Lab Manual

ECPC 303

Digital Signal Processing

for

Bachelor of Technology

in

Electronics & Communication Engineering

Department of Electronics & Communication Engineering

National Institute of Technology

Kurukshetra-136119

VISION

The Department of Electronics & Communication Engineering shall strive to impart state-of-the-art Electronics and Communication Engineering Education and Research responsive to global challenges.

MISSION

1) To prepare students with strong theoretical and practical knowledge by imparting quality education.
2) To produce comprehensively trained and innovative graduates in Electronics and Communication Engineering through hands-on practice and research to encourage them for entrepreneurship.
3) To inculcate team work spirit and professional ethics in students

Programme Educational Objectives (PEOs)

The graduates of the Electronics and Communication Engineering Program will:

1) Have a lead and successful role in their professional career.
2) Be able to analyze real life problems and design socially accepted and economically viable solutions in the Electronics and Communication Engineering area.
3) Be capable of lifelong learning and professional development by pursuing higher education and participation in research and development activities.
4) Have appropriate human and technical communication skills to be a good team-member/leader and responsible human being.

Program Specific Outcomes (PSOs)

At the end of the program, the student will:

1) Clearly understand the fundamental concepts of Electronics and Communication Engineering.
2) Formulate the real-life problems and develop solutions in the area of semiconductor technology, signal processing and communication systems.
3) Possess the skills to communicate effectively in both oral and written forms, demonstrating the practice of professional ethics, and responsive to societal and environmental needs

Programme Outcomes (POs)

After successful completion of B. Tech. (Electronics & Communication Engineering) program, the student will be able to:

1) Apply knowledge of Mathematics, Science and Engineering to solve technical problems in the domain of Electronics and Communication Engineering.
2) Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3) Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4) Use research-based knowledge and methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5) Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6) Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7) Understand the impact of the professional engineering solutions in societal and environmental contexts, demonstrate the knowledge of, and need for sustainable development.
8) Apply ethical principles and commit to professional ethics, responsibilities, and norms of the engineering practice.
9) Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10) Communicate effectively on complex engineering activities with the engineering community and with society, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11) Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12) Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ECPC 303

Digital Signal Processing Lab

	L	T	P
	0	0	2

Course Objectives	This lab emphasizes intuitive understanding and practical implementations of the theoretical concepts. Students will be familiarized with the most important methods in DSP, including digital filter design, transform-domain processing. Further, in this lab students will attain the knowledge about implementation of various filters and understand various tools like FDA Tool and Code Compose Studios for filter implementation in MATLAB.
Course Outcomes	At the end of the course, the students will be able to: 1. Simulate the function to compute DTFT of a finite length signal using MATLAB. 2. Compute Inverse Z transform function and convolution. 3. Design the band pass filter, band reject filter, low pass filter and high pass filters. 4. Implement FIR/IIR filters using DSP Kit.

B.Tech Electronics and Communication Engineering
DSP LAB (ECPC 303)

List of Experiments:

1. (a) Generate square wave of frequency (input from keyboard); from its harmonics of sinusoidal components.
   (b) Use the pause function from MATLAB to demonstrate the effect of addition of harmonics to its fundamental frequency.
2. Find the inverse Z-transform of the following:

$$X(z)=\frac{\left(1-1.22346z^{-1}+z^{-2}\right)\left(1-0.437833z^{-1}+z^{-2}\right)\left(1+z^{-1}\right)}{\left(1-1.433509z^{-1}\right)\left(1-1.293601z^{-1}+0.556929z^{-2}\right)\left(1-0.612159z^{-1}\right)}$$

3. (a) Compute Circular convolution of x(n)= {1,2,1,0,2,1}; n >=0 and y(n)= {2,4,0,1,1,0}; n>=0
   (b) A signal sequence x (n) = {1, 1, 1} is applied to a system with an unknown impulse response h (n). The observed output is y(n)= {1,4,8,10,8,4,1}. Write a program to find h (n).
4. A linear phase, bandpass FIR filter is required to meet the following specification:

Passband	12-16 kHz
Transition width	2 kHz
Passband ripple	1 db
Stopband attenuation	45 db
Sampling Frequency	50 kHz

   Estimate the filter length and use the optimal method to determine the filter coefficient and hence plot the magnitude-frequency response. Compare the pass bands stopband ripples of the filter with the specified values.
5. Design an IIR filter with following specifications:

Lower passband	0-50 Hz
Upper passband	450-500 Hz
Stop band	200-300 Hz
Passband ripple	3 db
Stopband attenuation	20 db
Sampling frequency	1 kHz

6. Implement the above filters using SIMULINK and verify the performance.
7. Implement an FIR filter on the DSP kit by using the FDA tool and code composer studio.
8. Implement an IIR filter on the DSP kit by using the FDA tool and code composer studio.

EXPERIMENT -1

(a) Generate square wave of frequency (input from keyboard) from its harmonics of sinusoidal components.

(b) Use the pause function from MATLAB to demonstrate the effect of addition of harmonics to its fundamental.

EQUIPMENT REQUIRED:

Hardware required	**Software Required**
PC	MATLAB

THEORY:

It has been found that *any* repeating, non-sinusoidal waveform can be equated to a combination of DC voltage, sine waves, and/or cosine waves (sine waves with a 90 degrees phase shift) at various amplitudes and frequencies. In particular, it has been found that square waves are mathematically equivalent to the sum of a sine wave at that same frequency, plus an infinite series of odd-multiple frequency sine waves at diminishing amplitude.

When two sinusoids are added together the result depends upon their amplitude, frequency and phase. The effects are easiest to observe when only one of these is varied between the two sinusoids being added. In the simplest case, when two sinusoids with the same frequency and phase but with different amplitudes are added together the result is a sinusoid whose amplitude is the sum of the originals and who's frequency and phase remain unchanged.

When the two sinusoids have different frequencies, the result is more complicated. The new signal is no longer a sinusoid since it doesn't follow the simple up and down pattern. Instead, we see the higher frequency sinusoid as a `ripple' superimposed on the lower frequency sinusoid (Fig. 1). The frequency of the resulting signal will be the lower of the two original frequencies. In Fig. 1 We can see that the resulting signal goes through two and a half cycles (that is it repeats itself this many times) just like the second sinusoid. The amplitude of the resulting signal is the sum of the originals and the phase is unchanged.

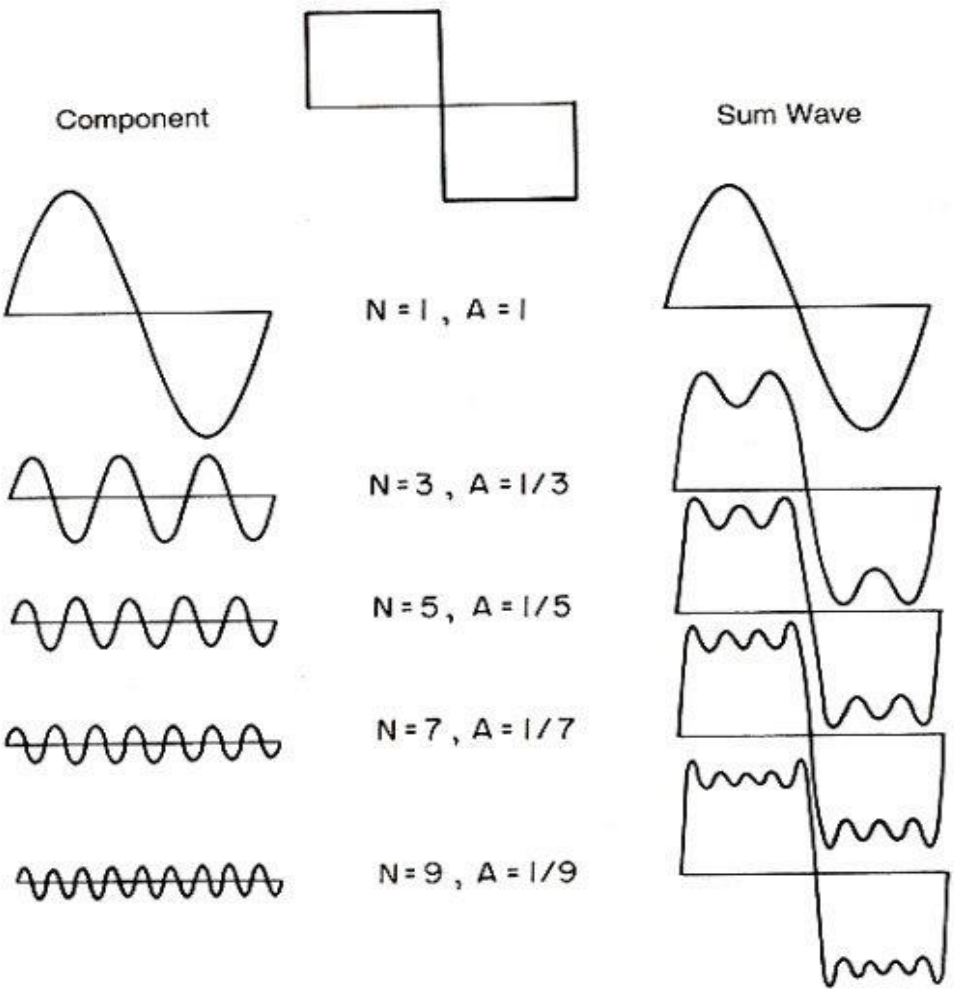


Fig. 1. Square wave as sum of sinusoidal waves.

The Fourier series expansion for a square wave is made up of a sum of odd harmonics. Using Fourier series expansion with cycle frequency f over time t, we can represent an ideal square wave with an amplitude of 1 as infinite series of the form

$$x_{square}(t) = \frac{4}{\pi}\sum_{k=1}^{\infty}\frac{\sin(2\pi(2k-1)ft)}{2k-1}$$

$$= \frac{4}{\pi}((\sin(2\pi ft) + \frac{1}{3}(\sin(6\pi ft) + \frac{1}{5}(\sin(10\pi ft) + ...)$$

An ideal mathematical square wave changes between the high and the low state instantaneously, and without under- or over-shooting. This is impossible to achieve in physical systems, as it would require infinite bandwidth. Square wave in physical systems have only finite bandwidth, and often exhibit ringing effects similar to those of the Gibbs phenomenon, or ripple effect similar to those of the σ approximations.

Program:

(a) Without Pause function

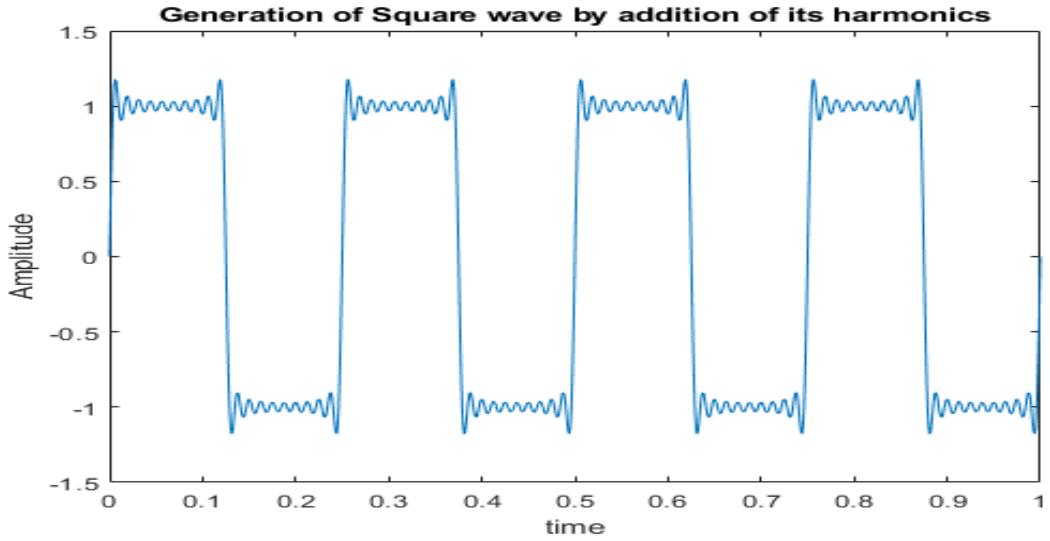
```
clc;                                    % clear command window
clear all;                              % clear workspace
close all;                              % delete all previous figures
freq=input('Enter the frequency_');
n=input('Enter the number of harmonics_');
t=0:0.001:1;                            %time
sq_wave=0;

for k=1:1:n
   harmonic=(4/pi)*(1/(2*k-1))*sin(2*pi*freq*(2*k-1)*t);
   sq_wave=sq_wave+harmonic;
end

plot(t,sq_wave);
title('Generation of Square wave by addition of its harmonics');
xlabel('time');
ylabel('Amplitude');
```

(b) With Pause function

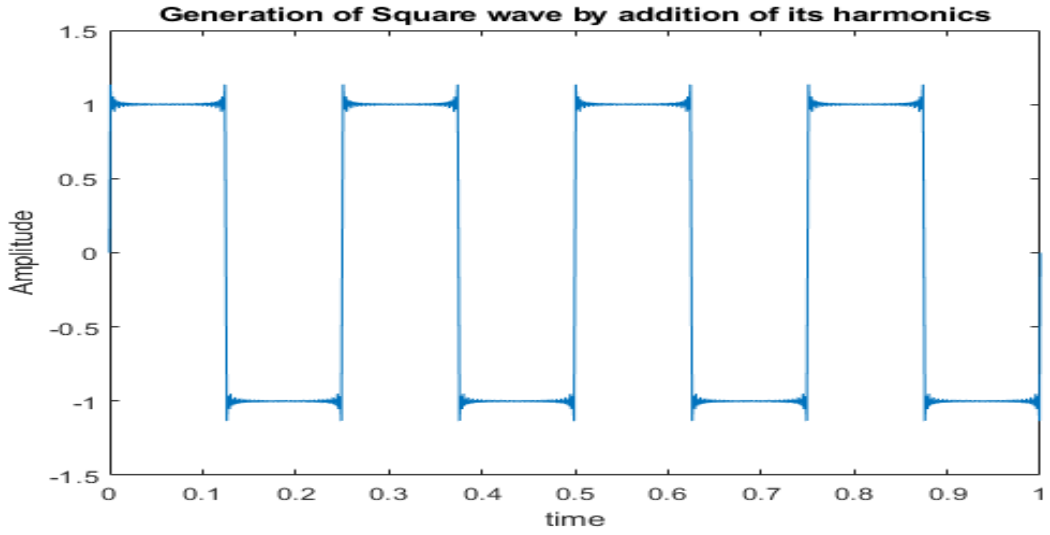
```
clc;
close all;
freq=input('Enter the frequency');
n=input('Enter the number of harmonics');
t=0:0.001:1;
sq_wave=0;
title('Generation of Square wave by addition of its harmonics with pause
function ');

for k=1:1:n
   harmonic=(4/pi)*(1/(2*k-1))*sin(2*pi*freq*(2*k-1)*t);
   sq_wave=sq_wave+harmonic;
   plot(t,sq_wave);
   pause(0.5)        % pause time is 0.5 seconds
end
```

OUTPUT:

Frequency=4 Hz
Number of harmonics=10



Frequency=4 Hz
Number of harmonics=50



CONCLUSION:

It is observed that as number of harmonics increase the number of ripples increases and the square wave generated becomes smoother.

QUESTIONS:

1. Define Fourier series.
2. Explain the Gibbs phenomenon.
3. Write the Fourier series expression for triangular and sawtooth waveforms.

EXPERIMENT -2

Find the inverse Z-transform of the following:

$$X(z)=\frac{\left(1-1.22346z^{-1}+z^{-2}\right)\left(1-0.437833z^{-1}+z^{-2}\right)\left(1+z^{-1}\right)}{\left(1-1.433509z^{-1}\right)\left(1-1.293601z^{-1}+0.556929z^{-2}\right)\left(1-0.612159z^{-1}\right)}$$

EQUIPMENT REQUIRED:

Hardware required	**Software Required**
PC	MATLAB

THEORY:

In mathematics and signal processing, the Z-transform converts a discrete-time signal, which is a sequence of real or complex numbers, into a complex frequency domain representation. It can be considered as a discrete-time equivalent of the Laplace transform. The inverse z-transform is denoted by $x(n)=Z^{-1}[X(z)]$. The inverse z-transform converts the complex frequency domain signal back to discrete-time signal.

Commands used

(1) [b,a] = sos2tf(sos) returns the transfer function coefficients of a discrete-time system described in second-order section form by `sos`.

Second-order section representation, specified as a matrix. sos is an *L*-by-6 matrix

$$\text{SOS}=\begin{bmatrix} b_{01} & b_{11} & b_{21} & 1 & a_{11} & a_{21} \\ b_{02} & b_{12} & b_{22} & 1 & a_{12} & a_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ b_{0L} & b_{1L} & b_{2L} & 1 & a_{1L} & a_{2L} \end{bmatrix}$$

whose rows contain the numerator and denominator coefficients b_{ik} and a_{ik} of the second-order sections of $H(z)$:

$$H(z)=g\prod_{k=1}^{L}H_k(z)=g\prod_{k=1}^{L}\frac{b_{0k}+b_{1k}z^{-1}+b_{2k}z^{-2}}{1+a_{1k}z^{-1}+a_{2k}z^{-2}}.$$

(2) [b,a] Transfer function coefficients, returned as row vectors. `b` and `a` contain the numerator and denominator coefficients of $H(z)$ stored in descending powers of z:

$$H(z)=\frac{B(z)}{A(z)}=\frac{b_1+b_2z^{-1}+\cdots+b_{n+1}z^{-n}}{a_1+a_2z^{-1}+\cdots+a_{m+1}z^{-m}}.$$

(3) [r,p,k] = residue(b,a) finds the residues, poles, and direct term of a Partial Fraction Expansion of the ratio of two polynomials, where the expansion is of the form

$$\frac{b(s)}{a(s)} = \frac{b_m s^m + b_{m-1} s^{m-1} + \ldots + b_1 s + b_0}{a_n s^n + a_{n-1} s^{n-1} + \ldots + a_1 s + a_0} = \frac{r_n}{s - p_n} + \ldots + \frac{r_2}{s - p_2} + \frac{r_1}{s - p_1} + k(s).$$

The inputs to residue are vectors of coefficients of the polynomials b = [b_m ... b_1 b_0] and a = [a_n ... a_1 a_0]. The outputs are the residues r = [r_n ... r_2 r_1], the poles p = [p_n ... p_2 p_1], and the polynomial k. For most textbook problems, k is 0 or a constant.

Program:

```
clear all;
clc;
close all;
arr=[1 -1.22346 1 1 -1.433509 0; 1 -0.437833 1 1 -1.293601 0.556929; 1 1 0 1 -0.612159 0];
[x,y]=sos2tf(arr);
x
y
n=length(x);
m=length(y);
n
m
[r,p,k]=residue(x,y);
r
p
k
l=length(r)
l
for i=1:10
   sum=0;
   j=1;
   while j<=l
      sum=r(j)*p(j)^(i-1)+sum;
      j=j+1;
   end
   h(i)=sum;
end
h
```

OUTPUT:

```
x =    1.0000   -0.6613    0.8744    0.8744   -0.6613    1.0000
y =    1.0000   -3.3393    4.0807   -2.2745    0.4887         0

n = 6
m = 6

r =
   8.6172 + 0.0000i
   3.9507 + 2.6892i
   3.9507 - 2.6892i
 -15.8869 + 0.0000i
   2.0461 + 0.0000i

p =
   1.4335 + 0.0000i
   0.6468 + 0.3723i
   0.6468 - 0.3723i
   0.6122 + 0.0000i
   0.0000 + 0.0000i
k = 1

l = 5

h =2.6780    5.7361   11.3752   19.5181   31.4948   48.5906   72.5687
    106.1360   153.4080   220.4651
```

CONCLUSION:

The inverse Z-Transform of the given equation has thus been calculated using the partial fraction method.

QUESTIONS:

1. Define one-sided and two-sided Z-transform.
2. What is the region of convergence (ROC)?
3. State the Initial value and final value theorem with regard to Z-transform.
4. Define Z-transform of unit step signal.
5. What are the different methods available for inverse Z-transform?

EXPERIMENT-3

a) Compute circular convolution of x(n) = {1,2,1,0,2,1}; n >= 0 and y(n) = {2,4,0,1,1,0}; n >= 0.
b) A signal sequence x(n) = {1,1,1} is applied to a system with an unknown impulse response h(n). The observed output is y(n) = {1,4,8,10,8,4,1}. Write a program to find h(n).

EQUIPMENT REQUIRED:

Hardware required	**Software Required**
PC	MATLAB

THEORY:

Convolution is a formal mathematical operation, just as multiplication, Addition, and integration. Addition takes two numbers and produces a third number, while convolution takes two signals and produces a third signal. 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. Convolution is an integral concatenation of two signals. It has many applications in numerous areas of signal processing. The most popular application is the determination of the output signal of a linear time-invariant system by convolving the input signal with the impulse response of the system. The linear convolution of two discrete-time signals x(n) and h(n) is defined as a system x(n) in the input signal, h(n) is the impulse response of the signal and y(n) is the output signal. Linear convolution obeys the properties: a) Commutative property b) Associative property c) Distributive property. The circular convolution (or periodic convolution) of two finite signal x(n) and h(n).

$$y(n) = x(n) * h(n) = \Sigma_{k=-\infty}^{\infty} x(k)\,h(n-k)$$

$$w[n] = x[n] \textcircled{N} h[n].$$

$$w(n) = \Sigma_{m=0}^{N-1} x(m)\,h((n-m))_N$$

Circular convolution:

1. Start
2. Enter the input sequence x(n) and h(n).
3. Compute the length L of x(n) and M of h(n) and find length N = max(L,M) of y(n).
4. Pad zeros to the end of x(n) and h(n) to keep both sequences the same length.
5. Initialize a loop of index i with count value of length N.
6. Initialize another loop of index j count value up to N.
7. For circular shifting of h(n), compute index for h, k = i-j+1 and if j is negative k =N + k.
8. Perform convolution y(i) = y(i) + x(j) * h(k).
9. Continue step 7 up to the inner loop count value.
10. Continue steps 7 to 9 up to the outer loop count value.
11. Display the result y(n).

Program:

(a) For Circular convolution

```
clc;

clear all;

first=input('Enter first sequence: ');

second=input('Enter second sequence: ');

len_first=length(first);

len_second=length(second);

N=max(len_first,len_second);

x=[first zeros(1,N-len_first)];                    %Padding Zeros

y=[second zeros(1,N-len_second)];                  %Padding Zeros

cir_conv=ifft ( fft(x,N).* fft(y,N) ,N);

disp('The resultant of convolution is');

cir_conv
```

Output:

```
Enter first sequence:      [1 2 1 0 2 1]

Enter second sequence:  [2 4 0 1 1 0]

The resultant of convolution is

cir_conv =  7    10    13     6     7    13
```

(b) For finding impulse response

```
clc;
close all;
clear all;
signal=input('Enter signal sequence: ');
output=input('Enter output sequence: ');
len_signal=length(signal);
len_output=length(output);
N=max(len_signal,len_output);
x=[signal zeros(1,N-len_signal)];
y=[output zeros(1,N-len_output)];
impulse_response=  ifft( fft(y,N) ./ fft(x,N) ,N);
impulse_response
xlabel('Samples');
ylabel('Amplitude');
title('Impulse response');
```

OUTPUT:

```
Enter signal sequence: [1 2 1 0 2 1]
Enter output sequence: [7 10 13 6 7 13]
impulse_response =    2.0000    4.0000    0.0000    1.0000    1.0000    0.0000
```

QUESTIONS:

1. What is DFT?
2. Difference between circular and linear convolution.

EXPERIMENT-4

To design a linear phase band pass filter with following specifications

Passband = 12-16 KHz

Transition Width = 2 KHz

Passband Ripple = 1 dB

Stopband Attenuation = 45 dB

Sampling Frequency = 50 KHz

Estimate the filter length and use the optimal method to determine the filter coefficients and hence plot magnitude of frequency response. Compare the passband and stopband values of filters with specified values.

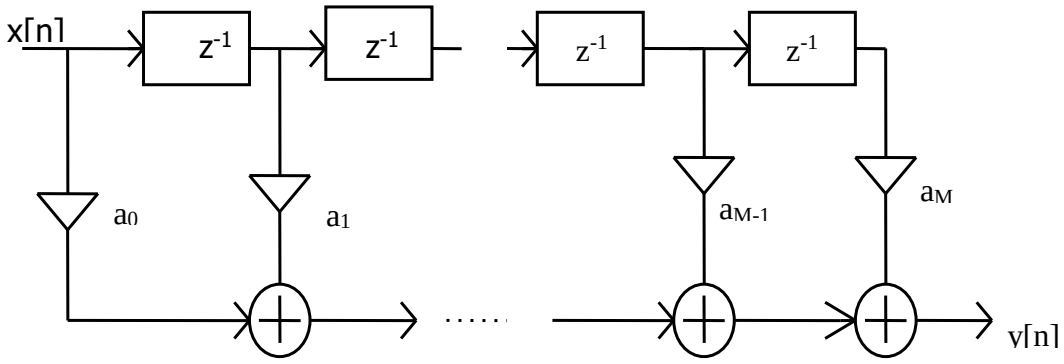
EQUIPMENT REQUIRED:

Hardware required	**Software Required**
PC	MATLAB

THEORY:

An FIR digital filter of order M may be implemented by programming the signal-flow-graph shown below. Its difference equation is:

$$y[n] = a_0x[n] + a_1x[n-1] + a_2x[n-2] + ... + a_Mx[n-M]$$



Its impulse-response is {..., 0, ..., $\underline{a_0}$, a_1, a_2,..., a_M, 0, ...} and its frequency-response is the DTFT of the impulse-response, i.e.

$$H(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\Omega n} = \sum_{n=0}^{M} a_n e^{-j\Omega n}$$

Now consider the problem of choosing the multiplier coefficients. a_0, a_1,..., a_M such that $H(e^{j\Omega})$ is close to some desired or target frequency-response $H'(e^{j\Omega})$ say. The inverse DTFT of $H'(e^{j\Omega})$ gives the required impulse-response:

$$h'[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H'(e^{j\Omega})e^{j\Omega n}d\Omega$$

The methodology is to use the inverse DTFT to get an impulse-response {h′[n]} and then realize some approximation to it. Note that the DTFT formula is an integral, it has complex numbers and the range of integration is from -Ω to Ω, so it involves negative frequencies. These can be designed almost as easily as low-pass as long as you remember to define the required gain-response G′(Ω) correctly from -π to +π; i.e. make G′(-Ω) = G′(Ω). For example, a band-pass filter with pass-band from $F_S/8$ to $F_S/4$ would have the following gain response ideally :-

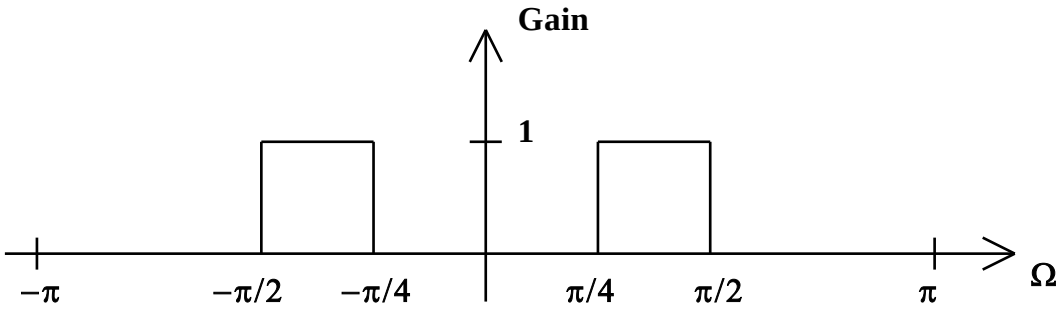


Fig. 2. Gain response of ideal band-pass filter

Taking φ′(Ω) = 0 for all Ω initially as before, we obtain:

$$H'(e^{j\Omega}) = \begin{cases} 0 & : \quad |\Omega| < \pi/4 \\ 1 & : \quad \pi/4 \le |\Omega| \le \pi/2 \\ 0 & : \quad \pi/2 < |\Omega| < \pi \end{cases}$$

and applying the inverse DTFT gives the following formula (not forgetting negative values of Ω:

$$h'[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H'(e^{j\Omega})e^{-j\Omega n}d\Omega = \frac{1}{2\pi}\int_{-\pi/2}^{-\pi/4} 1e^{-j\Omega n}d\Omega + \frac{1}{2\pi}\int_{\pi/4}^{\pi/2} 1e^{-j\Omega n}d\Omega$$

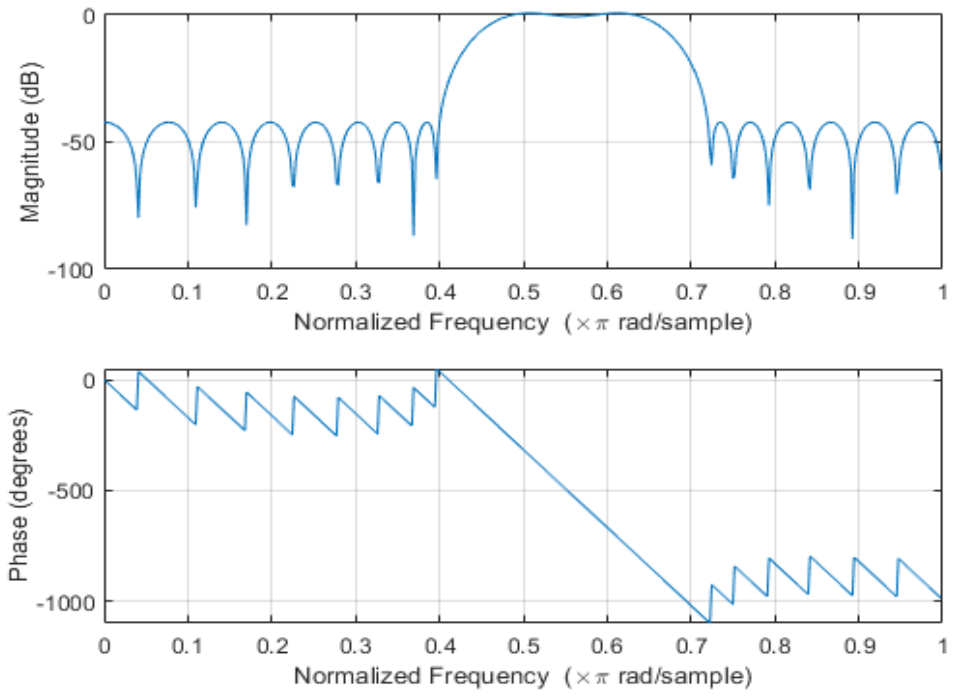
Evaluating the integrals in this expression will produce a nice formula for h′[n].

Truncating symmetrically, windowing and delaying this sequence, as for the low-pass case earlier, produces the causal impulse-response of an FIR filter of the required order. This will be a linear phase for the same reasons as in the low-pass case. Its impulse-response will be symmetric about n=M/2 when M is the order. MATLAB is able to design high-pass, band-pass and band-stop linear phase FIR digital filters by this method

Program:

```
clc;

rp=1;

rs=45;

fs=50000;

f= [10000 12000 16000 18000];

a= [0 1 0];

dev=[10^(-rs/20) (10^(rp/20)-1)/(10^(rp/20) +1) 10^(-rs/20)];

[n, fo, ao, w] =firpmord (f, a, dev, fs);

disp ('length of filter is');

disp(n);

b=firpm (n, fo, ao, w);

freqz(b);

title('Bandpass');
```

OUTPUT:



CONCLUSION:

Linear phase Band pass filter has been thus designed.

QUESTIONS:

1. What are the differences between FIR and IIR filters?
2. Explain windowing technique.
3. What is the DC gain of FIR filter?
4. What are linear phase filters?

EXPERIMENT-5

To design an IIR filter with following specifications

Lower passband: 0-50 Hz
Upper passband: 450-500 Hz
Stop band: 200-300 Hz
Passband ripple: 3 dB
Stop band Attenuation: 20 dB
Sampling Frequency: 1 KHz

EQUIPMENT REQUIRED:

Hardware required	**Software Required**
PC	MATLAB

THEORY:

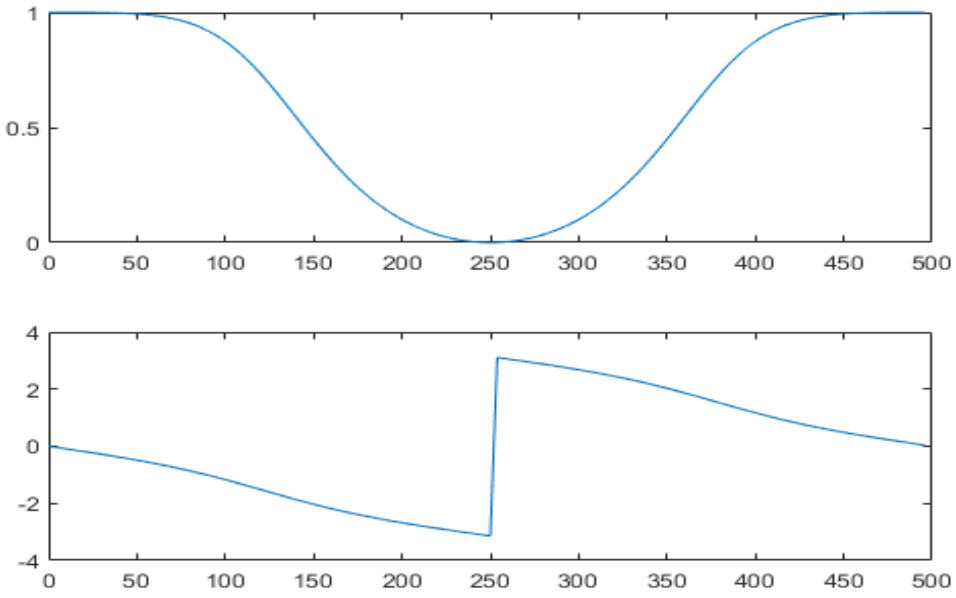
IIR is property applying to many linear time invariant systems. Common examples of linear time invariant systems are most electronic and digital filters. Systems with this property are known as IIR systems or IIR filters, and are distinguished by having an impulse response which does not become exactly zero past a certain point, but continuous indefinitely. This is in contrast to a finite impulse response in which the impulse response h(t) does become exactly zero at times t>T for some finite T, thus being of finite duration.

In practice, the impulse response, even of IIR systems, usually approaches zero and can be neglected past a certain point. However, the physical systems which give to IIR or FIR responses are dissimilar, and therein lies the importance of the distinction. For instance, analog electronic filters composed of resistors, capacitors, and /Or inductors are generally IIR filters.

CODE:

```
clc;
rp=3;
rs=20;
fs=1000;                    % sampling frequency
wp=[50 450]/(fs/2);         % normalizing passband frequency
ws=[200 300]/(fs/2);        % normalizing stopband frequency
[n,wn]=buttord(wp,ws,rp,rs);
[b,a]=butter(n,wn,'stop');
[h,d]=freqz(b,a,128,fs);
subplot(2,1,1);
plot(d,abs(h));
g=angle(h);
subplot(2,1,2);
plot(d,g);
```

OUTPUT:



CONCLUSION:

IIR Band Stop filter has thus been realized.

QUESTIONS:

1. Define IIR Filter
2. What is the region of convergence (ROC)?

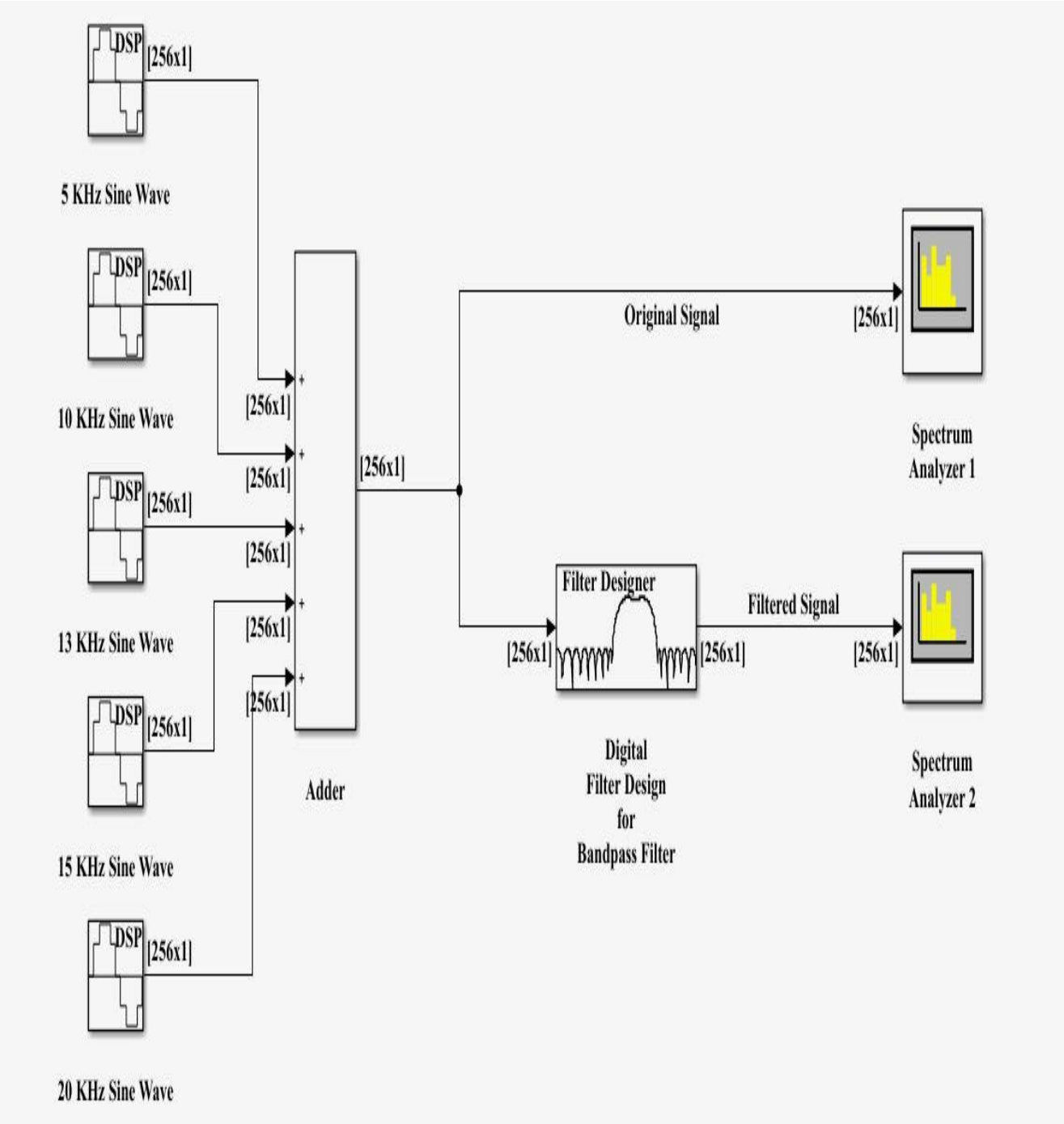
EXPERIMENT-6

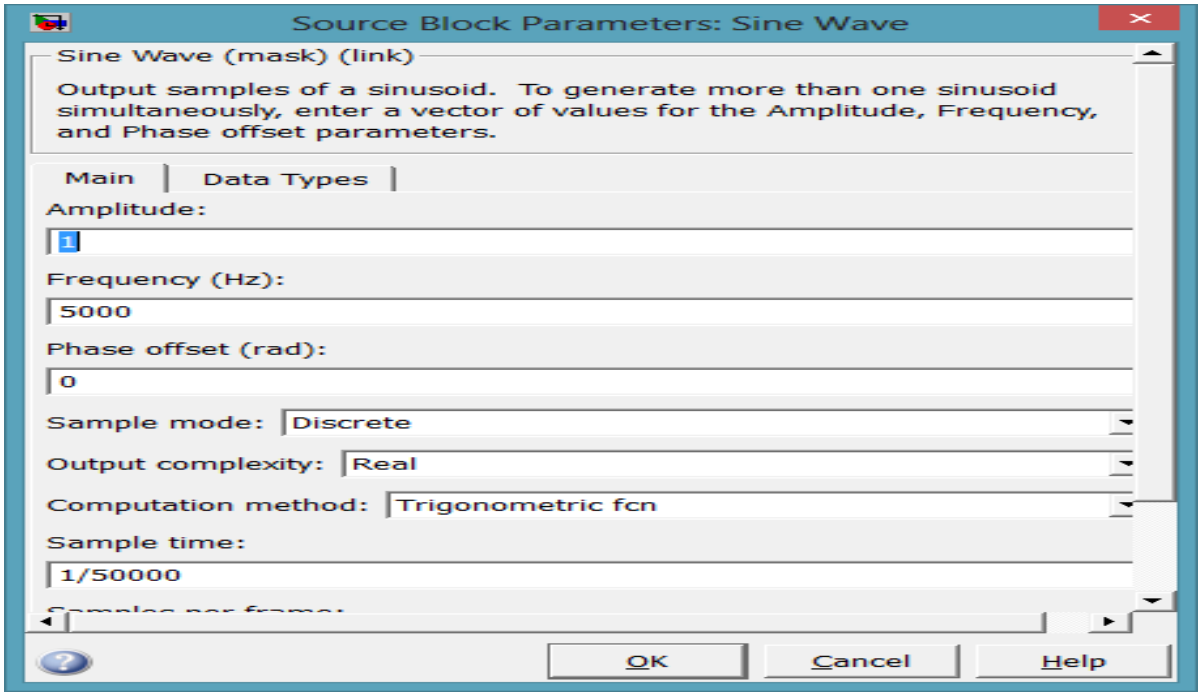
To implement the above filters using Simulink and verify the performances.

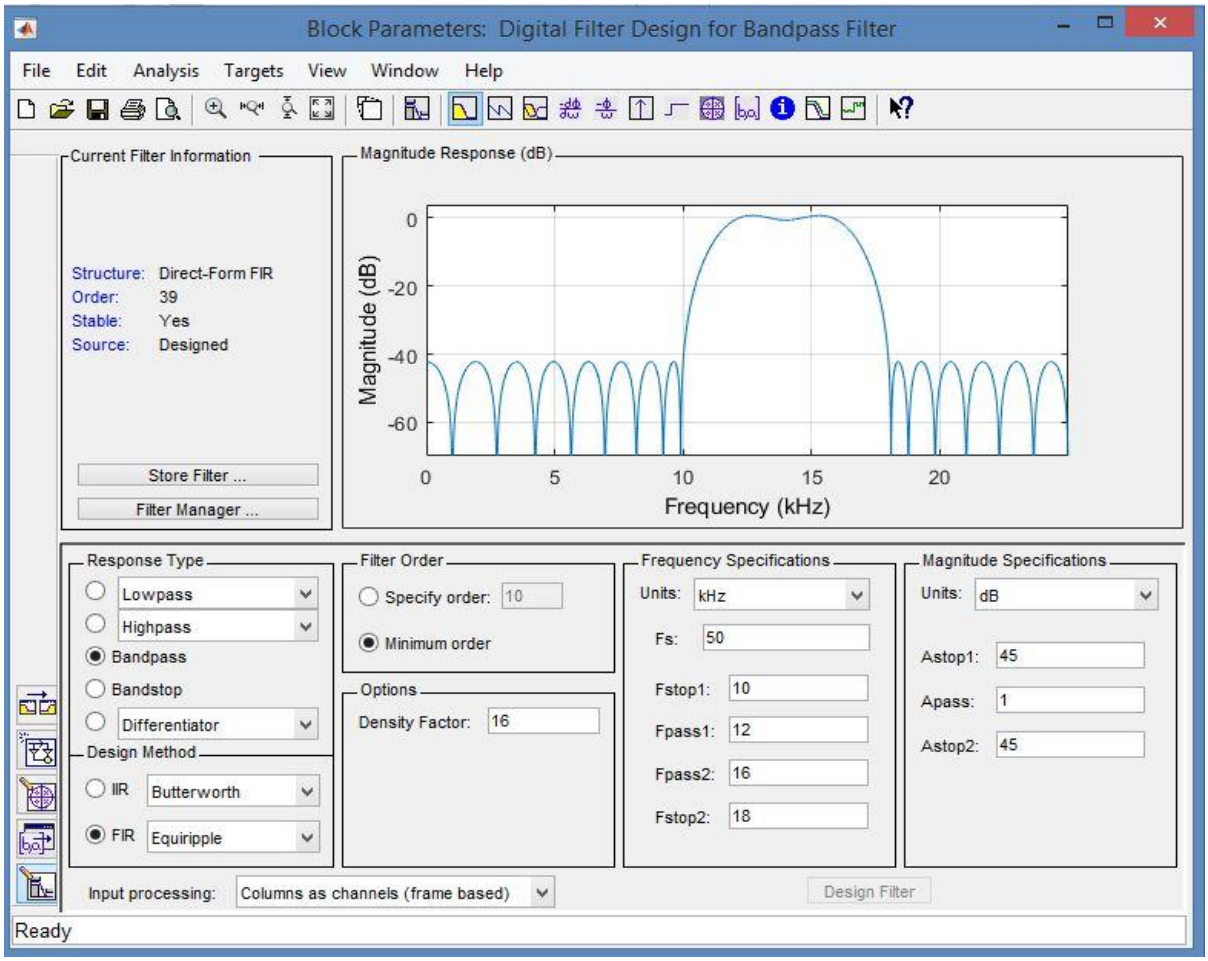
EQUIPMENT REQUIRED:

Hardware required	**Software Required**
PC	MATLAB

THEORY: The various steps to implement the filter is shown below:

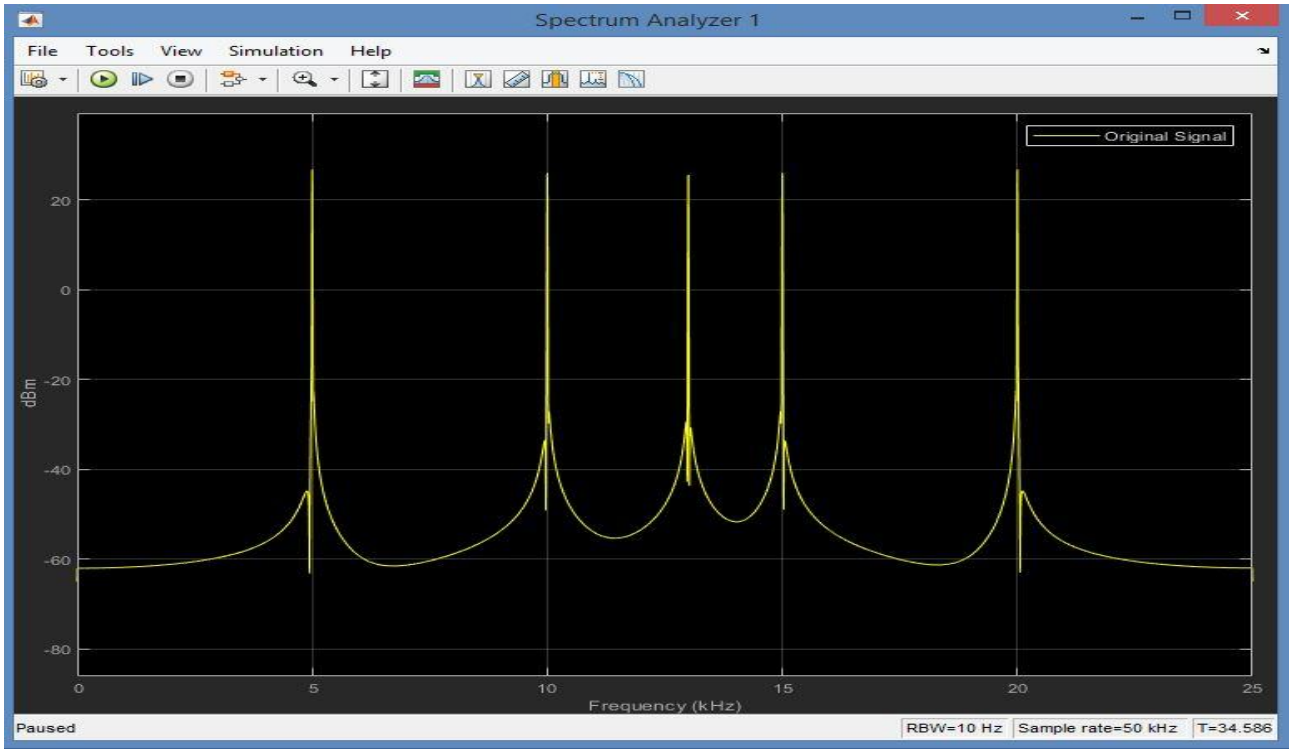



Source Block Parameters: Sine Wave
Sine Wave (mask) (link)
Output samples of a sinusoid. To generate more than one sinusoid simultaneously, enter a vector of values for the Amplitude, Frequency, and Phase offset parameters.
Main
Data Types
Amplitude:
1
Frequency (Hz):
5000
Phase offset (rad):
0
Sample mode: Discrete
Output complexity: Real
Computation method: Trigonometric fcn
Sample time:
1/50000
OK
Cancel
Help

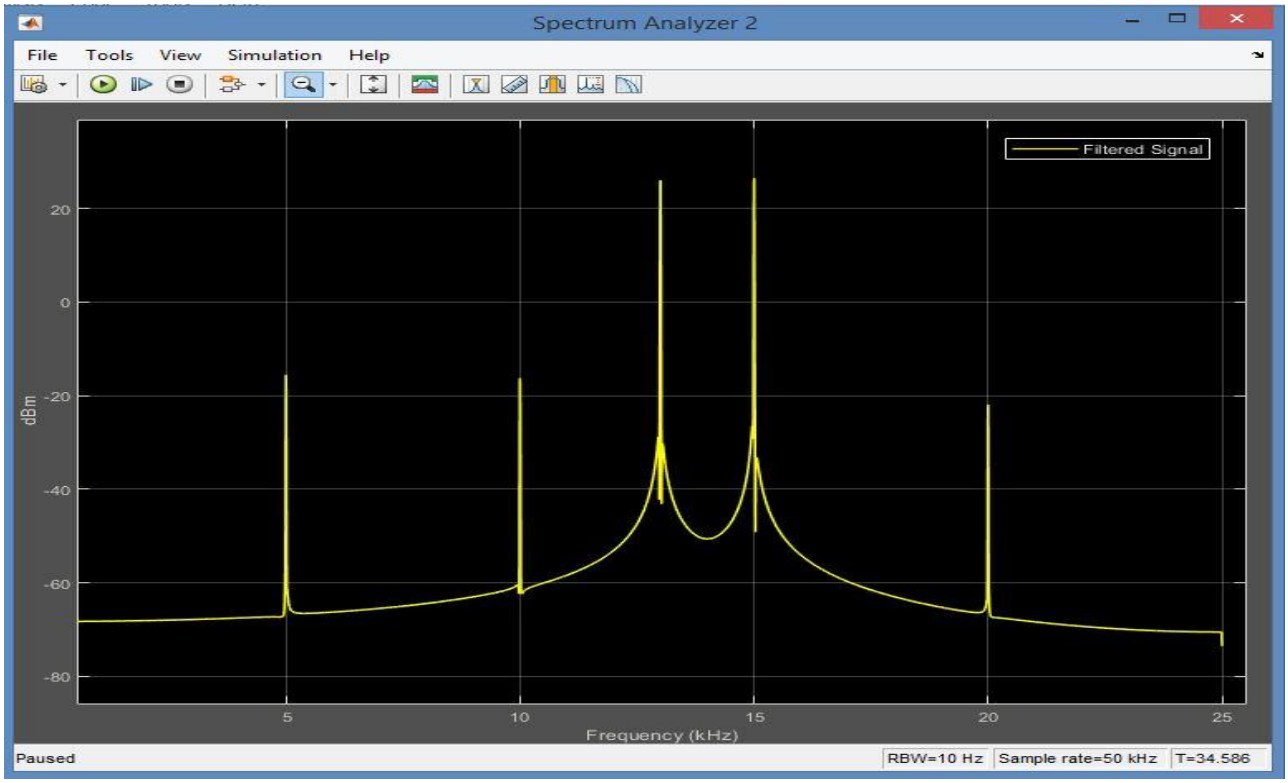

Block Parameters: Digital Filter Design for Bandpass Filter
File Edit Analysis Targets View Window Help
Current Filter Information
Structure: Direct-Form FIR
Order: 39
Stable: Yes
Source: Designed
Store Filter ...
Filter Manager ...
Magnitude Response (dB)
Magnitude (dB)
0
-20
-40
-60
0 5 10 15 20
Frequency (kHz)
Response Type
Lowpass
Highpass
Bandpass
Bandstop
Differentiator
Design Method
IIR Butterworth
FIR Equiripple
Filter Order
Specify order: 10
Minimum order
Options
Density Factor: 16
Frequency Specifications
Units: kHz
Fs: 50
Fstop1: 10
Fpass1: 12
Fpass2: 16
Fstop2: 18
Magnitude Specifications
Units: dB
Astop1: 45
Apass: 1
Astop2: 45
Input processing: Columns as channels (frame based)
Design Filter
Ready

OUTPUT:

Unfiltered Output



Filtered Output



CONCLUSION:

Band Pass Filter has thus been implemented using Simulink

QUESTIONS:

1. What is a linear phase bandpass filter?
2. What are various steps to implement the band stop filter in MATLAB?
3. What is sampling frequency?
4. What do you mean by attenuation?

EXPERIMENT-7

Implement an FIR filter on DSP kit by using FDA tool and code composer studio.

EQUIPMENT REQUIRED:

Hardware required	**Software Required**
Windows OS (win8)	MATLAB
DSP Kit TMS320C6416/6713	Code Composer Studio v5.4
Power Supply Adapter	

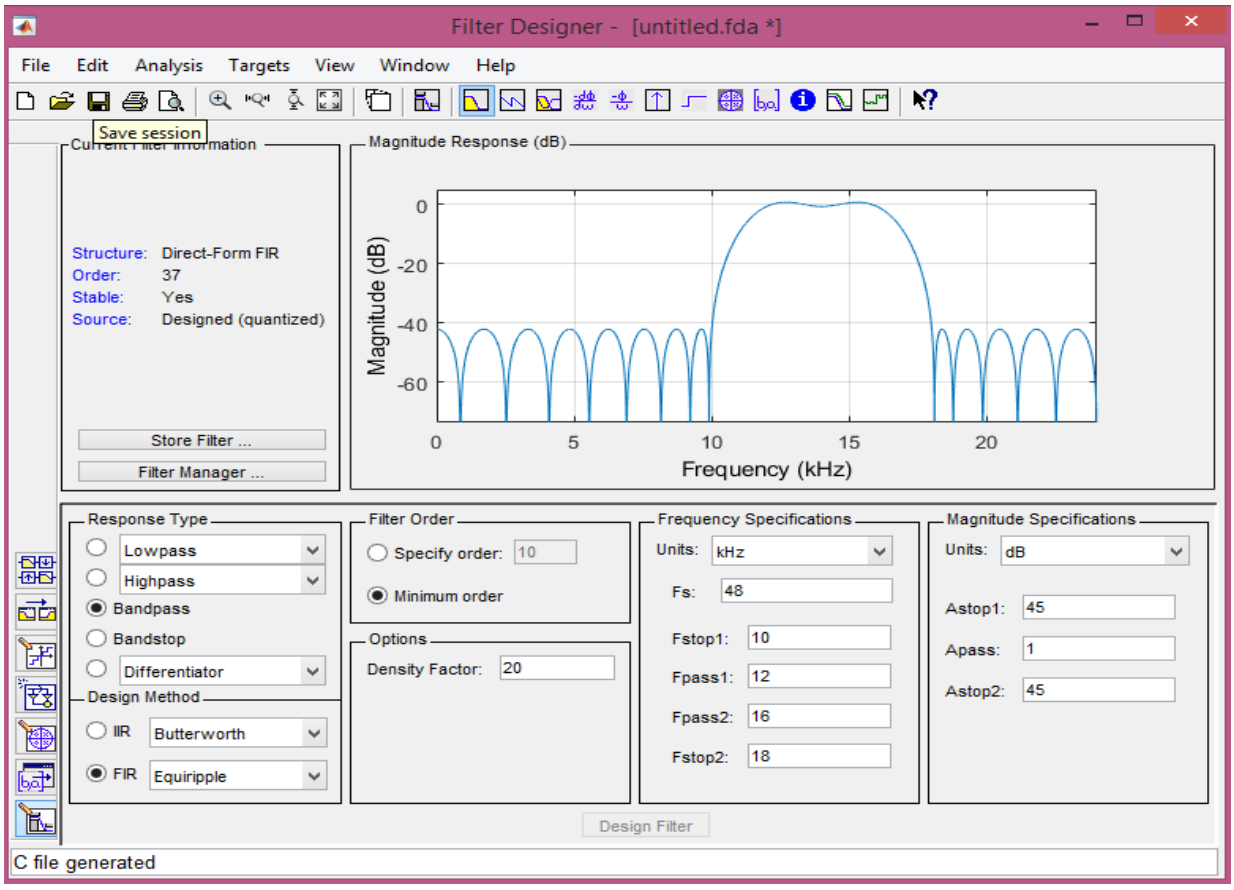
THEORY:

FDA Tool of MATLAB

The Filter Design and Analysis Tool (FDA Tool) is a powerful user interface for designing and analyzing filters quickly. FDA Tool enables you to design digital FIR or IIR filters by setting filter specifications, by importing filters from your MATLAB workspace, or by adding, moving or deleting poles and zeros. FDA Tool also provides tools for analyzing filters, such as magnitude and phase response and pole-zero plots.

Type fdatool in the MATLAB command prompt:

>>fdatool (for newer version of MATLAB type >>filterDesigner)



Designing a Filter

We will use an FIR Equiripple filter with these specifications.

Passband: 12-16 KHz

Transition width: 2 KHz

Pass band attenuation: 1 dB

Stop band attenuation: 45 dB

Sampling frequency: 48 KHz

Filter Order: Minimum Order

Quantization Parameter  : Filter Arithmetic – Single-precision Floating-point

Density Factor: The FIR Equiripple filter has an option which controls the density of the frequency grid. Increasing the value creates a filter which more closely approximates an ideal equiripple filter, but more time is required as the computation increases. Leave this value unchanged.

- After setting the design specifications, click the **Design Filter** button at the bottom of the GUI to design the filter. The magnitude response of the filter is displayed in the Filter Analysis area after the coefficients are computed.
- Click **Targets** and then click **Generate C Header**
    - Click on Export suggested: Single-precision Floating-point
    - Then click **Generate**
    - Save it as a header file
    - Open the file with notepad and copy filter coefficients
    - Paste them in the line of code *float fc[] = {"paste here"}*

The hardware experiments in the DSP lab are carried out on the Texas Instruments TMS320C6713 DSP Starter Kit (DSK), based on the TMS320C6713 floating point DSP running at 225 MHz. The basic clock cycle instruction time is 1/(225 MHz)= 4.44 nanoseconds. During each clock cycle, up to eight instructions can be carried out in parallel, achieving up to 8×225 = 1800 million instructions per second (MIPS). The C6713 processor has 256KB of internal memory, and can potentially address 4GB of external memory. The DSK board includes a 16MB SDRAM memory and a 512KB Flash ROM. It has an on-board 16-bit audio stereo codec (the Texas Instruments AIC23B) that serves both as an A/D and a D/A converter. There are four 3.5 mm audio jacks for microphone and stereo line input, and speaker and head-phone outputs. The AIC23 codec can be programmed to sample audio inputs at the following sampling rates: fs = 8, 16, 24, 32, 44.1, 48, 96 kHz

The ADC part of the codec is implemented as a multi-bit third-order noise-shaping delta-sigma converter (see Ch. 2 & 12 of [1] for the theory of such converters) that allows a variety of oversampling ratios that can realize the above choices of fs. The corresponding oversampling decimation filters act as anti-aliasing prefilters that limit the spectrum of the input analog signals effectively to the Nyquist interval [−fs/2, fs/2]. The DAC part is similarly implemented as a multi-bit second-order noise-shaping delta-sigma converter whose oversampling interpolation filters act as almost ideal reconstruction filters with the Nyquist interval as their passband. The DSK also has four user-programmable DIP switches and four LEDs that can be used to control and monitor programs running on the DSP. All features of the DSK are managed by the CCS, which is a complete integrated development environment (IDE) that includes an optimizing C/C++ compiler, assembler, linker, debugger, and program loader. The CCS communicates with the DSK via a USB connection to a PC. In addition to facilitating all programming aspects of the C6713 DSP, the CCS can also read signals stored on the DSP's memory, or the SDRAM, and plot them in the time or frequency domains.

A 12 - MHz crystal supplies the clock to the AIC23 codec (also to the DSP and the USB interface). Using this 12 - MHz master clock, with oversampling rates of 250Fs and 272Fs , exact audio sample rates of 48 kHz (12 MHz/250) and the CD rate of 44.1 kHz (12 MHz/272) can be obtained. The sampling rate of the AIC23 can be configured to be 8, 16, 24, 32, 44.1, 48, or 96 kHz. Communication with the AIC23 codec for input and output uses two multichannel buffered serial ports (McBSPs) on the C6713. McBSP0 is used as a unidirectional channel to send a 16 - bit control word to the AIC23. McBSP1 is used as a bidirectional channel to send and receive audio data. The codec can be configured for data - transfer word-lengths of 16, 20, 24, or 32 bits. The LINE IN and HEADPHONE OUT signal paths within the codec contain configurable gain elements with ranges of 12 to − 34 dB in steps of 1.5 dB, and 6 to − 73 dB in steps of 1 dB, respectively. Most of the programming examples in this booklet configure the codec for a sampling rate of 8 kHz, 32 - bit data transfer, and 0 - dB gain in the LINE IN and HEADPHONE OUT signal paths. The maximum allowable input signal level at the LINE IN inputs to the codec is 1 V rms. However, the C6713 DSK contain a potential divider circuit with a gain of 0.5 between their LINE IN sockets and the codec itself with the effect that the maximum allowable input signal level at the LINE IN sockets on the DSK is 2 V rms. Above this level, input signals will be distorted. Input and output sockets on the DSK are ac coupled to the codec. The C language source file for a program, loop_poll.c , that simply copies input samples read from the AIC23 codec ADC back to the AIC23 codec DAC as output samples is listed in Figure3A.2. Effectively, the MIC input socket is connected straight through to the HEADPHONE OUT socket on the DSK via the AIC23 codec and the digital signal processor. loop_poll.c uses the same polling technique for real - time input and output as program sine8_LED.c , presented in previous experiment .

The functions input_left_sample() , output_left_sample() , and comm_poll() are defined in the support file c6713dskinit.c. This way the C source file loop_poll.c is kept as small as possible and potentially distracting low level detail is hidden. The implementation details of these, and other, functions defined in c6713dskinit.c need not be studied in detail in order to carry out the examples presented in this booklet but are described here for completeness. Further calls are made by input_left_sample() and output_left_sample() to lower level functions contained in the board support library DSK6713bsl.lib. Function comm_poll() initializes the DSK and, in particular, the AIC23 codec such that its sample rate is set according to the value of the variable fs (assigned in loop_poll.c), its input source according to the value of the variable inputsource (assigned in loop_poll.c), and polling mode is selected. Other AIC23 configuration settings are determined by the parameters specified in file c6713dskinit.h. These parameters include the gain settings in the LINE IN and HEADPHONE out signal paths, the digital audio interface format, and so on. Similar values for all of these parameters are used by almost all of the program examples in this booklet. Only rarely will they be changed and so it is convenient to hide them out of the way in file c6713dskinit.h .

The two settings, sampling rate and input source, are changed sufficiently frequently, from one program example to another, that their values are set in each example program by initializing the values of the variables fs and inputsource. In function dsk6713_init() in file c6713dskinit.c , these values are used by functions DSK6713_AIC23_setFreq() and DSK6713_AIC23_rset(), respectively. In polling mode, function input_left_sample() polls, or tests, the receive ready bit (RRDY) of the McBSP serial port control register (SPCR) until this indicates that newly converted data is available to be read using function MCBSP_read(). Function output_left_sample() polls, or tests, the transmit ready bit (XRDY) of the McBSP serial port control register (SPCR) until this indicates that the codec is ready to receive a new output sample. A new output sample is sent to the codec using function McBSP_write() . Although polling is simpler than the interrupt technique used in sine8_buf.c, it is less efficient since the processor spends nearly all of its time repeatedly testing whether the codec is ready either to transmit or to receive data.

Program loop_intr.c is functionally equivalent to program loop_poll.c but makes use of interrupts. This simple program is important because many of the other example programs in this booklet are based on the same interrupt - driven model. Instead of simply copying the sequence of samples representing an input signal to the codec output, a digital filtering operation can be performed each time a new input sample is received. It is worth taking time to ensure that you understand how program loop_intr.c works. In function main() , the initialization function comm_intr() is called. comm_intr() is very similar to comm_poll() but in addition to initializing the DSK, codec, and McBSP, and not selecting polling mode, it sets up interrupts such that the AIC23 codec will sample the analog input signal and interrupt the C6713 processor, at the sampling frequency defined by the line Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; //set sampling rate It also initiates communication with the codec via the McBSP. In this example, a sampling rate of 8 kHz is used and interrupts will occur every 0.125 ms. (Sampling rates of 16, 24, 32, 44.1, 48, and 96 kHz are also possible.) Following initialization, function main() enters an endless while loop, doing nothing but waiting for interrupts. The functions that will act as interrupt service routines for the various different interrupts are specified in the interrupt service table contained in file vectors_intr.asm. This assembly language file differs from the file vectors_poll.asm in that function c_int11() is specified as the interrupt service routine for interrupt INT11. On interrupt, the interrupt service routine (ISR) c_int11() is called and it is within that routine that the most important program statements are executed. Function output_left_sample() is used to output a value read from the codec using function input_left_sample() .

Format of Data Transferred to and from AIC 23 Codec

The AIC23 ADC converts left - and right - hand channel analog input signals into 16 - bit signed integers and the DAC converts 16 - bit signed integers to left - and right - hand channel analog output signals. Left - and right - hand channel samples are combined to form 32 - bit values that are communicated via the multichannel buffered serial port (McBSP) to and from the C6713. Access to the ADC and DAC from a C program is via the functions Uint32 input_sample(), short input_left_sample(), short input_right_sample(), void output_sample(int out_data) , void output_left_sample(short out_data), and void output_right_sample(short out_data). The 32 - bit unsigned integers (Uint32) returned by input_sample() and passed to output_sample() contain both left and right channel samples.

In file dsk6713init.h declares a variable that may be handled either as one 32 - bit unsigned integer (AIC_data.uint) containing left and right channel sample values, or as two 16 - bit signed integers (AIC_data.channel[0] and AIC_data.channel[1]).

Most of the program examples in this booklet use only one channel for input and output and for clarity most use the functions input_left_sample() and output_left_sample() . These functions are defined in the file c6713dskinit.c, where the unpacking and packing of the signed 16 - bit integer left - hand channel sample values out of and into the 32 - bit words received and transmitted from and to the codec are carried out.

File Types

You will be working with a number of files with different extensions. They include:

1. file.pjt : to create and build a project named file.

2. file.c : C source program.

3. file.asm : assembly source program created by the user, by the C compiler,or by the linear optimizer.

4. file.sa : linear assembly source program. The linear optimizer uses *file.sa* as input to produce an assembly program *file.asm* .

5. file.h : header support file.

6. file.lib : library file, such as the run - time support library file rts6700.lib .

7. file.cmd : linker command file that maps sections to memory.

8. file.obj : object file created by the assembler.

9. file.out : executable file created by the linker to be loaded and run on the C6713 processor.

10. file.cdb : configuration file when using DSP/BIOS.

INTRODUCTION TO TMS320 C6713 DSK

The high–performance board features the TMS320C6713 floating-point DSP. Capable of performing 1350 million floating point operations per second, the C6713 DSK the most powerful DSK development board. The DSK is USB port interfaced platform that allows to efficiently develop and test applications for the C6713. With extensive host PC and target DSP software support, the DSK provides ease of use and capabilities that are attractive to DSP engineers. The 6713 DSP Starter Kit (DSK) is a low-cost platform which lets customers evaluate and develop applications for the Texas Instruments C67X DSP family. The primary features of the DSK are:

1. 225 MHz TMS320C6713 Floating Point DSP
2. AIC23 Stereo Codec
3. Four Position User DIP Switch and Four User LEDs
4. On-board Flash and SDRAM

C6713 DSK Functional Block Diagram

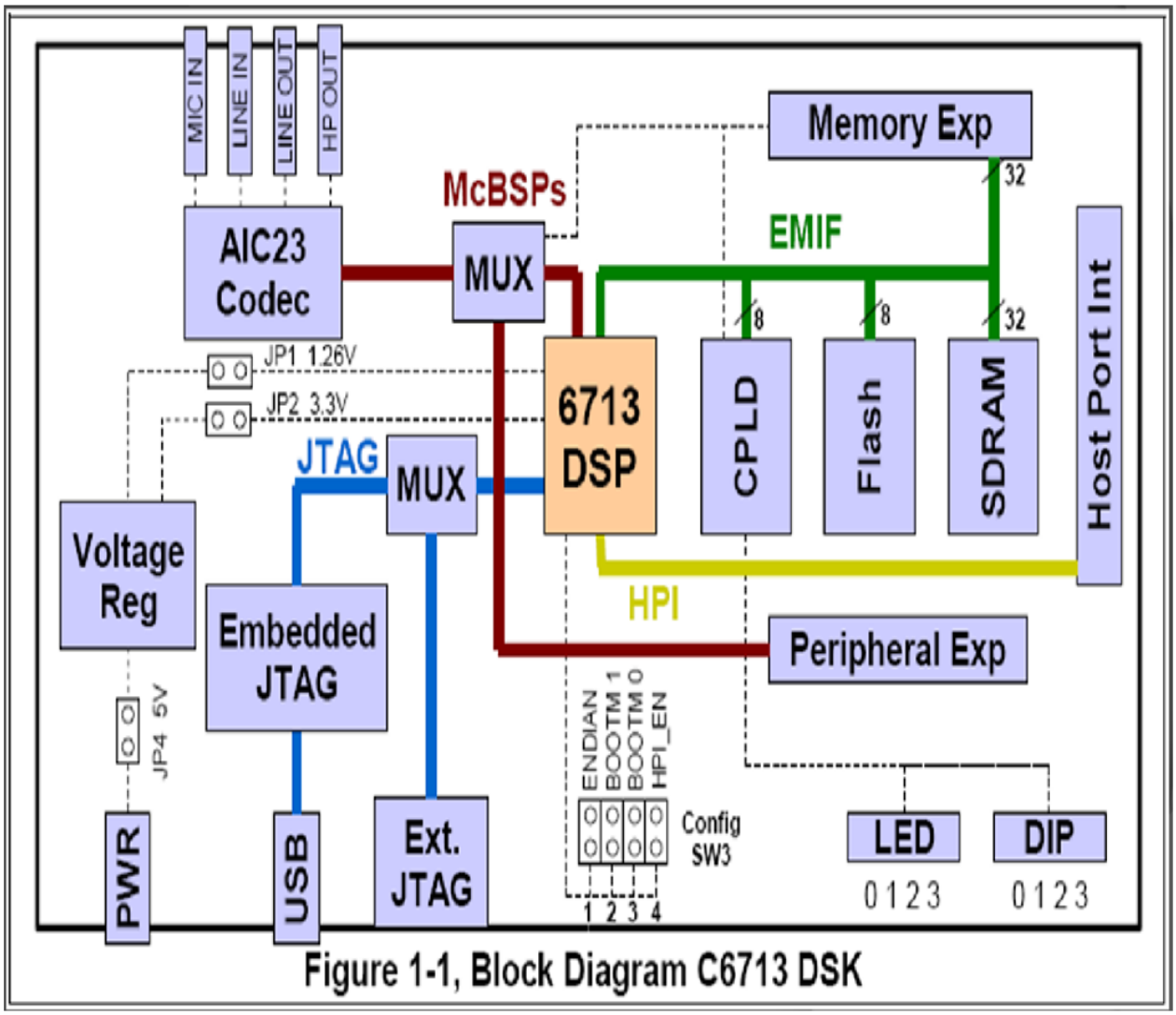


Figure 1-1, Block Diagram C6713 DSK

➢ C program to implement bandpass filter on the TMS320C6711 DSK

```
#include "dsk6713.h"            //this file is added to initialize the DSK6713
#include "dsk6713_aic23.h"
Uint32 fs = DSK6713_AIC23_FREQ_48KHZ;
// FILTER COFFICIENTS AS CALCULATED USING MATLAB
float fc[]=
{
0.00139923906, -0.00468252087,-0.004775044508,  0.02214358561,
-0.003064858727,-0.03045471571,    0.022862738,  0.02239111252,
-0.02766902,-0.002124678111,-0.00319130579,  0.01155226957,
 0.04955762252, -0.07796351612, -0.05158014223, 0.1677316874,
-0.02588679269,  -0.2025853097,   0.1402306259,   0.1402306259,
-0.2025853097, -0.02588679269, 0.1677316874, -0.05158014223,
-0.07796351612,0.04955762252, 0.01155226957, -0.00319130579,
-0.002124678111, -0.02766902,0.02239111252,   0.022862738,
-0.03045471571, -0.003064858727, 0.02214358561,- 0.004775044508,
-0.00468252087, 0.00139923906
};
static short in_buffer[38];                 //input buffer of subject to filter order
Uint32 input_sample();
void output_sample();
void comm_intr();

void main()
{
comm_intr();   //ISR function is called, using the given command
while(1);
}
interrupt void c_int11()
{
        Uint32 indata;
        int i=0;
        signed int output=0;

        indata = input_sample();
        in_buffer[0] = indata;
        for (i=37; i>=0; i--)
```

```
        in_buffer[i] = in_buffer[i-1];  // shuffle the buffer
        for (i=0; i<38; i++)
        output = output + fc[i] * in_buffer[i];
// coefficients multiplied with the input sine_wave & stored into output variable and ouput variable is again added with the output
        output_sample(output);
}
```

Setting up the 6713 DSK for filter implementation

Initial Set-up of the Equipment

- Connect the parallel printer port cable between the USB port on the DSK board (J201) and the USB port on the computer.
- Connect the 5V power supply to the power connector next to the USB port on the DSK board (J5). LED_1 glow for 2 seconds. Then you should see 4 LEDs blink next to some dip switches and after some time all LEDs glow continuously. Our DSK is connected to PC now.

Once the DSK board is connected to your PC and the power supply has been connected, you can start CCS. To do this, click on **Start**, go to **Program**, and then go to **Texas Instruments** then **CCStudio**. In our case the CCStudio version is CCSv5.4

Creating a new project file

- In CCS, select **File** and then **New** and then **CCS Project**.
- Enter your project name and move down to **Family** option and select **C6000** option.
- In **Variant** option select **DSK6713.**
- Move down to connection option and select **Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator**
- Move to Project Template and examples option
    - Select Empty Project (with main.c)
- Click **Finish**

Adding support files to the project:

- Right-Click on your project in the project explorer, then click Properties
- Move to *Include* options in the Build-C6000 Compiler menu
- Click add (+) icon. A new add directory path window will open.
- Click file system and move to C drive and then to DSK6713 folder. After that click c6000 folder option, then to dsk6713 folder and finally click on *include* folder and click ok.
- Click add icon again. A new add directory path window will open.

- Click file system and move to C drive and then to DSK6713 folder. After that click c6000 folder option, then to dsk6713 folder and finally click on *lib* folder and click ok.
- Click add icon again. A new add directory path window will open.
- Click file system and move to C drive and then to C6xCSL folder. After that click on *include* folder and click ok.
- Click add icon again. A new add directory path window will open.
- Click file system and move to C drive and then to C6xCSL folder. After that click on *lib_3x* folder and click ok.

Similarly add more files to project by right clicking Project name and clicking on the **Add files** icon. Few more files like *c6713dskinit.c, c6713dskinit.h, C6713DSK.cmd, vectors_intr.asm.(These files can be found in folder named "Necessary files")*

- Click Properties → Build → C6000 Compiler → Processor Options → Write 6700.
- Click Advance Options → Predefined Symbols → Write CHIP_6713
- Then → Diagnostic Options → Add Suppress Diagnostic <id> → Write 16002
- Then → Runtime model Options → Data Access Model → Far
- Click C6000 Linker → File Search Path → Add file → DSK6713→c6000→dsk6713→lib→dsk6713_bsl.lib
- Then → Add file → C6xCSL→lib_3x→CSL6713.lib

➢ Hardware setup for the filtering of signal

- Generate a signal of varying frequency with amplitude of 2 V_{pp} Volts, using signal generator *(Wavegen function in DSO).*
- Connect Gen Out terminal of DSO with LINE IN 3.5 mm female Jack of C6713 DSK kit & LINE OUT 3.5 mm female Jack terminal with Channel 1 of DSO.
- Verify the output signal, both in time and frequency domains, using the FFT function of Digital Storage Oscilloscope (DSO).

➢ Running the DSK and making measurements

- Go to **Project** pull-down menu in the **CCS window**, and then select **Build**.
- A new sub-window will appear on the bottom of the CCS window. When building is complete, you should see the following message in the new sub-window:

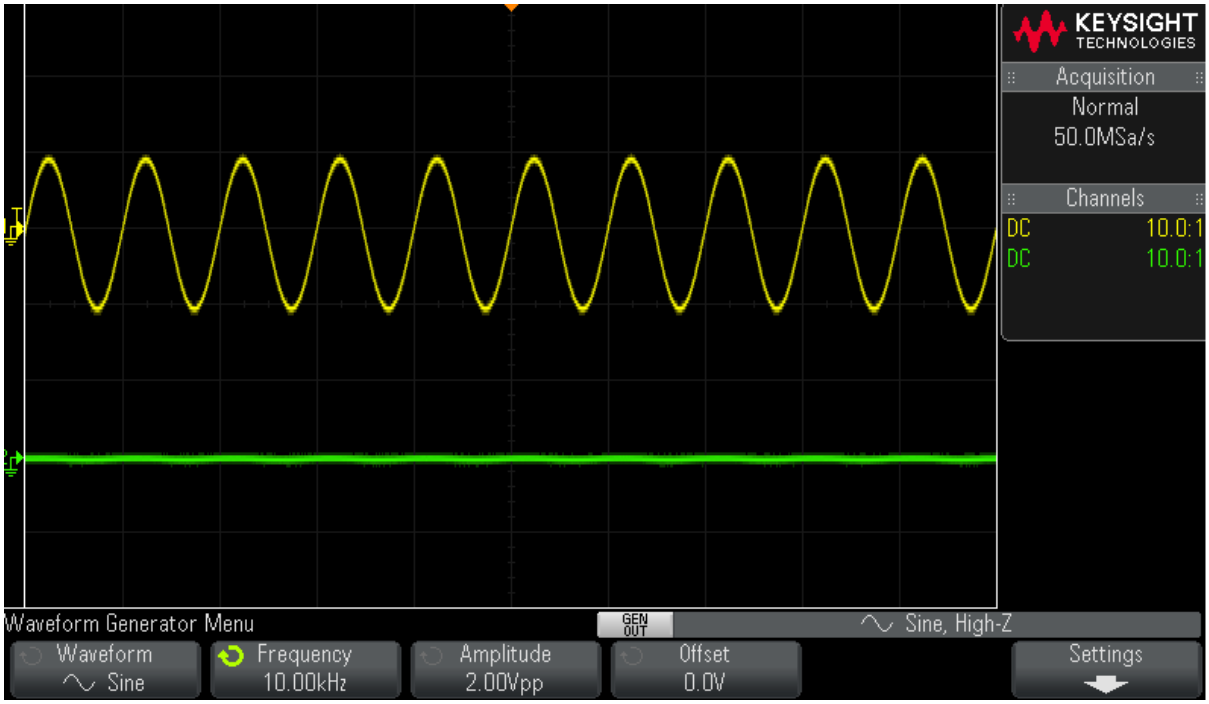
  Build Complete,
  0 Errors, 0 Warnings, 0 Remarks
  (Warnings, if any, can be ignored)
- Click on the **Debug** pull-down menu.
- Then, to load the program onto the DSK, click on the **File** pull-down menu and select **Load Program**.
- In CCS, select the **Debug** pull down menu and then select **Run** *(Green Play button).*

OUTPUT:

Input signal

(10 KHz)

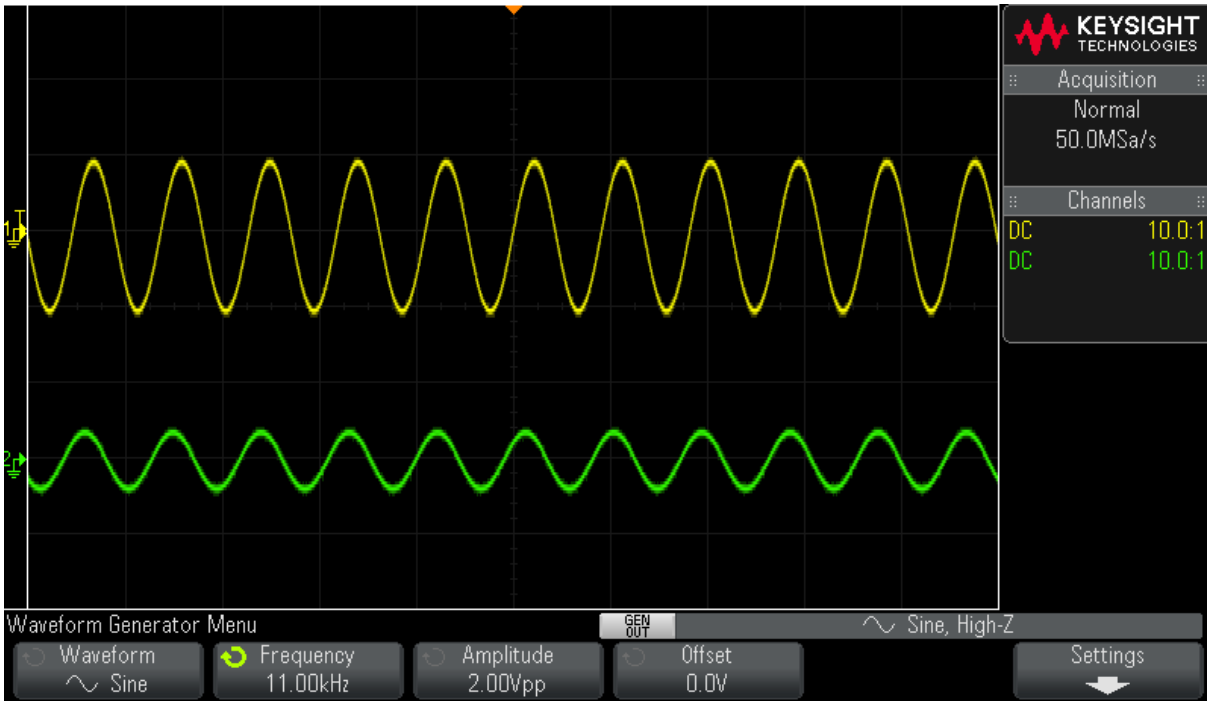
Output Signal



Input signal

(11 KHz)

Output Signal



Input signal

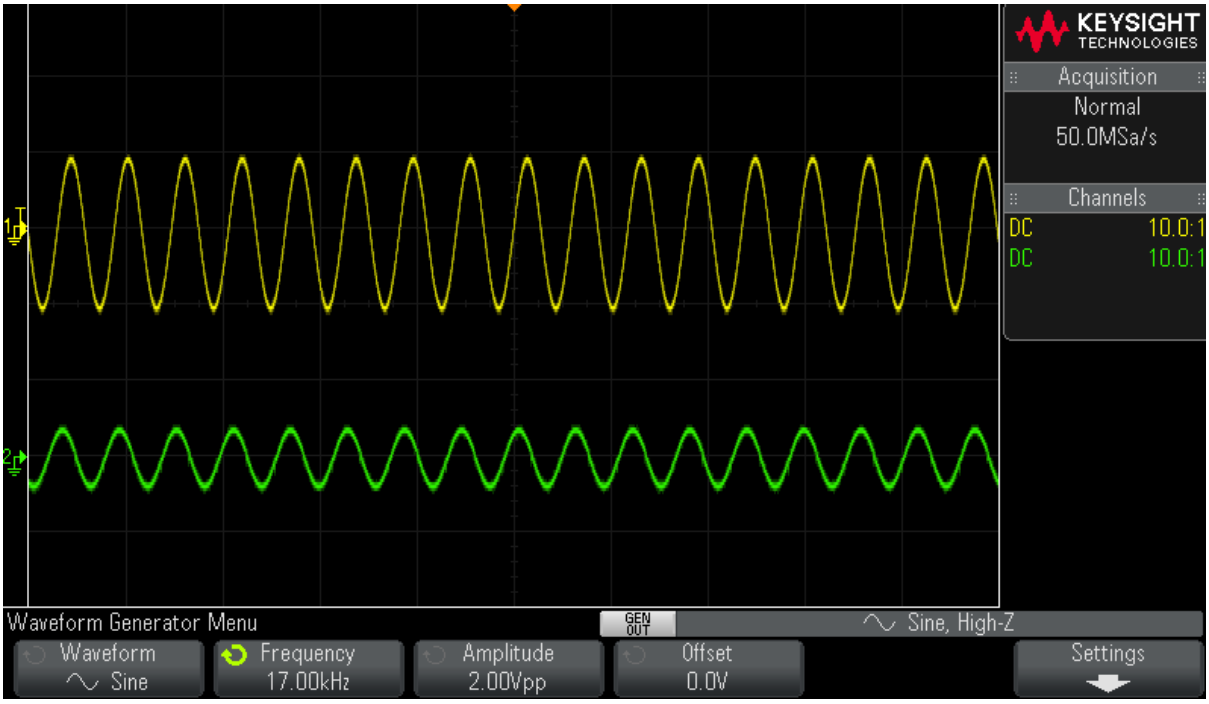
(12 KHz)

Output Signal

Input signal

(17 KHz)

Output Signal



CONCLUSION:

FIR filter using FDA tool is successfully implemented on the DSP Kit TMS320C6713.

EXPERIMENT-8

Implement an IIR filter on the DSP kit by using the FDA tool and code composer studio.

EQUIPMENT REQUIRED:

Hardware required	**Software Required**
Computer System	MATLAB
DSP Kit TMS320C6416/6713	Code Composer Studio v5.4
Power Supply Adapter	

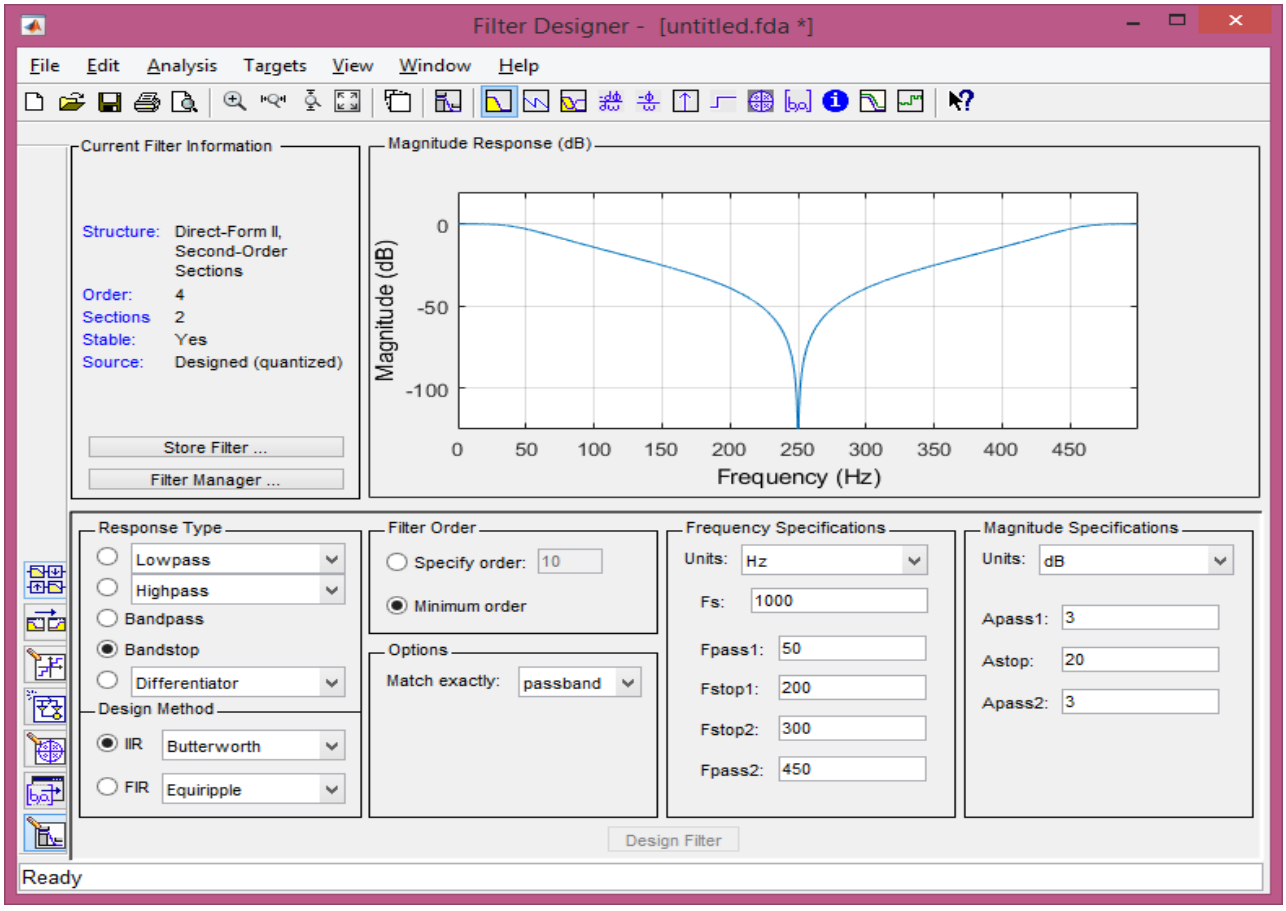
THEORY:

FDA Tool of MATLAB

The Filter Design and Analysis Tool (FDA Tool) is a powerful user interface for designing and analyzing filters quickly. FDA Tool enables you to design digital FIR or IIR filters by setting filter specifications, by importing filters from your MATLAB workspace, or by adding, moving or deleting poles and zeros. FDA Tool also provides tools for analyzing filters, such as magnitude and phase response and pole-zero plots.

Type fdatool in the MATLAB command prompt:

>>fdatool (for newer version of MATLAB type >>filterDesigner)



Designing a Filter

We will use an IIR Butterworth filter with these specifications.

Lower passband	0-50 Hz
Upper passband	450-500 Hz
Stop band	200-300 Hz
Passband ripple	3 db
Stopband attenuation	20 db
Sampling frequency	1 kHz
Filter Order:	Minimum Order
Quantization Parameter  :	Filter Arithmetic – Single-precision Floating-point

Note : The IIR Butterworth filter has a **Density Factor** option which controls the density of the frequency grid. Increasing the value creates a filter which more closely approximates an ideal Butterworth filter, but more time is required as the computation increases. Leave this value unchanged.

- After setting the design specifications, click the **Design Filter** button at the bottom of the GUI to design the filter. The magnitude response of the filter is displayed in the Filter Analysis area after the coefficients are computed.
- Click **Targets** and then click **Generate C Header**
    - Click on Export suggested: Single-precision Floating-point
    - Then click **Generate**
    - Save it as a header file
    - Open the file with notepad and copy filter coefficients
    - Paste them in the line of code *float fc[] = {"paste here"}*

C program to implement bandpass filter on the TMS320C6711 DSK

```
#include "DSK6713_aic23.h"                  //this file is added to initialize the DSK6713
#include "BPF_coeff.h"                      // BPF coefficient file

Uint32 fs=DSK6713_AIC23_FREQ_8KHZ; // sampling frequency of codec
short dly[Ns][3] = {0};                     // filter states in each biquad
interrupt void c_int11()                    // ISR call, At each Interrupt, program execution
                                            goes to the interrupt service routine
        {
                short i, input;                             // variable declaration
                int un, yn;
                input = input_sample();             // input from codec

                for (i = 0; i < Ns; i++)            // repeat for each biquad
                {
                un =  input-((b[i][0]*dly[i][0])>>15)-((b[i][1]*dly[i][1])>>15);
                yn= ((a[i][0]*un)>>15)+((a[i][1]*dly[i][0])>>15)+((a[i][2]*dly[i][1])>>15);

                // update states in current biquad
                dly[i][1] = dly[i][0];
                dly[i][0] = un;
                input     = yn;
                }
                        output_sample((short)yn);       // output final result for time n, sends the
                    signal on to the out_buffer of the CODEC of LINE-OUT port
                return;
        }
void main()
{
        comm_intr();            //ISR function is called, using the given command
        while(1);
}
```

➢ Initial Set-up of the Equipment

- Connect the parallel printer port cable between the USB port on the DSK board (J201) and the USB port on the computer.
- Connect the 5V power supply to the power connector next to the USB port on the DSK board (J5). LED_1 glow for 2 seconds. Then you should see 4 LEDs blink next to some dip switches and after some time all LEDs glow continuously. Our DSK is connected to PC now.

Once the DSK board is connected to your PC and the power supply has been connected, you can start CCS. To do this, click on **Start**, go to **Program**, and then go to **Texas Instruments** then **CCStudio**. In our case the CCStudio version is CCSv5.4

➢ Creating a new project file

- In CCS, select **File** and then **New** and then **CCS Project**.
- Enter your project name and move down to **Family** option and select **C6000** option.
- In **Variant** option select **TMS320C6713.**
- Move down to connection option and select **Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator**
- Move to Project Template and examples option
  - Select Empty Project (with main.c)
- Click **Finish**

➢ Adding support files to the project:

- Right-Click on your project in the project explorer, then click Properties
- Move to Include options in the Build-C6000 Compiler menu
- lick add icon. A new add directory path window will open.
- Click file system and move to C drive and then to DSK6713 folder. After that click c6000 folder option, then to dsk6713 folder and finally click on include folder and click ok.
- Click add icon again. A new add directory path window will open.
- Click file system and move to C drive and then to DSK6713 folder. After that click c6000 folder option, then to dsk6713 folder and finally click on lib folder and click ok.
- Click add icon again. A new add directory path window will open.
- Click file system and move to C drive and then to C6xCSL folder. After that click on include folder and click ok.
- Click add icon again. A new add directory path window will open.
- Click file system and move to C drive and then to C6xCSL folder. After that click on lib_3x folder and click ok.

Similarly add more files to project by right clicking Project name and clicking on the Add files icon. Few more files like *c6713dskinit.c , C6713DSK.cmd , vectors_intr.asm.*

Also, define chip name for the project. Right click project name and click properties. Move to Build, then C6000 compiler, then processor options. In the Target processor version box type 6700.

➢ Hardware setup for the filtering of signal

- Generate a signal of varying frequency ranging from 0 Hz to 500 Hz with amplitude of 2 V_{pp} Volts, using signal generator.
- Connect Gen Out terminal of DSO with LINE IN 3.5 mm female Jack of C6713 DSK kit & LINE OUT 3.5 mm female Jack terminal with Channel 1 of DSO.
- Verify the output signal, both in time and frequency domains, using the FFT function of Digital Storage Oscilloscope (DSO).

➢ Running the DSK and making measurements

- Go to **Project** pull-down menu in the **CCS window**, and then select **Build**.
- A new sub-window will appear on the bottom of the CCS window. When building is complete, you should see the following message in the new sub-window:

  Build Complete,

  0 Errors, 0 Warnings, 0 Remarks
- Click on the **Debug** pull-down menu and select **Reset CPU**.
- Then, to load the program onto the DSK, click on the **File** pull-down menu and select **Load Program**.
- In the new window that appears, double-click on the folder **Debug**, select the **filtering_twosignals.out** file and click on **Open**.

In CCS, select the **Debug** pull down menu and then select **Run**, or just simply click on the top **running man** on the left side toolbar. You should now see the filtered output with a predominant peak at 3 kHz on the Signal Analyzer. However, there may be a small component at 1.5 kHz, hence measure the power level (dBm) at both 1.5 kHz and 3 kHz.

CONCLUSION:

IIR filter using FDA tool is successfully implemented on the DSP Kit TMS320C6713.

QUESTIONS:

1. Define Filters and its types.
2. Explain different types of FIR filters.
3. How do we use filter design, analysis tool and code composer studio?