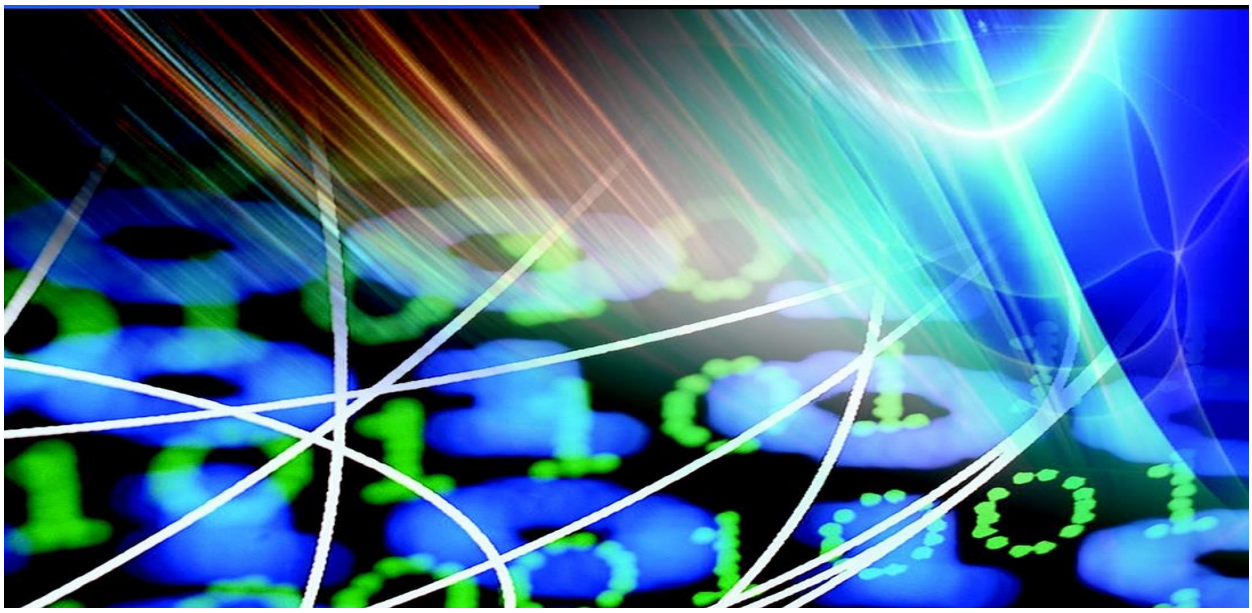
DIGITAL SIGNAL PROCESSING LAB MANUAL



Department of Electronics & Communication Engineering

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

DIGITAL SIGNAL PROCESSING LAB MANUAL

Name:_____

H.T.No:_____

Year/Semester:_____

Department of Electronics & Communication Engineering

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)

VEMU Institute of Technology
Dept. of Electronics and Communication Engineering

Vision of the institute

To be one of the premier institutes for professional education producing dynamic and vibrant force of technocrats with competent skills, innovative ideas and leadership qualities to serve the society with ethical and benevolent approach.

Mission of the institute

Mission_1: To create a learning environment with state-of-the art infrastructure, well equipped laboratories, research facilities and qualified senior faculty to impart high quality technical education.

Mission_2: To facilitate the learners to inculcate competent research skills and innovative ideas by Industry-Institute Interaction.

Mission_3: To develop hard work, honesty, leadership qualities and sense of direction in learners by providing value based education.

Vision of the department

To develop as a center of excellence in the Electronics and Communication Engineering field and produce graduates with Technical Skills, Competency, Quality, and Professional Ethics to meet the challenges of the Industry and evolving Society.

Mission of the department

Mission_1: To enrich Technical Skills of students through Effective Teaching and Learning practices to exchange ideas and dissemination of knowledge.

Mission_2: To enable students to develop skill sets through adequate facilities, training on core and multidisciplinary technologies and Competency Enhancement Programs.

Mission_3: To provide training, instill creative thinking and research attitude to the students through Industry-Institute Interaction along with Professional Ethics and values.

Programme Educational Objectives (PEOs)

PEO 1: To prepare the graduates to be able to plan, analyze and provide innovative ideas to investigate complex engineering problems of industry in the field of Electronics and Communication Engineering using contemporary design and simulation tools.

PEO-2: To provide students with solid fundamentals in core and multidisciplinary domain for successful implementation of engineering products and also to pursue higher studies.

PEO-3: To inculcate learners with professional and ethical attitude, effective communication skills, teamwork skills, and an ability to relate engineering issues to broader social context at work place

Programme Outcomes(Pos)

PO_1	**Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO_2	**Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO_3	**Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO_4	**Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO_5	**Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO_6	**The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO_7	**Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO_8	**Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO_9	**Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO_10	**Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO_11	**Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO_12	**Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcome(PSOs)

PSO_1	**Higher Education :** Qualify in competitive examination for pursuing higher education by applying the fundamental concepts of Electronics and Communication Engineering domains such as Analog & Digital Electronics, Signal Processing, Communication & Networking, Embeded Systems, VLSI Design and Control systems etc.,
PSO_2	**Employment**: Get employed in allied industries through their proficiency in program specific domain knowledge, Specalized software packages and Computer programming or became an entrepreneur.

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR

III B.Tech. II-Sem (ECE)

(19A04602P) DIGITAL SIGNAL PROCESSING LAB

CO No.	Description	BL
C327.1	Analyze discrete time signals & systems using MATLAB and verify using Digital Signal Processor kit with code composer studio(CCS)	04
C327.2	Design IIR & FIR filters using MATLAB & implement using DSP processor kit with code composer studio(CCS)	04

PART-A

The following experiments shall be conducted using MATLAB / Lab View / C Programming/ Equivalent software.

1. Generation of sinusoidal waveform / signal based on recursive difference equations.
2. Find DFT / IDFT of given discrete time signal.
3. Find frequency response of a system given in transfer function/ differential equation form.
4. Implementation of FFT of given Sequence.
5. Design and implementation of IIR filter using bilinear transformation and impulse invariant method.
6. Design and implementation of IIR Butterworth (LP/HP) filter.
7. Design and implementation of IIR Chebyshev (LP/HP) filter.
8. Design and implementation of FIR with low pass filter using any three windowing techniques. Plot its magnitude and phase responses.
9. Design and implementation of FIR filter with high pass filter using any three windowing techniques. Plot its magnitude and phase responses.
10. Design and implementation of FIR filter with band pass / band stop filter using any three windowing techniques. Plot its magnitude and phase responses.

PART-B

The following experiments shall be conducted using (TI / Analog Devices / Motorola / Equivalent DSP processors).

11. Study the architecture of DSP chips – TMS 320C 5X/6X Instructions.
12. Find DFT / IDFT of given discrete time signal.
13. Implementation of FFT of given Sequence.
14. Design and implementation of IIR Butterworth / Chebyshev (LP/HP) filter.
15. Design and implementation of FIR with low pass / high pass filter using any three windowing techniques. Plot its magnitude and phase responses.

VEMU INSTITUTE OF TECHNOLOGY::P.KOTHAKOTA

NEAR PAKALA, CHITTOOR-517112
(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anantapuramu)
Department of Electronics &Communication Engineering

LIST OF EXPERIMENTS TO BE CONDUCTED

PART-A (Using Matlab)

1. Find DFT / IDFT of given discrete time signal.
2. Find frequency response of a system given in transfer function/ differential equation form.
3. Implementation of FFT of given Sequence.
4. Design and implementation of FIR with low pass filter using any three windowing techniques. Plot its magnitude and phase responses.
5. Design and implementation of FIR filter with high pass filter using any three windowing techniques. Plot its magnitude and phase responses.
6. Design and implementation of FIR filter with band pass / band stop filter using any three windowing techniques. Plot its magnitude and phase responses.
7. Design and implementation of IIR Butterworth (LP/HP) filter.
8. Design and implementation of IIR Chebyshev (LP/HP) filter.
9. Generation of FIR Filter Coefficients using windowing techniques.
10. Generation of IIR Butterworth/Chebyshev Filter Coefficients

PART-B (Using CCS)

11. Study the architecture of DSP chips – TMS 320C 5X/6X.
12. Find DFT / IDFT of given discrete time signal.
13. Implementation of FFT of given Sequence.
14. Design and implementation of FIR with low pass / high pass filter using any three windowing techniques. Plot its magnitude and phase responses.
15. Design and implementation of IIR Butterworth / Chebyshev (LP/HP) filter.

(Additional Experiments)

16. Sum of Sinusoidal Signal
17. Power Density Spectrum

CONTENTS

S.NO.	NAME OF THE EXPERIMENT	PAGE NO
PART-A (USING MATLAB)		
1	Find DFT / IDFT of given discrete time signal.	2-5
2	Find frequency response of a system given in transfer function/ differential equation form.	6-8
3	Implementation of FFT of given Sequence.	9-10
4	Design and implementation of FIR with low pass filter using any three windowing techniques. Plot its magnitude and phase responses.	11-14
5	Design and implementation of FIR filter with high pass filter using any three windowing techniques. Plot its magnitude and phase responses.	15-17
6	Design and implementation of FIR filter with band pass / band stop filter using any three windowing techniques. Plot its magnitude and phase responses.	18-20
7	Design and implementation of IIR Butterworth (LP/HP) filter.	21-24
8	Design and implementation of IIR Chebyshev (LP/HP) filter.	25-28
9	Generation of FIR Filter Coefficients using windowing techniques.	29-38
10	Generation of IIR Butterworth/Chebyshev Filter Coefficients.	39-44
PART-B (USING CODE COMPOSURE STUDIO)		
11	Study the architecture of DSP chips – TMS 320C 5X/6X.	46-49
12	Find DFT / IDFT of given discrete time signal.	50-54
13	Implementation of FFT of given Sequence.	55-58
14	Design and implementation of FIR with low pass / high pass filter using any three windowing techniques. Plot its magnitude and phase responses.	59-85
15	Design and implementation of IIR Butterworth / Chebyshev (LP/HP) filter.	86-109
ADDITIONAL EXPERIMENTS		
16	Sum of Sinusoidal Signals	111-113
17	Power Spectral Density	114-119

DOS & DONTS IN LABORATORY

1. While entering the Laboratory, the students should follow the dress code (Wear shoes, White Apron & Female students should tie their hair back).
2. The students should bring their observation note book, Lab manual, record note book, calculator, and necessary stationary items.
3. While sitting in front of the system, check all the cable connections and Switch on the computer.
4. If a student notices any fluctuations in power supply, immediately the same thing is to be brought to the notice of technician/lab in charge.
5. At the end of practical class the system should be switch off safely and arrange the chairs properly.
6. Each program after completion should be written in the observation note book and should be corrected by the lab in charge on the same day of the practical class.
7. Each experiment should be written in the record note book only after getting signature from the lab in charge in the observation note book.
8. Record should be submitted in the successive lab session after completion of the experiment.
9. 100% attendance should be maintained for the practical classes.

SCHEME OF EVALUVATION

S No	Date	Name of The Experiment	Marks Awarded			Sign.
			Observation (10M)	Viva voce (10M)	Total (20M)	
PART-A (USING MATLAB)						
1		Find DFT / IDFT of given discrete time signal.				
2		Find frequency response of a system given in transfer function/ differential equation form.				
3		Implementation of FFT of given Sequence.				
4		Design and implementation of FIR with low pass filter using any three windowing techniques. Plot its magnitude and phase responses.				
5		Design and implementation of FIR filter with high pass filter using any three windowing techniques. Plot its magnitude and phase responses.				
6		Design and implementation of FIR filter with band pass / band stop filter using any three windowing techniques. Plot its magnitude and phase responses.				
7		Design and implementation of IIR Butterworth (LP/HP) filter.				
8		Design and implementation of IIR Chebyshev (LP/HP) filter.				
9		Generation of FIR Filter Coefficients using windowing techniques.				
10		Generation of IIR Butterworth/Chebyshev Filter Coefficients.				
PART-B (USING CODE COMPOSURE STUDIO)						
11		Study the architecture of DSP chips – TMS 320C 5X/6X.				

12		Find DFT / IDFT of given discrete time signal.				
13		Implementation of FFT of given Sequence.				
14		Design and implementation of FIR with low pass / high pass filter using any three windowing techniques. Plot its magnitude and phase responses.				
15		Design and implementation of IIR Butterworth / Chebyshev (LP/HP) filter.				
ADDITIONAL EXPERIMENT						
16		Sum of Sinusoidal Signals				
17		Power Spectral Density				

Signature of Lab In-charge

PART-A

Exp .No: 1 **Date:**

DISCRETE FOURIER TRANSFORM/IDFT

1. AIM: To find Discrete Fourier Transform and Inverse Discrete Fourier Transform of given digital signal.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

THEORY:

Basic equation to find the DFT of a sequence is given below.

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

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

Basic equation to find the IDFT of a sequence is given below.

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1.$$

4. PROGRAM:

```
clc;
close all;
clear all;
xn=input('Enter the sequence x(n)'); %Get the sequence from user
ln=length(xn); %find the length of the sequence
xk=zeros(1,ln); %initialize an array of same size as that of input sequence
ixk=zeros(1,ln); %initialize an array of same size as that of input
sequence
%DFT of the sequence
%-----------------------------------------------------
for k=0:ln-1
for n=0:ln-1
```

```
xk(k+1)=xk(k+1)+(xn(n+1)*exp((i)*2*pi*k*n/ln));
end
end
%-------------------------------------------------------
%Plotting input sequence
%-------------------------------------------------------
t=0:ln-1;
subplot(221);
stem(t,xn);
ylabel ('Amplitude');
xlabel ('Time Index');
title('Input Sequence');
%-------------------------------------------------------------------

magnitude=abs(xk); % Find the magnitudes of individual DFT points
% plot the magnitude response
%-------------------------------------------------------------
t=0:ln-1;
subplot(222);
stem(t,magnitude);
ylabel ('Amplitude');
xlabel ('K');
title('Magnitude Response');
%-------------------------------------------------------------
phase=angle(xk); % Find the phases of individual DFT points % plot the
magnitude sequence
%-------------------------------------------------------------
t=0:ln-1;
subplot(223);
stem(t,phase);
ylabel ('Phase');
xlabel ('K');
title ('Phase Response');
%-------------------------------------------------------------
%IDFT of the sequence
%-------------------------------------------------------------
for n=0:ln-1
for k=0:ln-1
ixk(n+1)=ixk(n+1)+(xk(k+1)*exp(i*2*pi*k*n/ln));
end
end
ixk=ixk./ln;
%-------------------------------------------------------------
%code block to plot the input sequence
%-------------------------------------------------------------
t=0:ln-1;
subplot(224);
stem(t,ixk);
ylabel ('Amplitude');
xlabel ('Time Index');
title ('IDFT sequence');
%-------------------------------------------------------
```

5. INPUT

Enter the sequence x(n) = [1 2 3 4 5 6 7 8]

6. OUTPUT:

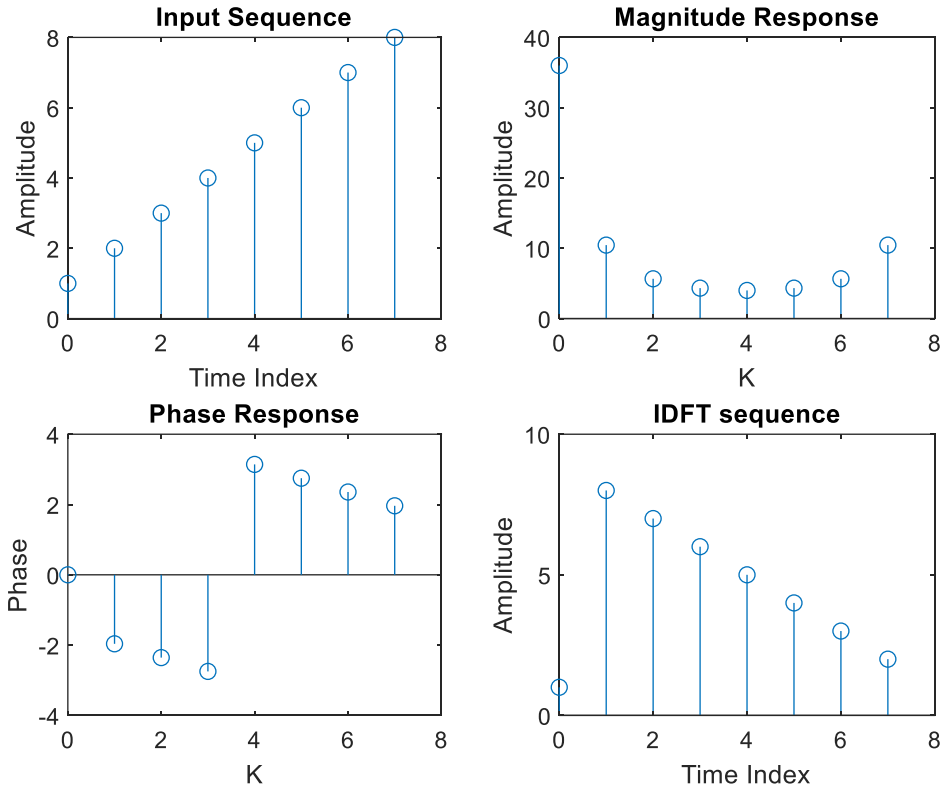
xk =

Columns 1 through 6

36.0000 + 0.0000i -4.0000 - 9.6569i -4.0000 - 4.0000i -4.0000 - 1.6569i -4.0000 + 0.0000i -4.0000 + 1.6569i

Columns 7 through 8

-4.0000 + 4.0000i -4.0000 + 9.6569i



6. RESULT:

7. CONCLUSIONS:

8. VIVA QUESTIONS:

1. What is the difference between DTFT and DFT?

2. Write any Two Properties of DFT?

3. What is zero Padding and Explain the effect of it on magnitude Spectrum?

4. Write the Two properties of Twiddle Factor?

5. How many no. of Complex Multiplications and Additions are required to compute N-Point DFT?

Exp .No: 2 **Date:**

FREQUENCY RESPONSE OF A SYSTEM

1. AIM: To find frequency response of a given system in Transfer function/difference equation form.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

THEORY: Systems respond differently to inputs of different frequencies. Some systems may amplify components of certain frequencies, and attenuate components of other frequencies. The way that the system output is related to the system input for different frequencies is called the *frequency response* of the system. Since the frequency response is a complex function, we can convert it to polar notation in the complex plane. This will give us a magnitude and an angle. We call the angle the *phase*.

Magnitude Response: For each frequency, the magnitude represents the system's tendency to amplify or attenuate the input signal.

$A(Z) = |H(z)|$

Phase Response: The phase represents the system's tendency to modify the phase of the input sinusoids. The phase response, or its derivative the group delay, tells us how the system delays the input signal as a function of frequency.

$\Phi(z) = \text{Angle of } (H(z))$

Given Differential equation is

$$y(n)-\frac{1}{6}y(n-1)-\frac{1}{6}y(n-2)=x(n)$$

Algorithm:

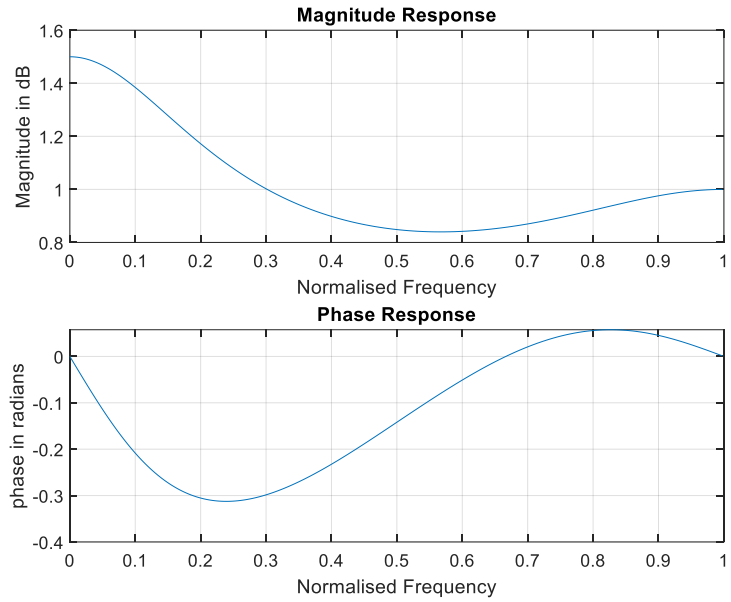
Step I : Give numerator coefficients of the given transfer function difference equation.
Step II : Give denominator coefficients of the given transfer function or difference equation
Step III : Pass these coefficients to matlab command freqz to find frequency response.
Step IV : Find magnitude and phase response using matlab commands abs and angle.
Step V : Plot magnitude and phase response.

4. PROGRAM:

```
%MATLAB program to plot the frequency response (magnitude and phase
response)of a given difference equation.
clc;
clear all;
b=input('Enter the numerator coefficients:');
a=input('Enter the denominator coefficients:');
[h,w]=freqz(b,a);
subplot(2,1,1);
plot(w/pi,abs(h));
grid;
xlabel('Normalised Frequency');
ylabel('Magnitude in dB');
```

```
title('Magnitude Response');
subplot(2,1,2);
plot(w/pi,angle(h));
grid;
xlabel('Normalised Frequency');
ylabel('phase in radians');
title('Phase Response');
```

5. INPUT:
Enter the numerator coefficients: [1]
Enter the denominator coefficients: [1 -1/6 -1/6]
6. OUTPUT:



7. RESULT:

8. CONCLUSION:

9. VIVA QUESTIONS

1. Define Impulse response & Frequency response?

2. Define Magnitude response?

3. Define Phase response?

4. Define transfer function. What is Pole and Zero of the system?

5. Write the difference equation for FIR & IIIR System?

Exp .No: 3 **Date:**

FAST FOURIER TRANSFORM

1. AIM: Write a MATLAB program to perform N-point FFT of a given signal.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- .Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

4. PROGRAM:

```
clc;
clear all;
close all;
n=input('enter value of n=');
x=input('enter input sequence=');
a=1:1:n;
y=fft(x,n);
disp('fft of input sequence');
disp(y);
z=ifft(y);
disp('ifft of input sequence');
disp(z);
```

5. OUTPUT:

```
enter value of n=8
enter input sequence=[1 2 3 4 5 6 7 8]
fft of input sequence
 Columns 1 through 6
 36.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i  -4.0000 + 0.0000i  -
4.0000 - 1.6569i
 Columns 7 through 8
 -4.0000 - 4.0000i  -4.0000 - 9.6569i
ifft of input sequence
    1     2     3     4     5     6     7     8
```

6. RESULT:

7. CONCLUSION:

8. VIVA QUESTIONS:

1.) What is the need of FFT?

2.) What is mean by radix-2 and radix – 4 FFT algorithm?

3.) Write the Block diagram of 8-Point DIT FFT & DIF DFFT radix 2 Algorithm?

4.) Write the Applications of FFT?

5.) How many no. of Complex Multiplications and Additions are required to compute N-Point DFT using FFT?

Exp .No: 4 **Date:**

DESIGN OF FIR LPF USING WINDOWING TECHNIQUE

1. AIM: To write a MATLAB program to design FIR with Low Pass filter using any three Windowing techniques.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

4. PROGRAM:

```
clc;
clear all;
close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter passband freq');
fs=input('enter stopband freq');
f=input('enter sampling freq');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=-14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end;
c=input('enter your choice of window function 1.rectangular 2.triangular 3.kaiser:\n');
if (c==1)
   y=rectwin(n1);
   disp('rectangular window filter response');
end
if(c==2)
   y=triang(n1);
   display('triangular window filter response');
end
if(c==3)
   y=kaiser(n1)
   display('kaiser window filter response');
end
%lpf
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,1);
```

```
plot(o/pi,m);
title('lpf');
ylabel('gain in db-->');
xlabel('(a)normalized freq__>');
```

INPUT:
enter passband ripple2
enter the stopband ripple10
enter passband freq1000
enter stopband freq3000
enter sampling freq7000
enter your choice of window function 1.rectangular 2.triangular 3.kaiser:
1
OUTPUT:

5. RESULT:

6. CONCLUSIONS:

7. VIVA QUESTIONS:

1.) Define a Filter? Compare FIR and IIR filter?

2.) What is mean by Linear Phase Filter? Give Examples

3.) What is window method? How you will design an FIR filter using window method?

4.) Compare all windows used in FIR filter design?

5.) Draw the Linear Phase realization for y(n) = 5 x(n) + 4 x(n-1) + 2 x(n-2) + 4 x(n-3) + 5 x(n-4)?

Exp .No: 5 **Date:**

DESIGN OF FIR HPF USING WINDOWING TECHNIQUE

1. AIM: To write a MATLAB program to design FIR with High Pass filter using any three Windowing techniques.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

4. PROGRAM:

```
clc;
clear all;
close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter passband freq');
fs=input('enter stopband freq');
f=input('enter sampling freq');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=-14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end;
c=input('enter your choice of window function 1.rectangular 2.triangular 3.kaiser:\n');
if (c==1)
   y=rectwin(n1);
   disp('rectangular window filter response');
end
if(c==2)
   y=triang(n1);
   display('triangular window filter response');
end
if(c==3)
   y=kaiser(n1)
   display('kaiser window filter response');
end
%hpf
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,2);
plot(o/pi,m);
```

```
title('hpf');
ylabel('gain in db___>');
xlabel('(b)normalized freq__>');
```

INPUT:
enter passband ripple2
enter the stopband ripple10
enter passband freq1000
enter stopband freq3000
enter sampling freq7000
enter your choice of window function 1.rectangular 2.triangular 3.kaiser:
1

OUTPUT:

5. RESULT:

6. CONCLUSIONS:

7. VIVA QUESTIONS:

1.) Find the Transfer of FIR Filter?

2.) FIR Filter is all zero system? Explain with examples

3.) FIR Filter is always stable? Explain

4.) Define Ripple? What is pass band ripple and stop band ripple.

5.) What is 3-db frequency and Transition band?

Exp .No: 6 **Date:**

DESIGN OF FIR BPF/BSF USING WINDOWING TECHNIQUE

1. AIM: To write a MATLAB program to design FIR with Band Pass/Band Stop filter using any three Windowing techniques.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

4. PROGRAM:

```
clc;
clear all;
close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter passband freq');
fs=input('enter stopband freq');
f=input('enter sampling freq');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=-14.6*(fs-fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
   n1=n;
   n=n-1;
end;
c=input('enter your choice of window function 1.rectangular 2.triangular 3.kaiser:\n');
if (c==1)
   y=rectwin(n1);
   disp('rectangular window filter response');
end
if(c==2)
   y=triang(n1);
   display('triangular window filter response');
end
if(c==3)
   y=kaiser(n1)
   display('kaiser window filter response');
end
%bpf
wn=[wp,ws];
b=fir1(n,wn,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,3);
```

```
plot(o/pi,m);
title('bpf');
ylabel('gain in db__>');
xlabel('(c)normalized freq__>');
%bsf
b=fir1(n,wn,'stop',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
subplot(2,2,4);
plot(o/pi,m);
title('bsf');
ylabel('gain in db__>');
xlabel('(d)normalized freq__>');
```

INPUT:

enter passband ripple2
enter the stopband ripple10
enter passband freq1000
enter stopband freq3000
enter sampling freq7000
enter your choice of window function 1.rectangular 2.triangular 3.kaiser:
1

OUTPUT:

5. RESULT:

6. CONCLUSIONS:

7. VIVA QUESTIONS:

1.) Write the Expression for Rectangular, Triangular and Hamming window?

2.) Explain the Fourier series method for FIR Filter Design?

3.) What is unit of Analog and Digital Frequency?

4.) How do you design an FIR filter using window method?

5.) What are the realization techniques available for FIR Filter and Compare them?

Exp .No: 7 **Date:**

DESIGN OF IIR BUTTERWORTH (LP/HP) FILTERS

1. AIM: To write a MATLAB program to design butterworth IIR LP\HP filters.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

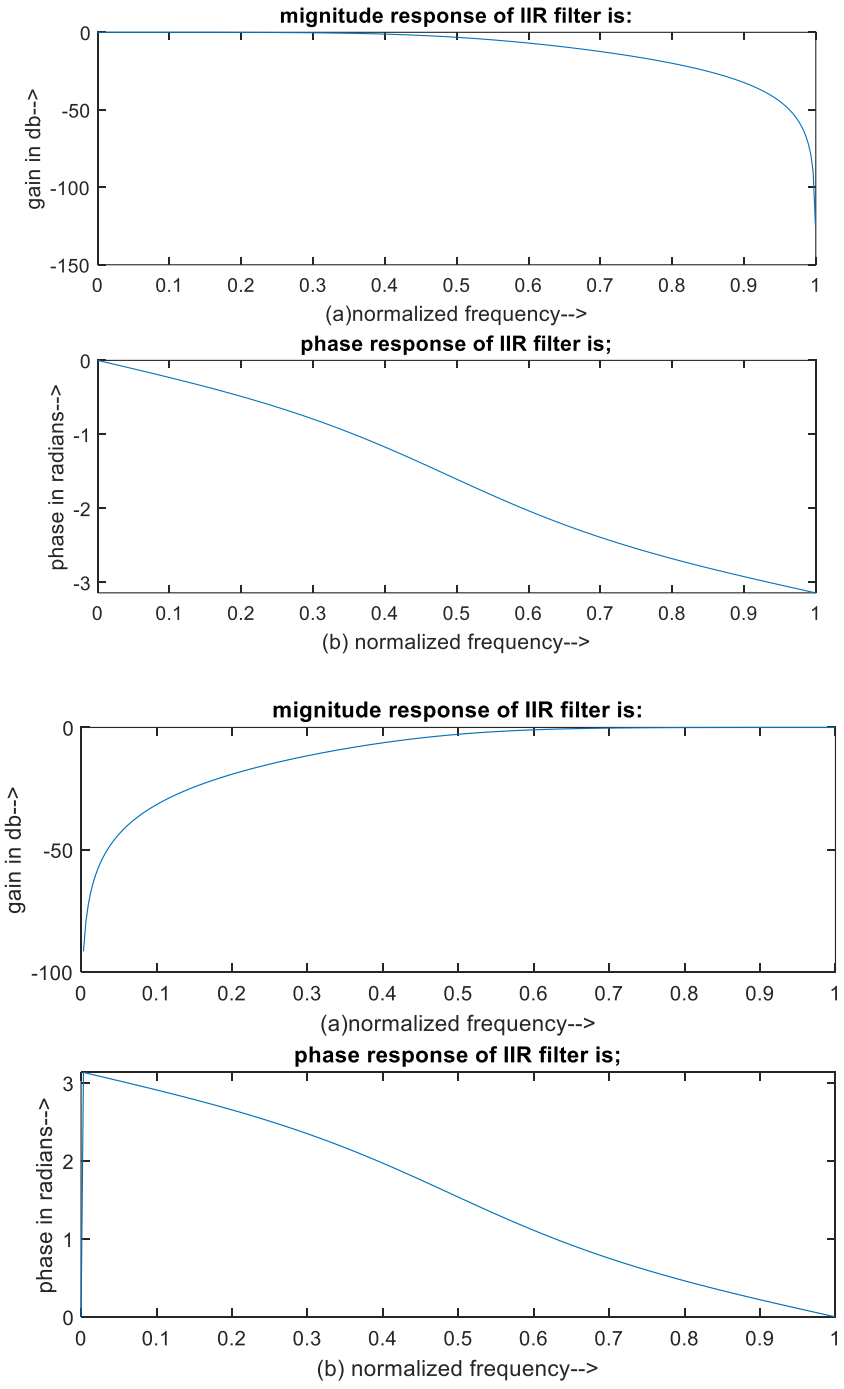
4. PROGRAM:

```
clc;
clear all;
close all;
display('enter the iir filter design specifications');
        rp=input('enter the pass band ripple:');
        rs=input('enter the stop band ripple:');
        wp=input('enter the pass band freq:');
ws=input('enter the stop band freq:');
fs=input('enter the sampling freq:');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
c=input('enter choice filter 1.lpf 2.hpf /n');
if(c==1)
display('frequency response of IIR lpf is:');
[b,a]=butter(n,wn,'low');
end
if(c==2)
display('freq response of IIR hpf IS:');
[b,a]=butter(n,wn,'high');
end
w=0:0.01:pi;
h=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure;
subplot(2,1,1);
plot(w/pi,m);
title('mignitude response of IIR filter is:');
xlabel('(a)normalized frequency-->');
ylabel('gain in db-->');
subplot(2,1,2);
plot(w/pi,an);
titel('phase response of IIR filter is;');
xlabel('(b) normalized frequency-->');
ylabel('phase in radians-->');
```

INPUT:

enter the iir filter design specifications
enter the pass band ripple:2
enter the stop band ripple:20
enter the pass band freq:1000
enter the stop band freq:2000
enter the sampling freq:5000
enter choice filter 1.lpf 2.hpf /n

5. OUTPUT:



6. RESULT:

7. CONCLUSION:

8. VIVA QUESTIONS:

1.) What do you mean by cut-off frequency?

2.) Give the differences between analog and digital filters?

3.) What is the difference between type 1 and type 2 filter structures?

4.) What is the role of delay element in filter design?

5.) List the Differences between Butterworth and chebyshev filters?

Exp .No: 8 **Date:**

DESIGN OF IIR CHEBYSHEV (LP/HP) FILTERS

1. AIM: To write a MATLAB program to design chebyshev IIR LP\HP filters.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

4. PROGRAM:

% Program for the design of Chebyshev Type-1 low-pass filter

```
clc;
close all;clear all;
format long
rp=input(„enter the passband ripple...“);
rs=input(„enter the stopband ripple...“);
wp=input(„enter the passband freq...“);
ws=input(„enter the stopband freq...“);
fs=input(„enter the sampling freq...“);
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=cheb1ord(w1,w2,rp,rs,“s“);
[b,a]=cheby1(n,rp,wn,“s“);
W=0:.01:pi;
[h,om]=freqs(b,a,w);
M=20*log10(abs(h));
An=angle(h);
subplot(2,1,1);
plot(om/pi,m);
ylabel(„Gain in dB --.“);
xlabel(„(a) Normalised frequency --.“);
subplot(2,1,2);
plot(om/pi,an);
xlabel(„(b) Normalised frequency --.“);
ylabel(„Phase in radians --.“);
```

% Program for the design of Chebyshev Type-2 High pass analog filter

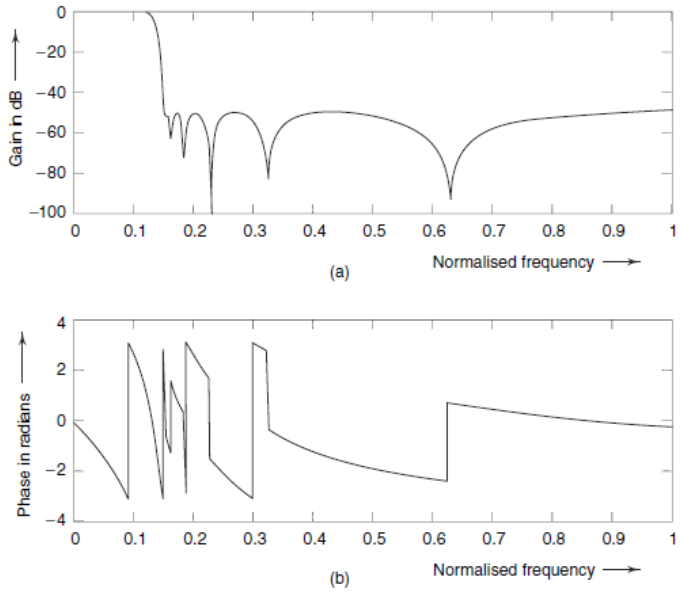
```
clc;
close all;clear all;
format long
rp=input('enter the passband ripple...');
rs=input('enter the stopband ripple...');
wp=input('enter the passband freq...');
ws=input('enter the stopband freq...');
fs=input('enter the sampling freq...');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=cheb2ord(w1,w2,rp,rs,'s');
```

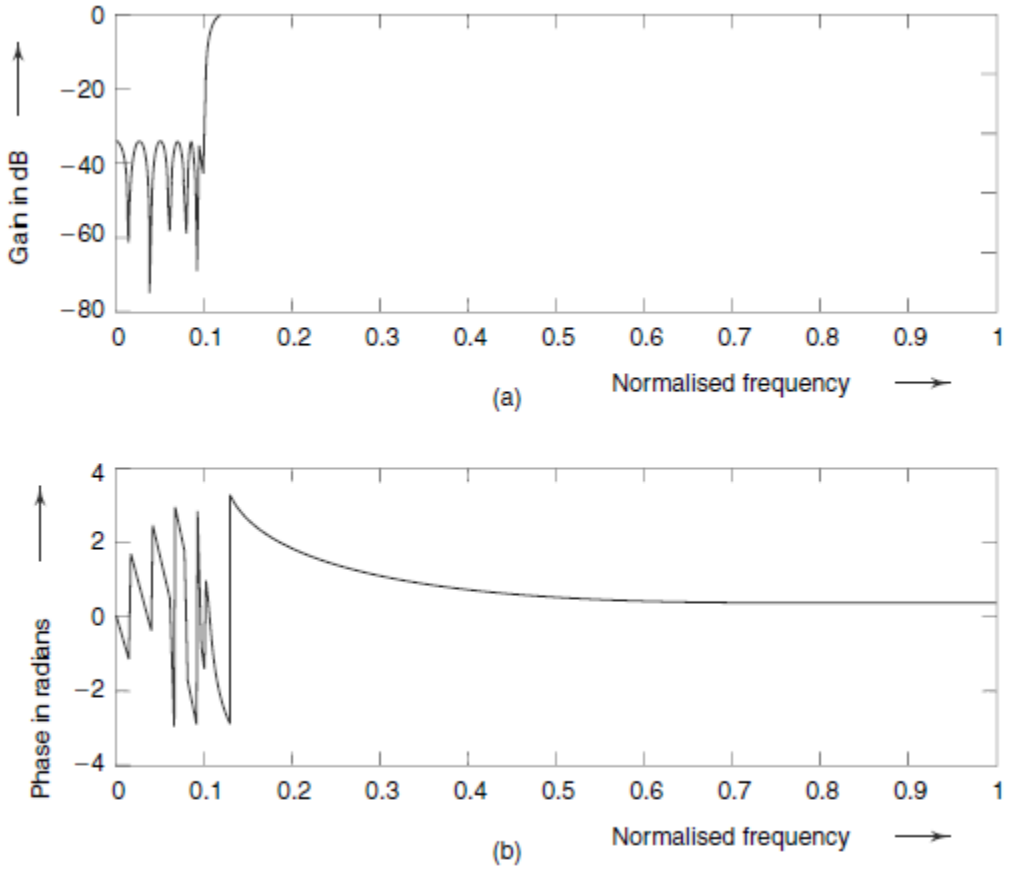
```
[b,a]=cheby2(n,rs,wn,'high','s');
w=0:.01:pi;
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
subplot(2,1,1);
plot(om/pi,m);
ylabel('Gain in dB --.');
xlabel('(a) Normalised frequency --.');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b) Normalised frequency --.');
ylabel('Phase in radians --.');
```

INPUT:

```
LPF
enter the passband ripple... 0.23
enter the stopband ripple... 47
enter the passband freq... 1300
enter the stopband freq... 1550
enter the sampling freq... 7800
HPF
enter the passband ripple... 0.34
enter the stopband ripple... 34
enter the passband freq... 1400
enter the stopband freq... 1600
enter the sampling freq... 10000
```

5. OUTPUT:





Chebyshev Type - 2 High-pass Analog Filter
(a) Amplitude Response and (b) Phase Response

6. RESULT:

8. CONCLUSION:

7. VIVA QUESTIONS:

1.) What is the difference between type 1 and type 2 filter structures?

2.)IIR Filters are non linear phase filters. Why?

3) IIR filters are not always stable. Why?

4.) Write the realization Techniques for IIR Filter?

5.) Compare all realization techniques vailable for IIR Filter?

Exp .No: 9 **Date:**

GENERATION OF FIR LOW PASS/HIGH PASS FILTER COEFFICIENTS

1.AIM: -

To generate FIR filter (LP/HP) coefficients using windowing techniques

- Using Rectangular Window
- Using Triangular Window
- Using Keiser Window

2.APPARATUS REQUIRED: -

MATLAB and PC

3.THEORY: -

Steps to design linear phase FIR filter using windowing methods:

1. Clearly specify the filter specifications
   a. Order of filter (**N**)
   b. Sampling Rate ($\mathbf{F_s}$)
   c. Cutoff frequency ($\mathbf{f_c}$)
2. Compute the Cutoff frequency (ω_c)
   1. $\omega_c = 2\pi * f_c / F_s$
3. Compute the desired impulse response h(n) using particular window.
   - Lowpass Filter with Cutoff frequency(ω_c): -
     - Coefficients of linear phase filter

$$h_d(n) = \omega_c/\pi \quad \text{for } n=\alpha$$
$$= \sin(\omega_c(n-\alpha))/(\pi(n-\alpha)) \quad \text{for } n\neq\alpha$$

Where α=(N-1)/2.

- Window Sequence
  - **a.** For Rectagular Window

$$W(n)=1 \quad \text{for } 0\leq n\leq(N-1)$$
$$=0 \quad \text{otherwise}$$

  - **b.** For Triangular Window

$$W(n)=2n/(N-1) \quad \text{for } 0\leq n\leq(N-1)/2$$
$$=2-\{2n/(N-1) \quad \text{for } (N-1)/2\leq n\leq(N-1)$$

  - **c.** For Keiser Window

$$W(n)=I_0[\beta((N-1)^2/4-[n-(N-1)/2]^2)^{1/2}]/I_0[\beta(N-1)/2] \text{ for } 0\leq n\leq(N-1)$$
$$=0 \quad \text{otherwise}$$

where β is the variable parameter whose choice control the tradeoff between side lobe amplitude and side lobe width.

$4<\beta[(N-1)/2]<9$

- The Impulse Response
  $h(n)=h_d(n)*w(n)$

➢ Highpass Filter with Cutoff frequency(ω_c): -

- Coefficients of linear phase filter

$$\mathbf{h_d(n)= 1-\omega_c/\pi \quad for\ n=\alpha}$$
$$\mathbf{= (1/(\pi(n-\alpha))) [\sin(\pi(n-\alpha))-\sin(\omega_c(n-\alpha))] \quad for\ n\neq\alpha}$$

Where α=(N-1)/2.

- Window Sequence
  - **d.** For Rectagular Window

$$\mathbf{W(n)=1 \quad for\ 0\leq n\leq (N-1)/2}$$
$$\mathbf{=0 \quad otherwise}$$

  - **e.** For Triangular Window

$$\mathbf{W(n)=2n/(N-1) \quad for\ 0\leq n\leq (N-1)/2}$$
$$\mathbf{=2-\{2n/(N-1) \quad for\ (N-1)/2\leq n\leq (N-1)}$$

  - **f.** For Keiser Window

$$\mathbf{W(n)=I_0\ [\beta((N-1)^2/4-[n-(N-1)/2]^2)^{1/2}]/I_0[\beta(N-1)/2]}\ \text{for } 0\leq n\leq (N-1)$$
$$\mathbf{=0 \quad otherwise}$$

  where β is the variable parameter whose choice control the tradeoff between side lobe amplitude and side lobe width.

  $4<\beta[(N-1)/2]<9$

- The Impulse Response
  $h(n)=h_d(n)*w(n)$

4. Convolve input sequence with truncated impulse response
   $x(n)*h(n)$.

4.PROGRAM: -

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\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_highpass_rectangular.txt');
```

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%s\n','Cutoff -400Hz');
fprintf(fid,'\nfloat b_rect1[31]={');
fprintf(fid,'%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%s\n','Cutoff -800Hz');
fprintf(fid,'\nfloat b_rect2[31]={');
fprintf(fid,'%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%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 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};

T.2 :Matlab generated Coefficients for FIR Low Pass Rectangular filter

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};

T.3 : Matlab generated Coefficients for FIR Low Pass Triangular filter

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};

T.1 : MATLAB generated Coefficients for FIR High Pass Kaiser filter:

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};

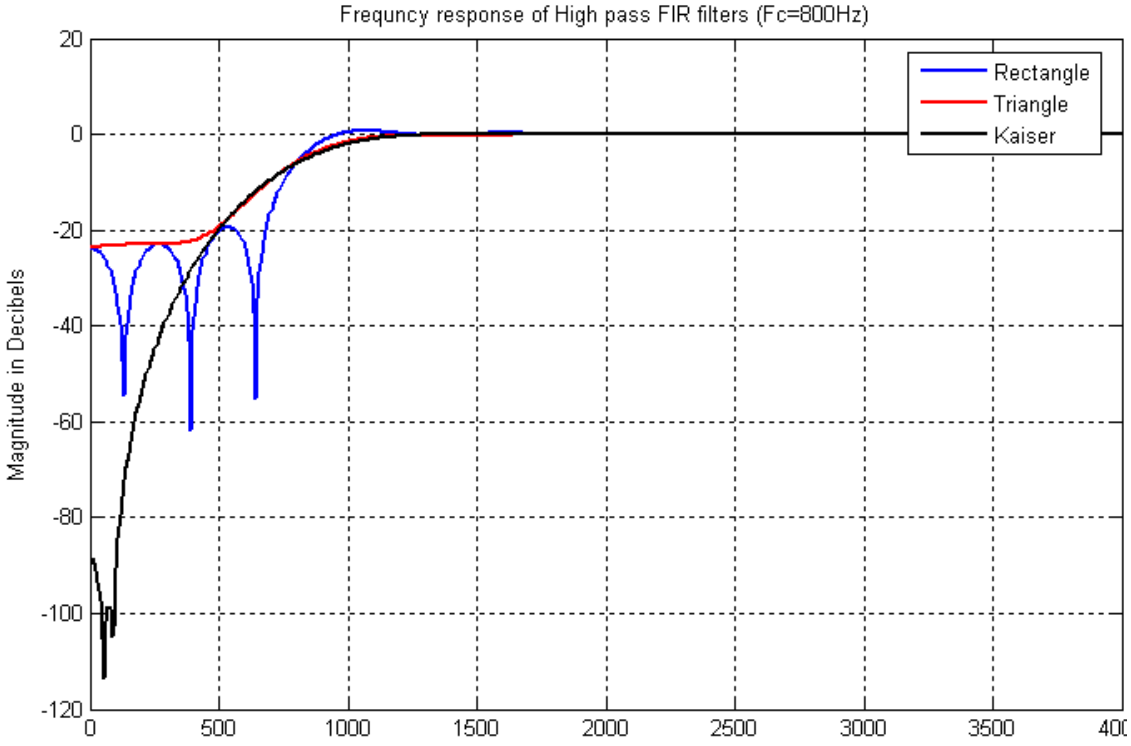
Cutoff -1200Hz

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};

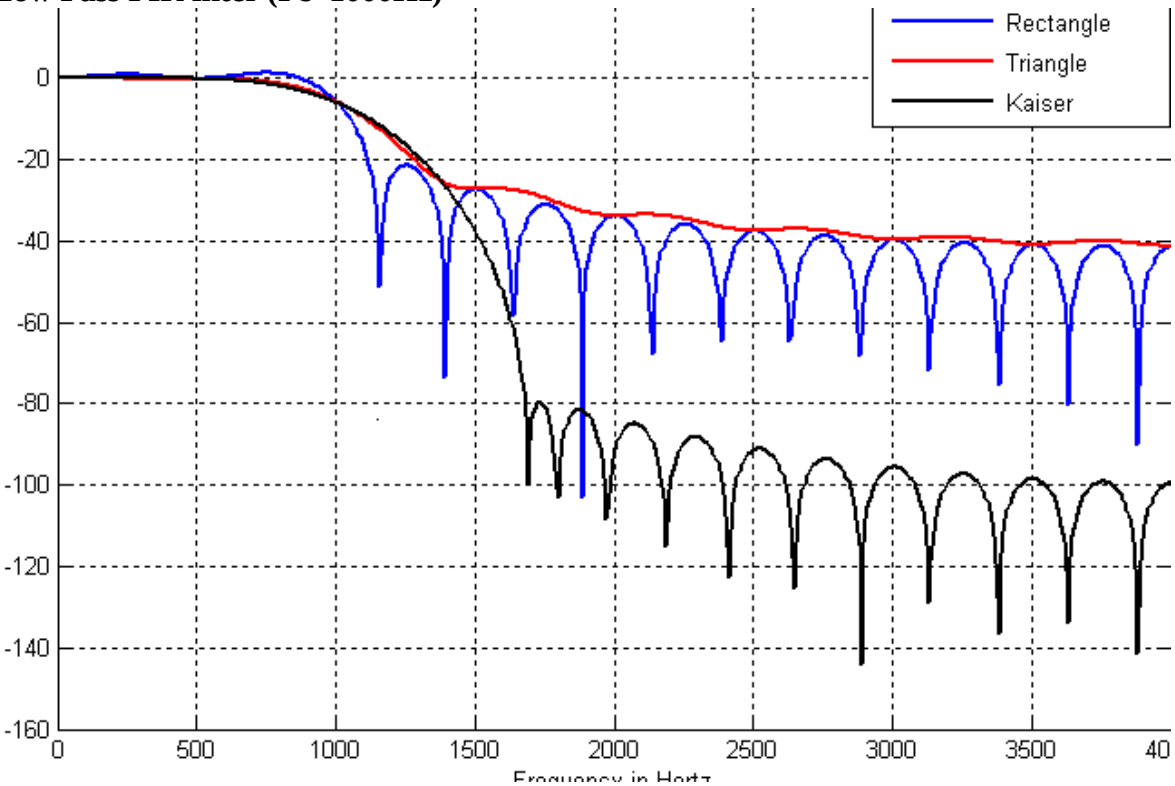
5.MODEL GRAPHS:-

MATLAB GENERATED FREQUENCY RESPONSE

High Pass FIR filter(Fc= 800Hz).



Low Pass FIR filter (Fc=1000Hz)



6. RESULT: -

7.CONCLUSION

8. VIVA QUESTIONS:

1. What are windowing techniques in FIR filters?

2. Which window is best for FIR filter?

3.Which technique is not used in design os FIR FILETR?

4. Why FIR filters are always stable?

5.Why windows are necessary in FIR filters

Exp .No: 10 **Date:**

GENERATION OF IIR BUTTERWORTH/CHEBYSHEV FILTER COEFFICIENTS

1. AIM: -

To generate IIR filter butterworth/chebyshev coefficients

2. APPARATUS REQUIRED: -

MATLAB and PC

3.PROGRAM: -

```
% 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');
fprintf(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]);
```

IIR_CHEB_LP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=800Hz		Fc=8000Hz	
	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)
B0	0.044408	1455	0.005147	168	0.354544	11617
B1	0.088815	1455[B1/2]	0.010295	168[B1/2]	0.709088	11617 [B1/2]
B2	0.044408	1455	0.005147	168	0.354544	11617
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.412427	-23140 [A1/2]	-1.844881	-30225 [A1/2]	0.530009	8683[A1/2]
A2	0.663336	21735	0.873965	28637	0.473218	15506

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_BUTTERWORTH_LP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=800Hz		Fc=8000Hz	
	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)
B0	0.072231	2366	0.009526	312	0.465153	15241
B1	0.144462	2366[B1/2]	0.019052	312[B1/2]	0.930306	15241 [B1/2]
B2	0.072231	2366	0.009526	312	0.465153	15241
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18179 [A1/2]	-1.705552,	-27943 [A1/2]	0.620204	10161 [A1/2]
A2	0.398152	13046	0.743655	24367	0.240408	7877

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

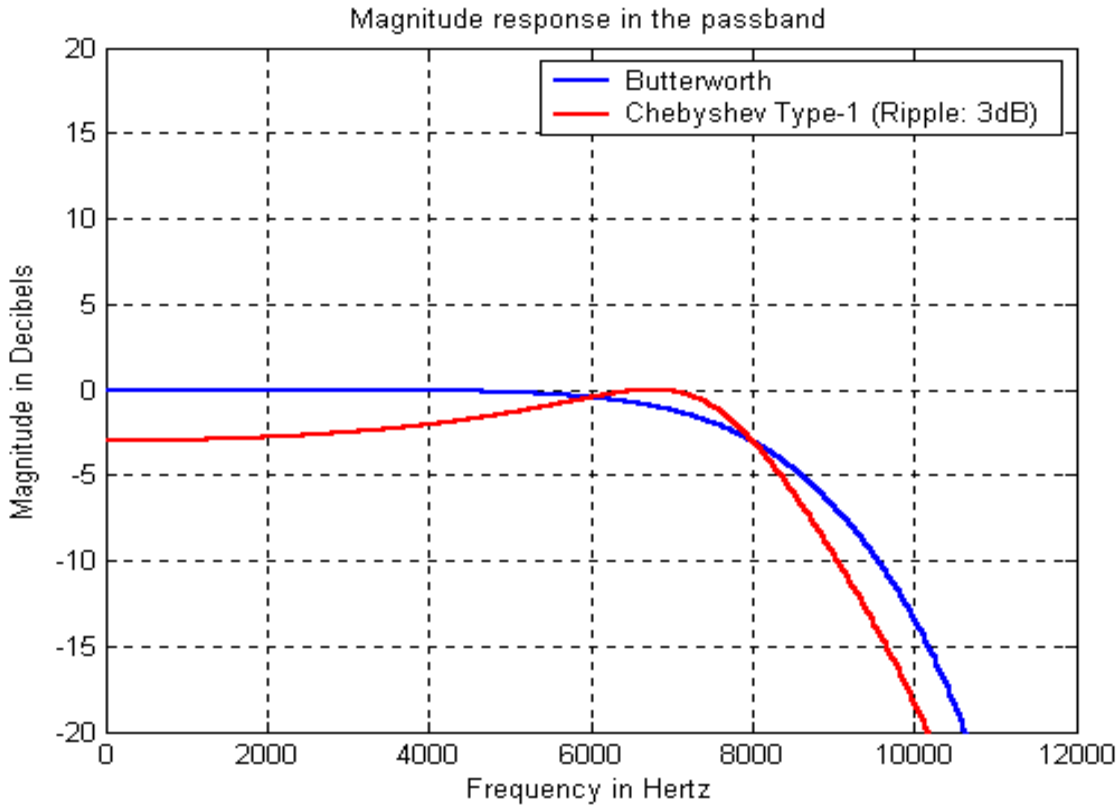
IIR_CHEB_HP FILTER CO-EFFICIENTS:

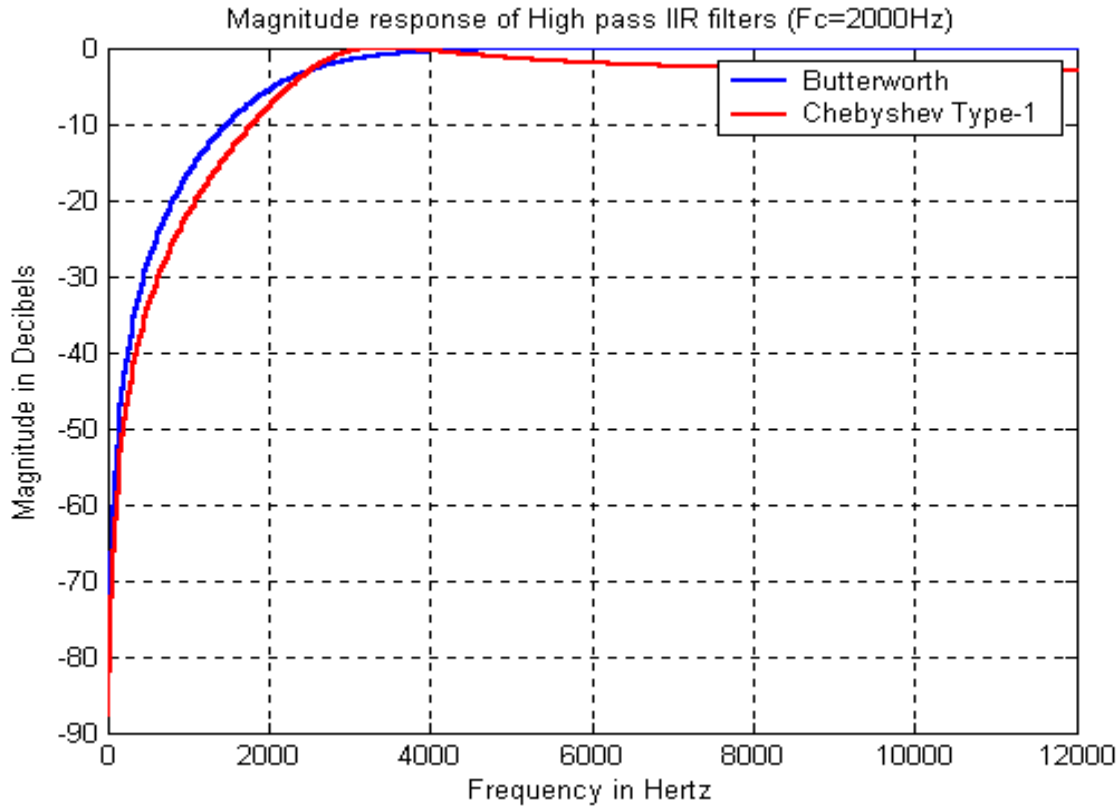
Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)
B0	0.388513	12730	0.282850	9268	0.117279	3842
B1	-0.777027	-12730[B1/2]	-0.565700	-9268[B1/2]	-0.234557	-3842[B1/2]
B2	0.388513	12730	0.282850	9268	0.117279	3842
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.118450	-18324[A1/2]	-0.451410	-7395[A1/2]	0.754476	12360[A1/2]
A2	0.645091	21137	0.560534	18367	0.588691	19289

Note: We have Multiplied Floating Point Values with 32767(2^{15}) to get Fixed Point Values.

IIR_BUTTERWORTH_HP FILTER CO-EFFICIENTS:

Co-Efficients	Fc=2500Hz		Fc=4000Hz		Fc=7000Hz	
	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)	Floating Point Values	Fixed Point Values (Q15)
B0	0.626845	20539	0.465153	15241	0.220195	7215
B1	-1.253691	-20539 [B1/2]	-0.930306	-15241 [B1/2]	-0.440389	-7215 [B1/2]
B2	0.626845	20539	0.465153	15241	0.220195	7215
A0	1.000000	32767	1.000000	32767	1.000000	32767
A1	-1.109229	-18173 [A1/2]	-0.620204	-10161 [A1/2]	0.307566	5039 [A1/2]
A2	0.398152	13046	0.240408	7877	0.188345	6171


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

4. RESULT: -

5. CONCLUSION

6. VIVA QUESTIONS:

1. What is bilinear transformation in IIIR filter design?

2. Which method is not suitable for designing IIR filters?

3. What are the steps involved to design digital filter using bilinear transformations?

4. What are the limitations of impulse invariance method

5. What are the differences between IIR and FIR filters

PART-B

Exp .No: 11 **Date:**

Study the architecture of DSP chips – TMS 320C 5X/6X

1. AIM: - To Study the architecture of DSP chips – TMS 320C 5X/6X.

.

INTRODUCTION

The 6745 PL-DSP is a low-cost standalone development platform that enables users to evaluate and develop applications for the TI C67XX DSP family. The DSP also serves as a hardware reference design for the TMS320C6745 DSP. Schematics, logic equations and application notes are available to ease hardware development and reduce time to market.

An on-board AIC23 codec allows the DSP to transmit and receive analog signals. SPI is used for the codec control interface and McASP0 is used for data. Analog audio I/O is done through two 3.5mm audio jacks that correspond to microphone input, and headphone output and also line input, line output. The codec can select the microphone or the line input as the active input. The analog output is driven to both the line out (fixed gain) and headphone (adjustable gain) connectors.

The PL-DSP includes 8 LEDs, 8 DIP switches, 4*4 LED matrix, LCD and Seven segment as a simple way to provide the user with interactive feedback. It also includes phone keypad to study DTMF signals.

The PL-DSP includes Real time clock displayed on LCD to learn RTC and I2C protocol.

Code Composer communicates with the DSP through an embedded JTAG emulator with a USB host interface.

TMS320C6745PL-DSP Features

The PL-DSP features the TMS320C6745PL-DSP, a 375 MHz device delivering up to 3648 million instructions per second (MIPs) and 2736 MFLOPS.

This PL-DSP generation is designed for applications that require high precision accuracy. The C6745 is based on the TMS320C6000 DSP platform designed to needs of high-performing high-precision applications such as pro-audio, medical and diagnostic.

Other hardware features of the TMS320C6745 DSK board include:

- Embedded JTAG supported via USB
- TLV320AIC23B programmable stereo codec
- Two 3.5mm audio jacks for microphone and speaker
- Expansion for port connector for plug-in modules
- Power supply : +5V, ±12V, GND
- 8 DIP switches for inputs
- 8 LED indication for output
- Provision for manual Reset
- 4*4 LED matrix
- Noise generator : White noise generator
  : Amplitude 0 ~ 5Vpp
- 20*2 character LCD display.
- 2 No. 7 segment displays.
- RTC interface: I2C based RTC section
- Phone keypad: 0 to 9 digits and *, # characters

BLOCK DIAGRAM OF TMS320C6745

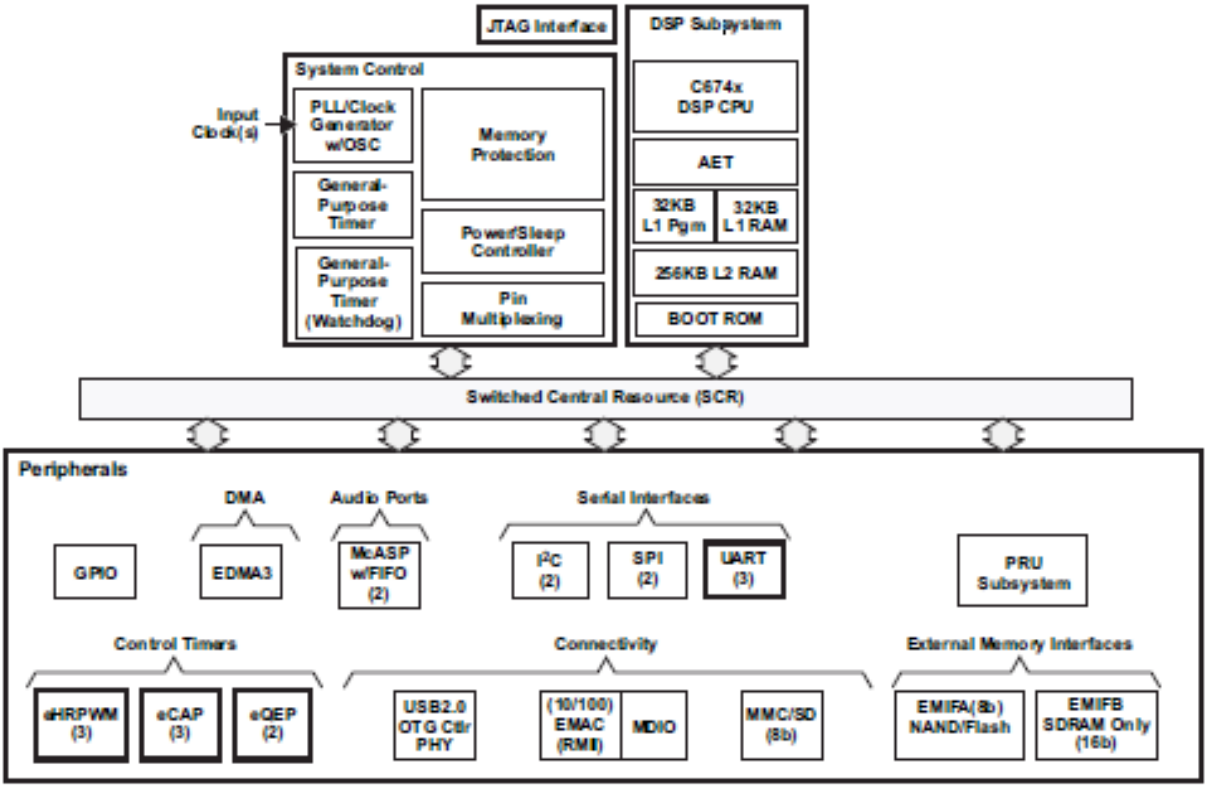
The TMS320C6745/6747 device is a low-power digital signal processor based on a TMS320C674x DSP core. It consumes significantly lower power than other members of the TMS320C6000™ platform of DSPs.

The TMS320C6745/6747 device enables original-equipment manufacturers (OEMs) and original-design manufacturers (ODMs) to quickly bring to market devices featuring high processing performance.

The TMS320C6745/6747 DSP core uses a two-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 other hosts in the system, an additional 128KB of RAM shared memory (TMS320C6747 only) is available for use by other hosts without affecting DSP performance.

The peripheral set includes: a 10/100 Mbps Ethernet MAC (EMAC) with a management data input/output(MDIO) module; two I2C Bus interfaces; 3 multichannel audio serial ports (McASPs) with 16/9 serializers and FIFO buffers; two 64-bit general-purpose timers each configurable (one configurable as watchdog); a configurable 16-bit host-port interface (HPI) [TMS320C6747 only]; up to 8 banks of 16 pins of general purpose input/output (GPIO) with programmable interrupt/event generation modes, multiplexed with other peripherals; 3 UART interfaces (one with both RTS and CTS); three enhanced high-resolution pulse width modulator (eHRPWM) peripherals; three 32-bit enhanced capture (eCAP) module peripherals which can be configured as 3 capture inputs or 3 auxiliary pulse width modulator (APWM) outputs; two 32-bit enhanced quadrature encoded pulse (eQEP) peripherals; and 2 external memory interfaces: an asynchronous and SDRAM external memory interface (EMIFA) for slower memories or peripherals, and a higher speed memory interface (EMIFB) for SDRAM.

The Ethernet Media Access Controller (EMAC) provides an efficient interface between the TMS320C6745/6747 device and the network. The EMAC supports both 10Base-T and 100Base-TX, or 10 Mbps and 100 Mbps in either half- or full-duplex mode. Additionally, an MDIO interface is available for PHY configuration.



6. RESULT:

7. Conclusion

8. Viva Questions

1. What is TMS in DSP?

2. What is the process of down-sampling called?

3. What are types of instruction of TMS320C5X/6X?

4. What is the reason for the need of high speed DSP?

5. What are the types of digital signal processor?

Exp .No: 12 **Date:**

DFT/IDFT OF A SIGNAL

1. AIM: - To compute the N (=4/8/16) point DFT of the given sequence.

2. APPARATUS REQUIRED

- CODE COMPOSER STUDIO(CCS)V5
- TMS320C6745 DSP TRAINER KIT

3. PROCEDURE

1. To create the New Project
2. Project→ New CCS Project (Give name to project with location to save or use default
3. location)
4. Select project type→ Executable
5. Device Family→C6000
6. Variant→C674xFloating-point DSP
7. Connection →Texas Instruments XDS100V2USB Emulator
8. Click on Empty Project then Finish
9. To create a Source file
10. File →New→ Source file (Save & give file name, Eg: sum.c).Click on Finish
11. Write the C-Program To build the program project →Build All
12. After Build Finished without errors, Now DSP kit is turned ON
13. Debug the Program after loading is done
14. Run the Program and verify the output
15. Program asks for values of x[N], you have to enter values as 1st real & 2nd imaginary for all N no of values.

    Eg. x[N]={1+0j,1+0j,1+0j,1+0j,1+0j,1+0j,1+0j,1+0j}.
    It enter in program as → 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

4. PROGRAM:

DFT

```
#include<stdio.h>
#include<math.h>
#define pi 3.14

typedef struct {
                        float real;
                        float img;
                }com;

void main()
{
        com x[50],y[50],temp[50];
        int pofd=0,i,k,n;
        float WN,c,s,WK;
        for(i=0;i<50;i++)                       /*make output array value to 0.0*/
        {
                y[i].real=0.0;
                y[i].img=0.0;
```

```
	}

	printf("\nEnter the no of points of dft:");
	scanf("%d",&pofd);
	printf("\nEnter the function variables of f(n) to calculate DFT in the
complex form (e.g. 3+j4 enter as a 3 4)");
	for(i=0;i<pofd;i++)
	{
		//scanf("%f%f",&x[i].real,&x[i].img);
		scanf("%f",&x[i].real);
		scanf("%f",&x[i].img);
	}
	WN=(2*pi)/pofd;
	for(k=0;k<pofd;k++)
	{
		temp[k].real=0.0;
		temp[k].img=0.0;

			WK=k*WN;
			for(n=0;n<pofd;n++)	/*DFT calculation*/
			{
				c=cos(WK*n);
				s=sin(WK*n);
				temp[k].real=temp[k].real+(x[n].real*c)+(x[n].img*s);
				temp[k].img=temp[k].img+(x[n].img*c)-(x[n].real*s);
			}


			 y[k].real=1/sqrt(pofd)*temp[k].real;
			 y[k].img=1/sqrt(pofd)*temp[k].img;


		 }
		 printf("\nDFT OF GIVEN SIGNAL:-\n");  /*Display values of F[k]*/
		 for(i=0;i<pofd;i++)
		 {
			printf("\nF(%d)=(%0.1f)+j(%0.1f)\n",i,y[i].real,y[i].img);
		 }
}
```

IDFT:

```
#include<stdio.h>
#include<math.h>
#define pi 3.14

typedef struct {
		float real;
		float img;
		 }com;

void main()
{
	com x[20],y[20],temp[20];
	int pofd,i,k,n;
	float WN,c,s,WK;
	for(i=0;i<=20;i++)				/*make output array value to
0.0*/
	{
		y[i].real=0.0;
```

```
			y[i].img=0.0;
		}
		printf("\nEnter the no of points of idft:");
		scanf("%d",&pofd);
		printf("\nEnter the function variables of f(n) to calculate IDFT in the
complex form (e.g. 3+j4 enter as a 3 4)\n");
		for(i=0;i<pofd;i++)
		{
			scanf("%f\t%f",&x[i].real,&x[i].img);
		}
		WN=(2*pi)/pofd;
		for(k=0;k<pofd;k++)
		{
			temp[k].real=0.0;
			temp[k].img=0.0;

				WK=k*WN;
				for(n=0;n<pofd;n++)			/*DFT calculation*/
				{
					c=cos(WK*n);
					s=sin(WK*n);
					temp[k].real=temp[k].real+(x[n].real*c)-(x[n].img*s);
					temp[k].img=temp[k].img+(x[n].img*c)+(x[n].real*s);
				}


				 y[k].real=1/sqrt(pofd)*temp[k].real;
				 y[k].img=1/sqrt(pofd)*temp[k].img;


			}
			printf("\nIDFT OF GIVEN SIGNAL:-\n");  /*Display values of F[k]*/
			for(i=0;i<pofd;i++)
			{
				printf("\nf(%d)=(%0.1f)+j(%0.1f)\n",i,y[i].real,y[i].img);
			}
}
```

5. OUTPUT:

DFT:

- Input x[n]= {1+0j, 1+0j, 1+0j, 1+0j, 1+0j, 1+0j, 1+0j, 1+0j}.
- Output X[K]={2.8+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0,0.0+j0.0}.

```
N_POINT_DFT:CIO
[C674X_0]
[C674X_0] DFT OF GIVEN SIGNAL:-
[C674X_0]
[C674X_0] F(0)=(2.8)+j(0.0)
[C674X_0]
[C674X_0] F(1)=(-0.0)+j(-0.0)
[C674X_0]
[C674X_0] F(2)=(-0.0)+j(-0.0)
[C674X_0]
[C674X_0] F(3)=(-0.0)+j(-0.0)
[C674X_0]
[C674X_0] F(4)=(0.0)+j(-0.0)
[C674X_0]
[C674X_0] F(5)=(0.0)+j(-0.0)
[C674X_0]
[C674X_0] F(6)=(0.0)+j(-0.0)
[C674X_0]
[C674X_0] F(7)=(0.0)+j(-0.0)
```

IDFT:

- Input x[n]= {2.8+0j, 0+0j, 0+0j, 0+0j, 0+0j, 0+0j, 0+0j, 0+0j}.
- Output X[K]={1.0+j0.0, 1.0+j0.0, 1.0+j0.0, 1.0+j0.0, 1.0+j0.0, 1.0+j0.0, 1.0+j0.0, 1.0+j0.0}.

```
N_PIONT_IDFT:CIO
[C674X_0] Enter the no of points of idft:8
[C674X_0]
[C674X_0] Enter the function variables of f(n) to calculate IDFT in the complex form (e.g. 3+j4 enter as a 3 4)
2.8 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[C674X_0]
[C674X_0] IDFT OF GIVEN SIGNAL:-
[C674X_0]
[C674X_0] f(0)=(1.0)+j(0.0)
[C674X_0]
[C674X_0] f(1)=(1.0)+j(0.0)
[C674X_0]
[C674X_0] f(2)=(1.0)+j(0.0)
[C674X_0]
[C674X_0] f(3)=(1.0)+j(0.0)
[C674X_0]
[C674X_0] f(4)=(1.0)+j(0.0)
[C674X_0]
[C674X_0] f(5)=(1.0)+j(0.0)
[C674X_0]
[C674X_0] f(6)=(1.0)+j(0.0)
[C674X_0]
[C674X_0] f(7)=(1.0)+j(0.0)
```

6. RESULT:

7. Conclusion

8. Viva Questions

1. Define DFT of a discrete time sequence?

2. Define inverse DFT.

3. What is the relation between DTFT and DFT?

4. What is the drawback in Fourier transform and how is it overcome?

5. List any four properties of DFT.

Exp .No: 13 **Date:**

N POINT FFT ALGORITHM

1. AIM: To write a C-program to compute N-Point FFT of given sequences.

2. APPARATUS REQUIRED

- CODE COMPOSER STUDIO
- TMS320C6745 DSP TRAINER KIT

3. PROCEDURE

1. To create the New Project
2. Project→ New CCS Project (Give name to project with location to save or use default
3. location)
4. Select project type→ Executable
5. Device Family→C6000
6. Variant→C674xFloating-point DSP
7. Connection →Texas Instruments XDS100V2USB Emulator
8. Click on Empty Project then Finish
9. To create a Source file
10. File →New→ Source file (Save & give file name, Eg: sum.c).Click on Finish
11. Write the C-Program To build the program project →Build All
12. After Build Finished without errors, Now DSP kit is turned ON
13. Debug the Program after loading is done
14. Run the Program and verify the output
15. Program ask for values of x[N], you have to enter values as 1st real & 2nd imaginary for all N no of values.

    Eg. x[N]={1+0j,1+0j,1+0j,1+0j,1+0j,1+0j,1+0j,1+0j}.
    It enter in program as 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0

4. PROGRAM:

```
#include<stdio.h>
#include<math.h>
#define N 8
#define PI 3.14159
typedef struct
{
float real,imag;
}
complex;
main()
{
int i;
complex w[N];
complex x[8]={0,0.0,1,0.0,2,0.0,3,0.0,4,0.0,5,0.0,6,0.0,7,0.0};
```

```
complex temp1,temp2;
int j,k,upper_leg,lower_leg,leg_diff,index,step;
for(i=0;i<N;i++)
{
w[i].real=cos((2*PI*i)/(N*2.0));
w[i].imag=-sin((2*PI*i)/(N*2.0));
}
leg_diff=N/2;
step=2;
for(i=0;i<3;i++)
{
index=0;
for(j=0;j<leg_diff;j++)
{
for(upper_leg=j;upper_leg<N;upper_leg+=(2*leg_diff))
{
lower_leg=upper_leg+leg_diff;
temp1.real=(x[upper_leg]).real+(x[lower_leg]).real;
temp1.imag=(x[upper_leg]).imag+(x[lower_leg]).imag;
temp2.real=(x[upper_leg]).real-(x[lower_leg]).real;
temp2.imag=(x[upper_leg]).imag-(x[lower_leg]).imag;
(x[lower_leg]).real=temp2.real*(w[index]).real-temp2.imag*(w[index]).imag;
(x[lower_leg]).imag=temp2.real*(w[index]).imag+temp2.imag*(w[index]).real;
(x[upper_leg]).real=temp1.real;
(x[upper_leg]).imag=temp1.imag;
}
index+=step;
}
leg_diff=(leg_diff)/2;
step=step*2;
}
j=0;
for(i=1;i<(N-1);i++)
{
k=N/2;
while(k<=j)
{
j=j-k;
k=k/2;
}
j=j+k;
if(i<j)
{
temp1.real=(x[j]).real;
temp1.imag=(x[j]).imag;
(x[j]).real=(x[i]).real;
```

```
(x[j]).imag=(x[i]).imag;
(x[i]).real=temp1.real;
(x[i]).imag=temp1.imag;
}
}
printf("the fft of the given input sequence is \n");
for(i=0;i<8;i++)
{
printf("%f %f \n",(x[i]).real,(x[i]).imag);
}
}
```

5. OUTPUT:

- Input x[n]= {1+0j, 1+0j, 1+0j, 1+0j, 1+0j, 1+0j, 1+0j, 1+0j}.
- Output X[K]={8.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0, 0.0+j0.0}.

6. RESULT:

7.CONCLUSION

8.VIVA QUESTIONS

1. What is FFT, What it's importance?

2. Compare FFT and DFT?

3. What are the various algorithms to calculate FFT?

4. Draw the DIT FFT structure with the length of 8?

5. Draw the DIF FFT structure with the length of 8?

Exp .No: 14 **Date:**

DESIGN OF FIR FILTER USING WINDOWING TECHNIQUES

1. AIM: To design and implement a FIR LP/HP filters using any three windowing techniques.

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

2. APPARATUS REQUIRED

- CODE COMPOSER STUDIO
- TMS320C6745 DSP TRAINER KIT

3. PROCEDURE:

1. To create the New Project
2. Project→ New CCS Project (Give name to project with location to save or use default
3. location)
4. Select project type→ Executable
5. Device Family→C6000
6. Variant→C674xFloating-point DSP
7. Connection →Texas Instruments XDS100V2USB Emulator
8. Click on Empty Project then Finish
9. To create a Source file
10. File →New→ Source file (Save & give file name, Eg: sum.c).Click on Finish
11. Write the C-Program To build the program project →Build All
12. After Build Finished without errors, Now DSP kit is turned ON
13. Debug the Program after loading is done
14. Run the Program and verify the output
15. Connect CRO to the **LINE OUT.**
16. Connect a Signal Generator to the **LINE IN.**
17. Switch on the Signal Generator with a sine wave of frequency 100 Hz. and Vp-p=1.0v&vary the frequency.

1. FOR LOW PASS FILTER:

a. For fir low pass rectangular window (cutoff 500Hz)
b. For fir low pass triangular window (cutoff 1000Hz)
C. For fir low pass Kaiser Window (cutoff 1500Hz)

2. FOR HIGH PASS FILTER:

a. For fir high pass rectangular window (cutoff 400Hz)
b. For fir high pass triangular window (cutoff 800Hz)
C. For fir high pass triangular window (cutoff 1200Hz)

1(a) RECTANGULAR WINDOW FOR LPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;

void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {
    }
}
void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );
    /* Release SPI */
    SPI_SPIGCR0 = 1;

    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS
      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //
    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );

    SPI_SPIDAT1 = spidat1;
    SPI_SPIDELAY = 0
```

```
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY
   SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high
    SPI_SPIINT = 0;
    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}

float filter_Coeff[] ={-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};
           static short in_buffer[100];
           signed int FIR_FILTER(float  h[], signed int x);
           void main( )
              {
           Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
           Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
           Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
           Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
           Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
           Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
           Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
           Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
           Uint32 GBLCTL_XFRST_ON               =      0X00001000;
           Uint32 GBLCTL_RFRST_ON               =      0X00000010;
         int temp,i;
       signed int input_data1, input_data2;
       Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8,
        0X0811, 0X0A00, 0X0C00, 0X0E53, 0X100C, 0x1201};
       PINMUX7 = 0x10110000;
       PINMUX9 = 0x11011000;
       PINMUX10 = 0x00000110;
       spi_init( );
       temp = SPI_SPIDAT1;

             for(i=0; i<10; i++)
             {
                   SPI_SPIDAT1= temp | codec_reg_val[i];
                   while( !(SPI_SPIFLG&0x200));
                   DSP6745_wait( 1000 );
             }
             SPI_SPIGCR0 = 0;
                   DSP6745_wait( 1000 );

        MCASP0_GBLCTL     = 0;
        MCASP0_RGBLCTL    = 0;
        MCASP0_XGBLCTL    = 0;
        MCASP0_PWRDEMU    = 1;
        MCASP0_RMASK      = 0xFFFFFFFF;
        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no pad, CFGBus
        MCASP0_AFSRCTL    = 0x00000000; // 2TDM,1bit Rising edge INTERNAL FS, word
       MCASP0_ACLKRCTL    =  0x00000081;//0x000000AF;//Rising INTERNAL CLK(from tx
side)
      MCASP0_AHCLKRCTL   = 0x00008000;//INT CLK (from tx side)
      MCASP0_RTDM        = 0x00000003; // Slots 0,1
```

```
      MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                           MCASP0_PFUNC      = 0x00;
                        MCASP0_PDIR          = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used


             /* Starting sections of the McASP*/
                  MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                  while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                  MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                  while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                  MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                  while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                  MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                  while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                  MCASP0_XSTAT = 0x0000ffff;        // Clear all
                  MCASP0_RSTAT = 0x0000ffff;        // Clear all

                  MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                  while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                  MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                  while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );
```

```
                /* Write a 0, so that no underrun occurs after releasing the
state machine */

                MCASP0_XBUF0 = 0;

                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );



                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);


                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

               while(1)
                {

                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = FIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }


}



signed int FIR_FILTER(float  h[], signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

1(b) TRIANGULAR WINDOW FOR LPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;
void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}
void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );
              /* Release SPI */
    SPI_SPIGCR0 = 1;

    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );
```

```
 SPI_SPIDAT1 = spidat1;
         SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;
    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}

float filter_Coeff[] ={0.000000,-0.001591,-
0.002423,0.000000,0.005728,0.011139,0.010502,
-0.000000,-0.018003,-0.033416,-
0.031505,0.000000,0.063010,0.144802,0.220534,0.262448,
0.220534,0.144802,0.063010,0.000000,-0.031505,-0.033416,-0.018003,-
0.000000,0.010502,
0.011139,0.005728,0.000000,-0.002423,-0.001591,0.000000};

static short in_buffer[100];

signed int FIR_FILTER(float  h[], signed int x);
void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
          Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
          Uint32 GBLCTL_XFRST_ON       =      0X00001000;
          Uint32 GBLCTL_RFRST_ON       =      0X00000010;
      int temp,i;
      signed int input_data1, input_data2;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
                  DSP6745_wait( 1000 );
                      MCASP0_GBLCTL     = 0;
                      MCASP0_RGBLCTL    = 0;
                      MCASP0_XGBLCTL    = 0;
                      MCASP0_PWRDEMU    = 1;
```

```
                        MCASP0_RMASK      = 0xFFFFFFFF;
                        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                        MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                        MCASP0_RTDM       = 0x00000003; // Slots 0,1
                        MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256
                         MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                           MCASP0_PFUNC      = 0x00;
                        MCASP0_PDIR          = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used

             /* Starting sections of the McASP*/
                 MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                 while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                 MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                 while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                 MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                 while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                 MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                 while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                 MCASP0_XSTAT = 0x0000ffff;        // Clear all
                 MCASP0_RSTAT = 0x0000ffff;        // Clear all

                 MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
```

```
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                /* Write a 0, so that no underrun occurs after releasing the
state machine */

                MCASP0_XBUF0 = 0;

                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );



                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);


                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

               while(1)
                {
                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = FIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);
                                }
                    }
signed int FIR_FILTER(float  h[], signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

1(c) KAISER WINDOW FOR LPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;
void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {
    }
}
void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );
    /* Release SPI */
    SPI_SPIGCR0 = 1;
              /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS
      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //
    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );
    SPI_SPIDAT1 = spidat1;
    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high
```

```
    SPI_SPIINT = 0;
    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}
float filter_Coeff[] ={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};
static short in_buffer[100];
signed int FIR_FILTER(float  h[], signed int x);
void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
          Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
          Uint32 GBLCTL_XFRST_ON       =      0X00001000;
          Uint32 GBLCTL_RFRST_ON       =      0X00000010;
      int temp,i;
      signed int input_data1, input_data2;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
                  DSP6745_wait( 1000 );

                     MCASP0_GBLCTL     = 0;
                  MCASP0_RGBLCTL    = 0;
                  MCASP0_XGBLCTL    = 0;
                  MCASP0_PWRDEMU    = 1;

                  MCASP0_RMASK      = 0xFFFFFFFF;
                  MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                  MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                  MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                  MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                  MCASP0_RTDM       = 0x00000003; // Slots 0,1
                  MCASP0_RINTCTL    = 0x00000000; // Not used
```

```
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                        MCASP0_PFUNC    = 0x00;
                        MCASP0_PDIR       = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used


            /* Starting sections of the McASP*/
                MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                MCASP0_XSTAT = 0x0000ffff;        // Clear all
                MCASP0_RSTAT = 0x0000ffff;        // Clear all

                MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                /* Write a 0, so that no underrun occurs after releasing the
state machine */
```

```
                MCASP0_XBUF0 = 0;

                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );



                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);


                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

               while(1)
                {

                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = FIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }


}



signed int FIR_FILTER(float  h[], signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

2(a) RECTANGULAR WINDOW FOR HPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;
void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}
void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );

    /* Release SPI */
    SPI_SPIGCR0 = 1;
    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers

    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );

    SPI_SPIDAT1 = spidat1;
```

```
    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;

    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}

float filter_Coeff[] ={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};
static short in_buffer[100];
signed int FIR_FILTER(float  h[], signed int x);
void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
          Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
          Uint32 GBLCTL_XFRST_ON       =      0X00001000;
          Uint32 GBLCTL_RFRST_ON       =      0X00000010;

      int temp,i;
      signed int input_data1, input_data2;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
                  DSP6745_wait( 1000 );

                      MCASP0_GBLCTL     = 0;
                      MCASP0_RGBLCTL    = 0;
                      MCASP0_XGBLCTL    = 0;
                      MCASP0_PWRDEMU    = 1;
```

```
                        MCASP0_RMASK      = 0xFFFFFFFF;
                        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                        MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                        MCASP0_RTDM       = 0x00000003; // Slots 0,1
                        MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                        MCASP0_PFUNC      = 0x00;
                        MCASP0_PDIR          = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used


            /* Starting sections of the McASP*/
                MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                MCASP0_XSTAT = 0x0000ffff;        // Clear all
                MCASP0_RSTAT = 0x0000ffff;        // Clear all
```

```
                MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                /* Write a 0, so that no underrun occurs after releasing the
state machine */

                MCASP0_XBUF0 = 0;

                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );



                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);

                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

               while(1)
                {         while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                         input_data1 = MCASP0_RBUF1_32BIT;
                         input_data2 = FIR_FILTER(filter_Coeff , input_data1);
                         while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                         MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }
}
signed int FIR_FILTER(float  h[], signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

2(b) TRIANGULAR WINDOW FOR HPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char

static Uint32 spidat1;

void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}
void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );

    /* Release SPI */
    SPI_SPIGCR0 = 1;

    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers

    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );
```

```
    SPI_SPIDAT1 = spidat1;

    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;

    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}

float filter_Coeff[] ={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};

static short in_buffer[100];

signed int FIR_FILTER(float  h[], signed int x);

void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
          Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
          Uint32 GBLCTL_XFRST_ON       =      0X00001000;
          Uint32 GBLCTL_RFRST_ON       =      0X00000010;

      int temp,i;
      signed int input_data1, input_data2;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
```

```
                    DSP6745_wait( 1000 );
                        MCASP0_GBLCTL     = 0;
                        MCASP0_RGBLCTL    = 0;
                        MCASP0_XGBLCTL    = 0;
                        MCASP0_PWRDEMU    = 1;

                        MCASP0_RMASK      = 0xFFFFFFFF;
                        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                        MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                        MCASP0_RTDM       = 0x00000003; // Slots 0,1
                        MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256
                        MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256
                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN
                        MCASP0_PFUNC      = 0x00;
                        MCASP0_PDIR          = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used

            /* Starting sections of the McASP*/
                MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                MCASP0_XSTAT = 0x0000ffff;        // Clear all
                MCASP0_RSTAT = 0x0000ffff;        // Clear all
```

```
                MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                /* Write a 0, so that no underrun occurs after releasing the
state machine */

                MCASP0_XBUF0 = 0;

                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );
                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);
                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

               while(1)
                {
                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = FIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }
}
signed int FIR_FILTER(float  h[], signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

2(c) KAISER WINDOW FOR HPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;
void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}
void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );

    /* Release SPI */
    SPI_SPIGCR0 = 1;

    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers

    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );
```

```
    SPI_SPIDAT1 = spidat1;
    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;
    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}
float filter_Coeff[] ={-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};
static short in_buffer[100];
signed int FIR_FILTER(float  h[], signed int x);
void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
          Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
          Uint32 GBLCTL_XFRST_ON       =      0X00001000;
          Uint32 GBLCTL_RFRST_ON       =      0X00000010;
      int temp,i;
      signed int input_data1, input_data2;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
                  DSP6745_wait( 1000 );
                      MCASP0_GBLCTL     = 0;
                      MCASP0_RGBLCTL    = 0;
                      MCASP0_XGBLCTL    = 0;
                      MCASP0_PWRDEMU    = 1;

                      MCASP0_RMASK      = 0xFFFFFFFF;
```

```
                        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                        MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                        MCASP0_RTDM       = 0x00000003; // Slots 0,1
                        MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256                     MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                        MCASP0_PFUNC      = 0x00;
                        MCASP0_PDIR          = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used
                                /* Starting sections of the McASP*/
                MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                MCASP0_XSTAT = 0x0000ffff;        // Clear all
                MCASP0_RSTAT = 0x0000ffff;        // Clear all

                MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
```

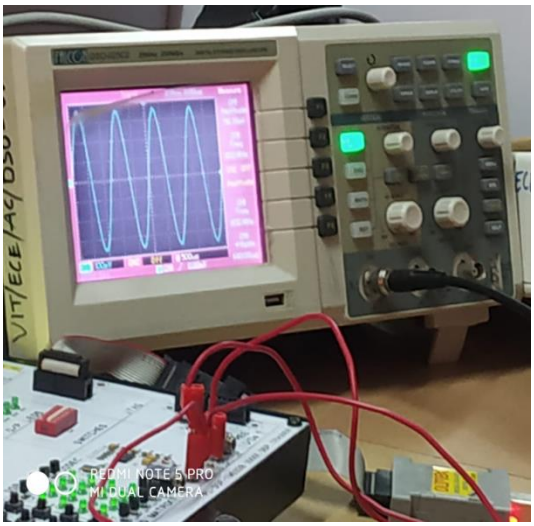
```
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );
                /* Write a 0, so that no underrun occurs after releasing the
state machine */

                MCASP0_XBUF0 = 0;
                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );
                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);
                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

               while(1)
                {
                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                         input_data1 = MCASP0_RBUF1_32BIT;
                         input_data2 = FIR_FILTER(filter_Coeff , input_data1);
                         while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                         MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }
}
signed int FIR_FILTER(float  h[], signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

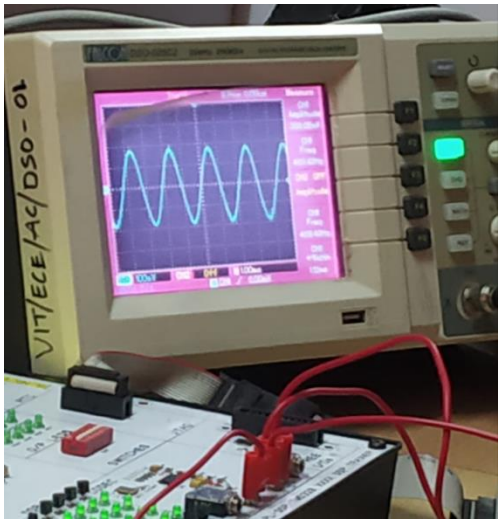
4.OUTPUT:

- As we giving applied sine input through line in, the output will appear as per the filter type on DSO.
- Output will decreasing after the cutoff frequency for low pass filter
- Output will appear at the cutoff frequency for high pass filter

LPF

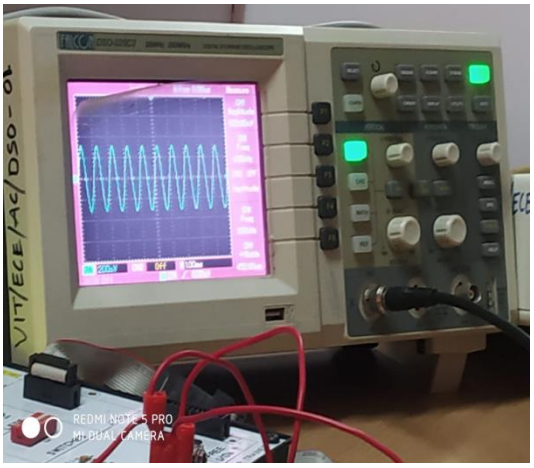
INPUT FREQUENCY (500Hz) OUTPUT

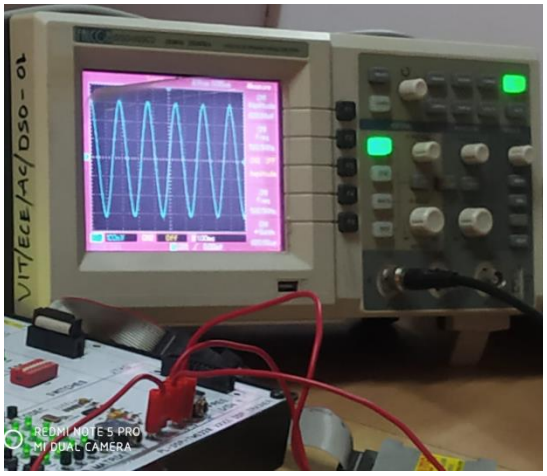




HPF

INPUT FREQUENCY (400Hz) OUTPUT





5. Result

6. Conclusion

7.Viva Questions

1. Write the procedure for design of FIR filters by using Kaiser Window?

2. Write the expression for rectangular widow and what is Peak amplitude of side lobe (db, Main lobe width and minimum stop band attenuation?

3. Write the expression for Triangular widow and what is Peak amplitude of side lobe (db, Main lobe width and minimum stop band attenuation?

4. Write the expression for Hanning widow and what is Peak amplitude of side lobe (db, Main lobe width and minimum stop band attenuation?

5. Write the expression for Hamming widow and what is Peak amplitude of side lobe (db, Main lobe width and minimum stop band attenuation?

Exp .No: 15(a) **Date:**

DESIGN OF IIR BUTTERWORTH LP/HP FILTER

1. AIM: To design and implement an IIR Butterworth LP/HP Filter for given specification.

2. APPARATUS REQUIRED

- CODE COMPOSER STUDIO
- TMS320C6745 DSP TRAINER KIT

3. Procedure

1. To create the New Project
2. Project→ New CCS Project (Give name to project with location to save or use default location)Select project type→ Executable
3. Device Family→C6000
4. Variant→C674xFloating-point DSP
5. Connection →Texas Instruments XDS100V2USB Emulator
6. Click on Empty Project then Finish
7. To create a Source file
8. File →New→ Source file (Save & give file name, Eg: sum.c).Click on Finish
9. Write the C-Program To build the program project →Build All
10. After Build Finished without errors, Now DSP kit is turned ON
11. Debug the Program after loading is done
12. Run the Program and verify the output
13. Connect CRO to the LINE OUT.
14. Connect a Signal Generator to the LINE IN.
15. Switch on the Signal Generator with a sine wave of frequency 100 Hz. and Vp-p=1.0v & vary the frequency.

1. IIR low pass Butterworth filter (cutoff 800Hz)
2. IIR high pass Butterworth filter (cutoff 2500Hz)

1. IIR Butterworth LPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char

static Uint32 spidat1;

void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}


void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );
            /* Release SPI */
    SPI_SPIGCR0 = 1;
    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
```

```
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );

    SPI_SPIDAT1 = spidat1;
    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;

    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}
const signed int filter_Coeff[] =
{
312,312,312,32767,-27943,24367 /*LP 800 */
} ;
signed int IIR_FILTER(const signed int  h[], signed int x1);

void main( )
{
             Uint32 GBLCTL_XHCLKRST_ON =      0X00000200;
           Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
           Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
           Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
           Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
           Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
           Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
           Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
           Uint32 GBLCTL_XFRST_ON       =      0X00001000;
           Uint32 GBLCTL_RFRST_ON       =      0X00000010;

      int temp,i;
      signed int input_data1, input_data2;
      //Uint16 x, y, z;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
                  DSP6745_wait( 1000 );

                        MCASP0_GBLCTL     = 0;
                    MCASP0_RGBLCTL    = 0;
                    MCASP0_XGBLCTL    = 0;
```

```
                        MCASP0_PWRDEMU    = 1;

                        MCASP0_RMASK      = 0xFFFFFFFF;
                        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                        MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                        MCASP0_RTDM       = 0x00000003; // Slots 0,1
                        MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                        MCASP0_PFUNC      = 0x00;
                        MCASP0_PDIR       = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used

            /* Starting sections of the McASP*/
                MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                MCASP0_XSTAT = 0x0000ffff;        // Clear all
                MCASP0_RSTAT = 0x0000ffff;        // Clear all
```

```
                MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                /* Write a 0, so that no underrun occurs after releasing the
state machine */

                MCASP0_XBUF0 = 0;

                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );

                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);


                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

                while(1)
                {
                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = IIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }

}
signed int IIR_FILTER(const signed int  h[], signed int x1)
{
static signed int x[6] = { 0, 0, 0, 0, 0, 0 }; /* x(n), x(n-1), x(n-2). Must be
static */
static signed int y[6] = { 0, 0, 0, 0, 0, 0 }; /* y(n), y(n-1), y(n-2). Must be
static */
int temp=0;
temp = (short int)x1; /* Copy input to temp */
x[0] = (signed int) temp; /* Copy input to x[stages][0] */
temp = ( (int)h[0] * x[0]) ; /* B0 * x(n) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
```

```
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[2] * x[2]); /* B2 * x(n-2) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[5] * y[2]); /* A2 * y(n-2) */
/* Divide temp by coefficients[A0] */
temp >>= 15;
if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
temp = -32767;
}
y[0] = temp ;
/* Shuffle values along one place for next time */
y[2] = y[1]; /* y(n-2) = y(n-1) */
y[1] = y[0]; /* y(n-1) = y(n) */
x[2] = x[1]; /* x(n-2) = x(n-1) */
x[1] = x[0]; /* x(n-1) = x(n) */
/* temp is used as input next time through */
return (temp<<2);
}
```

2. Butterworth HPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;

void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}

void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );

    /* Release SPI */
    SPI_SPIGCR0 = 1;

    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
```

```
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers

    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );

    SPI_SPIDAT1 = spidat1;

    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;

    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}

const signed int filter_Coeff[] =
{
            3403, -3403, 3403, 32767, 15565, 17595//1621,-
1621,1621,32767,20964,15649//17926, -17926, 17926, 32767, -14372, 10192//
} ;
signed int IIR_FILTER(const signed int  h[], signed int x1);

void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
          Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
```

```
            Uint32 GBLCTL_XFRST_ON       =      0X00001000;
            Uint32 GBLCTL_RFRST_ON       =      0X00000010;

      int temp,i;
      signed int input_data1, input_data2;
      //Uint16 x, y, z;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
                  DSP6745_wait( 1000 );

                        MCASP0_GBLCTL     = 0;
                  MCASP0_RGBLCTL    = 0;
                  MCASP0_XGBLCTL    = 0;
                  MCASP0_PWRDEMU    = 1;

                  MCASP0_RMASK      = 0xFFFFFFFF;
                  MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                  MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                  MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                  MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                  MCASP0_RTDM       = 0x00000003; // Slots 0,1
                  MCASP0_RINTCTL    = 0x00000000; // Not used
                  MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                  MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                  MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                  MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                  MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                  MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                  MCASP0_XTDM       = 0x00000003; // Slots 0,1
                  MCASP0_XINTCTL    = 0x00000000; // Not used
                  MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                  MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                  MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                     MCASP0_PFUNC      = 0x00;
                  MCASP0_PDIR          = 0x00000001;
```

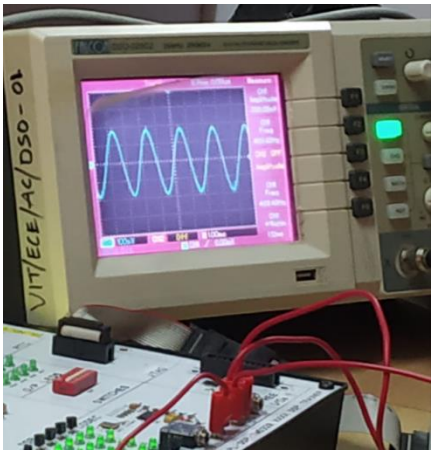
```
                        MCASP0_DITCTL      = 0x00000000; // Not used
                        MCASP0_DLBCTL      = 0x00000000; // Not used
                        MCASP0_AMUTE       = 0x00000000; // Not used

                /* Starting sections of the McASP*/
                    MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                    MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                    MCASP0_XSTAT = 0x0000ffff;        // Clear all
                    MCASP0_RSTAT = 0x0000ffff;        // Clear all

                    MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                    /* Write a 0, so that no underrun occurs after releasing the
state machine */

                    MCASP0_XBUF0 = 0;

                    MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );

                    MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                    MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);
```

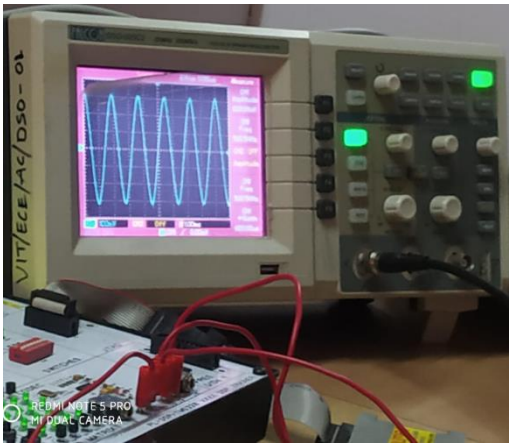
```
                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

                while(1)
                {

                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = IIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }


}

signed int IIR_FILTER(const signed int  h[], signed int x1)
{
static signed int x[6] = { 0, 0, 0, 0, 0, 0 }; /* x(n), x(n-1), x(n-2). Must be
static */
static signed int y[6] = { 0, 0, 0, 0, 0, 0 }; /* y(n), y(n-1), y(n-2). Must be
static */
int temp=0;
temp = (short int)x1; /* Copy input to temp */
x[0] = (signed int) temp; /* Copy input to x[stages][0] */
temp = ( (int)h[0] * x[0]) ; /* B0 * x(n) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[2] * x[2]); /* B2 * x(n-2) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[5] * y[2]); /* A2 * y(n-2) */
/* Divide temp by coefficients[A0] */
temp >>= 15;

if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
temp = -32767;
}
y[0] = temp ;

/* Shuffle values along one place for next time */
y[2] = y[1]; /* y(n-2) = y(n-1) */
y[1] = y[0]; /* y(n-1) = y(n) */
x[2] = x[1]; /* x(n-2) = x(n-1) */
x[1] = x[0]; /* x(n-1) = x(n) */
/* temp is used as input next time through */
return (temp<<2);

}
```

4. OUTPUT:

- As we giving applied sine input through line in, the output will appear as per the filter type on DSO.
- Output will decreasing after the cutoff frequency for low pass filter
- Output will appear at the cutoff frequency for high pass filter

INPUT FREQUENCY (800Hz) OUTPUT





5. RESULT

6. CONCLUSION

7. VIVA QUESTIONS

1. Write the design procedure of butter worth IIR filter using Bilinear Transformation Method?

2. Write the design procedure of butter worth IIR filter using Impulse Invariant Method?

3. Write relation between S & Z in BLT?

4. Write relation between S & Z in IIM?

5. Compare BLT & IIM?

6. Write the expression for order of Butterworth filter?

Exp .No: 15(b) **Date:**

DESIGN OF IIR CHEBYSHEV LP/HP FILTER

1. AIM: To design and implement IIR chebyshev LP/HP Filter for given specification.

2. APPARATUS REQUIRED

CODE COMPOSER STUDIO
TMS320C6745 DSP TRAINER KIT

3. Procedure

1. To create the New Project
2. Project→ New CCS Project (Give name to project with location to save or use default location)Select project type→ Executable
3. Device Family→C6000
4. Variant→C674xFloating-point DSP
5. Connection →Texas Instruments XDS100V2USB Emulator
6. Click on Empty Project then Finish
7. To create a Source file
8. File →New→ Source file (Save & give file name, Eg: sum.c).Click on Finish
9. Write the C-Program To build the program project →Build All
10. After Build Finished without errors, Now DSP kit is turned ON
11. Debug the Program after loading is done
12. Run the Program and verify the output
13. Connect CRO to the LINE OUT.
14. Connect a Signal Generator to the LINE IN.
15. Switch on the Signal Generator with a sine wave of frequency 100 Hz. and Vp-p=1.0v & vary the frequency.

1. Low pass Chebyshev filter (cutoff 1000Hz)
2. High pass Chebyshev filter (cutoff 1000Hz)

1. Chebyshev LPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;
void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}

void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );

    /* Release SPI */
    SPI_SPIGCR0 = 1;

    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
        //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers

    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );
```

```
    SPI_SPIDAT1 = spidat1;

    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;

    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}

const signed int filter_Coeff[] =
{
            1455,1455,1455,32767,-23140,21735 /*LP 2500 */
} ;

signed int IIR_FILTER(const signed int  h[], signed int x1);

void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
          Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
          Uint32 GBLCTL_XFRST_ON       =      0X00001000;
          Uint32 GBLCTL_RFRST_ON       =      0X00000010;
      int temp,i;
      signed int input_data1, input_data2;
      //Uint16 x, y, z;
      Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
      PINMUX7 = 0x10110000;
      PINMUX9 = 0x11011000;
      PINMUX10 = 0x00000110;
      spi_init( );
      temp = SPI_SPIDAT1;

            for(i=0; i<10; i++)
            {
                  SPI_SPIDAT1= temp | codec_reg_val[i];
                  while( !(SPI_SPIFLG&0x200));
                  DSP6745_wait( 1000 );
            }
            SPI_SPIGCR0 = 0;
                  DSP6745_wait( 1000 );

                MCASP0_GBLCTL     = 0;
                MCASP0_RGBLCTL    = 0;
                MCASP0_XGBLCTL    = 0;
                MCASP0_PWRDEMU    = 1;
```

```
                        MCASP0_RMASK      = 0xFFFFFFFF;
                        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                        MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                        MCASP0_RTDM       = 0x00000003; // Slots 0,1
                        MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                           MCASP0_PFUNC      = 0x00;
                        MCASP0_PDIR           = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used

             /* Starting sections of the McASP*/
                 MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                 while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                 MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                 while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                 MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                 while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                 MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                 while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                 MCASP0_XSTAT = 0x0000ffff;        // Clear all
                 MCASP0_RSTAT = 0x0000ffff;        // Clear all
```

```
                MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                /* Write a 0, so that no underrun occurs after releasing the
state machine */

                MCASP0_XBUF0 = 0;

                MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );

                MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);


                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

                while(1)
                {

                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = IIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);
                }


}
signed int IIR_FILTER(const signed int  h[], signed int x1)
{
static signed int x[6] = { 0, 0, 0, 0, 0, 0 }; /* x(n), x(n-1), x(n-2). Must be
static */
static signed int y[6] = { 0, 0, 0, 0, 0, 0 }; /* y(n), y(n-1), y(n-2). Must be
static */
int temp=0;
temp = (short int)x1; /* Copy input to temp */
x[0] = (signed int) temp; /* Copy input to x[stages][0] */
temp = ( (int)h[0] * x[0]) ; /* B0 * x(n) */
```

```
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[2] * x[2]); /* B2 * x(n-2) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[5] * y[2]); /* A2 * y(n-2) */
/* Divide temp by coefficients[A0] */
temp >>= 15;
if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
temp = -32767;
}
y[0] = temp ;
/* Shuffle values along one place for next time */
y[2] = y[1]; /* y(n-2) = y(n-1) */
y[1] = y[0]; /* y(n-1) = y(n) */
x[2] = x[1]; /* x(n-2) = x(n-1) */
x[1] = x[0]; /* x(n-1) = x(n) */
/* temp is used as input next time through */
return (temp<<2);
}
```

2. Chebyshev HPF

PROGRAM:

```
#include<stdio.h>
#include "sysreg.h"
#define Uint32  unsigned int
#define Uint16  unsigned short
#define Uint8   unsigned char
#define Int32   int
#define Int16   short
#define Int8    char
static Uint32 spidat1;

void DSP6745_wait( Uint32 delay )
{
    volatile Uint32 i;
    for ( i = 0 ; i < delay ; i++ )
    {

    }
}

void spi_init( )
{
    /* Reset SPI */
    SPI_SPIGCR0 = 0;
      DSP6745_wait( 1000 );

    /* Release SPI */
    SPI_SPIGCR0 = 1;

    /* SPI 4-Pin Mode setup */
    SPI_SPIGCR1 = 0
        | ( 0 << 24 )//Deactivates SPI
```

```
        | ( 0 << 16 )//Internal loop-back test mode disabled/enabled=0/1
       //| ( 1 << 1 )//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers
        | ( 3 << 0 );//MASTER MODE. SPIx_CLK is an output and the SPI initiates
transfers

    SPI_SPIPC0 = 0
        | ( 1 << 11 )   // DI
        | ( 1 << 10 )   // DO
        | ( 1 << 9 )    // CLK
        | ( 1 << 8 )    // EN0
        | ( 1 << 0 );   // CS

      spidat1 = 0
        | ( 0 << 28 )   // CSHOLD
        | ( 0 << 24 )   // DFSEL Format [0]
        | ( 1 << 26 )
        | ( 0 << 16 )   // CSNR
        | ( 0 << 0 );   //

    SPI_SPIFMT0 = 0
        | ( 0 << 20 )   // SHIFTDIR
        | ( 0 << 17 )   // Polarity
        | ( 1 << 16 )   // Phase
        | ( 14 << 8 )    // Prescale to 30MHz (150/(value+1))//29
        | ( 16 << 0 );   // Char Len | ( 1F << 0 );

    SPI_SPIDAT1 = spidat1;

    SPI_SPIDELAY = 0
        | ( 8 << 24 )   // C2TDELAY
        | ( 8 << 16 );  // T2CDELAY

    SPI_SPIDEF = 0
        | ( 1 << 1 )    // EN1 inactive high
        | ( 1 << 0 );   // EN0 inactive high

    SPI_SPIINT = 0;

    SPI_SPIDEF = 0x01;
    SPI_SPIGCR1 |= ( 1 << 24 );  //Enable SPI
}

const signed int filter_Coeff[] =
{
             11617, -11617, 11617, 32767, -8683, 15505//963, -963, 963, 32767,
25417, 23512//
} ;
signed int IIR_FILTER(const signed int  h[], signed int x1);

void main( )
{
          Uint32 GBLCTL_XHCLKRST_ON    =      0X00000200;
          Uint32 GBLCTL_RHCLKRST_ON    =      0x00000002;
          Uint32 GBLCTL_XCLKRST_ON     =      0X00000100;
          Uint32 GBLCTL_RCLKRST_ON     =      0X00000001;
          Uint32 GBLCTL_XSRCLR_ON      =      0X00000400;
          Uint32 GBLCTL_RSRCLR_ON      =      0X00000004;
          Uint32 GBLCTL_XSMRST_ON      =      0X00000800;
```

```
            Uint32 GBLCTL_RSMRST_ON      =      0X00000008;
            Uint32 GBLCTL_XFRST_ON       =      0X00001000;
            Uint32 GBLCTL_RFRST_ON       =      0X00000010;

       int temp,i;
       signed int input_data1, input_data2;
       //Uint16 x, y, z;
       Uint16 codec_reg_val[] = {0X0017, 0X0217, 0X04D8, 0X06D8, 0X0811, 0X0A00,
0X0C00, 0X0E53, 0X100C, 0x1201};
       PINMUX7 = 0x10110000;
       PINMUX9 = 0x11011000;
       PINMUX10 = 0x00000110;
       spi_init( );
       temp = SPI_SPIDAT1;

              for(i=0; i<10; i++)
              {
                     SPI_SPIDAT1= temp | codec_reg_val[i];
                     while( !(SPI_SPIFLG&0x200));
                     DSP6745_wait( 1000 );
              }
              SPI_SPIGCR0 = 0;
                     DSP6745_wait( 1000 );

                           MCASP0_GBLCTL     = 0;
                        MCASP0_RGBLCTL    = 0;
                        MCASP0_XGBLCTL    = 0;
                        MCASP0_PWRDEMU    = 1;

                        MCASP0_RMASK      = 0xFFFFFFFF;
                        MCASP0_RFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSRCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKRCTL   = 0x00000081;//0x000000AF; // Rising
INTERNAL CLK(from tx side)
                        MCASP0_AHCLKRCTL  = 0x00008000;//INT CLK (from tx side)
                        MCASP0_RTDM       = 0x00000003; // Slots 0,1
                        MCASP0_RINTCTL    = 0x00000000; // Not used
                        MCASP0_RCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_XMASK      = 0xFFFFFFFF; // No padding used
                        MCASP0_XFMT       = 0x00018078; // MSB 16bit, 0-delay, no
pad, CFGBus
                        MCASP0_AFSXCTL    = 0x00000000; // 2TDM, 1bit Rising edge
INTERNAL FS, word
                        MCASP0_ACLKXCTL   = 0x00000081;//0x000000AF; // ASYNC,
Rising INTERNAL CLK, div-by-16
                        MCASP0_AHCLKXCTL  = 0x00008000;// INT CLK, div-by-4
                        MCASP0_XTDM       = 0x00000003; // Slots 0,1
                        MCASP0_XINTCTL    = 0x00000000; // Not used
                        MCASP0_XCLKCHK    = 0x00FF0008; // 255-MAX 0-MIN, div-by-
256

                        MCASP0_SRCTL1     = 0x0002;     // MCASP0.AXR0[1] <-- DOUT
                        MCASP0_SRCTL0     = 0x0001;     // MCASP0.AXR0[0] --> DIN

                           MCASP0_PFUNC      = 0x00;
```

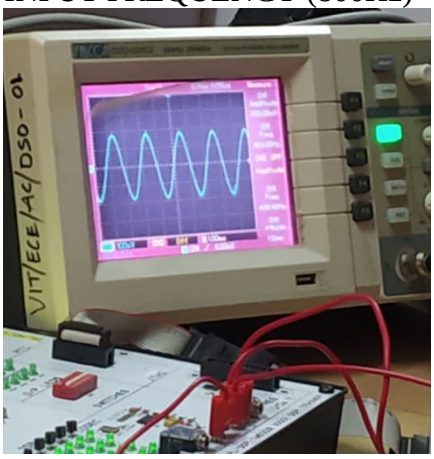
```
                        MCASP0_PDIR           = 0x00000001;
                        MCASP0_DITCTL     = 0x00000000; // Not used
                        MCASP0_DLBCTL     = 0x00000000; // Not used
                        MCASP0_AMUTE      = 0x00000000; // Not used

                /* Starting sections of the McASP*/
                    MCASP0_XGBLCTL |= GBLCTL_XHCLKRST_ON;
// HS Clk
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XHCLKRST_ON ) !=
GBLCTL_XHCLKRST_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RHCLKRST_ON;
// HS Clk
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RHCLKRST_ON ) !=
GBLCTL_RHCLKRST_ON );

                    MCASP0_XGBLCTL |= GBLCTL_XCLKRST_ON;
// Clk
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XCLKRST_ON ) !=
GBLCTL_XCLKRST_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RCLKRST_ON;
// Clk
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RCLKRST_ON ) !=
GBLCTL_RCLKRST_ON );

                    MCASP0_XSTAT = 0x0000ffff;        // Clear all
                    MCASP0_RSTAT = 0x0000ffff;        // Clear all

                    MCASP0_XGBLCTL |= GBLCTL_XSRCLR_ON;
// Serialize
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XSRCLR_ON ) !=
GBLCTL_XSRCLR_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RSRCLR_ON;
// Serialize
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RSRCLR_ON ) !=
GBLCTL_RSRCLR_ON );

                    /* Write a 0, so that no underrun occurs after releasing the
state machine */

                    MCASP0_XBUF0 = 0;

                    MCASP0_XGBLCTL |= GBLCTL_XSMRST_ON;
// State Machine
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XSMRST_ON ) !=
GBLCTL_XSMRST_ON );
                    MCASP0_RGBLCTL |= GBLCTL_RSMRST_ON;
// State Machine
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RSMRST_ON ) !=
GBLCTL_RSMRST_ON );

                    MCASP0_XGBLCTL |= GBLCTL_XFRST_ON;
// Frame Sync
                    while ( ( MCASP0_XGBLCTL & GBLCTL_XFRST_ON ) != GBLCTL_XFRST_ON
);
                    MCASP0_RGBLCTL |= GBLCTL_RFRST_ON;
// Frame Sync
                    while ( ( MCASP0_RGBLCTL & GBLCTL_RFRST_ON ) != GBLCTL_RFRST_ON
);
```

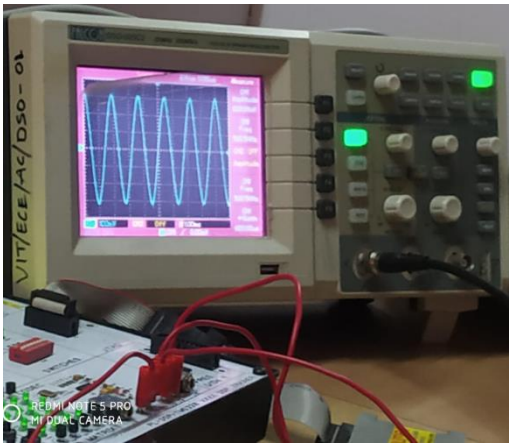
```
                /* Start by sending a dummy write */
                while( ! ( MCASP0_SRCTL0 & 0x10 ) );  // Check for Tx ready
                MCASP0_XBUF0 = 0;

                while(1)
                {

                        while ( ! ( MCASP0_SRCTL1 & 0x20 ) );
                        input_data1 = MCASP0_RBUF1_32BIT;
                        input_data2 = IIR_FILTER(filter_Coeff , input_data1);
                        while ( ! ( MCASP0_SRCTL0 & 0x10 ) );
                        MCASP0_XBUF0_32BIT = (input_data2 << 16);

                }


}

signed int IIR_FILTER(const signed int  h[], signed int x1)
{
static signed int x[6] = { 0, 0, 0, 0, 0, 0 }; /* x(n), x(n-1), x(n-2). Must be
static */
static signed int y[6] = { 0, 0, 0, 0, 0, 0 }; /* y(n), y(n-1), y(n-2). Must be
static */
int temp=0;
temp = (short int)x1; /* Copy input to temp */
x[0] = (signed int) temp; /* Copy input to x[stages][0] */
temp = ( (int)h[0] * x[0]) ; /* B0 * x(n) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[2] * x[2]); /* B2 * x(n-2) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[5] * y[2]); /* A2 * y(n-2) */
/* Divide temp by coefficients[A0] */
temp >>= 15;
if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
temp = -32767;
}
y[0] = temp ;
/* Shuffle values along one place for next time */
y[2] = y[1]; /* y(n-2) = y(n-1) */
y[1] = y[0]; /* y(n-1) = y(n) */
x[2] = x[1]; /* x(n-2) = x(n-1) */
x[1] = x[0]; /* x(n-1) = x(n) */
/* temp is used as input next time through */
return (temp<<2);
}
```

4. OUTPUT:

- As we giving applied sine input through line in, the output will appear as per the filter type on DSO.
- Output will decreasing after the cutoff frequency for low pass filter
- Output will appear at the cutoff frequency for high pass filter

INPUT FREQUENCY (800Hz) OUTPUT





5. RESULT

6. Conclusion

7.Viva Questions

1. Write the expression for the order of chebyshev filter?

2. Write the steps in designing chebyshev filter?

3. State the equation for finding the poles in chebyshev filter.

4. What is warping effect?

5. What are the properties of chebyshev filter?

PART- C
ADVANCED EXPERIMENTS

Exp .No: 16 **Date:**

SUM OF TWO SINUSOIDAL SIGNALS

1. AIM: To write a MATLAB program to find sum of two signals.

2. SOFTWARE REQUIRED:

- PC and MATLAB software

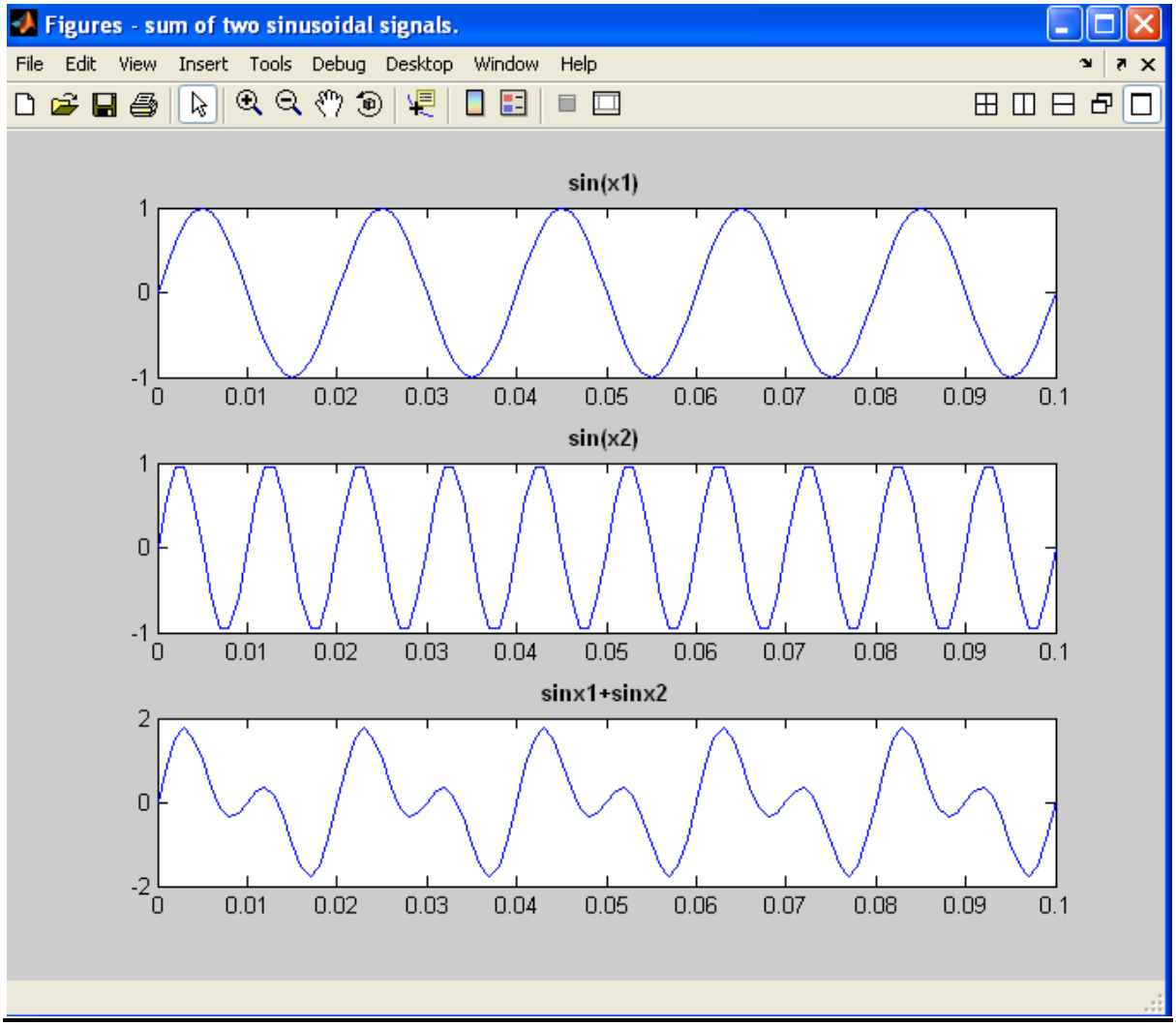
3. PROCEDURE:

- Open MATLAB
- Open new M-file
- Type the program
- Save in current directory
- Compile and Run the program
- For the output see command window\ Figure window

4. PROGRAM:

```
clc;
Clear all;
Close all;
t=0:0.001:0.1;
f1=50;
x1=2*pi*f1*t;
y1=sin(x1);
figure;
subplot (3,1,1);
plot (t,y1);
title('sin(x1');
f2=100;
x2=2*pi*f2*t;
y2=sin(x2);
subplot(3,1,2);
plot(t,y2);
title('sin(x2)');
y=y1+y2;
subplot(3,1,3);
plot(t,y);
title('sinx1=sinx2')
```

5. OUTPUT:



6. RESULT:

7. CONCLUSION

7. VIVA QUESTIONS

1. How do you find sum of sinusoid?

2. How do you add two sinusoidal currents?

3. What is amplitude of sinusoidal function?

4. What is meant by sinusoidal signal?

5. What is phase of sinusoidal function?

Exp .No: 17 **Date:**

POWER DENSITY SPECTRUM

1. AIM: To write a C- program to compute power density spectrum of given one – dimensional signal and plot.

2. APPARATUS REQUIRED:

- Code Composer Studio
- TMS320C6745 DSP Trainer Kit

3. PROCEDURE:

1. To create the New Project
2. Project→ New CCS Project (Give name to project with location to save or use default
3. location)
4. Select project type→ Executable
5. Device Family→C6000
6. Variant→C674xFloating-point DSP
7. Connection →Texas Instruments XDS100V2USB Emulator
8. Click on Empty Project then Finish
9. To create a Source file
10. File →New→ Source file (Save & give file name, Eg: sum.c).Click on Finish
11. Write the C-Program To build the program project →Build All
12. After Build Finished without errors, Now DSP kit is turned ON
13. Debug the Program after loading is done
14. Run the Program and verify the output

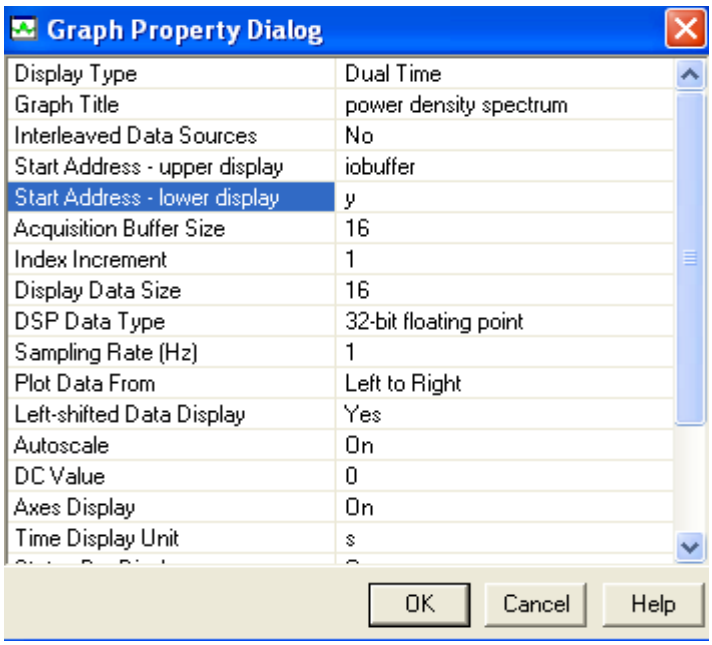
4. PROGRAM:

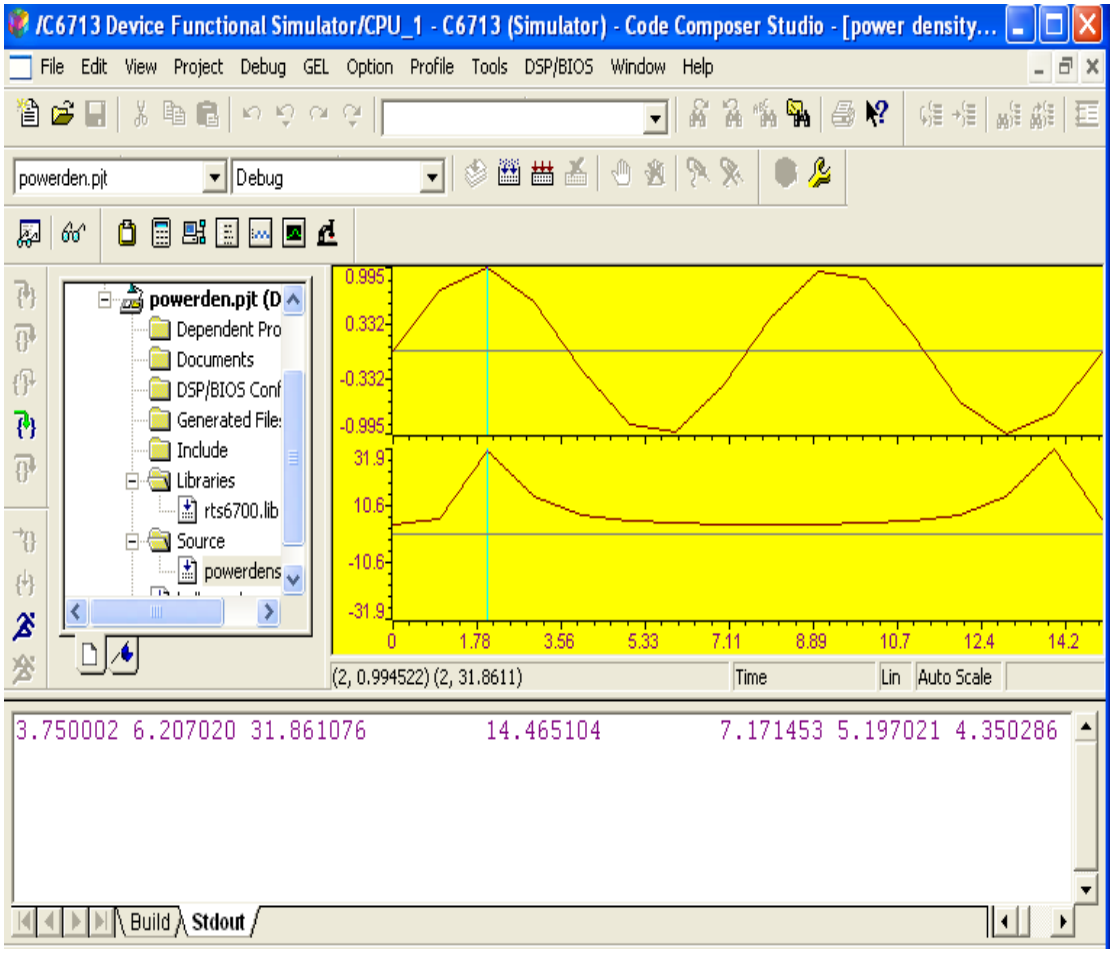
```
#include<stdio.h>
#include<math.h>
#define N 16
#define PI 3.14159
typedef struct
{
float real,imag;
}
complex;
complex x[N];
float iobuffer[N];
float x1[N],y[N];
int i;
main()
{
complex w[N];
complex temp1,temp2;
float sum=0.0;
int j,k,n,upper_leg,lower_leg,leg_diff,index,step;
```

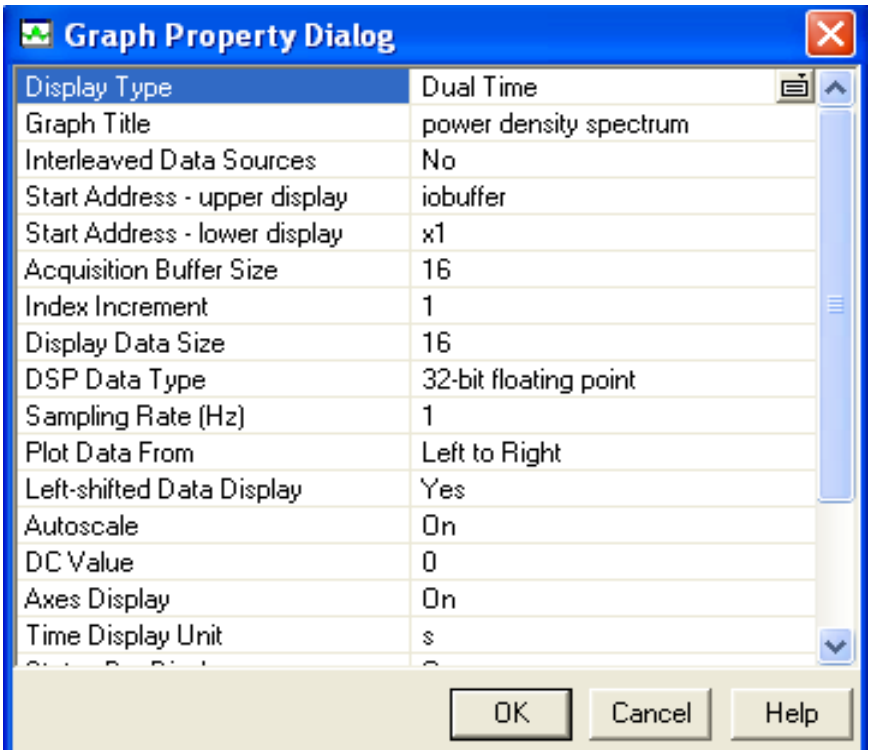
```
for(i=0;i<N;i++)
{

iobuffer[i]=sin((2*PI*2*i)/15.0);
}
for(n=0;n<N;n++)
{
sum=0;
for(k=0;k<N-n;k++)
{
sum=sum+(iobuffer[k]*iobuffer[n+k]);
}
x1[n]=sum;
}
for(i=0;i<N;i++)
{
x[i].real=x1[i];
x[i].imag=0.0;
}
for(i=0;i<N;i++)
{
w[i].real=cos((2*PI*i)/(N*2.0));
w[i].imag=-sin((2*PI*i)/(N*2.0));
}
leg_diff=N/2;
step=2;
for(i=0;i<4;i++)
{
index=0;
for(j=0;j<leg_diff;j++)
{
for(upper_leg=j;upper_leg<N;upper_leg+=(2*leg_diff))
{
lower_leg=upper_leg+leg_diff;
temp1.real=(x[upper_leg]).real+(x[lower_leg]).real;
temp1.imag=(x[upper_leg]).imag+(x[lower_leg]).imag;
temp2.real=(x[upper_leg]).real-(x[lower_leg]).real;
temp2.imag=(x[upper_leg]).imag-(x[lower_leg]).imag;
(x[lower_leg]).real=temp2.real*(w[index]).real-temp2.imag*(w[index]).imag;
(x[lower_leg]).imag=temp2.real*(w[index]).imag+temp2.imag*(w[index]).real;
(x[upper_leg]).real=temp1.real;
(x[upper_leg]).imag=temp1.imag;
}
index+=step;
}

leg_diff=(leg_diff)/2;
step=step*2;
}
j=0;
```

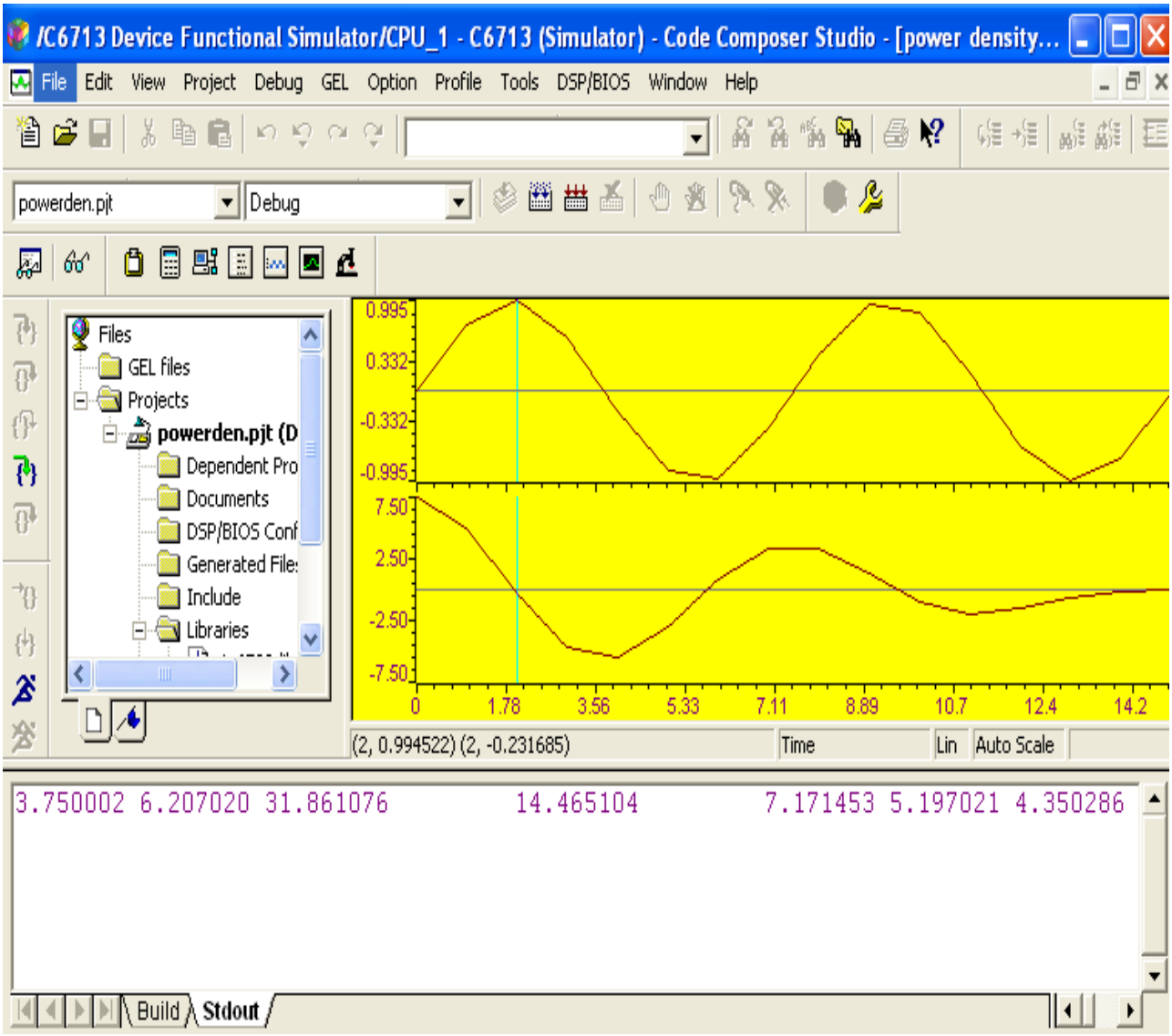
```
for(i=1;i<(N-1);i++)
{
k=N/2;
while(k<=j)
{
j=j-k;
k=k/2;
}
j=j+k;
if(i<j)
{
temp1.real=(x[j]).real;
temp1.imag=(x[j]).imag;
(x[j]).real=(x[i]).real;
(x[j]).imag=(x[i]).imag;
(x[i]).real=temp1.real;
(x[i]).imag=temp1.imag;
}
}
for(i=0;i<N;i++)
{
y[i]=sqrt((x[i].real*x[i].real)+(x[i].imag*x[i].imag));
}
for(i=0;i<N;i++)
{
printf("%f\t",y[i]);
}
return(0);
}
```

5. OUTPUT:




/C6713 Device Functional Simulator/CPU_1 - C6713 (Simulator) - Code Composer Studio - [power density...
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
powerden.pjt
Debug
powerden.pjt (D
Dependent Pro
Documents
DSP/BIOS Conf
Generated File
Include
Libraries
rts6700.lib
Source
powerdens
0.995
0.332
-0.332
-0.995
31.9
10.6
-10.6
-31.9
0 1.78 3.56 5.33 7.11 8.89 10.7 12.4 14.2
(2, 0.994522) (2, 31.8611)
Time
Lin
Auto Scale
3.750002 6.207020 31.861076 14.465104 7.171453 5.197021 4.350286
Build
Stdout


Graph Property Dialog
Display Type Dual Time
Graph Title power density spectrum
Interleaved Data Sources No
Start Address - upper display iobuffer
Start Address - lower display x1
Acquisition Buffer Size 16
Index Increment 1
Display Data Size 16
DSP Data Type 32-bit floating point
Sampling Rate (Hz) 1
Plot Data From Left to Right
Left-shifted Data Display Yes
Autoscale On
DC Value 0
Axes Display On
Time Display Unit s
OK
Cancel
Help



6. RESULT:

7. Conclusion

8. Viva Questions

1. What do you mean by phase spectrum and magnitude spectrum give comparison?

2. What is power spectral density used for?

3. How Power Spectral Density is used to Classify Noise?

4. What is Energy Spectral Density?

5. Which has the same power spectral density?