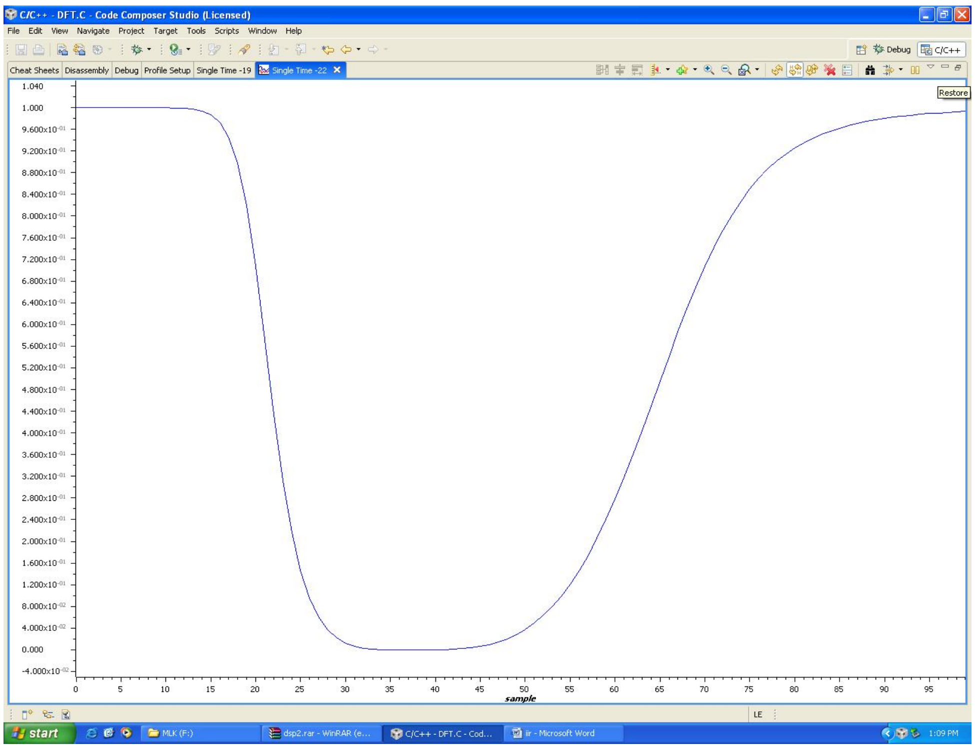
h[24]=0.000014	


C/C++ - DFT.C - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
Cheat Sheets
Disassembly
Debug
Profile Setup
Single Time -19
Single Time -20
Single Time -21
sample
start

<u>Brf</u>

enter order of the filter2	h[24]=0.217024
enter wl20 70	h[25]=0.146176
enter the choice 1.bpf 2.brf2	h[26]=0.095759
h[0]=1.000000	h[27]=0.060918
h[1]=1.000000	h[28]=0.037455
h[2]=1.000000	h[29]=0.022084
h[3]=1.000000	h[30]=0.012345
h[4]=1.000000	h[31]=0.006435
h[5]=0.999999	h[32]=0.003050
h[6]=0.999997	h[33]=0.001262
h[7]=0.999990	h[34]=0.000424
h[8]=0.999968	h[35]=0.000100

h[9]=0.999908	h[36]=0.000011
h[10]=0.999761	h[37]=0.000000
h[11]=0.999416	h[38]=0.000000
h[12]=0.998647	h[39]=0.000015
h[13]=0.996992	h[40]=0.000100
h[14]=0.993536	h[41]=0.000353
h[15]=0.986501	h[42]=0.000903
h[16]=0.972576	h[43]=0.001902
h[17]=0.946013	h[44]=0.003523
h[18]=0.898180	h[45]=0.005954
h[19]=0.819645	h[46]=0.009391
h[20]=0.707107	h[47]=0.014044
h[21]=0.571175	h[48]=0.020125
h[22]=0.433355	h[49]=0.027855
h[23]=0.312576	

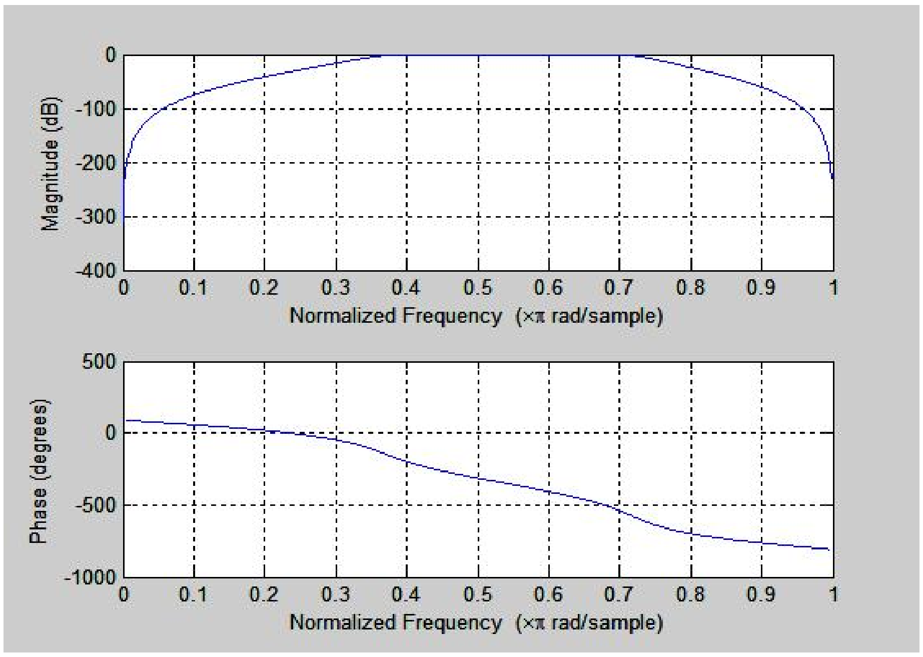

C/C++ - DFT.C - Code Composer Studio (Licensed)
File Edit View Navigate Project Target Tools Scripts Window Help
Cheat Sheets
Disassembly
Debug
Profile Setup
Single Time -19
Single Time -22
Restore
sample
start

10.2 Program

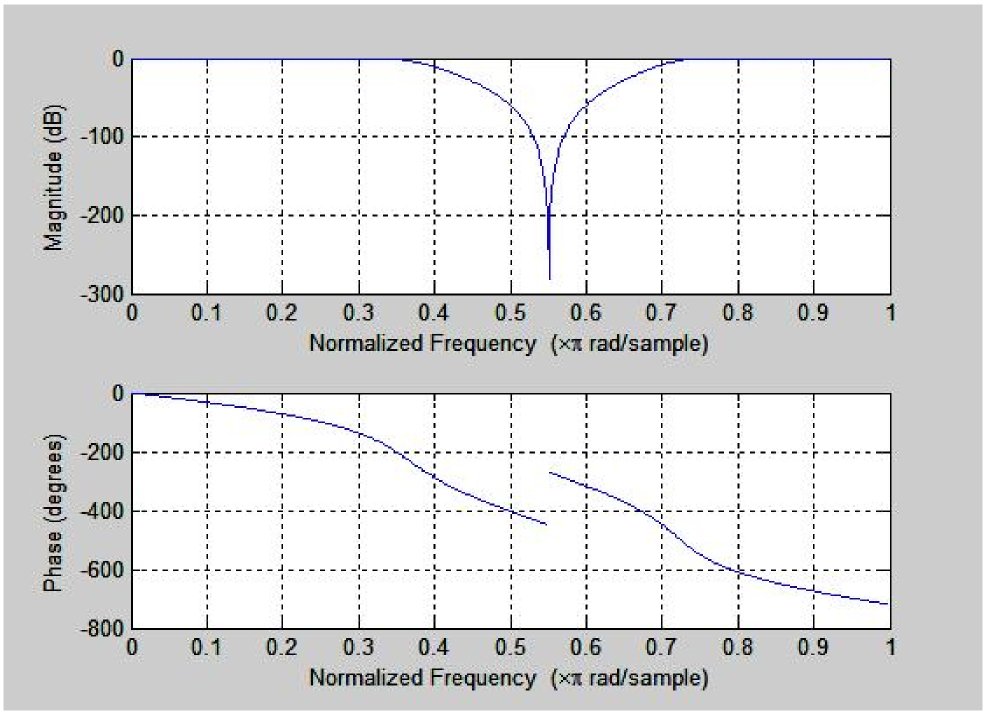
```
clc;
clear;
ap=input('Enter PB attn in dB : ');
as=input('Enter SB attn in dB : ');
fp=input('Enter PB freq in hz : ');
fs=input('Enter SB freq in hz : ');
F=input('Enter the sampling freq : ');
wp=2*pi*fp/F;
ws=2*pi*fs/F;
wp=2*F*tan(wp/2);
ws=2*F*tan(ws/2);
[N,wc]=buttord(wp,ws,ap,as,'s')
c=input('enter choice of filter 1. BPF 2. BRF \n ');
if(c==1)
    disp('Frequency response of IIR LPF is:');
[b,a]=butter(N,wc,'bandpass','s');
end
if(c==2)
    disp('Frequency response of IIR HPF is:');
[b,a]=butter(N,wc,'stop','s');
end
%[b,a]=butter(N,wc,'s')
[bz,az]=bilinear(b,a,F)
freqz(bz,az)
```

<u>Comand wind</u>

```
Enter PB attn in dB : 0.15
Enter SB attn in dB : 60
Enter PB freq in hz : [1500 4500]
Enter SB freq in hz : [2500 3000]
Enter the sampling freq : 10000
N =    5
wc =   1.0e+04 *[  1.2856    4.2824]
enter choice of filter 1. BPF 2. BRF : 1
```



```
Enter PB attn in dB : 0.15
Enter SB attn in dB : 60
Enter PB freq in hz : [1500 4500]
Enter SB freq in hz : [2500 3000]
Enter the sampling freq : 10000
N =    5
wc =   1.0e+04 *[  1.2856    4.2824]
enter choice of filter 1. BPF 2. BRF : 2
```



RESULT: IIR Filter (BP/BR) has been implemented and their magnitude responses are visualized through DSP processor and MATLAB.

Experiment-11. FIR FILTERS USING FREQUENCY SAMPLING METHOD

AIM: To Design of FIR filters using frequency sampling method

EQUIPMENTS:

Operating System	– Windows XP
Constructor	**-** Simulator
Software	- CCStudio 3 & MATLAB 7.5
Hardware	- DSK6713 kit, USB probe,5V DC supply

THEORY:

FIR Filter Design by Frequency Sampling

The technique described below is simpler than some other approaches. Herein, it is assumed here that the filter length, M, is odd.

The basic method is to specify the desired magnitude of the frequency response, |Hd(F)|, at a set of frequencies Fk = k * dF = k /M. The unit sample response is then found via the inverse DFT. The troublesome point with this method is properly setting up the desired magnitude |Hd(F)| with even symmetry, and appropriate phase, <Hd(F), (with odd symmetry). These requirements are needed to ensure the resulting filter is causal, has a constant delay (linear phase shift), and has real coefficients.

If a zero phase shift was specified for each <Hd(F) then the resulting h(n) would be centered at the origin. Hence a minimum delay of g = (M-1)/2 samples is needed to make the filter causal. By the time-shift property of the DTFT, this corresponds to multiplication of |Hd(F)| by exp(-j*g*2*pi*F). Note the linear phase variation given in the exponential. The derivative of Hd with respect to w yields the group delay for the filter, which is the constant, g, (in samples). Real coefficients are assured because of the uniform spacing of Fk and because symmetric magnitudes are assigned to Hd(Fk) for positive and negative values of Fk.

Recall that with the method of FIR design by windowing the only inputs were the cutoff frequency and the filter length. The resulting error |H(F)| - |Hd(F)| was not known at any given frequency. In this technique frequency response will be exact at the specified frequencies, F = Fk, but will be unknown at other frequencies.

General Procedure:

1) Given |Hd(F)| for 0 <= F < 1.

2) Select the filter length = M.

3) Find the frequency spacing, dF, for the sampling of Hd(F): dF = 1 / M (cycles / sample) Hd(F) will be specified for F = k * dF, with k = 0 , M-1, thus giving samples at frequencies across the range of 0 <= F < 1.

4) Find the group delay for the filter = g = (M-1) / 2.

5) Setup symmetric |Hd(k)|. Setup <Hd(k) = -g * 2 * pi * dF * k.

6) Find h(n) = IDFT{ Hd(k) }. This is an M-point IDFT.

7) Find actual frequency response, H(k) = DFT{ h(n) }. Use a large value of N, zero padding h(n) to yield an acceptable frequency resolution.

8) Assess filter by examining plot of magnitude of frequency response.

9) Repeat as needed, going to step #2.

Program

```
fs=8e3;
ts=1/fs; ns=512;
t=[0:ts:ts*(ns-1)];
f1=500;  f2=1800;  f3=2000;  f4=3200;
x1=sin(2*pi*f1*t);
x2=sin(2*pi*f2*t);
x3=sin(2*pi*f3*t);
x4=sin(2*pi*f4*t);
x=x1+x2+x3+x4;
grid on;
n=16;
w=[0.4 0.6];
w1=0.4;
a=1;
c=input('enter 1.lpf 2.hpf 3.bpf 4.brf');
if(c==1)
  b=fir1(n,w1,'low');
end;
if(c==2)
  w=0.5;
  b=fir1(n,w,'high');
end;
if(c==3)
  w=[0.4 0.6];
  b=fir1(n,w,'DC-0');
end;
if(c==4)
  w=[0.4 0.6];
  b=fir1(n,w,'DC-1');
end;
freqz(b,a);
pause;figure;
subplot(2,1,1);
npts=200;
plot(t(1:npts),x(1:npts));
title('time plot of input and output');
xlabel('time');
ylabel('input signal');
y=filter(b,a,x);
subplot(2,1,2);
plot(t(1:npts),y(1:npts));
xlabel('time');
ylabel('filtered signal');
pause;figure;
subplot(2,1,1);
xfftmag=(abs(fft(x,ns)));
xfftmagh=xfftmag(1:length(xfftmag)/2);
f=(1:1:length(xfftmagh))*fs/ns;
plot(f,xfftmagh);
title('input and output spectrum');
xlabel('frequency');ylabel('input spectrum');
subplot(2,1,2);
yfftmag=(abs(fft(y,ns)));
yfftmagh=yfftmag(1:length(yfftmag)/2);
f=(1:1:length(yfftmagh))*fs/ns;
plot(f,yfftmagh);
xlabel('frequency');ylabel('filtered spectrum');
```

Low Pass Filter

command window display

```
Enter 1.lpf 2.hpf 3.bpf 4.brf 1
```

OUTPUT GRAPHS

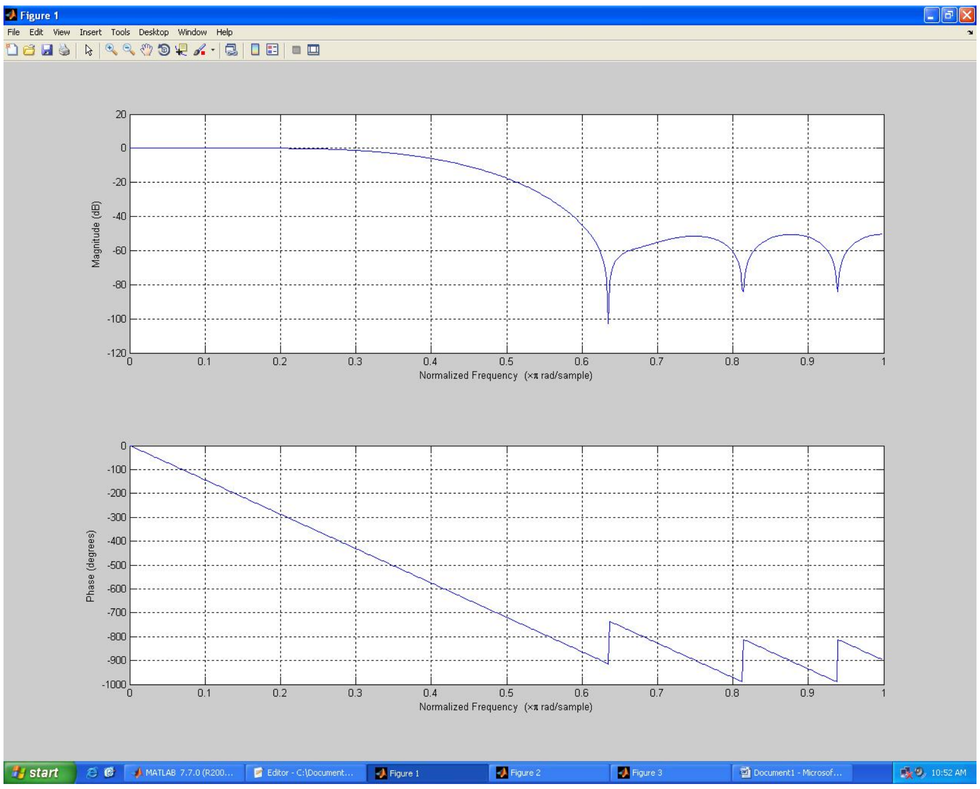


Figure 1
File Edit View Insert Tools Desktop Window Help
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)
start
MATLAB 7.7.0 (R200...
Editor - C:\Document...
Figure 1
Figure 2
Figure 3
Document1 - Microsof...
10:52 AM

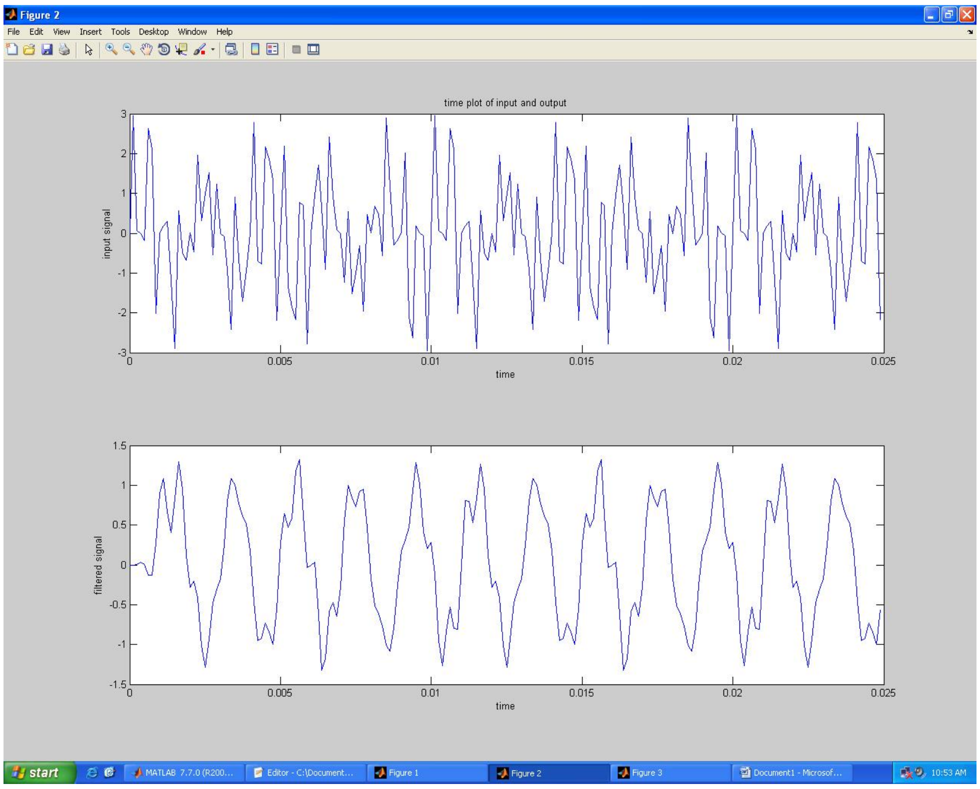
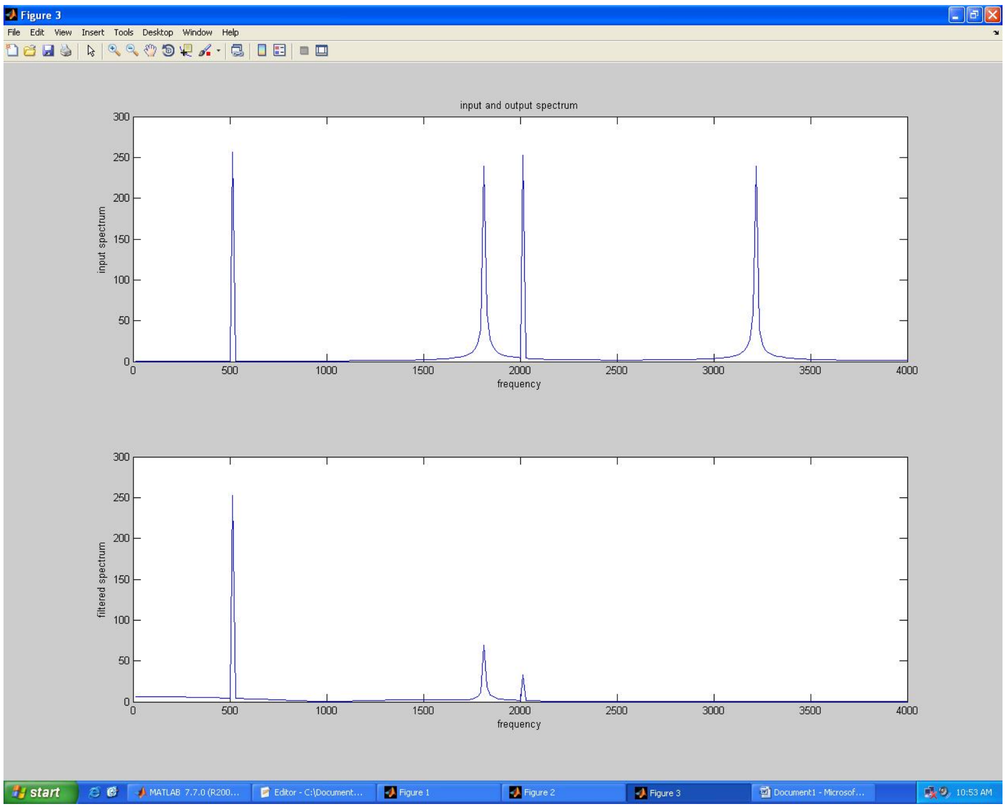


Figure 2
File Edit View Insert Tools Desktop Window Help
time plot of input and output
input signal
time
filtered signal
time
start
MATLAB 7.7.0 (R200...
Editor - C:\Document...
Figure 1
Figure 2
Figure 3
Document1 - Microsof...
10:53 AM



High Pass Filter

command window display

Enter 1.lpf 2.hpf 3.bpf 4.brf 2

OUTPUT GRAPHS

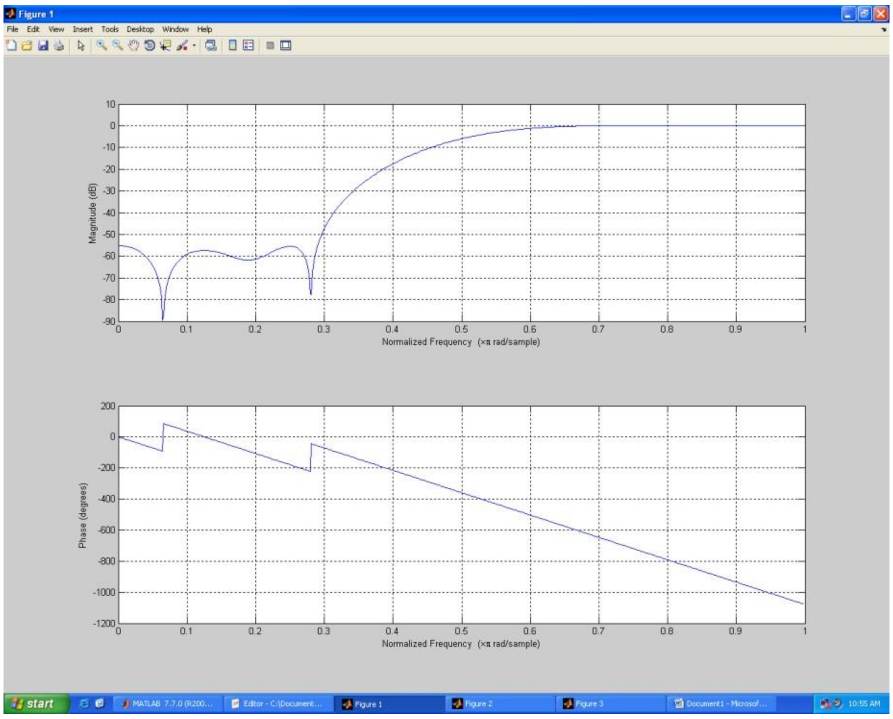


Figure 1
File Edit View Insert Tools Desktop Window Help
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)
start
MATLAB 7.7.0 (R200...
Editor - C:\Document...
Figure 1
Figure 2
Figure 3
Document1 - Microsof...
10:55 AM

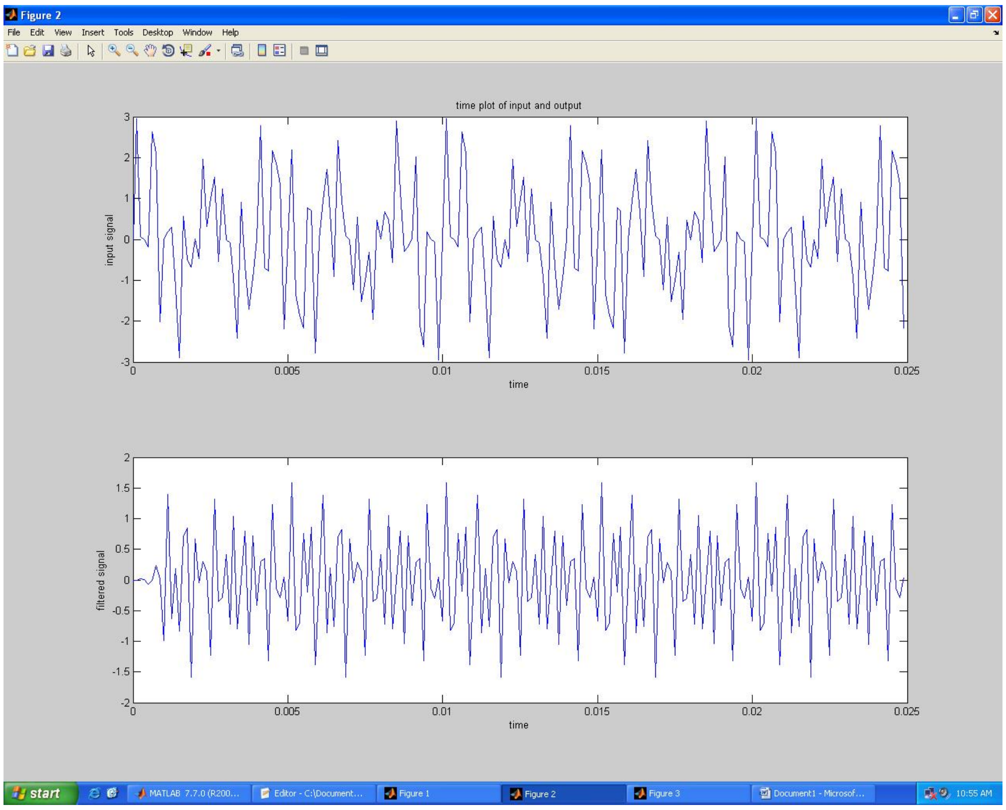
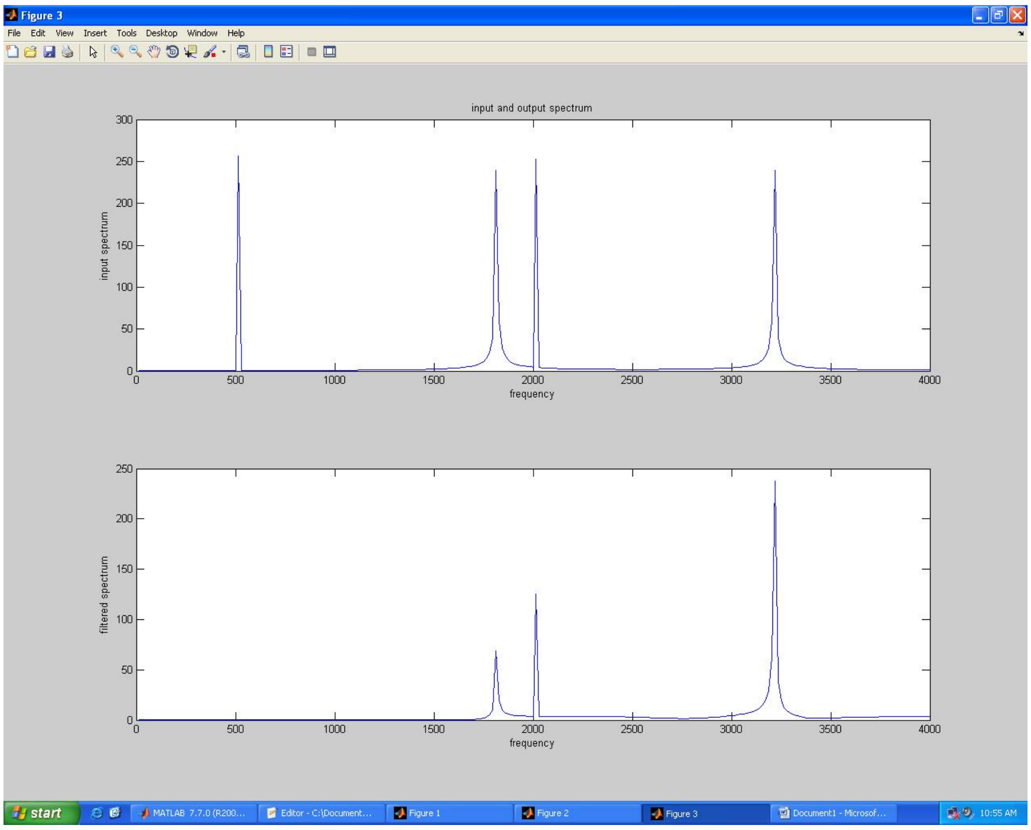


Figure 2
File Edit View Insert Tools Desktop Window Help
time plot of input and output
input signal
time
filtered signal
time
start
MATLAB 7.7.0 (R200...
Editor - C:\Document...
Figure 1
Figure 2
Figure 3
Document1 - Microsof...
10:55 AM



Band Pass Filter

command window display

Enter 1.lpf 2.hpf 3.bpf 4.brf 3

OUTPUT

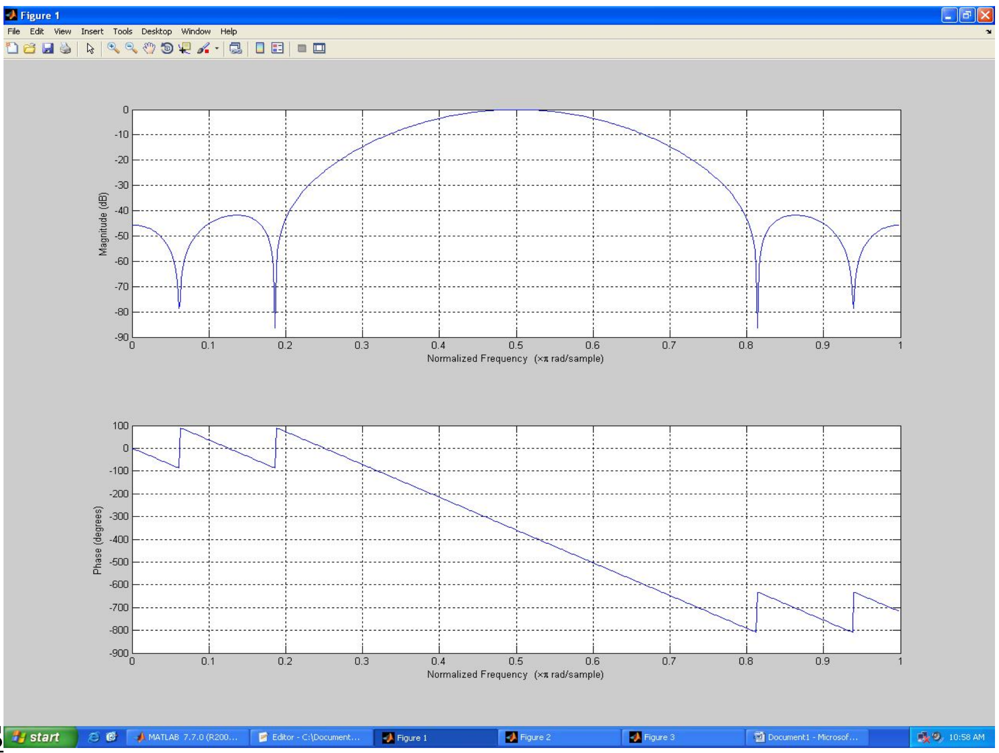


Figure 1
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)

GRAPHS

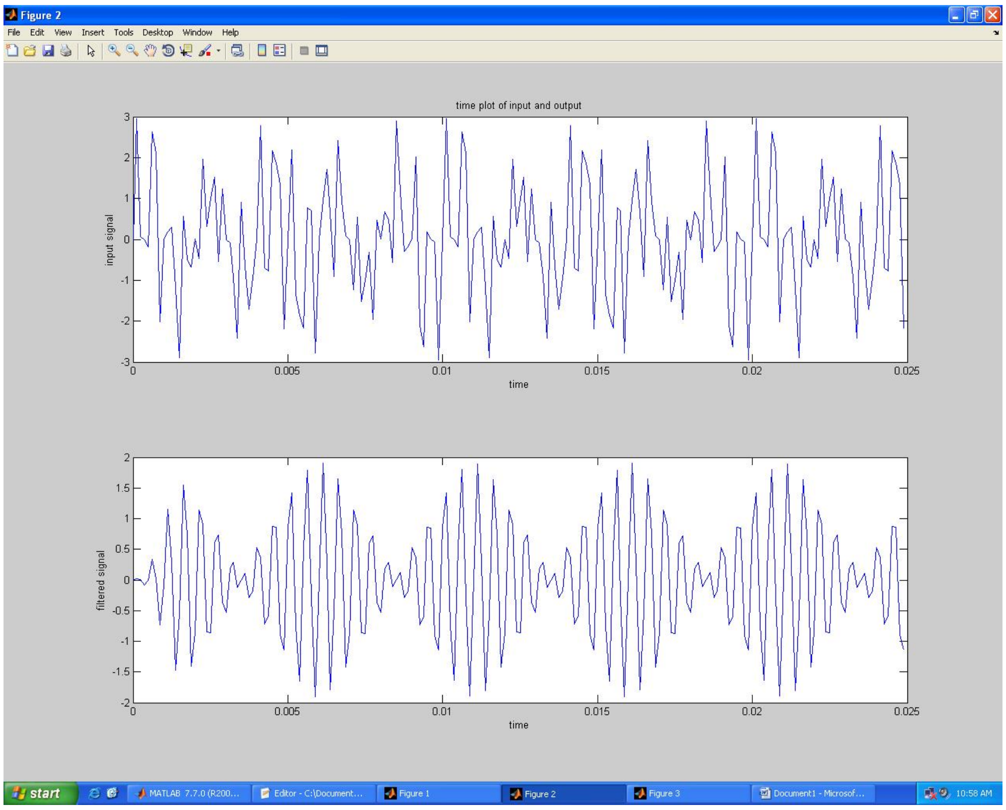
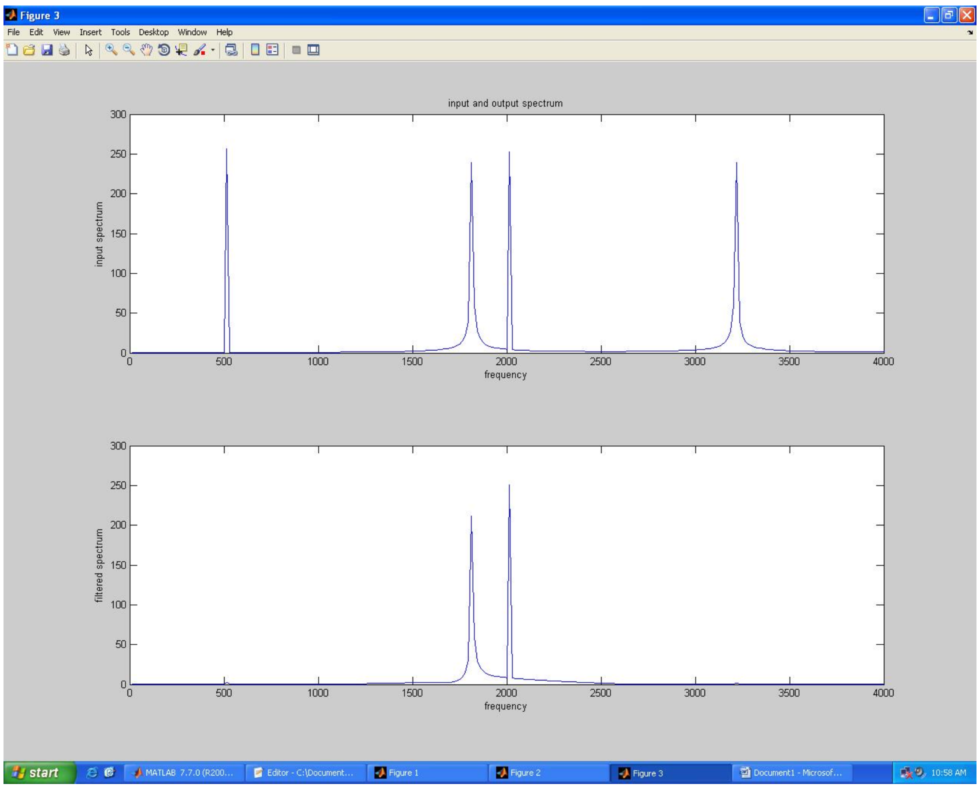


Figure 2
time plot of input and output
input signal
time
filtered signal
time



Band Reject Filter

command window display

```
Enter 1.lpf 2.hpf 3.bpf 4.brf  4
```

OUTPUT GRAPHS

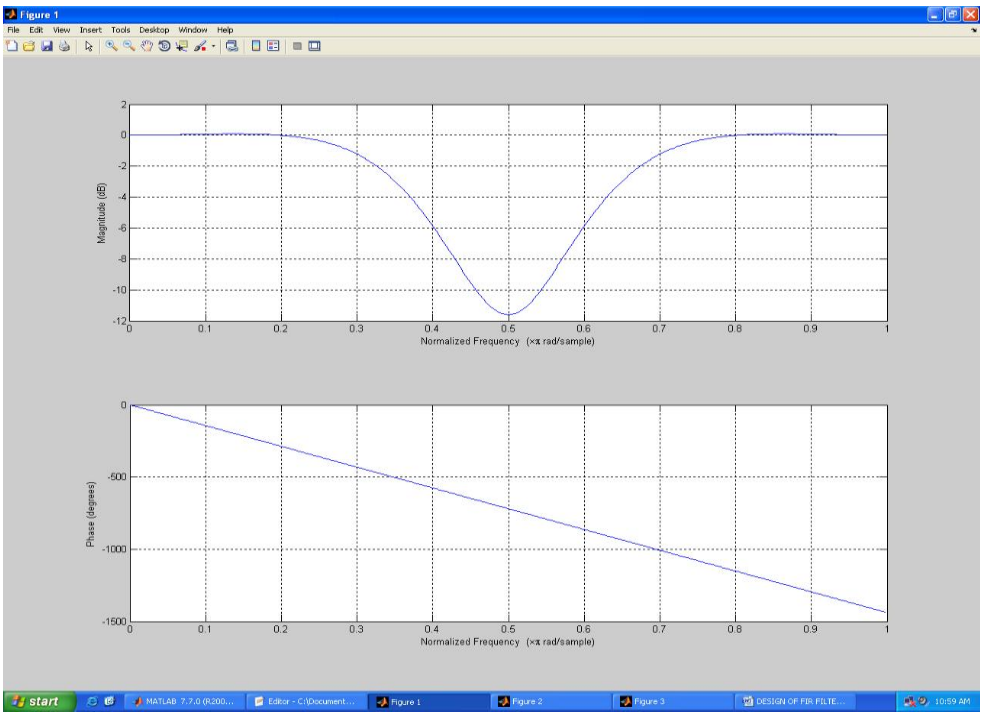


Figure 1
File Edit View Insert Tools Desktop Window Help
Magnitude (dB)
Normalized Frequency (×π rad/sample)
Phase (degrees)
Normalized Frequency (×π rad/sample)
start
MATLAB 7.7.0 (R200...
Editor - C:\Document...
Figure 1
Figure 2
Figure 3
DESIGN OF FIR FILTE...
10:59 AM

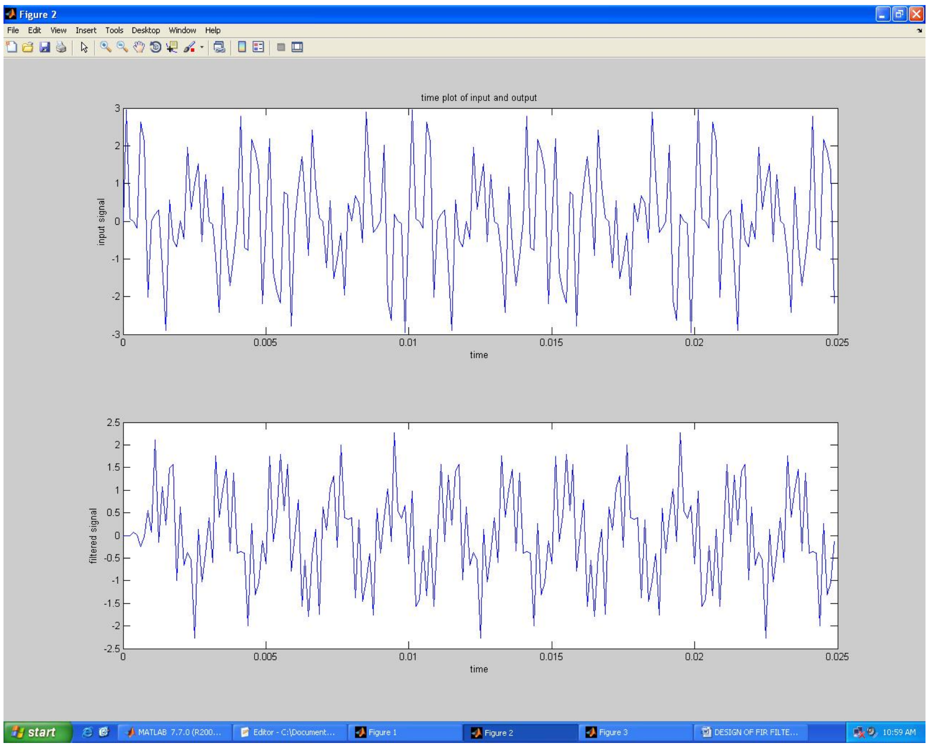
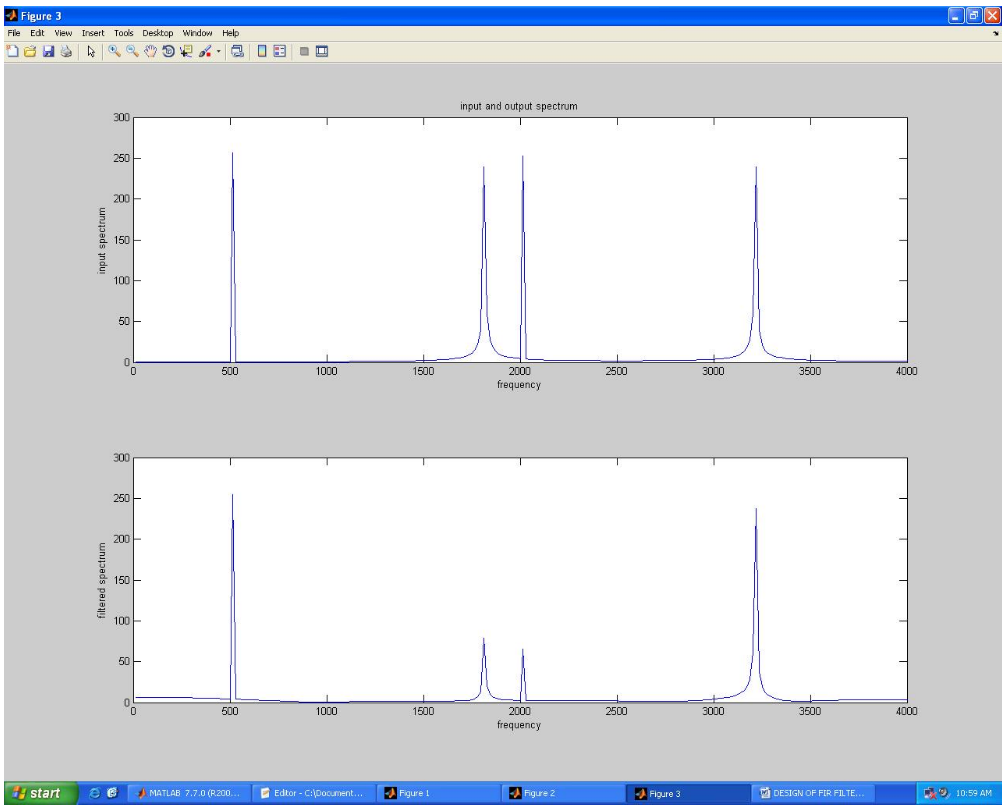


Figure 2
File Edit View Insert Tools Desktop Window Help
time plot of input and output
input signal
time
filtered signal
time
start
MATLAB 7.7.0 (R200...
Editor - C:\Document...
Figure 1
Figure 2
Figure 3
DESIGN OF FIR FILTE...
10:59 AM



RESULT: FIR Filter (LP?BP?BP/BR) using frequency sampling methed has been implemented and their magnitude responses are visualized through DSP processor and MATLAB.

Experiment-12 Linear to circular convolution

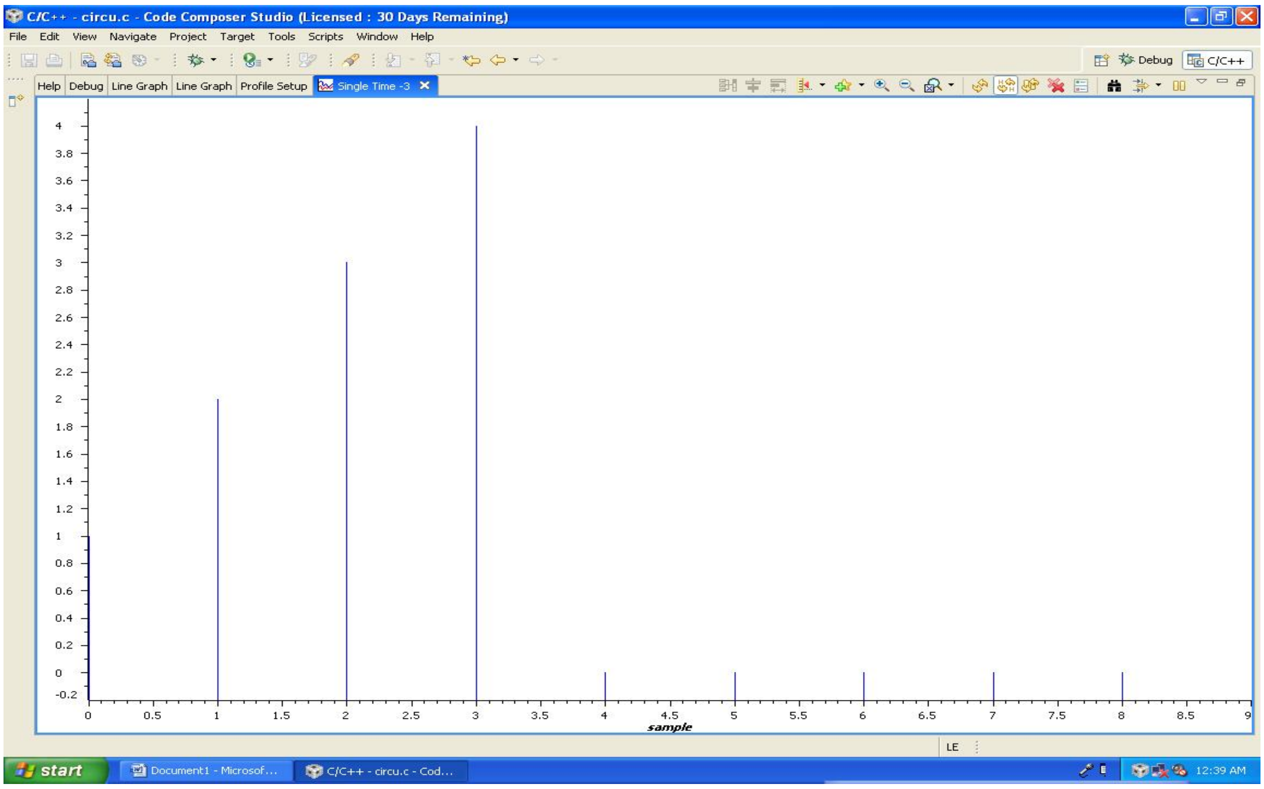
Program code:

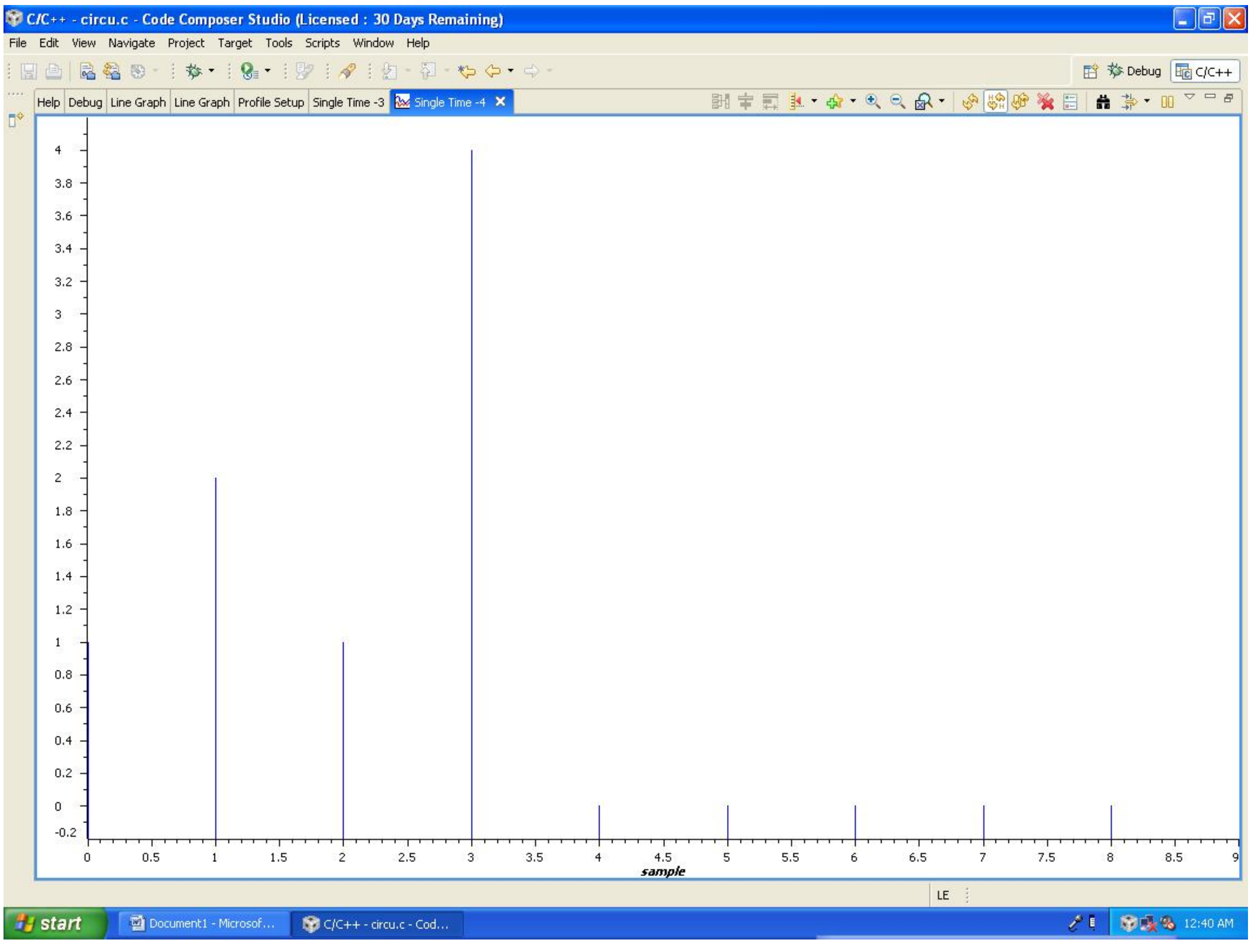
```
#include<stdio.h>
#include<math.h>
int x[30],h[30],y[30],x2[30],a[30],m,n,i,j;
main()
{
      printf("enter length of two sequences");
      scanf("%d%d",&m,&n);
      printf("enter the first sequence \n");
      for(i=0;i<m;i++)
      scanf("%d",&x[i]);
      printf("enter the second sequence \n");
      for(i=0;i<n;i++)
      scanf("%d",&h[i]);
            for(i=0;i<m+n-1;i++)
{
            y[i]=0;
            for(j=0;j<=i;j++)
            {
                  y[i]=y[i]+(x[j]*h[i-j]);
            }
}
      for(i=0;i<n;i++)
      y[i]=y[n+i]+y[i];
      for(i=0;i<n;i++)
      printf("Result is \n y[%d]=%d",i,y[i]);
}
```

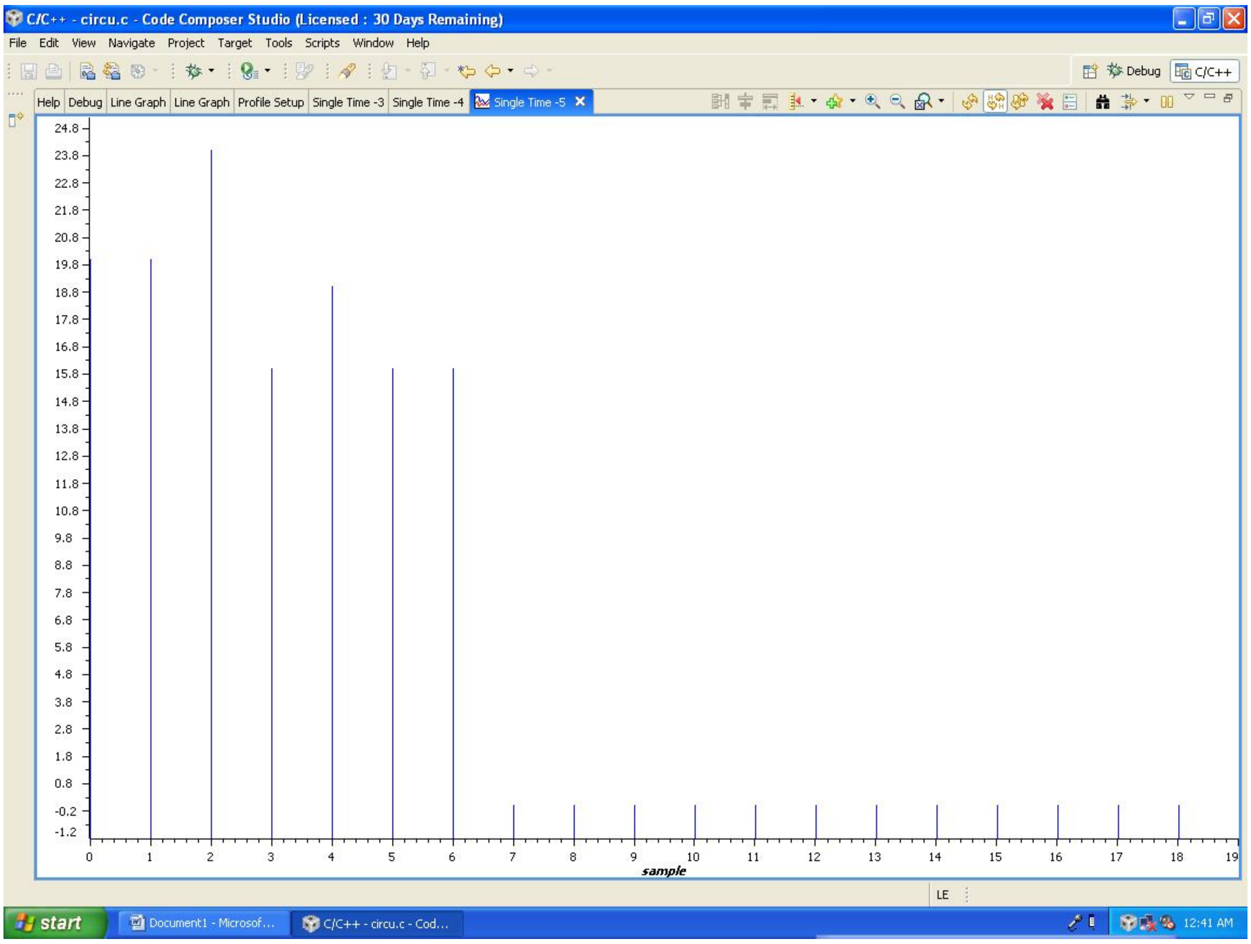
Output:

```
enter length of two sequences4
4
enter the first sequence
1
2
3
4
enter the second sequence
1
2
1
4
Result is
y[0]=20
y[1]=20
y[2]=24
y[3]=16
```

Waveforms:


C/C++ - circu.c - Code Composer Studio (Licensed : 30 Days Remaining)
File Edit View Navigate Project Target Tools Scripts Window Help
Debug C/C++
Help Debug Line Graph Line Graph Profile Setup Single Time -3
4
3.8
3.6
3.4
3.2
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9
sample
LE
start Document1 - Microsof... C/C++ - circu.c - Cod... 12:39 AM


C/C++ - circu.c - Code Composer Studio (Licensed : 30 Days Remaining)
File Edit View Navigate Project Target Tools Scripts Window Help
Debug C/C++
Help Debug Line Graph Line Graph Profile Setup Single Time -3 Single Time -4
4
3.8
3.6
3.4
3.2
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9
sample
LE
start Document1 - Microsof... C/C++ - circu.c - Cod... 12:40 AM


C/C++ - circu.c - Code Composer Studio (Licensed : 30 Days Remaining)
File Edit View Navigate Project Target Tools Scripts Window Help
Debug
C/C++
Help Debug Line Graph Line Graph Profile Setup Single Time -3 Single Time -4 Single Time -5
24.8
23.8
22.8
21.8
20.8
19.8
18.8
17.8
16.8
15.8
14.8
13.8
12.8
11.8
10.8
9.8
8.8
7.8
6.8
5.8
4.8
3.8
2.8
1.8
0.8
-0.2
-1.2
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
sample
LE
start
Document1 - Microsof...
C/C++ - circu.c - Cod...
12:41 AM

Result:

This lab manual has been updated by

Faculty Name

Dr. Kaustubh Kumar Shukla
kaustubh.shukla@gnindia.dronacharya.info

Cross checked By
HOD ECE/EEE/ECZ

Verified By

Director, DGI Greater Noida

Please spare some time to provide your valuable feedback.