Digital Signal Processing

Laboratory Manual

Digital Signal Processing Division, VIT University
Course Code : ECE2006

School of Electronics Engineering

School of Electronics Engineering

B. Tech – Electronics & Communication Engineering

ECE2006 - Digital Signal Processing Laboratory

Laboratory Course Description

L T P C : 2 0 2 4 4
Course Type : Program Core
Semester Offered : Fall
Academic Year : 2022-2023

Laboratory Venue : AB1 707

Faculty name with designation	Slot	Cabin No
Dr. K. Mohanaprasad Associate Professor	L1+2, L31+32	Faculty cabin 7th floor AB1 cabin 3
Dr. John Sahaya Rani Alex Associate Professor	L1+L2	Faculty cabin 4th floor cabin 34
Prof. Ramesh	L13+14	Faculty cabin 5th floor 505 cabin 6
Dr. Suchetha	L31+L32	AB1 707 DSP Lab cabin

Course Learning Outcomes

7. Ability to analyse and exploit the real-time signal processing applications

Student Learning Outcomes (SLO):	**2,5,17**
2. Having a clear understanding of the subject related concepts and of contemporary issues 5. Having design thinking capability 17. Having an ability to use techniques, skills and modern engineering tools necessary for engineering practice	

CO – SLO mapping

Module	CO	SLO
L & P	CO_07	2, 5, 17

Lab Course Plan:

Day	Name of the Experiment	Page. No	References
	MatLab based Experiments		
1	Generation of Basic Elementary Digital Signals • Unit Sample & Unit Step sequences • Exponential sequences • Sinusoidal sequences Operation on Sequences • Signal Smoothening by averaging	5	R1,R2,R7
2	Convolution of Signal Sequences • Linear convolution • Circular Convolution	12	R1,R2,R7
3	Correlation of Signal sequences	15	R1,R2,R7
4	Computation of DTFT & its properties (and also relationship with DFT)	17	R1,R2,R7
5	System response & stability analysis using Z-transform	23	R1,R2,R7
6	DFT computation by FFT • Spectrum analysis with the FFT & MatLab	26	R1,R2,R7
7	IIR Filter Design • Analog IIR Filter design • Digital IIR Filter design	30	R1,R2,R7
8	FIR Filter Design	41	R1,R2,R7
9	Learning SPTool & FDATool • ECG signal smoothening analysis with FIR Filter • Speech & Music Signal analysis with IIR Filter	48	R1,R2,R7
10	GENERAL PROCEDURE FOR WORKING WITH THE REAL TIME PROJECTS USING C6748DSK:	62	R8
11	Advance Discrete Time Filter Design (FIR) Finite Impulse Response Filter	66	R8
12	ADVANCE DISCRETE TIME FILTER DESIGN (IIR)	74	R8

Lab Evaluation procedure

- The Continuous assessment made based on experiments is evaluated for 50 marks.
- Each experiment is evaluated for 15 marks under the following scheme:

Viva (Aim, Objective of the Experiment, Manual Calculation, Flow chart or Algorithm techniques)	- 5 Marks
Program Execution (Successful compilation of the coding)	- 5 Marks
Result and Inference	- 2 Marks
Lab note book completion and submission in time	- 3 Marks

- All the lab experiments must be evaluated on the same day of practical session. If the student is completed the viva on the same day he/she will be awarded marks based on his performance (15 Marks). If he/she failed to complete the work on the same day, marks will be reduced gradually.
- The average of all the experiment marks will be converted to 50.
- A Term end Lab exam at the end of the semester will be conducted for 50 marks.

DSP, Digital Signal Processing mainly used to extract certain information present on signals represented in 1's & 0's by doing some manupulations. This lab manual was prepared to help the learners to understand the concepts of DSP easily.

Major parts of this Lab Manual were prepared with the help of Online Resources and text book written by **Dr. Sanjit. K. Mitra, Professor.** Thanks to him.

Thanks to Texas Instruments and Mathworks Company for their DSP hardware, software & DSP teaching materials provided by them.

Thanks to the members of DSP division for their various kind of support.

Most of the programs were coded in Matlab and verified, however if error persists, please bring it to my attention by mailing to kmohanaprasad@vit.ac.in

I. Matlab based Experiments:

1. Generation of basic Elementary Digital Signals

Generation of sequences:

Unit Sample and Unit step sequences

$$\delta(n) = \begin{cases} 1, & for\ n = 0, \\ 0, & for\ n \neq 0. \end{cases}$$

$$u(n) = \begin{cases} 1, & for\ n \geq 0, \\ 0, & for\ n < 0. \end{cases}$$

A unit sample sequence u[n] of length N can be generated using the MATLAB command
u = [1 zeros(1,N -1)];
A unit sample sequence ud[n] of length N and delayed by M samples, whereM < N, can be generated using the MATLAB command

ud = [zeros(1,M) 1 zeros(1,N - M - 1)];

Likewise, a unit step sequence s[n] of length N can be generated using the MATLAB command

s = [ones(1,N)];

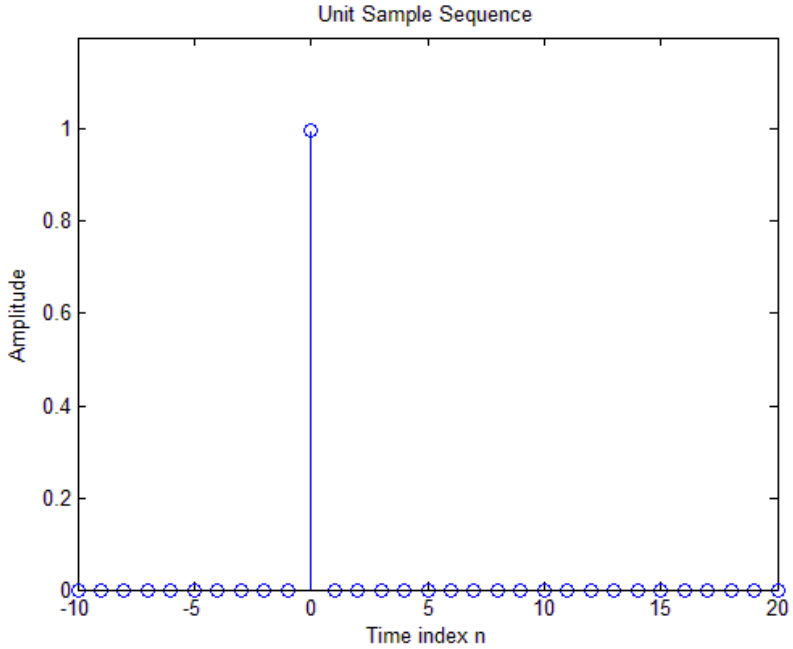
A delayed unit step sequence can be generated in a manner similar to that used in the generation of a delayed unit sample sequence.

```
% Program_P1.1
% Generation of a Unit Sample Sequence
clf;
% Generate a vector from -10 to 20
n = -10:20;
% Generate the unit sample sequence
u = [zeros(1,10) 1 zeros(1,20)];
% Plot the unit sample sequence
stem(n,u);
xlabel('Time index n');ylabel('Amplitude');
title('Unit Sample Sequence');
axis([-10 20 0 1.2]);
```

Questions:

Q1. Run Program P1.1 to generate the unit sample sequence u[n] and display it.
Q2. What are the purposes of the commands clf, axis, title, xlabel, and ylabel?
Q3. Modify Program P1.1 to generate a delayed unit sample sequence ud[n] with a delay of 11 samples. Run the modified program and display the sequence generated.
Q4. Modify Program P1.1 to generate a unit step sequence s[n]. Run the modified program and display the sequence generated.

Q5. Modify Program P1.1 to generate a delayed unit step sequence sd[n] with an advance of 7 samples. Run the modified program and display the sequence generated.



Exponential Signals

Another basic discrete-time sequence is the exponential sequence. Such a sequence can be generated using the MATLAB operators .^ and exp.

The exponential sequence is given by

$$x(n) = A\alpha^n$$

where A and α are real or complex numbers. By expressing

$$\alpha = e^{\sigma_0 + j\omega_0}, and\ A = |A|e^{j\varphi}$$

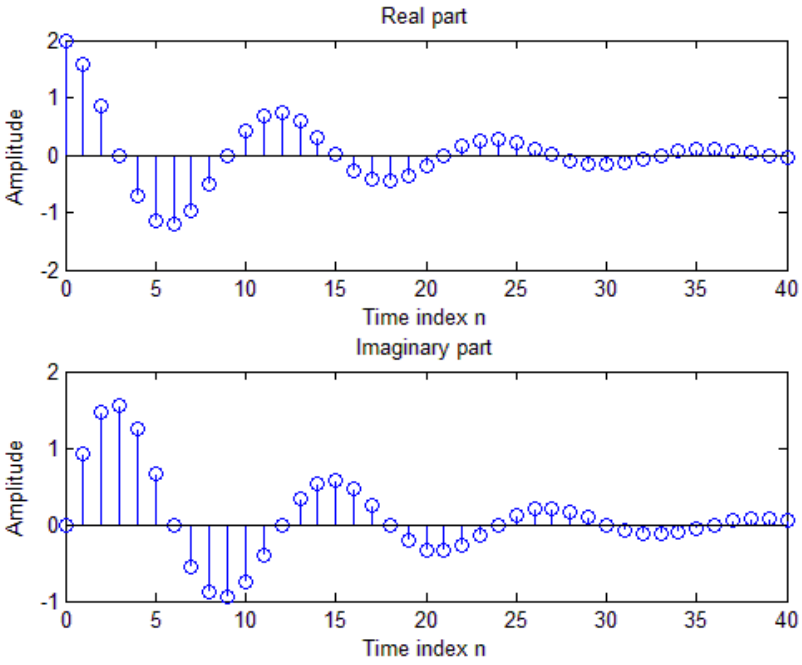
We can rewrite as

$$x[n] = |A|e^{\sigma_0 n + j(\omega_0 n + \varphi)} = |A|e^{\sigma_0 n}\cos(\omega_0 n + \varphi) + j|A|e^{\sigma_0 n}\sin(\omega_0 n + \varphi)$$

Program P1.2 given below can be employed to generate a complex-valued exponential sequence.

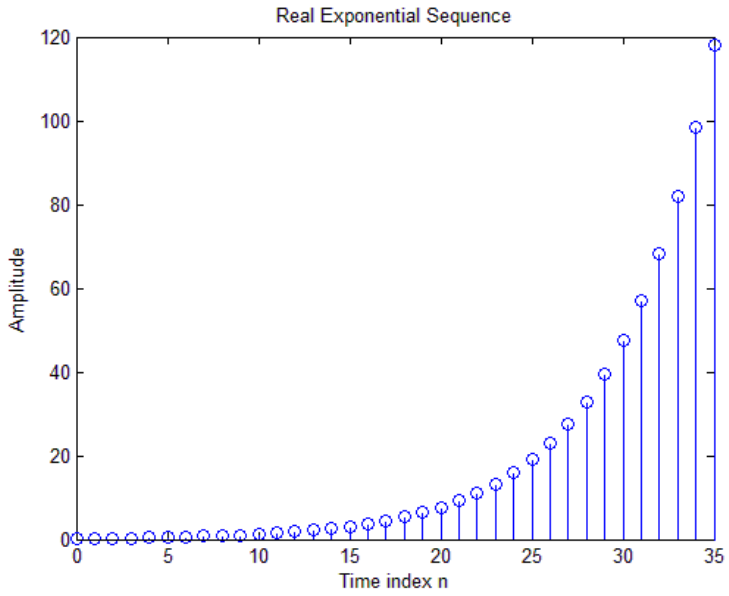
```
% Program_P1.2
clf;
c = -(1/12)+(pi/6)*i;
K = 2;
n = 0:40;
x = K*exp(c*n);
subplot(2,1,1);
stem(n,real(x));
xlabel('Time index n');ylabel('Amplitude');
title('Real part');
subplot(2,1,2);
stem(n,imag(x));
xlabel('Time index n');ylabel('Amplitude');
```

```
title('Imaginary part');
```



Program P1.3 given below can be employed to generate a real-valued exponential sequence.

```
% Program_P1.3
% Generation of a real exponential sequence
clf;
n = 0:35; a = 1.2; K = 0.2;
x = K*a.^+n;
stem(n,x);
xlabel('Time index n');ylabel('Amplitude');
title('Real Exponential Sequence');
```



Questions:
Q1. Run Program P1.2 and generate the complex-valued exponential sequence.
Q2. Which parameter controls the rate of growth or decay of this sequence? Which parameter controls the amplitude of this sequence?
Q3. What will happen if the parameter c is changed to (1/12)+(pi/6)*i?
Q4. What are the purposes of the operators real and imag?
Q5. What is the purpose of the command subplot?
Q6. Run Program P1.2 and generate the real-valued exponential sequence.
Q7. Which parameter controls the rate of growth or decay of this sequence? Which parameter controls the amplitude of this sequence?
Q8. What is the difference between the arithmetic operators ^ and .^?
Q9. What will happen if the parameter a is less than 1? Run Program P1.3 again with the parameter 'a' changed to 0.9 and the parameter K changed to 20.
Q10. What is the length of this sequence and how can it be changed?
Q11. You can use the MATLAB command sum(s.*s) to compute the energy of a real sequence s[n] stored as a vector s. Evaluate the energy of the real-valued exponential sequences x[n] generated in Questions Q6 and Q9.

Sinusoidal Sequences

Another very useful class of discrete-time signals is the real sinusoidal sequence of the form

$$x(n) = A\cos(\omega_0 n + \varphi)$$

Such sinusoidal sequences can be generated in MATLAB using the trigonometric operators cos and sin.

```
% Program_P1.4
% Generation of a sinusoidal sequence
n = 0:40;
f = 0.1;
```

```
phase = 0;
A = 1.5;
arg = 2*pi*f*n - phase;
x = A*cos(arg);
clf; % Clear old graph
stem(n,x); % Plot the generated sequence
axis([0 40 -2 2]);
grid;
title('Sinusoidal Sequence');
xlabel('Time index n');
ylabel('Amplitude');
axis;
```

Questions:

Q1. Run Program P1.4 to generate the sinusoidal sequence and display it.

Q2. What is the frequency of this sequence and how can it be changed? Which parameter controls the phase of this sequence? Which parameter controls the amplitude of this sequence? What is the period of this sequence?

Q3. What is the length of this sequence and how can it be changed?

Q4. Compute the average power of the generated sinusoidal sequence.

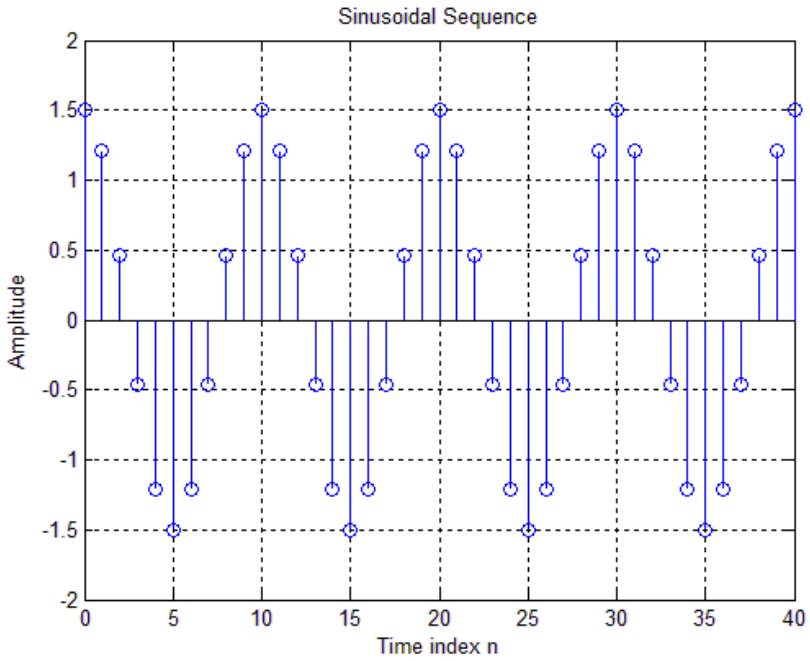
Q5. What are the purposes of the axis and grid commands?

Q6. Modify Program P1.4 to generate a sinusoidal sequence of frequency 0.9 and display it. Compare this new sequence with the one generated in Q1. Now, modify Program P1.4 to generate a sinusoidal sequence of frequency 1.1 and display it. Compare this new sequence with the one generated in Q1. Comment on your results.

Q7. Modify the above program to generate a sinusoidal sequence of length 50, frequency 0.08, amplitude 2.5, and phase shift 90 degrees and display it. What is the period of this sequence?

Q8. Replace the stem command in Program P1.4 with the plot command and run the program again. What is the difference between the new plot and the one generated in Q1?

Q9 Replace the stem command in Program P1.4 with the stairs command and run the program again. What is the difference between the new plot & those generated in Q1 & Q8?



Operation on Sequences:

Signal Smoothing:

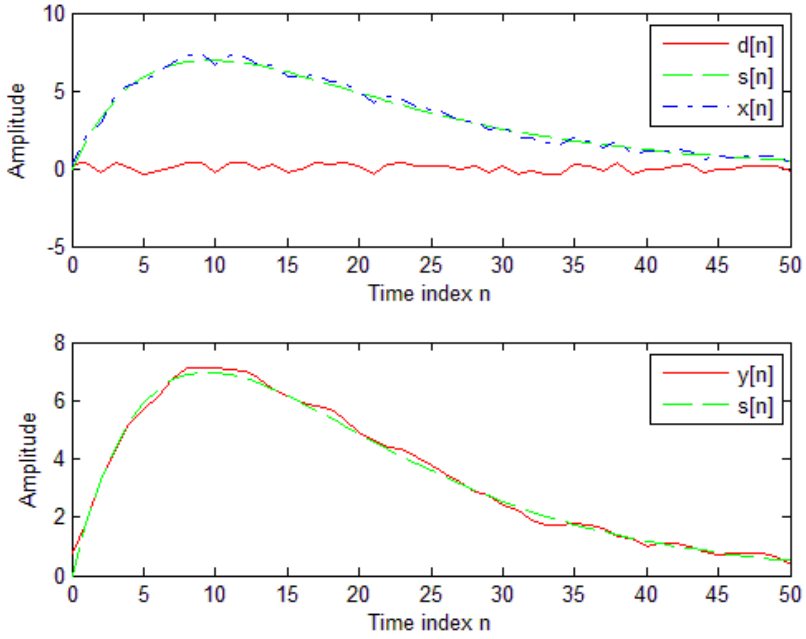
A common example of a digital signal processing application is the removal of the noise component from a signal corrupted by additive noise. Let $s[n]$ be the signal corrupted by a random noise $d[n]$ resulting in the noisy signal $x[n] = s[n] + d[n]$. The objective is to operate on $x[n]$ to generate a signal $y[n]$ which is a reasonable approximation to $s[n]$. To this end, a simple approach is to generate an output sample by averaging a number of input samples around the sample at instant n. For example, a three-point moving average algorithm is given by

$$y[n] = 13\ (x[n-1] + x[n] + x[n+1])$$

Program_1.5 implements the above algorithm.

```
% Program_1.5
% Signal Smoothing by Averaging
clf;
R = 51;
d = 0.8*(rand(R,1) - 0.5);      % Generate random noise
m = 0:R-1;
s = 2*m.*(0.9.^m);              % Generate uncorrupted signal
x = s + d';                     % Generate noise corrupted signal
subplot(2,1,1);
plot(m,d','r-',m,s,'g--',m,x,'b-.');
xlabel('Time index n');ylabel('Amplitude');
legend('d[n] ','s[n] ','x[n] ');
x1 = [0 0 x];x2 = [0 x 0];x3 = [x 0 0];
```

```
y = (x1 + x2 + x3)/3;
subplot(2,1,2);
plot(m,y(2:R+1),'r-',m,s,'g--');
legend('y[n] ','s[n] ');
xlabel('Time index n');ylabel('Amplitude');
```



Questions:

Q1. Run Program 1.5 and generate all pertinent signals.

Q2. What is the form of the uncorrupted signal s[n]? What is the form of the additive noise d[n]?

Q3. Can you use the statement x = s + d to generate the noise-corrupted signal? If not, why not?

Q4. What are the relations between the signals x1, x2, and x3, and the signal x?

Q5. What is the purpose of the legend command?

2. Convolution of Signal Sequences

The response $y[n]$ of a linear, time-invariant discrete-time system characterized by an impulse response $h[n]$ to an input signal $x[n]$ is given by

$$y[n] = \sum_{k=-\infty}^{\infty} h[k]x[n-k]$$

which can be alternately written as

$$y[n] = \sum_{k=-\infty}^{\infty} h[n-k]x[k]$$

by a simple change of variables. The above equation is called the *convolution sum* of the sequences $x[n]$ and $h[n]$, and is represented compactly as:

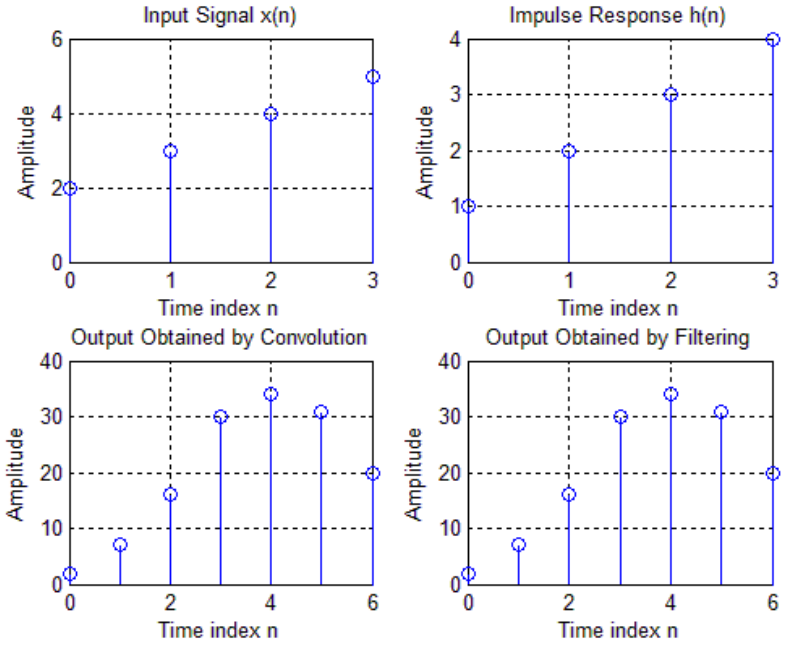
$$y[n] = h[n] \circledast x[n]$$

where the notation $\circledast$ denotes the *convolution sum*.

The convolution operation can be implemented in MATLAB by the command conv, provided the two sequences to be convolved are of finite length. For example, the output sequence of an FIR system can be computed by convolving its impulse response with a given finite-length input sequence. The following MATLAB program illustrates this approach.

```
% Program_2.1
% Linear Convolution of signal sequences
clf;
clear all;
h = input('Input impulse response samples:');
x = input('Input samples of incoming signal:');
hlen = length(h);
xlen = length(x);
for i = 1:xlen
    for j = 1:hlen
        y(i, i+j-1) = x(i) * h(j);
    end
end
y = sum(y);
outl=length(y);
disp('Convolution Output is:');disp(y);
subplot(2,2,1);stem((0:xlen-1),x);
xlabel('Time index n'); ylabel('Amplitude');
title('Input Signal x(n)'); grid;
subplot(2,2,2);stem((0:hlen-1),h);
xlabel('Time index n'); ylabel('Amplitude');
title('Impulse Response h(n)'); grid;
subplot(2,2,3);stem((0:outl-1),y);
xlabel('Time index n'); ylabel('Amplitude');
title('Output Obtained by Convolution');grid;
n=0:xlen+hlen-2;
```

```
x1=[x zeros(1,hlen-1)];
y1=filter(h,1,x1);
subplot(2,2,4);stem(n,y1);
xlabel('Time index n'); ylabel('Amplitude');
title('Output Obtained by Filtering');grid;
```



Questions:

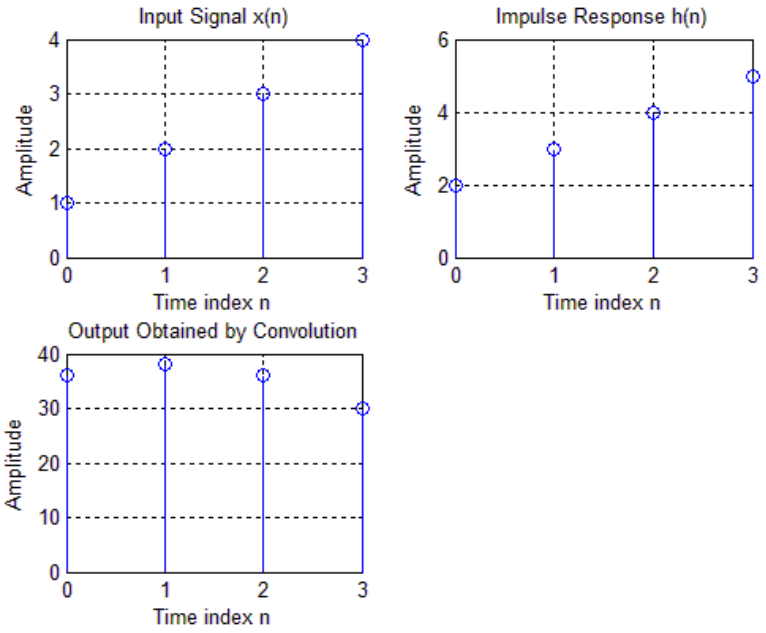
Q1. Run Program P2.1 to generate y[n] obtained by the convolution of the sequences h[n] and x[n], and to generate y1[n] obtained by filtering the input x[n] by the FIR filter h[n]. Is there any difference between y[n] and y1[n]? What is the reason for using x1[n] obtained by zero-padding x[n] as the input for generating y1[n]?

Q2. Modify Program P2.1 to develop the convolution of a length-15 sequence h[n] with a length-10 sequence x[n], and repeat Q2. Use your own sample values for h[n] and x[n].

Program_2.2 implements the Circular Convolution algorithm.

```
% Program_2.2
clf; clear all;
x1 = input('Input sequence1:');
x2 = input('Input sequence2:');
L1 = length(x1); L2 = length(x2);
if L1 ~= L2, error('Sequences of unequal lengths'), end
y = zeros(1,L1);
x2tr = [x2(1) x2(L2:-1:2)];
for k = 1:L1
    M = 1-k; x = x2tr;
if abs(M) > length(x)
M = rem(M,length(x));
end
if M < 0
```

```
M = M + length(x);
end
sh = [x(M+1:length(x)) x(1:M)];
h = x1.*sh;
y(k) = sum(h);
end
out1=length(y);
disp('Circular Convolution Output is:');disp(y);
subplot(2,2,1);stem((0:L1-1),x1);
xlabel('Time index n'); ylabel('Amplitude');
title('Input Signal x(n)'); grid;
subplot(2,2,2);stem((0:L2-1),x2);
xlabel('Time index n'); ylabel('Amplitude');
title('Impulse Response h(n)'); grid;
subplot(2,2,3);stem((0:out1-1),y);
xlabel('Time index n'); ylabel('Amplitude');
title('Output Obtained by Convolution');grid;
```



Questions:

Q1. What is the purpose of the command rem in the Program P2.2?

Q2. Explain how the Program P2.2 implements the circular time-shifting operation.

Q3. What is the purpose of the operator ~= in the function circonv?

Q4. Explain how the Program P2.2 implements the circular convolution operation.

3. Correlation of Signal Sequences

Correlation of two signals is to measure the degree to which the two signals are similar and thus to extract some information that depends to a large extent on the application. Correlation of signal is often encountered in radar, sonar, digital communication, geology and other areas in science and engineering.

The cross correlation of x(n) and y(n) is a sequence $\gamma_{xy}(l)$, which is defined as

$$\gamma_{xy}(l) = \sum_{n=-\infty}^{\infty} x[k]y[n-l] \qquad l = 0, \pm 1, \pm 2 \ldots \ldots$$

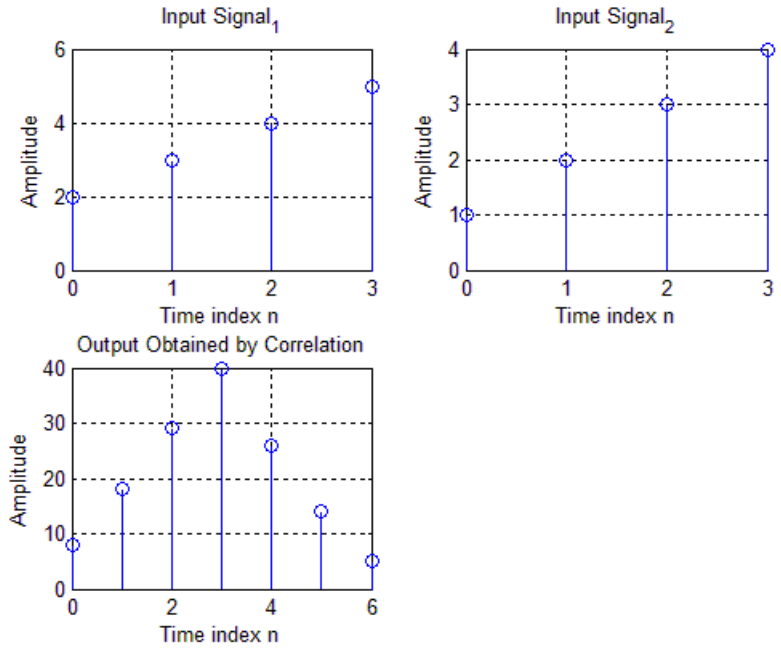
which can be alternately written as

$$\gamma_{xy}(l) = \sum_{n=-\infty}^{\infty} x[n+l]y[n] \qquad l = 0, \pm 1, \pm 2 \ldots \ldots$$

by a simple change of variables.

The following MATLAB program illustrates this approach.

```
% Program_3.1
% Cross Correlation of signal sequences
clf;
clear all;
x1 = input('Input sequence1:');
x2 = input('Input sequence2:');
xrev = fliplr(x2);
hlen = length(x1);
xlen = length(x2);
for i = 1:xlen
    for j = 1:hlen
        y(i, i+j-1) = xrev(i) * x1(j);
    end
end
y = sum(y);
outl=length(y);
disp('Cross Correlation Output is:');disp(y);
subplot(2,2,1);stem((0:xlen-1),x1);
xlabel('Time index n'); ylabel('Amplitude');
title('Input Signal_1'); grid;
subplot(2,2,2);stem((0:hlen-1),x2);
xlabel('Time index n'); ylabel('Amplitude');
title('Input Signal_2'); grid;
subplot(2,2,3);stem((0:outl-1),y);
xlabel('Time index n'); ylabel('Amplitude');
title('Output Obtained by Correlation');grid;
```



Questions:

Q1. Run Program P3.1 to generate y[n] obtained by the cross correlation of the sequences x1[n] and x2[n].

Q2. Modify Program P3.1 to develop the auto correlation.

4. Computation of DTFT & it's properties (and also relationship with DFT)

The discrete-time Fourier transform (DTFT) $X(e^{j\omega})$ of a sequence x[n] is defined by

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n}$$

In general $X(e^{j\omega})$ is a complex function of the real variable ω and can be written as

$$X(e^{j\omega}) = X_{re}(e^{j\omega}) + jX_{im}(e^{j\omega}),$$

where $X_{re}(e^{j\omega})$ and $X_{im}(e^{j\omega})$ are, respectively, the real and imaginary parts of $X(e^{j\omega})$, and are real functions of ω. $X(e^{j\omega})$ can alternately be expressed in the form

$$X(e^{j\omega}) = |X(e^{j\omega})|e^{j\theta(\omega)},$$

The quantity $|X(e^{j\omega})|$ is called the magnitude function and the quantity θ(ω) is called the phase function , with both functions again being real functions of ω. In many applications, the Fourier transform is called the Fourier spectrum and, likewise, $|X(e^{j\omega})|$ and θ(ω) are referred to as the magnitude spectrum and phase spectrum, respectively.

- The DTFT $X(e^{j\omega})$ is a periodic continuous function in ω with a period 2π.
- For a real sequence x[n], the real part $X_{re}(e^{j\omega})$ of its DTFT and the magnitude function $|X(e^{j\omega})|$ are even functions of ω, whereas the imaginary part Xim(ejω) and the phase function θ(ω) are odd functions of ω.
- The inverse discrete-time Fourier transform x[n] of $X(e^{j\omega})$ is given by

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega$$

- The Fourier transform $X(e^{j\omega})$ of a sequence x[n] exists if x[n] is absolutely summable, that is,

$$\sum_{n=-\infty}^{\infty} |x(n)| < \infty$$

Discrete Fourier Transform

The N-point discrete Fourier transform (DFT) of a finite-length sequence x[n], defined for $0 \le n \le N-1$, is given by

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn}, \qquad k = 0,1,2 \dots \dots \dots \dots N-1$$

Where $W_N = e^{-j\frac{2\pi}{N}}$

1. The N-point DFT X[k] of a length-N sequence x[n], n = 0, 1, . . . , N−1, is simply the frequency samples of its DTFT $X(e^{j\omega})$ evaluated at N uniformly spaced frequency points, $\omega = \omega_k = 2\pi k/N$, k = 0, 1, . . . , N − 1, that is,

$$X[k] = X(ej\omega)|_{\omega=2\pi k/N},\ k= 0, 1, \ldots, N - 1.$$

2. The inverse discrete Fourier transform x[n] of X(k) is given by

$$x(n) = \sum_{k=0}^{N-1} X(k)W_N^{-kn}, \qquad n = 0,1,2 \ldots \ldots \ldots \ldots N-1$$

The DFT satisfies a number of useful properties that are often utilized in a number of applications.

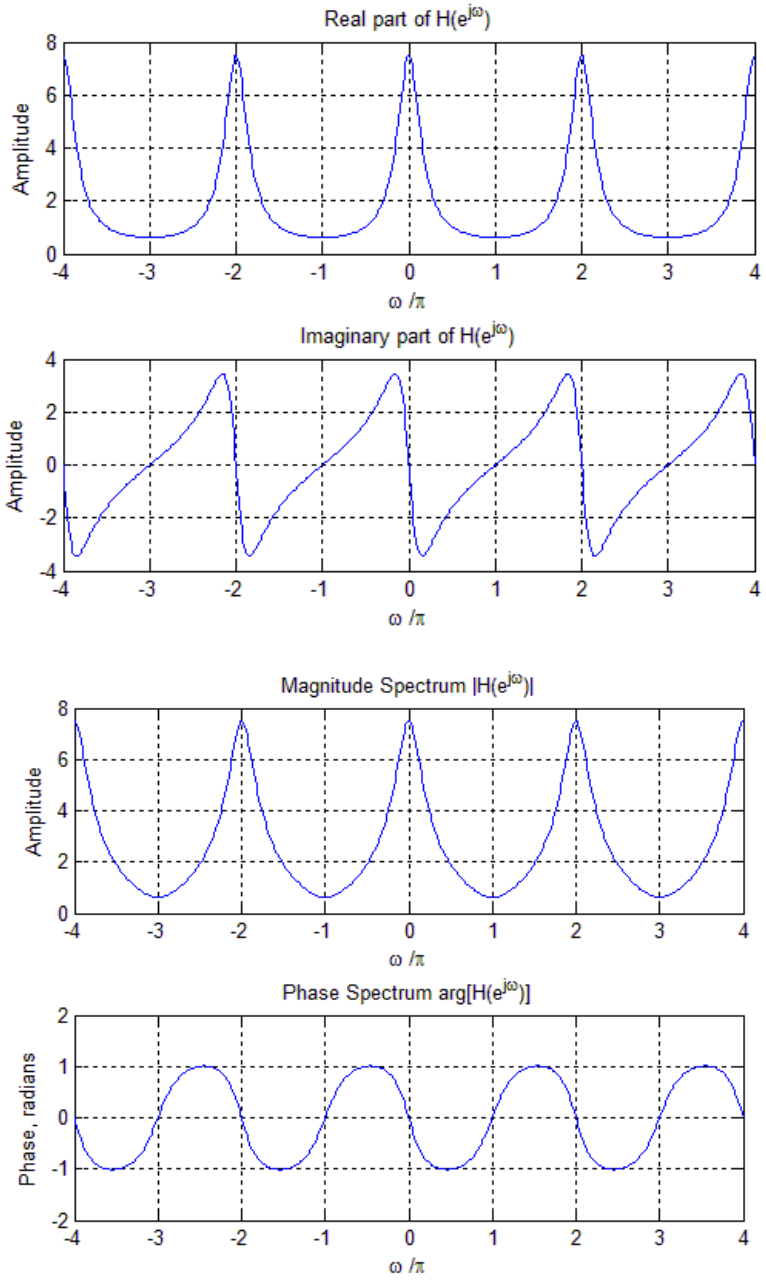
1) ***Circular Time-Shifting Property*** – If $G[k]$ denotes the N-point DFT of a length-N sequence $g[n]$, then the N-point DFT of the circularly time-shifted sequence $g[(n - n_o)_N]$ is given by $W_N^{kn_0}G(k)$ *where* $W_N = e^{-j\frac{2\pi}{N}}$.
2) ***Circular Frequency-Shifting Property*** – If $G[k]$ denotes the N-point DFT of a length-N sequence $g[n]$, then the N-point DFT of the sequence $W_N^{-k_0 n}g(n)$ is given by $G[(k - k_o)_N]$.
3) ***Circular Convolution Property*** – If $G[k]$ and $H[k]$ denote the N-point DFTs of the length-N sequences $g[n]$ and $h[n]$, respectively, then the N-point DFT of the circularly convolved sequence $g[n]* h[n]$ is given by $G[k]H[k]$.
4) ***Parseval's Relation*** – If $G[k]$ denotes the N-point DFT of a length-N sequence $g[n]$, then

$$\sum_{n=0}^{N-1}|g(n)|^2 = \frac{1}{N}\sum_{k=0}^{N-1}|G(k)|^2$$

Program P4.1 can be used to evaluate and plot the DTFT equation.

```
% Program P4_1
% Evaluation of the DTFT
clf;
% Compute the frequency samples of the DTFT
w = -4*pi:8*pi/511:4*pi;
num = [2 1];den = [1 -0.6];
h = freqz(num, den, w);
% Plot the DTFT
subplot(2,1,1)
plot(w/pi,real(h));grid
title('Real part of H(e^{j\omega})')
xlabel('\omega /\pi');
ylabel('Amplitude');
subplot(2,1,2)
plot(w/pi,imag(h));grid
title('Imaginary part of H(e^{j\omega})')
xlabel('\omega /\pi');
ylabel('Amplitude');
pause
```

```
subplot(2,1,1)
plot(w/pi,abs(h));grid
title('Magnitude Spectrum |H(e^{j\omega})|')
xlabel('\omega /\pi');
ylabel('Amplitude');
subplot(2,1,2)
plot(w/pi,angle(h));grid
title('Phase Spectrum arg[H(e^{j\omega})]')
xlabel('\omega /\pi');
ylabel('Phase, radians');
```



Questions:

Q1. What is the expression of the DTFT being evaluated in Program P4.1? What is the function of the MATLAB command pause?

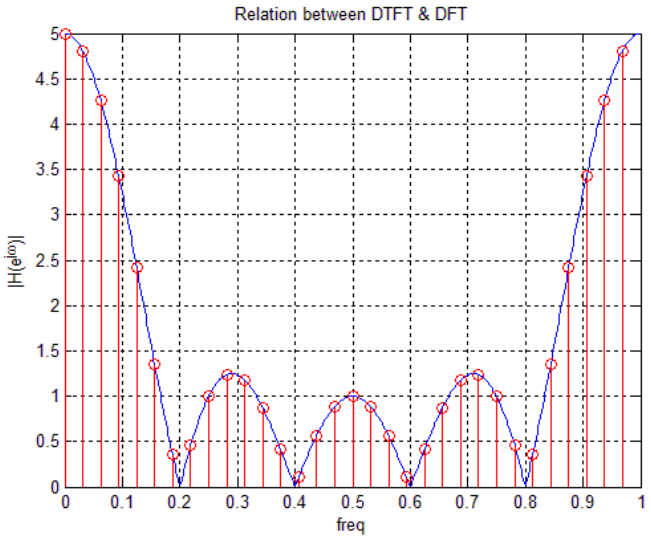
Q2. Run Program P4.1 and compute the real and imaginary parts of the DTFT, and the magnitude and phase spectra . Is the DTFT a periodic function of ω? If it is, what is the period? Explain the type of symmetries exhibited by the four plots.
Q3. Modify Program P4.1 to evaluate in the range $0 \le \omega \le \pi$ the following DTFT:

$$U(e^{j\omega}) = \frac{0.7 - 0.5e^{-j\omega} + 0.3e^{-j2\omega} + e^{-j3\omega}}{1 + 0.3e^{-j\omega} - 0.5e^{-j2\omega} + 0.7e^{-j3\omega}}$$

and repeat Question **Q2**. Comment on your results. Can you explain the jump in the phase spectrum ? The jump can be removed using the MATLAB command unwrap. Evaluate the phase spectrum with the jump removed.
Q4. Modify Program P4.1 to evaluate the DTFT of the following finite-length sequence: g[n]=[1 3 5 7 9 11 13 15 17], and repeat Question **Q2**. Comment on your results. Can you explain the jumps in the phase spectrum?
Q5. How would you modify Program P4.1 to plot the phase in degrees?

Program P4.2 can be used to relate DTFT with DFT.

```
% Program_P4.2
% DTFT and DFT relationship
clc;clear all;close all;
k=0:31;
x=[1 1 1 1 1 zeros(1,27)];%cos(2*pi*k*3/16);
[h w]=freqz(x,1,'whole');
plot(w/(2*pi),abs(h));grid;
xlabel('freq');ylabel('|H(e^{j\omega})|');
title('Relation between DTFT & DFT');
X=fft(x); hold on;
stem(k/32, abs(X),'ro');grid;
```



Questions:
Q1. Run Program P4.2 to obtain the relationship between DTFT & DFT.
Q2. Modify the command stem by plot in Program P4.2 and comment on it.

Program P4.3 can be used to verify Parseval's realtion property of DFT.

```
% Program P4.3
% Parseval's Relation
x = [(1:128) (128:-1:1)];
XF = fft(x);
a = sum(x.*x)
b = round(sum(abs(XF).^2)/256)
```

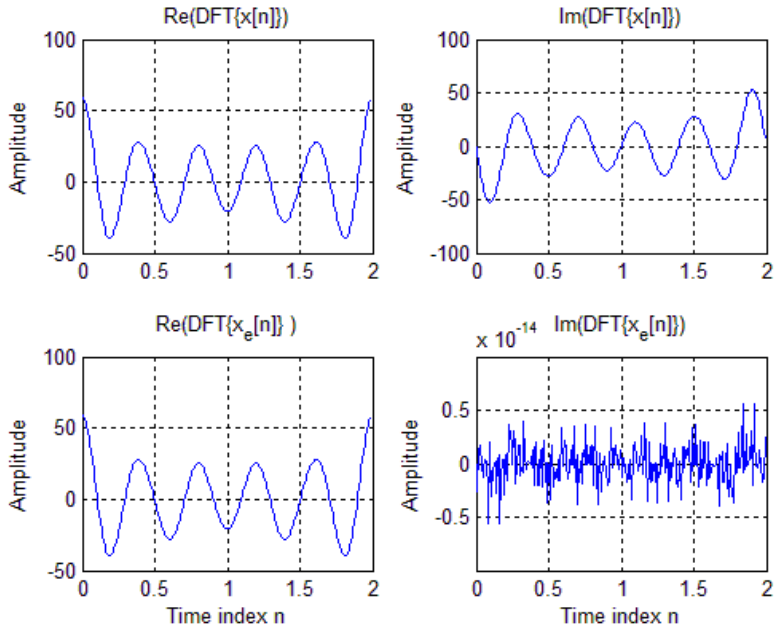
Questions:

Q1. Run Program P4.3. Do you get the same values for a and b?

Q2. Modify the program in such a way that you do not have to use the command abs(XF). Use the MATLAB command conj(x) to compute the complex conjugate of x.

Program P4.4 can be used to verify the relation between the DFT of a real sequence, and the DFTs of its periodic even and the periodic odd parts.

```
% Program P4.4
% Relations between the DFTs of the Periodic Even
% and Odd Parts of a Real Sequence
x = [1 2 4 2 6 32 6 4 2 zeros(1,247)];
x1 = [x(1) x(256:-1:2)];
xe = 0.5 *(x + x1);
XF = fft(x);
XEF = fft(xe);
clf;
k = 0:255;
subplot(2,2,1);
plot(k/128,real(XF)); grid;ylabel('Amplitude');
title('Re(DFT\{x[n]\})');
subplot(2,2,2);
plot(k/128,imag(XF)); grid; ylabel('Amplitude');
title('Im(DFT\{x[n]\})');
subplot(2,2,3);
plot(k/128,real(XEF)); grid;
xlabel('Time index n'); ylabel('Amplitude');
title('Re(DFT\{x_{e}[n]\} )');
subplot(2,2,4);
plot(k/128,imag(XEF)); grid;
xlabel('Time index n');ylabel('Amplitude');
title('Im(DFT\{x_{e}[n]\})');
```



Questions:

Q1. What is the relation between the sequences x1[n] and x[n]?

Q2. Run Program P4.4. The imaginary part of XEF should be zero as the DFT of the periodic even part is simply the real part of XEF of the original sequence. Can you verify that? How can you explain the simulation result?

Q3. Modify the program to verify the relation between the DFT of the periodic odd part and the imaginary part of XEF.

5. System Response & Stability Analysis using Z-transform

The Z-transform X(z) of a sequence x[n] is defined by

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

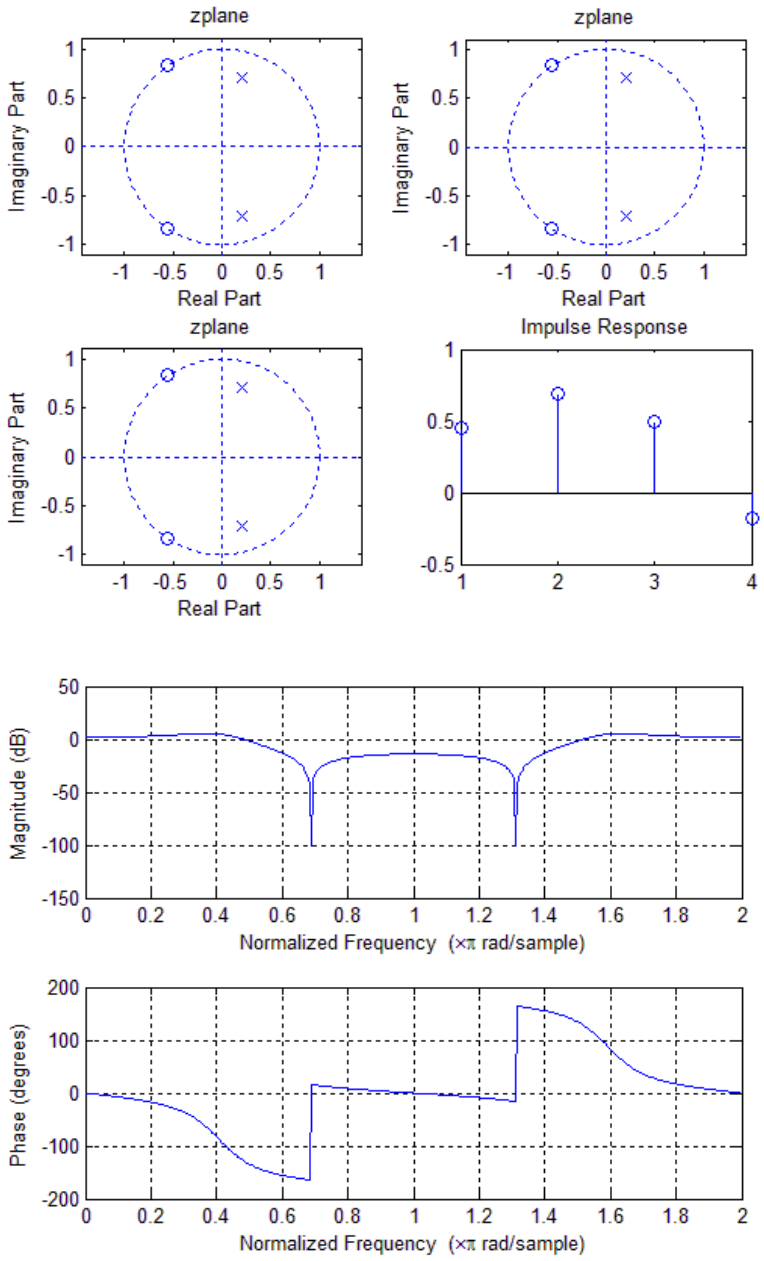
- The Z-transform is used for analysis of Discrete tiem signals & systems.
- A Causal & Stable system must have a system function that converges for |z| > r < 1. Since the ROC cannot contain any poles of H(z), it follows that a causal linear time invariant system is BIBO stable if and only if all the poles of H(z) are inside the unit circle.
- For a sytem to be stable, it has to satisfy the condition given below

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty$$

Program P5.1 can be used to analyse the system using z-transform.

```
% Program P5.1
clc;
clear all;
close all;
% plotting poles & zeros
num=input('Enter the numerator coefficients:'); %[0.45 0.5 0.45];
den=input('Enter the denominator coefficients:'); % [1 -0.43
0.56];
subplot(2,2,1);zplane(num,den);
title('zplane');
num_a=roots(num);
den_b=roots(den);
subplot(2,2,2);zplane(num_a,den_b);
title('zplane');
[z, p, k] = tf2zp(num,den);
subplot(2,2,3);zplane(z,p);
title('zplane');
disp('The poles are:');disp(p);
disp('The zeros are:');disp(z);
% FACTORED FORM OF Z-TRANSFORM
sos = zp2sos(z,p,k);
% FINDING IMPULSE RESPONSE
N = max(length(num),length(den));
[g t]=impz(num,den,N+1);
subplot(2,2,4);stem(g);
title('Impulse Response');
disp('Impulse response of a system is:');disp(g);
% GETTING PARTIAL FRACTION FORM OF Z-TRANSFORM
[r s t]=residue(num,den);
```

```
% Z-TRANSFORM
syms z n a;
eq=2^n;
ztrans_out=ztrans(eq);
% inverse Z-transform
inv_ztransform=iztrans(ztrans_out);
% frequency response
figure,freqz(num,den,'whole');
```



Questions:

Q1. Write a MATLAB program to compute and display the poles and zeros, to compute and display the factored form, and to generate the pole-zero plot of a *z*-transform that is a ratio of two polynomials in z^{-1}. Using this program, analyze the *z*-transform

$$G(z) = \frac{2 + 5z^{-1} + 9z^{-2} + 5z^{-3} + 3z^{-4}}{5 + 45z^{-1} + 2z^{-2} + z^{-3} + z^{-4}}$$

Q2. From the pole-zero plot generated in Question Q1., determine the number of regions of convergence (ROC) of $G(z)$. Show explicitly all possible ROCs . Can you tell from the pole-zero plot whether or not the DTFT exists?

Q3. Write a MATLAB program to compute the first L samples of the inverse of a rational z-transform where the value of L is provided by the user through the command input. Using this program compute and plot the first 50 samples of the inverse of $G(z)$ in Q1. Use the command stem for plotting the sequence generated by the inverse transform.

Q4. Write a MATLAB program to determine the partial-fraction expansion of a rational z-transform. Using this program determine the partial-fraction expansion of $G(z)$ in Q1 and then its inverse z-transform $g[n]$ in closed form. Assume $g[n]$ to be a causal sequence.

6. DFT Computation by FFT

Why the FFT?

If you look at the equation for the *Discrete Fourier Transform* you will see that it is quite complicated to work out as it involves many additions and multiplications involving complex numbers. Even a simple eight sample signal would require 64 complex multiplications and 56 complex additions to work out the DFT. At this level it is still manageable, however a realistic signal could have 1024 samples which requires over 10,48,576 complex multiplications. As you can see the number of calculations required soon mounts up to unmanageable proportions.

The Fast Fourier Transform is a simply a method of laying out the computation, which is much faster for large values of N, where N is the number of samples in the sequence. It is an ingenious way of achieving rather than the DFT's clumsy timing.

The idea behind the FFT is the *divide and conquer* approach, to break up the original N point sample into two (N / 2) sequences. This is because a series of smaller problems is easier to solve than one large one. The DFT requires N^2 complex multiplications and N(N-1) complex additions as opposed to the FFT's approach of breaking it down into a series of 2 point samples which only require 1 multiplication and 2 additions and the recombination of the points which is minimal.

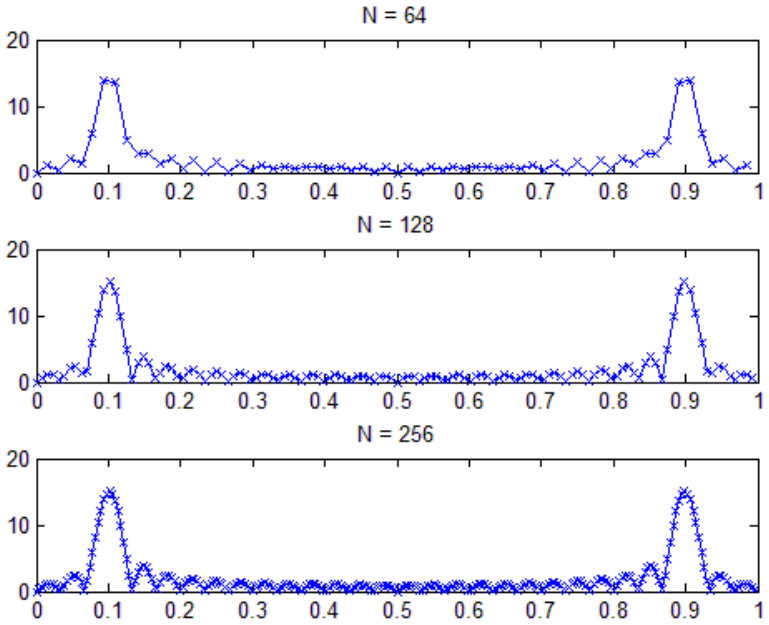
For example *Seismic Data* contains hundreds of thousands of samples and would take months to evaluate the DFT. Therefore we use the FFT.

The typical syntax for computing the FFT of a signal is FFT(x,N) where x is the signal, x[n], you wish to transform, and N is the number of points in the FFT. N must be at least as large as the number of samples in x[n]. To demonstrate the effect of changing the value of N, sythesize a cosine with 30 samples at 10 samples per period.

Program P6.1 can be used to analyse the FFT for different values of N.

```
% Program P6.1
clc;
clear all;
close all;
n = [0:29];
x = cos(2*pi*n/10);
N1 = 64;
N2 = 128;
N3 = 256;
X1 = abs(fft(x,N1));
X2 = abs(fft(x,N2));
X3 = abs(fft(x,N3));
F1 = [0 : N1 - 1]/N1;
F2 = [0 : N2 - 1]/N2;
F3 = [0 : N3 - 1]/N3;
subplot(3,1,1)
plot(F1,X1,'-x'),title('N = 64'),axis([0 1 0 20])
subplot(3,1,2)
plot(F2,X2,'-x'),title('N = 128'),axis([0 1 0 20])
```

```
subplot(3,1,3)
plot(F3,X3,'-x'),title('N = 256'),axis([0 1 0 20])
```



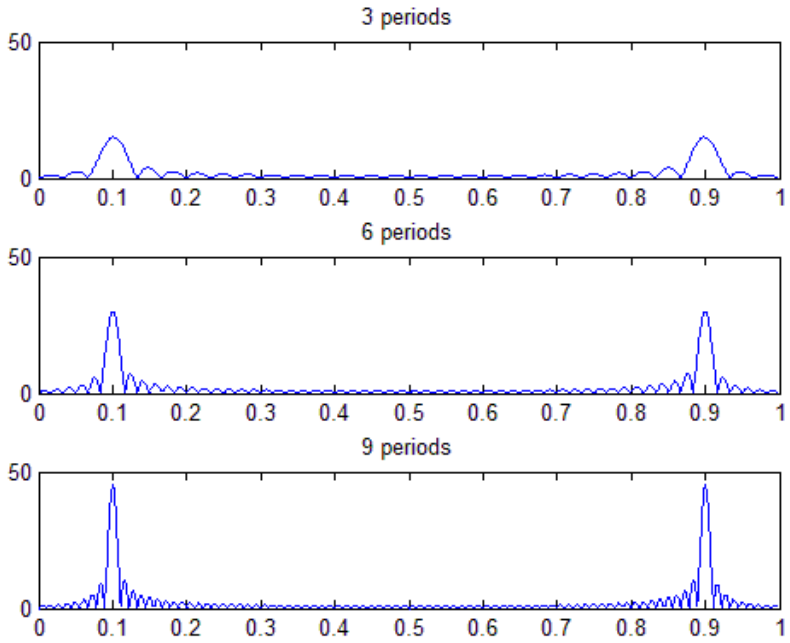
Questions:
Upon examining the plot one can see that each of the transforms adheres to the same shape, differing only in the number of samples used to approximate that shape.
Q1. What happens if N is the same as the number of samples in x[n]? To find out, set N1 = 30. What does the resulting plot look like? Why does it look like this?
Q2. What happens if number of samples is increased from 30 to 60(or any higher value).

In the the last example the length of x[n] was limited to 3 periods in length. Now, let's choose a large value for N (for a transform with many points), and vary the number of repetitions of the fundamental period.

Program P6.2 can be used to analyse the FFT with more periods.

```
% Program P6.2
clc; clear all; close all;
n = [0:29];
x1 = cos(2*pi*n/10); % 3 periods
x2 = [x1 x1]; % 6 periods
x3 = [x1 x1 x1]; % 9 periods
N = 2048;
X1 = abs(fft(x1,N));
X2 = abs(fft(x2,N));
X3 = abs(fft(x3,N));
F = [0:N-1]/N;
subplot(3,1,1)
plot(F,X1),title('3 periods'),axis([0 1 0 50])
subplot(3,1,2)
```

```
plot(F,X2),title('6 periods'),axis([0 1 0 50])
subplot(3,1,3)
plot(F,X3),title('9 periods'),axis([0 1 0 50])
```



The previous code will produce three plots. The first plot, the transform of 3 periods of a cosine, looks like the magnitude of 2 sincs with the center of the first sinc at 0.1fs and the second at 0.9fs.

The second plot also has a sinc-like appearance, but its frequency is higher and it has a larger magnitude at 0.1fs and 0.9fs.

Similarly, the third plot has a larger sinc frequency and magnitude. As x[n] is extended to an large number of periods, the sincs will begin to look more and more like impulses.

If noticed, a sinusoid transformed to an impulse, why do we have sincs in the frequency domain? When the FFT is computed with an N larger than the number of samples in x[n], it fills in the samples after x[n] with zeros. The program P6.2 an x[n] that was 30 samples long, but the FFT had an N = 2048. When Matlab computes the FFT, it automatically fills the spaces from n = 30 to n = 2047 with zeros. This is like taking a sinusoid and mulitipying it with a rectangular box of length 30. A multiplication of a box and a sinusoid in the time domain should result in the convolution of a sinc with impulses in the frequency domain. Furthermore, increasing the width of the box in the time domain should increase the frequency of the sinc in the frequency domain. The previous Matlab experiment supports this conclusion.

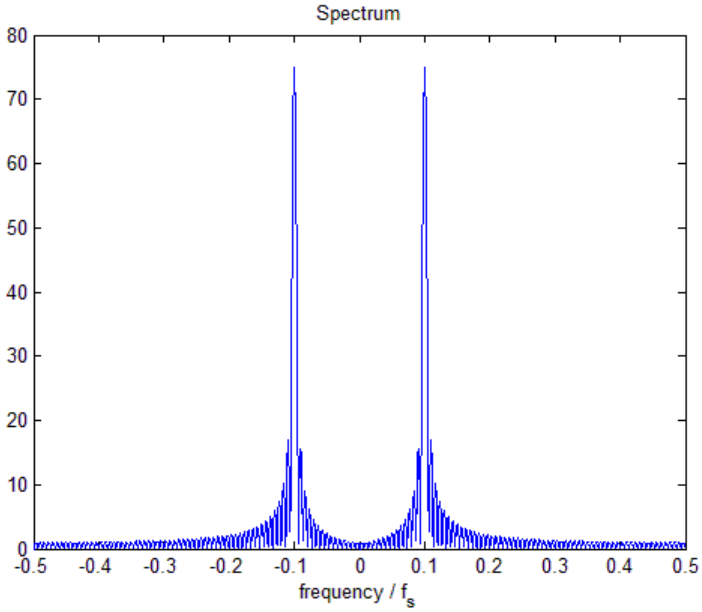
Spectrum Analysis with the FFT and Matlab

The FFT does not directly give you the spectrum of a signal. As we have seen with the last two experiments, the FFT can vary dramatically depending on the number of points (N) of the FFT, and the number of periods of the signal that are represented. There is another problem as well. The FFT contains information between 0 and fs, however, we know that the sampling frequency must be at least twice the highest frequency component. Therefore, the signal's spectrum should be entirly below fs/2 , the Nyquist frequency.

Recall also that a real signal should have a transform magnitude that is symmetrical for for positive and negative frequencies. So instead of having a spectrum that goes from 0 to fs, it would be more appropriate to show the spectrum from –fs/2 to fs/2. This can be accomplished by using Matlab's ***fftshift*** function as the following code demonstrates.

Program P6.3 can be used to obtain the spectrum.

```
% Program P6.3
clc; clear all; close all;
n = [0:149];
x1 = cos(2*pi*n/10);
N = 2048;
X = abs(fft(x1,N));
X = fftshift(X);
F = [-N/2:N/2-1]/N;
plot(F,X),
xlabel('frequency / f_s')
title('Spectrum');
```

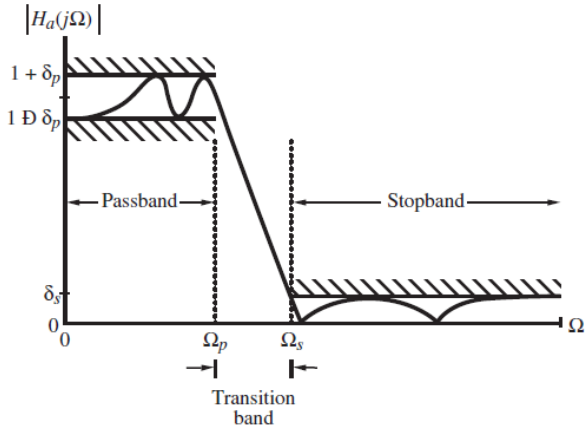


7. IIR Filter Design

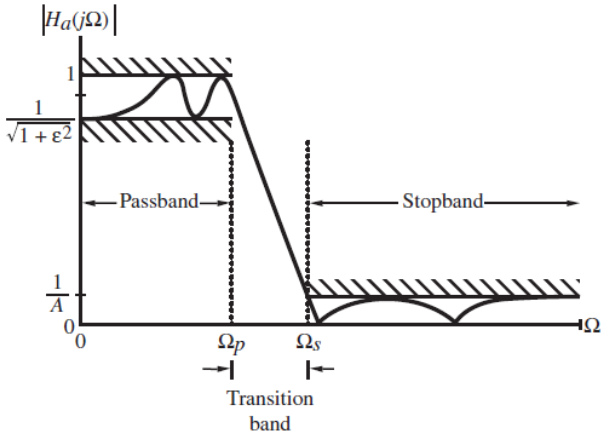
Analog IIR Filter Design

The filter specifications are usually stated in terms of the magnitude response. For example, the magnitude |Ha(jΩ)| of an analog lowpass filter is usually specified as indicated in the figure below. In the *passband*, defined by $0 \le \Omega \le \Omega p$, we require

$$1 - \delta p \le |H_a(j\Omega)| \le 1 + \delta p, \text{ for } |\Omega| \le \Omega p,$$



or, in otherwords, the magnitude approximates unity within an error of ±*δp*. In the *stopband*, defined by $\Omega s \le |\Omega| \le \infty$, we require $|Ha(j\Omega)| \le \delta s$, for $\Omega s \le |\Omega| \le \infty$, implying that the magnitude approximate zero within an error of *δs*. The frequencies *Ωp* and *Ωs* are called, respectively, the *passband edge frequency* and the *stopband edge frequency*. The maximum limits of the tolerances in the passband and stopband, *δp* and *δs*, are called *ripples* .



- In most applications, the analog filter specifications are given as indicated in figure above. Here, in the *passband* defined by $0 \le \Omega \le \Omega p$, the maximum and the minimum values of the magnitude are, respectively, unity and $1/\sqrt{1+\epsilon^2}$. The *peak passband ripple* is $R_p = 20 \log_{10} \sqrt{1+\epsilon^2}$ dB

  The maximum ripple in the *stopband*, defined by $\Omega s \le \Omega \le \infty$, is denoted by 1/*A*. The *minimum stopband attenuation* is therefore given by *Rs* = 20 log10 *A* dB.
- *Butterworth Approximation*. The magnitude-squared response of an analog lowpass Butterworth filter *Ha*(*s*) of *N*th order is given by

$$|H_a(j\Omega)|^2 = \frac{1}{1+\left(\frac{\Omega}{\Omega_c}\right)^{2N}}$$

The Butterworth filter is said to have a maximally-flat magnitude at Ω = 0 as the first $2N$ – 1 derivatives of $|\text{Ha}(\text{j}\Omega)|^2$ at Ω = 0 are equal to zero. At Ω = Ω*c*, the gain $G(\Omega) = 10\log10|\text{Ha}(\text{j}\Omega)|^2$ is 3 dB less than that at Ω = 0 and, hence, Ω*c* is called the *3-dB cutoff frequency*.

The two parameters completely characterizing a Butterworth lowpass filter are therefore the 3-dB cutoff frequency Ω*c* and the order *N* . These are determined from the specified *passband edge* Ω*p*, the *stopband edge* Ω*s*, the *peak passband ripple Rp* in dB, and the *minimum stopband attenuation Rs* in dB.

The transfer function of the Butterworth lowpass filter is of the form

$$H_a(s) = \frac{K}{\sum_{l=0}^{N} a_l s^l} = \frac{K}{\prod_{l=1}^{N}(s-p_l)}$$

- *Type 1 Chebyshev Approximation*. The Type 1 Chebyshev lowpass transfer function $Ha(s)$ has a magnitude response given by

$$|H_a(j\Omega)|^2 = 1/(1+\epsilon^2 T_N^2(\omega-\Omega_p)$$

where $T_N(\Omega)$ is the Chebyshev polynomial of order *N*:

$$T_N(\Omega) = \begin{cases} \cos(N\cos^{-1}\Omega), & |\Omega| \le 1, \\ \cosh(N\cosh^{-1}\Omega), & |\Omega| > 1. \end{cases}$$

The above polynomial can also be derived via a recurrence relation given by

$$T_r(\Omega) = 2\Omega T_{r-1}(\Omega) - T_{r-2}(\Omega),\ r \ge 2,$$

with $T_0(\Omega) = 1$ and $T_1(\Omega) = \Omega$.

The order *N* of the Type 1 Chebyshev lowpass filter is determined from the specified passband edge Ω*p* , the stopband edge Ω*s*, the peak passband ripple *Rp* in dB, and the minimum stopband attenuation *Rs* in dB.

- *Type 2 Chebyshev Approximation*. The square-magnitude response expression here is given by

$$|H_a(j\Omega)|^2 = \frac{1}{1+\epsilon^2\left[\frac{T_n\left(\frac{\Omega_s}{\Omega_p}\right)}{T_n\left(\frac{\Omega_s}{\Omega}\right)}\right]^2}$$

The transfer function of a Type 2 Chebyshev lowpass filter is no longer an all-pole function, as it has both poles and zeros. It is of the form

$$H_a(s) = K\frac{\sum_{l=0}^{N} b_l s^l}{\sum_{l=0}^{N} a_l s^l} = K\frac{\prod_{l=1}^{N}(s-z_l)}{\prod_{l=1}^{N}(s-p_l)}$$

The zeros *h*ere are on the *j*Ω-axis. The order*N* of the Type 2 Chebyshev lowpass filter is determined from the specified passband edge Ω*p* , the stopband edge Ω*s*, the peak passband ripple *Rp* in dB, and the minimum stopband attenuation *Rs* in dB.

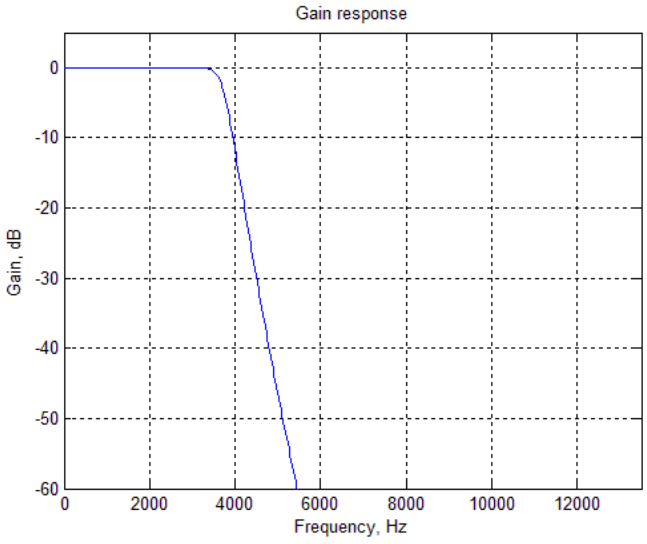
- *Elliptic Approximation*. The square-magnitude response of an elliptic lowpass filter is given by

$$|H_a(j\Omega)|^2 = \frac{1}{1 + \epsilon^2 R_n^2\left(\frac{\Omega}{\Omega_p}\right)}$$

where $RN(\Omega)$ is a rational function of order N satisfying the property $R_N(1/\Omega) = 1/R_N(\Omega)$ with the roots of its numerator lying within the interval $0 < \Omega < 1$ and the roots of its denominator lying in the interval $1 < \Omega < \infty$. The order N of the elliptic lowpass filter is determined from the specified passband edge Ω*p*, the stopband edge Ω*s*, the peak passband ripple *Rp* in dB, and the minimum stopband attenuation *Rs* in dB.

Program P7.1 illustrates the design of analog Butterworth lowpass filter.

```
% Program P7.1
% Design of Analog Lowpass Filter
clf;
Fp = 3500;Fs = 4500;
Wp = 2*pi*Fp; Ws = 2*pi*Fs;
[N, Wn] = buttord(Wp, Ws, 0.5, 30,'s');
[b,a] = butter(N, Wn, 's');
wa = 0:(3*Ws)/511:3*Ws;
h = freqs(b,a,wa);
plot(wa/(2*pi), 20*log10(abs(h)));grid
xlabel('Frequency, Hz');ylabel('Gain, dB');
title('Gain response');
axis([0 3*Fs -60 5]);
```



Questions:

Q7.1. What are the passband ripple Rp in dB and the minimum stopband attenuation Rs in dB in Program P7.1? What are the passband and the stopband edge frequencies in Hz?

Q7.2. Run Program P7.1 and display the gain response. Does the filter as designed meet the given specifications? What are the filter order *N* and the 3-dB cutoff frequency in Hz of the filter as designed?

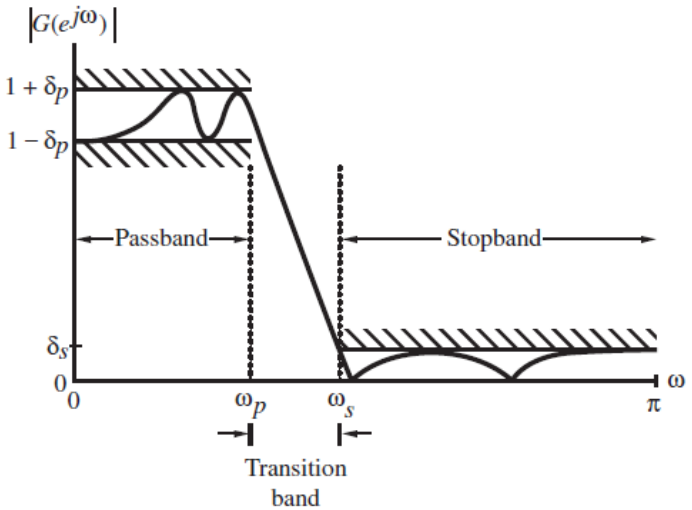
Q7.3. Using cheb1ord and cheby1 modify Program P7.1 to design a Type 1 Chebyshev lowpass filter meeting the same specifications as in Program P7.2. Run the modified program and display the gain response. Does the filter as designed meet the given specifications? What are the filter order *N* and the passband edge frequency in Hz of the filter as designed?

Q7.4. Using cheb2ord and cheby2 modify Program P7.1 to design a Type 2 Chebyshev lowpass filter meeting the same specifications as in Program P7.1. Run the modified program and display the gain response. Does the filter as designed meet the given specifications? What are the filter order *N* and the stopband edge frequency in Hz of the filter as designed?

Q7.5. Using ellipord and ellip modify Program P7.1 to design an elliptic lowpass filter meeting the same specifications as in Program P7.1. Run the modified program and display the gain response. Does the filter as designed meet the given specifications? What are the filter order *N* and the passband edge frequency in Hz of the filter as designed?

Digital IIR Filter Design

The process of deriving the transfer function $G(z)$ whose frequency response $G(e^{j\omega})$ approximates the given frequency response specifications is called *digital filter design*. After $G(z)$ has been obtained, it is then realized in the form of a suitable filter structure. In this laboratory exercise you will learn how to design an IIR or FIR digital filter to meet a specified magnitude or gain response.



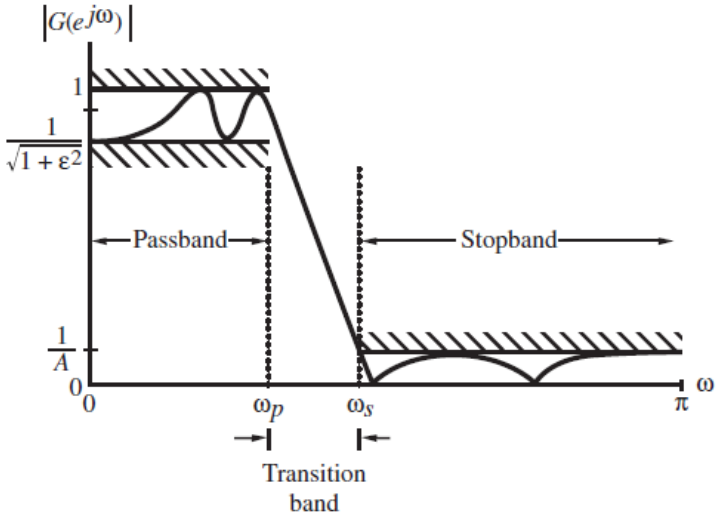
- The filter specifications are usually specified in terms of its magnitude response. For example, the magnitude $|G(e^{j\omega})|$ of a lowpass filter $G(z)$ is usually specified as indicated in the above figure. In the *passband* defined by $0 \le \omega \le \omega_p$, we require

$$1 - \delta p \le |G(ej\omega)| \le 1 + \delta p, \text{ for } |\omega| \le \omega p,$$

or in otherwords, the magnitude approximates unity within an error of $\pm\delta p$. In the *stopband*, defined by $\omega s \le |\omega| \le \pi$, we require

$$|G(e^{j\omega})| \le \delta s, \text{ for } \omega s \le |\omega| \le \pi,$$

implying that the magnitude approximate zero within an error of δs. The frequencies ω_p and ω_s are, respectively, called the *passband edge frequency* and the *stopband edge frequency*. The maximum limits of the tolerances in the passband and stopband, δp and δs, are called *ripples*.



- In most applications, the digital filter specifications are given as indicated in the above figure. Here, in the *passband* defined by $0 \le \omega \le \omega_p$, the maximum and the minimum values of the magnitude are, respectively, unity and $1/\sqrt{1+\epsilon^2}$. The *peak passband ripple* in dB is

$$R_p = 20 \log_{10} \sqrt{1+\epsilon^2} \text{ dB.}$$

The maximum ripple in the *stopband*, defined by $\omega_s \le \omega \le \pi$, is denoted by $1/A$ and the *minimum stopband attenuation* in dB is given by $R_s = 20 \log_{10} A$ dB.

- If the passband edge frequency *Fp* and the stopband edge frequency *Fs* are specified in Hz along with the sampling rate *FT* of the digital filter, then the normalized angular edge frequencies in radians are given by

$$\omega_p = \frac{\Omega_p}{F_T} = \frac{2\pi F_p}{F_T} = 2\pi F_p T$$

$$\omega_s = \frac{\Omega_s}{F_T} = \frac{2\pi F_s}{F_T} = 2\pi F_s T$$

- The first step in the filter design process is the estimation of the order of the transfer function. For the design of an IIR digital lowpass filter $G(z)$ based on the conversion of an analog lowpass filter $H_a(s)$, an analytical formula exists for the estimation of the filter order.

- The most widely used approach to IIR filter design is based on the bilinear transformation from s-plane to z-plane given by

$$s = \frac{2}{T}\left(\frac{1 - z^{-1}}{1 + z^{-1}}\right)$$

Using the above transformation an analog transfer function$Ha(s)$ is converted into a digital transfer function $G(z)$ according to:

$$G(z) = H(z)|_{s=\frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)}$$

- For the bilinear transformation, the relation between the imaginary axis ($s = j\Omega$) in the s-plane and the unit circle ($z = e^{j\omega}$) in the z-plane is given by

$$\Omega = \tan(\omega/2),$$

which maps the entire imaginary axis in the s-plane to the unit circle in the z-plane introducing a distortion in the frequency axis called *warping*. To develop a digital filter meeting a specified magnitude response, the analog equivalents (Ωp and Ωs) of the critical band-edge frequencies (ω_p and ω_s) of the digital filter are first obtained using the relation of the above equation, the analog prototype $H_a(s)$ is then designed using the prewarped critical frequencies, and $H_a(s)$ is transformed using the bilinear transformation to obtain the desired digital filter transfer function $G(z)$.

Design Flow:

The most common method of IIR filter design is based on the bilinear transformation of a prototype analog transfer function.

The analog transfer function is usually one of the following types:

- Butterworth,
- Type 1 Chebyshev,
- Type 2 Chebyshev,
- Elliptic.

The difference between these filter types can be explained by considering the analog lowpass filter.

- The Butterworth lowpass transfer function has a maximally-flat magnitude response at dc, that is, $\Omega = 0$, and a monotonically decreasing magnitude response with increasing frequency.
- The Type 1 Chebyshev lowpass transfer function has an equiripple magnitude response in the passband and a monotonically decreasing magnitude response with increasing frequency outside the passband.
- The Type 2 Chebyshev lowpass transfer function has a monotonically decreasing magnitude response in the passband with increasing frequency and an equiripple magnitude response in the stopband.
- The elliptic lowpass transfer function has equiripple magnitude responses both in the passband and in the stopband.

Estimation of Order of IIR Filter

The first step in the filter design process is to choose the type of filter approximation to be employed and then to estimate the order of the transfer function from the filter specifications.
The MATLAB command for estimating the order of a Butterworth filter is

[N, Wn] = buttord(Wp, Ws, Rp, Rs),

where the input parameters are the normalized passband edge frequency Wp, the normalized stopband edge frequency Ws, the passband ripple Rp in dB, and the minimum stopband attenuation Rs in dB. Both Wp and Ws must be a number between 0 and 1 with the sampling frequency assumed to be 2 Hz. The output data are the lowest order N meeting the specifications and the normalized cutoff frequency Wn. If Rp = 3 dB, then Wn = Wp. buttord can also be used to estimate the order of a highpass, a bandpass, and a bandstop Butterworth filter. For a highpass filter design,Wp > Ws. For bandpass and bandstop filter designs, Wp and Ws are two-element vectors specifying both edge frequencies, with the lower edge frequency being the first element of the vector. In the latter cases, Wn is also a two-element vector.

For estimating the order of a Type 1 Chebyshev filter, the MATLAB command is

[N, Wn] = cheb1ord(Wp, Ws, Rp, Rs)

and for designing a Type 2 Chebyshev filter, the MATLAB command for estimating the order is

[N, Wn] = cheb2ord(Wp, Ws, Rp, Rs).

Finally, in the case of an elliptic filter design, the command is

[N, Wn] = ellipord(Wp, Ws, Rp, Rs).

As before, Wp and Ws are the passband and stopband edge frequencies with values between 0 and 1. Likewise, Rp and Rs are the passband ripple and the minimum stopband attenuation in dB. N contains the estimated lowest order and Wn is the cutoff frequency. It should be noted that for bandpass and bandstop filter designs, the actual order of the transfer function obtained using the appropriate filter design command is 2N.

Questions:

Q7_6. Using MATLAB determine the lowest order of a digital IIR lowpass filter of all four types. The specifications are as follows: sampling rate of 40 kHz, passband edge frequency of 4 kHz, stopband edge frequency of 8 kHz, passband ripple of 0.5 dB, and a minimum stopband attenuation of 40 dB. Comment on your results.

Q7_7. Using MATLAB determine the lowest order of a digital IIR highpass filter of all four types. The specifications are as follows: sampling rate of 3,500 Hz, passband edge frequency of 1,050 Hz, stopband edge frequency of 600 Hz, passband ripple of 1 dB, and a minimum stopband attenuation of 50 dB. Comment on your results.

Q7_8. Using MATLAB determine the lowest order of a digital IIR bandpass filter of all four types. The specifications are as follows: sampling rate of 7 kHz, passband edge frequencies at 1.4 kHz and 2.1 kHz, stopband edge frequencies at 1.05 kHz and 2.45 kHz, passband ripple of 0.4 dB, and a minimum stopband attenuation of 50 dB. Comment on your results.

Q7_9. Using MATLAB determine the lowest order of a digital IIR bandstop filter of all four types. The specifications are as follows: sampling rate of 12 kHz, passband edge frequencies at 2.1 kHz and 4.5 kHz, stopband edge frequencies at 2.7 kHz and 3.9 kHz, passband ripple of 0.6 dB, and a minimum stopband attenuation of 45 dB. Comment on your results.

IIR Filter Design

After the filter type has been selected and its order estimated, the next step is to determine the transfer function of the filter. To this end MATLAB provides functions for all four types of filters. For designing Butterworth digital lowpass or bandpass filters, the command is

[num,den] = butter(N,Wn)

where the input parameters N and Wn are determined through the use of the function buttord, and the output is the vectors num and den containing, respectively, the coefficients of the numerator and denominator polynomials of the transfer function in ascending powers of z^{-1}. If Wn is a scalar, butter returns a lowpass transfer function of order N, and if Wn is a two-element vector, it returns a bandpass transfer function of order 2N.

For designing a Butterworth digital highpass filter of order N, the command is

[num,den] = butter(N,Wn,'high')

whereas, the command

[num,den] = butter(N,Wn,'stop')

returns the transfer function of a Butterworth bandstop filter of order 2N provided Wn is a two-element vector.

For designing a Type 1 Chebyshev digital filter, the commands are

[num,den] = cheby1(N,Rp,Wn)
[num,den] = cheby1(N,Rp,Wn,'filtertype')

For designing a Type 2 Chebyshev digital filter, the commands are

[num,den] = cheby2(N,Rs,Wn)
[num,den] = cheby2(N,Rs,Wn,'filtertype')

Finally, for designing an elliptic digital filter, the commands are

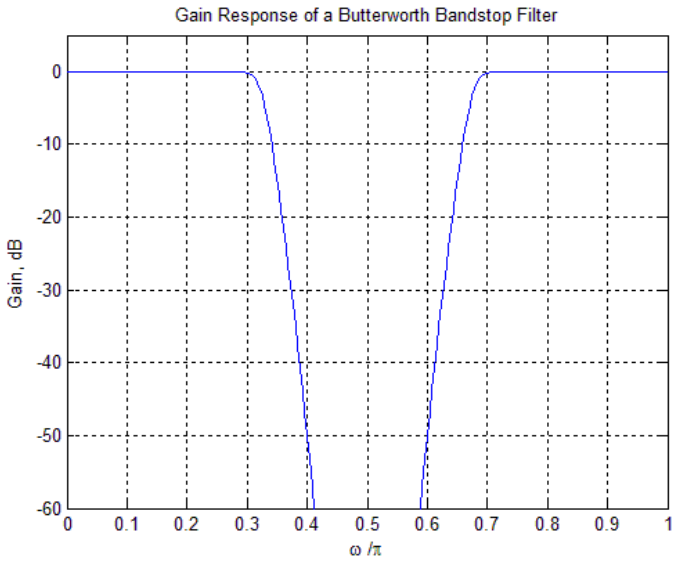
[num,den] = ellip(N,Rp,Rs,Wn)
[num,den] = ellip(N,Rp,Rs,Wn,'filtertype')

A lowpass transfer function of order N is returned in each case if Wn is a scalar, and a bandpass transfer function of order 2N is returned if Wn is a two-element vector. In each of the above commands, filtertype is high for designing a highpass filter with Wn being a scalar, and filtertype is stop for designing a bandstop filter with Wn being a two-element vector.

Program P7.2 illustrates the design of a Butterworth bandstop filter.

```
% Program P7.2
% Design of a Butterworth Bandstop Digital Filter
Ws = [0.4 0.6]; Wp = [0.3 0.7]; Rp = 0.4; Rs = 50;
% Estimate the Filter Order
[N1, Wn1] = buttord(Wp, Ws, Rp, Rs);
% Design the Filter
[num,den] = butter(N1,Wn1,'stop');
% Display the transfer function
disp('Numerator coefficients are ');disp(num);
disp('Denominator coefficients are ');disp(den);
% Compute the gain response
w = 0:pi/255:pi;
h = freqz(num,den,w);
```

```
g = 20*log10(abs(h));
% Plot the gain response
plot(w/pi,g);grid
axis([0 1 -60 5]);
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain Response of a Butterworth Bandstop Filter');
```



Questions:

Q7_10. Design the Butterworth bandstop filter by running Program P7.2. Write down the exact expression for the transfer function generated. What are the filter specifications? Does your design meet the specifications? Using MATLAB, compute and plot the filter's unwrapped phase response and the group delay response.

Q7_11. Modify Program P7.2 to design a Type 1 Chebyshev lowpass filter meeting the specifications given in Question Q7_6. Write down the exact expression for the transfer function generated. Does your design meet the specifications? Using MATLAB, compute and plot the filter's unwrapped phase response and the group delay response.

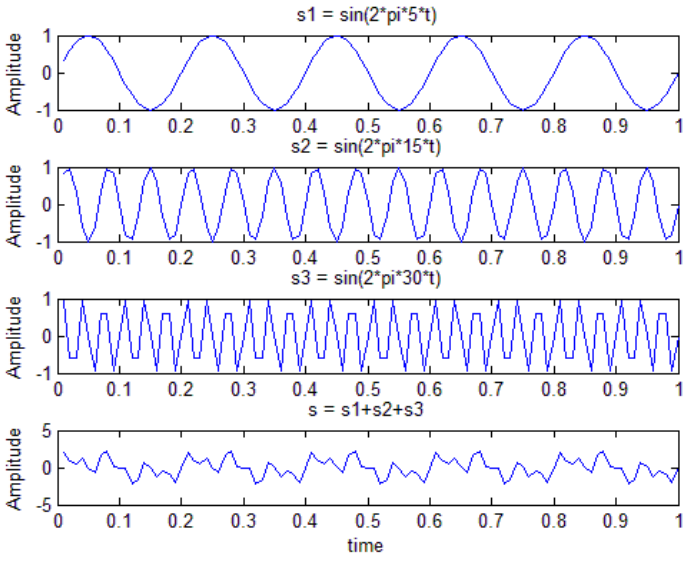
Q7_12. Modify Program P7.2 to design a Type 2 Chebyshev highpass filter meeting the specifications given in Question Q7_7. Write down the exact expression for the transfer function generated. Does your design meet the specifications? Using MATLAB, compute and plot the filter's unwrapped phase response and the group delay response.

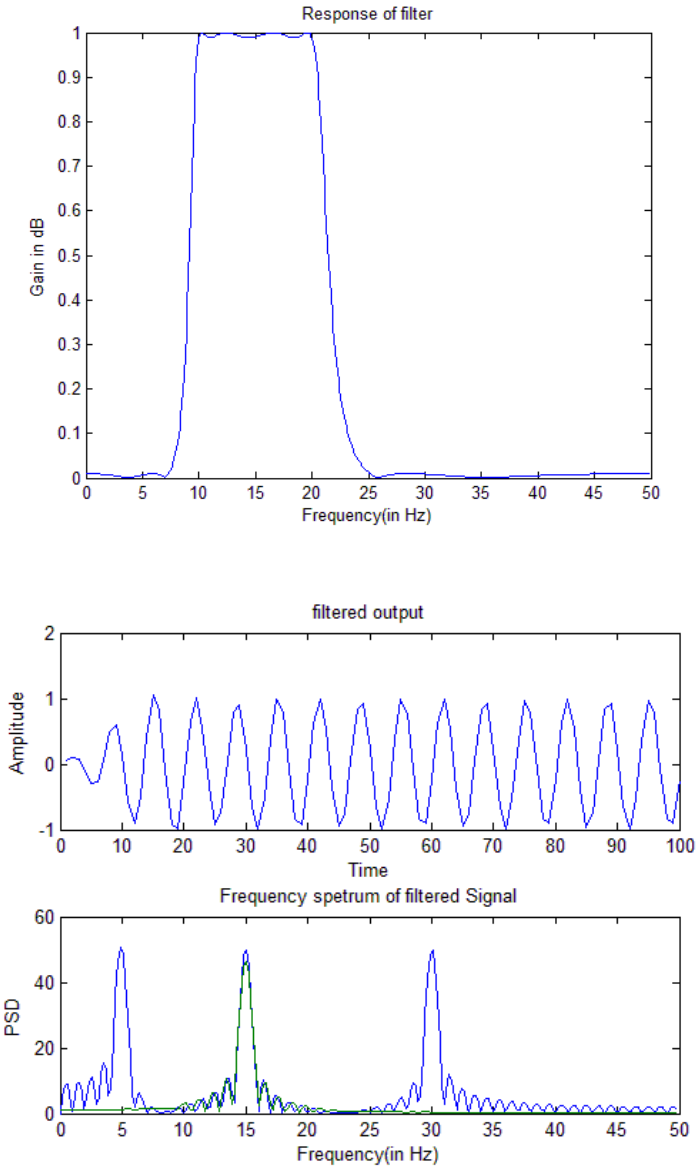
Q7_13. Modify Program P7.2 to design an elliptic bandpass filter meeting the specifications given in Question Q7_8. Write down the exact expression for the transfer function generated. Does your design meet the specifications? Using MATLAB, compute and plot the filter's unwrapped phase response and the group delay response.

Program P7.3 illustrates the design of elliptic bandpass filter.

```
% Program P7.2
clc;clear all;close all;
fs = 100;
t = (1:100)/fs;
s1 = sin(2*pi*5*t);
```

```
subplot(4,1,1);plot(t,s1);
title('s1 = sin(2*pi*5*t)');
xlabel('Time');ylabel('Amplitude');
s2 = sin(2*pi*15*t);
subplot(4,1,2);plot(t,s2);
title('s2 = sin(2*pi*15*t)');
xlabel('Time');ylabel('Amplitude');
s3 = sin(2*pi*30*t);
subplot(4,1,3);plot(t,s3);
title('s3 = sin(2*pi*30*t)');
xlabel('Time');ylabel('Amplitude');
s = s1+s2+s3;
subplot(4,1,4);plot(t,s);
title('s = s1+s2+s3');
xlabel('time');ylabel('Amplitude');
[b a] = ellip(4,0.1,40,[10 20]*2/fs);
[h w] = freqz(b,a,512);
figure(2);
plot(w*fs/(2*pi),abs(h));
title('Response of filter');
xlabel('Frequency(in Hz)');ylabel('Gain in dB');
sf = filter(b,a,s);
figure(3);
subplot(2,1,1);plot(sf);
title('filtered output');
xlabel('Time');ylabel('Amplitude');
s = fft(s,512);
sf = fft(sf,512);
p = (0:255)/256*(fs/2);
subplot(2,1,2);plot(p,abs([s(1:256)' sf(1:256)']));
title('Frequency spetrum of filtered Signal');
xlabel('Frequency(in Hz)');ylabel('PSD');
```





Questions:

Q7_14. Design the Elliptic bandpass filter by running Program P7.3. What are the filter specifications? Does your design meet the specifications?

Q7_15. Modify Program P7.3 to design a elliptical lowpass filter meeting the specifications given in Question Q7_6.

Q7_16. Modify Program P7.3 to design a elliptical highpass filter meeting the specifications given in Question Q7_7.

Q7_17. Modify Program P7.3 to design an elliptic bandstop filter meeting the specifications given in Question Q7_9.

Q7_18. Replace the ellip command by butter, cheby1 and cheby2 to design the lowpass, highpass, bandpass & bandstop filter and observe the changes in output wavforms.

Q7_19. Observe the spectrum changes in Program P7.3 by increasing & decreasing N value as well as by varying sampling frequency.

8. FIR Filter Design

For the design of FIR lowpass or highpass digital filters, there are several design formulae for estimating the minimum filter length N directly from the digital filter specifications: normalized passband edge angular frequency ωp, normalized stopband edge angular frequency ωs, peak passband ripple δp, and peak stopband ripple δs. A rather simple approximate formula developed by Kaiser [Kai74] is given by

$$N \cong \frac{-20\log_{10}\left(\sqrt{\delta_p \delta_s}\right) - 13)}{\frac{14.6\Delta\omega}{2\pi}}$$

where $\Delta\omega = |\omega p - \omega s|$ is the width of the transition band. The above formula also can be used for designing multitransition-band FIR filters, in which case $\Delta\omega$ is the width of the smallest of all transition bands.

- The most straight-forward method of FIR filter design is based on windowing the ideal infinite-length impulse response $h_D[n]$ obtained by an inverse discrete-time Fourier transform of the ideal frequency response $H_D(e^{j\omega})$ by an appropriate finite-length *window function* $w[n]$. The impulse response coefficients of the final design are then given by $h[n] = h_D[n] \cdot w[n]$.

- The ideal lowpass filter has a zero-phase frequency response

$$H_{LP}\left(e^{j\omega}\right) = \begin{cases} 1, & |\omega| \le \omega_c \\ 0, & \omega_c < |\omega| \le \pi \end{cases}$$

  The corresponding impulse response coefficients $hLP\,[n]$ are given by

$$h_{LP}(n) = \frac{sin\omega_c n}{\pi n}, \qquad -\infty \le n \le \infty.$$

  which is seen to be doubly infinite, not absolutely summable, and therefore unrealizable. By setting all impulse response coefficients outside the range $-M \le n \le M$ equal to zero, we arrive at a finite-length noncausal approximation of length $N = 2M + 1$ that, when shifted to the right, yields the coefficients of a causal FIR lowpass filter:

$$\hat{h}_{LP}(n) = \begin{cases} \frac{sin\omega_c(n-M)}{\pi(n-M)}, & 0 \le n \le N-1 \\ 0, & otherwise \end{cases}$$

  It should be noted that the above expression also holds for N even in which case M is a fraction, i.e. an integer plus a half.

  The impulse response coefficients $h_{HP}[n]$ of the ideal highpass filter are given by

$$h_{HP}(n) = \begin{cases} 1 - \frac{\omega_c}{\pi}, & for\ n = 0, \\ -\frac{\sin(\omega_c n)}{\pi n}, & for\ |n| > 0 \end{cases}$$

The impulse response coefficients h_{BP} [n] of an ideal bandpass filter with cutoffs at ω_{c1} and ω_{c2} are given by

$$h_{BP}(n) = \frac{\sin \omega_{c2} n}{\pi n} - \frac{\sin \omega_{c1} n}{\pi n}, \qquad -\infty \le n \le \infty.$$

and those of an ideal bandstop filter with cutoffs at ω_{c1} and ω_{c2} are given by

$$h_{BS}(n) = \begin{cases} 1 - \dfrac{\omega_{c2} - \omega_{c1}}{\pi}, & for\ n = 0, \\ \dfrac{\sin(\omega_{c1} n)}{\pi n} - \dfrac{\sin(\omega_{c2} n)}{\pi n}, & for\ |n| > 0. \end{cases}$$

- Like the ideal lowpass filter, all of the above ideal filters are unrealizable. They can be made realizable by truncating the impulse response sequences to finite lengths and shifting the truncated coefficients to the right appropriately.

- The causal FIR filters obtained by simply truncating the impulse response coefficients of the ideal filters given in the previous section exhibit an oscillatory behavior in their respective magnitude responses that is more commonly referred to as the *Gibbs phenomenon*. The Gibbs phenomenon can be reduced either by using a window that tapers smoothly to zero at each end or by providing a smooth transition from the passband to the stopband. Use of a tapered window causes the height of the side lobes to diminish with a corresponding increase in the main lobe width, resulting in a wider transition at the discontinuity. In all window-based lowpass filter designs, the cutoff frequency ωc is half of the sum of the passband and stopband edge frequencies.

- Some commonly used tapered windows of length $2M$ +1 with fixed ripples are:

Hanning: $w(n) = \frac{1}{2}\left[1 + \cos\left(\frac{2\pi n}{2M+1}\right)\right], \quad -M \le n \le M,$

Hamming: $w(n) = 0.54 + 0.46cos\left(\frac{2\pi n}{2M+1}\right), \quad -M \le n \le M,$

Blackman: $w(n) = 0.42 + 0.5cos\left(\frac{2\pi n}{2M+1}\right) + 0.08cos\left(\frac{4\pi n}{2M+1}\right) \quad -M \le n \le M,$

- The most widely used adjustable window is the *Kaiser window* given by:

$$w(n) = \frac{I_0\left(\beta\sqrt{1 - \left(\frac{n}{M}\right)^2}\right)}{I_0(\beta)} \qquad -M \le n \le M$$

where β is an adjustable parameter and $I_0(u)$ is the modified zeroth-order Bessel function, which can be expressed in a power series form:

$$I_0(u) = 1 + \sum_{r=1}^{\infty}\left[\frac{\left(\frac{u}{2}\right)^r}{r!}\right]^2$$

It can be seen that $I_0(u)$ is positive for all real values of u. In practice, it is sufficient to keep only the first 20 terms in the summation of above Eq. to arrive at a reasonably

accurate value of $I_0(u)$. The parameter β controls the minimum stopband attenuation $\alpha_s = -20\ log10\delta_s$ of the windowed filter response. Formulae for estimating β and the filter length $N = 2M+1$, for specified αs and transition bandwidth $\Delta f = Fp - Fs$, are given by

$$\beta = \begin{cases} 0.1102(\alpha_s - 8.7), & for\ \alpha_s > 50, \\ 0.5842(\alpha_s - 21)^{0.4} + 0.07886(\alpha_s - 21), & for\ 21 \le \alpha_s \le 50, \\ 0, & for\ \alpha_s < 21. \end{cases}$$

and

$$N = \begin{cases} \dfrac{\alpha_s - 7.95}{14.36\Delta f} + 1, & for\ \alpha_s > 21, \\ \dfrac{0.9222}{\Delta f} + 1, & for\ \alpha_s \le 21 \end{cases}$$

The Kaiser window provides no independent control over the passband ripple δp. However, in practice, δp is approximately equal to δs.

Conceptually the simplest approach to FIR filter design is to simply truncate to a finite number of terms the doubly infinite-length impulse response coefficients obtained by computing the inverse discrete-time Fourier transform of the desired ideal frequency response. However, a simple truncation results in an oscillatory behavior in the respective magnitude response of the FIR filter, which is more commonly referred to as the *Gibbs phenomenon*.

The Gibbs phenomenon can be reduced by windowing the doubly infinite-length impulse response coefficients by an appropriate finite-length window function. The functions fir1 and fir2 can be employed to design windowed FIR digital filters in MATLAB. Both functions yield a linear-phase design.

The function fir1 can be used to design conventional lowpass, highpass, bandpass, and bandstop linear-phase FIR filters. The command

b = fir1(N,Wn)

returns in vector b the impulse response coefficients, arranged in ascending powers of z^{-1}, of a lowpass/bandpass filter of order N for an assumed sampling frequency of 2 Hz. For lowpass design, the normalized cutoff frequency is specified by a scalar W_n, a number between 0 and 1. For bandpass design, Wn is a two-element vector $[W_{n1}, W_{n2}]$ containing the specified passband edges where $0 < W_{n1} < W_{n2} < 1$. The command

b = fir1(N,Wn,'high')

with N an even integer, is used for designing a highpass filter. The command

b = fir1(N,Wn,'stop')

with W_n a two-element vector, is employed for designing a bandstop FIR filter. If none is specified, the *Hamming window* is employed as a default. The command

b = fir1(N, Wn, taper)

makes use of the specified windowcoefficients of length N+1 in the vector taper. However, the window coefficients must be generated a priori using an appropriate MATLAB function such as blackman, hamming, hanning, chebwin, or kaiser. The commands to use are of the following forms:

taper = blackman(N) taper = hamming(N) taper = hanning(N)
taper = chebwin(N) taper = kaiser(N, beta)

The order N of the FIR filter to meet the given specifications can be estimated using Kaiser's formula. The MATLAB function kaiord given below implements Kaiser's formula:

Program P8.1 can be used to find the length of an FIR filter.

```
% Program P8.1
function N = kaiord(Fp, Fs, dp, ds, FT)
% Computation of the length of a linear-phase
% FIR multiband filter using Kaiser's formula
% dp is the passband ripple
% ds is the stopband ripple
% Fp is the passband edge in Hz
% Fs is the stopband edge in Hz
% FT is the sampling frequency in Hz.
% If none specified default value is 2
% N is the estimated FIR filter order
if nargin == 4,
FT = 2;
end
if length(Fp) > 1,
TBW = min(abs(Fp(1) - Fs(1)), abs(Fp(2) - Fs(2)));
else
TBW = abs(Fp - Fs);
end
num = -20*log10(sqrt(dp*ds)) - 13;
den = 14.6*TBW/FT;
N = ceil(num/den);
```

The function kaiserord in the Signal Processing Toolbox can also be used for estimating the filter order using Kaiser's formula. It can be used in one of the following forms:

[N, Wn, beta, ftype] = kaiserord(fedge, aval, dev)
[N, Wn, beta, ftype] = kaiserord(fedge, aval, dev, FT)
c = kaiserord(fedge, aval, dev, FT, 'cell')

where FT is the sampling frequency in Hz whose default value is 2 Hz if not specified; fedge is a vector of bandedge frequencies in Hz, in increasing order between 0 and FT/2; and aval is a vector specifying the desired values of the magnitude response at the specified bandedges given by fedge. The length of fedge is 2 less than twice the length of aval and therefore must be even. dev is a vector of maximum deviations or ripples in dB allowable for each band. If the deviations specified are unequal, the smallest one is used for all bands. The output data are in the desired format for use in fir1, with normalized bandedges Wn and the parameter beta used for computing the window coefficients as given in the equation mentioned above.The string ftype specifies the filter type for fir1. It is high for highpass filter design, and stop for bandstop filter design. The last form of kaiserord specifies a cell array whose elements are parameters to fir1.

Questions:

Q8.1. Using the function kaiord, estimate the order of a linear-phase lowpass FIR filter with the following specifications: passband edge = 2 kHz, stopband edge = 2.5 kHz, passband ripple δp = 0.005, stopband ripple δs = 0.005, and sampling rate of 10 kHz. What are the purposes of the commands ceil and nargin in the function kaiord?

Q8.2. Repeat Question Q8.1. for the following cases: (a) sampling rate of 20 kHz, (b) δp = 0.002 and δs = 0.002, and (c) stopband edge = 2.3 kHz. Compare the filter length obtained in each case with that obtained in Question Q8.1. Comment on the effect of the sampling rate, ripples, and the transition bandwidth on the filter order.

Q8.3. Repeat Question Q8.1. using the function kaiserord. Compare the value of the filter order obtained with that obtained in Question Q8.1.

Q8.4. Repeat Question Q8.1. using the function firpmord. Compare the value of the filter order obtained with those obtained in Questions Q8.1. and Q8.3.

Q8.5. Using the function kaiord, estimate the order of a linear-phase bandpass FIR filter with the following specifications: passband edges = 1.8 and 3.6 kHz, stopband edges 1.2 and 4.2 kHz, passband ripple δp = 0.1, stopband ripple δs = 0.02, and sampling rate of 12 kHz.

Q8.6. Repeat Question Q8.5. using the function kaiserord. Compare the value of the filter order obtained with that obtained in Question Q8.5.

Q8.7. Repeat Question Q8.5. using the function firpmord.Compare the value of the filter order obtained with that obtained in Questions Q8.5 and Q8.6.

Gibb's Phenomenon

The occurrence of Gibb's phenomenon can be illustrated by considering the design of an FIR filter obtained by truncating the impulse response of the ideal filters and then computing their frequency responses. The truncated impulse response coefficients of a lowpass filter can be generated in MATLAB using the function sinc, which can also be used with simple modifications to generate the truncated impulse response coefficients of a highpass, bandpass, or bandstop filter.

Questions:

Q8.8. Using the function sinc write a MATLAB program to generate the impulse response coefficients of four zero-phase lowpass filters with cutoffs at $\omega c = 0.4\pi$ and of lengths 81, 61, 41, and 21, respectively, and then compute and plot their magnitude responses. Use the colon ":" operator to extract the impulse response coefficients of the shorter length filters from that of the length-81 filter. Examine the oscillatory behavior of the frequency responses of each filter on both sides of the cutoff frequency. What is the relation between the number of ripples and the length of the filter? What is the relation between the heights of the largest ripples and the length of the filter? How would you modify the above program to generate the impulse response coefficients of a zero-phase lowpass filter of even lengths?

Q8.9. Using the function sinc write a MATLAB program to generate the impulse response coefficients of a zero-phase length-45 highpass filter with a cutoff at $\omega c = 0.4\pi$ and then compute and plot its magnitude response. Examine the oscillatory behavior of the frequency responses of each filter on both sides of the cutoff frequency. How would you modify the above program to generate the impulse response coefficients of a zero-phase highpass filter of even length?

Q8.10. Write a MATLAB program to generate the impulse response coefficients of four zero-phase differentiators of lengths 81, 61, 41, and 21, respectively, and then compute and plot their magnitude responses. The following code fragments show how to generate a differentiator of length 2M+1.

```
n = 1:M;
b = cos(pi*n)./n;
num = [-fliplr(b) 0 b];
```

Examine the oscillatory behavior of the frequency response of the differentiator for each case. What is the relation between the number of ripples and the length of the differentiator? What is the relation between the heights of the largest ripples and the length of the filter?

Q8.11. Write a MATLAB program to generate the impulse response coefficients of four discrete-time Hilbert transformers of lengths 81, 61, 41, and 21, respectively, and then compute and plot their magnitude responses. The following code fragments show how to generate a Hilbert transformer of length 2M+1.

```
n = 1:M;
c = sin(pi*n/2);
b = 2*(c.*c)./(pi*n);
num = [-fliplr(b) 0 b];
```

Examine the oscillatory behavior of the frequency responses of the Hilbert transformer for each case. What is the relation between the number of ripples and the length of the Hilbert transformer? What is the relation between the heights of the largest ripples and the length of the filter?

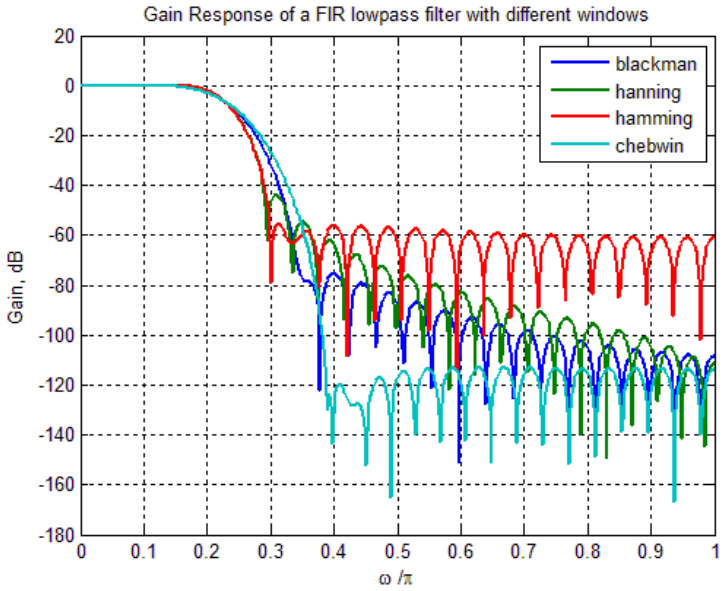
Program P8.2 can be used to find the gain response of FIR lowpass filter with different windows.

```
% Program P8.2
clc;clear all; close all;
% Design of a FIR lowpass filter using hamming window
Fp=2000; Fs=2500; dp=0.005, ds=0.005, FT=10000;
% Estimate the Filter Order
N = kaiord(Fp, Fs, dp, ds, FT)
% Design the Filter
[num,den] = fir1(N,2250/10000,blackman(N+1));
[num1,den1] = fir1(N,2250/10000,hanning(N+1));
[num2,den2] = fir1(N,2250/10000,hamming(N+1));
[num3,den3] = fir1(N,2250/10000,chebwin(N+1));
% Display the transfer function
% disp('Numerator coefficients are ');disp(num);
% disp('Denominator coefficients are ');disp(den);
% Compute the gain response
w = 0:pi/511:pi;
h = freqz(num,den,w);
g = 20*log10(abs(h));

h1 = freqz(num1,den1,w);
g1 = 20*log10(abs(h1));
h2 = freqz(num2,den2,w);
```

```
g2 = 20*log10(abs(h2));
h3 = freqz(num3,den3,w);
g3 = 20*log10(abs(h3));

% Plot the gain response
plot(w/pi,g,w/pi,g1,w/pi,g2,w/pi,g3,'linewidth',2);grid
legend('blackman','hanning','hamming','chebwin',1);
xlabel('\omega /\pi'); ylabel('Gain, dB');
title('Gain Response of a FIR lowpass filter with different
windows');
% figure, freqz(num,den,w)
```



Questions:

Q8.12. Using the function fir1, design a linear-phase FIR lowpass filter meeting the specifications given in Question Q8.1 and plot its gain and phase responses. Use the order estimated using Kaiser's formula in Question Q8.1. Show the filter coefficients in a tabular form. Does your design meet the specifications? If it does not, adjust the filter order until the design meets the specifications. What is the order of the filter meeting the specifications?

Q8.13. Repeat Question Q8.12 using each of the following windows: Hanning, Blackman, and Dolph–Chebyshev windows.

Q8.14. Design an FIR lowpass filter using a Kaiser window. The filter specifications are: ωp = 0.31, ωs 0.41, and As = 50 dB. Note that the function kaiser requires the values of the parameter β and the order N which must first be calculated using their corresponding equations discussed above. Does your design meet the specifications?

Q8.15. Repeat Question Q8.14 using the functions kaiserord and fir1.

Q8.16. Using fir2 design an FIR filter of order 95 with three different constant magnitude levels: 0.4 in the frequency range 0 to 0.25, 1.0 in the frequency range 0.3 to 0.45, and 0.8 in the frequency range 0.5 to 1.0. Plot the magnitude response of the filter designed. Does your design meet the specifications?

9. Learning SPTool & FDATool

The Filter Design and Analysis Tool (FDATool) is a powerful graphical user interface (GUI) in the Signal Processing Toolbox for designing and analyzing filters.

FDATool enables you to quickly design digital FIR or IIR filters by setting filter performance specifications, by importing filters from your MATLAB workspace or by adding, moving or deleting poles and zeros. FDATool also provides tools for analyzing filters, such as magnitude and phase response plots and pole-zero plots.

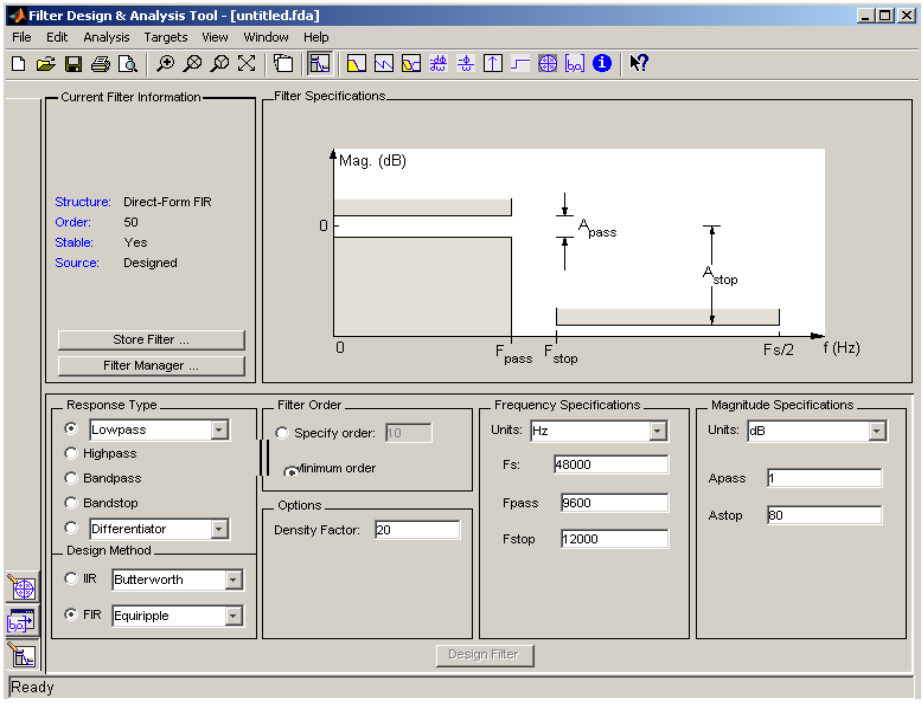
You can use FDATool as a convenient alternative to the command line filter design functions.

Getting Started

Type fdatool at the MATLAB command prompt:

```
>>fdatool
```

A Tip of the Day dialog displays with suggestions for using FDATool. Then, the GUI diplays with a default filter.



The GUI has three main regions:

The Current Filter Information region, the Filter Display region and the Design panel. The upper half of the GUI displays information on filter specifications and responses for the current filter. The Current Filter Information region, in the upper left, displays filter properties, namely the filter structure, order, number of sections used and whether the filter is stable or not. It also provides access to the Filter manager for working with multiple filters.

The Filter Display region, in the upper right, displays various filter responses, such as, magnitude response, group delay and filter coefficients.

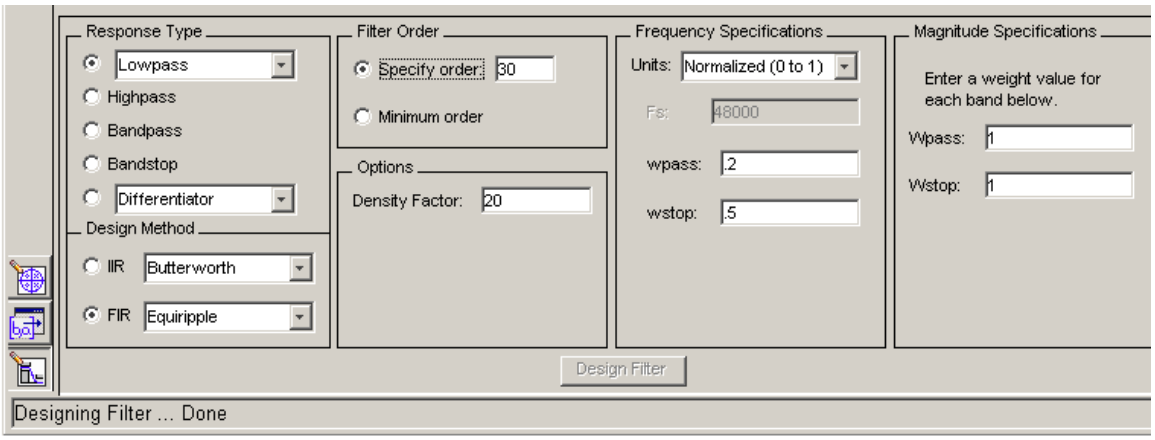
The lower half of the GUI is the interactive portion of FDATool. The Design Panel, in the lower half is where you define your filter specifications. It controls what is displayed in the other two upper regions. Other panels can be displayed in the lower half by using the sidebar buttons.

Designing a filter

We will design a low pass filter that passes all frequencies less than or equal to 20% of the Nyquist frequency (half the sampling frequency) and attenuates frequencies greater than or equal to 50% of the Nyquist frequency. We will use an FIR Equiripple filter with these specifications:

Passband attenuation 1 dB
Stopband attenuation 80 dB
A passband frequency 0.2 [Normalized (0 to 1)]
A stopband frequency 0.5 [Normalized (0 to 1)]

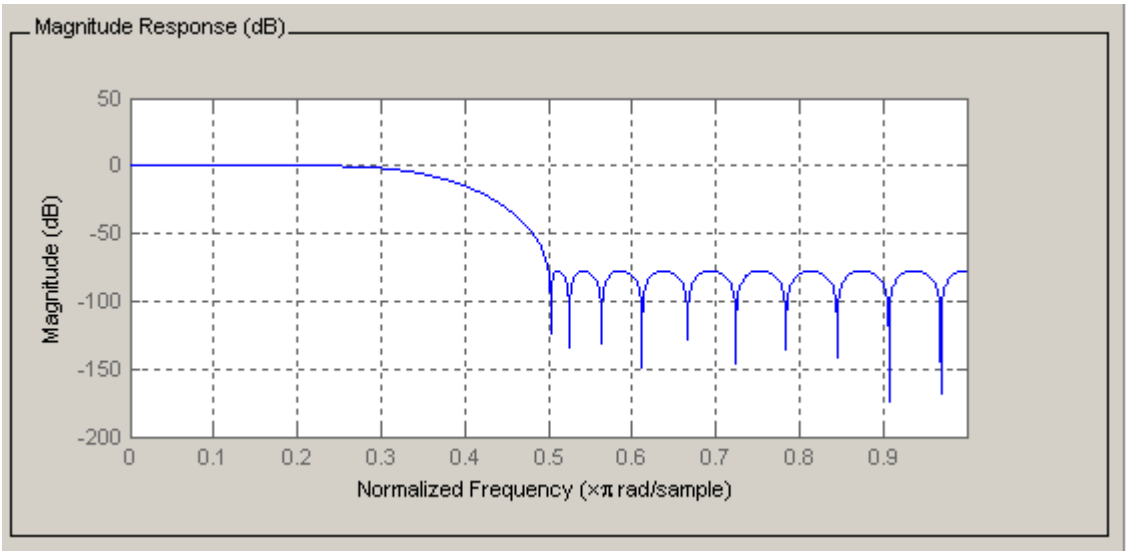
To implement this design, we will use the following specifications:



1) Select Lowpass from the dropdown menu under Response Type and Equiripple under FIR Design Method. In general, when you change the Response Type or Design Method, the filter parameters and Filter Display region update automatically.
2) Select Specify order in the Filter Order area and enter 30.
3) The FIR Equiripple 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 equiripple filter, but more time is required as the computation increases. Leave this value at 20.
4) Select Normalized (0 to 1) in the Units pull down menu in the Frequency Specifications area.
5) Enter 0.2 for wpass and 0.5 for wstop in the Frequency Specifications area.
6) Wpass and Wstop, in the Magnitude Specifications area are positive weights, one per band, used during optimization in the FIR Equiripple filter. Leave these values at 1.

7) 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.



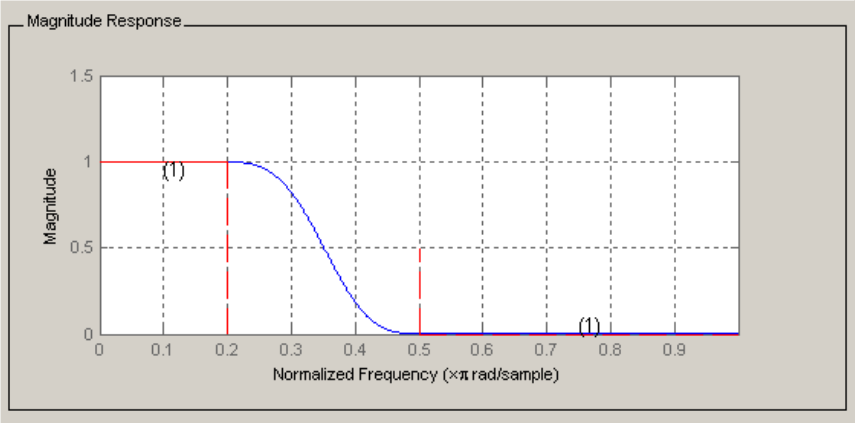
Viewing other Analyses

Once you have designed the filter, you can view the following filter analyses in the display window by clicking any of the buttons on the toolbar:

In order from left to right, the buttons are
Magnitude response Phase response Magnitude and Phase responses Group delay response Phase delay response Impulse response Step response Pole-zero plot Filter Coefficients Filter Information

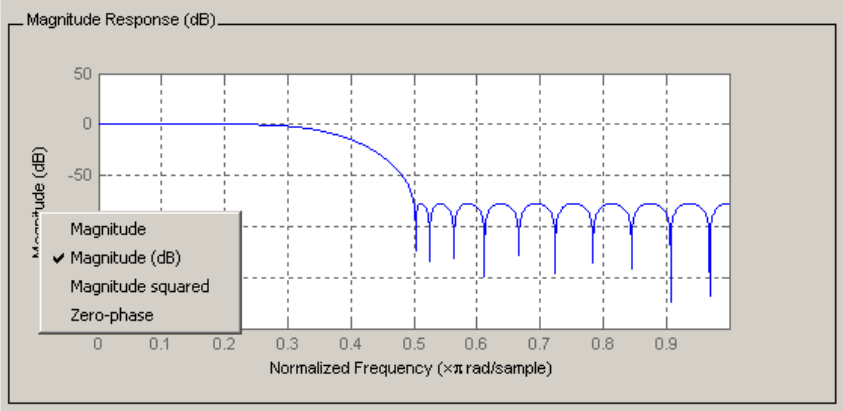
Comparing the design to filter specifications

FDATool allows you to measure how closely your design meets the filter specifications by using Specification masks which overlay the filter specifications on the response plot. In the Display Region, when the Magnitude plot is displayed, select Specification Mask from the View menu to overlay the filter specifications on the response plot.
The magnitude response of the filter with Specification mask is shown below:

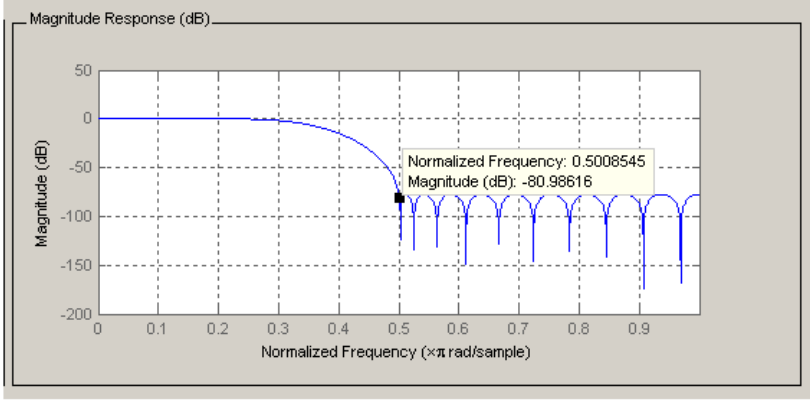


Changing Axis Units

You can change the x- or y-axis units by right-clicking the mouse on an axis label and selecting the desired units. The current units have a checkmark.



Marking data points



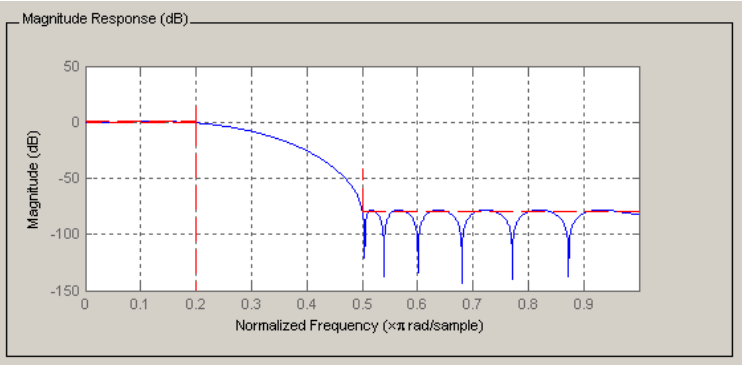
In the Display region, you can click on any point in the plot to add a data marker, which displays the values at that point. Right-clicking on the data marker displays a menu where you can move, delete or adjust the appearance of the data markers.

Optimizing the design

To minimize the cost of implementation of the filter, we will try to reduce the number of coefficients by using Minimum Order option in the design panel.

Change the selection in Filter Order to Minimum Order in the Design Region and leave the other parameters as they are.

Click the Design Filter button to design the new filter.

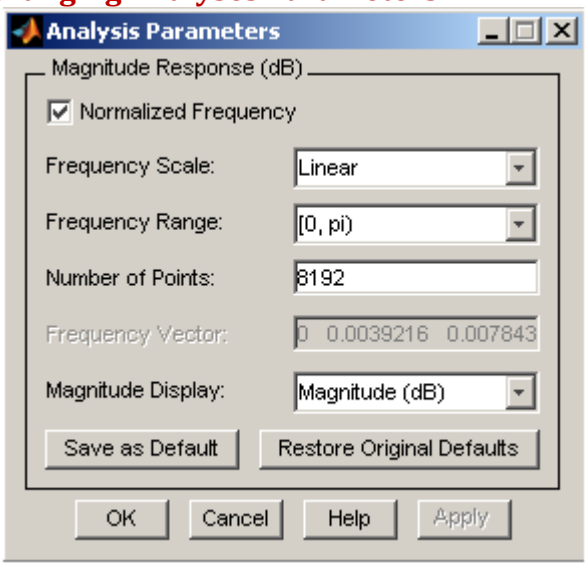


As you can see in the Current Filter Information area, the filter order decreased from 30 to 16, the number of ripples decreased and the transition width became wider. The passband and the stopband specifications still meet the design criteria.

Using a different filter structure

Our filter is a Direct-form FIR. Typically, the Direct-Form FIR transposed structure is implemented in hardware. You can use Convert Structure dialog from the Edit menu to change the current filter to a new structure. Filters can be converted to the following representations:
State-Space Direct-Form FIR, Direct-Form FIR Transposed, Direct-Form Symmetric FIR

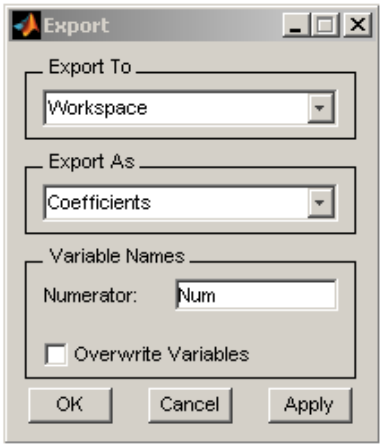
Changing Analyses Parameters



By right-clicking on the plot and selecting Analysis Parameters, you can display a dialog box for changing analysis-specific parameters. (You can also select Analysis Parameters from the Analysis menu.)

To save the displayed parameters as the default values, click Save as Default. To restore the MATLAB-defined default values, click Restore Original Defaults.
Exporting the filter

Once you are satisfied with your design, you can export your filter to the following destinations: MATLAB workspace MAT-file Text-file.Select Export from the File menu.



If exporting to the MATLAB workspace, you can export as coefficients or as an object by selecting from the Export from the pulldown menu.
If you want to export as an object, the object's properties control many aspects of its apearance and behaviour. You can use GET and SET commands from the MATLAB command prompt to have access and manipulate the property values of the object.

ECG SIGNAL SMOOTHENING WITH FIR FILTER

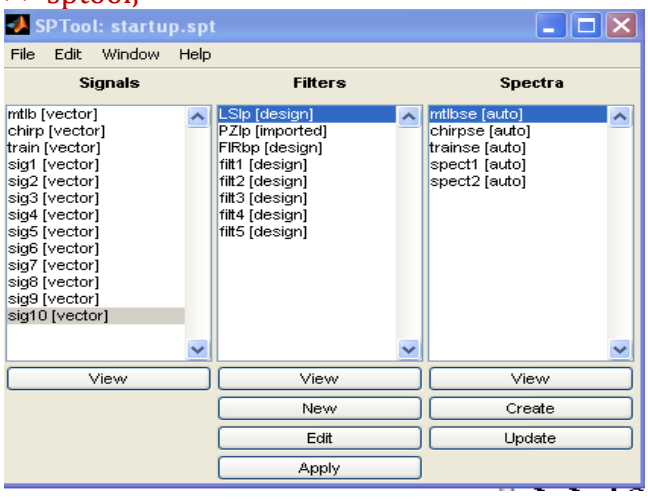
```
>>load ecg2x60.dat –ascii
>>c=ecg2x60
>>csvwrite('xxx.dat',c)
```

Filter design using FDA tool

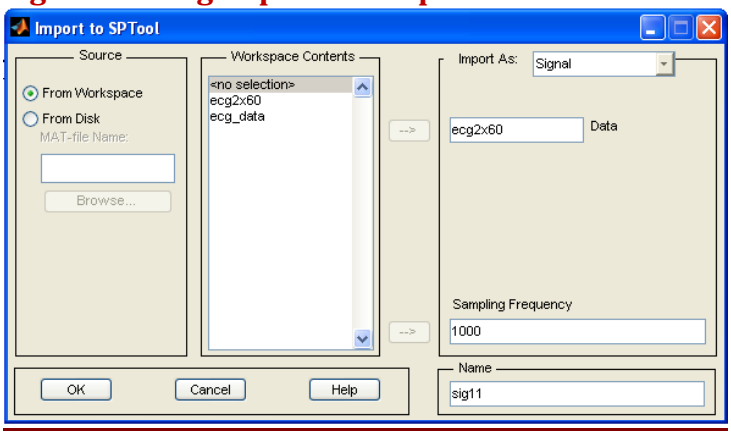
Filter : FIR
Response type : Low Pass
Filter Method : Window - Hamming
Filter order : 17
Sampling frequency : 500Hz
Cut off frequency : 30Hz

Filter Design using Sptool:

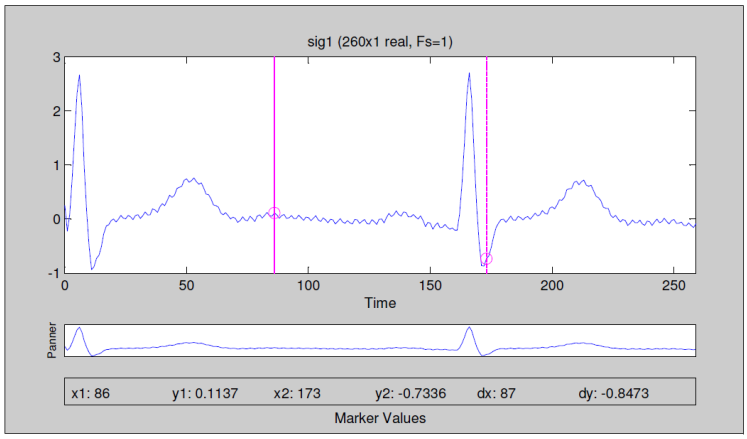
```
>> sptool;
```



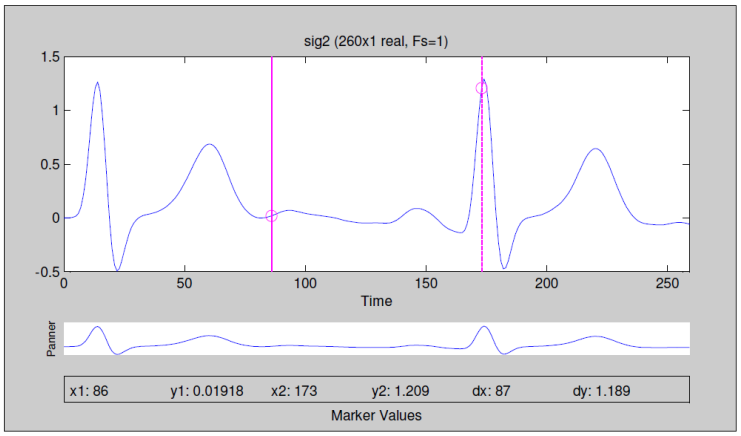
Signal is being imported to sptool



Unfiltered Ecg Signal:



Select the appropriate filter in the filter menu to smoothen out the given ecg signal.



APPLYING DESIGNED FILTER FOR SPEECH AND MUSIC SIGNAL

In the previous examples we test the functionality of the filters by giving inputs generated by program. In this section let us see how to read recorded speech or music signal which is stored as .wav files in PC as matrix variable in matlab , apply filter to them and hear it back to percept what actually happened due to filtering.

Explore a available .wav file rich in music signal and make it available in work folder of the matlab. (Then you can access the file with out path name).

Read the information in the file as a matrix.
[x,fs,bits]=wavread('filename');
Variable x will contain the the samples read from wav file .variable x consists of 1 column if the information is mono or 2 columns if the information is stereo one for right ear and the other for left. The number of rows of x will be equal to no of samples in the wav file.

fs will have the sampling frequency read from the file.
bits will have value of number of bits used to represent the sample in that specific file.

To play back the sound we can use

```
Wavplay(x,fs)
Soundsc(x)
```

x should contain rows or columns of samples.
Fs—sampling frequency as required.
By this function matlab will normalize the variable x such that they have values -1-to +1 and send them to the sound card available in the pc and configure the sound card to play it with sampling frequency of fs.

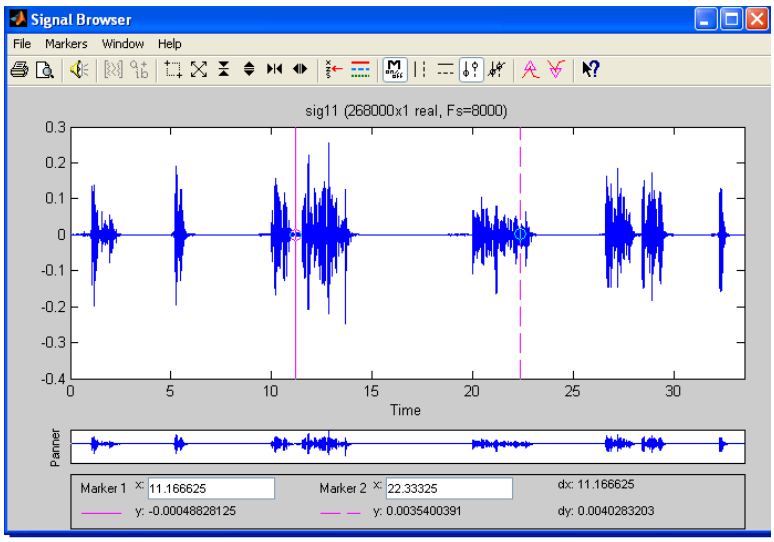
```
[x,fs,bits]=wavread('filename');
Wavplay(x,fs)
```

The above code plays back whatever information in the file.
Now design a different types filter with reference to previous exercises and apply the filter to variable x and play the result. LOOK HOW IT SOUND.

Example:

```
[x,fs,bits]=wavread('filename');
fc=3500;                    %cutoff at 3.5khz%
w=fc/fs;                    %digital frequency%
[b,a]=butter(8,w,'low');    %filter design%
y=filter(b,a,x);            %filter the input%
wavplay(x,fs);              %hear original sound%
wavplay(y,fs);              %hear the filtered sound%
```

```
>>sptool
>> load farspeech
```

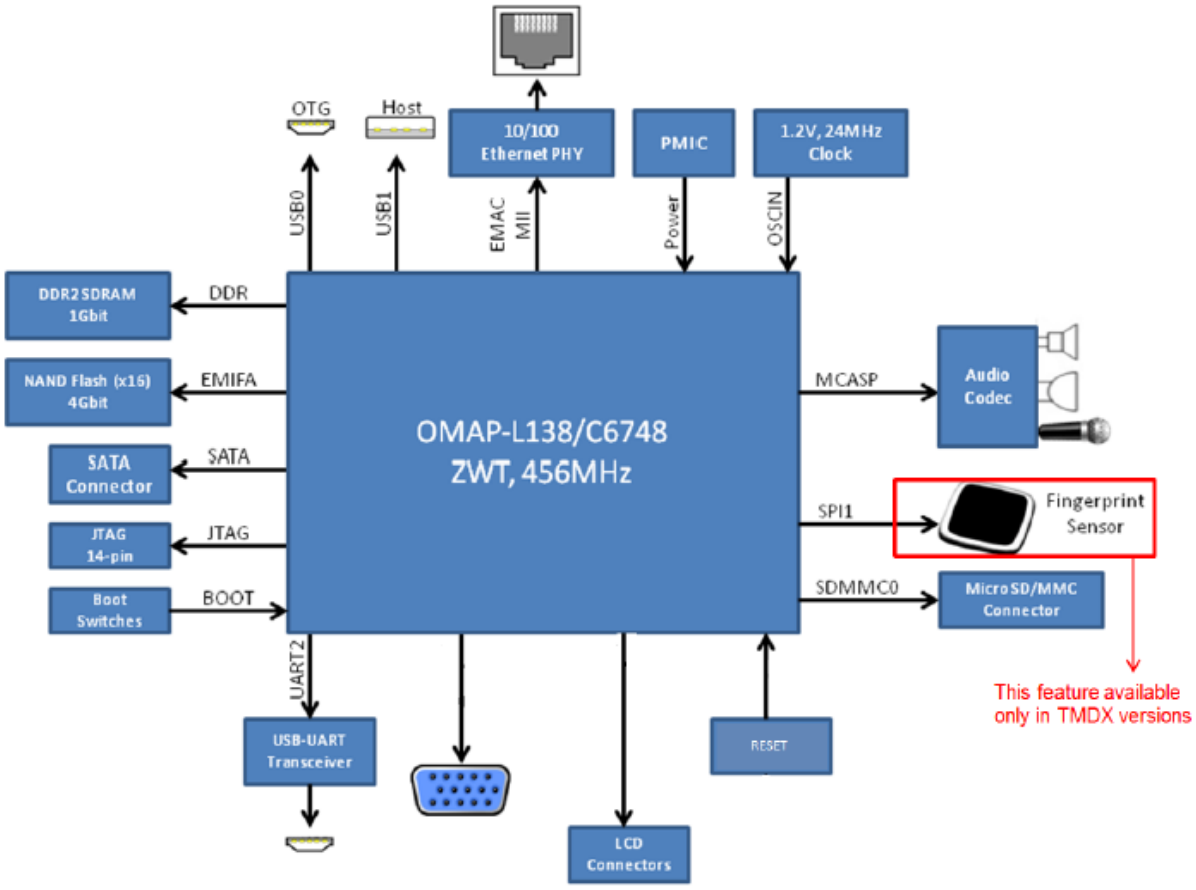


Play with above program with different types of filters like low pass , high pass, band pass, band stop and with different cutoff frequencies.

HARDWARE DESCRIPTION
OMAPL138/TMS320C6748 DSK

Description:

The TMS320C6748 DSP development kit (LCDK) is a scalable platform that breaks down development barriers for applications that require embedded analytics and real-time signal processing, including biometric analytics, communications and audio. The low-cost LCDK will also speed and ease your hardware development of real-time DSP applications. This new board reduces design work with freely downloadable and duplicable board schematics and design files. A wide variety of standard interfaces for connectivity and storage enable you to easily bring audio, video and other signals onto the board. The LCDK does not have an onboard emulator. An external emulator from TI (such as the XDS100, XDS200,XDS510, XDS560) or a third-party will be required to start development.



The OMAP-L138 C6000 DSP+ARM processor is a low-power applications processor based on an ARM926EJ-S and a C674x DSP core. This processor provides significantly lower power than other members of the TMS320C6000™ platform of DSPs. The dual-core architecture of the device provides benefits of both DSP and reduced instruction set computer (RISC) technologies, incorporating a high-performance TMS320C674x DSP core and an ARM926EJ-S core. The ARM926EJ-S is a 32-bit RISC processor core that performs 32-bit or 16-bit instructions and processes 32-bit, 16-bit, or 8-bit data. The core uses pipelining so that all

parts of the processor and memory system can operate continuously. The device DSP core uses a 2-level cache-based architecture. The level 1 program cache (L1P) is a 32- KB direct mapped cache, and the level 1 data cache (L1D) is a 32-KB 2-way, set-associative cache. The level 2 program cache (L2P) consists of a 256-KB memory space that is shared between program and data space. L2 memory can be configured as mapped memory, cache, or combinations of the two. Although the DSP L2 is accessible by the ARM9 and other hosts in the system, an additional 128KB of RAM shared memory is available for use by other hosts without affecting DSP performance.

Processor

- TI TMS320C6748 DSP Application Processor
- 456-MHz C674x Fixed/Floating Point DSP
- 456-MHz ARM926EJ RISC CPU (OMAP-L138 only)
- On-Chip RTC

Memory

- 128 MByte DDR2 SDRAM running at 150MHz
- 128 MByte 16-bit wide NAND FLASH
- 1 Micro SD/MMC Slot

Interfaces

- One mini-USB Serial Port (on-board serial to USB)
- One Fast Ethernet Port (10/100 Mbps) with status LEDs
- One USB Host port (USB 1.1)
- One SATA Port (3Gbps)
- One LCD Port (Beagleboard XM connectors)
- One Leopard Imaging Camera Sensor Input (36-pin ZIP connector)
- Three AUDIO Ports (1 LINE IN-J55 & 1 LINE OUT-J56 & 1 MIC IN-J57)
- 14-pin JTAG header (No onboard emulator; external emulator is required)

TMS320C6748 DSP Features

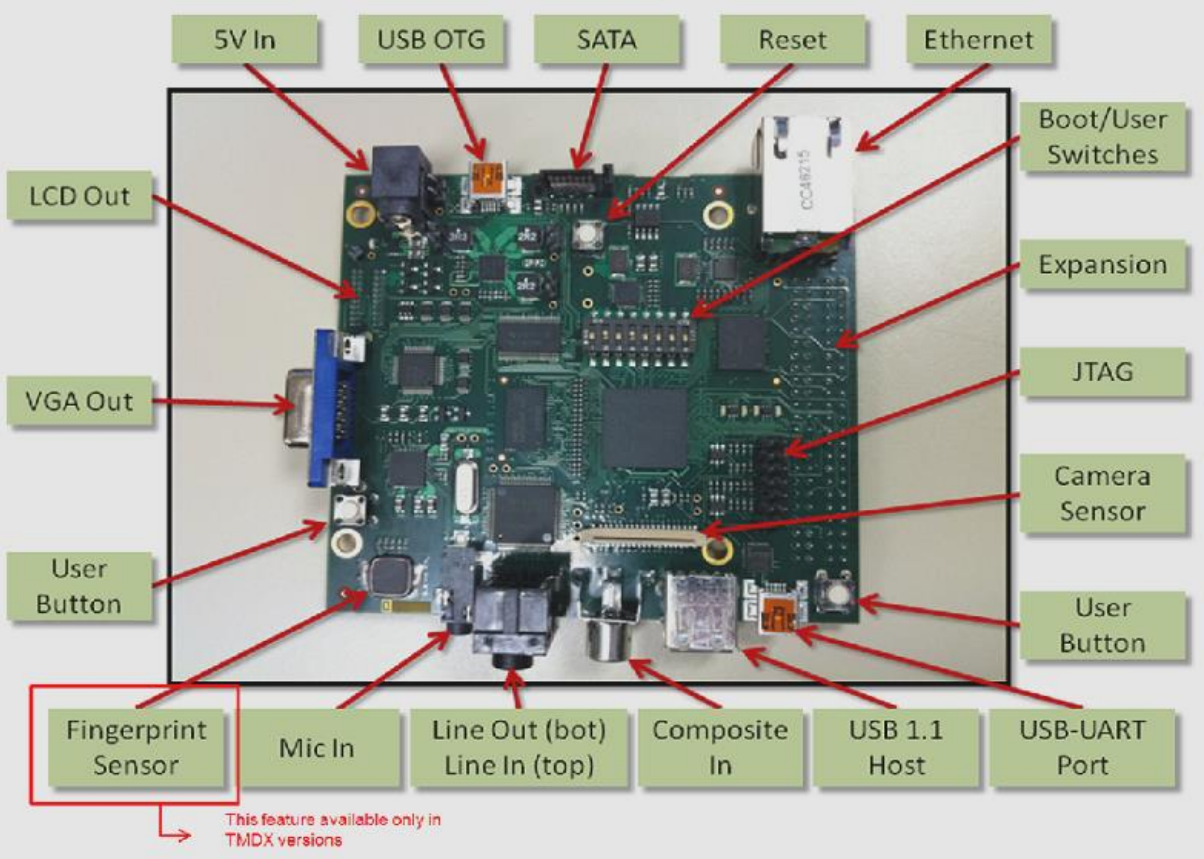
- Highest-Performance Floating-Point Digital Signal Processor (DSP):
- Eight 32-Bit Instructions/Cycle
- 32/64-Bit Data Word
- 375/456-MHz C674x Fixed/Floating-Point
- Up to 3648/2746 C674x MIPS/MFLOPS
- Rich Peripheral Set, Optimized for Audio
- Highly Optimized C/C++ Compiler
- xtended Temperature Devices Available

- Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core
- Eight Independent Functional Units:
- Two ALUs (Fixed-Point)
- Four ALUs (Floating- and Fixed-Point)
- Two Multipliers (Floating- and Fixed-Point)
- Load-Store Architecture With 64 32-Bit General-Purpose Registers
- Instruction Packing Reduces Code Size
- All Instructions Conditional

Instruction Set Features

- Native Instructions for IEEE 754
    - Single- and Double-Precision
- Byte-Addressable (8-, 16-, 32-Bit Data)
    - 8-Bit Overflow Protection
    - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- 67x cache memory.
    - 32K-Byte L1P Program Cache (Direct-Mapped)
    - 32K-Byte L1D Data Cache (2-Way)
    - 256K-Byte L2 unified Memory RAM\Cache.
    - Real-Time Clock With 32 KHz Oscillator and Separate Power Rail.
- Three 64-Bit General-Purpose Timers
    - Integrated Digital Audio Interface Transmitter (DIT) Supports:
        - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
        - Up to 16 transmit pins
        - Enhanced Channel Status/User Data
    - Extensive Error Checking and Recovery
- Two Inter-Integrated Circuit Bus (I2C Bus™) .
- 3 64-Bit General-Purpose Timers (each configurable as 2 32-bit timers)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG) Boundary-Scan-Compatible
- 3.3-V I/Os, 1.2-V Internal (GDP & PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)
- Two Serial Peripheral Interfaces (SPI) Each With Multiple Chip-Selects
- Two Multimedia Card (MMC)/Secure Digital Card Interface with Secure Data I/O (SDIO) Interfaces
- One Multichannel Audio Serial Port.
- Two Multichannel Buffered Serial Ports
- Instruction Set Features
    - Native Instructions for IEEE 754
        - Single- and Double-Precision
    - Byte-Addressable (8-, 16-, 32-Bit Data)

  - 8-Bit Overflow Protection
  - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- 67x cache memory.
  - 32K-Byte L1P Program Cache (Direct-Mapped)
  - 32K-Byte L1D Data Cache (2-Way)
- 256K-Byte L2 unified Memory RAM\Cache.
- Real-Time Clock With 32 KHz Oscillator and Separate Power Rail.
- Three 64-Bit General-Purpose Timers
  - Integrated Digital Audio Interface Transmitter (DIT) Supports:
    - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
    - Up to 16 transmit pins
    - Enhanced Channel Status/User Data
  - Extensive Error Checking and Recovery
- Two Inter-Integrated Circuit Bus (I2C Bus™) .
- 3 64-Bit General-Purpose Timers (each configurable as 2 32-bit timers)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG) Boundary-Scan-Compatible
- 3.3-V I/Os, 1.2-V Internal (GDP & PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)
- Two Serial Peripheral Interfaces (SPI) Each With Multiple Chip-Selects
- Two Multimedia Card (MMC)/Secure Digital Card Interface with Secure Data I/O (SDIO) Interfaces
- One Multichannel Audio Serial Port.
- Two Multichannel Buffered Serial Ports


5V In
USB OTG
SATA
Reset
Ethernet
Boot/User Switches
LCD Out
Expansion
JTAG
VGA Out
Camera Sensor
User Button
User Button
Fingerprint Sensor
Mic In
Line Out (bot) Line In (top)
Composite In
USB 1.1 Host
USB-UART Port
This feature available only in TMDX versions

10. GENERAL PROCEDURE FOR WORKING WITH THE REAL TIME PROJECTS USING C6748DSK:

AUDIO PLAYBACK

1.0 Unit Objective:

To configure the codec AIC3106 for a talk through program using the board support library.

2.0 Prerequisites

C6748LCDK, PC with Code Composer Studio, CRO, Audio Source, Speakers, headphone and Signal Generator.

3.0 Procedure

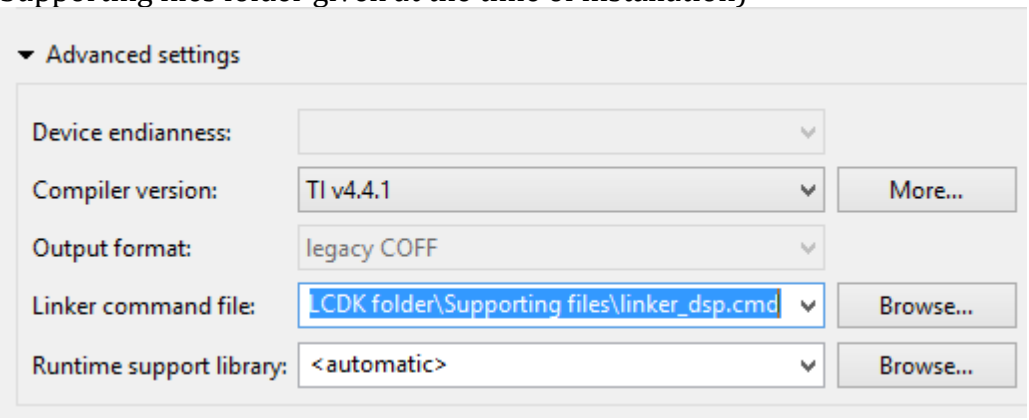
- All the Real time implementations covered in the Implementations module follow code Configuration using board support library.
- The board Support Library (CSL) is a collection of functions, macros, and symbols used to configure and control on-chip peripherals.
- The goal is peripheral ease of use, shortened development time, portability, hardware abstraction, and some level of standardization and compatibility among TI devices.

- Connect one end of a stereo cable to the LINE IN(J55) of the kit and other end to PC (or function generator).
- Connect the headphone (or CRO) to the LINE OUT(J56) of the kit.
- Connect the power supply to the board.
- Connect the External JTAG XDS100v2 with board and PC.

General Steps For Real Time Programs:

R1. Create project.

Goto File ->new->CCS project and enter the details as done in NR3 but only change is in "Advanced Settings" choose the Linker command file: as "**linker_dsp.cmd**" (Browse from Supporting files folder given at the time of installation)



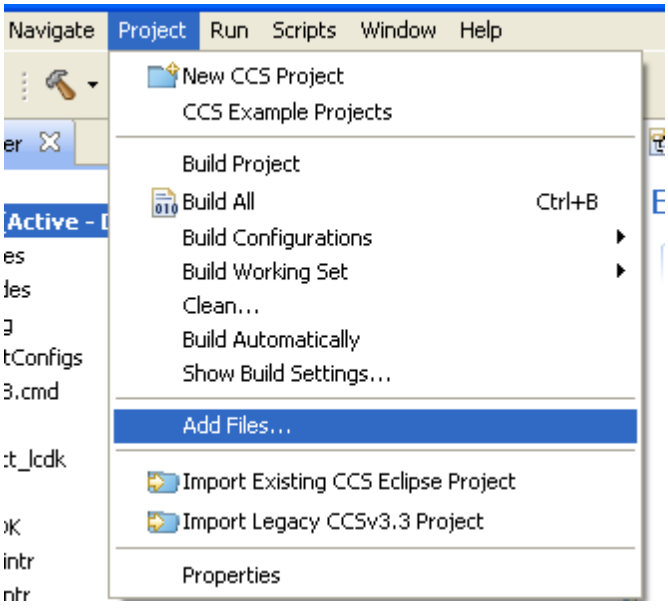
R2. Create source file to write code. In source file type (or copy) the following code.

```
// L138_loop_intr.c
//
#include "L138_LCDK_aic3106_init.h"
interrupt void interrupt4(void) // interrupt service routine
{
uint32_t sample;
sample = input_sample(); // read L + R samples from ADC
output_sample(sample); // write L + R samples to DAC
return;
}
int main(void)
{
L138_initialise_intr(FS_48000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT
);
while(1);
}
```

Once the program is written, save it.

R3. Now add the following library files and supporting files of codec to your project.

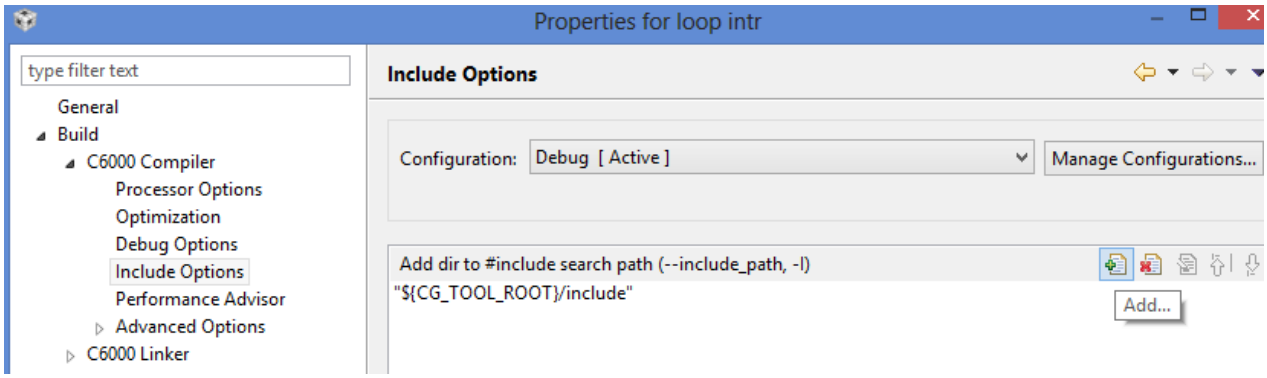
To add files goto Project-> Add files.



Follow the path given below to add the required files:

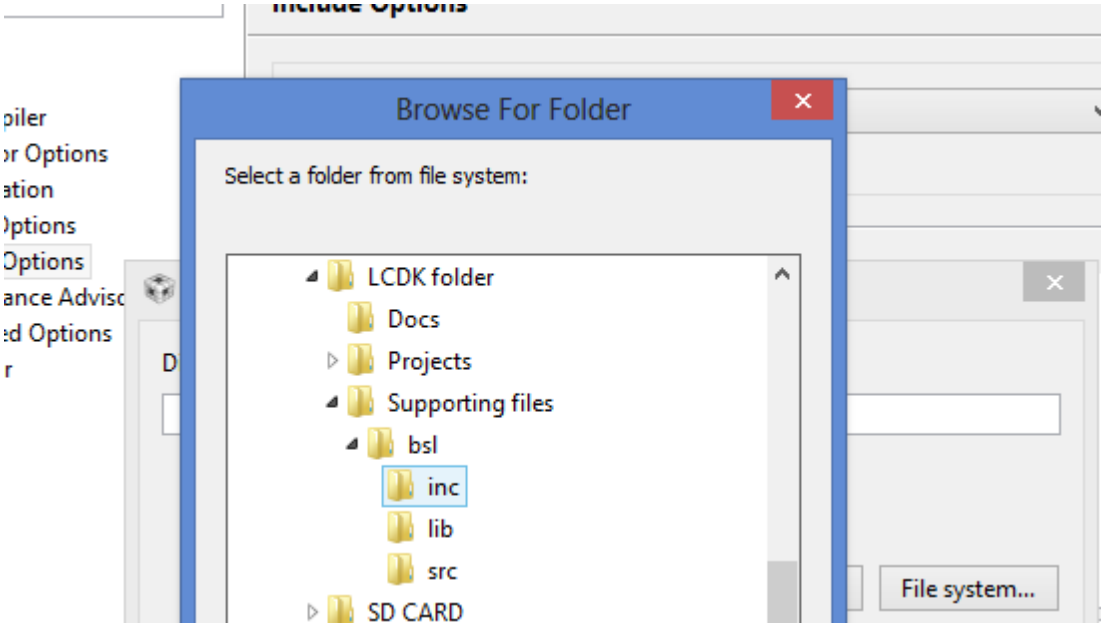
1. Supporting files**evmomapl138_bsl.lib**
2. Supporting files **L138_LCDK_aic3106_init.h**
3. Supporting files **L138_LCDK_aic3106_init.c**
4. Supporting files **vectors_intr.asm**
5. **supporting files\externalram.asm**

R4. Right click on the project name and choose "Show Build Settings.." -> Build-> C6000 Compiler-> Include Options -> Click "Add" as shown in the figure.



Now Click File system -> supporting files->bsl->inc->OK.

Ti\ccsv6\tools\complier\c6000_7.4.14\include



R5. Build Project, Load and Resume the project. After clicking the Resume option, play the music now. You can hear the song from the headphone without any noise. Alternatively you can also connect function generator to the LINE IN and CRO to the LINE OUT. At this time

you can observe same wave appearing on the CRO which is given in function generator. Thus the CODEC is verified. You can also try changing the sampling frequency, gain of the codec.

Note: if you want to play loop signal only in right channel then change the 6th line to **output_right_sample(sample);** If you want to play loop signal only in left channel then change the 6th line to **output_left_sample(sample);**

11. Advance Discrete Time Filter Design (FIR) Finite Impulse Response Filter

DESIGNING AN FIR FILTER:

Following are the steps to design linear phase FIR filters Using Windowing Method.

I. Clearly specify the filter specifications.

Eg: Order = 30;
Sampling Rate = 8000 samples/sec
Cut off Freq. = 400 Hz.

II. Compute the cut-off frequency Wc
Eg: Wc = 2*pi* fc / Fs
= 2*pi* 400/8000
= 0.1*pi

III. Compute the desired Impulse Response h d (n) using particular Window

Eg: b_rect1=fir1(order, Wc , 'high',boxcar(31));

IV. Convolve input sequence with truncated Impulse Response x (n)*h (n)

USING MATLAB TO DETERMINE FILTER COEFFICIENTS :Using FIR1 Function on Matlab

B = FIR1(N,Wn) designs an N'th order lowpass FIR digital filter and returns the filter coefficients in length N+1 vector B. The cut-off frequency Wn must be between 0 < Wn < 1.0, with 1.0
corresponding to half the sample rate. The filter B is real and has linear phase, i.e., even symmetric coefficients obeying B(k) = B(N+2-k), k = 1,2,...,N+1.

If Wn is a two-element vector, Wn = [W1 W2], FIR1 returns an order N bandpass filter with passband W1 < W < W2.
B = FIR1(N,Wn,'high') designs a highpass filter.
B = FIR1(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].
If Wn is a multi-element vector, Wn = [W1 W2 W3 W4 W5 ... WN], FIR1 returns an order N multiband filter with bands 0 < W < W1, W1 < W < W2, ..., WN < W < 1.
B = FIR1(N,Wn,'DC-1') makes the first band a passband.
B = FIR1(N,Wn,'DC-0') makes the first band a stopband.

For filters with a passband near Fs/2, e.g., highpass and bandstop filters, N must be even. By default FIR1 uses a Hamming window. Other available windows, including Boxcar, Hanning, Bartlett, Blackman, Kaiser and Chebwin can be specified with an optional trailing argument. For

example,
B = FIR1(N,Wn,kaiser(N+1,4)) uses a Kaiser window with beta=4.
B = FIR1(N,Wn,'high',chebwin(N+1,R)) uses a Chebyshev window.
By default, the filter is scaled so the center of the first pass band has magnitude exactly one after windowing. Use a trailing 'noscale' argument to prevent this scaling,
e.g. B = FIR1(N,Wn,'noscale'),
B = FIR1(N,Wn,'high','noscale'), B = FIR1(N,Wn,wind,'noscale').

Matlab Program to generate 'FIR Filter-Low Pass' Coefficients using FIR1

```
% FIR Low  pass filters using rectangular, triangular and kaiser windows
% sampling rate - 8000
order = 30;
cf=[500/4000,1000/4000,1500/4000];                cf--> contains set of cut-off  frequencies[Wc]

% cutoff frequency - 500
b_rect1=fir1(order,cf(1),boxcar(31));  Rectangular
b_tri1=fir1(order,cf(1),bartlett(31));     Triangular
b_kai1=fir1(order,cf(1),kaiser(31,8)); Kaisar [Where 8-->Beta Co-efficient]


% cutoff frequency - 1000
b_rect2=fir1(order,cf(2),boxcar(31));
b_tri2=fir1(order,cf(2),bartlett(31));
b_kai2=fir1(order,cf(2),kaiser(31,8));


% cutoff frequency - 1500
b_rect3=fir1(order,cf(3),boxcar(31));
b_tri3=fir1(order,cf(3),bartlett(31));
b_kai3=fir1(order,cf(3),kaiser(31,8));


fid=fopen('FIR_lowpass_rectangular.txt','wt');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
fseek(fid,-1,0);
fprintf(fid,'};');

fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');
fclose(fid);
winopen('FIR_lowpass_rectangular.txt');
```

T.1 : Matlab generated Coefficients for FIR Low Pass Kaiser filter:

Cutoff -500Hz

float b_kai1[31]={-0.000019,-0.000170,-0.000609,-0.001451,-0.002593,-0.003511,-0.003150,0.000000,0.007551,0.020655,0.039383,0.062306,0.086494,0.108031,0.122944,0.128279,0.122944,0.108031,0.086494,0.062306,0.039383,0.020655,0.007551,0.000000,-0.003150,-0.003511,-0.002593,-0.001451,-0.000609,-0.000170,-0.000019};

Cutoff -1000Hz

float b_kai2[31]={-0.000035,-0.000234,-0.000454,0.000000,0.001933,0.004838,0.005671,-0.000000,-0.013596,-0.028462,-0.029370,0.000000,0.064504,0.148863,0.221349,0.249983,0.221349,0.148863,0.064504,0.000000,-0.029370,-0.028462,-0.013596,-0.000000,0.005671,0.004838,0.001933,0.000000,-0.000454,-0.000234, -0.000035};

Cutoff -1500Hz

float b_kai3[31]={-0.000046,-0.000166,0.000246,0.001414,0.001046,-0.003421,-0.007410,0.000000,0.017764,0.020126,-0.015895,-0.060710,-0.034909,0.105263,0.289209,0.374978,0.289209,0.105263,-0.034909,-0.060710,-0.015895,0.020126,0.017764,0.000000,-0.007410,-0.003421,0.001046,0.001414,0.000246,-0.000166, -0.000046};

Cutoff -500Hz

float b_rect1[31]={-0.008982,-0.017782,-0.025020,-0.029339,-0.029569,-0.024895,-0.014970,0.000000,0.019247,0.041491,0.065053,0.088016,0.108421,0.124473,0.134729,0.138255,0.134729,0.124473,0.108421,0.088016,0.065053,0.041491,0.019247,0.000000,-0.014970,-0.024895,-0.029569,-0.029339,-0.025020,-0.017782,-0.008982};

Cutoff -1000Hz

float b_rect2[31]={-0.015752,-0.023869,-0.018176,0.000000,0.021481,0.033416,0.026254,-0.000000,-0.033755,-0.055693,-0.047257,0.000000,0.078762,0.167080,0.236286,0.262448,0.236286,0.167080,0.078762,0.000000,-0.047257,-0.055693,-0.033755,-0.000000,0.026254,0.033416,0.021481,0.000000,-0.018176,-0.023869,-0.015752};

Cutoff -1500Hz

float b_rect2[31]={-0.020203,-0.016567,0.009656,0.027335,0.011411,-0.023194,-0.033672,0.000000,0.043293,0.038657,-0.025105,-0.082004,-0.041842,0.115971,0.303048,0.386435,0.303048,0.115971,-0.041842,-0.082004,-0.025105,0.038657,0.043293,0.000000,-0.033672,-0.023194,0.011411,0.027335,0.009656,-0.016567,-0.020203};

Cutoff -500Hz

```
float b_tri1[31]={0.000000,-0.001185,-0.003336,-0.005868,-0.007885,-0.008298,-0.005988,
0.000000,0.010265,0.024895,0.043368,0.064545,0.086737,0.107877,0.125747,0.138255,
0.125747,0.107877,0.086737,0.064545,0.043368,0.024895,0.010265,0.000000,-0.005988,
-0.008298,-0.007885,-0.005868,-0.003336,-0.001185,0.000000};
```

Cutoff -1000Hz

```
float b_tri2[31]={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};
```

Cutoff -1500Hz

```
float b_tri3[31]={0.000000,-0.001104,0.001287,0.005467,0.003043,-0.007731,-0.013469,
0.000000,0.023089,0.023194,-0.016737,-0.060136,-0.033474,0.100508,0.282844,0.386435,
0.282844,0.100508,-0.033474,-0.060136,-0.016737,0.023194,0.023089,0.000000,-0.013469,
-0.007731,0.003043,0.005467,0.001287,-0.001104,0.000000};
```

MATLAB Program to generate 'FIR Filter-High Pass' Coefficients using FIR1

```
% FIR High pass filters using rectangular, triangular and kaiser windows
% sampling rate - 8000
order = 30;
cf=[400/4000,800/4000,1200/4000]; ;cf--> contains set of cut-off frequencies[Wc]
% cutoff frequency - 400
b_rect1=fir1(order,cf(1),'high',boxcar(31));
b_tri1=fir1(order,cf(1),'high',bartlett(31));
b_kai1=fir1(order,cf(1),'high',kaiser(31,8)); Where Kaiser(31,8)--> '8'defines the value of
'beta'.
% cutoff frequency - 800
b_rect2=fir1(order,cf(2),'high',boxcar(31));
b_tri2=fir1(order,cf(2),'high',bartlett(31));
b_kai2=fir1(order,cf(2),'high',kaiser(31,8));
% cutoff frequency - 1200
b_rect3=fir1(order,cf(3),'high',boxcar(31));
b_tri3=fir1(order,cf(3),'high',bartlett(31));
b_kai3=fir1(order,cf(3),'high',kaiser(31,8));
fid=fopen('FIR_highpass_rectangular.txt','wt');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect1);
fseek(fid,-1,0);
fprintf(fid,'};');
fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect2);
```

```
fseek(fid,-1,0);
fprintf(fid,'};');
fprintf(fid,'\n\n\n\n');
fprintf(fid,'\t\t\t\t\t\t%s\n','Cutoff -1200Hz');
fprintf(fid,'\nfloat b_rect3[31]={');
fprintf(fid,'%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,\n',b_rect3);
fseek(fid,-1,0);
fprintf(fid,'};');
fclose(fid);
winopen('FIR_highpass_rectangular.txt');
```

T.1 : MATLAB generated Coefficients for FIR High Pass Kaiserfilter:

Cutoff -400Hz

float b_kai1[31]={0.000050,0.000223,0.000520,0.000831,0.000845,-0.000000,-0.002478,
-0.007437,-0.015556,-0.027071,-0.041538,-0.057742,-0.073805,-0.087505,-0.096739,
0.899998,-0.096739,-0.087505,-0.073805,-0.057742,-0.041538,-0.027071,-0.015556,
-0.007437,-0.002478,-0.000000,0.000845,0.000831,0.000520,0.000223,0.000050};

Cutoff -800Hz

float b_kai2[31]={0.000000,-0.000138,-0.000611,-0.001345,-0.001607,-0.000000,0.004714,
0.012033,0.018287,0.016731,0.000000,-0.035687,-0.086763,-0.141588,-0.184011,0.800005,
-0.184011,-0.141588,-0.086763,-0.035687,0.000000,0.016731,0.018287,0.012033,0.004714,
-0.000000,-0.001607,-0.001345,-0.000611,-0.000138,0.000000};

Cutoff -1200Hz

float b_kai3[31]={-0.000050,-0.000138,0.000198,0.001345,0.002212,-0.000000,-0.006489,
-0.012033,-0.005942,0.016731,0.041539,0.035687,-0.028191,-0.141589,-0.253270,0.700008,
-0.253270,-0.141589,-0.028191,0.035687,0.041539,0.016731,-0.005942,-0.012033,-0.006489,
-0.000000,0.002212,0.001345,0.000198,-0.000138,-0.000050};

T.2 :MATLAB generated Coefficients for FIR High Pass Rectangular filter

Cutoff -400Hz

float b_rect1[31]={0.021665,0.022076,0.020224,0.015918,0.009129,-0.000000,-0.011158,
-0.023877,-0.037558,-0.051511,-0.064994,-0.077266,-0.087636,-0.095507,-.100422,0.918834,
-0.100422,-0.095507,-0.087636,-0.077266,-0.064994,-0.051511,-0.037558,-0.023877,
-0.011158,-0.000000,0.009129,0.015918,0.020224,0.022076,0.021665};

Cutoff -800Hz

float b_rect2[31]={0.000000,-0.013457,-0.023448,-0.025402,-0.017127,-0.000000,0.020933,
0.038103,0.043547,0.031399,0.000000,-0.047098,-0.101609,-0.152414,-0.188394,0.805541,
-0.188394,-0.152414,-0.101609,-0.047098,0.000000,0.031399,0.043547,0.038103,0.020933,
-0.000000,-0.017127,-0.025402,-0.023448,-0.013457,0.000000};

Cutoff -1200Hz

float b_rect3[31]={-0.020798,-0.013098,0.007416,0.024725,0.022944,-0.000000,-0.028043,
-0.037087,-0.013772,0.030562,0.062393,0.045842,-0.032134,-0.148349,-0.252386,0.686050,
-0.252386,-0.148349,-0.032134,0.045842,0.062393,0.030562,-0.013772,-0.037087,-0.028043,
-0.000000,0.022944,0.024725,0.007416,-0.013098,-0.020798};

T.3 : MATLAB generated Coefficients for FIR High Pass Triangular filter

Cutoff -400Hz

float b_tri1[31]={0.000000,0.001445,0.002648,0.003127,0.002391,-0.000000,-0.004383,
-0.010943,-0.019672,-0.030353,-0.042554,-0.055647,-0.068853,-0.081290,-0.092048,
0.902380,-0.092048,-0.081290,-0.068853,-0.055647,-0.042554,-0.030353,-0.019672,
-0.010943,-0.004383,-0.000000,0.002391,0.003127,0.002648,0.001445,0.000000};

Cutoff -800Hz

float b_tri2[31]={0.000000,-0.000897,-0.003126,-0.005080,-0.004567,-0.000000,0.008373,
0.017782,0.023225,0.018839,0.000000,-0.034539,-0.081287,-0.132092,-0.175834,0.805541,
-0.175834,-0.132092,-0.081287,-0.034539,0.000000,0.018839,0.023225,0.017782,0.008373,
-0.000000,-0.004567,-0.005080,-0.003126,-0.000897,0.000000};

Cutoff -1200Hz

float b_tri3[31]={0.000000,-0.000901,0.001021,0.005105,0.006317,-0.000000,-0.011581,
-0.017868,-0.007583,0.018931,0.042944,0.034707,-0.026541,-0.132736,-0.243196,0.708287,
-0.243196,-0.132736,-0.026541,0.034707,0.042944,0.018931,-0.007583,-0.017868,-0.011581,
-0.000000,0.006317,0.005105,0.001021,-0.000901,0.000000};

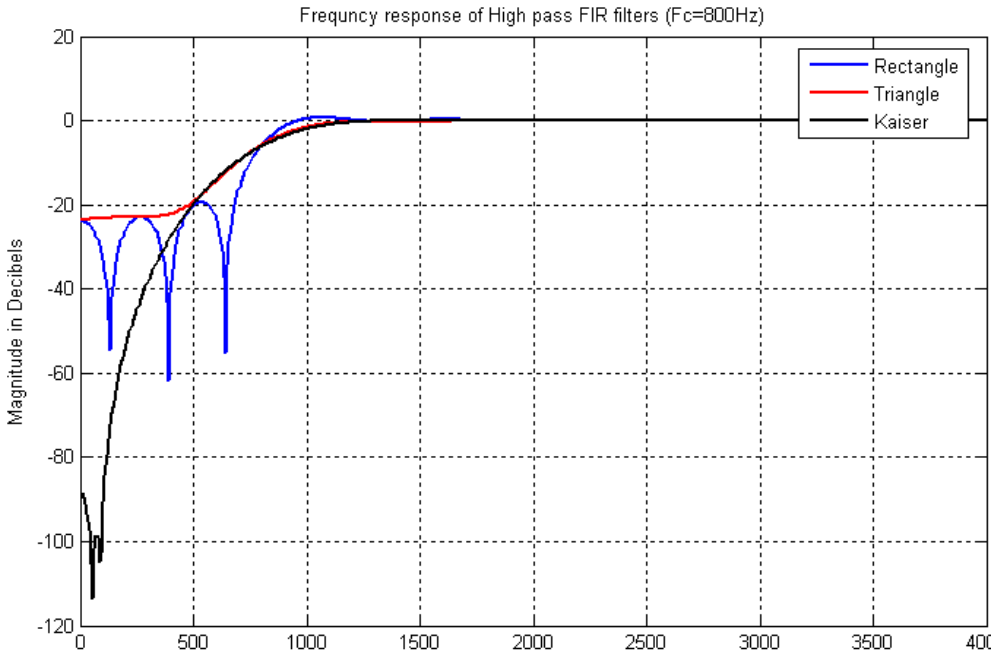
C PROGRAM TO IMPLEMENT FIR FILTER:

```
// L138_fir_intr.c
#include "L138_LCDK_aic3106_init.h"
#define N 5
float h[N] = {
2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001
};
float x[N]; // filter delay line
interrupt void interrupt4(void)
{
short i;
float yn = 0.0;
x[0] = (float)(input_left_sample()); // input from ADC
for (i=0 ; i<N ; i++) // compute filter output
yn += h[i]*x[i];
for (i=(N-1) ; i>0 ; i--) // shift delay line
x[i] = x[i-1];
output_left_sample((uint16_t)(yn)); // output to DAC
return;
}
int main(void)
{
L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
while(1);
}
```

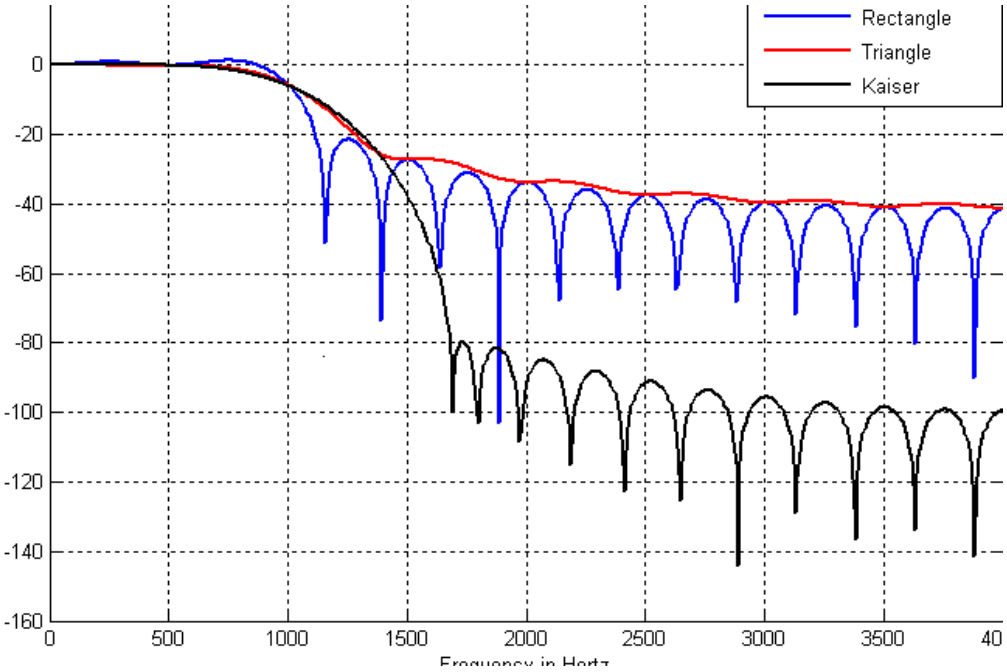
PROCEDURE:

- After successful generation of .out file connect target to host PC
- Connect signal generator to LINE INPUT with 2Vp-p Sine Wave frequency is set according to filter co-efficient
- Connect CRO to LINE OUT with volt/div at 1 and time/div at 5msec.

MATLAB GENERATED FREQUENCY RESPONSE
High Pass FIR filter(Fc= 800Hz).



Low Pass FIR filter (Fc=1000Hz)



12. ADVANCE DISCRETE TIME FILTER DESIGN (IIR)

IIR filter Designing Experiments
GENERAL CONSIDERATIONS:

In the design of frequency – selective filters, the desired filter characteristics are specified in the frequency domain in terms of the desired magnitude and phase response of the filter. In the filter design process, we determine the coefficients of a causal IIR filter that closely approximates the desired frequency response specifications.

IMPLEMENTATION OF DISCRETE-TIME SYSTEMS:
Discrete time Linear Time-Invariant (LTI) systems can be described completely by constant coefficient linear difference equations. Representing a system in terms of constant coefficient linear difference equation is it's time domain characterization. In the design of a simple frequency–selective filter, we would take help of some basic implementation methods for realizations of LTI systems described by linear constant coefficient difference equation.

UNIT OBJECTIVE:
The aim of this laboratory exercise is to design and implement a Digital IIR Filter & observe its frequency response. In this experiment we design a simple IIR filter so as to stop or attenuate required band of frequencies components and pass the frequency components which are outside the required band.

BACKGROUND CONCEPTS:
An Infinite impulse response (IIR) filter possesses an output response to an impulse which is of an infinite duration. The impulse response is "infinite" since there is feedback in the filter, which is if you put in an impulse, then its output must produced for infinite duration of time.

PREREQUISITES:

- Concept of discrete time signal processing.
- Analog filter design concepts.
- TMS320C6748 Architecture and instruction set.

EQUIPMENTS NEEDED:

- Host (PC).
- TMS320C6748 DSP Kit.
- Oscilloscope and Function generator.

ALGORITHM TO IMPLEMENT:

We need to realize the Butter worth band pass IIR filter by implementing the difference equation y[n] = b0x[n] + b1x[n-1]+b2x[n-2]-a1y[n-1]-a2y[n-2] where b0 – b2, a0-a2 are feed forward and feedback word coefficients respectively [Assume 2nd order of filter].These coefficients are calculated using MATLAB.A direct **form I** implementation approach is taken.

- **Step 1 - Initialize** the McASP, the DSP board and the on board codec.
- **Step 2 -** Initialize the discrete time system, that is, specify the initial conditions. Generally zero initial conditions are assumed.
- **Step 3 -** Take sampled data from codec while input is fed to DSP kit from the signal generator. Since Codec is stereo, take average of input data read from left and right channel. Store sampled data at a memory location.
- **Step 4** - Perform filter operation using above said difference equation and store filter Output at a memory location.
- **Step 5 -** Output the value to codec (left channel and right channel) and view the output at Oscilloscope.
- **Step 6 -** Go to step 3.

MATLAB PROGRAM TO GENRATE FILTER CO-EFFICIENTS

```
% IIR Low pass Butterworth and Chebyshev filters
% sampling rate - 24000
order = 2;
cf=[2500/12000,8000/12000,1600/12000];
% cutoff frequency - 2500
[num_bw1,den_bw1]=butter(order,cf(1));
[num_cb1,den_cb1]=cheby1(order,3,cf(1));
% cutoff frequency - 8000
[num_bw2,den_bw2]=butter(order,cf(2));
[num_cb2,den_cb2]=cheby1(order,3,cf(2));
fid=fopen('IIR_LP_BW.txt','wt');
fprintf(fid,'\t\t----------Pass band range: 0-2500Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Monotonic-----\n\n\');
fprintf(fid,'\n float num_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw1);
fprintf(fid,'\nfloat den_bw1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw1);
```

```
fprintf(fid,'\n\n\n\t\t----------Pass band range: 0-8000Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Monotonic-----\n\n');
fprintf(fid,'\nfloat num_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_bw2);
fprintf(fid,'\nfloat den_bw2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_bw2);
fclose(fid);
winopen('IIR_LP_BW.txt');
fid=fopen('IIR_LP_CHEB Type1.txt','wt');
fprintf(fid,'\t\t----------Pass band range: 2500Hz----------\n');
fprintf(fid,'\t\t----------Magnitude response: Rippled (3dB) -----\n\n\');
fprintf(fid,'\nfloat num_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb1);
fprintf(fid,'\nfloat den_cb1[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb1);
fprintf(fid,'\n\n\n\t\t----------Pass band range: 8000Hz----------\n');
fid,'\t\t----------Magnitude response: Rippled (3dB)-----\n\n');
fprintf(fid,'\nfloat num_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',num_cb2);
fprintf(fid,'\nfloat den_cb2[9]={');
fprintf(fid,'%f,%f,%f,%f,%f,\n%f,%f,%f,%f};\n',den_cb2);
fclose(fid);
winopen('IIR_LP_CHEB Type1.txt');
%%%%%%%%%%%%%%%%%%
figure(1);
[h,w]=freqz(num_bw1,den_bw1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb1,den_cb1);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
title('Magnitude response of Low pass IIR filters (Fc=2500Hz)');
figure(2);
[h,w]=freqz(num_bw2,den_bw2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2)
hold on
[h,w]=freqz(num_cb2,den_cb2);
w=(w/max(w))*12000;
plot(w,20*log10(abs(h)),'linewidth',2,'color','r')
grid on
legend('Butterworth','Chebyshev Type-1 (Ripple: 3dB)');
xlabel('Frequency in Hertz');
ylabel('Magnitude in Decibels');
```

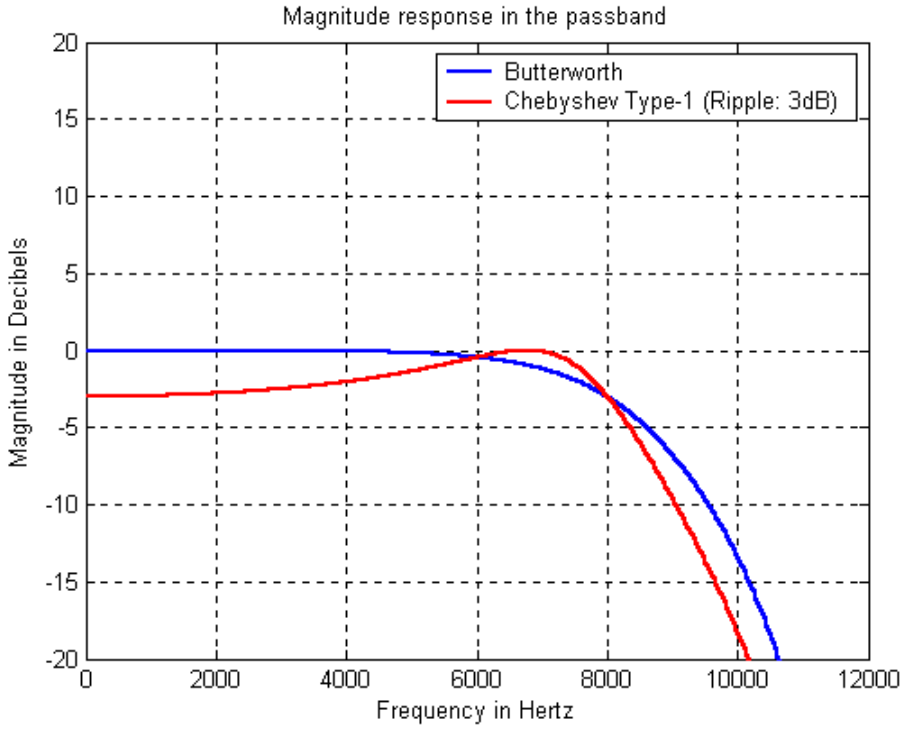
```
title('Magnitude response in the passband');
axis([0 12000 -20 20]);
```

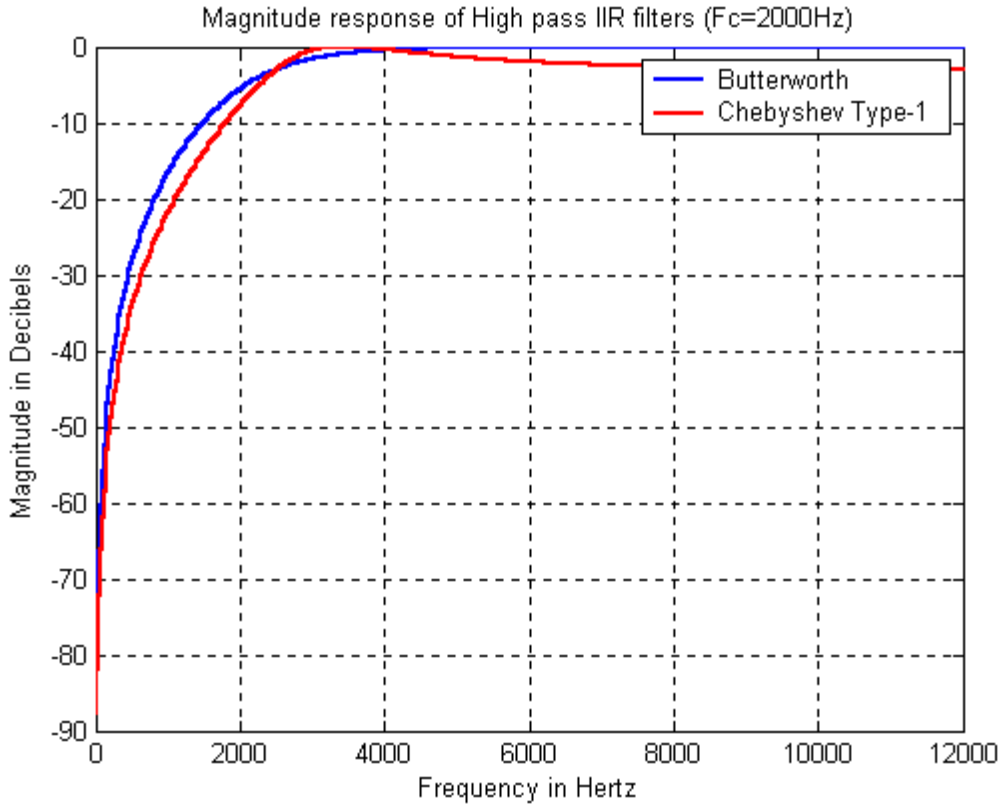
'C' PROGRAM TO IMPLEMENT IIR FILTER

```
// L138_iir_intr.c
// IIR filter implemented using second order sections
// integer coefficients read from file
#include "L138_LCDK_aic3106_init.h"
#include "bs1800int.cof"
int w[NUM_SECTIONS][2] = {0};
interrupt void interrupt4() //interrupt service routine
{
int section; // index for section number
int input; // input to each section
int wn,yn; // intermediate and output values in each stage
input = input_left_sample();
// input = (int)prbs();
for (section=0 ; section< NUM_SECTIONS ; section++)
{
//
wn = input - ((a[section][1]*w[section][0])>>15) -
((a[section][2]*w[section][1])>>15);
//
yn = ((b[section][0]*wn)>>15) + ((b[section][1]*w[section][0])>>15) +
((b[section][2]*w[section][1])>>15);
w[section][1] = w[section][0];
w[section][0] = wn;
input = yn; // output of current section will be input to next
}
output_left_sample((int16_t)(yn)); // before writing to codec
return; //return from ISR
}
int main(void)
{
L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);
while(1);
} // end of main()
```

PROCEDURE:

- Add **bs1800int.cof** file that contains filter co-efficient (will be in IIR filter folder)
- After successful generation of .out file connect target to host PC
- Connect signal generator to LINE INPUT with 2Vp-p Sine Wave frequency is set according to filter co-efficient
- Connect CRO to LINE OUT with volt/div at 1 and time/div at 5msec


Magnitude response in the passband
Butterworth
Chebyshev Type-1 (Ripple: 3dB)
Magnitude in Decibels
Frequency in Hertz
20
15
10
5
0
-5
-10
-15
-20
0
2000
4000
6000
8000
10000
12000


Magnitude response of High pass IIR filters (Fc=2000Hz)
Butterworth
Chebyshev Type-1
Magnitude in Decibels
Frequency in Hertz
0
-10
-20
-30
-40
-50
-60
-70
-80
-90
0
2000
4000
6000
8000
10000
12000

REFERENCES:

References

R1. S. K. Mitra, Digital Signal Processing, 3rd edition, TMH, 2006
R2. Vinay K.Ingle and John G. Proakis, Digital Signal Processing A Matlab –Based Approach, CENGAGE Learning, 2008
R3. Spru189 - TMS320C6000 CPU & Instruction Set Guide
R4. Spru190 - TMS320C6000 Peripherals Guide
R5. Spru186j - TMS320C6713 DSP
R6. Spru509 - Code Composer Studio Getting Started Guide
R7. Matlab Reference Guide
R8. TMS320C6748/OMAPL138 DEVELOPMENT KIT StarCom Information Technology Limited –TI Solutions