



Sri Eshwar
College of Engineering
Coimbatore | Tamilnadu
An Autonomous Institution
Affiliated to Anna University, Chennai



Kondampatti (Post), Vadasithur (Via), Coimbatore - 641 202.

Department of

.....
Record work of Laboratory

Certified bonafide Record of work done by

Name : Roll No. :

Class : Branch :

Reg. No. :

Place :

HOD

Faculty In charge

Date :

University Register No.

Submitted for the University Practical Examination held on

.....

Internal Examiner

External Examiner

Instructions for LABORATORY Classes

- ❖ Enter the Lab with observation, record notebook, calculator & necessary tools
- ❖ Enter the Lab with **CLOSED FOOTWEAR**
- ❖ Maintain Silence during the Lab Hours
- ❖ Boys should "**TUCK IN**" the Shirts
- ❖ Girls are instructed to wear **OVERCOATS**
- ❖ **HANGING CHAINS, RINGS, WRIST WATCHES** and like items are likely to cause accidents and hence are to be **AVOIDED**
- ❖ **LONG HAIR** should be protected, let it not be loose especially near **ROTATING MACHINERY**
- ❖ Any machine / equipment **SHOULD NOT BE OPERATED** other than the prescribed one for the day
- ❖ Read and follow the work instructions inside the laboratory
- ❖ **POWER SUPPLY** to your table should be obtained only through the **LAB TECHNICIAN**
- ❖ Do not **LEAN** and do not be **CLOSE** to the rotating components
- ❖ Handle the equipments with care
- ❖ **TOOLS & GAUGE** sets are to be **RETURNED** before leaving the lab
- ❖ **ALL SUB - HEADING** and **DETAILS** should be neatly written on the right hand side page of the record
- ❖ For example
 - Aim of the experiment
 - Apparatus / Tools / Instruments required
 - Procedure
 - Model calculation
 - Results / discussions / inference
- ❖ The sketches, block diagrams, circuits and flow charts are to be drawn on the left side blank page
- ❖ For example
 - Neat Diagram
 - Specifications / Design Details
 - Tabulations
 - Graph
- ❖ **GRAPHS / FIGURES** drawn on separate sheets in special cases, should be firmly pasted on to the record.
- ❖ Before writing the record, the student should get the corresponding observations signed by the **FACULTY - IN- CHARGE**
- ❖ The observation and record should be completed in all respects and submitted in the very **NEXT CLASS** itself.
- ❖ Begin every experiment on a **NEW PAGE**
- ❖ Write the **NAME OF EXPERIMENT** in Capital letters on the top of the page
- ❖ Experiment number with date should be written at the top left hand corner
- ❖ Be **PATIENT, STEADY, SYSTEMATIC** and **REGULAR**
- ❖ Follow the dress code for Laboratory classes
- ❖ Maintain punctuality

Record Index

S.No	Date	Name of the Experiment	Page No.	Program (25)	Procedure (20)	Inference (20)	Viva (10)	Total (75)	Faculty Sign
Average									

Signature of the Faculty member

VISION & MISSION OF THE DEPARTMENT

Vision:

“To emerge as a pioneering department, nurturing students into futuristic and globally competent Computer and Communication Engineering professionals.”

Mission:

M1: To offer high-quality engineering education emphasizing practical outcomes.

M2: To foster innovative research and project development among students and instil a sense of social responsibility.

M3: To equip students with industry-relevant skills and robust leadership qualities through pedagogical methodologies and industry engagement.

M4: To inculcate a culture of socially relevant values and nurture moral, ethical and interpersonal skills among students.

Program Educational Objectives (PEOs):

PEO1: Acquire core competencies in computational technologies communication engineering with the ability to utilise the latest software tools to effectively address real-time challenges.

PEO2: Active involvement in research, innovation, or entrepreneurial ventures, resulting in significant societal contributions.

PEO3: Demonstrate industry-relevant skills, effective leadership qualities, and strong ethical values to exhibit professionalism and foster collaborative work experience.

Program Outcomes:

PROGRAM OUTCOMES (POs)		Graduates of Computer and Communication Engineering will be able to
PO1.	Engineering knowledge	Apply knowledge of mathematics, natural science, computing, engineering fundamentals and an engineering specialization to develop the solution of complex engineering problems.
PO2.	Problem analysis	Identify, formulate, review research literature and analyze complex engineering problems reaching substantiated conclusions with consideration for sustainable development.
PO3.	Design/development of solutions	Design creative solutions for complex engineering problems and design/develop systems/components/processes to meet identified needs with consideration for public health and safety, whole-life cost, net zero carbon, culture, society, and environment as required.
PO4.	Conduct investigations of complex problems	Conduct investigations of complex engineering problems using research-based knowledge including design of experiments, modelling, analysis & interpretation of data to provide valid conclusions.
PO5.	Engineering Tool Usage	Create, select and apply appropriate techniques, resources and modern engineering & IT tools, including prediction and modelling recognizing their limitations to solve complex engineering problems.
PO6.	The Engineer and The World	Analyze and evaluate societal and environmental aspects while solving complex engineering problems for its impact on sustainability with reference to economy, health, safety, legal framework, culture, and environment.
PO7.	Ethics	Apply ethical principles and commit to professional ethics, human values, diversity and inclusion; adhere to national & international laws.

PO8.	Individual and Collaborative Teamwork	Function effectively as an individual, and as a member or leader in diverse/multi-disciplinary teams.
PO9.	Communication	Communicate effectively and inclusively within the engineering community and society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations considering cultural, language, and learning differences.
PO10.	Project management and finance	Apply knowledge and understanding of engineering management principles and economic decision-making and apply these to one's own work, as a member and leader in a team, and to manage projects and in multidisciplinary environments.
PO11.	Life-long learning	Recognize the need for, and have the preparation and ability for i) independent and life-long learning ii) adaptability to new and emerging technologies and iii) critical thinking in the broadest context of technological change.

Program Specific Outcomes (PSO):

PSO1: Apply the concepts of computer and communication systems to develop software for different applications.

PSO2: Acquire skills and knowledge to solve real-time challenges in communication networks.

Course Outcomes COs:

CO. No.	Course Outcome	BTL	POs	PSOs
U23CC352.1	Apply the principles of linear and circular convolution to solve given discrete-time signal processing problems.	K3	1,2,3,4,5,9	2
U23CC352.2	Examine DFT and FFT algorithms to compute the frequency spectrum of discrete signals.	K3	1,2,3,4,5,9	2
U23CC352.3	Implement Butterworth and Chebyshev IIR filters for various types of using digital signal processing techniques.	K3	1,2,3,4,5,9	2
U23CC352.4	Implement FIR filters using digital signal processing techniques.	K3	1,2,3,4,5,9	2
U23CC352.5	Study and Implement MAC operation, IIR and FIR filter design using Digital Signal Processor.	K3	1,2,3,4,5,9	2

CO & PO/PSO Mapping:

CO	PO 01	PO 02	PO 03	PO 04	PO 05	PO 06	PO 07	PO 08	PO 09	PO 10	PO 11	PO 12	PSO 01	PSO 02
U23CC352.1	3	2	2	2	3	-	-	-	1	-	-	-	-	2
U23CC352.2	3	2	2	2	3	-	-	-	1	-	-	-	-	2
U23CC352.3	3	2	2	2	3	-	-	-	1	-	-	-	-	2
U23CC352.4	3	2	2	2	3	-	-	-	1	-	-	-	-	2
U23CC352.5	3	2	2	2	3	-	-	-	1	-	-	-	-	2
Course to PO	3	2	2	2	3	-	-	-	1	-	-	-	-	2

“3”—High, “2”—Medium, “1”—Low, “-”—No Correlation

Syllabus:**U23CC352 – DISCRETE TIME SIGNAL PROCESSING LABORATORY****EXPERIMENTS USING MATLAB / EQUIVALENT SOFTWARE PACKAGE**

1. Compute the Linear and Circular convolution of the any two sequences.
2. Compare the computational complexity of DFT and FFT algorithm.
3. Design and demonstrate the filtering operation of Butterworth IIR filters (LPF/HPF/BPF/BSF).
4. Design and demonstrate the filtering operation of Chebyshev IIR filters (LPF/HPF/BPF/BSF).
5. Design and demonstrate the filtering operation of FIR filters (LPF/HPF/BPF/BSF) using windowing techniques.

EXPERIMENTS USING DIGITAL SIGNAL PROCESSOR

6. Study of architecture of Digital Signal Processor (TMS320C67XX).
7. Perform MAC operation using various addressing modes.
8. Generate various signals and random noise.
9. Implementation of Butter worth and Chebyshev IIR Filters for Low pass, High pass, Band pass and Band stop filtering.
10. Implementation of FIR Filter for Low pass, High pass, Band pass and Band stop Filtering DSP.

Ex.No : 01

LINEAR AND CIRCULAR CONVOLUTION

Date :

AIM:

To write a program to perform the linear and circular convolution of two sequences.

APPARATUS REQUIRED:

Computer with MATLAB / OCTAVE

ALGORITHM:

LINEAR CONVOLUTION:

1. Get the number of samples.
2. Generate the output sequence of the given two input signals using 'conv' command.
3. Plot the graph.

CIRCULAR CONVOLUTION

1. Get the number of samples.
2. Generate the sequence for the given two input signals using 'zeros' and 'ones' matrix command.
3. Generate the convoluted output sequence of the given two input signal using 'conv' command.
4. Plot the graph.

PROGRAM

LINEAR CONVOLUTION

```
clc;
x=input('enter 1st seq');
h=input('enter 2nd seq');
y=conv(x,h);
figure;
subplot(3,1,1);
stem(x);
ylabel('amplitude');
xlabel('a n--->');
subplot(3,1,2);
stem(h);
ylabel('amplitude');
xlabel('b n--->');
subplot(3,1,3);
stem(y);
ylabel('amplitude');
xlabel('c n--->');
disp('the result is');y
```

CIRCULAR CONVOLUTION

```
clc;
clearall;
x=[1 1 2 1 1];
h=[1 1 2 1];
Nx=length(x);
Nh=length(h);
N=max(Nx,Nh);
y=cconv(x,h,N);
n=0:1:Nx-1;
subplot(2,2,1);
stem(n,x);
xlabel('n');
ylabel('x(n)');
title('input sequence');
n=0:1:Nh-1;
subplot(2,2,2);
stem(n,h);
xlabel('n');
ylabel('h(n)');
title('impulse sequence');
n=0:1:N-1;
subplot(2,2,3);
stem(n,y);
title('circular convolution');
```

OUTPUT

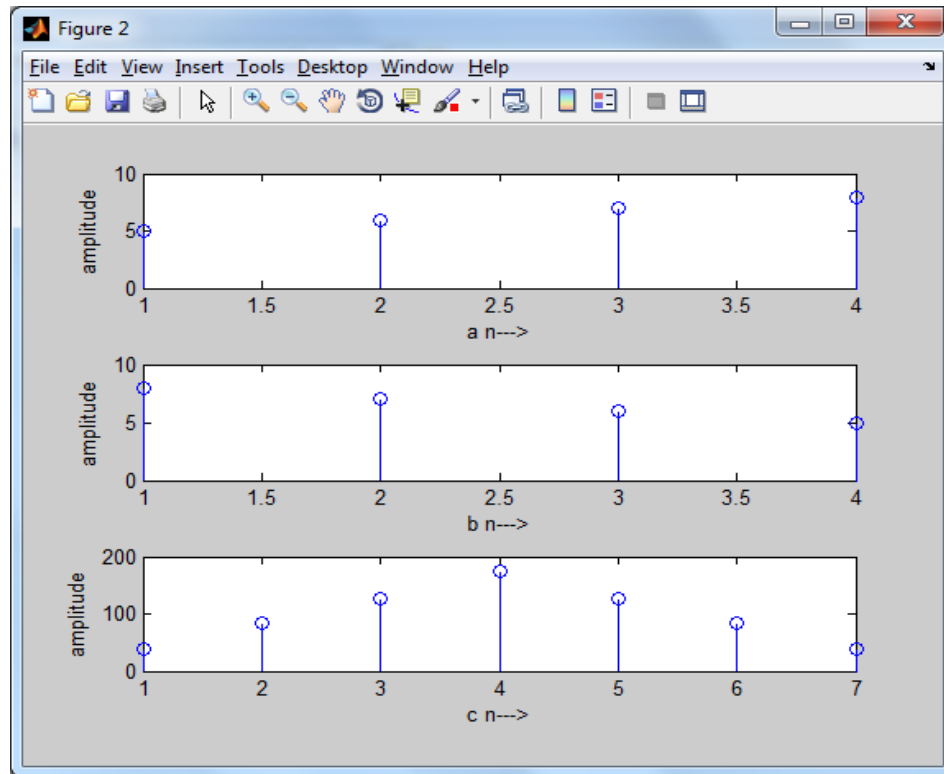
LINEAR CONVOLUTION

enter 1st seq [5 6 7 8]

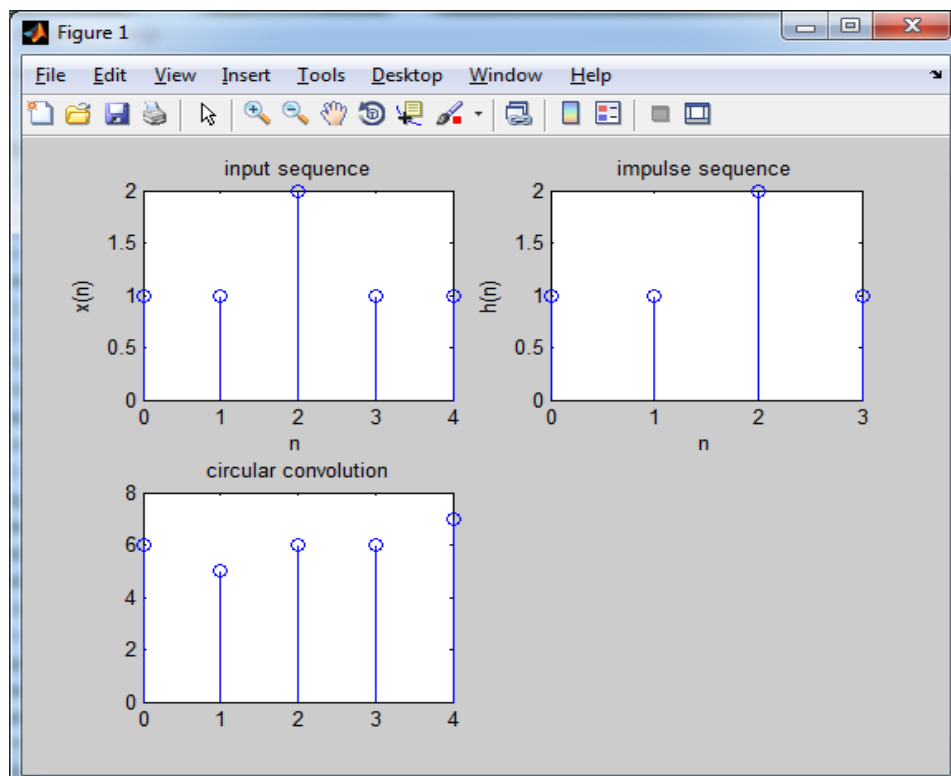
enter 2nd seq [8 7 6 5]

the result is

y = 40 83 128 174 128 83 40



CIRCULAR CONVOLUTION



Ex.No : 02

CALCULATION OF FFT

Date :

AIM:

To write a program for the calculation of FFT.

APPARATUS REQUIRED:

Computer with MATLAB / OCTAVE

ALGORITHM:

1. Get the input sequence and its length.
2. Calculate the Fast Fourier Transform and Inverse Fast Fourier Transform of the given sequence.

PROGRAM

```
clc;
clear all;
% Fast fourier transform
disp('Fast fourier transform');
l=input('enter the length ');
x=input('enter the input sequence');
y=fft(x,l);
disp('resultant FFT sequence');y
subplot(2,2,1);
stem(x);
title('input');xlabel('n--?'); ylabel('x(n)');
subplot(2,2,2);
stem(y);
title('FFT sequence'); xlabel('frequency'); ylabel('X(k)');

%Inverse Fast fourier transform
disp('inverse fast fourier transform');
x1=input('enter the FFT sequence');
y1=ifft(x1);
disp('resultant IFFT sequence');y1
subplot(2,2,3);
stem(x1);
title('FFT sequence'); xlabel('frequency'); ylabel('X(k)');
subplot(2,2,4);
stem(y1);
title('IFFT sequence'); xlabel('n--?'); ylabel('x(n)');
figure(2);
freqz(y);
title('fft sequence');
figure(3);
freqz(y1);
title('ifft sequence');
```

OUTPUT

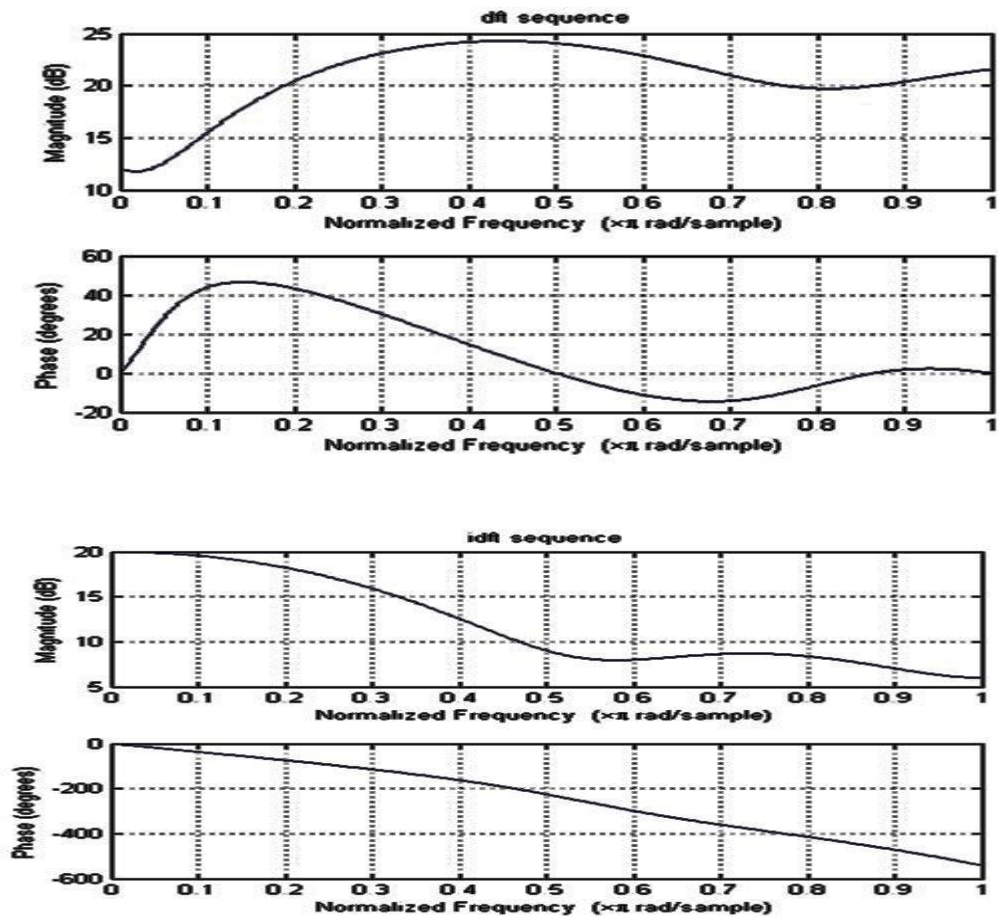
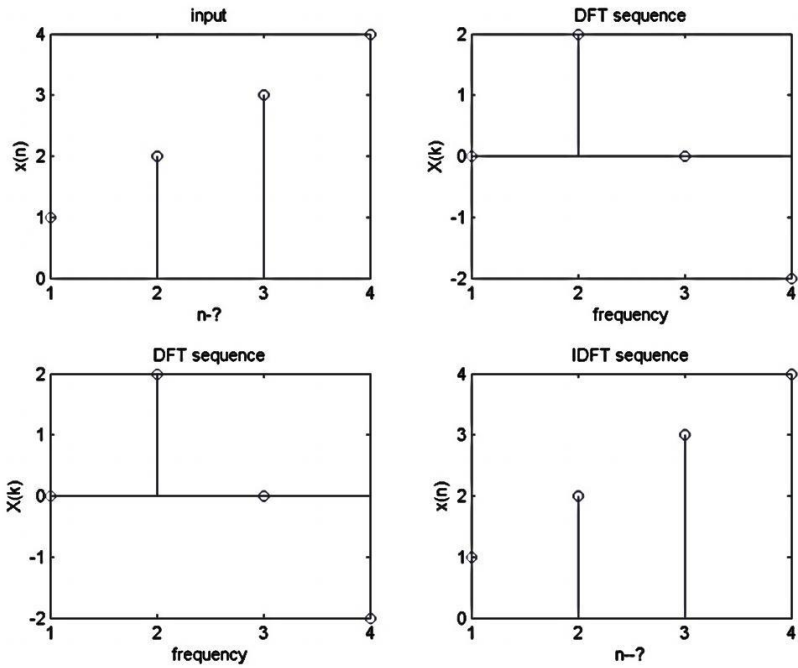
Length of the sequence = 4

Input Sequence for FFT = [1, 2, 3, 4]

FFT Output Sequence = [10, -2 + 2i, -2, 2 - 2i]

Input Sequence for IFFT = [10, -2 + 2i, -2, 2 - 2i]

IFFT Output Sequence = [1, 2, 3, 4]



Ex.No: 03

DESIGN OF BUTTERWORTH IIR FILTER

Date:

AIM:

To write a program to design a Butterworth IIR filter.

APPARATUS REQUIRED:

Computer with MATLAB / OCTAVE

ALGORITHM:

1. Get the pass band and stop band normalized frequency.
2. Get the pass band and stop band attenuation level.
3. Plot the magnitude and phase response graph.

PROGRAM

BUTTERWORTH LOW PASS FILTER

```
clc;
close all;
clear all;
format long
rp=input('enter the passband ripple (rp)=');
rs=input('enter the stopband ripple (rs)=');
wp=input('enter the passbandfreq (wp)=');
ws=input('enter the stopbandfreq (ws)=');
fs=input('enter the sampling freq (fs)=');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs);
[b,a]=butter(n,wn);
[bz,az]=impinvar(b,a,fs);
disp('bz = ');
disp(bz);
disp('az = ');
disp(az);
Hz=tf(bz,az);
w=0:.01:pi;
[h,om]=freqz(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 -->');
```

BUTTERWORTH BAND PASS FILTER

```
clc;
close all;
clear all;
format long
rp=input('enter the passband ripple (rp)=');
rs=input('enter the stopband ripple (rs)=');
wp=input('enter the passbandfreq (wp)=');
ws=input('enter the stopbandfreq (ws)=');
fs=input('enter the sampling freq (fs)=');
w1=2*wp/fs;
w2=2*ws/fs;

[n]=buttord(w1,w2,rp,rs);
wn=[w1,w2];
[b,a]=butter(n,wn,'bandpass');
[bz,az]=impinvar(b,a,fs);
disp('bz = ');
disp(bz);
disp('az = ');
disp(az);
Hz=tf(bz,az);
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure(1);
subplot(2,1,1);
plot(om/pi,m);
ylabel('Gain in deciBel');
xlabel('Normalized frequency in rad/sec');
title('Magnitude Plot (BPF)');
subplot(2,1,2);
plot(om/pi,an);
xlabel('Normalized frequency in rad/sec');
ylabel('Phase Angle in rad');
title('Phase Plot (BPF)');
```

BUTTERWORTH LOW PASS FILTER

Pass band ripple=0.5

Stop band ripple=50

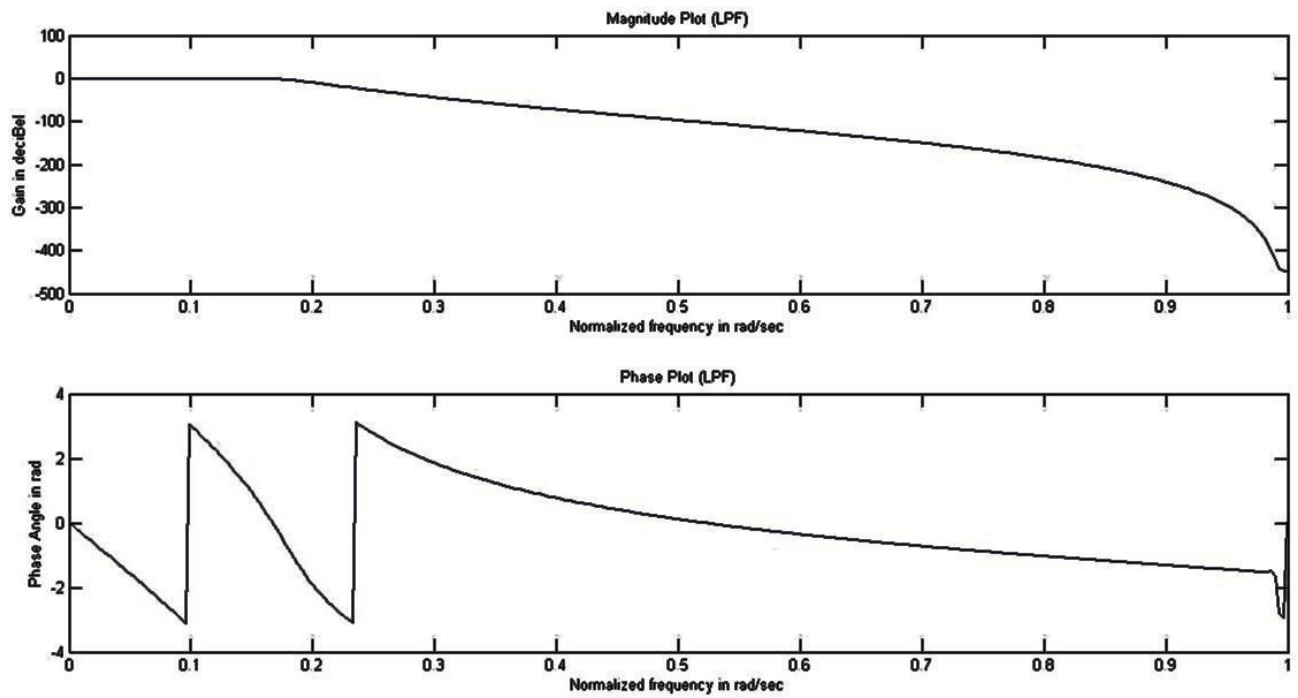
Pass band freq=1200

Stop band freq=2400

Sampling freq=10000

Filter order = 9

Cut-off frequency= 0.179



BUTTERWORTH BAND PASS FILTER

Pass band ripple=0.25

Stop band ripple=20

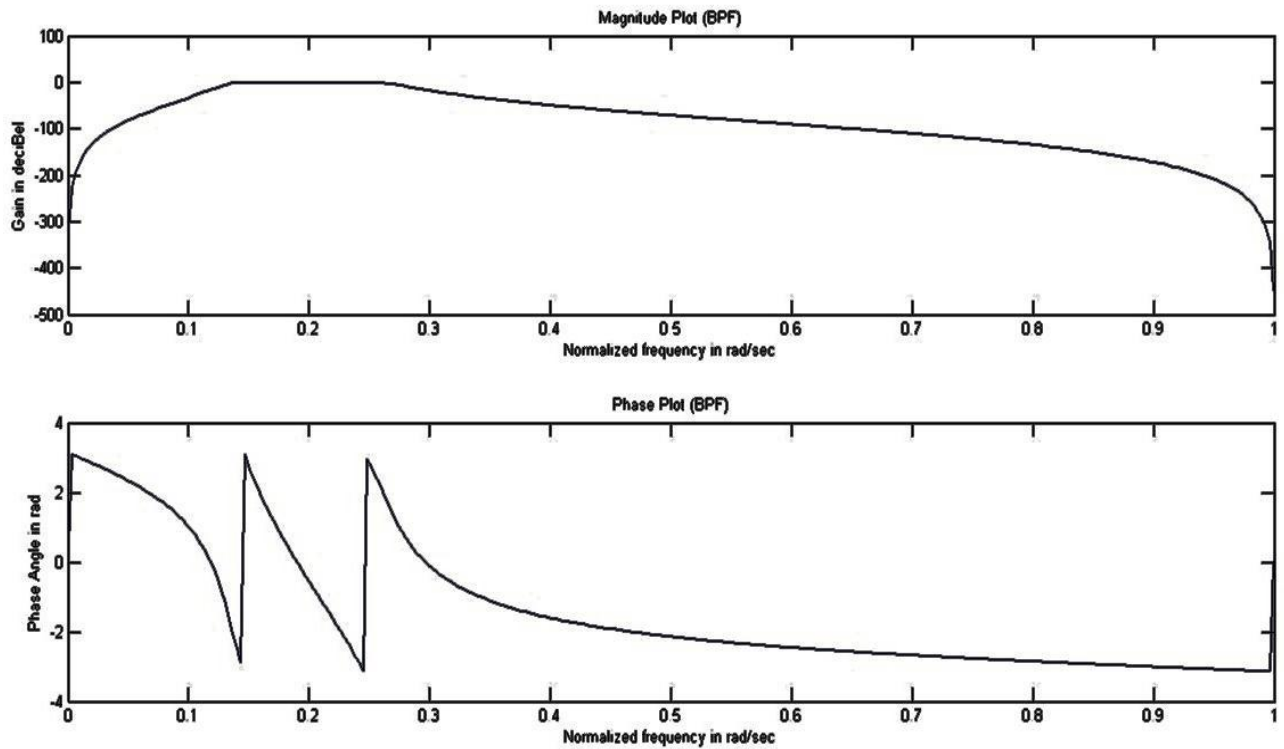
Pass band freq1000

Stop band freq2000

Sampling freq15000

Filter order = 6

Cut-off frequency= [0.133, 0.26]



Ex.No: 04

DESIGN OF CHEBYSHEV IIR FILTER

Date:

AIM:

To write a program to design a Chebyshev IIR filter.

APPARATUS REQUIRED:

Computer with MATLAB / OCTAVE

ALGORITHM:

1. Get the pass band and stop band normalized frequency.
2. Get the pass band and stop band attenuation level.
3. Plot the magnitude and phase response graph.

PROGRAM:

```
clear all;
clc;
close all;
rp=input('Enter the pass band ripple:');
rs=input('Enter the stop band ripple:');
wp=input('Enter the pass band frequency:');
ws=input('Enter the stop band frequency:');
fs=input('Enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
```

%LOW PASS FILTER

```
[n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rp,wn);
[bz,az]=impinvar(b,a,fs);
disp('bz = ');
disp(bz);
disp('az = ');
disp(az);
Hz=tf(bz,az);
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
disp('Filter order');
disp(n);
disp('Cut-off frequency');
disp(wn);
subplot(2,1,1);
plot(om/pi,m);
ylabel('Gain in db--->');
xlabel('(a)Normalized frequency--->');
```

```

title('Low pass filter');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b)Normalized frequency--->');
ylabel('Phase in radians--->');

```

%BAND PASS FILTER

```

clear all;
clc;
close all;
rp=input('Enter the pass band ripple:');
rs=input('Enter the stop band ripple:');
wp=input('Enter the pass band frequency:');
ws=input('Enter the stop band frequency:');
fs=input('Enter the sampling frequency:');
w1=2*wp/fs;
w2=2*ws/fs;
[n,wn]=cheb1ord(w1,w2,rp,rs);
[b,a]=cheby1(n,rS,wn,'bandpass');
[bz,az]=impinvar(b,a,fs);
disp('bz = ');
disp(bz);
disp('az = ');
disp(az);
Hz=tf(bz,az);
w=0:0.01:pi;
[h,om]=freqz(b,a,w);
m=20*log10(abs(h));
an=angle(h);
disp('Filter order');
disp(n);
disp('Cut-off frequency');
disp(wn);
subplot(2,1,1);
plot(om/pi,m);
ylabel('Gain in db--->');
xlabel('(a)Normalized frequency--->');
title('Low pass filter');
subplot(2,1,2);
plot(om/pi,an);
xlabel('(b)Normalized frequency--->');

```

CHEBYSHEV LOW PASS FILTER

Passband ripple=0.5

Stopband ripple=50

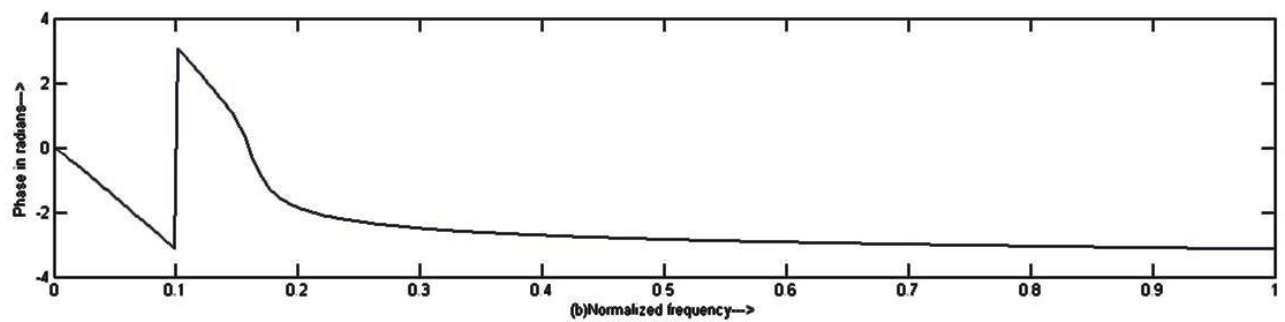
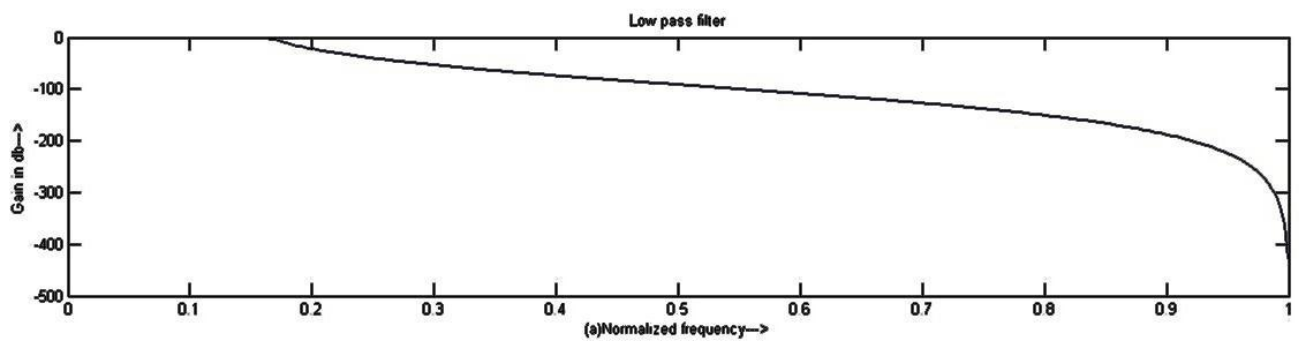
Passband freq=1200

Stop band freq=2400

Sampling freq=15000

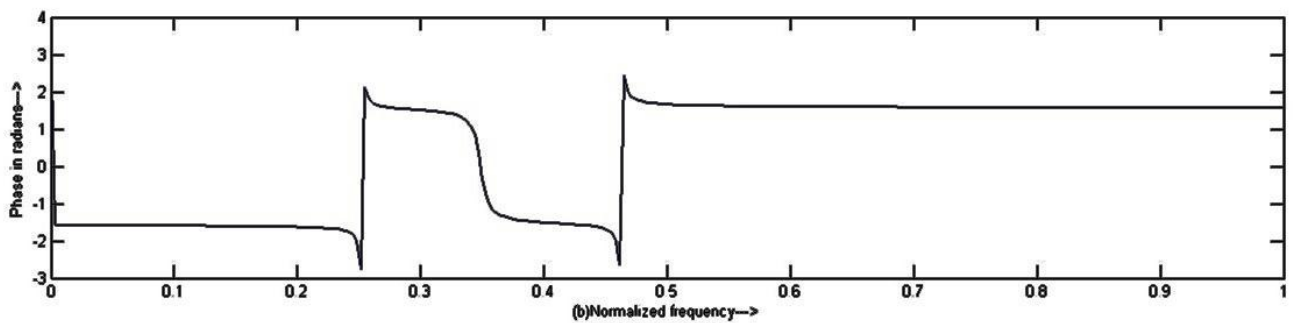
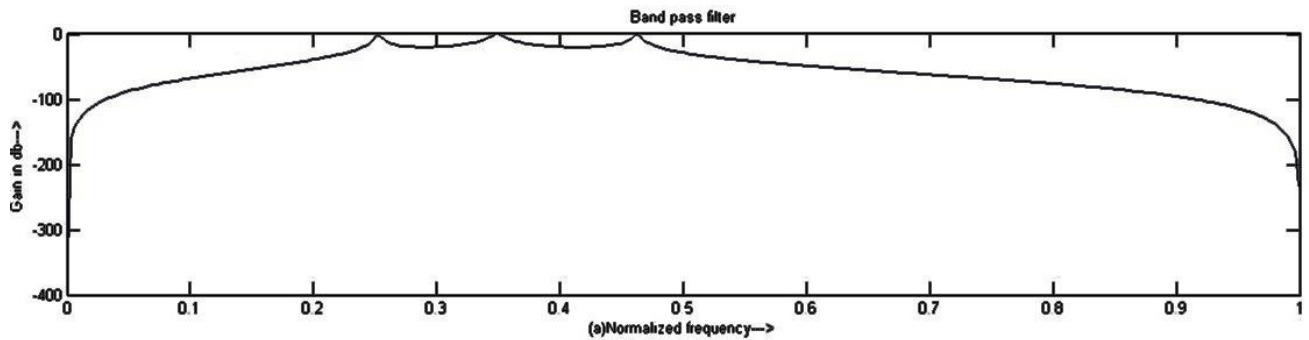
Order of the filter = 6

Cut off frequency= 0.16



CHEBYSHEV BAND PASS FILTER

Passband ripple=0.25
Stopband ripple=20
Passband freq=1200
Stop band freq=2400
Sampling freq=15000
Order of the filter = 3
Cut off frequency= [0.24,0.48]



Ex.No: 05

DESIGN OF FIR FILTER

Date:

AIM:

To write a program to design a FIR (Low pass, High pass, Band pass, Band Stop) filter.

APPARATUS REQUIRED:

Computer with MATLAB / OCTAVE

ALGORITHM:

1. Get the cut off frequency and the order of the filter.
2. Design the Low pass filter using various window techniques.
3. Plot the magnitude response graph.

PROGRAM FOR RECTANGULAR WINDOW

```
clear all;
clc;
disp('design of fir low pass filter using window function');
n=input(' enter number of sample = ');
w=input(' enter the cut of frequency = ');
n=n-1;
%Low Pass Filter
disp('LPF using rectangular window');
lpf=fir1(n,w/pi,rectwin(n+1));
[h,w1]=freqz(lpf,1,256);
m=20*log(abs(h));
subplot(2,2,1);
plot(w1/pi,m);
ylabel('gain in dB');
xlabel('normalized freq');
title('low pass filter');
title('LPF using rectangular window');

% High Pass filter
disp('HPF using rectangular window');
hpf=fir1(n,w/pi,'high',rectwin(n+1));
[h1,w2]=freqz(hpf,1,256);
m1=20*log(abs(h1));
subplot(2,2,2);
plot(w2/pi,m1);
ylabel('gain in dB');
xlabel('normalized freq');
title('High pass filter');
title('HPF using rectangular window');
```

% Band Pass filter

```
wc1=input(' Enter the first cut of frequency = ');
wc2=input(' Enter the second cut of frequency= ');
wn=[wc1 wc2];
disp('BPF using rectangular window');
bpf=fir1(n,wn/pi,'bandpass',rectwin(n+1));
[h2,w3]=freqz(bpf,1,256);
m2=20*log(abs(h2));
subplot(2,2,3);
plot(w3/pi,m2);
ylabel('gain in dB');
xlabel('normalized freq');
title('Band pass filter');
title('BPF using rectangular window');
```

% Band stop filter

```
wc3=input(' Enter the first cut of frequency = ');
wc4=input(' Enter the second cut of frequency = ');
wn1=[wc3 wc4];
disp('BSF using rectangular window');
bsf=fir1(n,wn1/pi,'stop',rectwin(n+1));
[h3,w4]=freqz(bsf,1,256);
m3=20*log(abs(h3));
subplot(2,2,4);
plot(w4/pi,m3);
ylabel('gain in dB');
xlabel('normalized freq');
title('Band Stop filter');title('BSF using rectangular window');
```

PROGRAM FOR HAMMING WINDOW

```
clear all;
clc;
disp('design of fir low pass filter using window function');
n=input('enter number of sample=');
w=input('enter the cut of frequency=');
n=n-1;
```

%Low Pass Filter

```
disp('LPF using Hamming window');
lpf=fir1(n,w/pi,hamming(n+1));
[h,w1]=freqz(lpf,1,256);
m=20*log(abs(h));
subplot(2,2,1);
plot(w1/pi,m);
ylabel('gain in dB');
xlabel('normalized freq');
title('low pass filter');
title('LPF using Hamming window');
```

% High Pass Filter

```
disp('HPF using Hamming window');
hpf=fir1(n,w/pi,'high',hamming(n+1));
[h1,w2]=freqz(hpf,1,256);
m1=20*log(abs(h1));
subplot(2,2,2);
plot(w2/pi,m1);
ylabel('gain in dB');
xlabel('normalized freq');
title('High pass filter');
title('HPF using Hamming window');
```

%Band Pass Filter

```
wc1=input('enter the first cut of frequency');
wc2=input('enter the second cut of frequency');
wn=[wc1 wc2];
disp('Bpf using Hamming window');
bpf=fir1(n,wn/pi,'bandpass',Hamming(n+1));
[h2,w3]=freqz(bpf,1,256);
m2=20*log(abs(h2));
subplot(2,2,3);
plot(w3/pi,m2);
ylabel('gain in dB');
xlabel('normalized freq');
title('Band pass filter');
title('BPF using Hamming window');
```

%Band Stop Filter

```
wc3=input('enter the first cut of frequency');
wc4=input('enter the second cut of frequency');
wn1=[wc3 wc4];
disp('BSF using Hamming window');
bsf=fir1(n,wn1/pi,'stop',Hamming(n+1));
[h3,w4]=freqz(bsf,1,256);
m3=20*log(abs(h3));
subplot(2,2,4);
plot(w4/pi,m3);
ylabel('gain in dB');
xlabel('normalized freq');
title('Stop pass filter');title('BSF using Hamming window');
```

PROGRAM FOR HANNING WINDOW

```
clear all;
clc;
disp('design of fir low pass filter using window function');
n=input('enter number of sample=');
w=input('enter the cut of frequency=');
n=n-1;
```

%Low Pass Filter

```
disp('LPF using Hanning window');
lpf=fir1(n,w/pi,hann(n+1));
[h,w1]=freqz(lpf,1,256);
m=20*log(abs(h));
subplot(2,2,1);
plot(w1/pi,m);
ylabel('gain in dB');
xlabel('normalized freq');
title('low pass filter');
title('LPF using Hanning window');
```

%High Pass Filter

```
disp('HPF using Hanning window');
hpf=fir1(n,w/pi,'high',hann(n+1));
[h1,w2]=freqz(hpf,1,256);
m1=20*log(abs(h1));
subplot(2,2,2);
plot(w2/pi,m1);
ylabel('gain in dB');
xlabel('normalized freq');
title('High pass filter');
title('HPF using Hanning window');
```

%Band Pass Filter

```
wc1=input('enter the first cut of frequency=');
wc2=input('enter the second cut of frequency=');
wn=[wc1 wc2];
disp('BPF using Hanning window');
bpf=fir1(n,wn/pi,'bandpass',hann(n+1));
[h2,w3]=freqz(bpf,1,256);
m2=20*log(abs(h2));
subplot(2,2,3);
plot(w3/pi,m2);
ylabel('gain in dB');
xlabel('normalized freq');
title('Band pass filter');
title('BPF using Hanning window');
```

%Band Stop Filter

```
wc3=input('enter the first cut of frequency=');
wc4=input('enter the second cut of frequency=');
wn1=[wc3 wc4];
disp('BSF using Hanning window');
bsf=fir1(n,wn1/pi,'stop',hann(n+1));
[h3,w4]=freqz(bsf,1,256);
m3=20*log(abs(h3));
subplot(2,2,4);
plot(w4/pi,m3);
ylabel('gain in dB');
xlabel('normalized freq');
title('Stop pass filter');title('BSF using Hanning window');
```

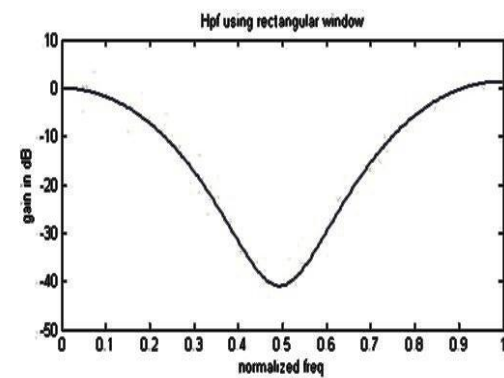
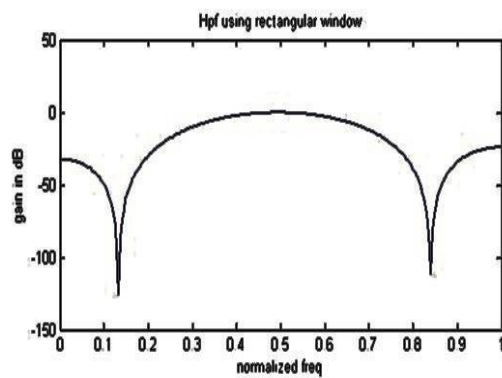
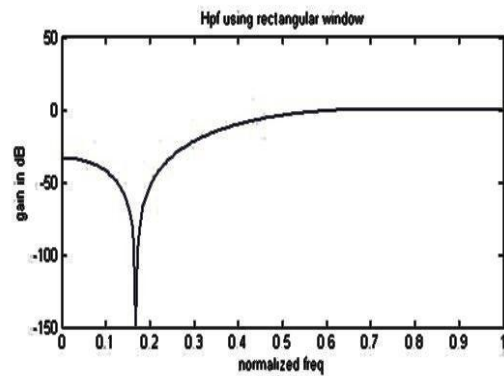
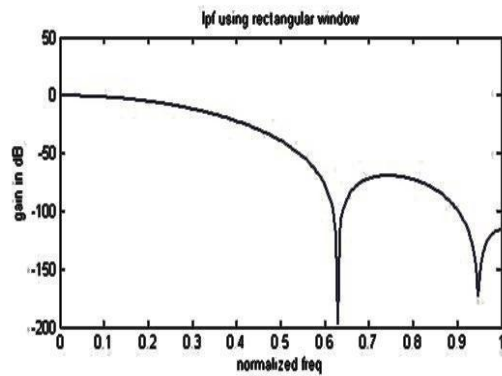
OUTPUT

RECTANGULAR WINDOW:

Number of Samples = 5

Cut off Frequency $W_c = 1.2$ (for LPF and HPF)

Cut off Frequency $W_{c1} = 1$ and $W_{c2} = 2$ (for BPF and BSF)



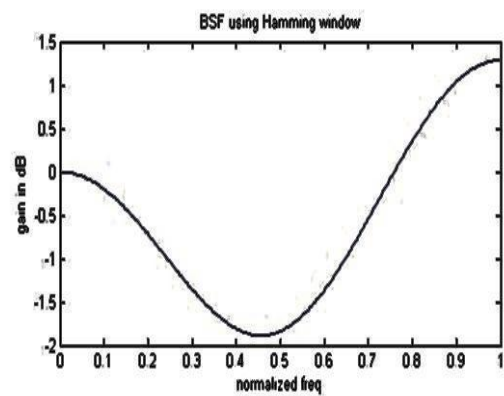
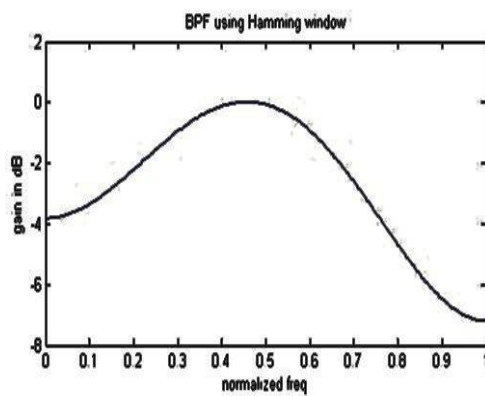
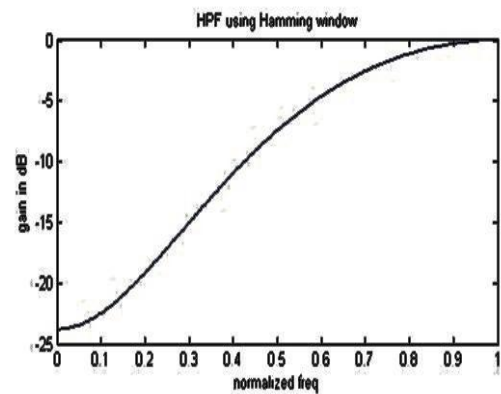
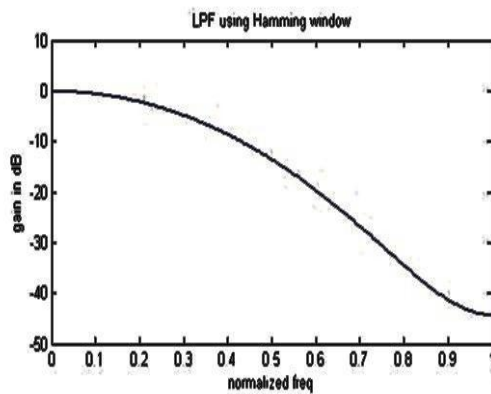
OUTPUT

HAMMING WINDOW:

Number of Samples = 5

Cut off Frequency $W_c = 1.2$ (for LPF and HPF)

Cut off Frequency $W_{c1} = 1$ and $W_{c2} = 2$ (for BPF and BSF)



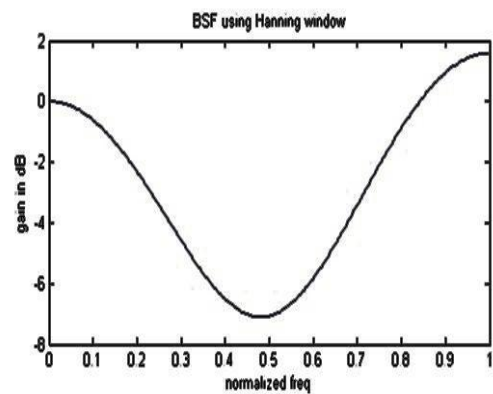
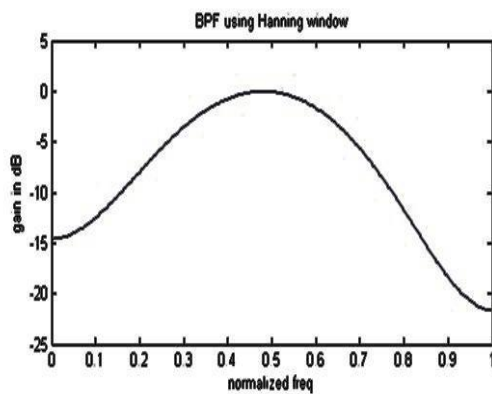
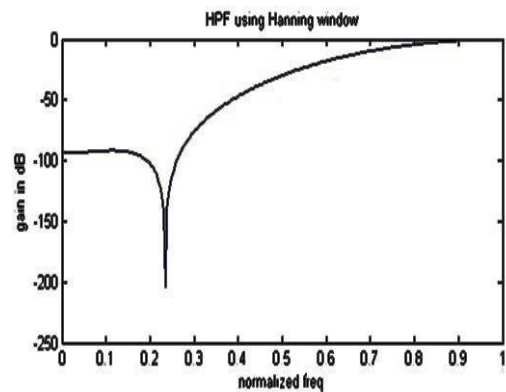
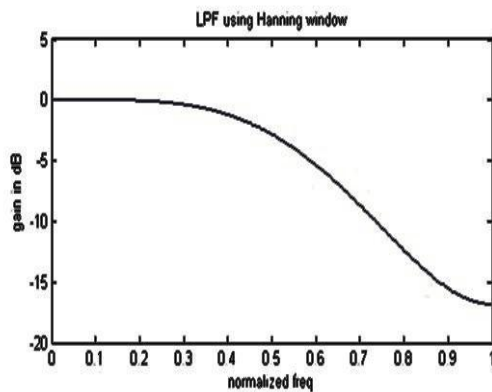
OUTPUT

HANNING WINDOW:

Number of Samples = 5

Cut off Frequency $W_c = 1.2$ (for LPF and HPF)

Cut off Frequency $W_{c1} = 1$ and $W_{c2} = 2$ (for BPF and BSF)



Date:**AIM:**

To study about the architecture of DSP Processor TMS320C6x.

THE DSP CHIP TMS320C6x:

The TMS320C6x devices are the first devices to feature VelociTI0, an advanced very long instruction word (VLIW) architecture developed by Texas Instruments, which allows performance of up to 1600 million instructions per second (MIPS). The first device in the series is the TMS320C6201, a fixed-point digital signal processor (DSP).

With a complete set of development tools, the 'C6x devices offer cost-effective solutions to high-performance DSP programming challenges. The 'C6x development tools include a new C compiler, an assembly optimizer, and a Windows-based debugger. VelociTI combines an advanced VLIW architecture with a high degree of parallelism to produce a device that enables applications such as: Unlimited Internet bandwidth, Universal wireless communications, new telephony features, Remote medical diagnostics, Automated cruise control, Personal home base station, Personalized home security. The 'C6x devices also can be used for improved performance on existing applications, such as: Wireless base stations, Pooled modems and remote access servers, Next-generation xDSL modems and cable modems, Multichannel telephony platforms, including central office switches, PBXs, and voice-messaging systems, Multimedia systems, Digital imaging.

The TMS320C6201 is the first fixed-point processor in the 'C6x generation. Following are key features of the TMS320C6201: VelociTI advanced very long instruction word (VLIW) architecture, Load/store architecture, Instruction packing for reduced code size, 100% conditional instructions for faster execution Intuitive, reduced instruction set computing (RISC)-like instruction set.

Central Processing Unit (CPU)

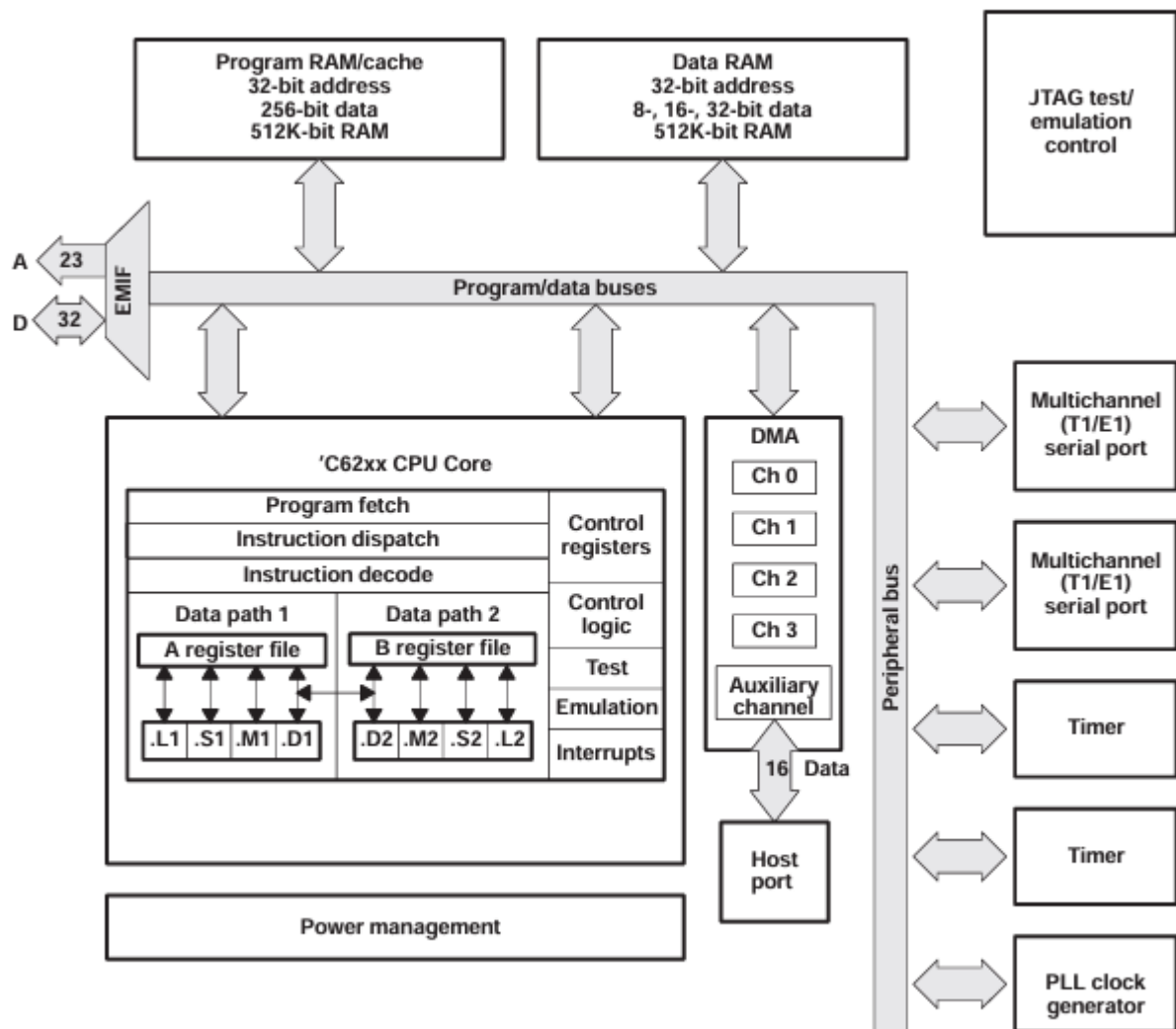
Eight independent functional units (including two 16-bit multipliers with 32-bit results and six arithmetic logic units [ALUs] with 32-/40-bit results), 32 32-bit registers, 1600 million instructions per second (MIPS), Five-ns cycle time, Up to eight 32-bit instructions per cycle, Byte-addressable 8-, 16-, 32-bit data, 32-bit address range, Dual-endian support, Saturation, Normalization, Bit-field instructions (extract, clear, left most bit detection).

The 'C6x central processing unit (CPU) is the central building block of all the TMS320C62xx devices. The CPU contains: Program fetch unit Instruction dispatch unit Instruction decode unit 32 general-purpose, 32-bit registers Two data paths, each with four functional units, including one multiplier and three arithmetic logic units (ALUs) on each data path Control registers Control logic Test, emulation, and interrupt logic The CPU has two data paths where processing occurs. Each data path has four functional units and a register file containing 16 32-bit registers. The functional units execute logic, shifting, multiply, and data address operations. All instructions operate on the registers. The two sets of data-addressing units are exclusively responsible for all data transfers between the register files and the memory.

Addressing Modes:

The addressing mode options on the 'C62xx are either linear or circular, as specified by the addressing-mode register (AMR).

FUNCTIONL BLOCK DIAGRAM:



Interrupts:

The CPU has 14 interrupts. These are the reset interrupt, the nonmaskable interrupt (NMI), and interrupts 4±15. These interrupts correspond to the RESET, NMI, and INT4±INT15 signals on the CPU boundary. In some 'C62xx devices, these signals may be tied directly to pins on the device, connected to on-chip peripherals, or may be disabled permanently by being tied inactive on chip. Generally, RESET and NMI are connected directly to pins on the device.

Memory/peripherals:

Synchronous external memory interface (EMIF), Two multichannel serial ports (MCSPs), Four-channel direct memory access (DMA), Two timers, X4 phase-locked-loop (PLL) option, Host-port interface (HPI), 1M-bit on-chip memory (divided into 2K by 256 bits of program memory and 64K bytes of data memory), 352-pin ball-grid array package.

Internal Memory:

The internal memory consists of 512K bits of on-chip program/cache memory and 512K bits of on-chip data memory. The program memory, configurable as cache or program, is organized in 2K of 256-bit fetch packets. The 'C6201 fetches all instructions one fetch packet at a time. The packets are processed at the maximum rate of one packet (eight 32-bit instructions) per CPU cycle or at a minimum of one instruction per cycle. The internal data memory is byte addressable by the CPU (for reads as well as writes) and supports bytes, half words, and full word transfers.

Data-Memory System

The TMS320C62xx data-memory system includes SRAM and a memory controller. The CPU can access data memory in 8-bit byte, 16-bit halfword, and 32-bit word-lengths. The data memory system supports two memory accesses per cycle. These accesses can be any combination of loads and stores from the two data buses of the CPU. Similarly, a simultaneous internal and external memory access is supported by the data memory system. The TMS320C62xx data memory system also supports direct-memory access (DMA) and external host accesses.

Program-Memory System

The TMS320C62xx program-memory system includes on-chip SRAM and a memory/cache controller. The program memory can operate as either an internal program memory or as a directly mapped program cache. There are four modes under which the program memory system operates: Program-memory mode, Cache-enable mode, Cache-freeze mode, Cache-bypass mode. The DMA can write data into an addressed space of program memory. The DMA cannot read from the internal program memory in program memory mode.

Peripherals:

In addition to on-chip memory, the TMS320C6201 contains the following peripherals:

External memory interface (EMIF)

Direct-memory access (DMA)

controller Host-port interface (HPI)

Power-down logic

Two multichannel serial ports (MCSPs)

Two 32-bit timers

External Memory Interface (EMIF)

All external data accesses by the CPU or DMA pass through the external memory interface (EMIF). The EMIF is the interface between the CPU and external memory such as synchronous dynamic random-access memory (SDRAM), synchronous-burst static RAM (SBSRAM), and asynchronous memory. The EMIF also provides 8-bit and 16-bit wide memory read capability to support low-cost boot ROM memories (flash, EEPROM, EPROM, and PROM). The interface is programmable to adapt to a variety of setup, hold, and strobe widths for asynchronous devices. SBSRAM supports zero-wait-state external access once bursts have begun. In all these types of access, the EMIF supports 8-bit, 16-bit, and 32-bit addressability for writes. All reads are performed as 32-bit transfers

Direct-Memory Access (DMA)

The on-chip DMA offers four independent, programmable channels that can be configured to transfer information from one location in the memory map to another without interfering with the operation of the CPU. This allows interfacing to slow external memories and peripherals without reducing the throughput to the CPU. The DMA controller contains its own address generators, source and destination registers, and transfer counter. The DMA has its own bus for addresses and data to keep the data transfers between memory and peripherals from conflicting with the CPU.

A DMA operation consists of a 32-bit word transfer to or from any of the three 'C62xx modules:

- Internal data memory
- Internal program memory that is not configured as cache as a destination of a transfer
- EMIF

One of the DMA channels can be used by the processor during the boot load startup procedure to initialize the internal program memory after reset. The DMA channels can be used to write to internal program memory. The boot loader uses the DMA to boot load code from off-chip memory to the internal program memory area. An external pin (sampled at reset) selects whether this boot load is performed. The serial port can also be used for booting. The DMA controller can access all internal program memory, all internal data memory, and all devices mapped to the EMIF. However, the DMA cannot use program memory as the source of a transfer, and it cannot access memories configured as cache or memory-mapped on-chip peripheral registers.

Host-Port Interface (HPI)

The HPI is a parallel port that can access the CPU memory space directly as an asynchronous interface. A host (external) processor can read from and write to the internal data memory through the 16-bit-wide access of the HPI. The HPI can boot load the CPU as well as access the full range of the 'C6201 memory. Also, the HPI offers improved performance and can operate without impacting CPU performance

Power-Down Logic

The 'C62xx supports three power-down modes (Idle1, 2, and 3) that can reduce system power requirements significantly. Idle1 halts the CPU except for the interrupt logic. Idle2 halts the CPU and the peripherals (except for the interrupt logic). Idle3 halts the phase-locked loop (PLL), stopping the clock tree from switching, which effectively halts the entire chip. Idle 3 requires a reset to wake up the device, while the other two modes can be restored using an interrupt or reset.

Multichannel Serial Port (MCSP)

The 'C6201 includes two MCSPs, supporting multivendor interface protocol (MVIP) and timers to allow easy algorithm integration. The MCSP is based on the standard TMS320C2x/C5x/C54x serial-port interface. In addition, it can buffer serial samples in memory automatically with the aid of the DMA. It also has multichannel capability compatible with the T1, E1, and MVIP standards. The MCSP provides:

- Full-duplex communication
- Double-buffered data registers
- Direct interface to other devices
- Clock generation or an internal programmable frequency shift clock
- Multichannel transmit and receive

Timers

The device has two 32-bit general purpose timers that can be used to:

- Time events
- Count events
- Generate pulses
- Interrupt the CPU
- Send synchronization events to the DMA

The timer has two signaling modes and can be clocked by an internal or an external source. The timer has an I/O pin that functions as an input clock, as an output clock, or as a general-purpose I/O pin

Ex.No: 7

STUDY OF ADDRESSING MODES

Date:

AIM:

To study about direct, indirect and immediate addressing modes in TMS320C6x debugger.

APPARATUS REQUIRED:

- 1.System with TMS320C60 debugger software
- 2.TMS320C6x Kit.

ALGORITHM:

DIRECT ADDRESSING MODE:

1. Initialize data pointer with 100H data.
2. Load the accumulator with first data, whose address is specified in the instruction.
- 3.Add the accumulator content with second data, whose address is specified in the instruction.
4. Store the accumulator content in specified address location.

IN-DIRECT ADDRESSING MODE:

1. Load the auxiliary register with address location of first data.
2. The auxiliary register (AR0) is modified indirectly as # symbol.
3. Load the second data into accumulator and perform addition operation.
- 4.Store the result.

PROGRAM

ADDITION PROGRAM (DIRECT METHOD)

.MMREGS
.TEXT

start:

```
LDP      #100H
LACC     #0004H,0
SACL     0000,0
LT       0000
add      #0005H
```

SACL	0002
SACH	0003

hlt:

b	hlt
---	-----

ADDITION PROGRAM (INDIRECT METHOD)

.MMREGS
.TEXT

start:

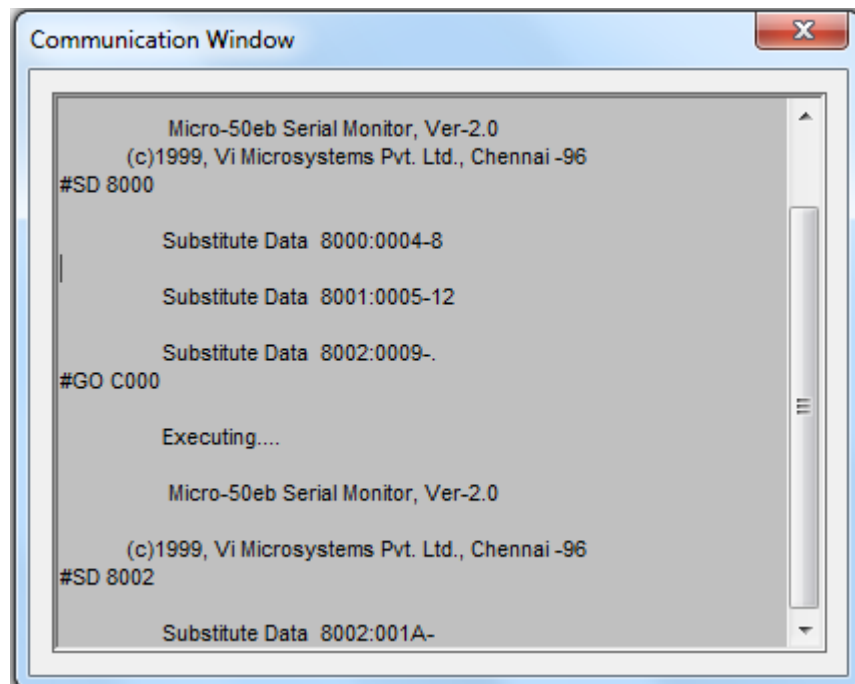
ldp	#100h
nop	
nop	
lacc	0h
add	1h
sac1	2h
sac1	3h

hlt:

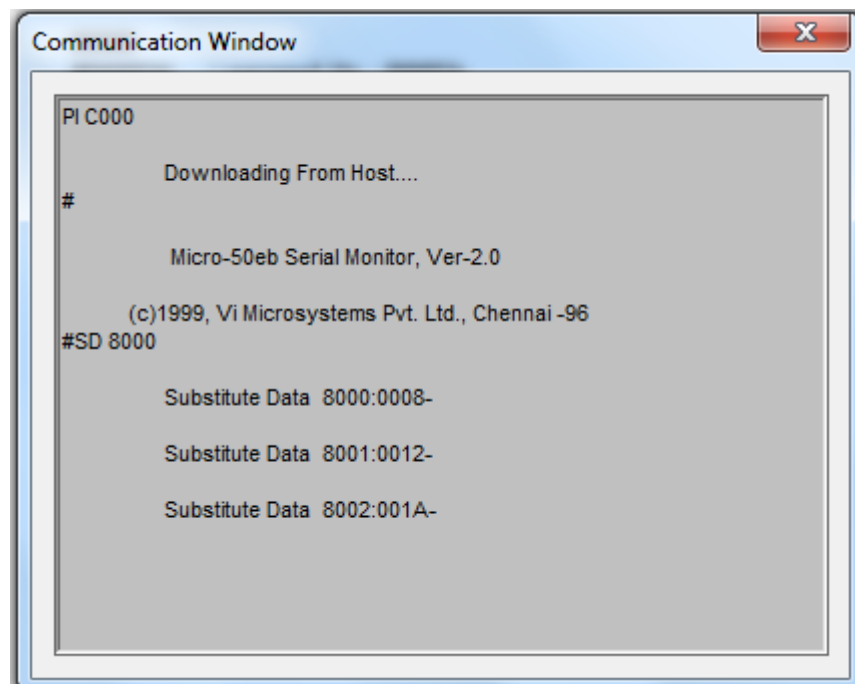
b	hlt
---	-----

OUTPUT

ADDITION PROGRAM (INDIRECT METHOD)



ADDITION PROGRAM (DIRECT METHOD)



Date:

AIM:

To write a program to generate Triangle Wave,
Square Wave and Saw Tooth Wave using TMS320C6x debugger.

APPARATUS REQUIRED:

- 1.System with TMS 320C6x debugger software
- 2.TMS 320C6x Kit.
- 3.CR0
- 4.Function Generator.

ALGORITHM:

TRIANGLE WAVE GENERATION:

1. Start the program.
2. Load the LSBs of the address data memory location.
3. Write full 16 bit pattern into address data memory location.
4. Load the content of address data memory location.
5. Add the content of address data memory location.
6. Store the data of accumulator into address data memory location.
7. Modify the current AR as specified.
8. Repeat the step 2,3.
9. Subtract the contents of address data memory location.
10. Repeat the steps 5,6,7.
11. Stop the program.

SQUARE WAVE GENERATION:

1. Start the program
2. Load the content of address data memory location.
3. Store the 16 LSBs of the accumulator into the address data memory location.
4. Load the content of address data memory location.
5. Complement the contents of the accumulator.
6. Stop the program.

SAWTOOTH WAVE GENERATION:

1. Start the program.
2. Load the LSBs of the address data memory location
3. Load the content of address data memory location into accumulator.
4. Store the 16 LSBs of the accumulator into the address data memory location
5. Write 16 bit value from from a data memory location to the specified I/O Port.
6. Add the content of address data memory location.
7. Subtract the content of the register.
8. Branch the program memory address if the specified conditions are met.
9. Stop the program.

PROGRAM: TRIANGULAR WAVE FORM

```
AMPLITUDE .SET 4
FREQ .SET 350
TEMP .SET ;
    .mmregs
    .text
START:
LDP      #100H
splk    #0,TEMP
CONT1:
    lar   AR2,#FREQ
CONT:
    out   TEMP,4
LACC     TEMP
ADD      #AMPLITUDE
SACL     TEMP
MAR      *,AR2
BANZ     CONT,*-
LAR      AR2,#FREQ
CONTx:
OUT       TEMP,4
LACC      TEMP
SUB       #AMPLITUDE
SACL      TEMP
MAR       *,AR2
BANZ      CONTx
B         CONT1      .end
```

SQUARE WAVE FORM

```
    .MMREGS
    .TEXT
START:
    LDP   #100H
    LACC  #0FFFH
LOOP:
    SACL  0
    RPT   #0FFH
    OUT   0,04
    CMPL
    B     LOOP
    .END
    LDP   #120H
    LACC  #0H
    SACL  0
LOOP:
    LACC  0
    OUT   0,04H
    ADD   #05h
    SACL  0
    SUB   #0FFFh
        BCND LOOP,LEQ
    B     START      .END
```

Exp.No:09

Design of IIR Filter

Date :

AIM:

To design a IIR low pass and High pass filter using TMS320C6x DSP Processor.

APPARATUS REQUIRED:

- 1.System with VI debugger software
- 2.TMS 320C6x Kit.
- 3.CRO

PROGRAM: (LOW PASS FILTER)

```
.MMREGS
.TEXT
    TEMP .SET 0
    INPUT .SET 1
    T1 .SET 2
    T2 .SET 3
    T3 .SET 4
;
    K .SET 315eh
    M .SET 4e9fh

;cut-off freq is 1Khz. = Fc
;sampling frequency is 100 æs (ie) 0.1ms.
; a = 2 * (355/113) * 1000 = 6283.18/1000 = 6.28 ;; divide by 1000 for secs
; K = aT/(1+aT) = 6.28*0.1 / (6.28*0.1+1) = 0.3857
; M = 1/(1+aT) = 1 / (6.28*0.1+1) = 0.61425
;convert to Q15 format
; K = K * 32767 = 12638.23 = 315Eh
; M = M * 32767 = 20127.12 = 4E9Fh

;Sampling Rate is 100 æs & Cut off Frequency is 1 Khz

    LDP    #100H
    LACC   #0
    SACL   T1
    SACL   T2
    SACL   TEMP
    OUT    TEMP,4      ;CLEAR DAC BEFORE START TO WORK
LOOP:
    LACC   #0
    SACL   TEMP
    OUT    TEMP,5      ;OUTPUT LOW TO DAC2 TO CALCULATE TIMING
;
    IN     TEMP,06     ;SOC
;
    LAR    AR7,#30h    ;CHANGE VALUE TO MODIFY SAMPLING FREQ.
                    ;sampling rate 100 æs.
```

```

        MAR *,AR7
BACK:    BANZBACK,*-
;
        IN    INPUT,4      ;INPUT DATA FROM ADC1
NOP
NOP
;
        LACC  INPUT
        AND   #0FFFH
        SUB   #800h
        SACL  INPUT
;
        LT    INPUT
        MPY   #K
        PAC
        SACH  T1,1
;;;CALL MULT ----MULTIPLICATION TO BE DONE WITH K
;;RESULT OF MULT IN T1
;
        LT    T2      ;PREVIOUS RESULT IN T2
        MPY   #M
        PAC
        SACH  T3,1
;;;CALL MULT ----MULTIPLICATION TO BE DONE WITH M
;;RESULT OF MULT IN T3+
;

```

HIGH PASS FILTER:

```

.MMREGS
        .TEXT
START:
        LDP #100H
        LACC #00H
        SACL 00H
        SACL 01H
        SACL 02H
        SACL 03H
        SACL 04H
        SACL 05H
LOOP:
        LACC #00H
        SACL 00H
        IN 0,06H
        LAR AR7,#30H
        MAR *,AR7
BACK:    BANZ BACK,*-
;
        LT 01H
;
        MPY #0FFFFB5DEH
;
        PAC
;
        SACH 05H,1

```

```

IN 0,04H
    NOPS
    NOP
    NOP
    NOP
    LT 01H
;    MPY #0FFFFB5DEH
    MPY #4A22H
;    MPY #315EH
    PAC
    SACH 05H,1
    LACC 00H
    AND #0FFFH
    XOR #800H
    SUB #800H
    SACL 00H
    SACL 01H
    ZAP
    LT 00H
;    DMOV 00H
;    LTD 00H
    MPY #4A22H
;    MPY #315EH
    PAC
    SACH 02H,1
    LT 03H
    MPY #1446H
;    MPY #4E9FH
    PAC
    SACH 04H,1
    LACC 02H
    ADD 04H
    SUB 05H
    SACL 03H
    ADD #800H
    SACL 00H
;    OUT 00H,1AH
    OUT 0,04H
    B LOOP
    NOP
    NOP
H:    B H

```

Ex.No: 10

DESIGN OF FIR FILTER

Date:

AIM:

To design a FIR low pass filter using Blackmann Windowing Technique.

APPARATUS REQUIRED:

- 1.System with VI debugger software
- 2.TMS 320C6x Kit.
- 3.CRO

ALGORITHM:

1. Start the program.
2. Initialize the C table value.
3. Load the auxillary register with 0200H.
4. Modify the auxillary register zero.
5. Block the C table to the program.
6. Set configuration control bit.
7. Load the data pointer with 0AH.
8. Initialize the analog to digital conversion.
9. Load the auxillary register 1 with 0300 content.
10. Load the accumulator in 8000H.
11. AND the accumulator with 0FFFH.
12. Subtract the accumulator content with data 800H.
13. Modify the auxillary register 1.
14. Store the accumulator data in 8000H.
15. Load the auxillary register 1 with content 0333H.
16. Zero the accumulator register.
17. Multiply the accumulator with data.
18. Load the program register, with PM bits to accumulator.
19. Load the auxillary register 1 with content 0300H.
20. Add accumulator content with 800H.
21. Shift the accumulator right 1 bit.
22. Store the accumulator content in 8200 location.
23. Branch the program to step 7.

PROGRAM

LOW PASS FILTER

```
.mmregs  
.text  
B      START
```

CTABLE:

```
.word 0FFE7H  
.word 0FFD3H  
.word 0FFC6H  
.word 0FFC4H  
.word 0FFD0H  
.word 0FFECH  
.word 018H  
.word 051H  
.word 08CH  
.word 0B9H  
.word 0C2H  
.word 092H  
.word 01AH  
.word 0FF5FH  
.word 0FE78H  
.word 0FD9AH  
.word 0FD10H  
.word 0FD30H  
.word 0FE42H  
.word 071H  
.word 03B5H  
.word 07CAH  
.word 0C34H  
.word 01054H  
.word 01387H  
.word 01547H  
.word 01547H  
.word 01387H  
.word 01054H  
.word 0C34H  
.word 07CAH  
.word 03B5H  
.word 071H  
.word 0FE42H  
.word 0FD30H  
.word 0FD10H  
.word 0FD9AH  
.word 0FE78H  
.word 0FF5FH  
.word 01AH  
.word 092H  
.word 0C2H  
.word 0B9H  
.word 08CH
```

```

.word 051H
.word 018H
.word 0FFECH
.word 0FFD0H
.word 0FFC4H
.word 0FFC6H
.word 0FFD3H
.word 0FFE7H

```

START:

```

MAR      *,AR0
LAR      AR0,#0200H
RPT      #33H
BLKP     CTABLE,*+
SETC     CNF
ISR: LDP      #0AH
LACC     #0
SACL     0
OUT      0,05
IN       0,06H
LAR      AR7,#0
MAR      *,AR7
BACK:    BANZBACK,*-
IN       0,4
NOP
NOP
NOP
NOP
MAR      *,AR1
LAR      AR1,#0300H
LACC     0
AND      #0FFFH
SUB      #800H
SACL *
LAR      AR1,#333H
MPY      #0
ZAC
RPT      #33H
MACD     0FF00H,*-
APAC
LAR      AR1,#0300H
SACH*
LACC*
ADD      #800h
SACL *
OUT      *,4
LACC     #0FFH
SACL     0
OUT      0,05
NOP
B        ISR

```

HIGH PASS FILTER

* Approximation type: Window design - Blackmann Window

* Filter type: highpass filter

* Filter Order: 52

* Cutoff frequency in KHz = 3.000000

.mmregs

.text

B START

CTABLE:

.word 0FCD3H

.word 05H

.word 0FCB9H

.word 087H

.word 0FD3FH

.word 01ADH

.word 0FE3DH

.word 0333H

.word 0FF52H

.word 04ABH

.word 0FFF8H

.word 0595H

.word 0FFACH

.word 0590H

.word 0FE11H

.word 047CH

.word 0FB0BH

.word 029DH

.word 0F6BAH

.word 0AEH

.word 0F147H

.word 01CH

.word 0E9FDH

.word 04C5H

.word 0D882H

.word 044BCH

.word 044BCH

.word 0D882H

.word 04C5H

.word 0E9FDH

.word 01CH

.word 0F147H

.word 0AEH

.word 0F6BAH

.word 029DH

.word 0FB0BH

.word 047CH

.word 0FE11H

.word 0590H

.word 0FFACH

.word 0595H

.word 0FFF8H

```

        .word 04ABH
        .word 0FF52H
.word 0333H
.word 0FE3DH
.word 01ADH
.word 0FD3FH
.word 087H
.word 0FCB9H
.word 0FCD3H
START:
    MAR *,AR0
    LAR AR0,#0200H
    RPT #33H
    BLKP CTABLE,*+
    SETC CNF
* Input data and perform convolution
ISR:  LDP #0AH
      LACC#0
      SACL 0
      OUT 0,05          ;pulse to find sampling frequency
      IN  0,06H
      LAR AR7,#0        ;change value to modify sampling freq.
      MAR *,AR7
BACK:  BANZBACK,*-
      IN  0,4
      NOP
      MAR *,AR1
      LAR AR1,#0300H
      LACC0
      AND #0FFFH
      SUB #800H
      SACL *
      LAR AR1,#333H
      MPY #0
      ZAC
      RPT #33H
      MACD 0FF00H,*-
      APAC
      LAR AR1,#0300H
      SACH*             ;give as sach *,1 incase of overflow
      LACC*
      ADD #800H
      SACL *
      OUT *,4
      LACC#0FFH
      SACL 0
      OUT 0,05
      NOP
      B    ISR
      .end

```