**LABORATORY NAME: DIGITAL SIGNAL PROCESSING LAB** (**20APC0422)**

**CLASS: III B.Tech II Semester**

**Course Outcomes:**

CO1:Analyze energy and power of a discrete time sequence

CO2: compute convolution and correlation of discrete time sequences.

CO3: compute fourier transform of discrete time sequence.

CO4: Design and analyze various analog filters.

CO5: Design and analyze various digital filters

**Course Syllabus:**

**List of Experiments: (Minimum of 5 experiments are to be conducted from each part)**

**Software Experiments (PART – A)**

1. Finding Power and Energy of a given sequence.

2.Convolution and Correlation (auto and cross correlation) of discrete sequences.

3. DTFT of a given signal

4. N – point FFT algorithm

5. Design of analog filters.

6. Design of IIR filter using windowing technique and verify the frequency response of the filter.

7. Design of FIR filter using any of the available methods and verify the frequency response of the filter.

**Using DSP Processor kits (Floating point) and Code Composure Studio (CCS) (PART – B)**

1. Finding Power and Energy of a given sequence.

2.Convolution and Correlation (auto and cross correlation) of discrete sequences.

3. DTFT of a given signal

4. N – point FFT algorithm

5. Design of analog filters.

6. Design of IIR filter using windowing technique and verify the frequency response of the filter.

7. Design of FIR filter using any of the available methods and verify the frequency response of the filter.

**Equipment/Software Required:**

1. Licensed MATLAB software with required tool boxes for 30 users.

2. DSP floating Processor Kits with Code Composure Studio (8 nos.)

3. Function generators

4. CROs

5. Regulated Power Supplies.

**PROCEDURE TO RUN MATLAB PROGRAMS:**

* Double Click on MATLAB icon on the desktop.
* Open New MATLAB script
* Type the program in the editor window and save file with .m extension
* Run the program. If errors occur, rectify and then run until zero errors.
* See the command window for numerical results and figure window for waveforms.
* Note down the outputs and draw the waveform neatly.

**1. Finding Power and Energy of a given sequence.**

**AIM:** To write a MATLAB program to compute Power and Energy

**EQUIPMENTS:**

PC with MATLAB Software

**PROGRAM*:***

clc;

close all;

clear all;

x=input('Enter input sequence to calculate Energy and power x(n) = ' );

n=0:length(x)-1;

stem(n,x);

axis([-2 10 -2 10]);

title('Sequence x(n)=[1,2,3,4,5,6,7,8,9] for Energy and Power calculation');

xlabel('----------->n');

ylabel('Magnitude ------>x(n)');

grid on;

E=sum(abs(x).^2);

disp(['Energy of the given sequence is E= ',num2str(E),' Joules']);

P=E/[2\*length(x)+1];

disp(['Eower of the given sequence is P= ',num2str(P),' Watts']);

**OUTPUT:**

Enter the input sequence:[1 2 3 4 5 6 7 8 9]

Energy of the given sequence is E= 285 Joules

Eower of the given sequence is P=15 Watts



**RESULTS:** Power And Energy of a given Sequence were Estimated using MATLAB

**2.Convolution and Correlation of Signals**

**AIM:** To write a MATLAB program to compute

(a) Linear Convolution of sequences

(b) Circular Convolution of sequences

(c) Auto correlation of a sequence

(d) Cross correlation of sequences using MATLAB

**EQUIPMENTS:**

PC with MATLAB Software

**PROGRAM*:***

**(A) LINEAR CONVOLUTION:**

clc;

close all;

clear all;

x=input('Enter the input sequence:');

h=input('Enter the impulse sequence:');

n1=length(x);

n2=length(h);

n3=n1+n2-1;

y=conv(x,h); % Convolution function

figure(1);

subplot(3,1,1);

t1=0:1:(n1-1);

stem(t1,x);

xlabel('------->(n)');

ylabel('------->x(n)');

title(' Input Sequence');

subplot(3,1,2);

t2=0:1:(n2-1);

stem(t2,h);

xlabel('------->(n)');

ylabel('------->h(n)');

title(' Impulse Sequence');

subplot(3,1,3);

t3=0:1:(n3-1);

stem(t3,y);

xlabel('------->(n)');

ylabel('------->y(n)');

title(' Resultent Sequence');

disp('The resultant signal is:'); y

**OUTPUT:**

Enter the input sequence:[5 4 1 8 4]

Enter the impulse sequence:[8 7 2 4]

The resultant signal is:

y = 40 67 46 99 106 48 40 16

****

**PROGRAM:**

**(B) CIRCULAR CONVOLUTION:**

clc;

clear all;

x=input('Enter the input sequence:');

y=input('Enter the impulse sequence:');

n1=length(x);

n2=length(y);

N = max(n1,n2);

n3 = n1-n2;

if n3>0

y = [y,zeros(1,n3)];

else

x = [x,zeros(1,n3)];

end

a = length(x);

subplot(3,1,1);

t1 = 0:1:(a-1);

stem(t1,x);

xlabel('n');

ylabel('x(n)');

title('First sequence after zero padding');

subplot(3,1,2);

b = length(y);

t2 = 0:1:(b-1);

stem(t2,y);

xlabel('n');

ylabel('y(n)');

title('Second sequence after zero padding');

for n = 1:N;

k(n) = 0;

for i=1:N

j=n-i+1;

if j<=0

j=N+j;

end

k(n) = k(n)+ x(i) \* y(j);

end

end

subplot(3,1,3);

t3 = 0:1:(a-1);

disp('Resultant Circular convolution Sequence');k

stem(t3,k);

xlabel('n');

ylabel('Circular convolution result');

**OUTPUT:**

Enter the input sequence:[1 3 4 6]

Enter the impulse sequence:[ 2 1 3 4]

Resultant Circular convolution Sequence

k = 32 41 38 29

**OUTPUT WAVEFORMS:**

****

**PROGRAM:**

**(C) AUTO CORRELATION:**

clc;

clear all;

close all;

x = input('Enter the sequence for calculating Autocorrelation');

n = length(x)-1;

k= (-n):n;

subplot(2,1,1);

t1 = 0:1:n;

stem(t1,x);

xlabel('-----samples (n)');

ylabel('---------Amplitude');

title('Input sequence');

r = xcorr(x,x); % Correlation calculating function

subplot(2,1,2);

stem(k,r);

xlabel('-----samples (n)');

ylabel('---------Amplitude');

title('Autocorrelation Resultant sequence');

disp('Autocorrelation Resultant sequence'); r

**OUTPUT:**

Enter the sequence for calculating Autocorrelation[1 3 5 7]

Autocorrelation Resultant sequence

r = 7.0000 26.0000 53.0000 84.0000 53.0000 26.0000 7.0000

****

**PROGRAM:**

**(D) CROSS CORRELATION:**

clc;

clear all;

close all;

x = input('Enter the first sequence for calculating Cross-correlation');

y = input('Enter the second sequence for calculating Cross-correlation');

n1 = length(x)-1;

n2 = length(y)-1;

k= (-n1):n2;

% displaying First sequence

subplot(3,1,1);

t1 = 0:1:n1;

stem(t1,x);

xlabel('-----samples (n)');

ylabel('---------Amplitude');

title('First Input sequence');

% Displaying Second sequence

subplot(3,1,2);

t2 = 0:1:n2;

stem(t2,y);

xlabel('-----samples (n)');

ylabel('---------Amplitude');

title('Second Input sequence');

% Calling the function to calculate cross correlation

r = xcorr(x,y);

% Displaying Resulatant sequence

subplot(3,1,3);

stem(k,r);

xlabel('-----samples (n)');

ylabel('---------Amplitude');

title('Cross - correlation Resultant sequence');

disp('Cross - correlation Resultant sequence'); r

**OUTPUT:**

Output: Enter the first sequence for calculating Cross-correlation [2 4 6 8]

Enter the second sequence for calculating Cross-correlation [1 3 5 7]

Cross - correlation Resultant sequence

r = 14.0000 38.0000 68.0000 100.0000 62.0000 30.0000 8.0000

**OUTPUT WAVEFORMS:**

****

**RESULT:**

Thus the convolution and correlation of a given sequences were computed using MATLAB.

**3.DTFT of a given signal**

**AIM:** To Compute the DTFT of a given signal using MATLAB.

**EQUIPMENTS:**

**PC with MATLAB Software**

**PROGRAM:**

clc;

clear all;

close all;

%// Generate input signal

t = linspace(0, 10, 1000); %(t = linspace(0,5 );)

x = sin(2 \* pi \* t); %(x = rectpuls(t);)

%// Compute DTFT and IDTFT

[X\_w, F] = dtft(x, 1000); %// DTFT

%X\_r = ifft(ifftshift(X\_w)); %// IDTFT

%// Plot the result

figure

subplot(2, 1, 1), plot(t, x);

xlabel('-----time period (t)');

ylabel('---------Amplitude');

title('Input Signal');

subplot(2, 1, 2), plot(F, X\_w);

xlabel('-----Frequency in Hz');

ylabel('---------Magnitude');

title('Fourier Transform of Signal');

function [H, W] = dtft(h, N)

N = fix(N);

L = length(h); h = h(:); %<-- for vectors ONLY !!!

` if( N < L )

error('DTFT: # data samples cannot exceed # freq samples')

end;

W = (2 \* pi / N) \* (0:(N-1))';

mid = ceil(N/2) + 1;

W(mid:N) = W(mid:N) - 2 \* pi;% <---move [pi,2pi) to [-pi,0)

W = fftshift(W);

H = fftshift(fft(h,N)); %<---move negative freq components

End



**RESULT:** Thus the DTFT of a given signal was computed using MATLAB

**4.N-Point FFT Algorithm**

**AIM:** To compute the Discrete Fourier Transform of a given signal using N- Point FFT algorithm using MATLAB.

**EQUIPMENTS:**

PC with MATLAB Software

**PROGRAM:**

clc;

clear all;

close all;

X=input('Enter the sequence : ');

N=input('Enter the Point : ');

n=length(X);

x=[X zeros(1,N-n)];

M=log2(N);

for m=1:M

d=2^(M-m+1);

for l=1:d:(N-d+1)

for k=0:(d/2)-1

w=exp(-1i\*2\*pi\*k/d);

z1=x(l+k);

z2=x(l+k+d/2);

x(l+k)=z1+z2;

x(l+k+d/2)=(z1-z2)\*w;

end

end

end

y=bitrevorder(x);

disp('FFT Sequence:');y

subplot(3,1,1), stem(abs(X));

title('Input Sequence'); xlabel('---samples(n)'); ylabel('---Amplitude');

subplot(3,1,2), stem(abs(y));

title('Magnitude Response'); xlabel('---samples(n)'); ylabel('---Amplitude');

subplot(3,1,3), stem(angle(y));

title('Phase Response');

xlabel('---samples(n)'); ylabel('---Amplitude');

**OUTPUT:**

Enter the sequence: [0 1 2 3 4 5 6 7]

Enter the Point : 8

FFT Sequence:

y =

28.0000 + 0.0000i -4.0000 + 9.6569i -4.0000 + 4.0000i -4.0000 + 1.6569i -4.0000 + 0.0000i -4.0000 - 1.6569i -4.0000 - 4.0000i -4.0000 - 9.6569i



**RESULT:** Thus the Discrete Fourier transform of a given signal was computed using FFT algorithm by using MATLAB.

**5.ANALOG FILTER DESIGN**

**AIM:** To find the frequency response of Butterworth Analog Low Pass , High Pass , Band Pass and Band Reject filters using MATLAB

**REQUIREMENTS:**

PC with MATLAB Software

**PROGRAM:**

**%ANALOG BUTTERWORTH LOW PASS FILTER%**

clc;

format long;

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);

figure(1);

title(' Analog Filters - Butterworth ');

subplot(2,1,1);

plot(om/pi,m);

title(' Low Pass Filter');

ylabel('gain in db-->');

xlabel('(a)normalised frequency-->');

subplot(2,1,2);

plot(om/pi,an);

xlabel('(b)normalised frequency-->');

**OUTPUT:**

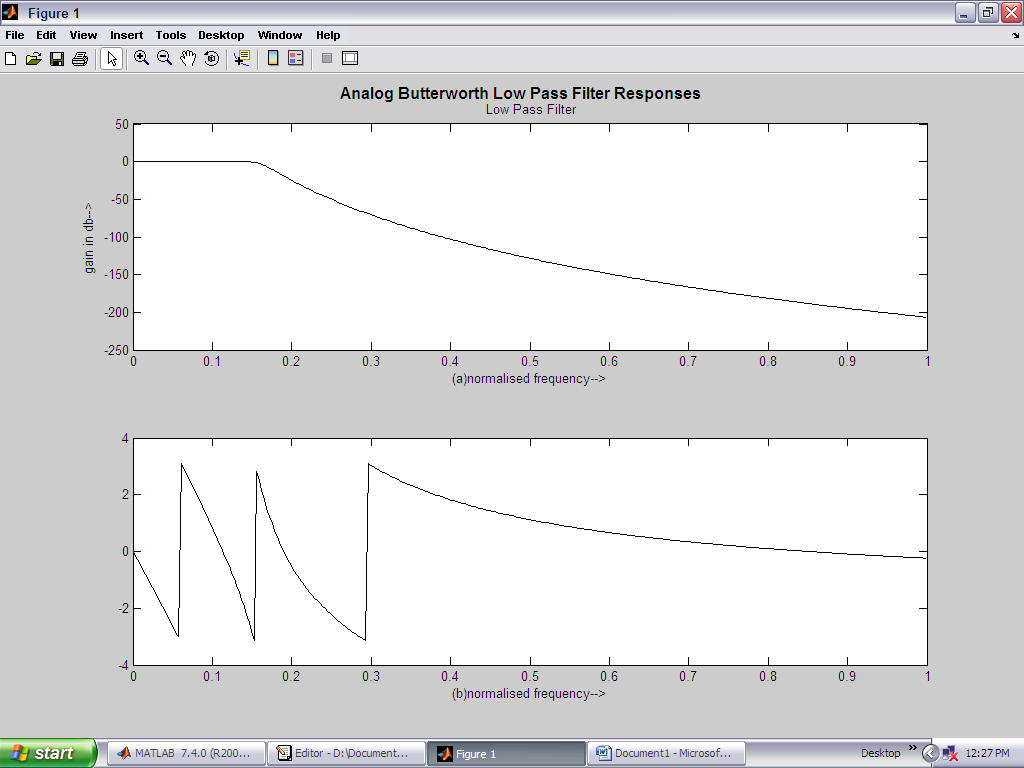
Enter the passband ripple -->0.15

Enter the stopband ripple-->60

Enter the passband frequency--->1500

Enter the stopband frequency--->3000

Enter the sampling frequency--->7000

****

**%BUTTERWORTH HIGH PASS FILTER%**

clc;

format long;

disp('Enter the values for Analog Butterworth High Pass Filter');

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 --->');

[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);

title(' High Pass Filter');

ylabel('Gain in dB-->');

xlabel('Normalised Frequency-->');

subplot(2,1,2);

plot(om/pi,an);

xlabel('Normalised Frequency-->');

ylabel('Phase in radians-->');

**OUTPUTS:**

Enter the values for Analog Butterworth High Pass Filter

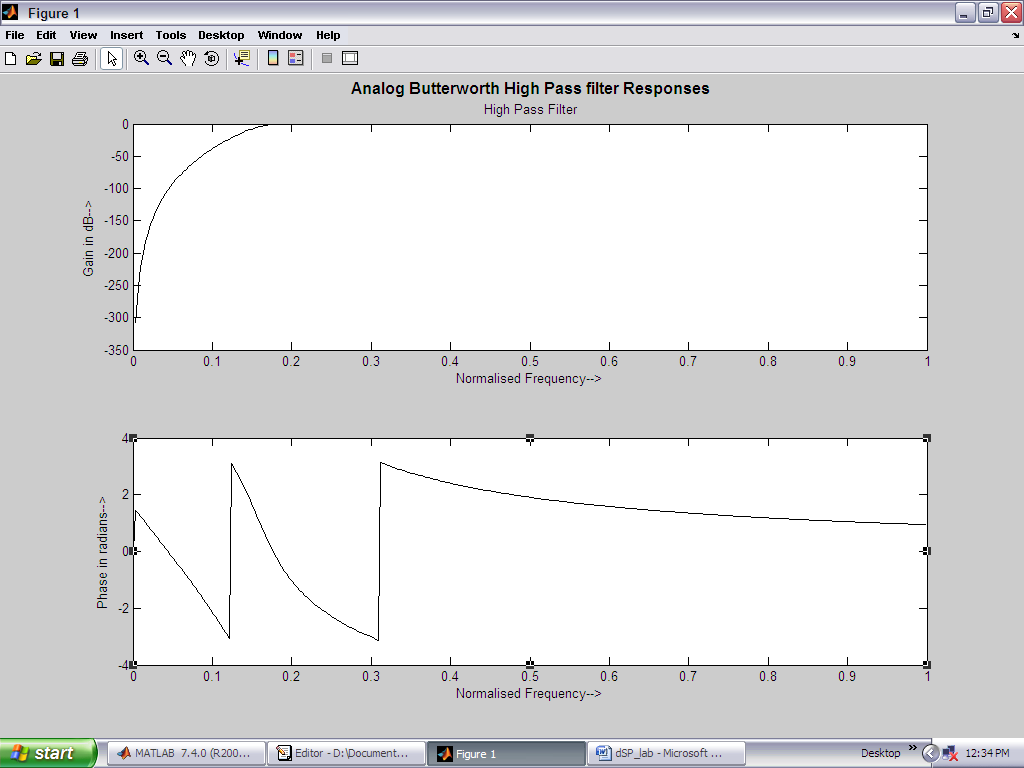
Enter the passband ripple -->0.2

Enter the stopband ripple -->40

Enter the passband frequency --->2000

Enter the stopband frequency --->3500

Enter the sampling frequency --->8000



**% ANALOG BUTTERWORTH BAND PASS FILTER**

clc;

format long;

disp('Enter the values for Analog Butterworth Band Pass Filter');

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,'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);

ylabel('Gain in dB-->');

xlabel('(a)normalised frequency-->');

subplot(2,1,2);

plot(om/pi,an);

xlabel('(b)normalised frequency-->');

ylabel('phase in radians-->');

**OUTPUT:**

Enter the values for Analog Butterworth Band Pass Filter

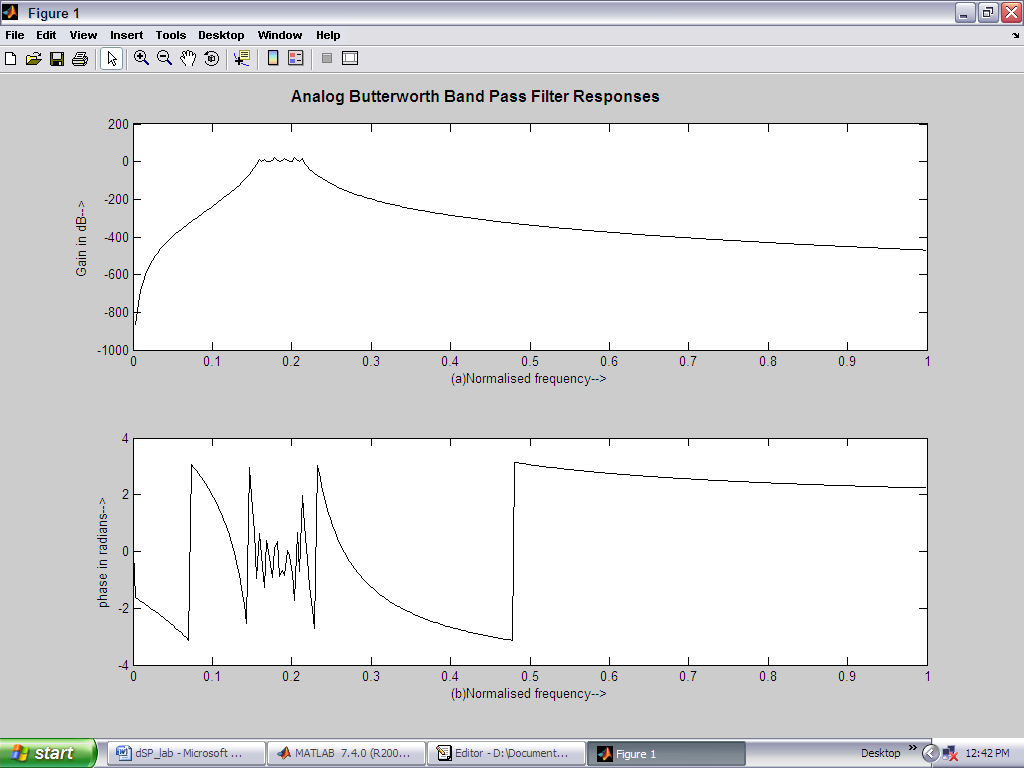
Enter the passband ripple -->0.36

Enter the stopband ripple -->36

Enter the passband frequency --->1500

Enter the stopband frequency --->2000

Enter the sampling frequency --->6000



**%ANALOG BUTTERWORTH BAND PASS FILTER**

clc;

format long;

disp('Enter the values for Analog Butterworth Band Pass Filter');

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);

ylabel('Gain in dB-->');

xlabel('(a)Normalised Frequency-->');

subplot(2,1,2);

plot(om/pi,an);

xlabel('(b)Normalised frequency-->');

ylabel('Phase in radians-->');

**OUTPUT:**

Enter the values for Analog Butterworth Band Pass Filter

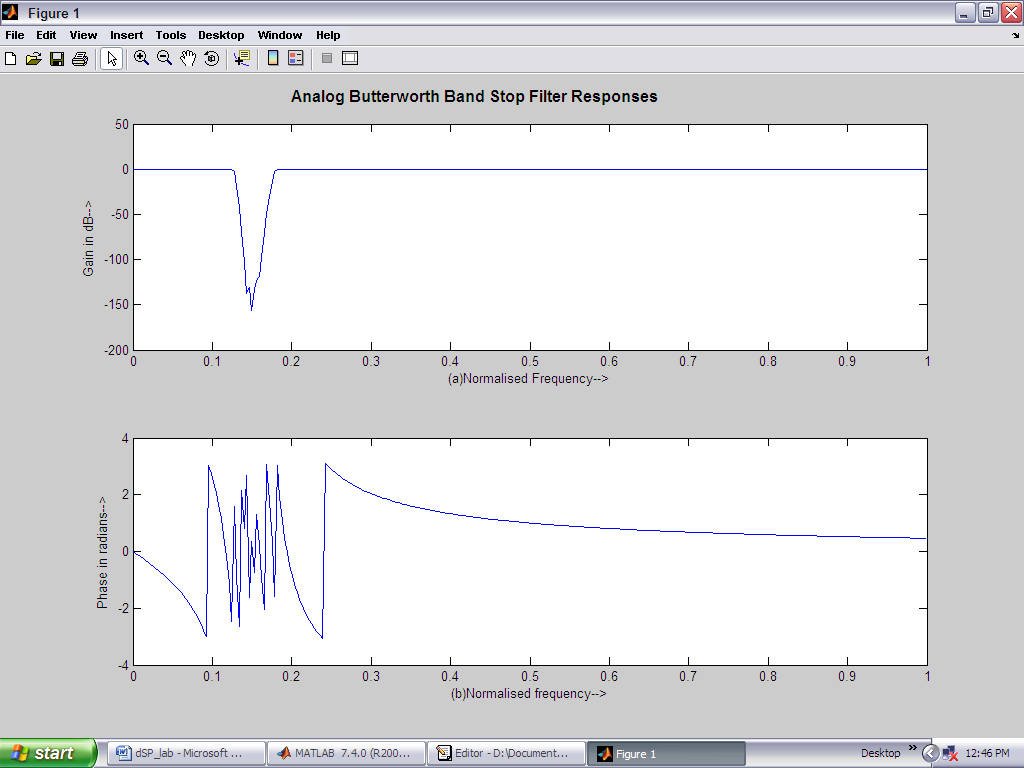
Enter the passband ripple -->0.28

Enter the stopband ripple -->28

Enter the passband frequency --->1000

Enter the stopband frequency --->1400

Enter the sampling frequency --->5000



**RESULT:** Thus Butterworth Analog filter was designed and its frequency responses characteristics were observed using MATLAB

**7.FIR Filter Design using Windowing Technique**

**AIM:** To find the frequency and phase response of the digital FIR filters using Rectangular windowing technique in MATLAB

**REQUIREMENTS:**

PC with MATLAB Software

**PROGRAM:**

**% FIR FILTER DESIGN USING RECTANGULAR WINDOW**

clc;

rp=input('Enter the Passband Ripple-->');

rs=input('Enter the Stop band Ripple-->');

fp=input('Enter the Pass band Freq-->');

fs=input('Enter the Stop band freq-->');

f=input('Enter the sampling freq-->');

wp=2\*fp/f;

ws=2\*fs/f;

num=-20\*log10(sqrt(rp\*rs))-13;

dem=14.6\*(fs-fp)/f;

n=ceil(num/dem);

n1=n+1;

if (rem(n,2)~=0)

n1=n;

n=n-1;

end

y=boxcar(n1);

**% LOW PASS FILTER**

b=fir1(n,wp,y);

[h,o]=freqz(b,1,256);

m=20\*log10(abs(h));

subplot(2,2,1);

plot(o/pi,m);

title(' Low Pass Filter ');

ylabel('Gain in dB-->');

xlabel('Normalised Frequency -->');

**%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(' High Pass Filter ');

ylabel('Gain in dB-->');

xlabel('Normalised Frequency -->');

**%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('Band Pass Filter ');

ylabel('Gain in dB-->');

xlabel('Normalised Frequency -->');

**%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(' Band Stop Filter ');

ylabel('Gain in dB-->');

xlabel('Normalised Frequency -->');

**OUTPUTS:**

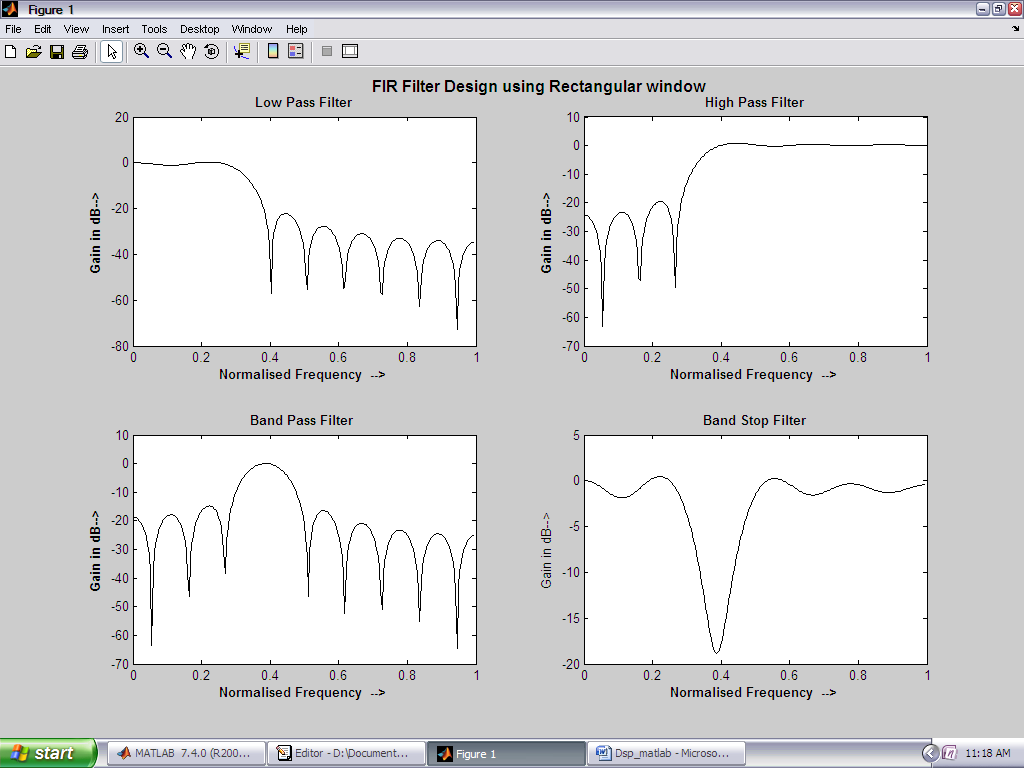
Enter the Pass band Ripple-->.04

Enter the Stop band Ripple-->.05

Enter the Pass band Freq-->1500

Enter the Stop band freq-->2000

Enter the sampling freq-->9000



**Using DSP Processor kits (Floating point) and Code Composure Studio (CCS) (**1. Finding Power and (or) Energy of a given sequence.

2.Convolution and Correlation (auto and cross correlation) of discrete sequences.

3. DTFT of a given signal

4. N – point FFT algorithm

5. Design of analog filters.

6. Design of IIR filter using windowing technique and verify the frequency response of the filter.

7. Design of FIR filter using any of the available methods and verify the frequency response of the filter

**PROCEDURE TO WORK WITH DSP PROCESSOR KIT USING CC STUDIO**

* Double click on CCStudio 6.0.1
* Workspace launcher window appear, specify path where to save the project then CC Studio default window will appear.
* Select project menu 🡪New CCS Project , then NEW CCS Project window opened there select the following options
* Target : C67XX / DSK6713
* Connection: Spectrum Digital DSK EVM ezdsp on board DSP emulator
* Project Name: <<Type Project Name here>>
* Project templates & Examples🡪Empty Project (with main.c)
* A New project window appears, then type the program code in the Editor window and save the program
* Build the project then .out file (Binary file) generated. IF errors there rectify and then build until zero errors. If warnings are there neglect.
* Debug and then Run. See the results in console window. Verify results with theoretical values.
* To view graph, Choose Tools menu-🡪Graph🡪Single Time🡪Give the parameters and view and plot the graph.

**1.Power and Energy of a given signal**

**AIM:** To find Power and (or) Energy of a given signal using DSP Processor kits and Code Composure Studio.

**REQUIREMENTS:**

DSP Processor Kit

Personal Computer with CCStudio S/W

**PROGRAM:**

#include<stdio.h>  
int main()  
{  
  int num,i,j,x[32];  
  long int sum=0;  
  printf("\nEnter the number of samples: ");  
  scanf("%d",&num);  
  printf("\nEnter samples: ");  
  for(j=0;j<num;j++)  
     scanf("%d",&x[j]);  
  for(i=0;i<num;i++)  
  {  
            sum+=x[i]\*x[i];  
  
  }  
  printf("\n the energy of above samples is\n %d",sum);  
  return 0;  
}

%Power of the given Signal

#include<stdio.h>  
int main(){  
  int num,i,j,x[32];  
                float  num1;  
  long int sum=0;  
  printf("\nEnter the number of samples: ");  
  scanf("%d",&num);  
  printf("\nEnter samples: ");  
  for(j=0;j<num;j++)  
     scanf("%d",&x[j]);  
  for(i=0;i<num;i++)  
  {  
            sum+=x[i]\*x[i];  
  
  }  
  num=num\*2;  
  num++;  
  num1 = sum / (float) num;  
  printf("\n the Average power of above samples is\n %.2f",num1);  
  return 0;  
}

**RESULTS:** Energy and Power estimation for duration sequences was performed using DSK – 6713 DSP processor Kit

1. **Convolution and Correlation of Signals**

**AIM:** To perform

(a) Linear Convolution of sequences

(b) Circular Convolution of sequences

(c) Auto correlation of a sequence

(d) Cross correlation of sequences using DSP Processor kit and Code Composure Studio.

**REQUIREMENTS:**

DSP Processor Kit

Personal Computer with CCStudio S/W

**PROGRAM:**

1. **LINEAR CONVOLUTION**

**/\* C- Program to implement Linear Convolution \*/**

#include <stdio.h>

#include <math.h>

int y[20];

main()

{

int m=6; /\* length of input sequence \*/

int n=6; /\* length of impulse sequence \*/

int i=0,j;

int x[15] = {1,2,34,5,6,0,0,0,0,0,0};

int h[15] = {1,2,34,5,6,0,0,0,0,0,0};

for (i=0; i<m+n-1; i++)

{

y[i]=0;

for (j=0; j<=i; j++)

y[i]+ = x[j] \* h[i-j];

}

printf("Liner convolution between x(n) and h(n) is y(n) = :\n");

for(i=0; i<m+n-1; i++)

printf("y[%d] = %d ", i, y[i] );

}

**OUTPUT:**

Linear Convolution between x(n) and h(n) is y(n) =

1 4 10 20 35 56 70 76 73 60 36

**RESULT:**

Linear convolution between two finite duration sequences is performed using DSK – 6713 DSP processor Kit

**PROGRAM:**

1. **CIRCULAR CONVOLUTION**

**/\* C- Program to implement Circular Convolution \*/**

#include<stdio.h>

#include<math.h>

int m, n, x[30], h[30], y[30], i, j, temp[30], K, X2[30], a[30];

void main()

{

printf("Enter the length of the first sequence\n");

scanf("%d",&m);

printf("Enter the length of the second sequence\n");

scanf("%d",&n);

printf("Enter the first sequence\n");

(i=0;i<m;i++)

scanf("%d",&x[i]);

printf("Enter the second sequence\n");

(j=0;j<n;j++)

scanf("%d",&h[j]);

if(m-n!=0) /\* If length of both sequences are not equal \*/

{

if(m>n)

{

for(i=n; i<m;i++)

h[i]=0;

n=m;

}

for(i=m; i<n;i++)

x[i]=0;

m=n;

}

y[0]=0;

a[0]=h[0];

for(j=1;j<n;j++) **/\* folding h(n) to h(-n) \*/**

a[j]=h[n-j];

**/\* CIRCULAR CONVOLUTION \*/**

for(i=0;i<n;i++)

y[0]+=x[i]\*a[i];

for(K=1;K<n;K++)

{

y[K]=0;

**/\* CIRCULAR SHIFT \*/**

for(j=1;j<n;j++)

x2[j]=a[j-1];

x2[0]=a[n-1];

for(i=0;i<n;i++)

{

a[i]=x2[i];

y[K]+=x[i]\*x2[i];

}

}

**/\* DISPLAYING THE RESULT \*/**

printf("The circular convolution is \n");

for(i=0;i<n;i++)

printf("%d\f",y[i]);

}

**OUTPUT:**

Enter the length of the first sequence 6

Enter the length of the second sequence 6

Enter the first sequence

x(n)= 1 2 3 4 5 6

Enter the second sequence

h(n) = 1 2 3 4 5 6

The circular convolution is y(n) =

71

80

83

80

71

56

**Result :** The circular convolution between two finite duration sequences was performed using DSK – 6713 DSP processor Kit.

**PROGRAM:**

**(C) AUTO CORRELATION**

**/\* C- Program to implement Auto Correlation\*/**

#include <stdio.h>

#include <math.h>

int y[15], x[10], h[10];

main()

{

int n=4;

int m=4;

int i=0,j=0;

int x[10]={1,2,3,4,0,0,0,0,0,0};

int h[10]={4,3,2,1,0,0,0,0,0,0};

for(i=0;i<m+n-1;i++)

{

y[i]=0;

for(j=0;j<=i;j++)

y[i]+=x[j]\*h[i-j];

}

printf("The Auto Correlation: \n");

for(i=0;i<m+n-1;i++)

printf("y[%d]=%d\n",i,y[i]);

}

**OUTPUT:**

The Auto Correlation:

4 11 20 30 20 11 4

**RESULT:** The Auto correlation of two finite length discrete time sequences was performed using DSP Processor kit and Code Composer Studio.

**PROGRAM:**

1. **CROSS CORRELATION**

**/\* C- Program to implement Cross Correlation\*/**

#include<stdio.h>

#include<math.h>

int y[20],x[10],h[10];

main()

{

int n=4;

int m=4;

int i=0,j=0;

int x[10]={1,2,3,4,0,0,0,0,0,0,0};

int h[10]={8,7,6,5,0,0,0,0,0,0,0};

for(i=0;i<m+n-1;i++)

{

y[i]=0;

for(j=0;j<=i;j++)

y[i]+=x[j]\*h[i-j];

}

printf("The Cross Correlation:\n");

for(i=0;i<m+n-1;i++)

printf("y[%d]=%d\n",i,y[i]);

}

**OUTPUT:**

The Cross Correlation :

8 23 44 70 56 39 20

**RESULT:** The cross correlation of two finite length discrete time sequences was performed successfully using DSP Processor Kit and Code Composer Studio.

1. **DTFT of a given Signal**

**AIM:** To compute DTFT (Discrete Time Fourier Transform) of a given signal using DSP Processor kit and Code Composure Studio.

**REQUIREMENTS:** DSP Processor Kit

Personal Computer with CCStudio S/W

**PROGRAM:**

#include<stdio.h>

#include<math.h>

int N,k,n,i;

float pi=3.1416,sum=0, j= sqrt(-1);

int x[32];

void main(void)

{

printf(" enter the length of the sequence\n");

scanf("%d",&N);

printf(" enter the sequence\n"); % Reading the Input sequence to which DTFT to be computed

for(i=0;i<N-1;i++)

scanf("%d",&x[i]);

sum[0]=x[0];

X[0] = sum[0];

for(k=1;k<N-1;k++)

{

sum[k] = sum[k-1]+exp(-j\*2\*pi\*k/N);

X[k] = sum[k];

}

mag\_X = abs(X);

ang

printf("X([%d])=\t%f\t+\t%fi\n",k,out\_real[k],out\_imag[k]);

}

}

**RESULTS:**DTFT for the given signal was computed using DSP Processor Kit and Code Composure Studio.

1. **N – point FFT algorithm**

**AIM:** To compute N- Point FFT for the given signal using DSP Processor kit and Code Composure Studio.

**REQUIREMENTS:**

DSP Processor Kit

Personal Computer with CCStudio S/W

**PROGRAM:**

#include <stdio.h>

#include <math.h>

#define PTS 64 //# of points for FFT

#define PI 3.14159265358979

typedef struct {float real,imag;} COMPLEX;

void FFT(COMPLEX \*Y, int n); //FFT prototype

float iobuffer[PTS]; //as input and output buffer

float x1[PTS]; //intermediate buffer

short i; //general purpose index variable

short buffercount = 0; //number of new samples in iobuffer

short flag = 0; //set to 1 by ISR when iobuffer full

COMPLEX w[PTS]; //twiddle constants stored in w

COMPLEX samples[PTS]; //primary working buffer

main()

{

for (i = 0 ; i<PTS ; i++) // set up twiddle constants in w

{

w[i].real = cos(2\*PI\*i/(PTS\*2.0)); //Re component of twiddle constants

w[i].imag =-sin(2\*PI\*i/(PTS\*2.0)); //Im component of twiddle constants

}

for (i = 0 ; i < PTS ; i++) //swap buffers

{

iobuffer[i] = sin(2\*PI\*10\*i/64.0);/\*10- > freq, 64 -> sampling freq\*/

samples[i].real=0.0;

samples[i].imag=0.0;

}

for (i = 0 ; i < PTS ; i++) //swap buffers

{

samples[i].real=iobuffer[i]; //buffer with new data

}

for (i = 0 ; i < PTS ; i++)

samples[i].imag = 0.0; //imag components = 0

FFT(samples,PTS); //call function FFT.c

for (i = 0 ; i < PTS ; i++) //compute magnitude

{

x1[i] = sqrt(samples[i].real\*samples[i].real+ samples[i].imag\*samples[i].imag);

}

} //end of main

// Function to calculate FFT

void FFT(COMPLEX \*Y, int N) //input sample array, # of points

{

COMPLEX temp1,temp2; //temporary storage variables

int i,j,k; //loop counter variables

int upper\_leg, lower\_leg; //index of upper/lower butterfly leg

int leg\_diff; //difference between upper/lower leg

int num\_stages = 0; //number of FFT stages (iterations)

int index, step; //index/step through twiddle constant

i = 1; //log(base2) of N points= # of stages

do

{

num\_stages +=1;

i = i\*2;

}while (i!=N);

leg\_diff = N/2; //difference between upper&lower legs

step = (PTS\*2)/N; //step between values in twiddle.h

for (i = 0;i < num\_stages; i++) //for N-point FFT

{

index = 0;

for (j = 0; j < leg\_diff; j++)

{

for (upper\_leg = j; upper\_leg < N; upper\_leg += (2\*leg\_diff))

{

lower\_leg = upper\_leg+leg\_diff;

temp1.real = (Y[upper\_leg]).real + (Y[lower\_leg]).real;

temp1.imag = (Y[upper\_leg]).imag + (Y[lower\_leg]).imag;

temp2.real = (Y[upper\_leg]).real - (Y[lower\_leg]).real;

temp2.imag = (Y[upper\_leg]).imag - (Y[lower\_leg]).imag;

(Y[lower\_leg]).real = temp2.real\*(w[index]).real

-temp2.imag\*(w[index]).imag;

(Y[lower\_leg]).imag = temp2.real\*(w[index]).imag

+temp2.imag\*(w[index]).real;

(Y[upper\_leg]).real = temp1.real;

(Y[upper\_leg]).imag = temp1.imag;

}

index += step;

}

leg\_diff = leg\_diff/2;

step \*= 2;

}

j = 0;

for (i = 1; i < (N-1); i++) //bit reversal for resequencing data

{

k = N/2;

while (k <= j)

{

j = j - k;

k = k/2;

}

j = j + k;

if (i<j)

{

temp1.real = (Y[j]).real;

temp1.imag = (Y[j]).imag;

(Y[j]).real = (Y[i]).real;

(Y[j]).imag = (Y[i]).imag;

(Y[i]).real = temp1.real;

(Y[i]).imag = temp1.imag;

}

}

return;

}

**OUTPUT:1. DFT via FFT**

Enter N value: 4

Enter Discrete time sequence x(n) is

x(0) = 1+j0

x(1) = 1+j0

x(2) = 1+j0

x(3) = 1+j0

Discrete Fourier Transform X(K):

X[0] = 4+j0

X[1] =0+j0

X[2] =0+j0

X[3] =0+j0

**RESULT:-**  N- FFT algorithm was executed by using DSP Processor kit and code composer studio.

**6.Design of IIR filter**

**AIM:** To design an IIR filter (Butterworth) and verify the frequency response of the filter using DSP Processor kit and Code Composure Studio.

**REQUIREMENTS:**

DSP Processor Kit

Personal Computer with CC Studio S/W

Cathode Ray Oscilloscope (CRO)

**PROGRAM:**

#include "C:\CCStudio\_v3.1\C6000\dsk6713\include\dsk6713.h"

#include "C:\CCStudio\_v3.1\C6000\dsk6713\include\dsk6713\_aic23.h"

const signed int filter\_Coeff[] =

{

2366,2366,2366,32767,-18179,13046

};

/\* Codec configuration settings \*/

DSK6713\_AIC23\_Config config = { \

0x0017, /\* 0 DSK6713\_AIC23\_LEFTINVOL Left line input channel volume \*/ \

0x0017, /\* 1 DSK6713\_AIC23\_RIGHTINVOL Right line input channel volume \*/\

0x00d8, /\* 2 DSK6713\_AIC23\_LEFTHPVOL Left channel headphone volume \*/ \

0x00d8, /\* 3 DSK6713\_AIC23\_RIGHTHPVOL Right channel headphone volume \*/ \

0x0011, /\* 4 DSK6713\_AIC23\_ANAPATH Analog audio path control \*/ \

0x0000, /\* 5 DSK6713\_AIC23\_DIGPATH Digital audio path control \*/ \

0x0000, /\* 6 DSK6713\_AIC23\_POWERDOWN Power down control \*/ \

0x0043, /\* 7 DSK6713\_AIC23\_DIGIF Digital audio interface format \*/ \

0x0081, /\* 8 DSK6713\_AIC23\_SAMPLERATE Sample rate control \*/ \

0x0001 /\* 9 DSK6713\_AIC23\_DIGACT Digital interface activation \*/ \

};

/\*

\* main() - Main code routine, initializes BSL and generates tone

\*/

void main()

{

DSK6713\_AIC23\_CodecHandle hCodec;

int l\_input, r\_input, l\_output, r\_output;

/\* Initialize the board support library, must be called first \*/

DSK6713\_init();

/\* Start the codec \*/

hCodec = DSK6713\_AIC23\_openCodec(0, &config);

DSK6713\_AIC23\_setFreq(hCodec, 3);

while(1)

{ /\* Read a sample to the left channel \*/

while (!DSK6713\_AIC23\_read(hCodec, &l\_input));

/\* Read a sample to the right channel \*/

while (!DSK6713\_AIC23\_read(hCodec, &r\_input));

l\_output=IIR\_FILTER(&filter\_Coeff ,l\_input);

r\_output=l\_output;

/\* Send a sample to the left channel \*/

while (!DSK6713\_AIC23\_write(hCodec, l\_output));

/\* Send a sample to the right channel \*/

while (!DSK6713\_AIC23\_write(hCodec, r\_output));

}

/\* Close the codec \*/

DSK6713\_AIC23\_closeCodec(hCodec);

}

signed int IIR\_FILTER(const signed int \* h, signed int x1)

{

static signed int x[6] = { 0, 0, 0, 0, 0, 0 }; /\* x(n), x(n-1), x(n-2). Must be static \*/

static signed int y[6] = { 0, 0, 0, 0, 0, 0 }; /\* y(n), y(n-1), y(n-2). Must be static \*/

int temp=0;

temp = (short int)x1; /\* Copy input to temp \*/

x[0] = (signed int) temp; /\* Copy input to x[stages][0] \*/

temp = ( (int)h[0] \* x[0]) ; /\* B0 \* x(n) \*/

temp += ( (int)h[1] \* x[1]); /\* B1/2 \* x(n-1) \*/

temp += ( (int)h[1] \* x[1]); /\* B1/2 \* x(n-1) \*/

temp += ( (int)h[2] \* x[2]); /\* B2 \* x(n-2) \*/

temp -= ( (int)h[4] \* y[1]); /\* A1/2 \* y(n-1) \*/

temp -= ( (int)h[4] \* y[1]); /\* A1/2 \* y(n-1) \*/

temp -= ( (int)h[5] \* y[2]); /\* A2 \* y(n-2) \*/

/\* Divide temp by coefficients[A0] \*/

temp >>= 15;

if ( temp > 32767 )

{

temp = 32767;

}

else if ( temp < -32767)

{

temp = -32767;

}

y[0] = temp ;

/\* Shuffle values along one place for next time \*/

y[2] = y[1]; /\* y(n-2) = y(n-1) \*/

y[1] = y[0]; /\* y(n-1) = y(n) \*/

x[2] = x[1]; /\* x(n-2) = x(n-1) \*/

x[1] = x[0]; /\* x(n-1) = x(n) \*/

/\* temp is used as input next time through \*/

return (temp<<2);

}

**RESULTS:** Thus an IIR Filter (Butterworth) was designed and its frequency response characteristics are verified using DSP Processor kit and Code Composure Studio.

**7.FIR Filter using Windowing Technique**

**AIM:** To design FIR filter using windowing technique and verify the frequency response of the filter using DPS Processor Kit and Code composure Studio

**REQUIREMENTS:**

DSP Processor Kit

Personal Computer with CCStudio S/W

Cathode Ray Oscilloscope (CRO)

**PROGRAM:**

#include "C:\CCStudio\_v3.1\C6000\dsk6713\include\dsk6713.h"

#include "C:\CCStudio\_v3.1\C6000\dsk6713\include\dsk6713\_aic23.h"

float filter\_Coeff[] = {

0.000000,-0.001591,-0.002423,0.000000,0.005728,

0.011139,0.010502,-0.000000,-0.018003,-0.033416,-0.031505,0.000000,

0.063010,0.144802,0.220534,0.262448,0.220534,0.144802,0.063010,0.000000,

-0.031505,-0.033416,-0.018003,-0.000000,0.010502,0.011139,0.005728,

0.000000,-0.002423,-0.001591,0.000000

};

static short in\_buffer[100];

DSK6713\_AIC23\_Config config = { \

0x0017, /\* 0 DSK6713\_AIC23\_LEFTINVOL Left line input channel volume \*/ \

0x0017, /\* 1 DSK6713\_AIC23\_RIGHTINVOL Right line input channel volume \*/\

0x00d8, /\* 2 DSK6713\_AIC23\_LEFTHPVOL Left channel headphone volume \*/ \

0x00d8, /\* 3 DSK6713\_AIC23\_RIGHTHPVOL Right channel headphone volume \*/ \

0x0011, /\* 4 DSK6713\_AIC23\_ANAPATH Analog audio path control \*/ \

0x0000, /\* 5 DSK6713\_AIC23\_DIGPATH Digital audio path control \*/ \

0x0000, /\* 6 DSK6713\_AIC23\_POWERDOWN Power down control \*/ \

0x0043, /\* 7 DSK6713\_AIC23\_DIGIF Digital audio interface format \*/ \

0x0081, /\* 8 DSK6713\_AIC23\_SAMPLERATE Sample rate control \*/ \

0x0001 /\* 9 DSK6713\_AIC23\_DIGACT Digital interface activation \*/ \

};

/\*

\* main() - Main code routine, initializes BSL and generates tone

\*/

void main()

{

DSK6713\_AIC23\_CodecHandle hCodec;

Uint32 l\_input, r\_input,l\_output, r\_output;

/\* Initialize the board support library, must be called first \*/

DSK6713\_init();

/\* Start the codec \*/

hCodec = DSK6713\_AIC23\_openCodec(0, &config);

DSK6713\_AIC23\_setFreq(hCodec, 1);

while(1)

{ /\* Read a sample to the left channel \*/

while (!DSK6713\_AIC23\_read(hCodec, &l\_input));

/\* Read a sample to the right channel \*/

while (!DSK6713\_AIC23\_read(hCodec, &r\_input));

l\_output=(Int16)FIR\_FILTER(&filter\_Coeff ,l\_input);

r\_output=(Int16)FIR\_FILTER(&filter\_Coeff ,r\_input);

/\* Send a sample to the left channel \*/

while (!DSK6713\_AIC23\_write(hCodec, l\_output));

/\* Send a sample to the right channel \*/

while (!DSK6713\_AIC23\_write(hCodec, r\_output));

}

/\* Close the codec \*/

DSK6713\_AIC23\_closeCodec(hCodec);

}

signed int FIR\_FILTER(float \* h, signed int x)

{

int i=0;

signed long output=0;

in\_buffer[0] = x; /\* new input at buffer[0] \*/

for(i=31;i>0;i--)

in\_buffer[i] = in\_buffer[i-1]; /\* shuffle the buffer \*/

for(i=0;i<31;i++)

output = output + h[i] \* in\_buffer[i];

return(output);

}

**RESULTS:**FIR filter was designed and its frequency response characteristics are verified using DSP Processor Kit and Code Composure Studio.