RAJALAKSHMI INSTITUTE OF TECHNOLOGY
BELIEVE IN THE POSSIBILITES
AN AUTONOMOUS INSTITUTION

RAJALAKSHMI INSTITUTE OF TECHNOLOGY

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

LAB MANUAL

ACADEMIC YEAR: 2024 – 2025 (ODD SEMESTER)

Subject Code & Name: EC23431 - DIGITAL SIGNAL PROCESSING

Year & Semester: II/IV

Vision of the College

To set a benchmark in the field of Engineering education by providing quality technical education that fosters the spirit of learning, research and globally competent professionalism.

Mission of the College

- To impart education that caters to the growing challenges of the Industry and social needs of our nation.
- To constantly upgrade the standards of teaching and learning in the field of engineering and technology while promoting a healthy research atmosphere.
- To foster a healthy symbiosis with the industry through meaningful and dynamic interactions.

Department Vision

- To initiate high-quality technical education and to nurture young minds towards creative thinking that inspires them to undertake innovations in the field of Electronics and Communication Engineering (ECE) and be competent in the global arena.

Department Mission

- Constantly upgrade engineering pedagogy that caters to the growing challenges of the industry.
- Develop conceptual learning that leads towards critical and innovative thinking.
- Establish good harmony with industry that fills the gap between academia and the outside world enabling the students to prepare for diverse and competitive career paths.
- To endorse higher studies and pursue research in the ECE discipline with sensitivity toward societal requirements.

Programme Educational Objectives

PEO I

To enable graduates to pursue research, or have a successful career in academia or industries associated with Electronics and Communication Engineering, or as entrepreneurs.

PEO II

To develop skills for applying necessary computing techniques and communication tools among the students, for catering to the current industrial needs in the broader domain of Electronics and Communication Engineering.

PEO III

To inculcate ethical values, leadership qualities, and team spirit for promoting innovations and entrepreneurship qualities among students in addressing societal concerns.

Programme Specific Outcomes

- To analyze, design and develop solutions by applying foundational concepts of electronics and communication engineering.
- To apply design principles and best practices for developing quality products for scientific and business applications
- To adapt to emerging information and communication technologies (ICT) to innovate ideas and solutions to existing/novel problems

Programme Outcomes

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
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.
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.
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.
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.
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.
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.
8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

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.
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.
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.

SYLLABUS

EC23431 DIGITAL SIGNAL PROCESSING **L T P C**

3 0 2 4

COURSE OBJECTIVES :

To introduce the usage of discrete Fourier transform in linear filtering

To explore the characteristics of digital filters and design digital IIR filters.

To explore the characteristics of digital filters and design digital FIR filters.

To explore the effects of finite precision representation on digital filters.

To introduce the concepts of DSP processor and to explore the fundamentals of multi rate signal processing and its applications.

LIST OF EXPERIMENTS

1. Generation of elementary Discrete-Time sequences
2. Linear and Circular convolutions
3. Auto correlation and Cross Correlation
4. Frequency Analysis using DFT
5. Design of FIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation
6. Design of Butterworth and Chebyshev IIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operations
7. Study of architecture of Digital Signal Processor
8. Perform MAC operation using various addressing modes
9. Generation of various signals and random noise
10. Design and demonstration of FIR Filter for Low pass, High pass, Band pass and Band stop filtering
11. Design and demonstration of Butter worth and Chebyshev IIR Filters for Low pass, High pass, Band pass and Band stop filtering
12. Implement an Up-sampling and Down-sampling operation in DSP Processor

COURSE OUTCOMES :

At the end of the course students will be able to:

CO1:Analyze of digital signals using DFT

CO2:Design IIR filters for given frequency

CO3: Design FIR filters for given frequency

CO4:Characterize the effects of finite precision representation in digital filters

CO5:Design of various Multirate filters

LIST OF EXPERIMENTS

S.No	Name of Experiments
	SIMULATION USING SCILAB
1.	Generation of Signals using SCILAB.
2.	Perform Linear Convolution using SCILAB.
3.	Perform Circular Convolution using SCILAB.
4.	Perform Auto Correlation using SCILAB.
5.	Perform Cross Correlation using SCILAB.
6.	Frequency Analysis using DFT
7.	Design of FIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation using SCILAB.
8.	Design of Butterworth and Chebyshev IIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operations using SCILAB.
	IMPLEMNTATION USING DSP PROCESSOR
9.	Study of architecture of Digital Signal Processor
10.	Perform MAC operation using various addressing modes
11.	Generation of various signals and random noise.
12.	Design and demonstration of FIR Filter for Low pass, High pass, Band pass and Band stop filtering.
13.	Design and demonstration of Butter worth and Chebyshev IIR Filters for Low pass, High pass, Band pass and Band stop filtering
14.	Implement an Up-sampling and Down-sampling operation in DSP Processor

INDEX

S.No	Date	Name of Experiments	Sign
		SIMULATION USING SCILAB	
1.		Generation of Signals using SCILAB.	
2.		Perform Linear Convolution using SCILAB.	
3.		Perform Circular Convolution using SCILAB.	
4.		Perform Auto Correlation using SCILAB.	
5.		Perform Cross Correlation using SCILAB.	
6.		Frequency Analysis using DFT	
7.		Design of FIR filters (LPF/HPF/BPF/BSF) and demonstrates the filtering operation using SCILAB.	
8.		Design of Butterworth and Chebyshev IIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operations using SCILAB.	
		IMPLEMNTATION USING DSP PROCESSOR	
9.		Study of architecture of Digital Signal Processor	
10.		Perform MAC operation using various addressing modes	
11.		Generation of various signals and random noise.	
12.		Design and demonstration of FIR Filter for Low pass, High pass, Band pass and Band stop filtering.	
13.		Design and demonstration of Butter worth and Chebyshev IIR Filters for Low pass, High pass, Band pass and Band stop filtering	
14.		Implement an Up-sampling and Down-sampling operation in DSP Processor	

SCILAB PROGRAMS

Expt. No:	Generation of Signals using SCILAB.	Date:

AIM:

To generate the given set of basic signals such as unit step, impulse, ramp, rectangular, triangular, sine and cosine signals.

REQUIREMENTS:

SCILAB, Personal computer.

ALGORITHM:

SINE SEQUENCE:

Step 1: Initialize the time period.

Step 2: Design the function of sine wave.

Step 3: Plot the sine sequence.

Step 4: Display the output

COSINE SEQUENCE:

Step 1: Initialize the time period.

Step 2: Design the function of cosine wave.

Step 3: Plot the cosine sequence.

Step 4: Display the output

RAMP SEQUENCE:

Step 1: Read the length of the ramp sequence.

Step 2: Define the length of the ramp sequence.

Step 3: Plot the ramp sequence using stem function.

Step 4: Display the output.

EXPONENTIAL SEQUENCE:

Step 1: Read the length of the exponential sequence.

Step 2: Read its amplitude.

Step 3: Plot the exponential sequence using stem function.

Step 4: Display the output.

UNIT-STEP SEQUENCE:

Step 1: Read the length of the unit-step sequence.

Step 2: Plot the unit-step sequence.

Step 3: Display the output.

UNIT-IMPULSE SEQUENCE:

Step 1: Read the length of the unit-impulse sequence.

Step 2: Plot the unit-impulse sequence.

Step 3: Display the output.

PROGRAM:

```
clc
clear
close
//Continuous time signal generation
// Impulse Signal
t=-5:5
[r,c]=size(t)
delta=[zeros(1,(c-1)/2), ones(1,1), zeros(1,(c-1)/2)]
subplot(421)
plot(t,delta)
xlabel('Time (s)')
ylabel('Amplitude')
title('Impulse Signal')
// Step Signal with step size A
// For unit step signal, keep A=1
A=2;
```

```
t1=0:0.01:10
[r1,c1]=size(t1)
u=A*ones(1,c1)
subplot(422)
plot(t1,u)
xlabel('Time (s)')
ylabel('Amplitude')
title('Step Signal')

// Step Signal with step size A
//For unit Ramp signal keep A=1
A=2;
t2=0:0.01:10
r=A*t2 subplot(423)
plot(t2,r)
xlabel('Time (s)')
ylabel('Amplitude')
title('Ramp Signal')

//Parabolic Signal
A=3
t3=0:0.01:2
p=(A*t^2)/2
subplot(424)
plot(t,p)
xlabel('Time (s)')
ylabel('Amplitude')
title('Parabolic Signal')
```

```
//Sinusoidal Signal
t4=0:0.001:1
f=2; // 2 Hz signal
Sin_Sig=A*sin(2*%pi*f*t4)
subplot(425)
plot(t4,Sin_Sig)
xlabel('Time (s)')
ylabel('Amplitude')
title('Sinusoidal Signal')
// Cosine Signal
t5=0:0.01:1
f1=3
Cos_Sig=A*cos(2*%pi*f1*t)
subplot(426)
plot(t5,Cos_Sig)
xlabel('Time (s)')
ylabel('Amplitude')
title('Cosinudoidal Signal')
// Eponential Signal
t6=0:0.01:1
A2=2
b=2
Exp_Sig=A2*%e^(b*t6)
subplot(427)
plot(t6,Exp_Sig)
xlabel('Time (s)')
ylabel('Amplitude')
title('Exponential Signal')
```

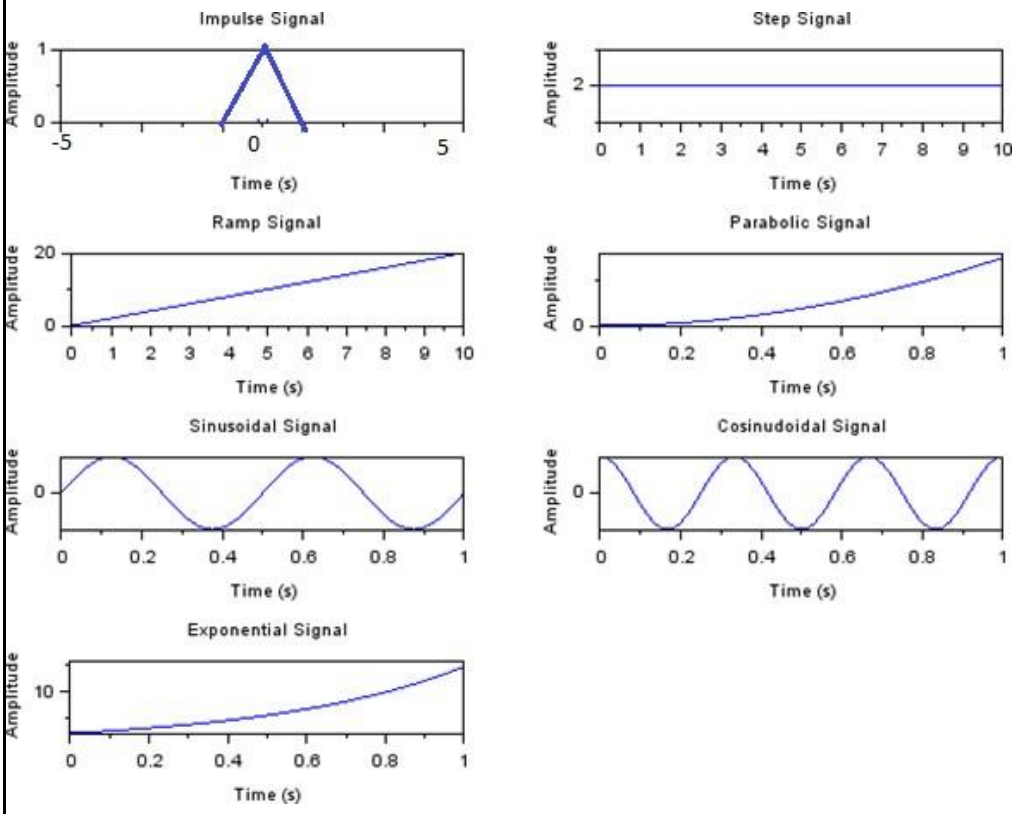
b. Generation of Signals(Discrete)

```
// Generation of Unit sample Sequence
n=-2:1:2
del_discrete=[zeros(1,2), ones(1,1), zeros(1,2)]
figure
subplot(231)
plot2d3(n,del_discrete)
xlabel('Time (s)')
ylabel('Amplitude')
title('Unit Impulse Sequence')

// Unit Step Sequence
1=0:1:5
N=length(n1)
Unit_Step=ones(1,N)
subplot(232)
plot2d3(n1,Unit_Step)
xlabel('Discrete Time n')
ylabel('Amplitude')
title('Unit Step Sequence')

//Ramp sequence
n2=0:1:10
Unit_ramp=n2 subplot(233)
plot2d3(n2,Unit_ramp)
xlabel('Discrete Time n')
ylabel('Amplitude')
title('Unit ramp Sequence')
```

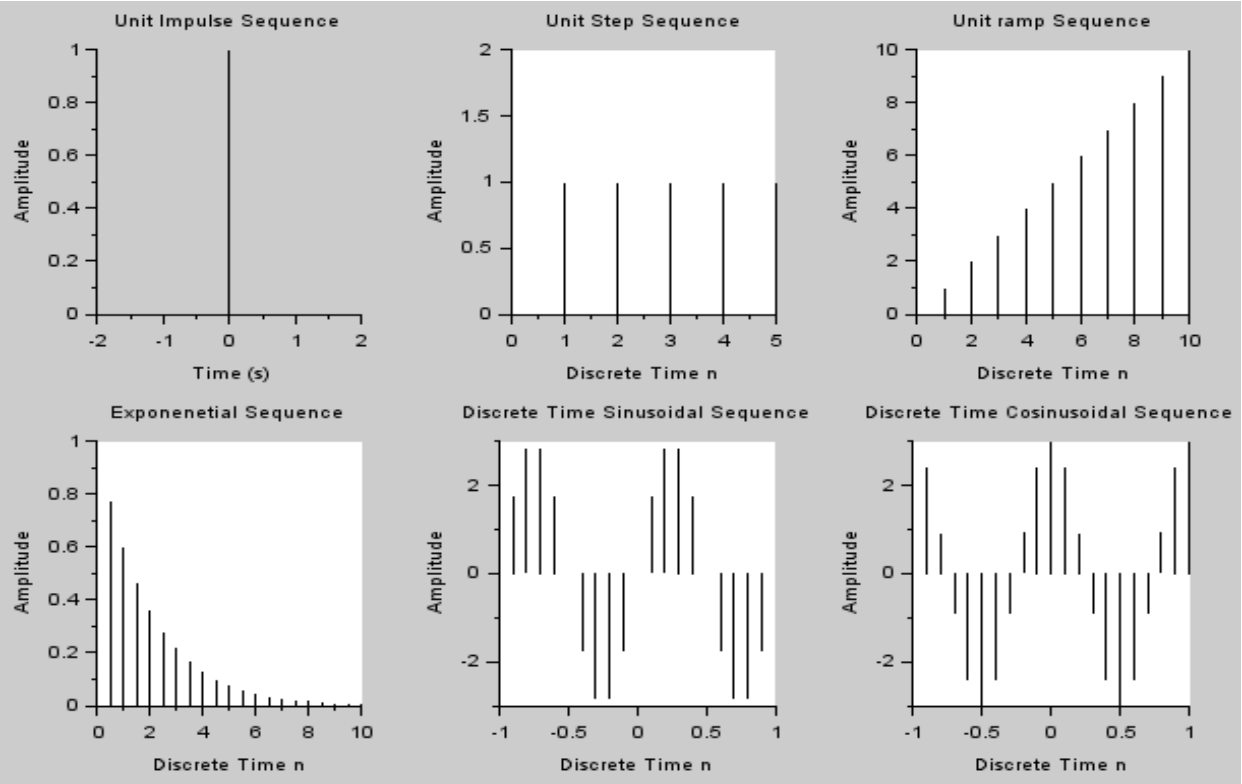
```
// Exponential Sequence
a=0.6
n3=0:0.5:10
Exp_seq=a^n3
subplot(234)
plot2d3(n3,Exp_seq)
xlabel('Discrete Time n')
ylabel('Amplitude')
title('Exponenetial Sequence')
//Sinusoidal Sequence
n4=-1:0.1:1
A6=2
f2=1
Sin_Dis=A*sin(2*%pi*f2*n4)
subplot(235)
plot2d3(n4,Sin_Dis)
xlabel('Discrete Time n')
ylabel('Amplitude')
title('Discrete Time Sinusoidal Sequence')
//Cosinusoidal Sequence
n4=-1:0.1:1
A6=2
f2=1
Sin_Dis=A*cos(2*%pi*f2*n4)
subplot(236)
plot2d3(n4,Sin_Dis)
xlabel('Discrete Time n')
ylabel('Amplitude')
title('Discrete Time Cosinusoidal Sequence')
```

OUTPUT:

Continuous Time Signals:



Discrete Time Signal



Conclusion:

Thus, various signals and waveforms are generated using Scilab and are plotted successfully.

Expt. No:	Perform Linear Convolution using SCILAB.	Date:

AIM:

To perform linear convolution of two given sequences.

REQUIREMENTS:

SCILAB, Personal computer.

THEORY:

Convolution is a mathematical operation equivalent to finite impulse response (FIR) filtering. *Convolution is important in digital signal processing because convolving two sequences in the time domain is equivalent to multiplying the sequences in the frequency-domain.* Convolution finds its application in processing signals especially analyzing the output of a system.

Consider the signals x1(n) and x2(n). Convolution of these two signals is given by

x3(n) = x1(n) * x2(n)

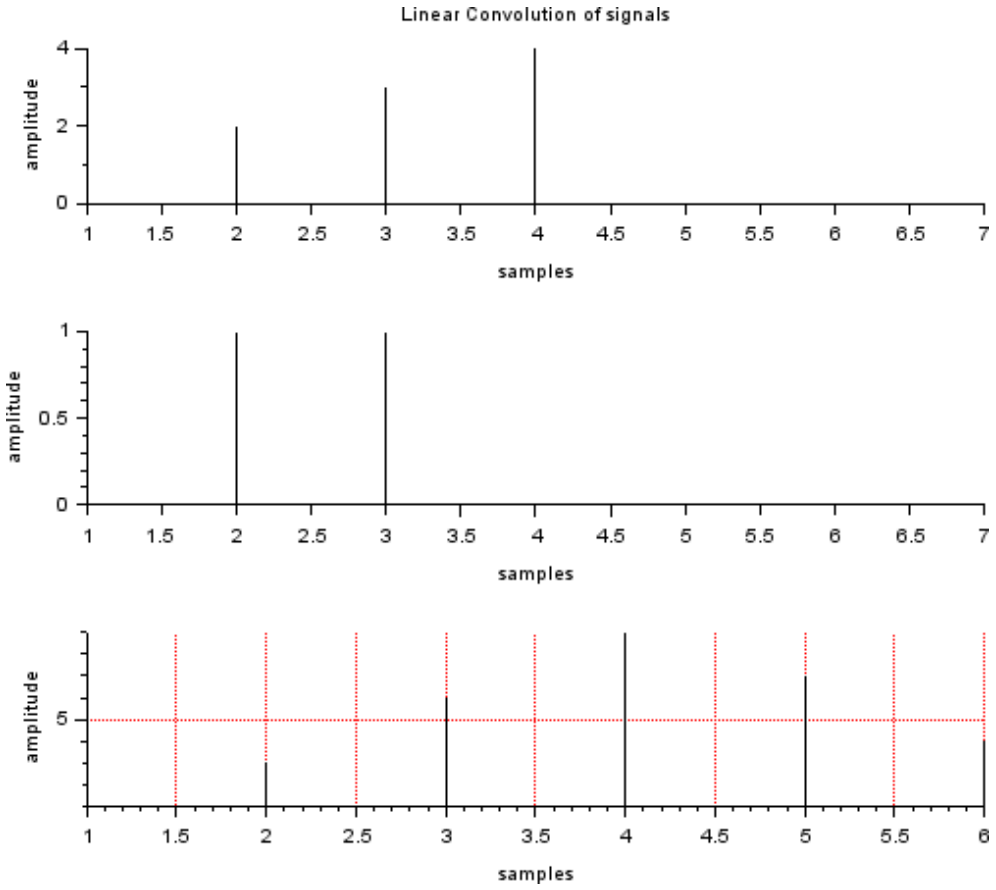
ALGORITHM:

1. Read the input sequences x(n), h(n).
2. Plot the input sequences.
3. Perform linear convolution between x(n) and h(n).
4. Display the convolved sequence.
5. Plot the convolved sequence.

PROGRAM:

```
clc
clear
close
x=input('enter the sequence :')
h=input('enter the sequence:')

y=conv(x,h)
subplot(3,1,1);
plot2d3(x);
xlabel('samples');
ylabel('amplitude');
title('Linear Convolution of signals');
subplot(3,1,2);
plot2d3(h);
xlabel('samples');
ylabel('amplitude');

subplot(3,1,3);
plot2d3(y);
xgrid(5, 1, 7)
xlabel('samples');
ylabel('amplitude');
disp('x=',x)
disp('h=',h)
disp('Y=',y)
```

OUTPUT:


Linear Convolution of signals
amplitude
samples
amplitude
samples
amplitude
samples

Viva Questions:

1. What is the length of the linearly convolved signal?
2. What is the difference between linear convolution and circular convolution?
3. Multiplication of two signals in time domain is equal to Convolution of the two signalsin frequency domain.
4. What are the different methods of linear convolution?

CONCLUSION:

Thus, linear convolution is performed between two sequences successfully.

Expt. No:	Perform Circular Convolution using SCILAB.	Date:

AIM:

To perform circular convolution of two given sequences.

REQUIREMENTS:

SCILAB, Personal computer.

THEORY:

The convolution property of DFT says that, the multiplication of the DFT's of the two sequence is equivalent to the DFT of the circular convolution of the two sequences.

$X_1(K) .X_2(K) = DFT\{ x_1(n) * x_2(n)\}$

ALGORITHM:

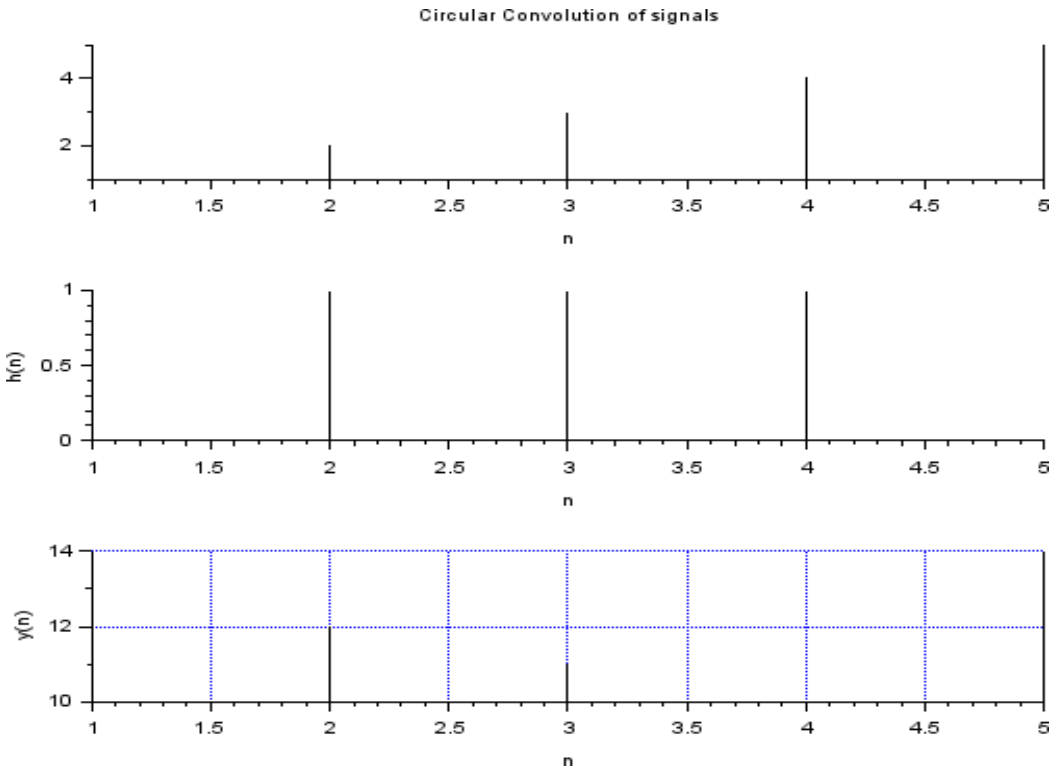
1. Read the input sequences x1(n), x2(n).
2. Calculate the length of the sequences.
3. If the lengths are not equal, do zero padding.
4. Plot the input sequences.
5. Perform circular convolution between x1(n) and x2(n).
6. Display the convolved sequence.
7. Plot the convolved sequence.

PROGRAM:

```
clc
clear
close
x=[1 2 3 4 5]//input('enter the elements of x:');
h=[1 1 1 1]//input('enter the elements of h:');
l=length(x);
m=length(h)
N=max(l,m);
if l<N
x=[x,zeros(1,N-l)]
end
if m<N
h=[h,zeros(1,N-m)]
end
////x=[x,zeros(1,N-l)];
////h=[h,zeros(1,N-1)];
y=zeros(1,N);
for i=1:N
y(i)=0
for j=1:N
k=i-j+1
if ( k<=0)
k=i-j+1+N
end
y(i)=y(i)+x(j)*h(k);
end
end
subplot(3,1,1);
```

```
plot2d3(x);
title('Circular Convolution of signals');
xlabel('n');
subplot(3,1,2);
plot2d3(h);
xlabel('n');
ylabel('h(n)')
subplot(3,1,3);
plot2d3(y);
xgrid(2)
xlabel('n');
ylabel('y(n)')
disp('output',y)
```

OUTPUT:



Viva Questions:

1. What is Zero Padding?
2. What is the length of the circularly convolved signal?
3. What is the difference between overlap add method and overlap save method?

CONCLUSION:

Thus, circular convolution is performed between two sequences successfully.

Expt. No:	Perform Auto Correlation using SCILAB.	Date:

AIM:

To perform Auto Correlation for the discrete sequences.

REQUIREMENTS :

SCILAB, Personal computer.

THEORY:

Autocorrelation, sometimes known as serial correlation in the discrete time case, is the correlation of a signal with a delayed copy of itself as a function of delay. It is often used in signal processing for analyzing functions or series of values, such as time domain signals.

ALGORITHM:

1. Read the input sequences.
2. Enter the starting index.
3. Perform the correlation.
4. Plot the input sequences.
5. Plot the Correlated result.

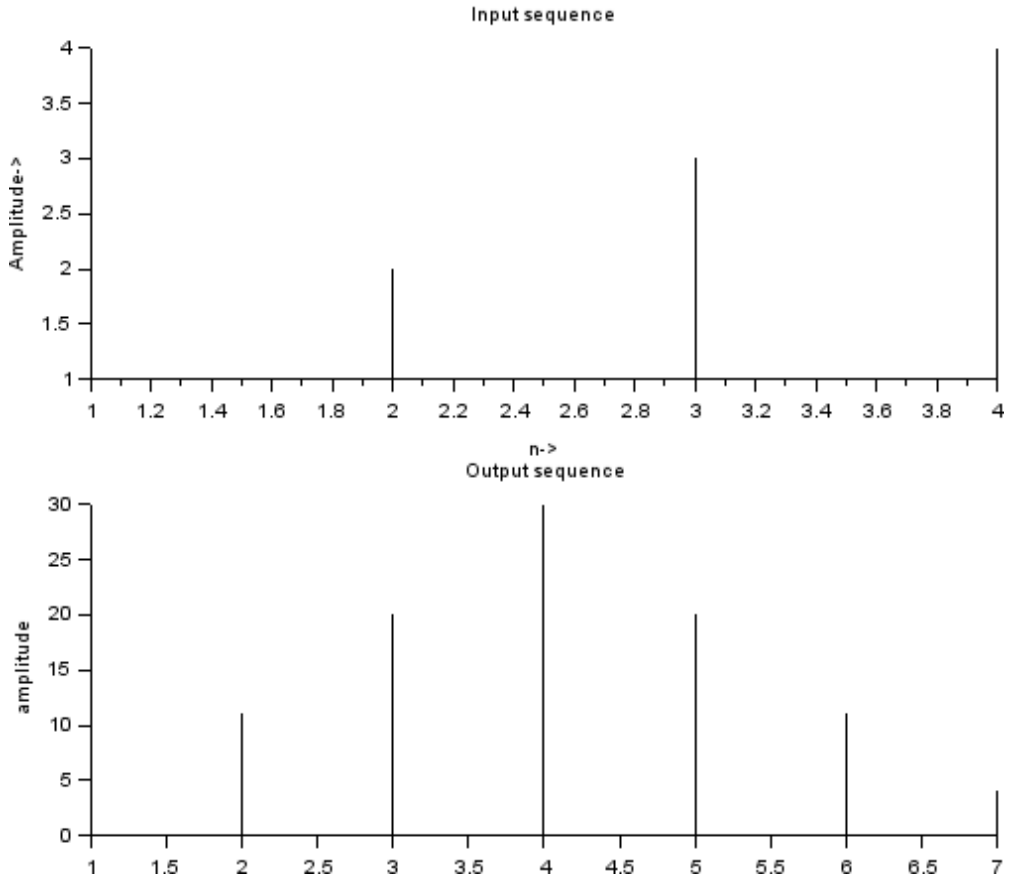
PROGRAM:

// **Auto Correlation**

```
clc;
clear all;
x=input('Enter the sequence 1: ');
y=xcorr(x,x);
subplot(2,1,1);
plot2d3(x);
ylabel('Amplitude->');
xlabel('n->');
title('Input sequence');
subplot(2,1,2);
```

```
plot2d3(y);
ylabel('amplitude');
xlabel('n->');
title('Output sequence');
disp('the resultant is',y);
```

Output:



Viva Questions:

1. List some of the fields where correlation concepts used
2. What are the applications of correlation in DSP?
3. How do you find the autocorrelation of a signal?

CONCLUSION:

Thus, the auto on the discrete sequence was performed and the result was plotted.

Expt. No:	Perform Cross Correlation using SCILAB.	Date:

AIM:

To perform Cross Correlation for the discrete sequences.

REQUIREMENTS:

SCILAB, Personal computer.

THEORY:

Cross-correlation is a measurement that tracks the movements of two or more sets of time series data relative to one another. It is used to compare multiple time series and objectively determine how well they match up with each other and, in particular, at what point the best match occurs.

ALGORITHM:

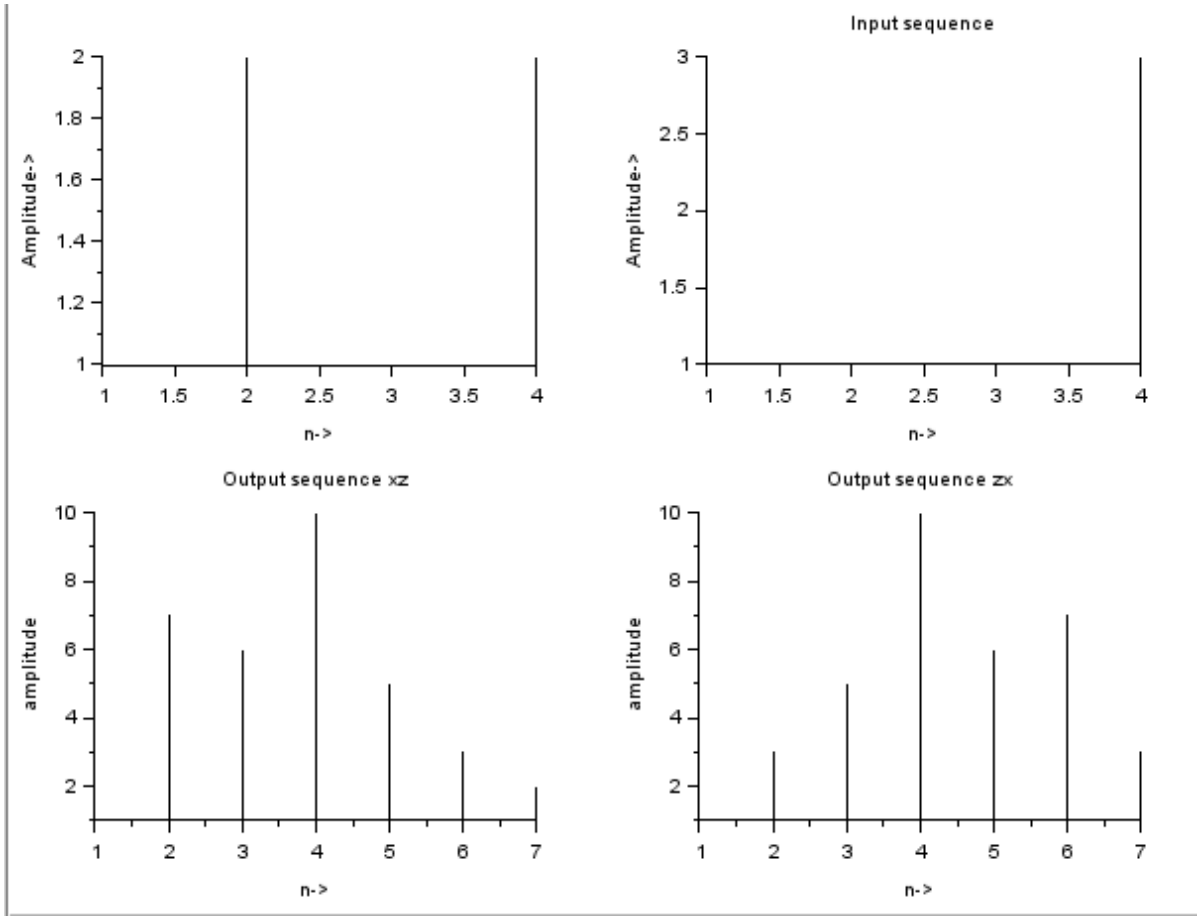
1. Read the input sequences.
2. Enter the starting index.
3. Perform the correlation.
4. Plot the input sequences.
5. Plot the Correlated result.

PROGRAM:

```
clc;
clear all;
x=input('Enter the sequence 1: ');
z=input('Enter the sequence2:');
y1=xcorr(x,z);
y2=xcorr(z,x);
subplot(2,2,1);
plot2d3(x);
ylabel('Amplitude->');
xlabel('n->');
subplot(2,2,2);
plot2d3(z);
```

```
ylabel('Amplitude->');
xlabel('n->');
title('Input sequence');
subplot(2,2,3);
plot2d3(y1);
title('Output sequence xz');
ylabel('amplitude');
xlabel('n->');
subplot(2,2,4);
plot2d3(y2);
title('Output sequence zx');
ylabel('amplitude');
xlabel('n->');
disp('the resultant signal xz is',y1);
disp('the resultant signal zx is',y2);
```

OUTPUT:



CONCLUSION:

Thus, the cross-correlation on the discrete sequence was performed and the result was plotted.

Expt. No:	Frequency Analysis using DFT	Date:

AIM:

To write a SCILAB program to get the single sided amplitude spectrum using FFT.

REQUIREMENTS:

Personal Computer with SCILAB

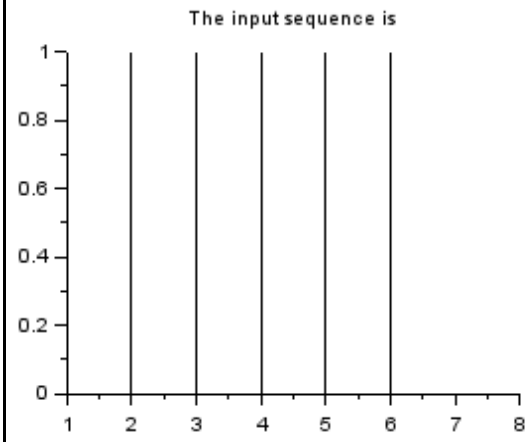
PROGRAM

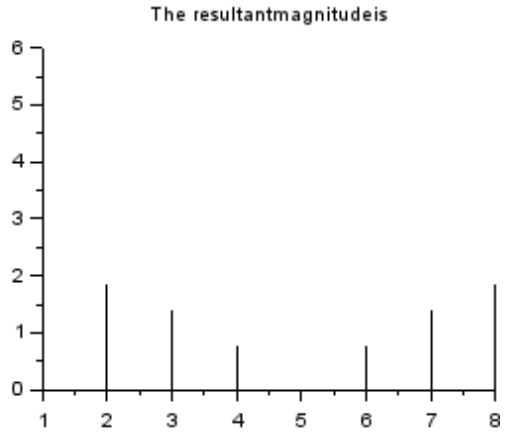
a. Program for calculation of DFT of a signal

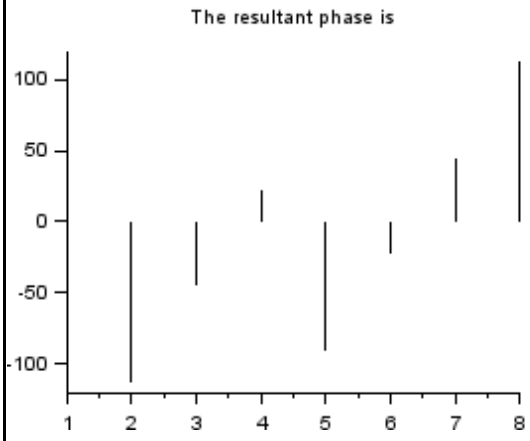
```
clc;
clear ;
x= input ( 'Enter the input sequence' );
N=length(x);
for k =1: N
y(k)=0;
for n =1: N
y(k)=y(k)+x(n).* exp(-%i *2* %pi *(k -1) *(n -1)/N);
end
end
mag = abs (y);
x1= atan ( imag (y),real (y));
phase =x1 *(180/ %pi );
disp ( 'The Resultant DFT Sequence is');
 disp (y);
disp ( 'The Magnitude response is' );
disp (mag);
disp ( 'The phase response is' );
```

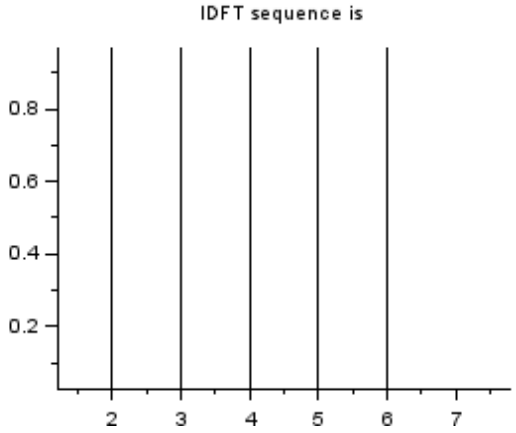
```
disp (phase );
for n=1: N
y(n)=0;
for k =1: N
y(n)=y(n) +(1/ N)*(y(k).* exp (%i *2* %pi *(k -1) *(n-1) /N));
C= real (x);
end
end
disp ( 'The resultant IDFT sequence is' );
disp (C);
subplot (2 ,2 ,1);
 plot2d3 (x);
title ( 'The input sequence is' );
 subplot (2 ,2 ,2);
plot2d3 (mag);
title ( 'The resultant magnitude is' );
subplot (2 ,2 ,3);
plot2d3 (phase);
title ( 'The resultant phase is' );
subplot (2 ,2 ,4);
 plot2d3 (C);
 title ( 'IDFT sequence is' );
```

OUTPUT:


The input sequence is
1
0.8
0.6
0.4
0.2
0
1 2 3 4 5 6 7 8


The resultantmagnitudeis
6
5
4
3
2
1
0
1 2 3 4 5 6 7 8


The resultant phase is
100
50
0
-50
-100
1 2 3 4 5 6 7 8


IDFT sequence is
0.8
0.6
0.4
0.2
2 3 4 5 6 7

b. Program for FFT of the signal

```
clc;
clear all;
x= input ( 'Enter the input sequence' );
 N=length(x)
y= fft (x);
mag = abs (y);
x1= atan ( imag (y),real (y));
phase =x1 *(180/ %pi );
disp ( 'The resultant FFT sequence is' );
disp (y);
disp ( 'The magnitude response is' );
disp (mag);
disp ( 'The phase response is' );
 disp ( phase );
z= ifft (y);
disp ( 'The resultant IFFT sequence is' );
disp(y);
```

OUTPUT:

```
Scilab 6.0.1 Console

Enter the input sequence[1 1 1 1 1 1 1 1]

 The resultant FFT sequence is

   8.   0.   0.   0.   0.   0.   0.   0.

 The magnitude response is

   8.   0.   0.   0.   0.   0.   0.   0.

 The phase response is

   0.   0.   0.   0.   0.   0.   0.   0.

 The resultant IFFT sequence is

   8.   0.   0.   0.   0.   0.   0.   0.

-->
```

Viva Questions:

1. Why FFT is needed?
2. What is FFT? (University)
3. How many multiplications and additions are required to compute N point DFT using radix-2 FFT? (University)
4. What are the applications of FFT algorithm?
5. What is the drawback in DTFT?

CONCLUSION:

Thus, the spectrum of analog signal using DFT-FFT is computed using Scilab.

Expt. No:	Design of FIR filters	Date:

AIM:

To design a FIR filter (lowpass/ High pass/ bandpass/ Band stop) using window techniques.

REQUIREMENTS:

SCILAB, Personal computer.

THEORY:

A discrete time filter produces a discrete-time output sequence y(n0 for the discrete-time input sequence x(n). A filter may be required to have a given frequency response, or a specific response to an impulse, step, or ramp, or simulate an analog system. Digital filters are classified as Finite Impulse Response (FIR) filters and Infinite Impulse Response(IIR) filters. In the FIR system, the impulse response sequence is of finite duration, i.e. it has a finite number of non-zero terms.

FIR filters are designed using following techniques:

1. Fourier series method
2. Frequency sampling method
3. Window techniques

ALGORITHM:

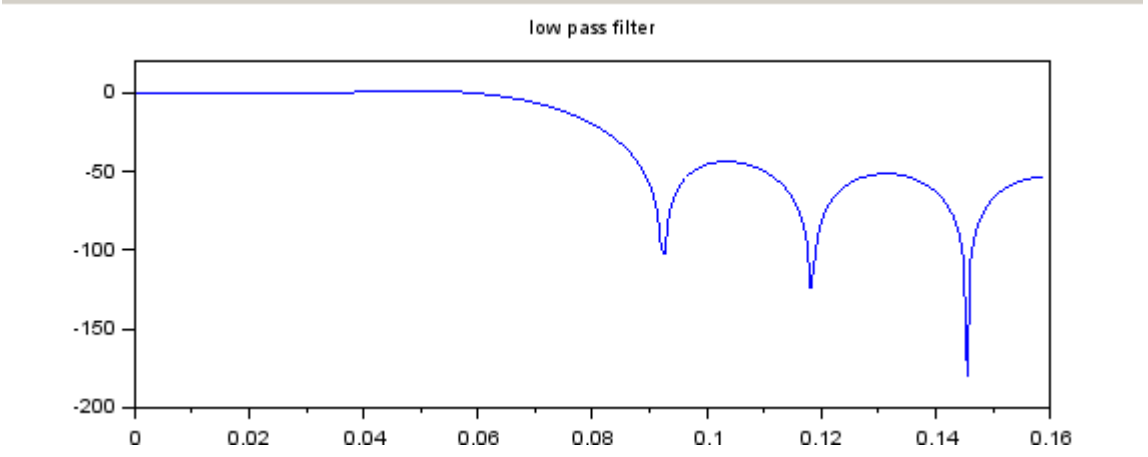
1. Get the passband and stopband ripples.
2. Get the passband and stopband edge frequencies.
3. Read the sampling frequency.
4. Calculate the order of the filter.
5. Find the window coefficients of the specified window.
6. Draw the magnitude and phase responses.

PROGRAM:

// Program to design FIR Low Pass Filter using Rectangular window

```
clc;
clear all;
N=input('enter the order');
wc=input('enter the cutt off frequency');
a=(N-1)/2;
for n=1:N
if(n-1==a)
hd(n)=(wc/%pi);
else
hd(n)=(sin(wc*(n-1-a)))/(%pi*(n-1-a));
end
w(n)=1;
h(n)=hd(n).*w(n);
end
[m,ph]=frmag(h,256);
mag=abs(m);
magdb=20*log(mag);
subplot(2,1,1);
plot(ph/%pi,magdb);
title(' low pass filter')
```

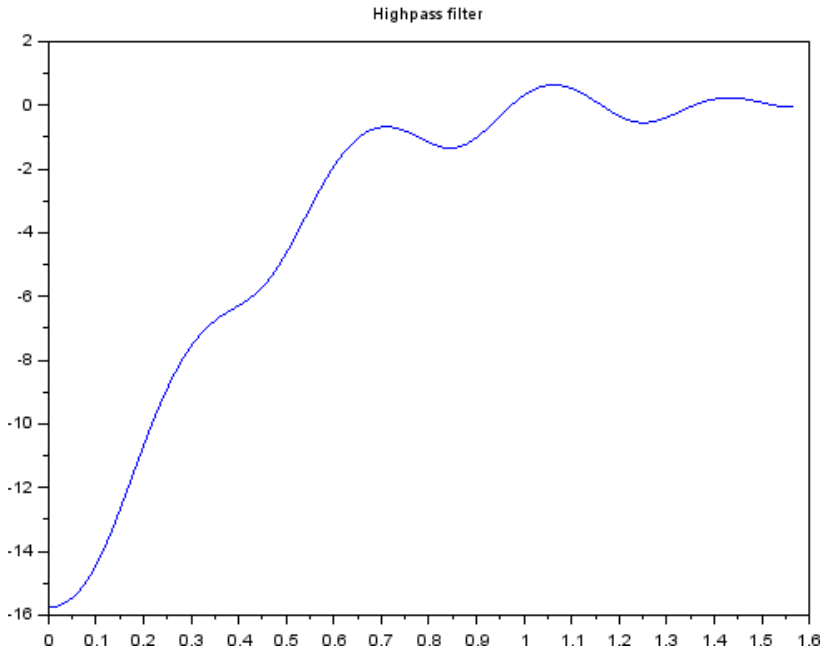
OUTPUT:



// Program to Design FIR High Pass Filter using Blackmann window

```
clc;
close;
N=input('enter the order');
wc=input('enter the cutt off frequency');
a=(N-1)/2;
for n=1:N
if(n-1==a)
hd(n)=1-(wc/%pi);
else
hd(n)=((sin(%pi*(n-1-a)))-sin(wc*(n-1-a)))/(%pi*(n-1-a));
end
w(n)=0.42-0.5*cos((2*%pi*(n-1))/(N-1))+0.08*cos((4*%pi*(n-1))/(N-1));
h(n)=hd(n).*w(n);
end
[m,ph]=frmag(h,256);
mag=abs(m)
magdb=20*log10(mag);
plot(ph/%pi,magdb);
title('Highpass filter')
```

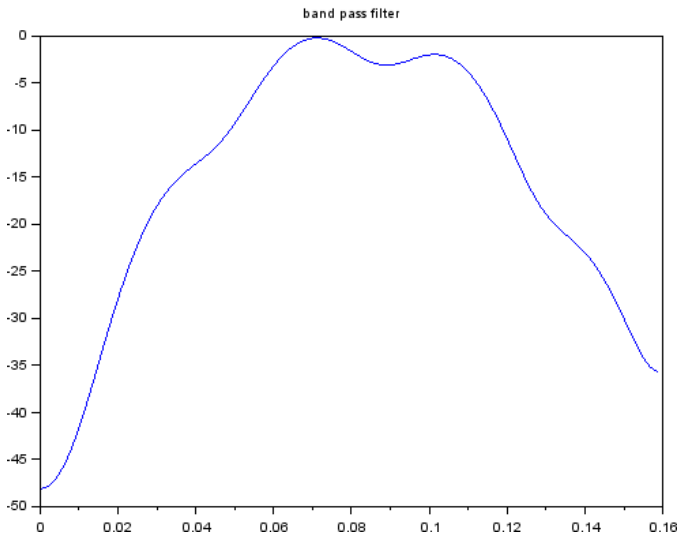
OUTPUT



//Program to Design FIR Band Pass Filter using Hanning window

```
clc;
close ;
N=input('enter the order');
wc1=input('enter the cutt off frequency 1');
wc2=input('enter the cutt off frequency 2');
a=(N-1)/2;
for n=1:N
if(n-1==a)
hd(n)=(wc2-wc1)/%pi;
else
hd(n)=((sin(wc2*(n-1-a)))-sin(wc1*(n-1-a)))/(%pi*(n-1-a));
end
w(n)=0.5-0.5*cos((2*%pi*(n-1))/(N-1));
h(n)=hd(n).*w(n);
end
[m,ph]=frmag(h,256);
mag=abs(m);
magdb=20*log(mag);
subplot(1,1,1);
plot(ph/%pi,magdb);
title('band pass filter')
```

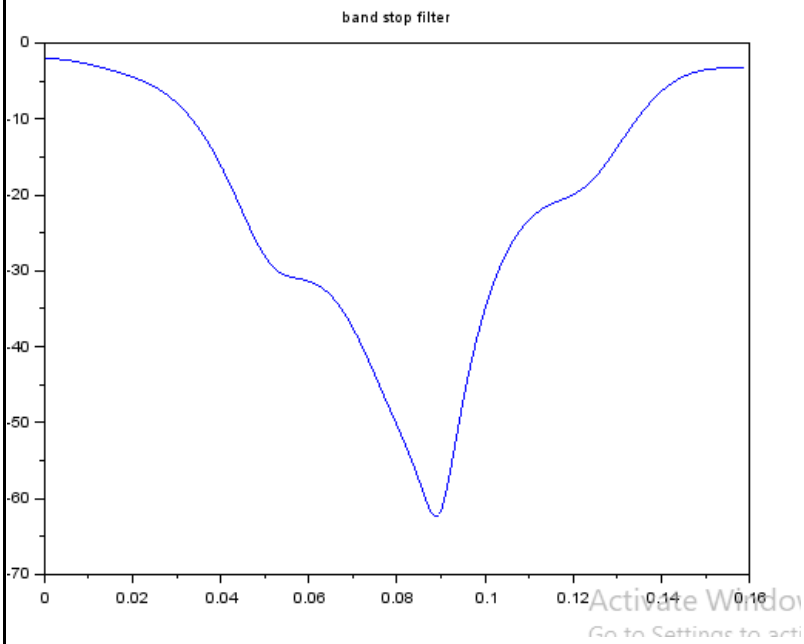
OUTPUT



// Program to Design FIR Band Reject Filter using Hamming window

```
clc;
close;
N=input('enter the order');
wc1=input('enter the cutt off frequency 1');
wc2=input('enter the cutt off frequency 2');
a=(N-1)/2;
for n=1:N
if(n-1==a)
hd(n)=1-(wc2-wc1)/%pi;
else
hd(n)=((sin(wc1*(n-1-a)))-sin(wc2*(n-1-a))+sin(%pi*(n-1-a)))/(%pi*(n-1-a));
end
w(n)=0.54-0.46*cos((2*%pi*(n-1))/(N-1));
h(n)=hd(n).*w(n);
end
[m,ph]=frmag(h,256);
mag=abs(m);
magdb=20*log(mag);
subplot(1,1,1);
plot(ph/%pi,magdb);
title('band stop filter')
```

OUTPUT



Viva Questions:

1. What are FIR filters?
2. Write the steps involved in FIR filter design?
3. What are the disadvantages of FIR Filter? (University)
4. List the well-known design technique for linear phase FIR filter design?

CONCLUSION:

Thus, the FIR filter is designed using windowing technique and the magnitude responses are verified successfully.

Expt. No:	Design of IIR filters	Date:

AIM

To design and implement IIR (LPF, HPF BPF and BSF) Filters.

REQUIREMENTS:

PC with Scilab

PROGRAM:

//Butterworth IIR LOW PASS Filter

```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=90;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(log(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/log(ws/wp));
n=ceil(N);
e=sqrt((10^(0.1*ap))-1);
l=sqrt((10^(0.1*as))-1);
omc=0.5*((wp/e^(1/n))+(ws/l^(1/n)));
hz=iir(n,'lp','butt',omc,[0    0]);
[mag    phase]=frmag(hz,256);
plot(phase,mag);
title("Butterworth IIR Low pass Filter")
xlabel("Frequency");
ylabel("Magnitude");
```

//Butterworth IIR HIGH PASS Filter

```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=90;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(log(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/log(ws/wp));
n=ceil(N);
e=sqrt((10^(0.1*ap))-1);
l=sqrt((10^(0.1*as))-1);
omc=0.5*((wp/e^(1/n))+(ws/l^(1/n)));
hz=iir(n,'hp','butt',omc,[0 0]);
[mag  phase]=frmag(hz,256);
plot(phase,mag);
title("Butterworth IIR High pass Filter")
xlabel("Frequency");
ylabel("Magnitude");
```

//Butterworth IIR Band Pass Filter

```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=90;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(log(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/log(ws/wp));
n=ceil(N);
e=sqrt((10^(0.1*ap))-1);
l=sqrt((10^(0.1*as))-1);
omc=0.5*((wp/e^(1/n))+(ws/l^(1/n)));
```

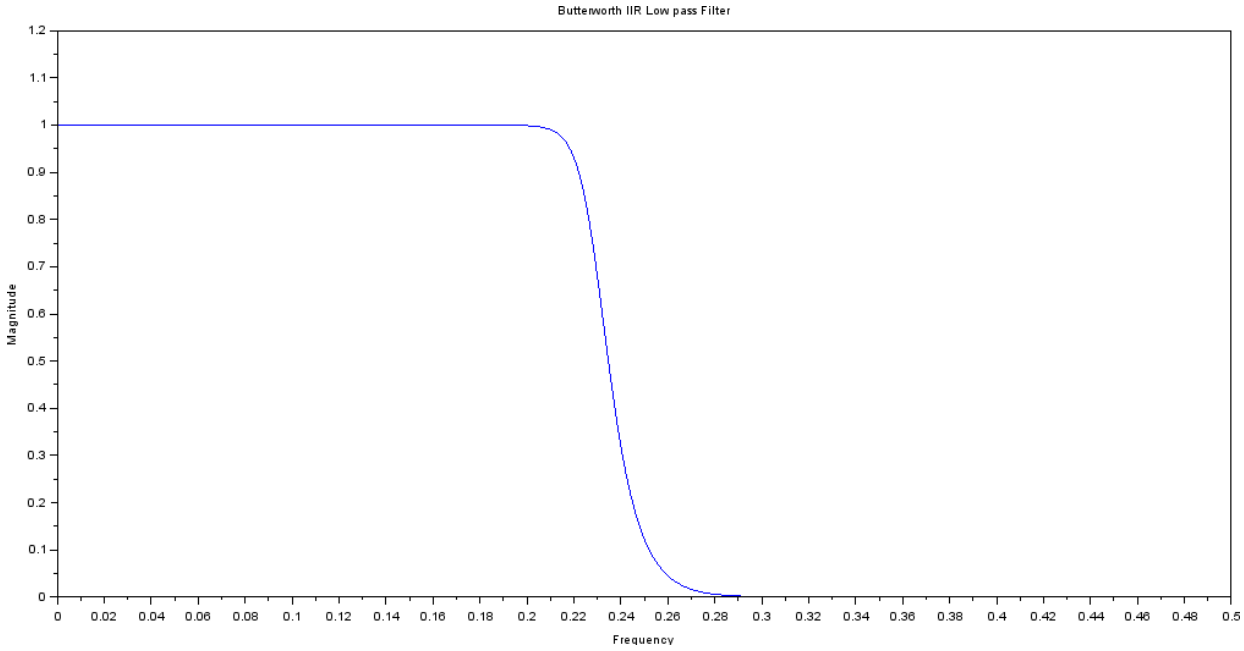
```
hz=iir(n,'bp','butt',[wp ws],[0 0]);
[mag phase]=frmag(hz,256);
plot(phase,mag);
title("Butterworth IIR Band pass Filter")
xlabel("Frequency"); ylabel("Magnitude");
```

//Butterworth IIR Band Stop Filter

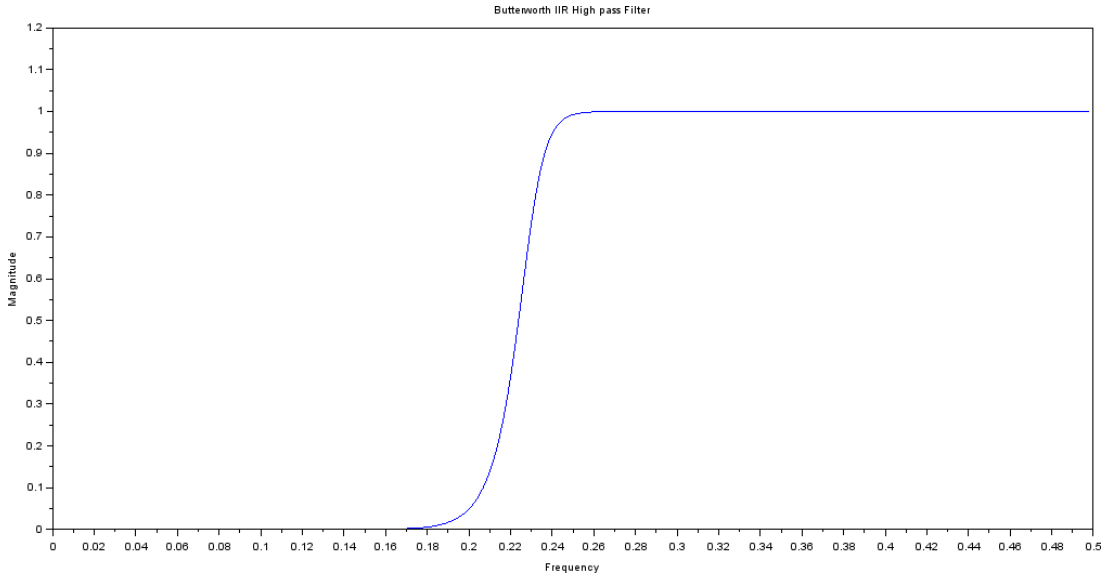
```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=90;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(log(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/log(ws/wp));
n=ceil(N);
e=sqrt((10^(0.1*ap))-1);
l=sqrt((10^(0.1*as))-1);
omc=0.5*((wp/e^(1/n))+(ws/l^(1/n)));
hz=iir(n,'sb','butt',[wp ws],[0 0]);
[mag phase]=frmag(hz,256);
plot(phase,mag);
title("Butterworth IIR Band stop Filter")
xlabel("Frequency"); ylabel("Magnitude");
```

OUTPUT:

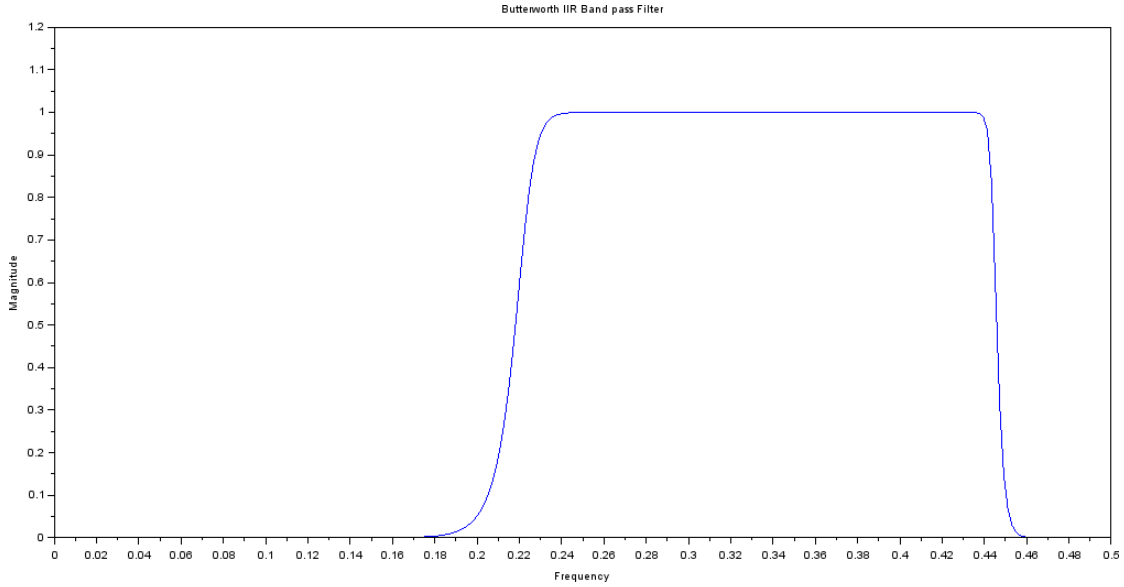
Butterworth IIR LOW PASS Filter



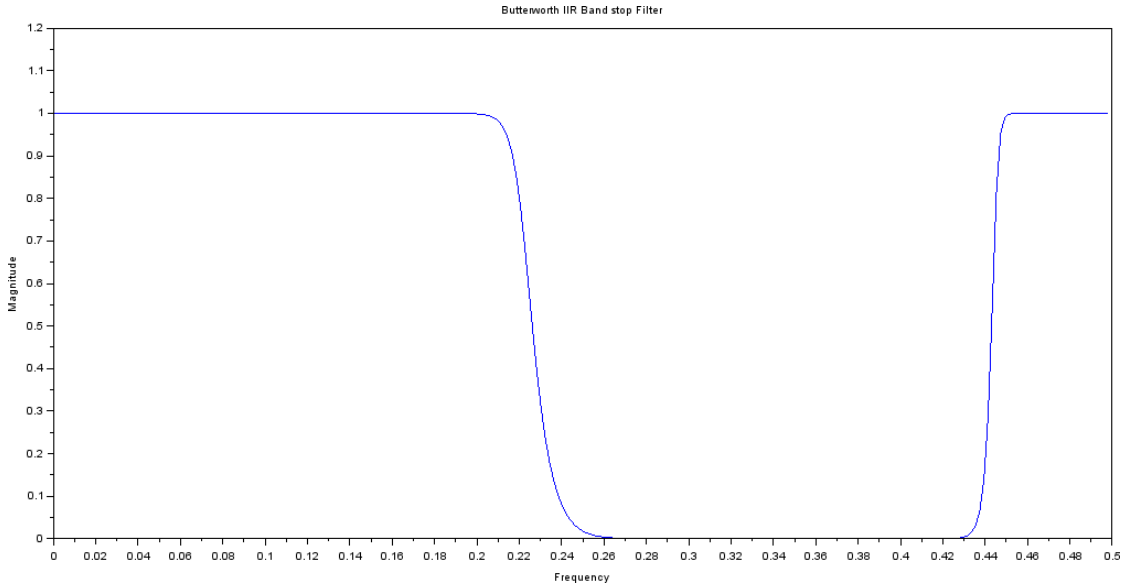
Butterworth IIR HIGH PASS Filter



Butterworth IIR Band Pass Filter



Butterworth IIR Band Pass Filter



PROGRAM:

//Chebyshev Type I IIR Low Pass Filter

```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=90;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(acosh(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/acosh(ws/wp));
n=ceil(N);
hz=iir(n,'lp','cheb1',wp,[0.5 1]);
[mag phase]=frmag(hz,256);
plot(phase,mag);
title("Chebyshev IIR LOW PASS Filter")
xlabel("Frequency");
ylabel("Magnitude");
```

//Chebyshev Type II IIR High Pass Filter

```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=60;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(acosh(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/acosh(ws/wp));
n=ceil(N);
hz=iir(n,'hp','cheb2',ws,[0 0.3]);
[mag phase]=frmag(hz,256);
plot(phase,mag);
title("Chebyshev IIR HIGH PASS Filter")
xlabel("Frequency");
ylabel("Magnitude");
```

//Chebyshev Type II IIR Band Pass Filter

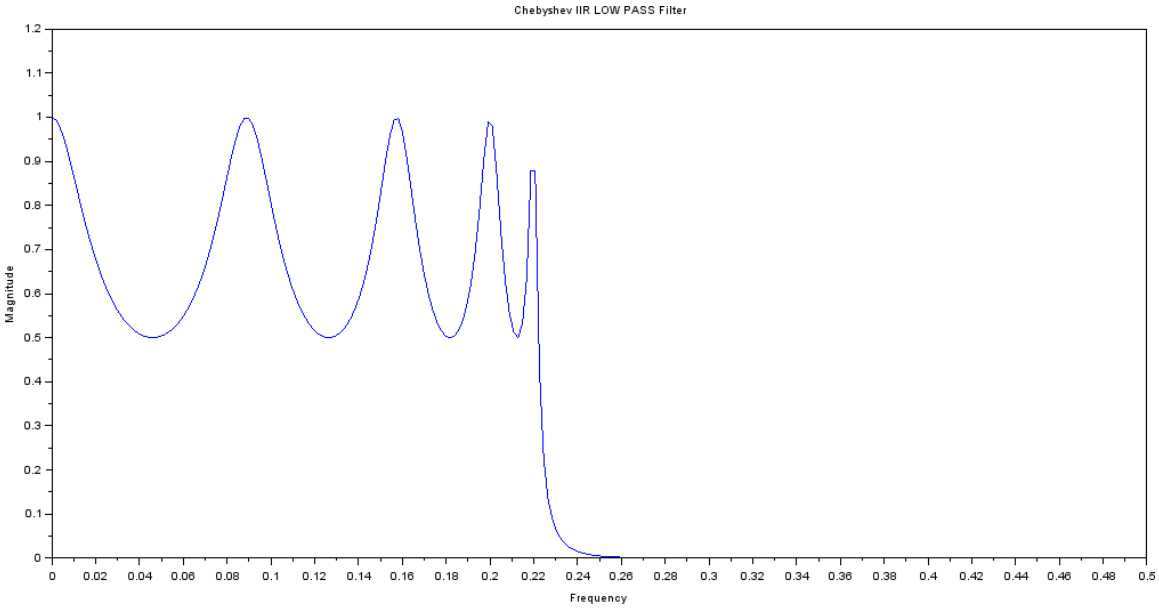
```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=60;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(acosh(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/acosh(ws/wp));
n=ceil(N);
hz=iir(n,'bp','cheb2',[wp ws],[0 0.3]);
[mag phase]=frmag(hz,256);
plot(phase,mag);
title("Chebyshev IIR BAND PASS Filter")
xlabel("Frequency");
ylabel("Magnitude");
```

//Chebyshev Type I IIR BAND STOP Filter

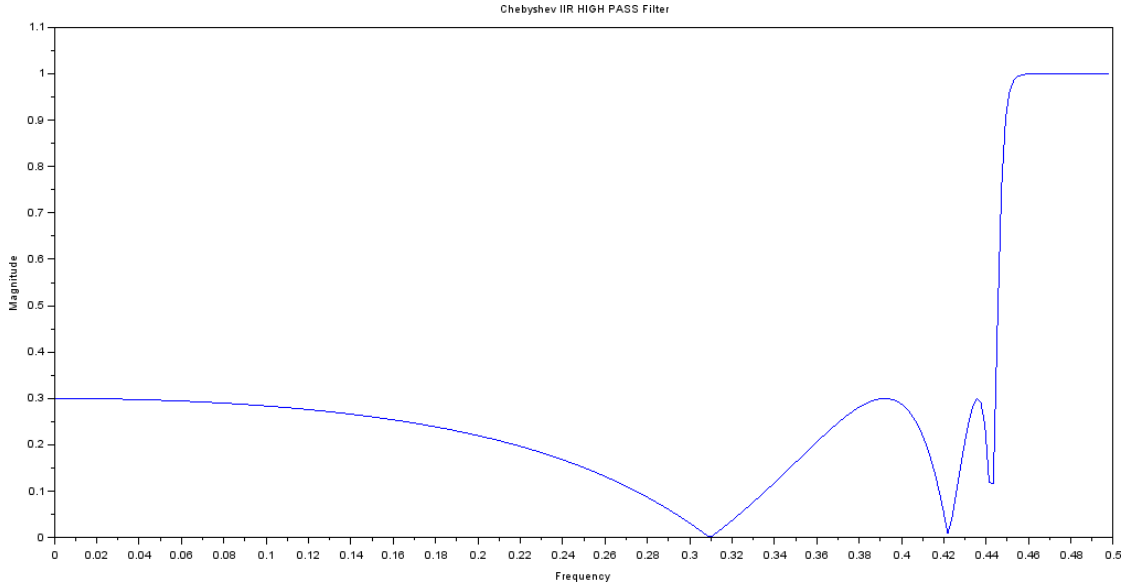
```
clc;
clear all;
Fs=2000;
Fp=1000;
Fsamp=9000;
ap=2;
as=90;
wp=(2*Fp)/Fsamp
ws=(2*Fs)/Fsamp;
N=(acosh(sqrt(((10^(0.1*as))-1)/((10^(0.1*ap))-1)))/acosh(ws/wp));
n=ceil(N);
hz=iir(n,'sb','cheb1',[wp ws],[0.5 1]);
[mag phase]=frmag(hz,256);
plot(phase,mag);
title("Chebyshev IIR Band stop Filter")
xlabel("Frequency");
ylabel("Magnitude");
```

OUTPUT:

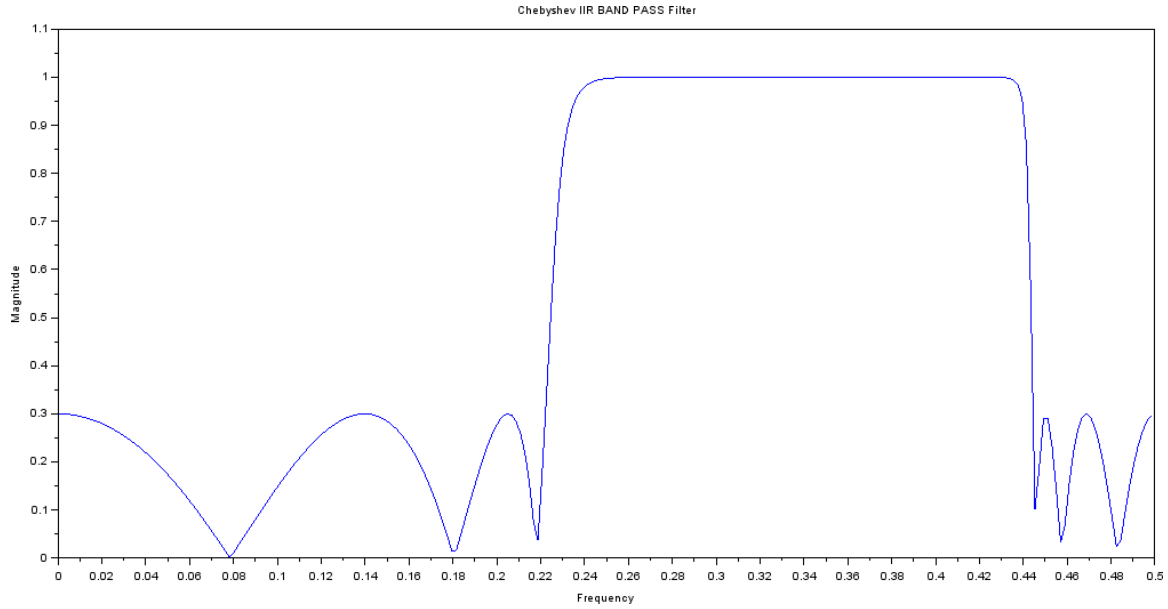
Chebyshev Type I IIR Low Pass Filter



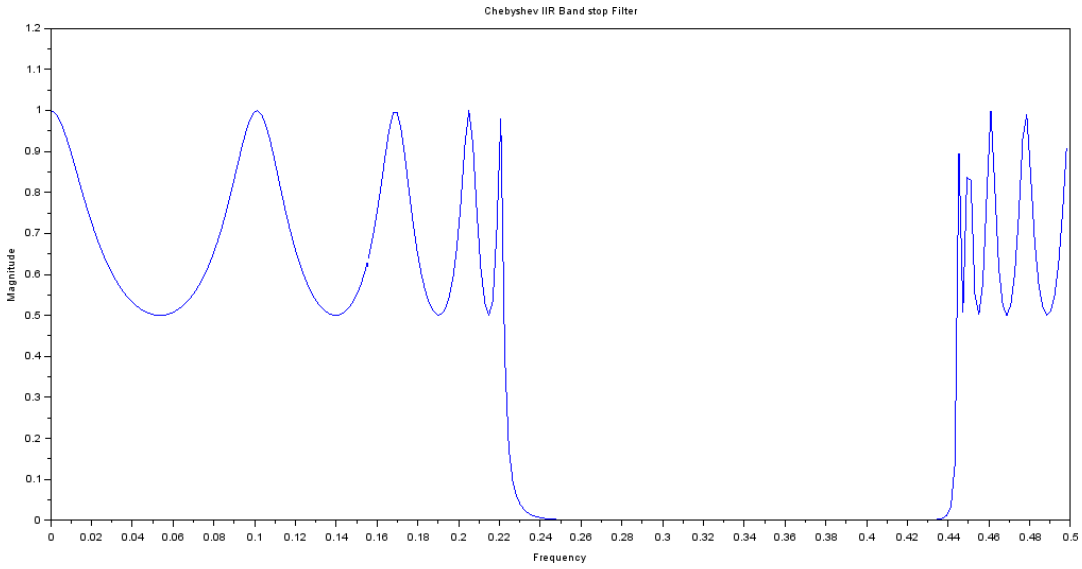
Chebyshev Type II IIR High Pass Filter



Chebyshev Type II IIR Band Pass Filter



Chebyshev Type I IIR BAND STOP Filter



VIVA QUESTIONS:

1. What are the different types of filters based on frequency response?
2. Mention the procedures for digitizing the transfer function of an analog filter?
3. What is Warping Effect? (University)
4. What are the advantages & disadvantages of bilinear transformation?

CONCLUSION:

Thus, the IIR filter is designed and the magnitude and phase responses are verified successfully.

DSP PROCESSOR

Expt. No:	Study of Architecture of Digital Signal Processor	Date:

A signal can be defined as a function that conveys information, generally about the state or behavior of a physical system. There are two basic types of signals viz Analog (continuous time signals which are defined along a continuum of times) and Digital (discrete-time).

Remarkably, under reasonable constraints, a continuous time signal can be adequately represented by samples, obtaining discrete time signals. Thus digital signal processing is an ideal choice for anyone who needs the performance advantage of digital manipulation along with today's analog reality.

Hence a processor which is designed to perform the special operations(digital manipulations) on the digital signal within very less time can be called as a Digital signal processor. The difference between a DSP processor, conventional microprocessor and a microcontroller are listed below.

Microprocessor or General Purpose Processor such as Intel xx86 or Motorola 680xx family

Contains - only CPU

-No RAM

-No ROM

-No I/O ports

-No Timer

Microcontroller such as 8051 family

Contains - CPU

- RAM
- ROM

-I/O ports

- Timer &
- Interrupt circuitry

Some Micro Controllers also contain A/D, D/A and Flash Memory

DSP Processors such as Texas instruments and Analog Devices

Contains:

- CPU
- RAM
- ROM
- I/O ports
- Timer

Optimized for:

- Fast arithmetic
- Extended precision
- Dual operand fetch
- Zero overhead loop
- Circular buffering

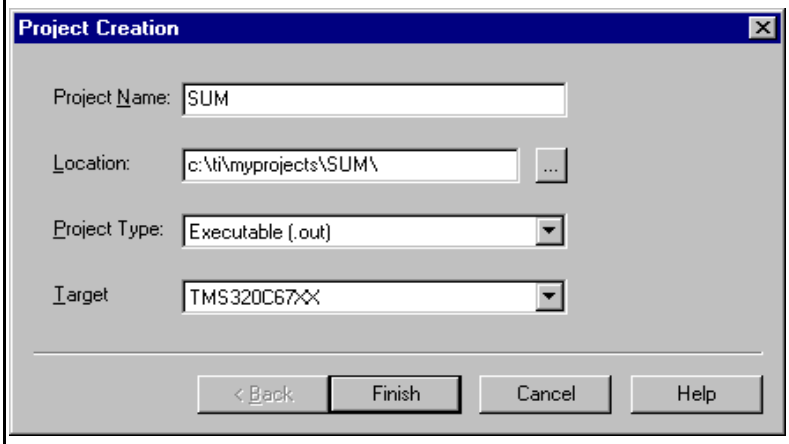
The basic features of a DSP Processor are

Feature	Use
Fast-Multiply accumulate	Most DSP algorithms, including filtering, transforms, etc. are multiplication- intensive
Multiple – access memory architecture	Many data-intensive DSP operations require reading a program instruction and multiple data items during each instruction cycle for best performance
Specialized addressing modes	Efficient handling of data arrays and first-in, first- out buffers in memory
Specialized program control	Efficient control of loops for many iterative DSP algorithms. Fast interrupt handling for frequent I/O operations.
On-chip peripherals and I/O interfaces	On-chip peripherals like A/D converters allow for small low cost system designs. Similarly I/O interfaces tailored for common peripherals allow clean interfaces to off-chip I/O devices.

PROCEDURE:

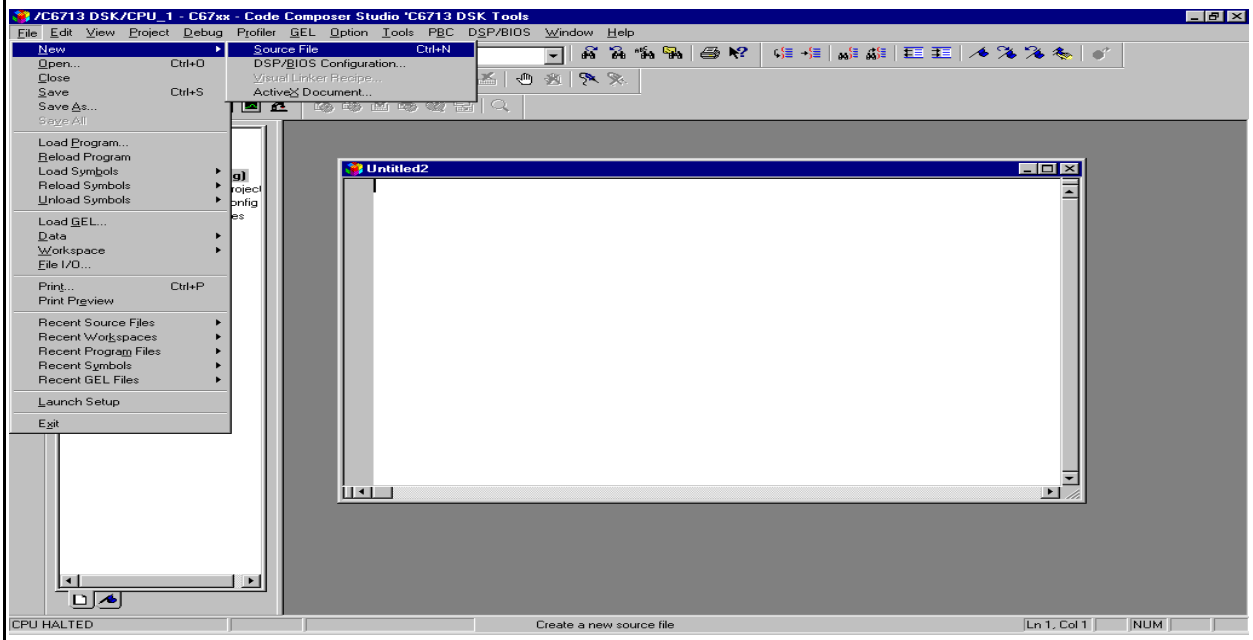
1. To create a New Project

*Project → New **(SUM.pjt)***



2. To Create a Source file

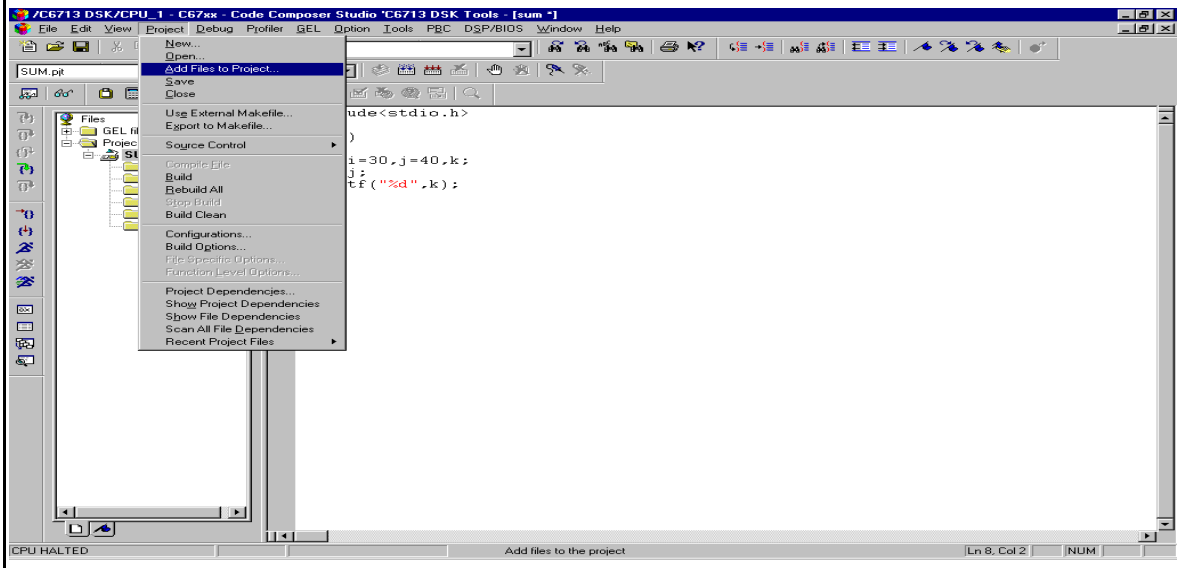
File → New

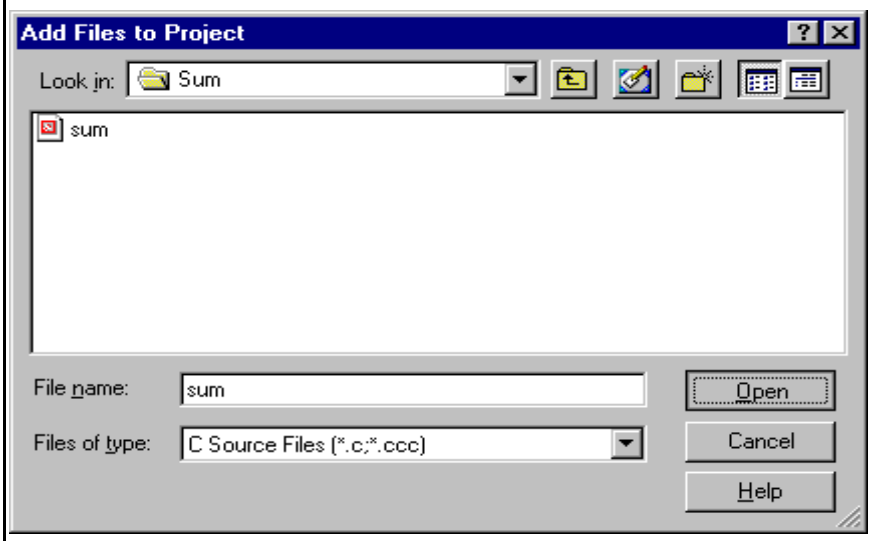


*Type the code (Save & give a name to file, Eg: **sum.c**).*

3. To Add Source files to Project

Project → Add files to Project → sum.c

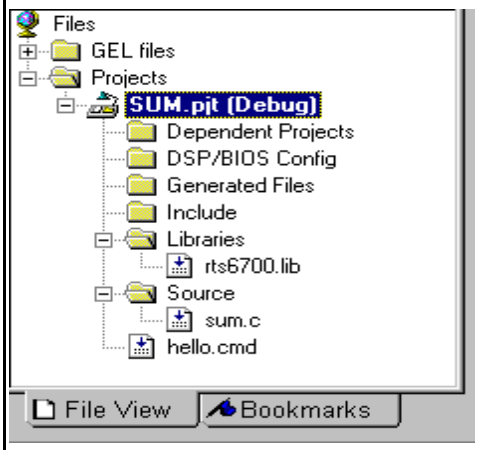




4. To Add rts6700.lib file & hello.cmd:

Project → Add files to Project →rts6700.lib
Path: c:\CCStudio\c6000\cgtools\lib\rts6700.lib
Note: Select Object & Library in(*.o,*.l) in Type of files

Project → Add files to Project →hello.cmd
Path: c:\ti\tutorial\dsk6713\hello1\hello.cmd
Note: Select Linker Command file(*.cmd) in Type of files



. To Compile:

Project → Compile File

6. To build or Link:

Project → build,
Which will create the final executable ***(.out)*** *file.(Eg. sum.out).*

7. Procedure to Load and Run program:

Load program to DSK:

File → Load program → sum. out

8. To execute project:

Debug → Run.

Expt. No:	Perform MAC operation using various Addressing Modes	Date:

AIM:

To study about direct, indirect and immediate addressing modes in TMS320C50 debugger.

EQUIPMENTS REQUIRED:

1. System with TMS 320C50 debugger software
2. TMS 320C50 Kit.

ALGORITHM:

IMMEDIATE ADDRESSING MODE:

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

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:

//Program for immediate addressing mode

```
 .mmregs
 .text
START
:
 LDP #100H EC 56-Digital Signal Processing Lab
 LACC #1241H
ADD #1200H
SACL 2H
H: B H
```

//Program for direct addressing mode

```
.mmregs
.text
START
:
  LDP #100H
  LACC 0H
  ADD 1H
  SACL 2H
H: B H
```

//Program for adding two numbers with indirect addressing mode.

```
.mmregs
.text
START
:
 LAR AR0,#8000H
 MAR *,AR0
 LACC *+,0 ;WITH ZERO
 SHIFT ADD *+
 SACL
*+ H: B
H
```

CONCLUSION:

Thus the Program which illustrates various addressing modes were executed with DSP debugger software.

Expt. No:	Generation of various Signals and Random Noise.	Date:

AIM:

To generation of different waveforms for Sine, Cosine, Exponential and Random noise signals using DSP processor C6713.

EQUIPMENTS:

CCS studio,PC,C6713 Kit

PROGRAM:

a) SINE WAVE GENERATION

```
#include   <stdio.h>
#include   <math.h>
int t,f;
float pi=3.14;
float x[256];
void main()
{
f=8000;
for(t=0; t<=256; t++)
{
x[t]=sin(2*pi*f*t);
printf("%f \n",x[t]);
}
}
```

b) COSINE WAVE GENERATION

```
#include   <stdio.h>
#include   <math.h>
int t,f;
float pi=3.14;
float x[256];
void main()
{
f=8000;
```

```
for(t=0; t<=256; t++)
{
x[t]=cos(2*pi*f*t);
printf("%f \n",x[t]);
}
}
```

C) EXPONENTIAL WAVEFORM:

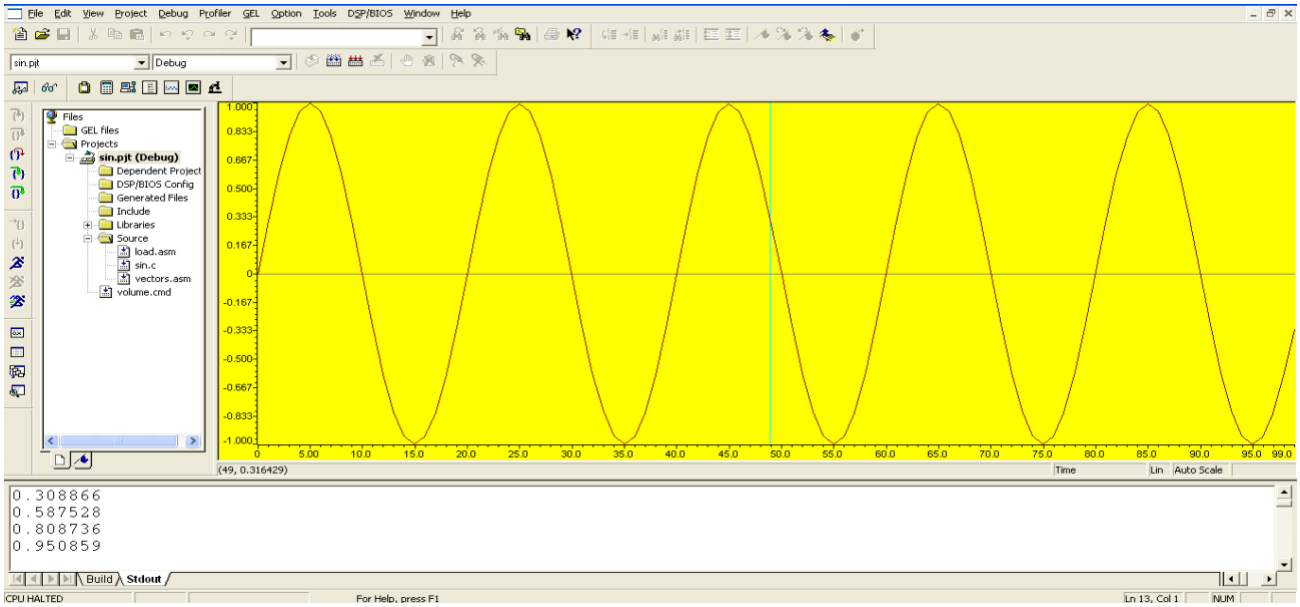
```
#include<stdio.h>
#include<math.h>
int i;
float a=0.2;
float ex[200];
void main()
{
for(i=0;i<=150;i++)
{
ex[i]= exp(a*i);
printf("The exponential wave :%f\n ",ex[i]);
}
}
```

D) RANDOM NOISE WAVEFORM:

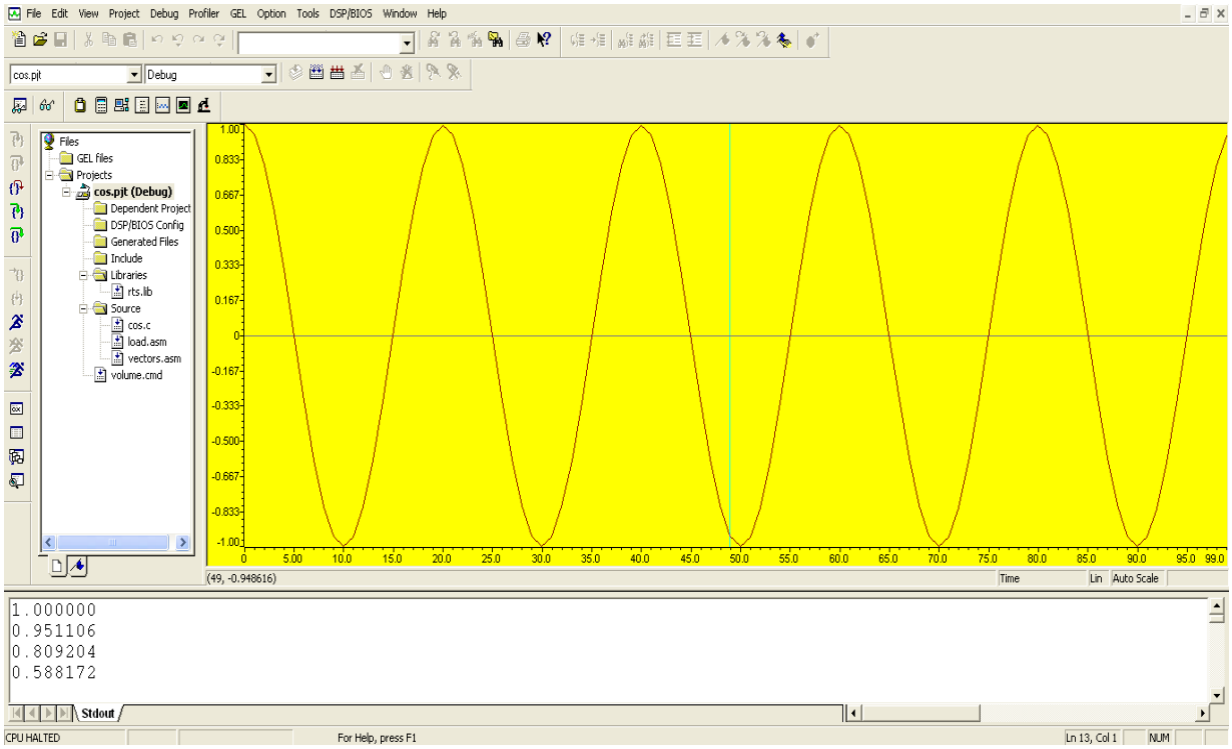
```
#include<stdio.h>
#include<math.h>
int i;
float  a=0.2;
float r[256];
void main()
{
for(i=0;i<=256;i++)
{
r[i]= 0.5* sin(2*3.14*1000*i)+0.02*cos(2*3.14*5000*i)+exp(a*i);
printf("The exponential wave :%f\n ",r[i]);
}
}
```

OUTPUT:

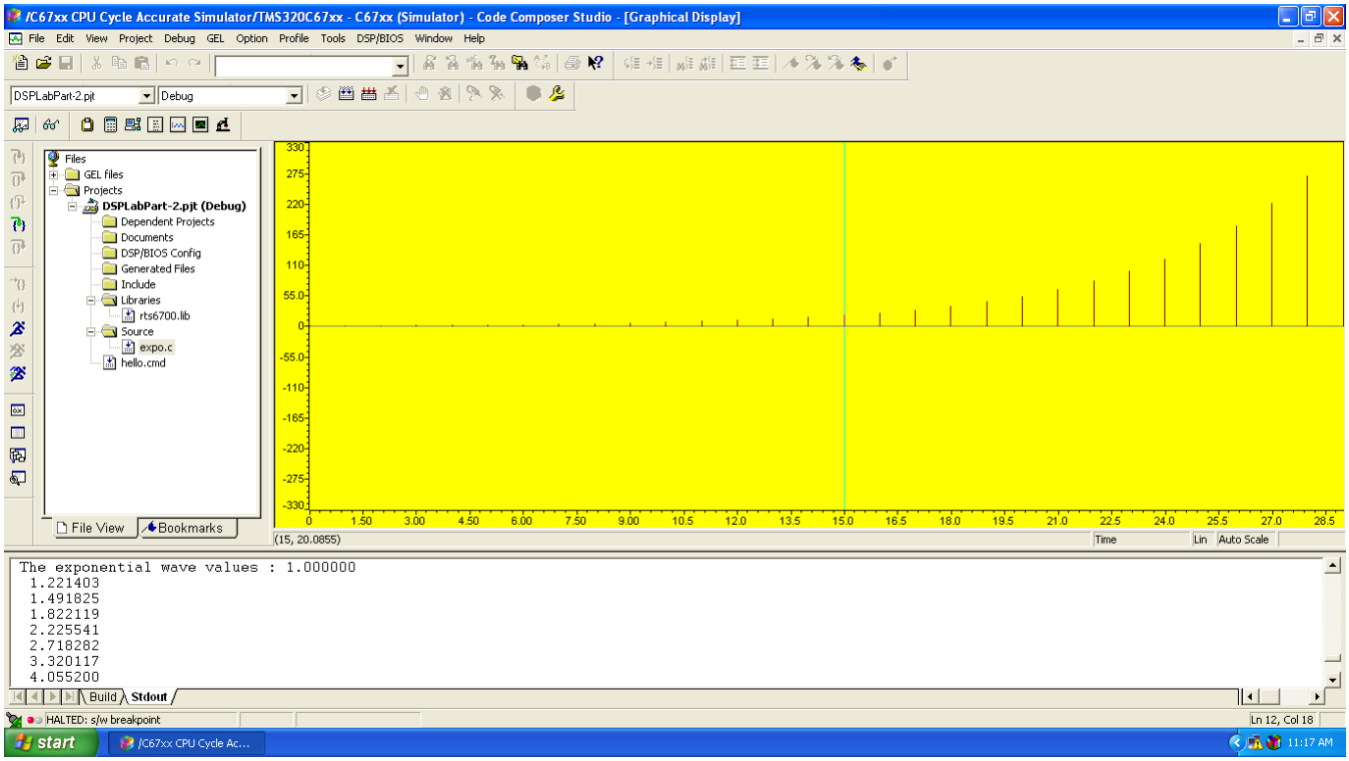
SINE WAVE GENERATION



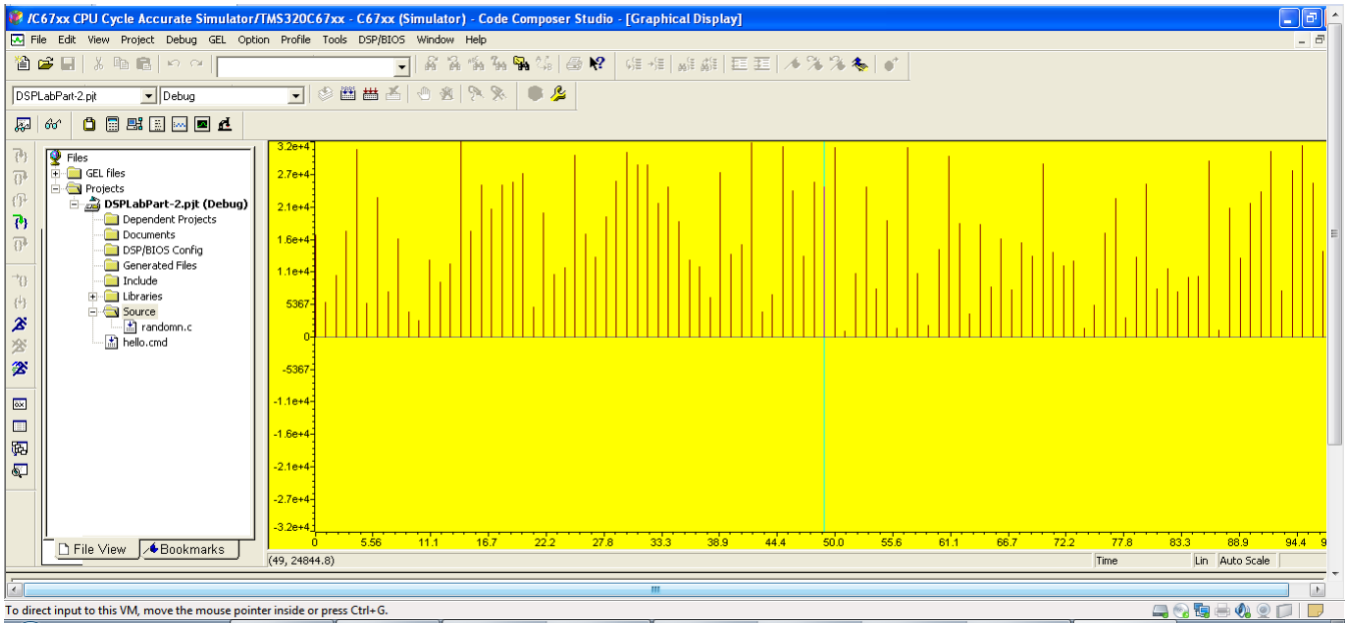
COSINE WAVE GENERATION



EXPONENTIAL WAVEFORM:



RANDOM NOISE WAVEFORM:



CONCLUSION:

Thus the generation of different waveform were implemented by using TMS320C6713.

Expt. No:	Design and demonstration of FIR Filter for Low pass, High pass, Band pass and Band stop filtering.	Date:

AIM

To verify FIR filter design parameters using C6713 Kit.

EQUIPMENTS:

Operating System – Windows XP
Constructor - Simulator and TMS kit
Software - CCStudio 3

PROGRAM:

(a). For Rectangular and Triangular windows:

```
#include<stdio.h>
#include<math.h>
#define pi 3.1415
int n,N,c;
float wr[64],wt[64];
void main()
{
printf("\n enter no. of samples,N= :");
scanf("%d",&N);

printf("\n enter choice of window function\n 1.rect \n 2. triang \n c= :"); scanf("%d",&c);
printf("\n elements of window function are:");
switch(c)
{
case 1:
for(n=0;n<=N-1;n++)
{
wr[n]=1;
printf(" \n wr[%d]=%f",n,wr[n]);
}
break;
case 2:
for(n=0;n<=N-1;n++)
{
```

```
wt[n]=1-(2*(float)n/(N-1));
printf("\n wt[%d]=%f",n,wt[n]);
}
break;
}
}
```

(b). FIR Filter Design using Hamming Window

```
#include<stdio.h>
#include<math.h>
#define pi 3.1415
int N,n;
float H[50],h[50],WH[50];
main()
{
printf("enter the length of the filter coefficients for h(n) and hamming window:");
scanf("%d",&N);
printf("\n calculated filter coefficients are:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
h[n]=0.0;
if(n==0)
h[n]=(float)1/3;
else
h[n]=sin(n*180/3)/(n*pi);
printf("h[%d]=%f\n",n,h[n]);
}
printf("\n calculated hamming window coefficient are :\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
WH[n]=0.0;
WH[n]=0.54+0.46*cos(2*pi*n/(N-1));
printf("WH[%d]=%f\n",n,WH[n]);
}
printf("\n the final FIR filter coefficient after window:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
H[n]=0.0;
H[n]=h[n]*WH[n];
printf("H[%d]=%f\n",n,H[n]);
}
}
```

(c). FIR Filter Design using Hannming Window

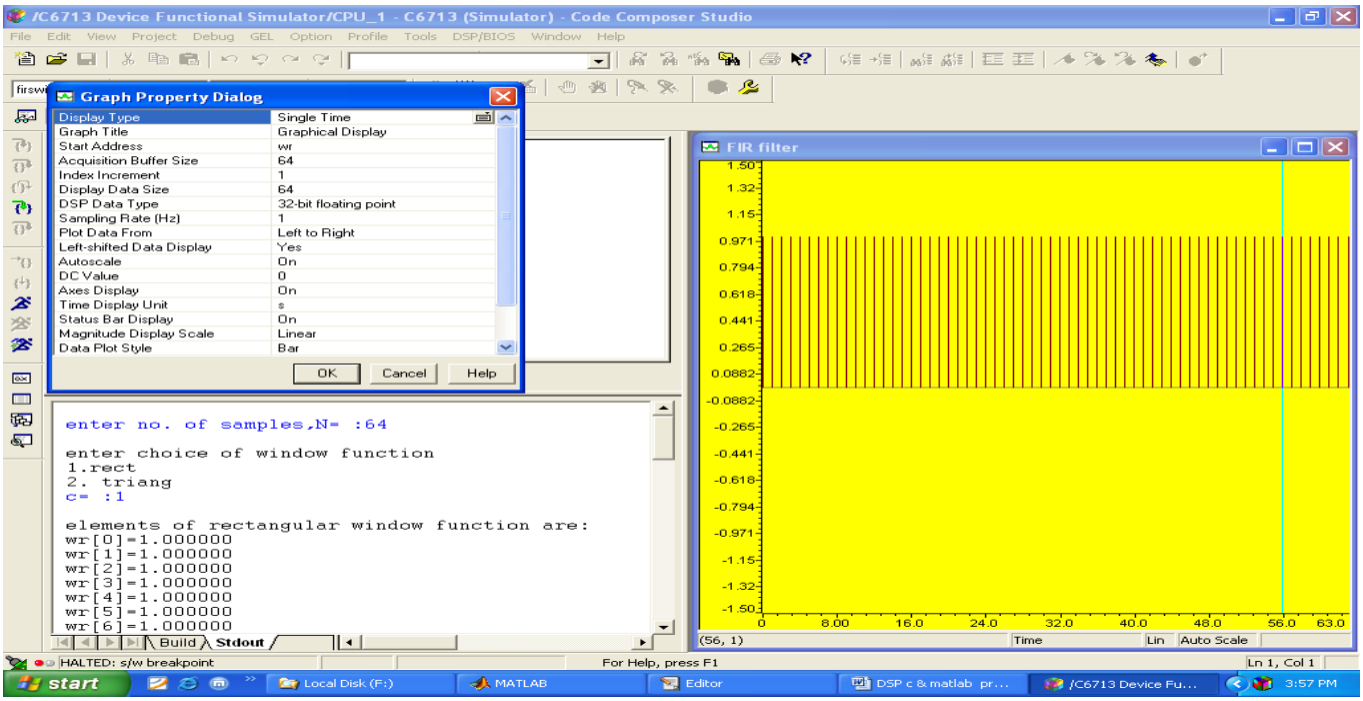
```
#include<stdio.h>
#include<math.h>
#define pi 3.1415
int N,n;
float H[50],h[50],wh[50];
main()
{
printf("enter the length of filter coefficient for h[n] and a hamming window:");
scanf("%d",&N);
printf("\n calculted filter coefficient are:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
h[n]=0.0;
if(n==0)
h[n]=(float)1/3;
else
h[n]=sin(n*180/3)/(n*pi);
printf("h[%d]=%f\n",n,h[n]);
}
printf("\n calculated hanning window coeffcient are:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
wh[n]=0.0;
wh[n]=0.5+0.5*cos(2*pi*n/(N-1));
printf("wh[%d]=%f\n",n,wh[n]);
}
printf("\n final filter coefficient after windowing:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
H[n]=0.0;
H[n]=h[n]*wh[n];
printf("H[%d]=%f\n",n,H[n]);
}
}
```

(d). FIR Filter Design using Blackman Window

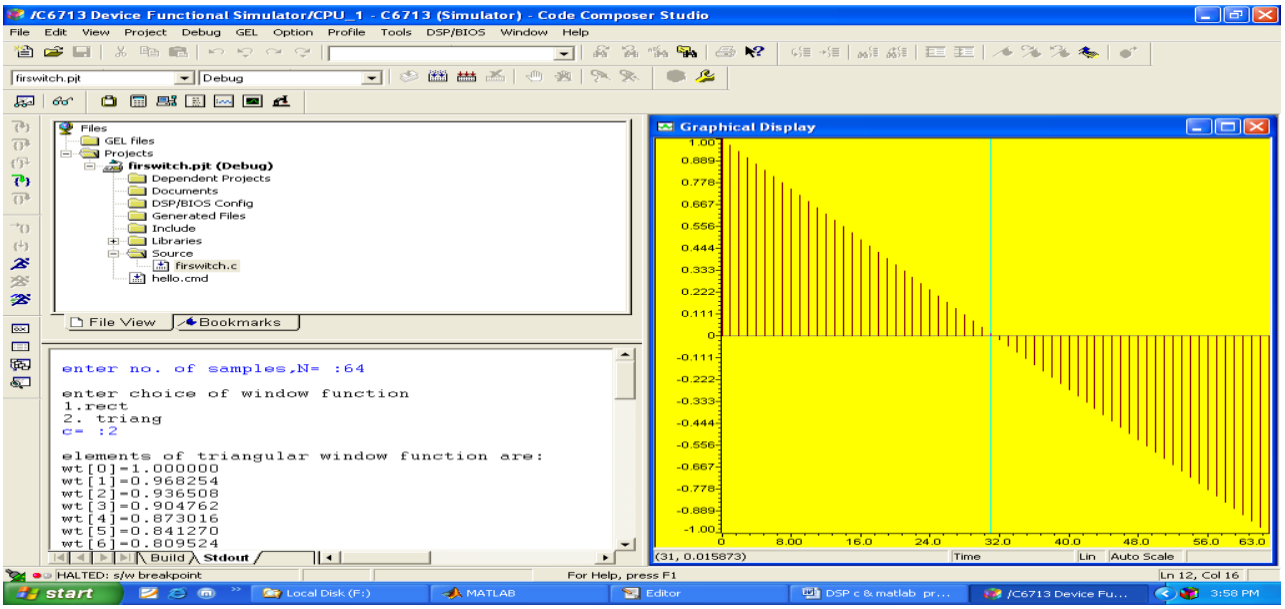
```
#include<stdio.h>
#include<math.h>
#define pi 3.1415
int N,n;
float H[50],h[50],wb[50];
main()
{
printf("enter the length of Fir filter coefficient h[n] and a black man window:\n");
scanf("%d",&N);
printf("\n calculated filter coefficient are:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
h[n]=0.0;
if(n==0)
h[n]=(float)1/3;
else
h[n]=sin(n*180/3)/(n*pi);
printf("h[%d]=%f\n",n,h[n]);
}
printf("calculated blackman window coefficient are:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
wb[n]=0.0;
wb[n]=0.42+0.5*cos(2*pi*n/(N-1))+0.08*cos(4*pi*n/(N-1));
printf("wb[%d]=%f\n",n,wb[n]);
}
printf("\nThe final fir filter coefficient after windowing:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
H[n]=0.0;
H[n]=h[n]*wb[n];
printf("H[%d]=%f\n",n,H[n]);
}
}
```

OUTPUT:

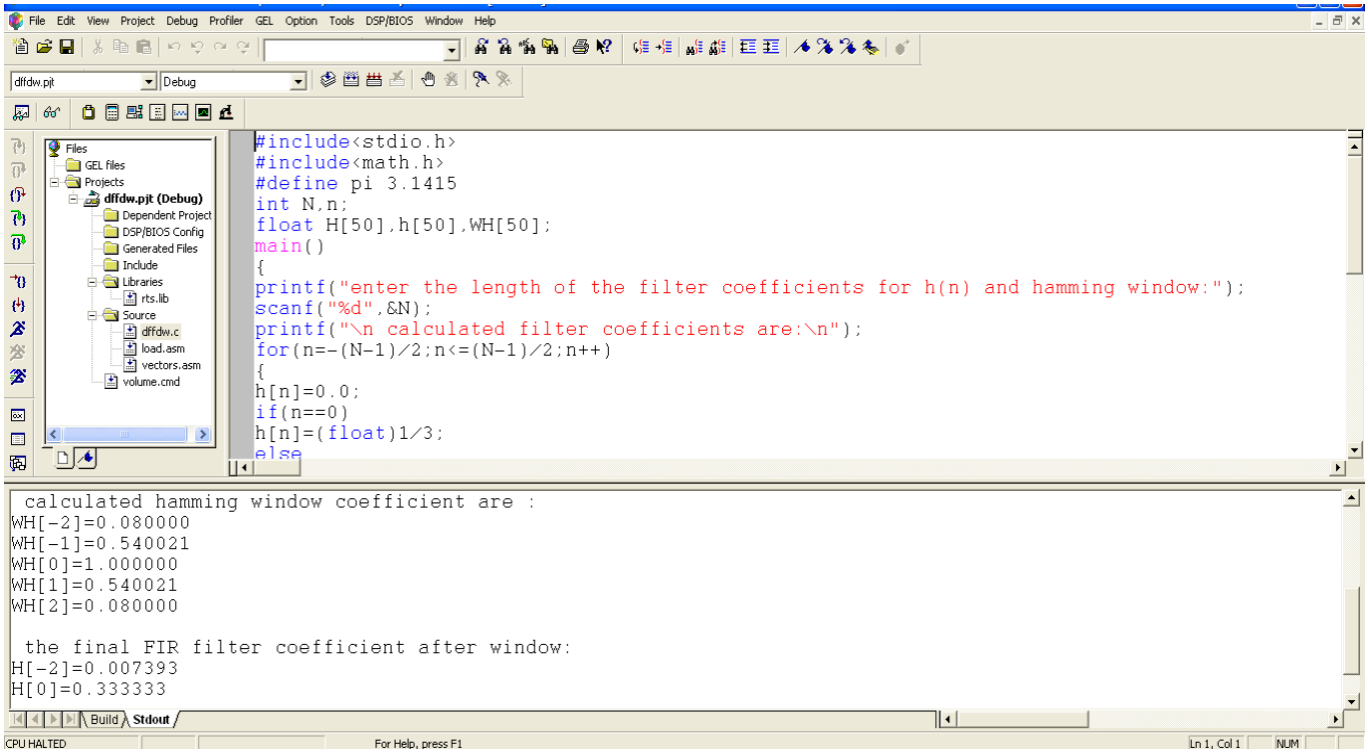
Rectangular Windows



Triangular windows



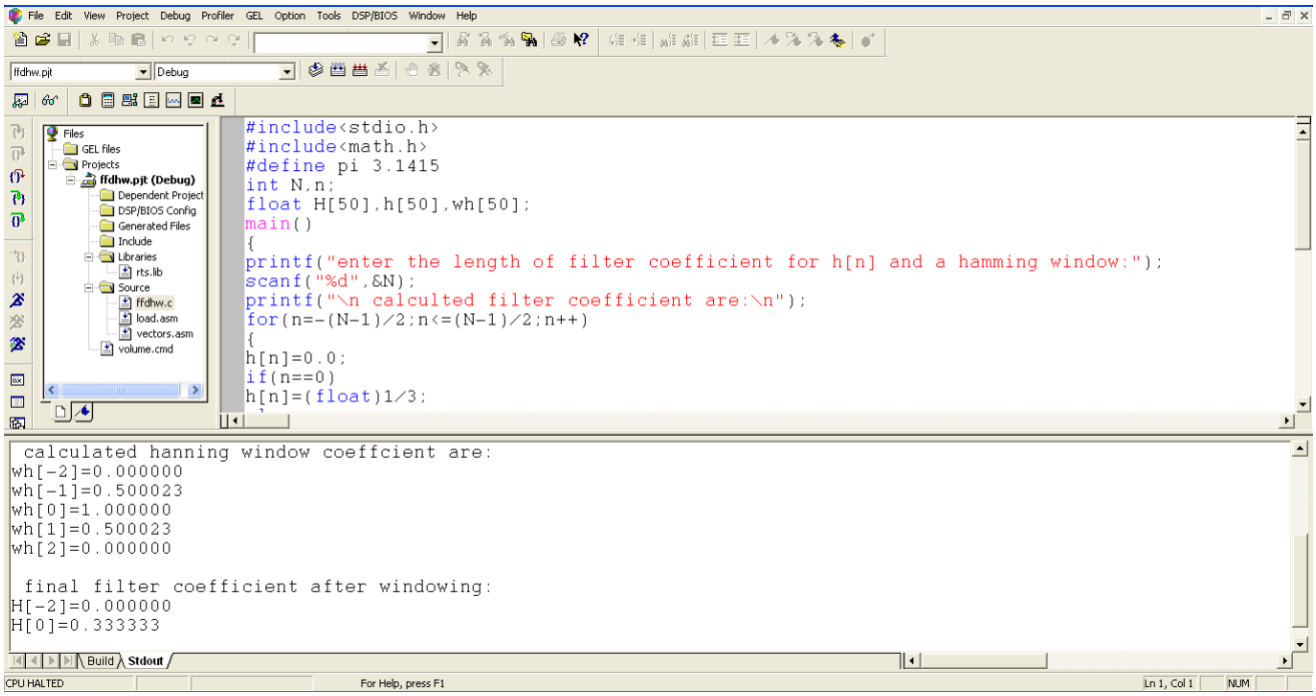
FIR Filter Design using Hamming Window



```
#include<stdio.h>
#include<math.h>
#define pi 3.1415
int N,n;
float H[50],h[50],WH[50];
main()
{
printf("enter the length of the filter coefficients for h(n) and hamming window:");
scanf("%d",&N);
printf("\n calculated filter coefficients are:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
h[n]=0.0;
if(n==0)
h[n]=(float)1/3;
else
```

```
 calculated hamming window coefficient are :
WH[-2]=0.080000
WH[-1]=0.540021
WH[0]=1.000000
WH[1]=0.540021
WH[2]=0.080000

 the final FIR filter coefficient after window:
H[-2]=0.007393
H[0]=0.333333
```

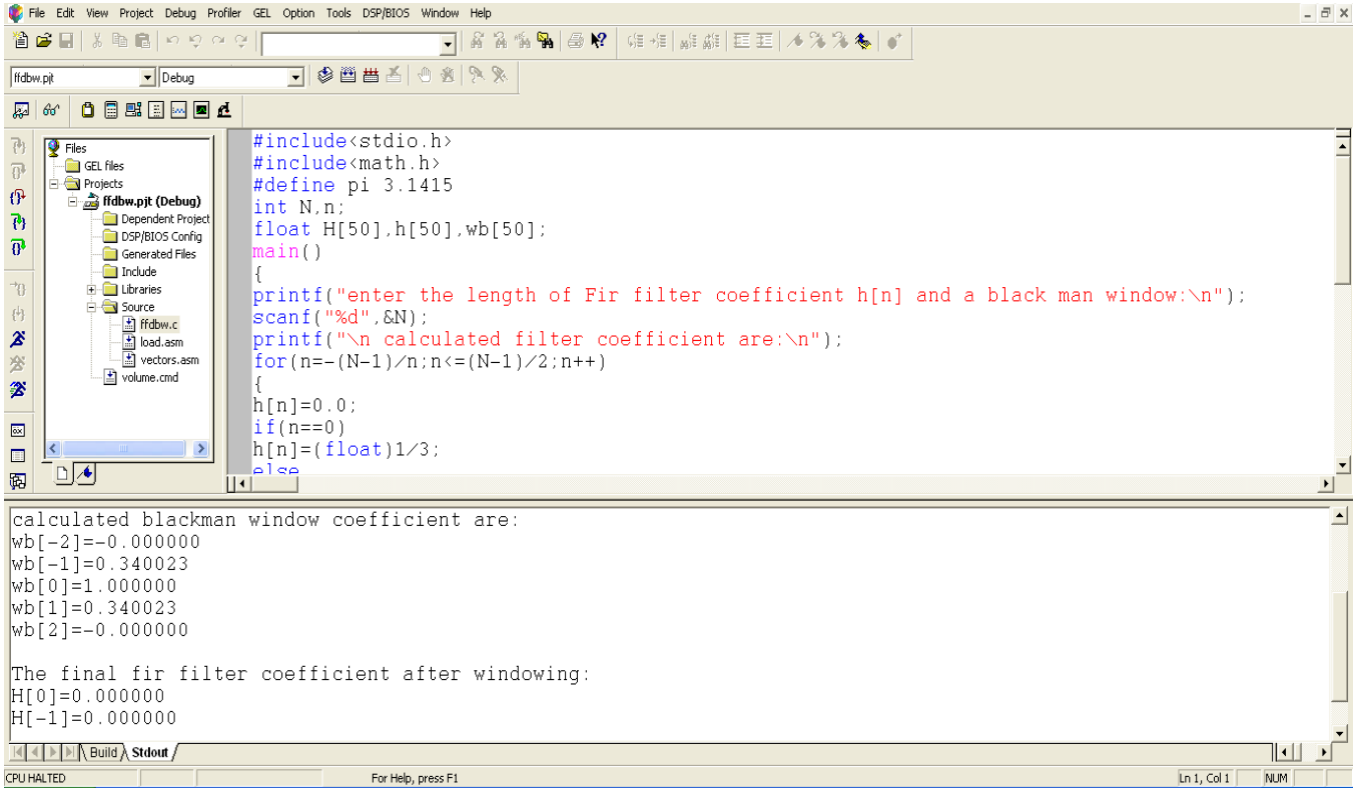
FIR Filter Design using Hanning Window



```
#include<stdio.h>
#include<math.h>
#define pi 3.1415
int N,n;
float H[50],h[50],wh[50];
main()
{
printf("enter the length of filter coefficient for h[n] and a hamming window:");
scanf("%d",&N);
printf("\n calculted filter coefficient are:\n");
for(n=-(N-1)/2;n<=(N-1)/2;n++)
{
h[n]=0.0;
if(n==0)
h[n]=(float)1/3;
```

```
 calculated hanning window coeffcient are:
wh[-2]=0.000000
wh[-1]=0.500023
wh[0]=1.000000
wh[1]=0.500023
wh[2]=0.000000

 final filter coefficient after windowing:
H[-2]=0.000000
H[0]=0.333333
```

FIR Filter Design using Blackman Window



CONCLUSION:

Thus, the FIR filters were designed by using TMS320C6713.

Expt. No:	Design of IIR Filters	Date:

AIM

To verify IIR filter design parameters using C6713 Kit.

EQUIPMENTS:

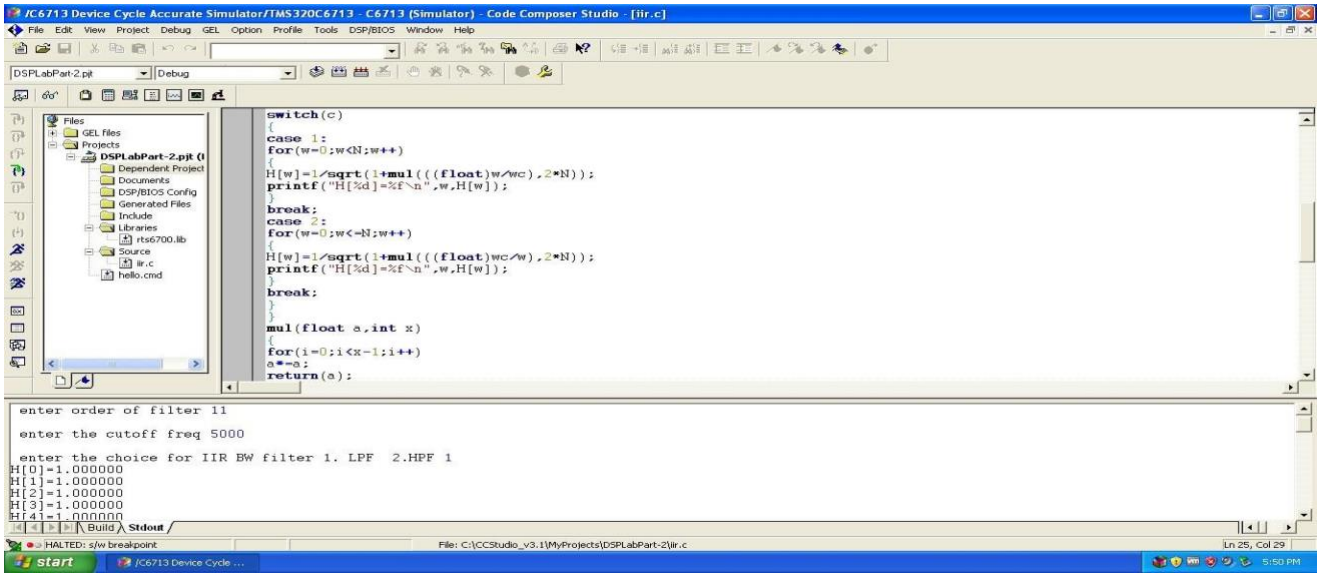
Operating System – Windows XP
Constructor - Simulator and TMS 6713 kit
Software - CCStudio v.3.1

PROGRAM:

```
//IIRFILTERS
#include<stdio.h>
#include<math.h>
inti,w,wc,c,N;
float H[100];
float mul(float, int);
void main()
{
printf("\n enter order of filter ");
scanf("%d",&N);
printf("\n enter the cutoff freq ");
scanf("%d",&wc);
printf("\n enter the choice for IIR filter 1. LPF 2.HPF ");
scanf("%d",&c);
switch(c)
{
case 1:
for(w=0;w<100;w++)
{
H[w]=1/sqrt(1+mul((w/(float)wc),2*N));
printf("H[%d]=%f\n",w,H[w]);
}
break;
```

```
case 2:
for(w=0;w<=100;w++)
{
H[w]=1/sqrt(1+mul((float)wc/w,2*N));
printf("H[%d]=%f\n",w,H[w]);
}
break;
}
}
float mul(float a,int x)
{
for(i=0;i<x-1;i++)
a*=a;
return(a);
}
```

OUTPUT:



CONCLUSION:

Thus, the IIR filters were designed by using TMS320C6713.

Expt. No:	Implement an Up-sampling and Down-sampling operation in DSP Processor	Date:

AIM

To verify the performance of Up-Sampling and Down-Sampling process using C6713 Kit.

EQUIPMENTS:

Operating System – Windows XP
Constructor - Simulator and TMS 6713 kit
Software - CCStudio v.3.1

PROGRAM:

Up-Sampling:

```
#include<stdio.h>
#include<math.h>
float out1[200], out2[200];
float amp=5,freq=50;
float fs=5000, t=0.0;
int i,j,k,L;
void main()
{
L=100;
k=2;
for(i=0;i<=L;i++)
{
out1[i]=amp*sin(2*3.14*freq*t); t=t+1/fs;
}
for(i=0;i<=(k*L);i++)
{
out2[i]=0;
}
i=0;
for(j=0;j<=(k*L);j=k+j)
{
out2[j]=out1[i];
i++;
}
for(j=0;j<=(k*L);j++)
{
```

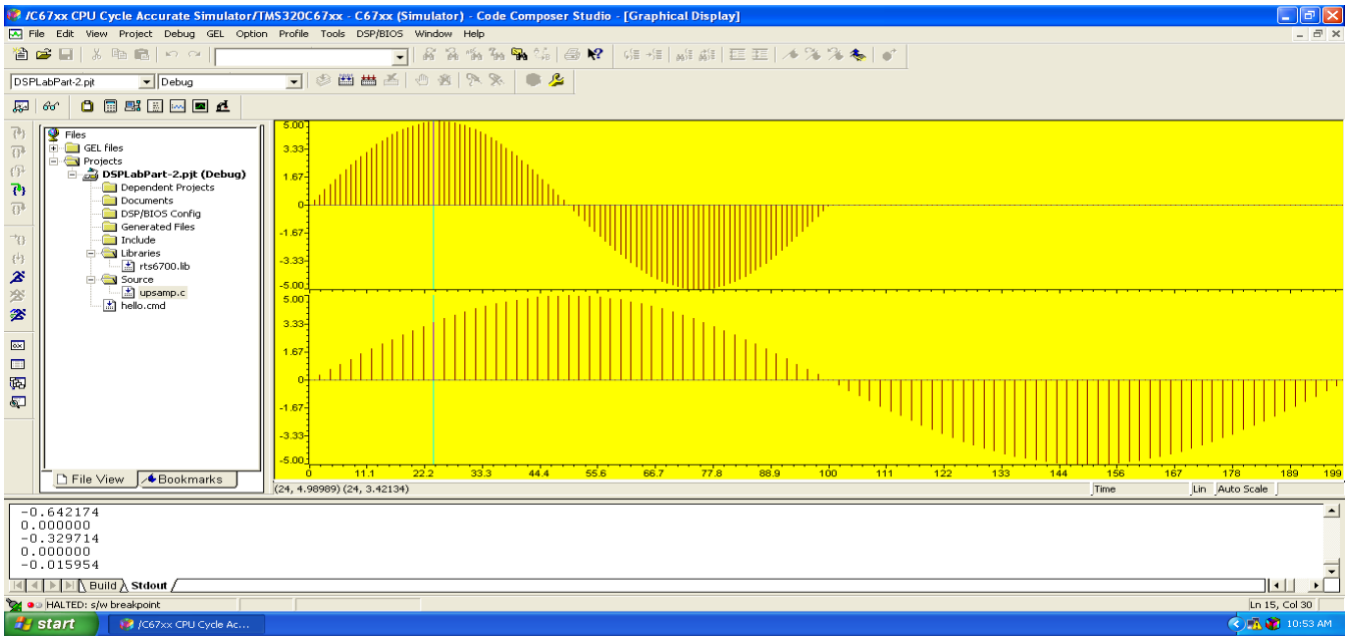
```
printf("\n %f",out2[j]);
}

}
```

Down-Sampling:

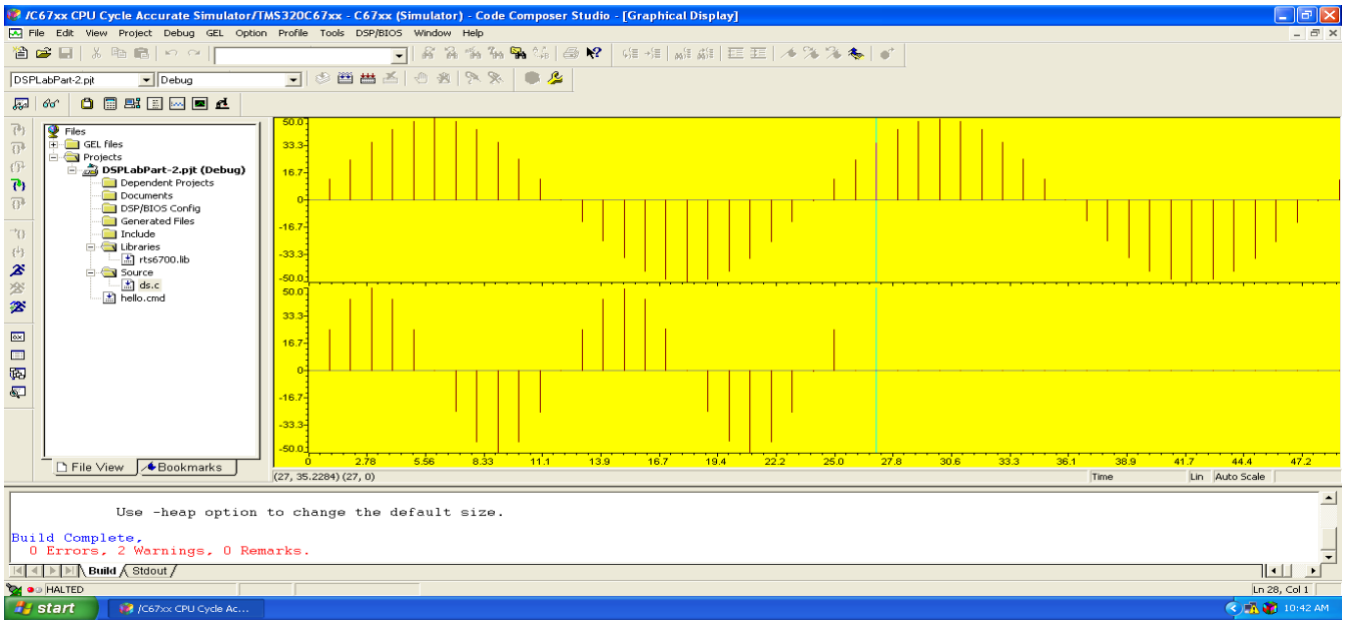
```
#include<stdio.h>
#include<math.h> float
s1[100],s2[50]; float
amp=50,freq=50; float
fs=1200;
float t=0.0;
int i,j,L,k=2;
void main()
{
L=100;
for(i=0;i<=100;i++)
{
s1[i]=amp*sin(2*3.14*freq*t);
t=t+(1/fs);
}
for(i=0;i<=(L/k);i++)
{
s2[i]=0;
}
i=0;
for(j=0;j<=(L/k);j=j+k)
{
s2[i]=s1[j];
i++;
printf("down Sampled Length: %f\n",s2[i]);
}
}
```

OUTPUT:

UpSampling:



Down-Sampling:



CONCLUSION:

Thus the performance of Up-sampling and Down-sampling is verified by using TMS320C67xx.