10 YEARS OF UNIVERSITY RECOGNITION

20 YEARS OF ACADEMIC EXCELLENCE

REVA UNIVERSITY

Bengaluru, India

School of Electronics and Communication Engineering

Program

B.Tech. in ECE

Digital Signal Processing lab

LABORATORY MANUAL

B20EN0605

VI Semester
2020-24

Rukmini Educational Charitable Trust

www.reva.edu.in

Vision of the University

''REVA University aspires to become an innovative university by developing excellent human resources with leadership qualities, ethical and moral values, research culture and innovative skills through higher education of global standards"

Mission of the University

- To create excellent infrastructure facilities and state-of-the-art laboratories and incubation centres
- To provide student-centric learning environment through innovative pedagogy and education reforms
- To encourage research and entrepreneurship through collaborations and extension activities
- To promote industry-institute partnerships and share knowledge for innovation and development
- To organize society development programs for knowledge enhancement in thrust areas
- To enhance leadership qualities among the youth and enrich personality traits, promote patriotism and moral values.

Vision of the School

The School of Electronics and Communication Engineering is envisioned to be a leading centre of higher learning with academic excellence in the field of electronics and communication engineering blended by research and innovation in tune with changing technological and cultural challenges supported with leadership qualities, ethical and moral values.

Mission of the School

- Establish a unique learning environment to enable the students to face the challenges in the field of Electronics and Communication Engineering and explore multidisciplinary which serve the societal requirements.
- Create state-of-the-art laboratories, resources, and exposure to the current industrial trends to enable students to develop skills for solving complex technological problems of current times and provide a framework for promoting collaborative and multidisciplinary activities.
- Promote the establishment of Centres of Excellence in niche technology areas to nurture the spirit of innovation and creativity among faculty and students.
- Offer ethical and moral value-based education by promoting activities which inculcate the leadership qualities, patriotism and set high benchmarks to serve the society

Program Educational Objectives (PEOs)

The Program Educational Objectives of B. Tech in Electronics and Communication Engineering are as follows:

PEO -1: To have successful professional careers in industry, government, academia, and military as innovative engineers.

PEO -2: To successfully solve engineering problems associated with the lifecycle of Electronics and Communication Systems either leading a team or as a team member.

PEO -3: To continue to learn and advance their careers through activities such as participation in professional organizations, attainment of professional certification for lifelong learning and seeking higher education.

PEO -4: To be active members ready to serve the society locally and internationally and will undertake entrepreneurship for the growth of economy and to generate employment.

Program Outcomes (POs)

On successful completion of the program, the graduates of B. Tech. (Electronics and Communication Engineering) program will be able to:

- **PO-1: Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals for the solution of complex problems in Electronics and communication Engineering.
- **PO-2: Problem analysis:** Identify, formulate, research literature, and analyze engineering problems to arrive at substantiated conclusions using first principles of mathematics, natural, and engineering sciences.
- **PO-3: Design/development of solutions:** Design solutions for complex engineering problems and design system components, processes to meet the specifications with consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO-4: Conduct investigations of complex problems:** Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO-5: Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO-6: The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO-7: Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO-8: Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO-9: Individual and team work:** Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.
- **PO-10: Communication:** Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports documentation. Make effective presentations, and give and receive clear instructions.
- **PO-11: Project management and finance:** Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments.
- **PO-12: Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Programme Specific Outcomes (PSOs)

On successful completion of the program, the graduates of B. Tech. (Electronics and Communication Engineering) program will be able to:

PSO-1: Isolate and solve complex problems in the domains of Electronics and Communication Engineering using latest hardware and software tools and technologies, along with analytical and managerial skills to arrive at cost effective and optimum solutions either independently or as a team.

PSO-2: Implant the capacity to apply the concepts of electronics, communications, signal processing, VLSI, embedded systems, etc. in the design, development and implementation of application oriented engineering systems.

PSO-3: Design, Model, Analyze and Build Electronics and Communication Systems to solve real life and industry problems.

CONTENTS

SYLLABUS

<table>
<tr><td>Course Title</td><td colspan="4">Digital Signal Processing Lab</td><td colspan="3">Course Type</td><td colspan="2">HC</td></tr>
<tr><td>Course Code</td><td>B20EN0605</td><td>Credits</td><td colspan="2">1</td><td colspan="3">Class</td><td colspan="2">VI Semester</td></tr>
<tr><td rowspan="6">Course Structure</td><td>LTP</td><td>Credits</td><td>Contact Hours</td><td>Work Load</td><td colspan="3" rowspan="2">Total Number of Classes Per Semester</td><td colspan="2" rowspan="2">Assessment in Weightage</td></tr>
<tr><td>Lecture</td><td></td><td></td><td></td></tr>
<tr><td>Tutorial</td><td>-</td><td>-</td><td>-</td><td rowspan="3">Theory</td><td rowspan="3">Practical</td><td rowspan="3">IA</td><td rowspan="3">SEE</td></tr>
<tr><td>Practice</td><td>1</td><td>2</td><td>2</td></tr>
<tr><td>-</td><td>-</td><td>-</td><td>-</td></tr>
<tr><td></td><td>1</td><td>2</td><td>2</td><td>-</td><td>28</td><td>50 %</td><td>50 %</td></tr>
</table>

COURSE OVERVIEW:

The objectives of this course are:

1. Explain the concept of DFT and FFT.
2. Calculate the DFT of a sequence, relate it to the DTFT, and use the DFT to compute the linear convolution of twosequences.
3. Apply the concept of FFT algorithms to compute DFT.
4. Design IIR filter using impulse invariant, bilinear transform.
5. Describe the concept of linear filtering Technique.
6. Demonstrate FIR & IIR filters for digital filter structures.

COURSE OUTCOMES (COs):

On successful completion of this course; the student shall be able to:

CO#	Course Outcomes	POs	PSOs
CO1	Apply DFT for the analysis and interpret the frequency content of Discrete TimeSignal	1,2,3,4,5,9,10	1,2,3
CO2	Calculate the Circular Convolution of Discrete Time Signals and verify the properties of DFT	1,2,3,4,5,9,10	1,2,3
CO3	Compare the two signals by computing correlation in time and frequency domain	1,2,3,4,5,9,10	1,2,3
CO4	Design and analyze IIR and FIR	1,2,3,4,5,9,10	1,2,3
CO5	Integrate CCS studio for real time implementation of DSP Experiments onDSP processor.	1,2,3,4,5,9,10	1,2,3
CO6	Implement the Convolution and DFT computation on DSP processor	1,2,3,4,5,9,10	1,2,3

BLOOM'S LEVEL OF THE COURSE OUTCOMES:

CO#	Bloom's Level					
	Remember (L1)	Understand(L2)	Apply (L3)	Analyze (L4)	Evaluate (L5)	Create (L6)
CO1	✓	✓	✓	✓		
CO2	✓	✓	✓	✓		
CO3	✓	✓	✓	✓		
CO4	✓	✓	✓	✓		
CO5	✓	✓	✓	✓		
CO6	✓	✓	✓	✓		

COURSE ARTICULATION MATRIX:

CO#/ POs	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1	3	3	3	2	3				3	3			2	2	2
CO2	3	3	3	2	3				3	3			2	2	2
CO3	3	3	3	2	3				3	3			2	2	2
CO4	3	3	3	2	3				3	3			2	2	2
CO5	3	3	3	2	3				3	3			2	2	2
CO6	3	3	3	2	3				3	3			2	2	2

Note: 1-Low, 2-Medium, 3-High

List of Challenging Experiments:

1.	Analysis of continuous time and discrete time signals.
2.	Consider a symmetric square wave with frequency 100 Hz. Plot the 4-term, 10-term and 25-term Fourier series approximations. Compare the FS Approximations with the actual square wave. Observe the approximation behavior at the points of discontinuity.
3.	Study the effects of signal length and windowing on the spectrum of a signal Computed with FFT.
4.	Plot the frequency response and impulse response of an ideal discrete-time Low-pass filter.
5.	Generate a sinusoidal signal which contains 50Hz, 70Hz, 100Hz and 120Hz frequencies. Analyze the frequency components present in the signal with and without AWGN for a SNR of 0.6. Obtain the plot and comment on the results.
6.	Signal processing methods for Music Signals using DSP Processor
7.	Signal processing mechanisms for Bio-Signals using DSP processor

List of Experiments Using MATLab / Octave:

Sl. No.	Name of the Practice Session	Tools and Techniques	Expected Skill /Ability
1	Perform the Linear convolution of any two given sequences in timedomain.	MATLab/Octave	Design and simulation Working in a team
2	Computation of N point DFT of a given sequence using the definitionof DFT and plot magnitude and phase spectrum, and verify using built in function (using FFT).	MATLab/Octave	Design and simulation Working in a team
3	Perform the Circular convolution of two given sequences in timedomain.	MATLab/Octave	Design and simulation Working in a team
4	Perform Circular convolution of any two given sequences infrequency domain by using DFT and IDFT.	MATLab/Octave	Design and simulation Working in a team
5	Obtain the Auto correlation and cross correlation of a givensequence and verify its properties.	MATLab/Octave	Design and simulation Working in a team
6	Verification of Sampling theorem.	MATLab/Octave	Design and simulation Working in a team
7	Design of digital Low-pass and High-pass Butterworth IIR filter tomeet the given specifications using Bilinear transformations.	MATLab/Octave	Design and simulation Working in a team
8	Design of digital Low-pass and High-pass Chebyshev IIR filter to meetthe given specifications using Bilinear transformations.	MATLab/Octave	Design and simulation Design and simulation Working in a team
9	Design of digital Low-pass FIR filter to meet the given specificationsusing windowing technique.	MATLab/Octave	Design and simulation Working in a team
	List of Experiments using DSP Processor:		
10	Linear convolution of two given sequences.	DSP Processor andCCS Studio	Design and simulation Working in a team
11	Circular convolution of two given sequences.	DSP Processor and CCS Studio	Design and simulation Working in a team
12	Computation of N-point DFT of a given sequence.	DSP Processor andCCS Studio	Design and simulation Working in a team
13	Solving a linear constant coefficient difference equation.	DSP Processor andCCS Studio	Design and simulation Working in a team

TEXT BOOKS:

1. Proakis & Monalakis, Digital signal processing – Principles Algorithms & Applications, PHI, 4th Edition, New Delhi, 2007.

REFERENCE BOOKS:

1. Oppenheim & Schaffer, Discrete Time Signal Processing, PHI, 2003.
2. S.K. Mitra, Digital Signal Processing, Tata Mc-Graw Hill, 2nd Edition, 2004.
3. Sanjit K Mitra, Digital signal Laboratory using MATLAB, MGH Edition.2000.
4. Ashok Ambardar, Digital signal processing: A modern Introduction, Cengage Learning, 2009.

MATLAB

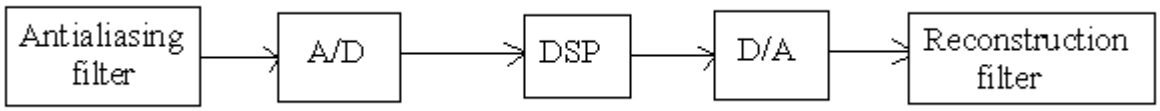
MATLAB is a high-level language and interactive environment that enables you to perform computationally intensive tasks faster than with traditional programming languages such as C, C++, and Fortran.

Introduction to Digital Signal Processing

A digital signal processing system uses a computer or a digital processor to process the signals. The real life signals are analog and therefore must be converted to digital signals before they can be processed with a computer. To convert a signal from analog to digital, an analog to digital (A/D) converter is used. After processing the signal digitally, it is usually converted to an analog signal using a device called a digital-to-analog(D/A) converter. The below block diagram shows the components of a DSP scheme. This figure contains two additional blocks, one is the antialiasing filter for filtering the signal before sampling and the second is the reconstruction filter placed after the D/A converter. The antialiasing filter ensures that the signal to be sampled does not contain any frequency higher than half of the sampling frequency. If such a filter is not used, the high frequency contents sampled with an inadequate sampling rate generate low frequency aliasing noise. The reconstruction filter removes high-frequency noise due to the "staircase" output of the D/A converter.

The signals that occur in a typical digital signal processing scheme are: continuous-time or analog signal, sampled signal sampled-data signal, quantized or digital signal, and the D/A output signal.

An analog signal is a continuous-time, continuous-amplitude signal that occurs in real systems. Such a signal is defined for any time and can have any amplitude within a given range. The sampling process generates a sampled signal. A sampled signal value is held by a hold circuit to allow an A/D converter to change it to the corresponding digital or quantized signal. The signal at the A/D converter input is called a ***sampled data*** signal and at the output is the digital signal. The processed digital signal, as obtained from the digital signal processor, is the input to the D/A converter. The analog output of a D/A converter has "staircase" amplitude due to the conversion process used in such a device. The signal, as obtained from the D/A, can be passed through a reconstruction lowpass filter to remove its high-frequency contents and hence smoothen it.



Block diagram of Digital Signal Processing system

MATLAB BASICS

Sl. No.	Arithmetic operations	MATLAB Expressions
01	2^5	2^5
02	$2^5 / 2^5-1$	2^5 / (2^5-1)
03	$3\frac{\sqrt{5}-1}{(\sqrt{5}+1)^2}$ -1	3*(sqrt(5)-1) / (sqrt(5)+1)^2-1
04	Area=πr^2 with r = $\pi^{1/3}-1$	Area=pi*(pi^(1/3)-1)^2

Sl. No.	Exponential and Logarithms	MATLAB Expressions
01	e^3	exp(3)
02	$\ln(e^3)$	log(exp(3))
03	$\log_{10}(e^3)$	log10(exp(3))
04	$\log_{10}(10^5)$	log10(10^5)
05	$e^{\pi\sqrt{163}}$	exp(pi*sqrt(163))

Sl. No.	Trigonometric functions	MATLAB Expressions
01	sin (π/6)	sin(pi/6)
02	cos (π)	cos(pi)
03	tan (π/2	tan(pi/2)
04	Sin^2 π/6 + $\cos^2$ π/6	(sin(pi/6))^2 + (cos(pi/6))^2

Sl. No.	Complex numbers	MATLAB Expressions
01	(1+3i) /(1-3i)	(1+3i) / (1-3i)
02	$e^{i\pi/4}$	exp(i*pi/4)
03	$e^{i\pi/2}$	exp(pi*i/2) → $e^{\pi/2\ i}$ →cos(π/2)+isin(π/2) = i

Example:

Given a=3; b=2;c=5; d=3;

1. Evaluate the following MATLAB assignment statements:
   a) y = a*b+c*d;
   b) y = a*(b+c)*d;
   c) y = (a*b)+(c*d);
   d) y = a^b^d;
   e) y = a^(b^d);

a) y = a*b+c*d
= 3*2+5*3 => 6+15 => 21 => **y = 21**

b) y = a*(b+c)*d
= 3*(2+5)*3 => 3*7*3 => **y = 63**

c) y = (a*b)+(c*d)
= (3*2)+(5*3) => 6+15 => **y = 21**

d) y = a^b^d;
= 3^2^3 => 9^3 => **y = 729**

e) y = a^(b^d)
= 3^(2^3) => 3^8 => **y = 6561**

Matrices	MATLAB command
1. $A = \begin{bmatrix} 1 & 2 & 5 \\ 3 & 9 & 0 \end{bmatrix}$	A =[1 2 5; 3 9 0];
2 $A = \begin{bmatrix} 1 & 3 & 9 \\ 5 & 10 & 15 \\ 0 & 0 & -5 \end{bmatrix}$	A = [1 3 9;5 10 15; 0 0 -5]; OR $A = \begin{bmatrix} 1 & 3 & 9 \\ 5 & 10 & 15 \\ 0 & 0 & -5 \end{bmatrix}$

Initializing with builtin functions in MATLAB / OCTAVE	MATLAB Expression	Comments
1. A = zeros(2);	$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$	Creates 2X2 with 0's
2. B = zeros(2,3);	$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	Creates 2X3 with 0's
3. C = ones(2,3);	$C = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	Creates 2X3 with 1's
4. D = eye(3);	$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Creates 3X3 with diagonal elements as 1
5. E = 4*eye(3);	$E = \begin{bmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{bmatrix}$	Creates 3X3 with diagonal elements as 4

Array operations:

1. .* => Element by element multiplication
2. ./ => Element by element left division
3. .\ => Element by element right division
4. .^ => Element by element exponentiation

MATRIX PROBLEMS:

1. Assume

$$A=\begin{bmatrix}1 & 0\\ 2 & 1\end{bmatrix} \quad B=\begin{bmatrix}-1 & 2\\ 0 & 1\end{bmatrix} \quad C=\begin{bmatrix}3\\ 2\end{bmatrix} \quad D=\begin{bmatrix}5\end{bmatrix}$$

A) $A+B=\begin{bmatrix}0 & 2\\ 2 & 2\end{bmatrix}$

B) $A.*B=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}$ $\quad A=\begin{bmatrix}1 & 0\\ 2 & 1\end{bmatrix} \quad B=\begin{bmatrix}-1 & 2\\ 0 & 1\end{bmatrix}$

C) $A*C=\begin{bmatrix}3+0\\ 6+2\end{bmatrix}=\begin{bmatrix}3\\ 8\end{bmatrix}$ $\quad A=\begin{bmatrix}1 & 0\\ 2 & 1\end{bmatrix} \quad C=\begin{bmatrix}3\\ 2\end{bmatrix}$

Simple x, y Plots

MATLAB Code for the given equation $y= x^2-10x+15$

```
clc;
x=0:1:10;
y=x.^2-10.*x+15;
plot(x,y);                    % plots the value x and y in continuous form
xlabel('x');
ylabel('y');
title('plot of y vs x');
grid on;
```

MATLAB Code for the given equation $y=e^{x/10} \sin(x)$

```
clc;
x=0:0.1:20;
y=exp(0.1*x).*sin(x);
plot(x,y);
xlabel('time in seconds');
ylabel('Amplitude in volts');
title('plot of y vs x');
```

MATLAB Code for the given equation $y= A \sin(2\pi ft)$

```
clc;
t=0:0.00001:0.01;
A=10;
f=200;
y=A*sin(2*pi*f*t);
plot(t,y);
xlabel('time axis');
ylabel('Amplitude ');
title('sine wave plot');
```

MATLAB Code for the given equation y= A cos(2πft)

```
clc;
t=0:0.00001:0.01;
A=10;
f=200;
y=A*cos(2*pi*f*t);
plot(t,y);
xlabel('time axis');
ylabel('Amplitude ');
title('cosine wave plot');
```

MATLAB Code to plot both sine and cosine in one figure window

```
clc;
t=0:0.00001:0.01;
A=10;
f=200;
y1=A*sin(2*pi*f*t);
y2=A*cos(2*pi*f*t);
plot(t,y1,t,y2);
xlabel('time axis');
ylabel('Amplitude ');
title('sine/cosine wave plot');
```

MATLAB Code to plot both sine and cosine in separate figure window

```
clc;
t=0:0.00001:0.01;
A=10;
f=200;
y1=A*sin(2*pi*f*t);
y2=A*cos(2*pi*f*t);
figure(1);
plot(t,y1);
xlabel('time axis');
ylabel('Amplitude ');
title('sine wave plot');
```

```
figure(2);
plot(t,y2);
xlabel('time axis');
ylabel('Amplitude ');
title('cosine wave plot');
```

MATLAB Code to plot both sine and cosine by dividing the figure window:

```
clc;
t=0:0.00001:0.01;
A=10;
f=200;
y1=A*sin(2*pi*f*t);
y2=A*cos(2*pi*f*t);

subplot(2,1,1);     % Divides the figure window into 2 rows, 1 column and 1st position respectively
plot(t,y1);         % plots the waveform in continuous form
xlabel('time axis');
ylabel('Amplitude ');
title('sine wave plot');

subplot(2,1,2);     % Divides the figure window into 2 rows, 1 column and 2nd position respectively
plot(t,y2);
xlabel('time axis');
ylabel('Amplitude ');
title('cosine wave plot');
```

EXPERIMENT -01

AIM: - TO COMPUTE THE LINEAR CONVOLUTION OF THE GIVEN INPUT SEQUENCE x (n) & THE IMPULSE RESPONSE OF THE SYSTEM h (n).

Theory:

Linear Convolution:

Linear convolution is the basic operation to calculate the output for any linear time invariant system given its input and its impulse response.

- The output y[n] of a LTI (linear time invariant) system can be obtained by convolving the input x[n] with the system's impulse response h[n].
    - The convolution sum is

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

- x[n] and h[n] can be both finite or infinite duration sequences.
- Even if one (or both) of the sequences is infinite (say, $h[n]=(0.9)^n u[n]$), we can analytically evaluate the convolution formula to get a functional form (closed form solution) of y[n].
- If both the sequences are of finite duration, then we can use the matlab function '***conv***' to evaluate the convolution sum to obtain the output y[n]. Convolution is implemented as polynomial multiplication (refer matlab help).
- The length of y[n] = xlength + hlength -1.
- The conv function assumes that the two sequences begin at n=0 and is invoked by y=conv(x,h).
- Even if one of the sequences begin at other values of n, say n=-3, or n=2; then we need to provide a beginning and end point to y[n] which are ybegin=xbegin+hbegin and yend=xend+hend respectively.

Algorithm:

1. Input the two sequences as x1, x2
2. Convolve both to get output y.
3. Plot the sequences.

MATLAB Implementation:

MATLAB recognizes index 1 to positive maximum. Index 0 is also not recognized. The timing information for a sequence is provided by another vector, say **n=-3:5;** creates a vector with values from -3 to 5 with an increment of 1.

During plotting, the time vector size and the sequence size should be the same, i.e., the number of elements in the sequence x1 and in its time vector n1 should be the same. Similarly for x2 and y.

MATLAB Program:

```
x1 = [3 11 7 0 -1 4 2];
n1 = -3:3;
x2 = [2 3 0 -5 2 1];
n2 = -1:4;
ybegin = min(n1)+min(n2);
yend = max(n1)+max(n2);
ny = [ybegin:yend];
y = conv(x1,x2);
disp('linear con of x1 & x2 is y=');
disp(y);
subplot(3,1,1);
stem(n1,x1);
xlabel('time index n');
ylabel('amplitude ');
title('plot of x1');
subplot(3,1,2);
stem(n2,x2);
xlabel('time index n');
ylabel('amplitude ');
title('plot of x2');
subplot (3,1,3);
stem(ny,y);
xlabel('time index n');
ylabel('amplitude ');
title('convolution output');
```

Calculations:-

x1 = [3 11 7 0 -1 4 2] x2 = [2 3 0 -5 2 1]

↑ ↑

x1*x2 = [3δ(n+3)+11δ(n+2)+7δ(n+1)-1δ(n-1)+4δ(n-2)+2δ(n-3)]*****[2 δ(n+1)+3 δ(n)-5 δ(n-2)+2δ(n-3)+δ(n- 4)]

= 6δ(n+4)+22δ(n+3)+14δ(n+2) - 2δ(n)+8δ(n-1)+4δ(n-2)

+9δ(n+3)+33δ(n+2)+21δ(n+1) - 3δ(n-1)+12δ(n-2)+6δ(n-3)

+6δ(n)+22δ(n-1)+14δ(n-2)

- 2δ(n-4)+8δ(n-5)+4δ(n-6)

+(-15δ(n+1))-55δ(n)-35δ(n-1) +5δ(n-3)-20δ(n-4)-10δ(n-5)

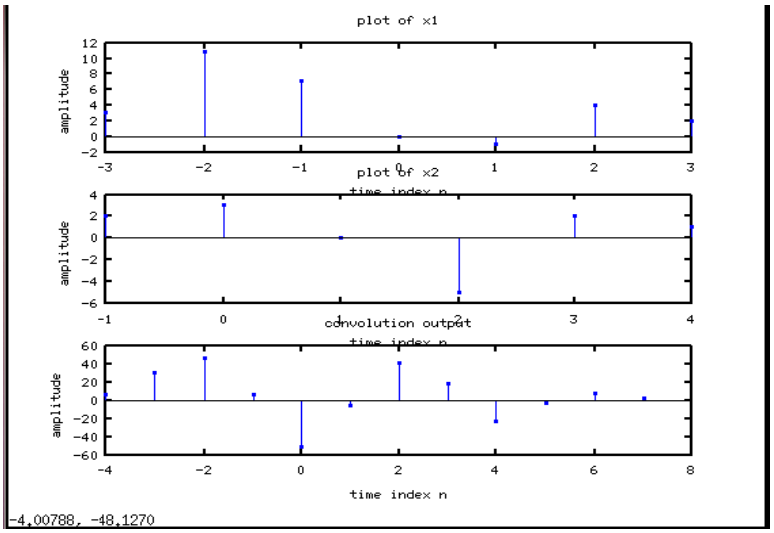
+3δ(n-1)+11δ(n-2)+7δ(n-3) δ(n-5)

+4δ(n-6)+2δ(n-7)

=+6δ(n+4)+31δ(n+3)+47δ(n+2)+6δ(n+1)-51δ(n)-5δ(n-1)+41δ(n-2)+18δ(n-3)-22δ(n-4)-3δ(n-5)+8δ(n-6)+2δ(n-7)

Therefore, y = [6, 31, 47, 6, -51, -5, 41, 18, -22, -3, 8, 2]

Figure:



Result and Inference:

x1 = [3 11 7 0 -1 4 2] n1 = [-3 -2 -1 0 1 2 3]

x2 = [2 3 0 -5 2 1] n2 = [-1 0 1 2 3 4]

linear con of x1 & x2 is y = [6, 31, 47, 6, -51, -5, 41, 18, -22, -3, 8, 2]

CHALLENGE EXPERIMENT:

1. Write a general matlab program for linear convolution for any two general sequences without using MATLAB inbuilt command ‘conv’

EXPERIMENT-02

AIM: - COMPUTATION OF N POINT DFT OF A GIVEN SEQUENCE USING THE DEFINITION OF DFT AND PLOT MAGNITUDE AND PHASE SPECTRUM, AND VERIFY USING BUILT IN FUNCTION (USING FFT).

Theory:

DFT & IDFT:

The **discrete Fourier transform** (**DFT**) converts a finite sequence of equally-spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. An inverse DFT is a Fourier series, using the DTFT samples as coefficients of complex sinusoids at the corresponding DTFT frequencies. It has the same sample-values as the original input sequence. The DFT is therefore said to be a frequency domain representation of the original input sequence.

Algorithm:

1. Input the sequence for which DFT is to be computed.
2. Input the length of the DFT required (say 4, 8, >length of the sequence).
3. Compute the DFT using the function dft.
4. Plot the magnitude & phase spectra.

MATLAB Implementation:

Create a matlab function dft. The procedure is as follows:

The commands and functions that comprise the new function must be put in a file whose name defines the name of the new function, with a filename extension of '.m'. At the top of the file must be a line that contains the syntax definition for the new function. For example, the existence of a file on disk called dft.m with body as follows:

```
function[Xk] =dft(xn,N)
n=[0:1:N-1];
k=[0:1:N-1];
WN=exp(-j*2*pi/N);
nk=n'*k;
WNnk=WN.^nk;
Xk=xn*WNnk;
```

defines a new function called ***dft*** that calculates the DFT of a sequence ***xn***. The variables within the body of the function are all local variables In the Main body of the program call this function to calculate the DFT. The magnitude spectrum is computed using the function ***abs*** (Absolute value). ***abs(x)*** is the absolute value of the elements of ***x***. When ***x*** is complex, ***abs(x)*** is the complex modulus (magnitude) of the elements of ***x***. The phase spectrum is computed using the function ***angle*** (Phase angle). ***angle (h)*** returns the phase angles, in radians, of a matrix with complex elements.

MATLAB Program:

```
clc;
clear all;
x=input('enter the sequence x(n)');
tic
y=dft(x);
toc
disp('DFT Computation using Direct Method');
disp(y);
tic
y1=fft(x);
toc
disp('DFT Computation using Fast Fourrier Transform Method');
disp(y1);
amp=abs(y);
ph=angle(y);
subplot(2,1,1)
stem(amp);
xlabel('K');
ylabel('MAGNITUDE');
title('magnitude plot');
subplot(2,1,2)
stem(ph*180/pi);
xlabel('K');
ylabel('PHASE');
title('phase plot');
```

```
function[Xk] =dft(xn)          % Save this sub program as dft.m
N=length(xn);
n=[0:N-1];
k=[0:N-1];
WN=exp(-j*2*pi/N);
nk=n'*k;
WNnk=WN.^nk;
Xk=xn*WNnk; end
```

Note:

N

x(n)<---------> x(k)

DFT

$$x(k) = \sum_{n=0}^{N-1} x(n) w_N^{nk} \qquad k = 0,1,\text{-----}N-1$$

where, $w_N = e^{-j 2\pi / N}$

$$x(k) = \sum_{n=0}^{N-1} x(n) e^{-j 2\pi k n / N} \qquad k = 0,1,\text{------}N-1$$

Calculations:

x(n) = [2 3 4 5] N = 4

$$x(k) = \sum_{n=0}^{3} x(n) e^{-j 2\pi k n / 4} \qquad k = 0,1,2,3$$

$$x(k) = x(0)e^{-j2\pi k * 0/4} + x(1)\, e^{-j 2\pi\, k *1/4} + x(2)\, e^{-j 2\pi k *2/4} + x(3)\, e^{-j 2\pi k*3/4}$$

$x(k) = 2 + 3e^{-j\, \pi k/2} + 4e^{-j\pi k} + 5e^{-j\, 3\pi k/2}$

x(0) = 2 + 3 + 4 +5 = **14**

$x(1) = 2 + 3e^{-j\pi/2} + 4e^{-j\pi} + 5e^{-j3\pi/2}$ **$\{e^{j\theta} = \cos\theta + j\sin\theta,\ e^{-j\theta} = \cos\theta - j\sin\theta\}$**

$= 2 + 3[\cos \pi/2 - j \sin \pi/2] + 4[\cos \pi - j \sin \pi] + 5[(0)\, 3\pi/2 - j\sin 3\pi/2]$

x(1) = -2 +2j

x(2) = $2 + 3e^{-j\pi/2} + 4e^{-j2\pi} + 5e^{-j3*2\pi/2}$

$= 2 + 3[e^{-j\pi} + 4e^{-j2\pi} + 5e^{-j3\pi}]$

$= 2 + 3[\cos \pi/2 - j \sin \pi/2] + 4[\cos \pi - j \sin \pi] + 5[(0)\, 3\pi/2 - j\sin 3\pi/2]$

$= 2 + 3[-1-0] + 4[1-0] + 5[-1-0]$

$= 2 - 3 + 4 - 5$

x(2) = -2

x(3) = $2 + 3e^{-j\pi/2} + 4e^{-j\,3\pi} + 5e^{-j3*3\pi/2}$

$= 2 + 3j - 4 - 5j$

x(3) = -2-2j

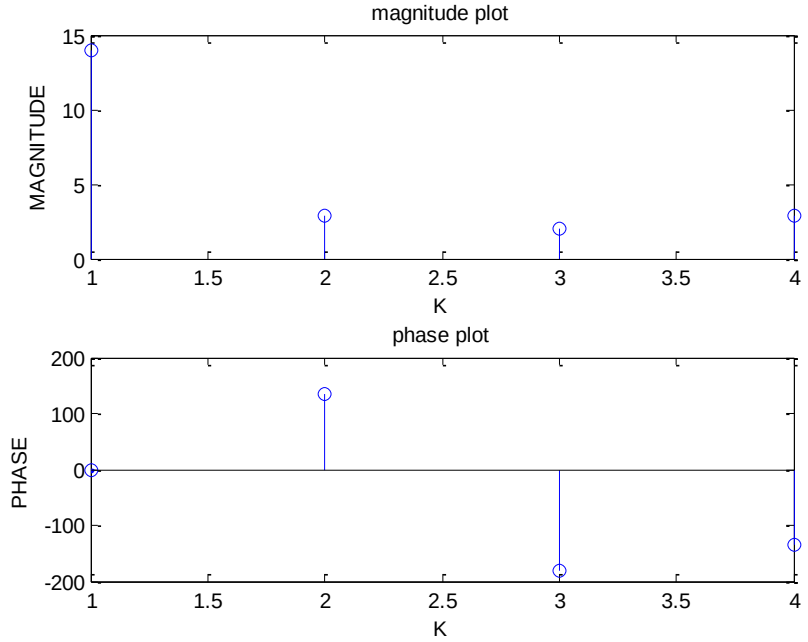
x(k) = [14, -2+2j, -2, -2-2j] *% rectangular form*

x(k) = [14, $2.82\angle 135^0$, $2\angle -180^0$, $2.82\angle -135^0$] *% polar form*

| x(k)| = [14, 2.82, 2, 2.82]

<x(k) = [0, 2.35, 3.143, -2.35] *% in radians*

Figure:



Inputs

Run1: Give 4 point sequence and find 4 point DFT

Run2: Give 8 point sequence and find 8 point DFT

Run3: Give a Sinusoidal Signal with a frequency 250 Hz with fs = 8 KHz and sample it for 64 points and perform 64 point DFT on it.

Run4: Give a Sinusoidal Signal with a frequency 250 Hz and 500 Hz with fs = 8 KHz and sample it for 64 points and perform 64 point DFT on it.

Result and Inference:

Elapsed time is 0.018641 seconds.

DFT Computation using Direct Method

14.0000 -2.0000 + 2.0000i -2.0000 - 0.0000i -2.0000 - 2.0000i

Elapsed time is 0.000026 seconds.

DFT Computation using Fast Fourier Transform Method

14.0000 -2.0000 + 2.0000i -2.0000 -2.0000 - 2.0000i

From this, it's clear that FFT Algorithm is much faster than regular DFT Computation.

EXPERIMENT-03

AIM: - TO COMPUTE THE CIRCULAR CONVOLUTION OF THE GIVEN TWO SEQUENCES IN TIME-DOMAIN

Theory:

Circular Convolution:

The **circular convolution**, also known as **cyclic convolution**, of two aperiodic functions occurs when one of them is convolved in the normal way with a periodic summation of the other function.

- As seen in the last experiment, the output y[n] of a LTI (linear time invariant) system can be obtained by convolving the input x[n] with the system's impulse response h[n].
- The above linear convolution is generally applied to aperiodic sequences. Whereas the Circular Convolution is used to study the interaction of two signals that are periodic.
- The linear convolution sum is

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

- To compute this equation, the linear convolution involves the following operations:
  - **Folding**- fold h[k] to get h[-k] (for y[0])
  - **Multiplication** – vk[n] = x[k] × h[n-k] (both sequences are multiplied sample by sample)
  - **Addition**- Sum all the samples of the sequence vk[n] to obtain y[n]
  - **Shifting** – the sequence h[-k] to get h[n-k] for the next n.
- The circular convolution sum is

$$\boldsymbol{y[n] = x[n](N)h[n] = \sum_{k=-\infty}^{+\infty} x[k]h[(n-k)_N] \quad where\ the\ index\ (n-k)_N}$$

  - Implies circular shifting operation and $\langle -k \rangle N$ implies folding the sequence circularly.
- Steps for circular convolution are the same as the usual convolution, except all index calculations are done "mod N" = "on the wheel".
  - Plot f [m] and h [−m] as shown in Fig. (use f(m) instead of x(k))
  - Multiply the two sequences
  - Add to get y[m]
  - "Spin" h[−m] n times Anti Clock Wise (counter-clockwise) to get h[nm].
- x[n] and h[n] can be both finite or infinite duration sequences. If infinite sequences, they should be periodic, and the N is chosen to be at least equal to the period. If they are finite sequences N is chosen as >= to max(xlength, hlength). Whereas in linear convolution N>= xlength+hlength-1.

- Say $x[n]=\{-2,3,\underset{\uparrow}{1},1\}$ and N = 5, then the x[-1] and x[-2] samples are plotted at

  - x[N-1] = x[-4] and x[N-2] =x[-3] places. Now x[n] is entered as

$$x[n]=\{\underset{\uparrow}{1},1,0,-2,3\}$$

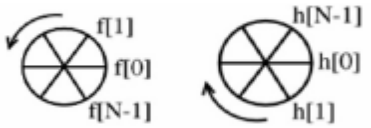


Fig. - Plotting of f(m) and h(-m) for circular convolution

Algorithm:

1. Input the two sequences as x and h.
2. Circularly convolve both to get output y.
3. Plot the sequences.

MATLAB Implementation:

Matlab recognizes index 1 to be positive maximum. Index 0 is not recognized. Hence in the below program wherever y, x and h sequences are accessed, the index is added with +1. the modulo index calculation for circular convolution is carried out using the function - MOD Modulus (signed remainder after division). MOD(x,y) is x - y.*floor(x./y) if y ~= 0. By convention, MOD(x,0) is x. The input x

and y must be real arrays of the same size, or real scalars. MOD(x,y) has the same sign as y while REM(x,y) has the same sign as x. MOD(x,y) and REM(x,y) are equal if x and y have the same sign, but differ by y if x and y have different signs.

MATLAB Program:

```
clc;
clear all;
x=input('Enter first sequence ');
h=input('Enter second sequence ');
N1=length(x);
```

```
N2=length(h);
N=max(N1,N2);
x=[x,zeros(1,N-N1)];
h=[h,zeros(1,N-N2)];
for r=1:N                          % To calculate the value of 'n'
y(r)=0;
for s=1:N
k=r-s+1;                           % Calculation of x index
if k<=0
k=k+N;
end                                % End of 'if' statement
y(r)=y(r)+x(s)*h(k);
end                                % End of inner 'for loop'
end                                % End of outer 'for loop'
disp('circular convolution of x & h is y=');
disp(y);
subplot(3,1,1);
stem(x);
xlabel('time index n');
ylabel('amplitude ');
title('plot of x');
subplot(3,1,2);
stem(h);
xlabel('time index n');
ylabel('amplitude ');
title('plot of h');
subplot (3,1,3);
n1=0:N-1;
stem(n1,y);
xlabel('time index n');
ylabel('amplitude ');
title('Circular convolution output y(n)');
```

Calculations:-

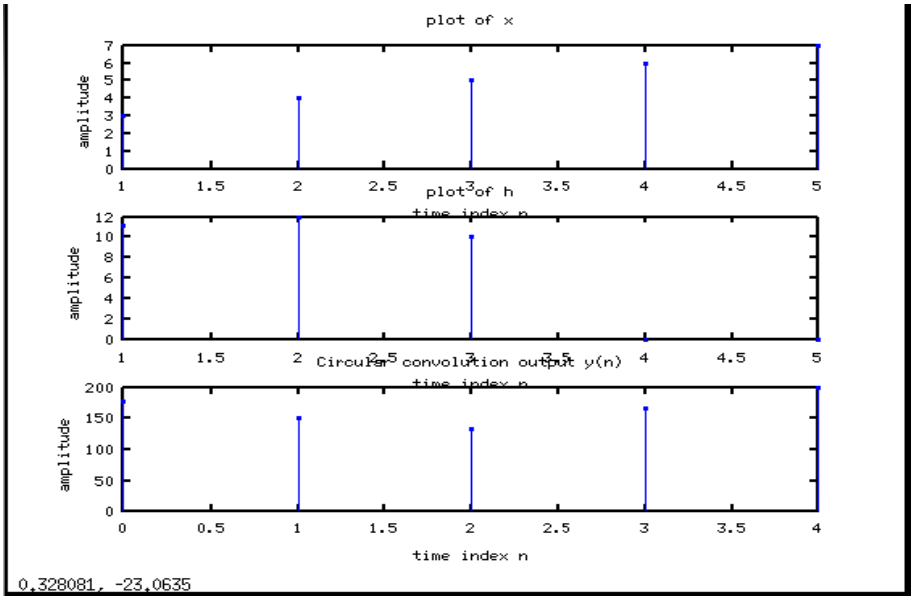
x(n) = [3 4 5 6 7] ------ length=5

h(n) = [11 12 10] ------ length=3

h(n) = [11 12 10 0 0]

n	x(m)	$h((n-m))_N$	y(n)
0	3,4,5,6,7	11,0,0,10,12	33+0+0+60+84=177
1	3,4,5,6,7	12,11,0,0,10	36+44+0+70=150
2	3,4,5,6,7	10,12,11,0,0	30+48+55+0+0=133
3	3,4,5,6,7	0,10,12,11,0	0+40+60+66+0=166
4	3,4,5,6,7	0,0,10,12,11	0+0+50+72+77=199

Figure:



Results and Inference:

Enter first sequence = [3 4 5 6 7]

Enter second sequence = [11 12 10]

Circular convolution output y(n) = [177,150,133,166,199]

For Two Periodic sequences, circular convolution can be performed using various methods like Graphical, Matrix or Concentric Circle Methods. We have verified the same using MATLAB Code.

EXPERIMENT-04

AIM:-TO PERFORM CIRCULAR CONVOLUTION OF TWO SEQUENCES USING DFT & IDFT (IN FREQUENCY DOMAIN)

Theory:

- By multiplying two N-point DFTs in the frequency domain, we get the circular convolution in the time domain.

$$y[n] = x[n] * h[n] \xrightarrow[N\text{-}DFT]{} Y(k) = X(k) * H(k)$$

- Circular Convolution is made identical to linear convolution by choosing the number of points in DFT as N ≥ xlength + hlength -1.
- For Circular Convolution, N = max (xlength, length).

Algorithm:

1. Input the two sequences as x1, x2
2. Compute **N=max(xlength + hlength))**
3. Obtain N-point DFTs (X1, X2) of both the sequences.
4. Multiply both the DFTs (Y=X1×X2).
5. Perform IDFT on Y to get y[n] (the linearly convolved sequence) in time domain.
6. Verify using the 'conv' command.
7. Plot the sequences.

MATLAB Implementation:

Matlab has the function fft(x,N) to perform the N point DFT. The function IFFT(X) performs the inverse discrete Fourier transform of X.

IFFT(X,N) is the N-point inverse transform.

MAX - Largest component. For vectors, MAX(X) is the largest element in X.

LENGTH Length of vector. LENGTH(X) returns the length of vector X.

In matlab/gnu-octave, the operator * implies Matrix multiplication. X*Y is the matrix product of X and Y. The number of columns of X must be equal to the number of rows of Y.

The operator .* Array multiplication. X.*Y denotes element-by-element multiplication. X and Y must have the same dimensions (unless one is a scalar). In the below program to multiply the two DFTs (vectors), we use .* (dot star) operator.

MATLAB Program using DFT:

```
clc;
x=[2 3 6 7 8];
h=[1 4 7];
x1=length(x);
h1=length(h);
n=max(x1,h1);
x=[x,zeros(1,n-x1)];
h=[h,zeros(1,n-h1)];
xd=fft(x,n);
hd=fft(h,n);
yd=xd.*hd;
y=ifft(yd,n);
disp(y);
stem(y);
```

Calculations:-

let us consider two sequences,

x(n)=[2 3 6 7 8]

h(n)=[1 4 7]

length of the convolution =5

4

$x(k) = \sum_{n=0} x(n)e^{-j 2\pi k n / 5}$

$x(k) = 2+3\,e^{-j 2\pi k / 5} + 6\,e^{-j2\pi 6k / 5} + 7\,e^{-j 2\pi 7k / 5} + 8\,e^{-j 2\pi 8k / 5}$

for n=0, $x(0) = 2 +3 +6+ 7+8 = \mathbf{26}$

for n=1, $\mathbf{x(1) = 2+3\,e^{-j 2\pi / 5} + 6\,e^{-j2\pi 6(1) / 5} + 7\,e^{-j 2\pi 7(1) / 5} + 8\,e^{-j2\pi 8(1) / 5}}$ **= -5.1180+5.34j**

for n=2, $\mathbf{x(2) = 2+3\,e^{-j 2\pi 2 / 5} + 6\,e^{-j2\pi 6(2) / 5} + 7\,e^{-j 2\pi 7(2) / 5} + 8\,e^{-j2\pi 8(2) / 5}}$ **= -2.882+1.98j**

for n=3, $\mathbf{x(3) = 2+3\,e^{-j 2\pi 3 / 5} + 6\,e^{-j2\pi 6(3) / 5} + 7\,e^{-j 2\pi 7(3) / 5} + 8\,e^{-j2\pi 8(3) / 5}}$ **= -2.882-1.98j**

for n=4, $\mathbf{x(4) = 2+3\,e^{-j 2\pi 4 / 5} + 6\,e^{-j2\pi 6(4) / 5} + 7\,e^{-j 2\pi 7(4) / 5} + 8\,e^{-j2\pi 8(4) / 5}}$ **= -5.1180-5.34j**

$$h(k) = \sum_{n=0}^{4} h(n) e^{-j 2\pi k n / 5}$$

$h(k) = h(0)+h(1)\, e^{-j 2\pi k / 5} + h(2)\, e^{-j 4\pi k / 5} + h(3)\, e^{-j 6\pi k / 5} + h(4)\, e^{-j 8\pi k / 5}$

for k=0, $h(0) = 1 + 4 + 7 =$ **12**

for k=1, $h(1) = h(0)+h(1)\, e^{-j 2\pi / 5} + h(2)\, e^{-j 4\pi / 5} + h(3)\, e^{-j 6\pi / 5} + h(4)\, e^{-j 8\pi / 5}$
= **- 3.4271 -7.918j**

for k=2, $h(2) = h(0)+h(1)\, e^{-j 4\pi / 5} + h(2)\, e^{-j 8\pi / 5} + h(3)\, e^{-j 12\pi / 5} + h(4)\, e^{-j 16\pi / 5}$
= **-0.0729+4.3063j**

for k=3, $h(3) = h(0)+h(1)\, e^{-j 6\pi / 5} + h(2)\, e^{-j 12\pi / 5} + h(3)\, e^{-j 18\pi / 5} + h(4)\, e^{-j 24\pi / 5}$
= -0.0729-4.3063j

for k=4, $h(4) = h(0)+h(1)\, e^{-j 8\pi / 5} + h(2)\, e^{-j 16\pi / 5} + h(3)\, e^{-j 24\pi / 5} + h(4)\, e^{-j 32\pi / 5}$
= **-3.4271+7.918j**

y(k) = x(k)*h(k)

= [3.12, 0.5985+0.22j, -0.0835-0.1256j, -0.0835+0.1256j, 0.5985-0.22j] 10^2

$$\mathbf{y(n) = \frac{1}{N} \sum_{k=0}^{N-1} y(k)\, e^{j 2\pi k n / N}}$$

$$\mathbf{y(n) = \frac{1}{5} [\, Y(0)\, e^{0} + y(1) e^{j 2\pi n / 5} + y(2)\, e^{j 4\pi n / 5} + y(3)\, e^{j 6\pi n / 5} + y(4)\, e^{j 8\pi n / 5} \,]}$$

for n=0, $y(0) = \frac{1}{5} [\, 3.12 + 0.585 - 0.083 + 0.5985 - 0.083 \,] =$ **83**

for n=1, $y(1) = \frac{1}{5} [\, y(0) + y(1) e^{j 2\pi / 5} + y(2)\, e^{j 4\pi / 5} + y(3)\, e^{j 6\pi / 5} + y(4)\, e^{j 8\pi / 5} \,]$

y(1) = 67

for n=2, $y(2) = \frac{1}{5} [\, y(0) + y(1) e^{j 4\pi / 5} + y(2)\, e^{j 8\pi / 5} + y(3)\, e^{j 12\pi / 5} + y(4)\, e^{j 16\pi / 5} \,]$

y(2) = 32

for n=3, $y(3) = \frac{1}{5}[y(0) + y(1)e^{j6\pi/5} + y(2)e^{j12\pi/5} + y(3)e^{j18\pi/5} + y(4)e^{j24\pi/5}]$

y(3) = 52

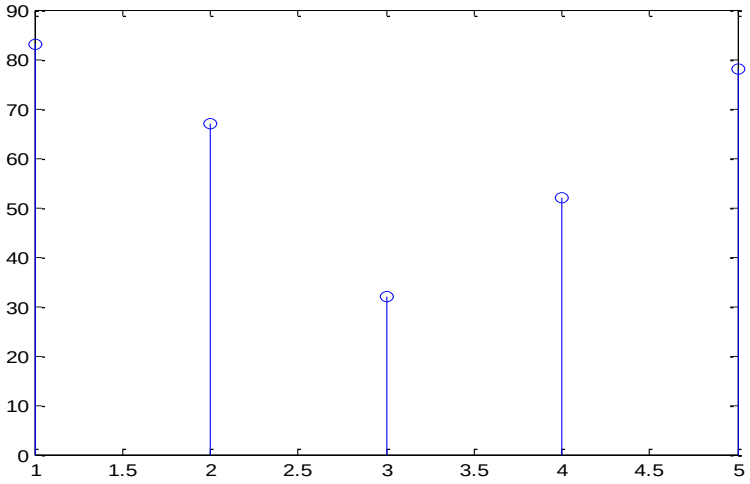
for n=4, $y(4) = \frac{1}{5}[y(0) + y(1)e^{j8\pi/5} + y(2)e^{j16\pi/5} + y(3)e^{j24\pi/5} + y(4)e^{j32\pi/5}]$

y(4) = 78

Therefore,

y(n) = [83, 67, 32, 52, 78]

Figure:



Results and Inference:

x= [2 3 6 7 8];

h= [1 4 7];

Circular convolution of x &h is y(n)= [83, 67, 32, 52, 78]

Circular Convolution in time domain is equivalent to Multiplication in frequency domain. We have verified the same using DFT, IDFT Calculations and MATLAB Code.

Challenge experiment:

1. Write a matlab program for linear convolution using circular convolution.

EXPERIMENT-05

AIM: - PROGRAM TO OBTAIN THE AUTO CORRELATION OF A GIVEN SEQUENCE AND VERIFY ITS PROPERTIES.

Theory:

Auto correlation:

Autocorrelation, also known as **serial correlation**, is the correlation of a signal with a delayed copy of itself as a function of delay. it is the similarity between observations as a function of the time lag between them. The analysis of autocorrelation is a mathematical tool for finding repeating patterns, such as the presence of a periodic signal obscured by noise, or identifying the missing fundamental frequency in a signal implied by its harmonic frequencies. It is often used in signal processing for analyzing functions or series of values, such as time domain signals.

- Correlation is mathematical technique which indicates whether 2 signals are related and in a precise quantitative way how much they are related. A measure of similarity between a pair of energy signals x[n] and y[n] is given by the cross correlation sequence $r_{xy}[l]$ defined by :

$$r_{xy}[l] = \sum_{n=-\infty}^{\infty} x[n]y[n-l]; l = 0, \pm 1, \pm 2, \dots$$

- The parameter '*l*' called 'lag' indicates the time shift between the pair.
- Autocorrelation sequence of x[n] is given by :-

$$r_{xx}[l] = \sum_{n=-\infty}^{\infty} x[n]x[n-l]; l = 0, \pm 1, \pm 2, \dots$$

- Some of the properties of autocorrelation are enumerated below –
  - The autocorrelation sequence is an even function i.e., $r_{xx}[l]=r_{xx}[-l]$
  - At zero lag, i.e., at l=0, the sample value of the autocorrelation sequence has its maximum value (equal to the total energy of the signal ex) i.e.,

$$r_{xx}[l] \leq r_{xx}[0] = \epsilon_x = \sum_{n=-\infty}^{\infty} x^2[n]$$

    This is verified in Fig. where the autocorrelation of the rectangular pulse (square) has a maximum value at l=0. All other samples are of lower value. Also the maximum value = 11 = energy of the pulse [$1^2+1^2+1^2$..].
  - A time shift of a signal does not change its autocorrelation sequence. For example, let y[n]=x[n-k]; then $r_{yy}[l] = r_{xx}[l]$ i.e., the autocorrelation of x[n] and y[n] are the same regardless of the value of the time shift k. This can be verified with a sine and cosine sequences of same amplitude and frequency will have identical autocorrelation functions.
  - For power signals the autocorrelation sequence is given by :

$$r_{xx}[l] = \lim_{k\to\infty} \frac{1}{2k+1} \sum_{n=-k}^{k} x[n]x[n-l]; l = 0, \pm 1, \pm 2, \dots$$

and for periodic signals with period N it is :

$$r_{xx}[l] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n-l]; l = 0, \pm 1, \pm 2, \ldots$$

and this **r_{xx}[l] is also periodic** with N. This is verified in Fig. 5.3 where we use the periodicity property of the autocorrelation sequence to determine the period of the periodic signal y[n] which is x[n] (=cos(0.25*pi*n)) corrupted by an additive uniformly distributed random noise of amplitude in the range [-0.5 0.5]

Algorithm:

1. Input the sequence as x.
2. Flip the x sequence from left to right and convolve with x sequence or
   Use the 'xcorr' function to get auto correlated output r.
3. Plot the sequences.

MATLAB Implementation:

Matlab has the inbuilt function XCORR(A), when A is a vector, is the auto-correlation sequence. If A is of length M vector (M>1), then the xcorr function returns the length 2*M-1 auto-correlation sequence. The zeroth lag of the output correlation is in the middle of the sequence at element M.

XCORR(...,MAXLAG) computes the (auto/cross) correlation over the range of lags:

-MAXLAG to MAXLAG, i.e., 2*MAXLAG+1 lags. If missing, default is MAXLAG = M-1.

[C,LAGS] = XCORR(...) returns a vector of lag indices (LAGS).

MATLAB Program for linear autocorrelation: (time domain)

```
clc;
x=[1 2 3 4];
rxxl=conv(x,fliplr(x));
N=length(x);
n=-(N-1):(N-1);
disp(rxxl);
stem(n,rxxl);
xlabel('lag variables');
ylabel('amplitude of rxxl');
title('auto correlation');
m=ceil(length(rxxl)/2);
mx= max(rxxl);
if(rxxl(m)==mx)
disp('autocorrelation peak occurs at lag=0');
end
if(rxxl==fliplr(rxxl))
disp('even property of auto correlation is verified');
end
```

Calculations:

$$\tilde{r}_{xx}(l) = \sum_{-\infty}^{\infty} x(k)x(k-l)$$

x = [1 2 3 4]

$$\tilde{r}_{xx}(l) = \sum_{-3}^{3} x(k)x(k-l)$$

$$\tilde{r}_{xx}(l) = \mathbf{x(-3)x(-3-l) + x(-2)x(-2-l) + x(-1)x(-1-l)} + x(0)x(0-l) + x(1)x(1-l) + x(2)\,x(2-l) + x(3)x(3-l)$$

$$\tilde{r}_{xx}(l) = x(0)x(0-l) + x(1)x(1-l) + x(2)\,x(2-l) + x(3)x(3-l)$$

$$\tilde{r}_{xx}(l) = 1x(0-l) + 2x(1-l) + 3x(2-l) + 4x(3-l)$$

$l=-3;\quad \tilde{r}_{xx}(-3) = x(3) + 2x(4) = \mathbf{4}$

$l=-2;\quad \tilde{r}_{xx}(-2) = x(2)+2x(3) = \mathbf{11}$

$l=-1;\quad \tilde{r}_{xx}(-1) = x(1)+2x(2)+3x(3) = \mathbf{20}$

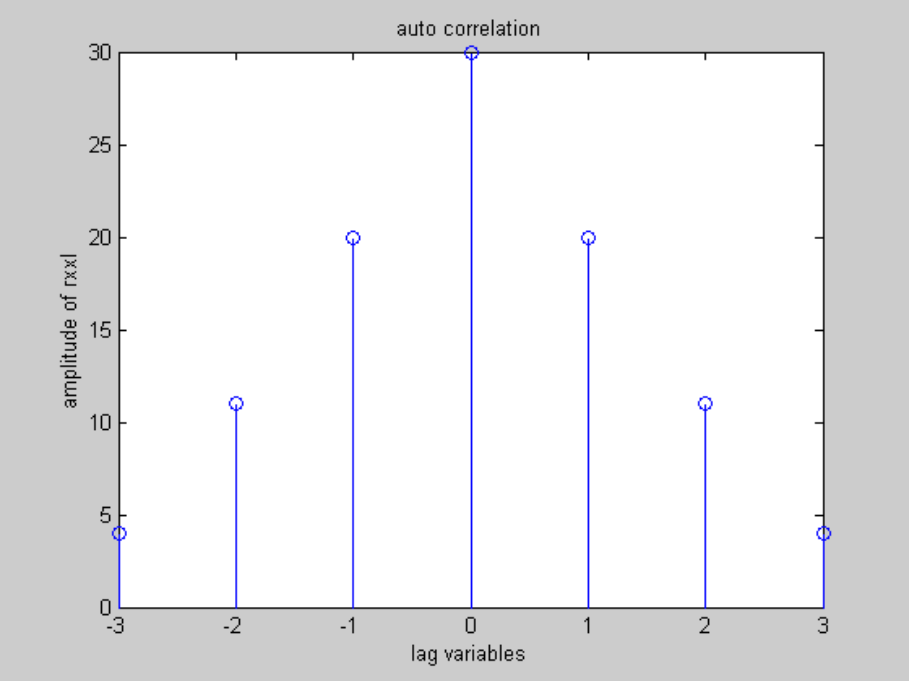
$l=0;\quad \tilde{r}_{xx}(0) = x(0)+2x(1)+3x(2)+4x(3) = \mathbf{30}$

$l=1;\quad \tilde{r}_{xx}(1) = 2x(0)+3x(1)+4x(2) = \mathbf{20}$

$l=2;\quad \tilde{r}_{xx}(2) = x(2)x(0)+x(3)x(1) = \mathbf{11}$

$l=3;\quad \tilde{r}_{xx}(3) = x(3)x(0) = \mathbf{4}$

Figure:



Results and Inference:

~

rxx(l) = [4 11 20 30 20 11 4]

Inference-1: Following autocorrelation properties are verified:-

1. Max peak of 30 (=energy of the pulse) at zero lag (l=0)
2. The autocorrelation sequence is an even function

MATLAB Program for circular autocorrelation: (frequency domain)

```
clc;
x=[1 2 3 4];
xd=fft(x);
rxxl=ifft(xd.*conj(xd));
disp(rxxl);
stem(rxxl);
xlabel('lag variables');
ylabel('amplitude of rxxl');
title('auto correlation in freq domain');
```

Calculations:

x(n) = [1 2 3 4]

DFT [x(n)] = x(k)

x(k) = [1 2 3 4]

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

x(k) = [10 -2+2j -2 -2-2j]

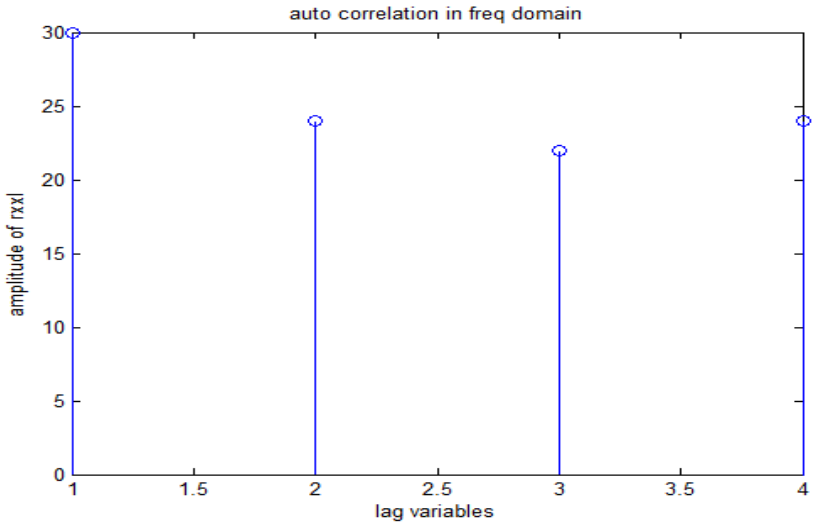
x*(k) = [10 -2-2j -2 -2+2j]

rxxl = x(k)x*(k) = **[100 8 4 8]**

IDFT[rxxl) = ¼ [100 8 4 8]

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}$$

Figure:



Results and Inference:

IDFT[rxxl) = [30 24 22 24]

AIM: - PROGRAM TO OBTAIN CROSS CORRELATION OF A GIVEN SEQUENCE AND VERIFY ITS PROPERTIES.

Theory:

Cross correlation:

In signal processing, **cross-correlation** is a measure of similarity of two series as a function of the displacement of one relative to the other. It is commonly used for searching a long signal for a shorter, known feature. It has applications in pattern recognition, single particle analysis, electron tomography, averaging, cryptanalysis, and neurophysiology.

- Cross Correlation has been introduced in the last experiment. Comparing the equations for the linear convolution and cross correlation we find that

$$r_{xy}[l] = \sum_{n=-\infty}^{\infty} x[n]y[n-l] = \sum_{n=-\infty}^{\infty} x[n]y[-(l-n)] = x[l] * y[-l]. \qquad \text{i.e.,}$$

  Convolving the reference signal with a folded version of sequence to be shifted (y[n]) results in cross correlation output. (Use 'fliplr' function for folding the sequence for correlation).

- The properties of cross correlation are

  1. The cross correlation sequence sample values are upper bounded by the inequality $r_{xx}[l] \leq \sqrt{r_{xx}[0]r_{yy}[0]} = \sqrt{\varepsilon_x \varepsilon_y}$
  2. The cross correlation of two sequences x[n] and y[n]= x[n-k] shows a peak at the value of k. Hence cross correlation is employed to compute the exact value of the delay k between the 2 signals. Used in radar and sonar applications, where the received signal reflected from the target is the delayed version of the transmitted signal (measure delay to determine the distance of the target).
  3. The ordering of the subscripts xy specifies that x[n] is the reference sequence That remains fixed in time, whereas the sequence y[n] is shifted w.r.t x[n]. If y[n] is the reference sequence then $r_{yx}[l] = r_{xy}[-l]$. Hence $r_{yx}[l]$ is obtained by time reversing the sequence $r_{xy}[l]$.

%Verification of the properties

%Small matlab/gnu-octave code

```
xi=[1,2,3,-1,0,0];
x2=[0,0,1,2,3,-1];
```

%=xr[n-2],i.e., k=2

```
>> xcorr(xi,x2)
ans =  {-1,1,5,15,5,1,-1,0,0,0,0}
                  ↑
```

Inference:

Strong peak of 15 at lag = -2 implies the delay between xi and x2 is 2. Also peak =15=energy of xi ($1+2^2+3^2+(-1)^2$) implies that both xi and x2 are strongly correlated.

% consider the below sequences

xr=[1,2,3,-1];

x2=[3,-1,1,2];

xcorr(xr,x2) ans = $\{2,5,7,\underset{\uparrow}{2},2,10,-3\}$

xcorr(x2,xr) ans = $\{-3,10,2,\underset{\uparrow}{2},7,5,2\}$

Inference:

Strong peak of 10 at lag = 2 implies the delay between xr and x2 is 2, but since10<15, it implies that xr and x2 are uncorrelated slightly (may be due to noise,etc).

$r_{yx}[l]=r_{xy}[-l]$ is verified.

Algorithm:

1. Input the sequence as x and y.
2. Use the 'xcorr' function to get cross correlated output r.
3. Plot the sequences.

MATLAB Implementation:

Matlab has the inbuilt function XCORR: Say C = XCORR(A,B), where A and B are length M vectors (M>1), returns the length 2*M-1 cross-correlation sequence C. If A and B are of different length, the shortest one is zero-padded. Using convolution to implement correlation, the instruction is **FLIPLR Flip matrix in left/right direction.** FLIPLR(X) returns X with row preserved and columns flipped in the left/right direction. X = 1 2 3 becomes 3 2 1.

MATLAB Program for linear cross correlation: (time domain)

```
clc;
x1=[1 2 3 4];
x2=[4 3 2 1];
N1=length(x1);
N2=length(x2);
rxyl=conv(x1,fliplr(x2));
n=-(N2-1):(N1-1);
disp(rxyl);
stem(n,rxyl);
```

xlabel('lag variables');
ylabel('amplitude of rxyl');
title('cross correlation');

Calculations:-

$$rxy(l) = \sum_{k=-\infty}^{\infty} x1(k)x2(k-l)$$

Where, l=lag variables
k=frequency

x1[n] = [1 2 3 4] x2[n] = [4 3 2 1]
n=[-(max(length(x1),length(x2))-1) : max(length(x1),length(x2))]
% n=[-3:3]

$$rxy(l) = \sum_{k=-3}^{3} x1(k)x2(k-l)$$

$$rxy(l) = x1(-3)x2(-3-l) + x1(-2)x2(-2-l) + x1(-1)x2(-1-l) + x1(0)x2(0-l) + x1(1)x2(1-l) + x1(2)\,x2(2-l) + x1(3)x2(3-l)$$

$$rxy(l) = x1(0)x2(0-l) + x1(1)x2(1-l) + x1(2)\,x2(2-l) + x1(3)x2(3-l)$$

$$rxy(l) = x2(-l) + 2x2(1-l) + 3x2(2-l) + 4x2(3-l)$$

l=-3;	rxy(-3)	=	x2(3)	= **1**
l=-2;	rxy(-2)	=	x2(2)+2x2(3)	= **4**
l=-1;	rxy(-1)	=	x2(1)+2x2(2)+3x2(3)	= **10**
l=0;	rxy(0)	=	x2(0)+2x2(1)+3x2(2)+4x2(3)	= **20**
l=1;	rxy(1)	=	x2(-1)+2x2(0)+3x2(1)+4x2(2)	= **25**

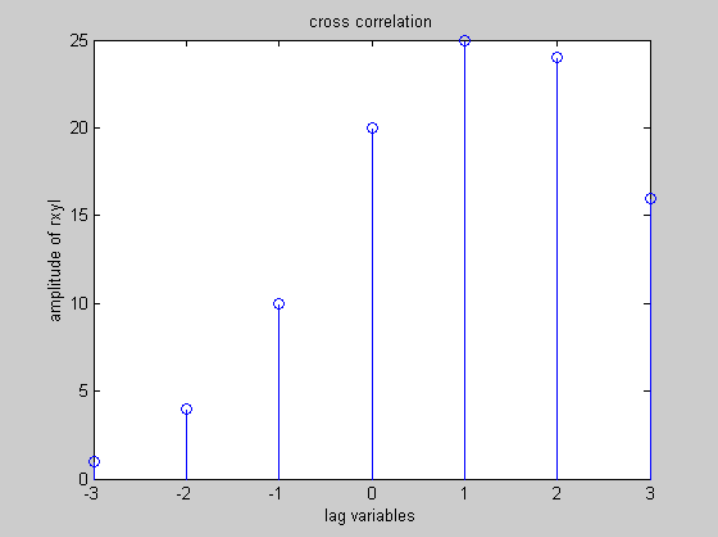
~

l=2; rxy(2) = 3x2(0)+4x2(1) = **24**

~

l=3; rxy(3) = 4x2(0) = **16**

Figure:



Results and Inference:

~

rxy(l) = [1, 4, 10, 20, 25, 24, 16]

Matlab Program for circular cross correlation: (frequency domain)

```
clc;
x=[1 2 3 4];
h=[4 3 2 1];
N=max(length(x),length(h));
x=[x,zeros(1,N-length(x))];
h=[h,zeros(1,N-length(h))];
xd=fft(x);
hd=fft(h);
yxy=ifft(xd.*conj(hd));
disp(yxy);
stem(yxy);
xlabel('l->');
ylabel('amplitude');
title('cross correlation in freq domain');
```

Calculations:

yxy(k) = x(k) h*(k)

x(n) = [1 2 3 4]

x(k) = [1 2 3 4]

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & j \end{bmatrix}$$

x(k) = [10 -2+2j -2 -2-2j]

h(n) = [4 3 2 1]

h(k) = [4 3 2 1]

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & j \end{bmatrix}$$

h(k) = [10 2-2j 2 2+2j]

yxy(k) = x(0).h(0) = 100

x(1).h(1) = -8

x(2).h(2) = -4

x(3).h(3) = -8

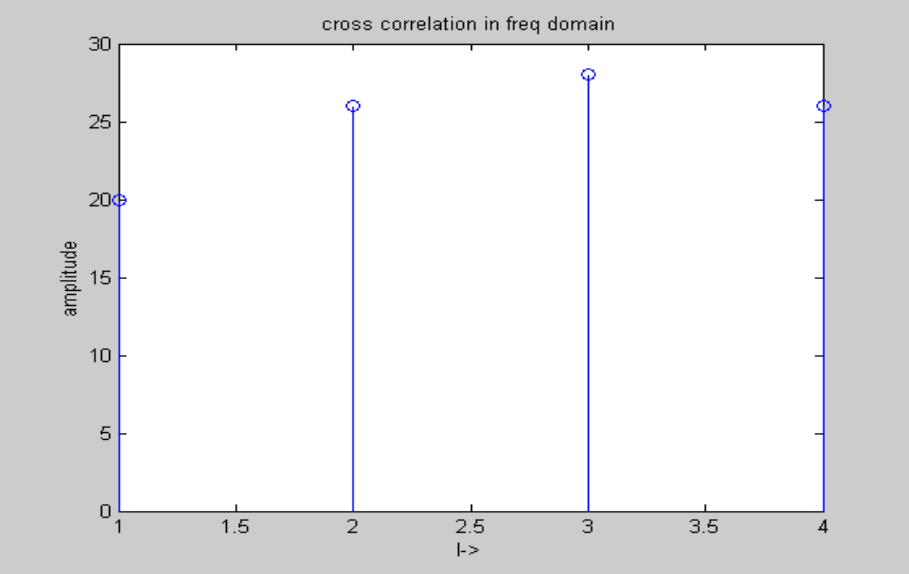
yxy(k) = [100 -8 -4 -8]

IDFT[yxy(k)] = ¼ [100 -8 -4 -8]

$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}$$

IDFT[yxy(k)] = [20 26 28 26]

Figure:



Results and Inference:

IDFT [yxy (k)] = [20, 26, 28, 26]

EXPERIMENT-06

AIM:- TO VERIFY SAMPLING THEOREM

Theory:

Sampling is a process of converting a continuous time signal (analog signal) x(t) into a discrete time signal x[n], which is represented as a sequence of numbers. (A/D converter) Converting back x[n] into analog (resulting in $x(t)$) is the process of reconstruction. (D/A converter)

Aliasing-A high frequency signal is converted to a lower frequency, results due to under sampling. Though it is undesirable in ADCs, it finds practical applications in stroboscope and sampling oscilloscopes.

MATLAB Program:

```
clc
clear all;
close all;
tfinal=0.05;
t=0:0.00005:tfinal;
f1=input('Enter first analog frequency');
f2=input('Enter second analog frequency');

%define input analog signal

xt=cos(2*pi*f1*t)+cos(2*pi*f2*t);
fm=max(f1,f2);

%simulate condition for undersampling i.e., fs < 2fm

fs=1.3*fm;
Ts=1/fs;
n=0:Ts:tfinal;

%Generate the under sampled signal

xn=cos(2*pi*f1*n)+cos(2*pi*f2*n);

%plot the analog & sampled signals

figure(1)
subplot(3,1,1);
plot(t,xt)
xlabel('t')
ylabel('x(t)')
title('Original analog signal');
subplot(3,1,2);
stem(n,xn)
xlabel('n')
ylabel('x(n)')
title('Under sampled signal');
```

```
subplot(3,1,3);
plot(t,xt,'b',n,xn,'r*-');
xlabel('t')
ylabel('x(t) & xr(t)')
title('Reconstructed analog signal after under sampling');
legend('original analog signal','reconstructed signal after sampling')
```

%condition for Nyquist plot

```
fs=2*fm;
Ts=1/fs;
n=0:Ts:tfinal;
```

%Generate the nyquist sampled signal

```
xn=cos(2*pi*f1*n)+cos(2*pi*f2*n);
```

%plot the analog & sampled signals

```
figure(2)
subplot(3,1,1);
plot(t,xt)
xlabel('t')
ylabel('x(t)')
title('Original analog signal');
subplot(3,1,2);
stem(n,xn)
xlabel('n')
ylabel('x(n)')
title('Nyquist sampled signal');
subplot(3,1,3);
plot(t,xt,'b',n,xn,'r*-');
xlabel('t')
ylabel('x(t) & xr(t)')
title('Reconstructed analog signal after nyquist sampling');
legend('original analog signal','reconstructed signal after sampling')
```

%condition for oversampling

```
fs=5*fm;
Ts=1/fs;
n=0:Ts:tfinal;
%Generate the oversampled signal
xn=cos(2*pi*f1*n)+cos(2*pi*f2*n);
```

%plot the analog & sampled signals

```
figure(3)
subplot(3,1,1);
plot(t,xt)
xlabel('t')
ylabel('x(t)')
title('Original analog signal');
subplot(3,1,2);
stem(n,xn)
xlabel('n')
ylabel('x(n)')
title('Over sampled signal');
subplot(3,1,3);
plot(t,xt,'b',n,xn,'r*-');
xlabel('t')
ylabel('x(t) & xr(t)')
title('Reconstructed analog signal after 0ver sampling');
legend('original analog signal','reconstructed signal after sampling');
```

Results and Inference

f1 = 100 Hz

f2 = 300 Hz

Figure 1:

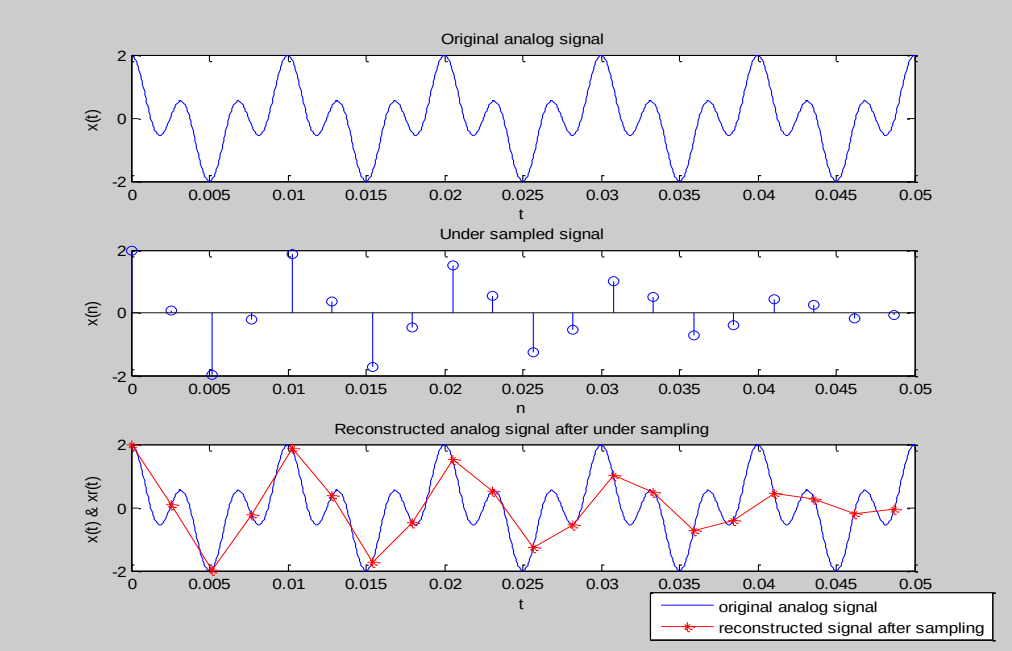


Figure 2:

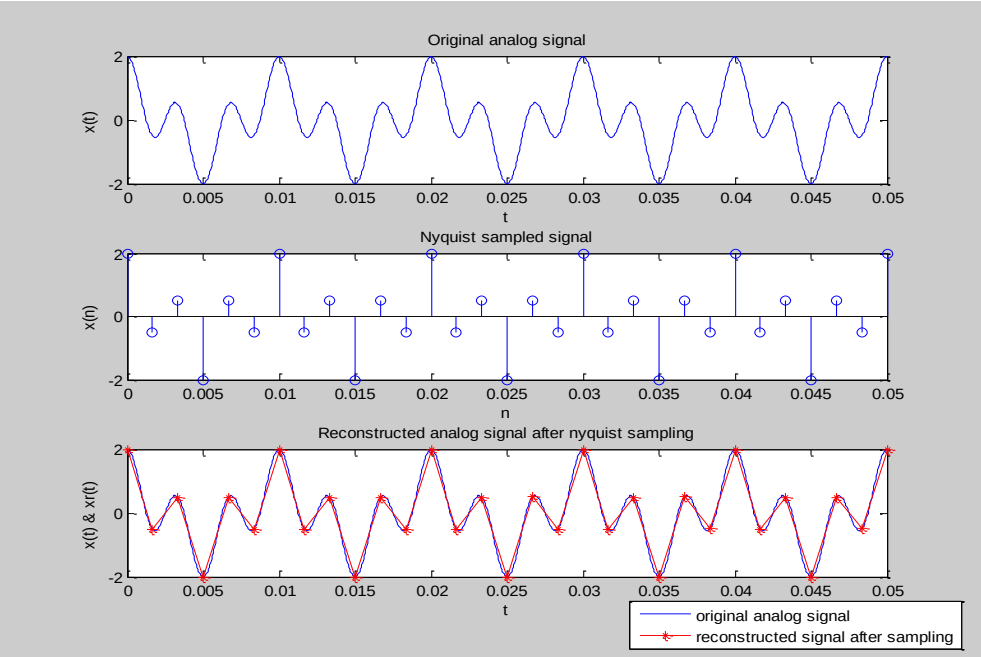
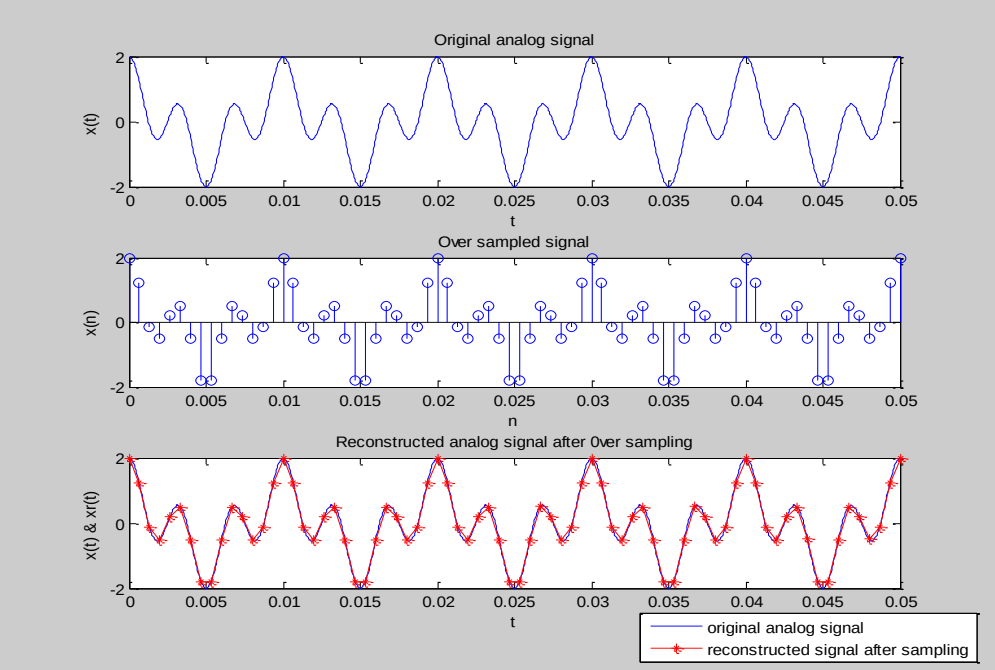


Figure 3:



All the three conditions of under sampling, Nyquist Rate Sampling, Over Sampling have been verified by reconstructing the sampled signal.

CHALLENGE EXPERIMENT:

1. Write a matlab program for sampling theorem with user inputs.

a. Number of sinusoids.

b. Reconstruct and compare with the original signal. Calculate errors under over sampling and under sampling cases.

EXPERIMENT-07

AIM: - TO DESIGN DIGITAL LPF/HPF BUTTERWORTH IIR FILTER TO MEET GIVEN SPECIFICATIONS

Theory:

There are two methods of stating the specifications as illustrated in previous program. In the first program, the given specifications are directly converted to digital form and the designed filter is also implemented. In the last two programs the butterworth and chebyshev filters are designed using bilinear transformation (for theory verification).

Method I: Given the order N, cutoff frequency fc, sampling frequency fs and the IIR filter type (butterworth,cheby1,cheby2).

Step 1: Compute the digital cut-off frequency Wc (in the range $-\pi < Wc < \pi$, with π corresponding to fs/2) for fc and fs in Hz.

For example let fc=400Hz,fs=8000Hz $Wc = 2*\pi* fc / fs = 2* \pi * 400/8000=0.1* \pi$ radians.

For matlab/gnu-octave the Normalized cut-off frequency is in the range 0 and 1,where 1 corresponds to fs/2 (i.e.,fmax)).

Hence to use the matlab/gnu-octave commands wc = fc / (fs/2) = 400/(8000/2) = 0.1

Note: *If the cut off frequency is in radians then normalized frequency is computed as wc = Wc / π*

Step 2: Compute the Impulse Response [b,a] coefficients of the required IIR filter and the response type (lowpass, bandpass, etc) using the appropriate butter,cheby1, cheby2 command. For example given a butterworth filter, order N=2, and a high pass response, the coefficients [b,a] of the filter are computed using the matlab/gnu-octave inbuilt command 'butter' as [b,a] =butter(N, wc , 'high');

Method 2: Given the pass band (Wp in radians) and Stop band edge (Ws in radians) frequencies, Pass band ripple Rp and stopband attenuation As.

Step 1: Since the frequencies are in radians divide by π to obtain normalized frequencies to Get wp=Wp/pi and ws=Ws/pi. If the frequencies are in Hz. (**note:** In this case the sampling frequency should be given), then obtain normalized frequencies as wp=fp/(fs/2), ws=fstop/(fs/2), where fp, fstop and fs are the passband, stop band and sampling frequencies in Hz

Step 2: Compute the order and cut off frequency as [N, wc] = BUTTORD(wp,ws, Rp, Rs)

Step 3: Compute the Impulse Response [b,a] coefficients of the required IIR filter and the response type as [b,a] =butter(N, wc , 'high');

IMPLEMENTATION OF THE IIR FILTER

1. Once the coefficients of the IIR filter [b,a] are obtained, the next step is to simulate an input sequence x[n], say input of 100, 200 & 400 Hz (with sampling frequency of fs), each of 20/30 points. Choose the frequencies such that they are >, < and = to fc.
2. Filter the input sequence x[n] with Impulse Response, to obtain the output of the filter y[n] using the 'filter' command.
3. Infer the working of the filter (low pass/ high pass, etc).

MATLAB IMPLEMENTATION

BUTTORD Butterworth filter order selection. [N, Wn] = BUTTORD(Wp, Ws, Rp,Rs) returns the order N of the lowest order digital Butterworth filter that loses no more than Rp dB in the passband and has at least Rs dB of attenuation in the stopband. Wp and Ws are the passband and stopband edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample).

For example,

Lowpass: Wp = .1, Ws = .2

Highpass: Wp = .2, Ws = .1

Bandpass: Wp = [.2 .7], Ws = [.1 .8]

Bandstop: Wp = [.1 .8], Ws = [.2 .7]

BUTTORD also returns Wn, the Butterworth natural frequency (or, the "3 dB frequency") to use with BUTTER to achieve the specifications. [N, Wn] =BUTTORD(Wp, Ws, Rp, Rs, 's') does the computation for an analog filter, in which case Wp and Ws are in radians/second. When Rp is chosen as 3 dB, the Wn in BUTTER is equal to Wp in BUTTORD.

BUTTER Butterworth digital and analog filter design. [B,A] = BUTTER(N,Wn) designs an Nth order lowpass digital Butterworth filter and returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cutoff frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. If Wn is a two element vector, Wn = [W1 W2], BUTTER returns an order 2N bandpass filter with passband W1 < W < W2.

[B,A] = BUTTER(N,Wn,'high') designs a highpass filter.

[B,A] = BUTTER(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

BUTTER(N,Wn,'s'), BUTTER(N,Wn,'high','s') and BUTTER(N,Wn,'stop','s') design analog Butterworth filters. In this case, Wn is in [rad/s] and it can be greater than 1.0.

Matlab program for digital butterworth LPF filter

```
clc;
clear all;
close all;
wp=400*pi;
ws=600*pi;
Ap=1;
As=4;
fs=2000;
wp=wp/fs;
ws=ws/fs;
```

%prewarping

```
op=2*tan(wp/2);
os=2*tan(ws/2);
[n,oc]=buttord(op,os,Ap,As,'s');
n=ceil(n);
disp(n);
disp(oc);
[num,den]=butter(n,oc,'low','s');
disp(num);
disp(den);
[nr,dr]=bilinear(num,den,1);
disp(nr);
disp(dr);
[h,w]=freqz(nr,dr,512,fs);
subplot(2,1,1);
plot(w,20*log10(abs(h)));
grid on;
xlabel('frequency axis in Hz');
ylabel('mag. of frequency response in db');
title('mag. frequency response of digital butterworth lpf');
subplot(2,1,2);
```

```
plot(w,(angle(h)));
xlabel('frequency axis in Hz');
ylabel('mag. of phase response in rad');
title('mag. phase response of digital butterworth lpf ');
grid on;
```

Calculations:

Given, wp=400*pi rad/sec. ws=600*pi rad/sec. Ap=-1dB, As=-4dB fs = 2000

Step1: conversion of analog domain specifications into digital domain specification

wp = Ωp/fs => 400π/2000 => **0.2 π**

ws = Ωs/fs => 600π/2000 => **0.3 π**

Step2: prewarping

$\Omega p = \frac{2}{T}\tan(wp/2)$ => ((2 tan(0.2 π /2)) => **0.6498** -------Where T = 1

$\Omega s = \frac{2}{T}\tan(ws/2)$ => ((2 tan(0.3 π /2)) => **1.019**

Step3:

a. Finding the order of the filter design

$N = \frac{\log(10^{-0.1ap}-1 / 10^{-0.1as}-1)}{2\log(\Omega p / \Omega s)}$ => $\frac{\log(10^{0.1}-1 / 10^{0.4}-1)}{2\log(0.6498/1.019)}$ => $\frac{-0.7663}{-0.3909}$

N = 1.96 ≃ **2**

b. Finding the cut-off frequency

$\Omega_0 = \frac{\Omega s / \Omega p}{(10^{-0.1as}-1)^{1/2N}}$ => $\frac{1.019 / 0.6498}{(10^{0.4}-1)^{1/4}}$ => **1.414**

Ωc = Ω_0 * Ωp => 1.414 * .06498 => **0.9189**

Step4: Transfer function H(s) for the filter for N =2.

$$H_N(s) = \frac{1}{S^2+\sqrt{2}s+1}$$

s→ s / Ωc => s / 0.9189

$$H(s) = \frac{1}{\left(\frac{s}{0.9189}\right)^2 + \sqrt{2}\ \left(\frac{s}{0.9189}\right) + 1}$$

$$H(s) = \frac{1}{\frac{s^2 + 1.299s + 0.8443}{0.8443}}$$

$$\boldsymbol{H(s) = \frac{0.8443}{s^2 + 1.299s + 0.8443}}$$

Step5: conversion from analog to digital i.e., H(s)→H(z)

$$\mathbf{s = \frac{2}{T}\ \frac{[1-z^{-1}]}{[1+z^{-1}]}} \qquad \textbf{where T =1}$$

$$\therefore \quad \mathbf{s = 2\ \frac{z-1}{z+1}}$$

$$H_{(Z)} = \frac{0.8443}{2\left(\frac{(z-1)}{z+1}\right) + 1.299 * 2\left(\frac{z-1}{z+1}\right) + 0.8443}$$

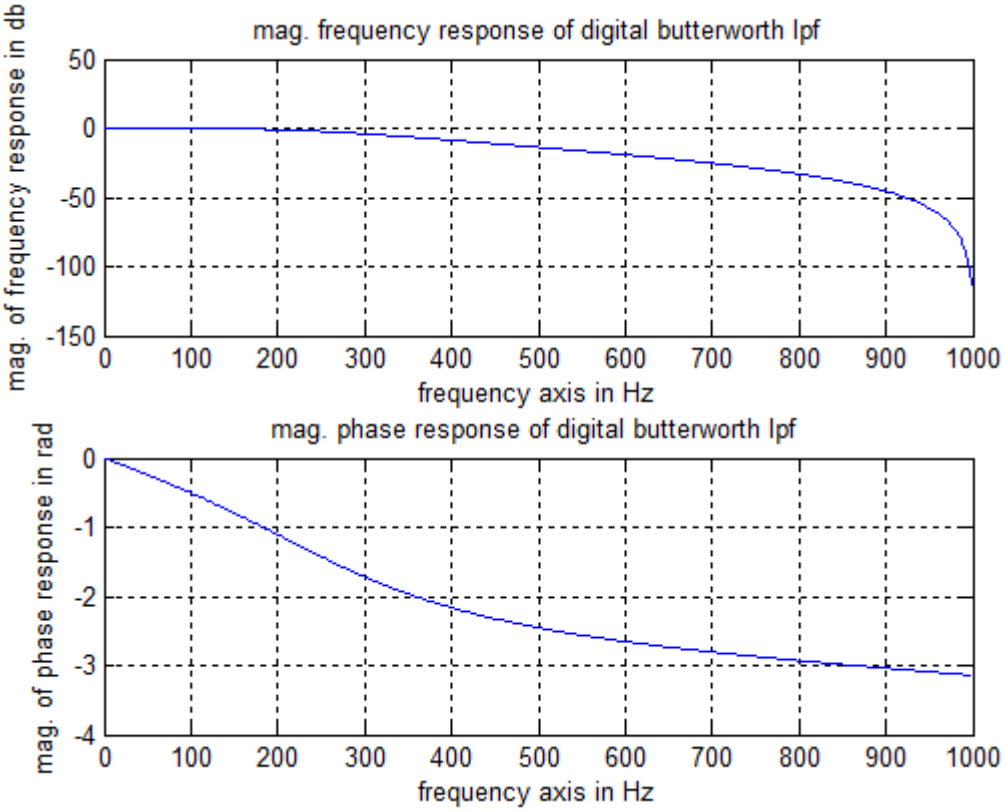
$$H_{(Z)} = \frac{0.8443}{\frac{4z^2 + 4 - 8z + 2.598\,(z^2 - 1) + 0.8443\,(z^2 + 1 + 2z)}{(z+1)^2}}$$

$$H_{(Z)} = \frac{0.8443(z^2 + 1 + 2z)}{4z^2 + 4 - 8z + 2.598z^2 - 2.598 + 0.8443z^2 + 0.8443 + 1.6886z}$$

$$H_{(Z)} = \frac{0.8443z^2 + 0.8443 + 1.6886z}{7.4423z^2 - 6.3114z + 2.2463}$$

$$\therefore \quad \boldsymbol{H_{(Z)} = \frac{0.1134z^2 + 0.2269z + 0.1134}{z^2 - 0.848z + 0.3018}}$$

Figure:



Results and Inference:

N = 2 oc = 0.9184

num = [0 0 0.8446]; den = [1.0000 1.2997 0.8446];

nr = [0.1135 0.2269 0.1135]; dr = [1.0000 -0.8478 0.3016];

The Order of the filter has been designed and verified. Pass band frequency of the desired filter is designed and verified. Also, Filter coefficients are calculated and verified.

Matlab program for digital butterworth HPF filter

```
clc;
clear all;
close all;
wp=600*pi;
ws=400*pi;
Ap=1;
As=4;
fs=2000;
%analog to digital domain conversion
wp=wp/fs;
ws=ws/fs;
%prewarping
op=2*tan(wp/2);
os=2*tan(ws/2);
[n,oc]=buttord(op,os,Ap,As,'s');
n=ceil(n);
disp(n);
disp(oc);
[num,den]=butter(n,oc,'high','s');
disp(num);
disp(den);
[nr,dr]=bilinear(num,den,1);
disp(nr); disp(dr);
[h,w]=freqz(nr,dr,512,fs);
subplot(2,1,1);
plot(w,20*log10(abs(h)));
xlabel('frequency axis in Hz');
ylabel('mag. of frequency response in db');
title('mag. frequency response of digital butterworth hpf');
subplot(2,1,2);
plot(w,(angle(h)));
xlabel('frequency axis in Hz');
ylabel('mag. of phase response in rad');
title('mag. phase response of digital butterworth hpf');
```

Calculations:

Given, wp=600*pi rad/sec. ws=400*pi rad/sec. Ap=-1dB, As=-4dB fs = 2000

Step1: conversion of analog domain specifications into digital domain specification

wp = Ωp/fs => 600π/2000 => **0.3 π**

ws = Ωs/fs => 400π/2000 => **0.2 π**

Step2: prewrapping

Ωp= $\frac{2}{T}$ tan(wp/2) => 2 tan(0.3 π /2) => **1.019** -------Where T = 1

Ωs = $\frac{2}{T}$ tan(ws/2) => 2 tan(0.2 π /2) => **0.6498**

Step3:

a. Finding the order of the filter design

$N = \frac{\log(10^{-0.1ap}-1 / 10^{-0.1as}-1)}{2\log(\Omega p / \Omega s)}$ => $\frac{\log(10^{0.1}-1 / 10^{0.4}-1)}{2\log(1.019/0.6498)}$ => **2**

b. Finding the cut-off frequency

$\Omega_0 = \frac{\Omega p / \Omega s}{(10^{-0.1as}-1)^{1/2N}}$ => $\frac{1.019 / 0.6498}{(10^{0.4}-1)^{1/4}}$ => **1.4142**

Ωc = Ωp / Ω_0 => 1.019 / 1.4142 => **0.7206**

Step4: H(s) for N = 2

$H_N(s) = 1 / s^2 + \sqrt{2}s + 1$

s→ Ωc / s => 0.7206 / s

$$H(s) = \frac{1}{\left(\frac{0.7206}{s}\right)^2 + \sqrt{2}\left(\frac{0.7206}{s}\right) + 1}$$

$$H(s) = \frac{1}{\frac{0.5193 + 1.019S + s^2}{s^2}}$$

$$\therefore \boldsymbol{H(s) = \frac{s^2}{s^2 + 1.019s + 0.5193}}$$

Step5:

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

Where T=1

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

$$H_{(Z)} = \frac{\left\{2\frac{(z-1)}{(z+1)}\right\}^2}{\left\{2\frac{(z-1)}{(z+1)}\right\}^2 + 1.019 * 2\frac{(z-1)}{(z+1)} + 0.5193}$$

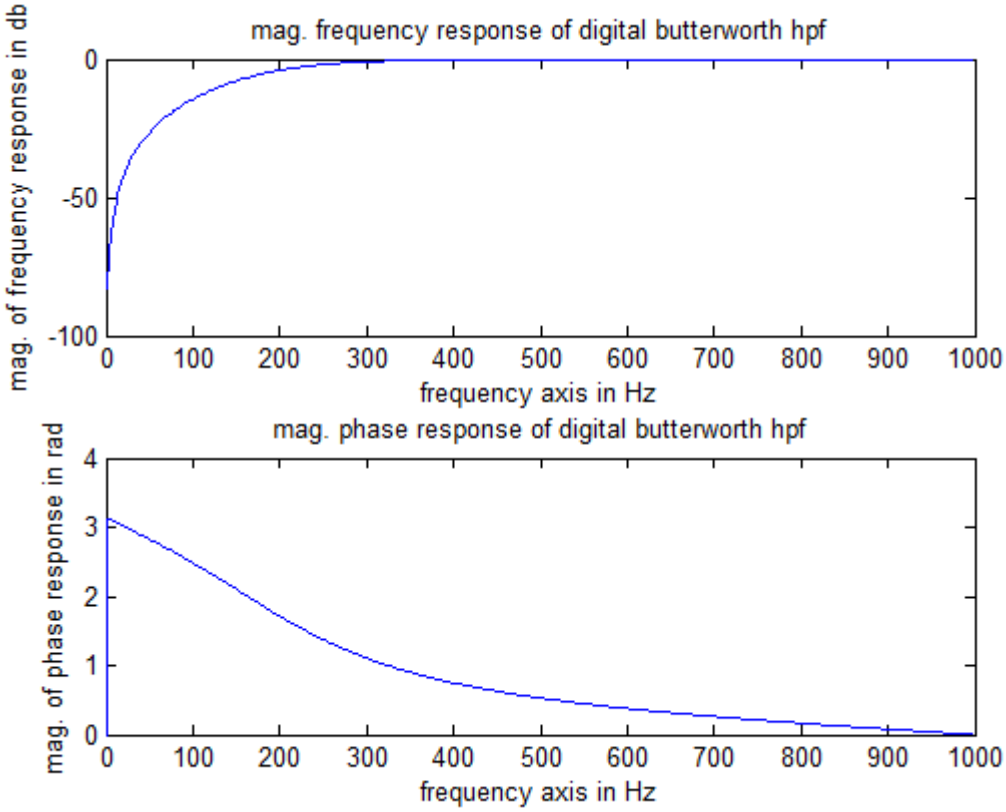
$$H_{(Z)} = \frac{\frac{4(z^2+1-2z)}{(z+1)^2}}{\frac{4(z^2+1-2z)+2.038(z^2-1)+0.5193(z^2+1+2z)}{(z+1)^2}}$$

$$H_{(Z)} = \frac{4z^2+4-8z}{4z^2+4-8z+2.038z^2-2.038+0.5193z^2+0.5193+1.0386z}$$

$$H_{(Z)} = \frac{4z^2-8z+4}{6.5573z^2-6.9614z+2.4813}$$

$$\therefore \boldsymbol{H_{(Z)} = \frac{0.61z^2-1.22z+0.61}{z^2-1.0616z+0.378}}$$

Figure:



Results and Inference:

N = 2 oc = 0.7206

num =[1 0 0]; den = [1.0000 1.0191 0.5192];

nr = [0.6100 -1.2200 0.6100]; dr = [1.0000 -1.0616 0.3784];

The Order of the filter have been designed and verified. Pass band frequency of the desired filter is designed and verified. Also, Filter coefficients are calculated and verified.

EXPERIMENT-08

AIM:- TO DESIGN DIGITAL LPF/HPF CHEBYSHEV IIR FILTER TO MEET GIVEN SPECIFICATIONS

Matlab program for digital Chebyshev LPF filter

```
clc;
clear all;
close all;
wp=200*pi;
ws=1000*pi;
Ap=2;
As=20;
fs=4000;
%analog to digital domain conversion
wp=wp/fs;
ws=ws/fs;
%prewarping
op=2*tan(wp/2);
os=2*tan(ws/2);
[n,oc]=cheb1ord(op,os,Ap,As,'s');
N=ceil(n);
disp(N);
disp(oc);
[num,den]=cheby1(n,Ap,oc,'low','s');
disp(num);
disp(den);
[nr,dr]=bilinear(num,den,1);
disp(nr);
disp(dr);
[h,w]=freqz(nr,dr,512,fs);
subplot(2,1,1);
plot(w,20*log10(abs(h)));
grid on;
xlabel('frequency axis in Hz');
ylabel('mag. of frequency response in db');
title('mag. frequency response of digital chebyshev lpf');
```

```
subplot(2,1,2);
plot(w,(angle(h)));
xlabel('frequency axis in Hz');
ylabel('mag. of phase response in rad');
title('mag. phase response of digital chebyshev lpf');
grid on;
```

Calculations:

Given, Ωp=200*pi rad/sec. Ωs=1000*pi rad/sec. Ap=-2dB, As=-20dB fs = 4000 T=1

Ωp=Ωp/fs => 0.1570

Ωs=Ωs/fs => 0.7853

wp = 2/T tan(Ωp/2) => 2.tan(0.1570/2) => 0.1573

ws = 2/T tan(Ωs/2) => 2.tan(0.7853/2) => 0.8283

$$\mathbf{N} = \frac{\cosh^{-1}((10^{-as/10}-1 / 10^{-ap/10}-1))^{1/2}}{\cosh^{-1}(\Omega s / \Omega p)} \Rightarrow \frac{\cosh^{-1}((10^{2}-1 / 10^{0.2}-1))^{1/2}}{\cosh^{-1}(0.7853 / 0.1570)} \Rightarrow 1.421 \simeq \mathbf{2}$$

$$\boldsymbol{\varepsilon} = \sqrt{10^{\,-0.1\,ap}-1} = 0.7647$$

$$D = \frac{1+\sqrt{1+\varepsilon^2}}{\varepsilon} \Rightarrow \frac{1+\sqrt{1+0.7647^2}}{0.7647} \Rightarrow 2.9539$$

a = ½ [$D^{1/N} - D^{-1/N}$] => ½ [$(2.953)^{1/2} - (2.953)^{-1/2}$] => 0.5682

b = ½ [$D^{1/N} + D^{-1/N}$] => ½ [$(2.953)^{1/2} + (2.953)^{-1/2}$] => 1.1501

$s_k = \sigma_k + j\,\Omega_k$

$$\sigma_K = -\boldsymbol{a}\sin\left(\frac{(2k-1)\pi}{2N}\right) \qquad \Omega_K = \boldsymbol{b}\cos\left(\frac{(2k-1)\pi}{2N}\right) \qquad ; \quad k=1,2$$

k=1; σ_1 = -0.4019 Ω_1 = 0.8133 s_1 = -0.4019 + j0.8133

k=2; σ_2 = -0.4019 Ω_2 = -0.8133 s_2 = -0.4019 - j0.8133

$$H(s) = \frac{k_N}{(s-s1)\,(s-s2)} \Rightarrow \frac{k_N}{(s-(-0.4019 + j0.8133))\,(s-(-0.4019 - j0.8133))}$$

$k_N = \frac{b_0}{\sqrt{1+\varepsilon^2}}$ => 0.6536

$H(s) = \frac{0.6536}{s^2 + 0.80365 + 0.8229}$ $s \rightarrow s / 0.1573$ ------ **Denormalization**

$H(s) = \frac{0.6536}{(s / 0.1573)^2 + 0.80365(s / 0.1573) + 0.8229}$

$H(s) = \frac{0.01619}{s^2 + 0.1264s + 0.02036}$ $s \rightarrow 2 / T (1 – z^{-1} / 1 + z^{-1})$ ----- **Digitization**

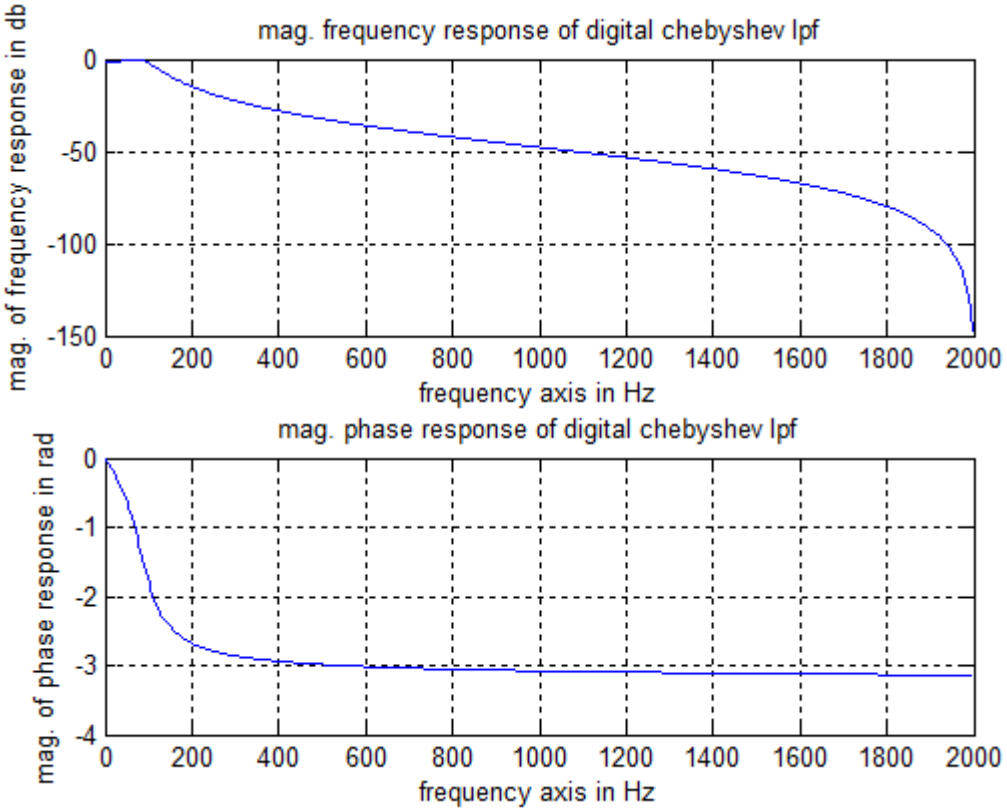
$H(z) = \frac{0.01619}{2/T (1 – z^{-1} / 1+z^{-1})^2 + 0.1264\ 2 / T (1 – z^{-1} / 1+z^{-1}) + 0.02036}$

$H(z) = \frac{0.01619 (1 + z^{-1})^2}{z^2 (1 – z^{-1})^2 + 0.1264*2 (1 – z^{-1})(1+z^{-1}) + 0.02036 (1 + z^{-1})^2}$

$H(z) = \frac{0.01619 + 0.01619z^{-2} + 0.03238z^{-1}}{3.76756z^{-2} – 7.95928z^{-1} + 4.27316}$ ---------- ***divide both num and den by b_0***

$H(z) = \frac{0.00378z^{-2} + 0.00757z^{-1} + 0.00378z^0}{0.8816z^{-2} – 1.8625z^{-1} + 1}$

Figure:



Results and Inference:

N = 2 oc = 0.1574

num = [0 0 0.0162]; den = [1.0000 0.1265 0.0204];

nr = [0.0038 0.0076 0.0038]; dr = [1.0000 -1.8625 0.8816];

The Order of the filter have been designed and verified. Pass band frequency of the desired filter is designed and verified. Also, Filter coefficients are calculated and verified.

Matlab program for digital Chebyshev HPF filter

```
clc;
clear all;
close all;
ws=200*pi;
wp=1000*pi;
Ap=2;
As=20;
fs=4000;
%analog to digital domain conversion
wp=wp/fs;
ws=ws/fs;
%prewarping
op=2*tan(wp/2);
os=2*tan(ws/2);
[n,oc]=cheb1ord(op,os,Ap,As,'s');
N=ceil(n);
disp(N);
disp(oc);
[num,den]=cheby1(n,Ap,oc,'high','s');
disp(num);
disp(den);
[nr,dr]=bilinear(num,den,1);
disp(nr);
disp(dr);
[h,w]=freqz(nr,dr,512,fs);
subplot(2,1,1);
plot(w,20*log10(abs(h)));
grid on;
xlabel('frequency axis in Hz');
ylabel('mag. of frequency response in db');
title('mag.frequency response of digital chebyshev hpf');
subplot(2,1,2);
plot(w,(angle(h)));
xlabel('frequency axis in Hz');
```

```
ylabel('mag. of phase response in rad');
title('mag. phase response of digital chebyshev hpf');
grid on;
```

Calculations:

Given, Ωp=1000*pi rad/sec, Ωs=200*pi rad/sec Ap=-2dB, As=-20dB fs = 4000 T=1

Wp=Ωp/fs => 0.7853

Ws=Ωs/fs => 0.1570

Ω'_p = 2/T tan (Wp/2) => 2.tan (0.7853/2) => 0.8283

Ω'_s = 2/T tan (Ws/2) => 2.tan (0.1570/2) => 0.1573

Step1: Normalized=> $\Omega'_p = 1r/s$ $\Omega'_s = 5.26r/s$

Step2: To find the order of the filter

$$\mathbf{N} = \frac{\cosh^{-1}((10^{-as/10}-1 / 10^{-ap/10}-1))^{1/2}}{\cosh^{-1}(\Omega's / \Omega'p)} \Rightarrow \frac{\cosh^{-1}((10^{2}-1 / 10^{0.2}-1))^{1/2}}{\cosh^{-1}(5.26)} \Rightarrow 1.38 \simeq \mathbf{2}$$

$$\boldsymbol{\varepsilon} = \sqrt{10^{-0.1\,ap}-1} = 0.7647$$

$$D = \frac{1+\sqrt{1+\varepsilon^2}}{\varepsilon} \Rightarrow \frac{1+\sqrt{1+0.7647^2}}{0.7647} \Rightarrow 2.9539$$

$$\mathbf{a = \tfrac{1}{2}\ [\ D^{1/N} - D^{-1/N}\ ] \Rightarrow \tfrac{1}{2}\ [(2.953)^{1/2} - (2.953)^{-1/2}] \Rightarrow 0.5682}$$

$$\mathbf{b = \tfrac{1}{2}\ [\ D^{1/N} + D^{-1/N}\ ] \Rightarrow \tfrac{1}{2}\ [(2.953)^{1/2} + (2.953)^{-1/2}] \Rightarrow 1.1501}$$

$$\sigma_K = -\boldsymbol{a}\sin\left(\frac{(2\boldsymbol{k}-1)\pi}{2\boldsymbol{N}}\right) \qquad \Omega_K = \boldsymbol{b}\cos\left(\frac{(2\boldsymbol{k}-1)\pi}{2\boldsymbol{N}}\right)$$

$s_k = \sigma_k + j\,\Omega_k$

s1 = -0.402 + j0.813

s2 = -0.402 - j0.813

k	σ_K	Ω_K
1	-0.4016	0.813
2	-0.4016	-0.813

∴ **Transfer function,**

$$H_{(s)} = \frac{k_N}{(s-s1)(s-s2)}$$

$$H_{(s)} = \frac{k_N}{(s+0.402-j0.813)(s+0.402+j0.813)}$$

$$\mathrm{H_{(s)}} = \frac{\mathrm{k_N}}{(s+0.402)^2+(0.813)^2}$$

$$H_{(s)} = \frac{k_N}{s^2+0.804s+0.8226}$$

$$K_{N=}\frac{b_0}{\sqrt{1+\Sigma^2}} = \frac{0.8226}{\sqrt{1+(0.764)^2}} \cong 0.6536$$

$$\therefore \quad \boldsymbol{H_{(s)}} = \frac{\mathbf{0.6536}}{\boldsymbol{s^2+0.804s+0.8226}}$$

Denormalizing:-

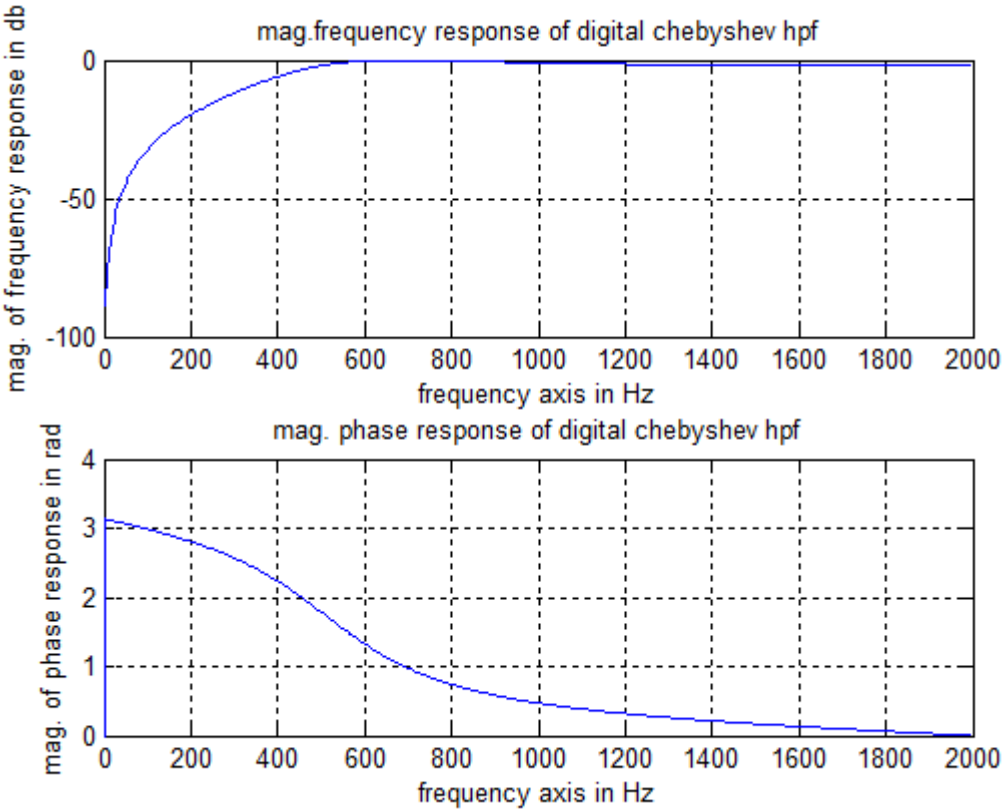
$$H_{a(s)} = H_{(s)} \mid s \rightarrow \Omega'_p = \frac{\mathrm{wp}}{s}$$

$$H_{a(s)} = \frac{0.7978s^2}{s^2 + 0.808s+0.832}$$

$$H_{(z)} = H_{a(s)} \mid s \rightarrow \frac{2}{T}\left(\frac{1-Z^{-1}}{1+Z^{-1}}\right)$$

$$\boldsymbol{H_{(Z)}} = \frac{\mathbf{0.4925z^{-2}-0.9849z^{-1}+0.4925}}{\boldsymbol{z^{-2}-0.9815z^{-1}+0.4984}}$$

Figure:



Results and Inference:

N = 2 oc = 0.8284

num = [0.7943 0 0]; den = [1.0000 0.8091 0.8338];

nr = [0.4925 -0.9849 0.4925]; dr = [1.0000 -0.9815 0.4984];

The Order of the filter have been designed and verified. Pass band frequency of the desired filter is designed and verified. Also, Filter coefficients are calculated and verified.

Challenge experiment:

1. Write a matlab program for IIR filter as per calculations done in theory class.

EXPERIMENT-09

AIM:- TO DESIGN A DIGITAL FIR FILTER TO MEET GIVEN SPECIFICATIONS.

Theory:

There are two types of systems – Digital filters (perform signal filtering in time domain) and spectrum analyzers (provide signal representation in the frequency domain).

The design of a digital filter is carried out in 3 steps- specifications, approximations and implementation.

DESIGNING AN FIR FILTER (using window method):

Method I: *Given the order N, cutoff frequency fc, sampling frequency fs and the window.*

Step 1: Compute the digital cut-off frequency Wc (in the range - π < Wc < π, with π corresponding to fs/2) for fc and fs in Hz. For example let fc=400Hz,fs=8000Hz

Wc = 2* π * fc / fs = 2* π * 400/8000 = 0.1* π radians

For matlab the Normalized cut-off frequency is in the range 0 and 1, where 1 corresponds to fs/2 (i.e.,fmax)). Hence to use the matlab commands

wc = fc / (fs/2) = 400/(8000/2) = 0.1

Note: If the cut off frequency is in radians then the normalized frequency is computed as

wc = Wc / π

Step 2: Compute the Impulse Response h(n) of the required FIR filter using the given Window type and the response type (lowpass, bandpass, etc). For example given a rectangular window, order N=20, and a high pass response, the coefficients (i.e., as h[n] samples) of the filter are computed using the matlab inbuilt command 'fir1'

h =fir1(N, wc , 'high', boxcar(N+1));

Note: In theory we would have calculated h[n]=hd[n]×w[n], where hd[n] is the desired impulse response (low pass/ high pass,etc given by the sinc function) and w[n] is the window coefficients.

We can also plot the window shape as stem(boxcar(N)).

Plot the frequency response of the designed filter h(n) using the **freqz** function and observe the type of response (lowpass / highpass).

Method 2: *Given the pass band (wp in radians) and Stop band edge (ws in radians) frequencies, Pass band ripple (Rp) and stopband attenuation (As).*

Step 1: Select the window depending on the stop-band attenuation required. Generally if As>40 dB, choose Hamming window. (Refer table)

Step 2: Compute order **N** based on the edge frequencies as Transition bandwidth = tb = ws-wp; N=ceil (6.6*pi/tb);

Step 3: Compute the digital cut-off frequency Wc as Wc=(wp+ws)/2. Now compute the normalized frequency in the range 0 to 1 for matlab/gnu-octave as wc=Wc/pi;

Note: *In step 2 if frequencies are in Hz, then obtain radian frequencies (for computation of tb and N) as wp=2*pi*fp/fs, ws=2*pi*fstop/fs, where fp, fstop and fs are the passband, stop band and sampling frequencies in Hz*

Step 4: Compute the Impulse Response h(n) of the required FIR filter using N, selected window, type of response(low/high, etc) using 'fir1' as in step 2.

IMPLEMENTATION OF THE FIR FILTER

1. Once the coefficients of the FIR filter h[n] are obtained, the next step is to simulate an input sequence x[n], say input of 100, 200 & 400 Hz (with sampling frequency of fs), each of 20/30 points. Choose the frequencies such that they are >, < and = to fc.
2. Convolve input sequence x[n] with Impulse Response, i.e., x (n)*h (n) to obtain the output of the filter y[n]. We can also use the 'filter' command.
3. Infer the working of the filter (low pass/ high pass, etc).

MATLAB IMPLEMENTATION

FIR1 Function

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

If Wn is a two-element vector, Wn = [W1 W2], FIR1 returns an order N bandpass filter with passband W1 < W < W2. B = FIR1(N,Wn,'high') designs a highpass filter.

B = FIR1(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

If Wn is a multi-element vector, Wn = [W1 W2 W3 W4 W5 ... WN], FIR1 returns an order N multiband filter with bands 0 < W < W1, W1 < W < W2, ..., WN < W < 1.

FREQZ Digital filter frequency response. [H,W] = FREQZ(B,A,N) returns the Npoint complex frequency response vector H and the N-point frequency vector W in radians/sample of the filter whose numerator and denominator coefficients are in vectors B and A. The frequency response is evaluated at N points equally spaced around the upper half of the unit circle. If N isn't specified, it defaults to 512.

For FIR filter enter A=1 and B = h[n] coefficients. Appropriately choose N as 128,256, etc

Commonly used window function characteristics

Window Name	Transition Approximate	Width Exact values	Min. Stop band Attenuation	matlab/gnu-octave Command
Rectangular	$\frac{4\Pi}{M}$	$\frac{1.8\Pi}{M}$	– 21db	B =FIR1(N,Wn,boxcar)
Bartlett	$\frac{8\Pi}{M}$	$\frac{6.1\Pi}{M}$	– 25db	B=FIR1(N,Wn,bartlett)
Hanning	$\frac{8\Pi}{M}$	$\frac{6.2\Pi}{M}$	– 44db	B=FIR1(N,Wn,hanning)
Hamming	$\frac{8\Pi}{M}$	$\frac{6.6\Pi}{M}$	– 53db	B= FIR1(N,Wn)
Blackman	$\frac{12\Pi}{M}$	$\frac{11\Pi}{M}$	– 74db	B=FIR1(N,Wn,blackman)

% Design and implementation of FIR low pass filter:

MATLAB Program:

```
clc;
wp=0.3*pi;
ws=0.45*pi;
ap=-3;
as=-50;
wt=ws-wp;
n1=ceil((8*pi)/wt);
wc=wp/pi;
N=n1+rem(n1-1,2);
disp(N);
wn=hamming(N);
disp(wn);
a=(N-1)/2;
h=fir1(N-1,wc,wn);    % fir1 function will calculate both hd(n) and h(n)
disp(h);
figure;
freqz(h);
figure;
n=0:N-1;
stem(n,h);
```

```
xlabel('time axis');
ylabel('h(n)');
title('impulse response of FIR lowpass filter');
```

Calculations:

wp=0.3*pi r/s, ws=0.45*pi r/s, ap=-3dB, as=-50dB

Step1: The transition width(wt)

wt=ws-wp => 0.45π - 0.3π =>0.15π =>0.4712 r/s

Step2: length of frequency response/windowing coefficient

length => n1=8 π/wt

n1=8 π/0.4712 =>54

N=n1+1 =>54+1 =>55

Note: N value should be odd to get the frequency response symmetric

Step3: Desired impulse response

$hd(n) = \sin wc(n-\alpha) / \pi(n-\alpha)$; $n \neq \alpha$

wc / π ; $n = \alpha$

the windowing function w(n)

$w(n) = 0.54 – 0.46\cos(2\pi n / N-1)$ $0 \leq n \leq N-1$

$\alpha = N-1/2$ => 55-1/2 => 27

wc=wp/π => 0.3π => 0.3

Therefore, impulse response of the filter design is <u>h(n) = hd(n).w(n)</u>

Step4:

1. w(n) => w(27) = 0.54 – 0.46cos((2π*27) / (55-1)) => 1
   hd(27) = wc/π => 0.3
   h(27) = w(27)*hd(27) => 1*0.3 => 0.3

2. w(26) = 0.54 – 0.46cos((2π*26) / (55-1)) => 0.9969
   hd(26) = sin(0.3π(26-27)) / π(26-27) => 0.2575
   h(26) = w(26)*hd(26) => 0.9969*0.2575 => 0.2567

3. w(25) = 0.54 – 0.46cos((2π*25) / (55-1)) => 0.9876
   hd(25) = sin(0.3π(25-27)) / π(25-27) => 0.1513
   h(25) = w(25)*hd(25) => 0.9876*0.1513 => 0.15
4. w(24) = 0.54 – 0.46cos((2π*24) / (55-1)) => 0.9722

hd(24) = sin(0.3π(24-27)) / π(24-27) => 0.03278
h(24) = w(24)*hd(24) => 0.9722*0.03278 => 0.03186

5. w(23) = 0.54 – 0.46cos((2π*23) / (55-1)) => 0.9510
   hd(23) = sin(0.3π(23-27)) / π(23-27) => - 0.04677
   h(23) = w(23)*hd(23) => 0.9510*- 0.04677 => - 0.04447

6. w(22) = 0.54 – 0.46cos((2π*22) / (55-1)) => 0.9243
   hd(22) = sin(0.3π(22-27)) / π(22-27) => - 0.06366
   h(22) = w(22)*hd(22) => 0.9243*- 0.06366 => - 0.05884

7. w(28) = 0.54 – 0.46cos((2π*28) / (55-1)) => 0.9969
   hd(28) = sin(0.3π(28-27)) / π(28-27) => 0.2575
   h(28) = w(28)*hd(28) => 0.9969*0.2575 => 0.2567

8. w(29) = 0.54 – 0.46cos((2π*29) / (55-1)) => 0.9876
   hd(29) = sin(0.3π(29-27)) / π(29-27) => 0.1513
   h(29) = w(29)*hd(29) => 0.9876*0.1513 => 0.15

9. w(30) = 0.54 – 0.46cos((2π*30) / (55-1)) => 0.9722
   hd(30) = sin(0.3π(30-27)) / π(30-27) => 0.03278
   h(30) = w(30)*hd(30) => 0.9722*0.03278 => 0.03186

10. w(31) = 0.54 – 0.46cos((2π*31) / (55-1)) => 0.9510
    hd(31) = sin(0.3π(31-27)) / π(31-27) => - 0.04677
    h(31) = w(31)*hd(31) => 0.9510*- 0.04677 => - 0.04447

11. w(32) = 0.54 – 0.46cos((2π*32) / (55-1)) => 0.9243
    hd(32) = sin(0.3π(32-27)) / π(32-27) => - 0.06366
    h(32) = w(32)*hd(32) => 0.9243*- 0.06366 => - 0.05884

Figure1:

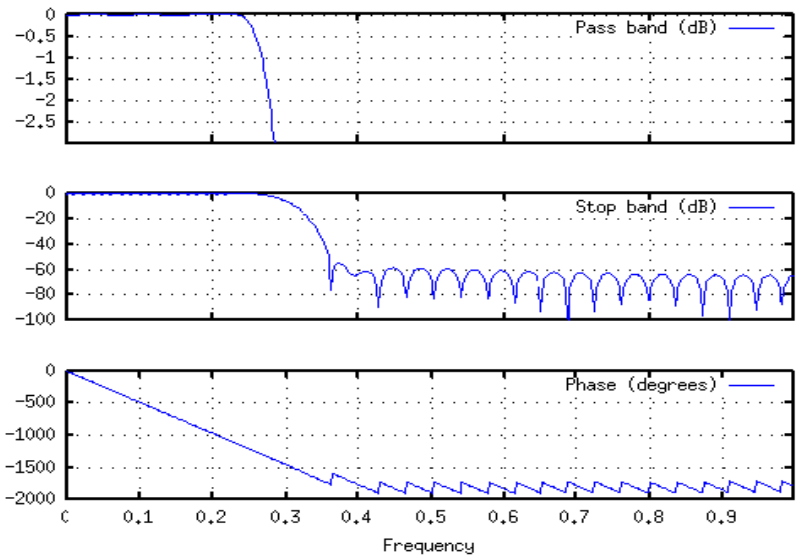
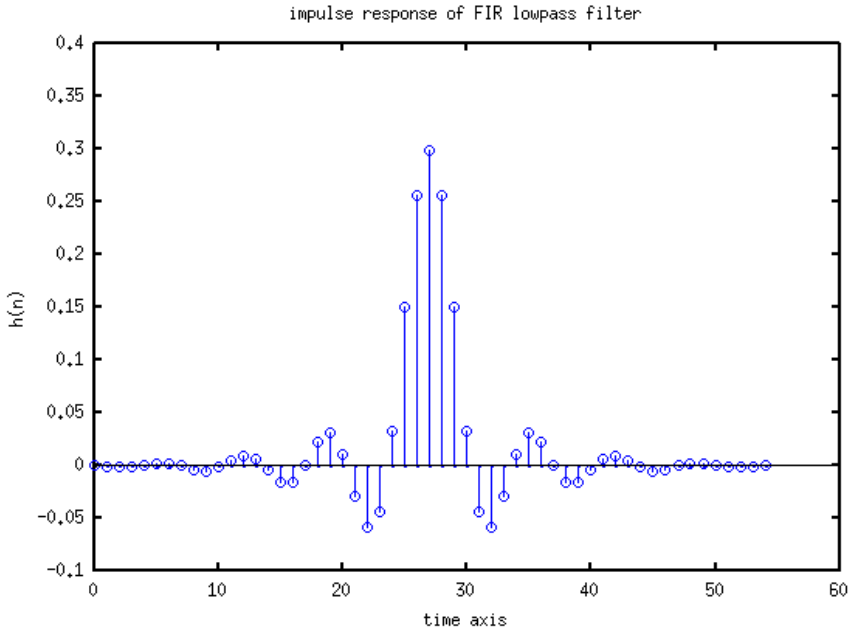


Figure2:



Results and Inference:

N =55

w(n) =>

0.0800, 0.0831, 0.0924, 0.1077, 0.1289, 0.1557, 0.1876, 0.2243, 0.2653, 0.3100, 0.3578,

0.4081, 0.4601, 0.5133, 0.5667, 0.6199, 0.6719, 0.7222, 0.7700, 0.8147, 0.8557, 0.8924,

0.9243, 0.9511, 0.9723, 0.9876, 0.9969, 1.0000, 0.9969, 0.9876, 0.9723, 0.9511, 0.9243,

0.8924, 0.8557, 0.8147, 0.7700, 0.7222, 0.6719, 0.6199, 0.5667, 0.5133, 0.4601, 0.4081,

0.3578, 0.3100, 0.2653, 0.2243, 0.1876, 0.1557, 0.1289, 0.1077, 0.0924, 0.0831, 0.0800

The Order of the filter have been designed and verified. Window function coefficients are calculated and verified. Also, Filter response is obtained and verified for symmetry.

Challenge experiment:

1. Write a matlab program for FIR filter design using other window functions.
2. Write a matlab program to design a FIR filter using frequency sampling technique.

TMS320C6713 DSK

The C6713 DSK has a TMS320C6713 DSP onboard that allows full-speed verification of code with Code Composer Studio. The C6713 DSK provides:

- A USB Interface
- SDRAM and ROM
- An analog interface circuit for Data conversion (AIC)
- An I/O port
- Embedded JTAG emulation support

Connectors on the C6713 DSK provide DSP external memory interface (EMIF) and peripheral signals that enable its functionality to be expanded with custom or third party daughter boards.

The DSK provides a C6713 hardware reference design that can assist you in the development of your own C6713-based products. In addition to providing a reference for interfacing the DSP to various types of memories and peripherals, the design also addresses power, clock, JTAG, and parallel peripheral interfaces.

The C6713 DSK includes a stereo codec. This Analog Interface Circuit (AIC) has the following characteristics:

High-Performance Stereo *Codec*

- 90-dB SNR Multibit Sigma-Delta ADC (A-weighted at 48 kHz)
- 100-dB SNR Multibit Sigma-Delta DAC (A-weighted at 48 kHz)
- 1.42 V – 3.6 V Core Digital Supply: Compatible With TI C54x DSP Core Voltages
- 2.7 V – 3.6 V Buffer and Analog Supply: Compatible Both TI C54x DSP Buffer Voltages
- 8-kHz – 96-kHz Sampling-Frequency Support

Software Control Via TI McBSP-Compatible Multiprotocol Serial Port

- I 2 C-Compatible and SPI-Compatible Serial-Port Protocols
- Glueless Interface to TI McBSPs

Audio-Data Input/output Via TI McBSP-Compatible Programmable Audio Interface

- I 2 S-Compatible Interface Requiring Only One McBSP for both ADC and DAC
- Standard I 2 S, MSB, or LSB Justified-Data Transfers
- 16/20/24/32-Bit Word Lengths

The C6713DSK has the following features:

The 6713 DSK is a low-cost standalone development platform that enables customers to evaluate and develop applications for the TI C67XX DSP family. The DSK also serves as a hardware reference design for the TMS320C6713 DSP. Schematics, logic equations and application notes are available to ease hardware development and reduce time to market.

The DSK uses the 32-bit EMIF for the SDRAM (CE0) and daughter card expansion interface (CE2 and CE3). The Flash is attached to CE1 of the EMIF in 8-bit mode.

An on-board AIC23 codec allows the DSP to transmit and receive analog signals. McBSP0 is used for the codec control interface and McBSP1 is used for data. Analog audio I/O is done through four 3.5mm audio jacks that correspond to microphone input, line input, line output and headphone output. The codec can select the microphone or the line input as the active input. The analog output is driven to both the line out (fixed gain) and headphone (adjustable gain) connectors. McBSP1 can be re-routed to the expansion connectors in software.

A programmable logic device called a CPLD is used to implement glue logic that ties the board components together. The CPLD has a register based user interface that lets the user configure the board by reading and writing to the CPLD registers. The registers reside at the midpoint of CE1.

The DSK includes 4 LEDs and 4 DIP switches as a simple way to provide the user with interactive feedback. Both are accessed by reading and writing to the CPLD registers.

An included 5V external power supply is used to power the board. On-board voltage regulators provide the 1.26V DSP core voltage, 3.3V digital and 3.3V analog voltages. A voltage supervisor monitors the internally generated voltage, and will hold the board in reset until the supplies are within operating specifications and the reset button is released. If desired, JP1 and JP2 can be used as power test points for the core and I/O power supplies.

Code Composer communicates with the DSK through an embedded JTAG emulator with a USB host interface. The DSK can also be used with an external emulator through the external JTAG connector.

TMS320C6713 DSP Features

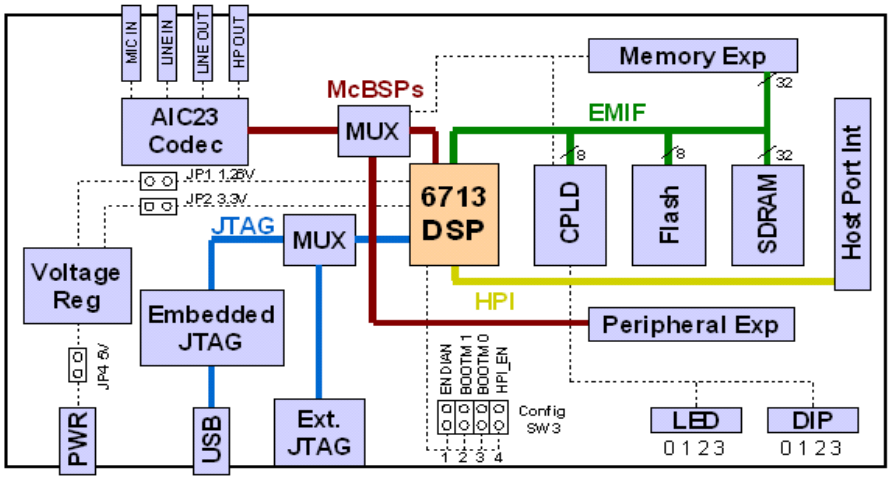
- Highest-Performance Floating-Point Digital Signal Processor (DSP):
  - Eight 32-Bit Instructions/Cycle
  - 32/64-Bit Data Word
  - 300-, **225-**, 200-MHz (GDP), and **225-**, 200-, 167-MHz (PYP) Clock Rates
  - 3.3-, 4.4-, 5-, 6-Instruction Cycle Times
  - 2400/1800, 1800/1350, 1600/1200, and 1336/1000 MIPS /MFLOPS
  - Rich Peripheral Set, Optimized for Audio
  - Highly Optimized C/C++ Compiler
  - Extended Temperature Devices Available
- Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core

- Eight Independent Functional Units:
  - Two ALUs (Fixed-Point)
  - Four ALUs (Floating- and Fixed-Point)
  - Two Multipliers (Floating- and Fixed-Point)
- Load-Store Architecture With 32 32-Bit General-Purpose Registers
- Instruction Packing Reduces Code Size
- All Instructions Conditional

- Instruction Set Features
  - Native Instructions for IEEE 754
    - Single- and Double-Precision
  - Byte-Addressable (8-, 16-, 32-Bit Data)
  - 8-Bit Overflow Protection
  - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- L1/L2 Memory Architecture
  - 4K-Byte L1P Program Cache (Direct-Mapped)
  - 4K-Byte L1D Data Cache (2-Way)
  - 256K-Byte L2 Memory Total: 64K-Byte L2 Unified Cache/Mapped RAM, and 192K-Byte Additional L2 Mapped RAM
- Device Configuration
  - Boot Mode: HPI, 8-, 16-, 32-Bit ROM Boot
  - Endianness: Little Endian, Big Endian
- 32-Bit External Memory Interface (EMIF)
  - Glueless Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
  - 512M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (16 Independent Channels)
- 16-Bit Host-Port Interface (HPI)
- Two Multichannel Audio Serial Ports (McASPs)
  - Two Independent Clock Zones Each (1 TX and 1 RX)
  - Eight Serial Data Pins Per Port:
    Individually Assignable to any of the Clock Zones
  - Each Clock Zone Includes:
    - Programmable Clock Generator
    - Programmable Frame Sync Generator
    - TDM Streams From 2-32 Time Slots
    - Support for Slot Size:
      8, 12, 16, 20, 24, 28, 32 Bits

- Data Formatter for Bit Manipulation
- Wide Variety of I2S and Similar Bit Stream Formats
- Integrated Digital Audio Interface Transmitter (DIT) Supports:
  - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
  - Up to 16 transmit pins
  - Enhanced Channel Status/User Data
- Extensive Error Checking and Recovery

- Two Inter-Integrated Circuit Bus (I^2C Bus™) Multi-Master and Slave Interfaces
- Two Multichannel Buffered Serial Ports:
  - Serial-Peripheral-Interface (SPI)
  - High-Speed TDM Interface
  - AC97 Interface
- Two 32-Bit General-Purpose Timers
- Dedicated GPIO Module With 16 pins (External Interrupt Capable)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG †) Boundary-Scan-Compatible
- Package Options:
  - 208-Pin PowerPAD™ Plastic (Low-Profile) Quad Flatpack (PYP)
  - 272-BGA Packages (GDP and ZDP)
- 0.13-µm/6-Level Copper Metal Process
  - CMOS Technology
- 3.3-V I/Os, 1.2 ‡ -V Internal (GDP & PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)



TMS320C6713 DSK Overview Block Diagram

INTRODUCTION TO CODE COMPOSER STUDIO

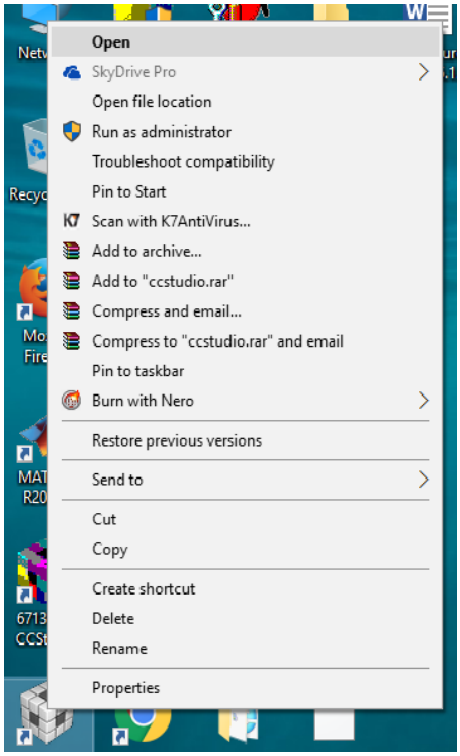
Code Composer is the DSP industry's first fully integrated development environment (IDE) with DSP-specific functionality. With a familiar environment liked MS-based C++TM, Code Composer lets you edit, build, debug, profile and manage projects from a single unified environment. Other unique features include graphical signal analysis, injection/extraction of data signals via file I/O, multi-processor debugging, automated testing and customization via a C-interpretive scripting language and much more.

CODE COMPOSER FEATURES INCLUDE:

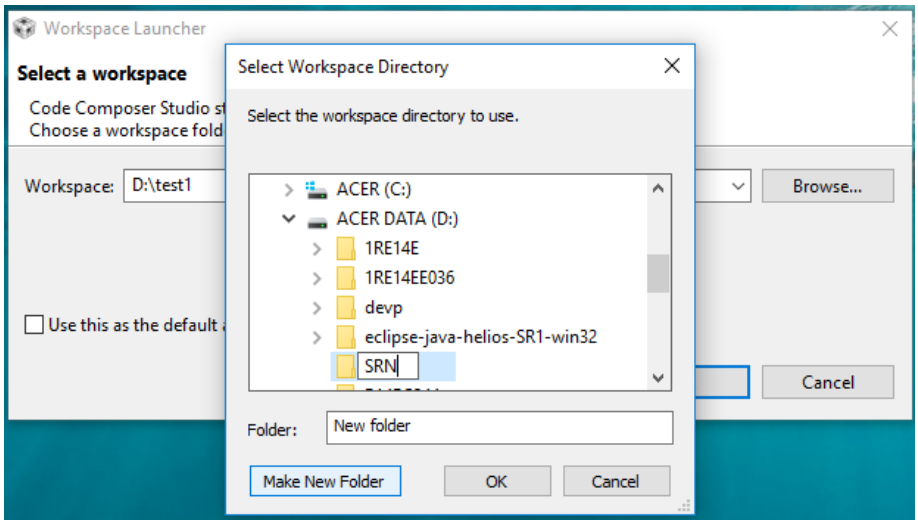
- IDE
- Debug IDE
- Advanced watch windows
- Integrated editor
- File I/O, Probe Points, and graphical algorithm scope probes
- Advanced graphical signal analysis
- Interactive profiling
- Automated testing and customization via scripting
- Visual project management system
- Compile in the background while editing and debugging
- Multi-processor debugging
- Help on the target DSP

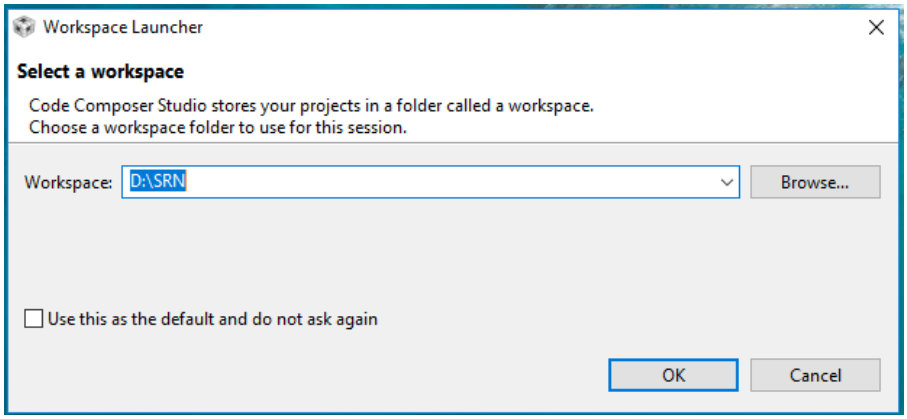
Procedure for interfacing programs using CODE COMPOSER STUDIO6.1.3

1. Open Code Composer Studio 6.1.3 icon which is created on the desktop.

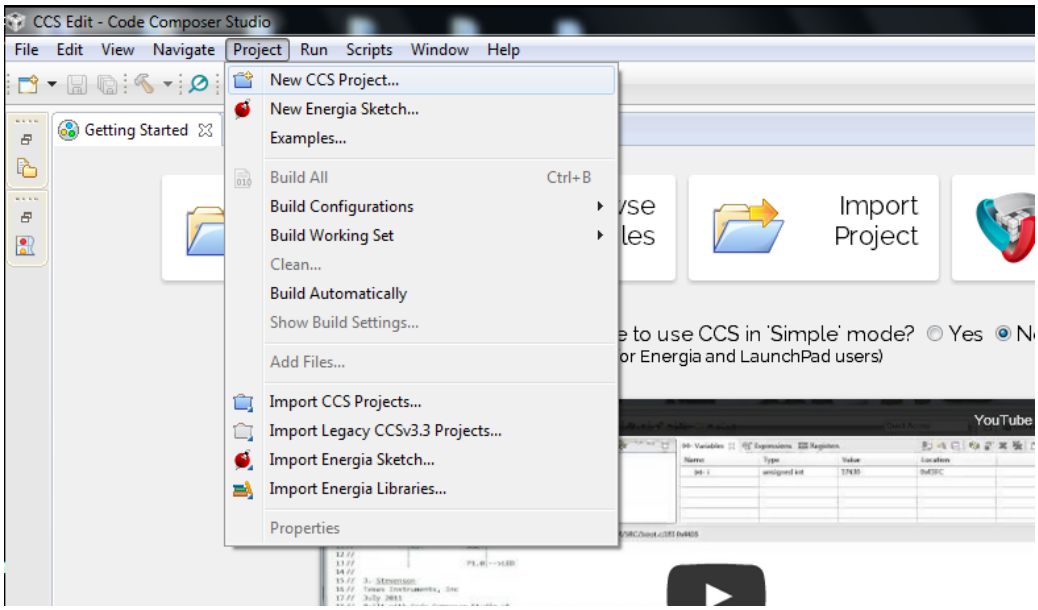


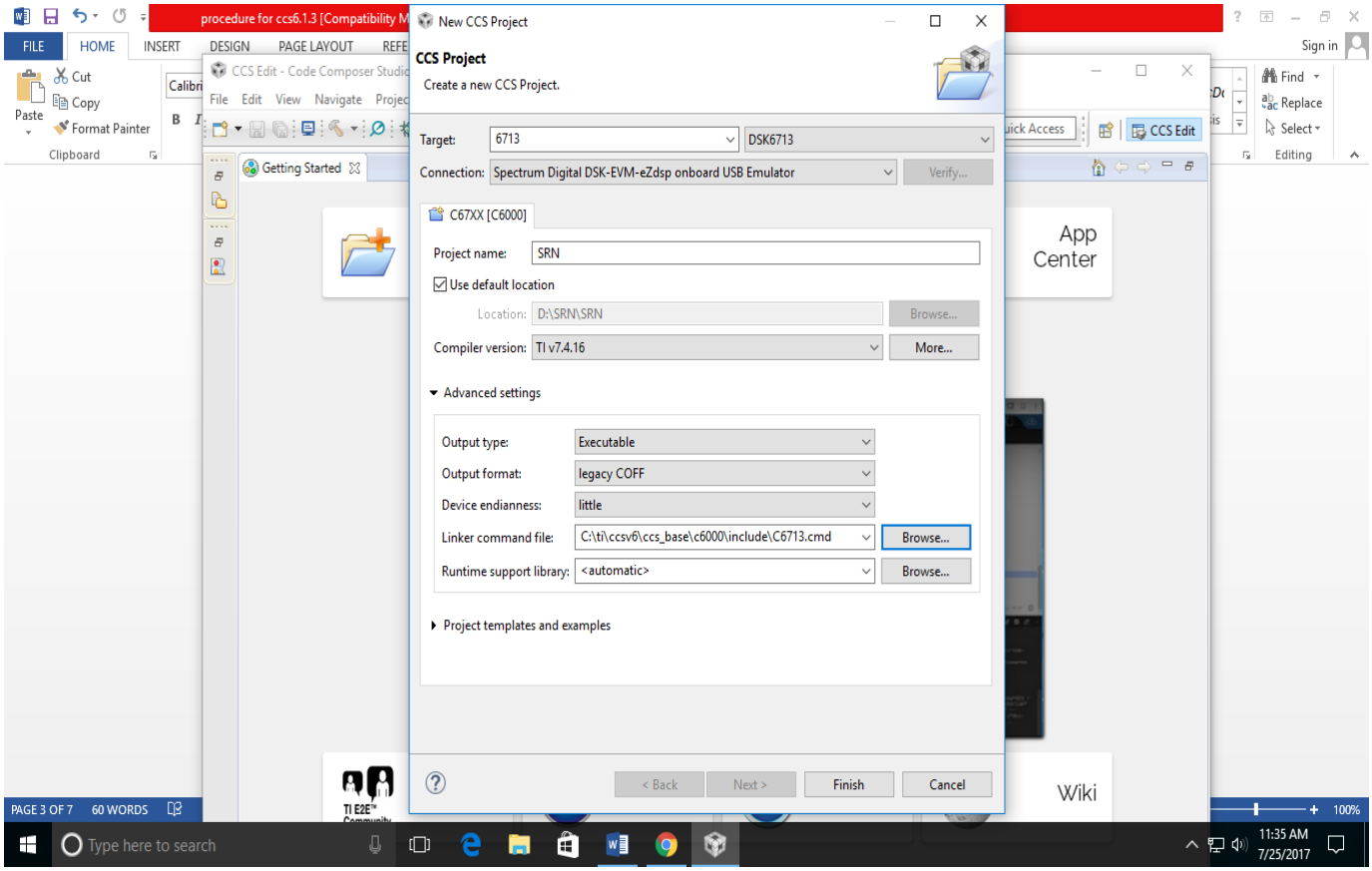
2. You will get workspace Launcher window in workspace select the drive other than 'C' and create a new folder by your SRN and click OK.





3. Click on project and select New CCS Project in that select the desired properties as given in the picture and click on Finish button.

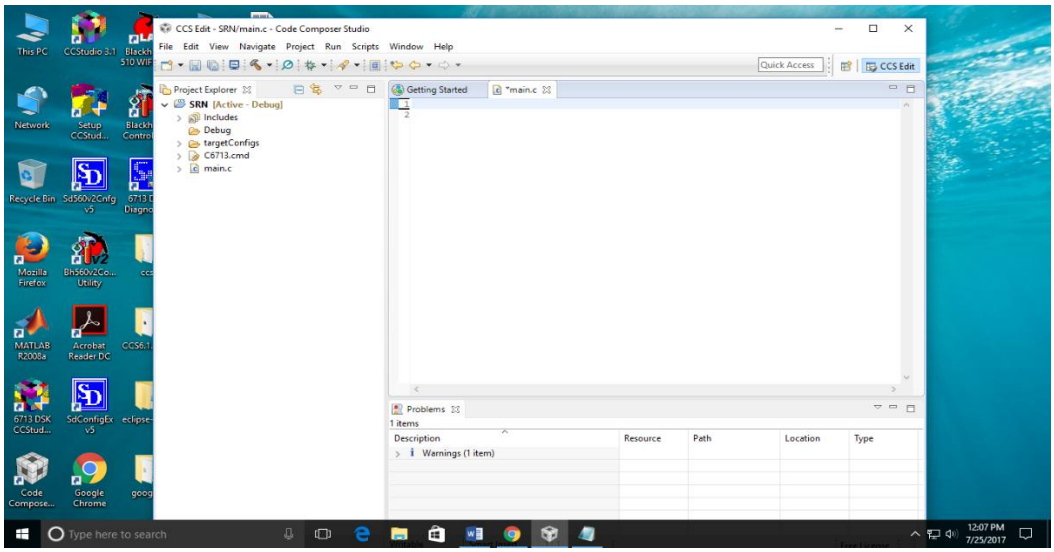




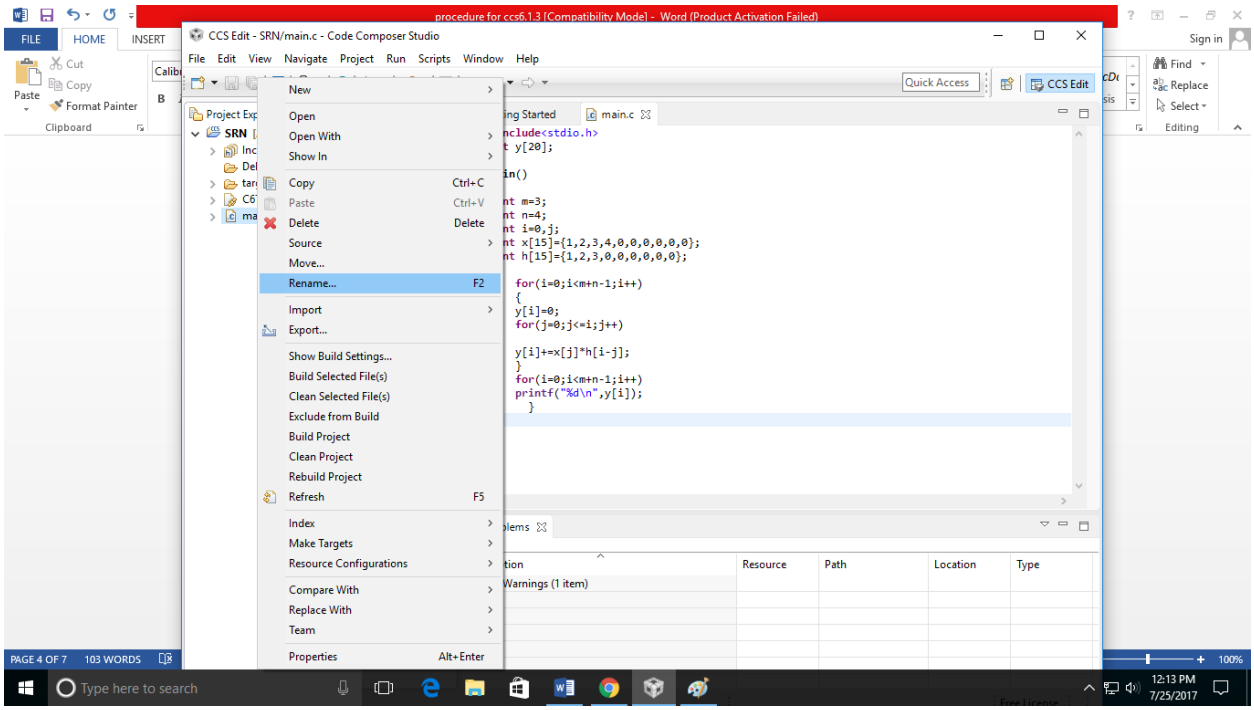
NOTE:

For real time programs select **hello.cmd** file and for non-real programs select **C6713.cmd**

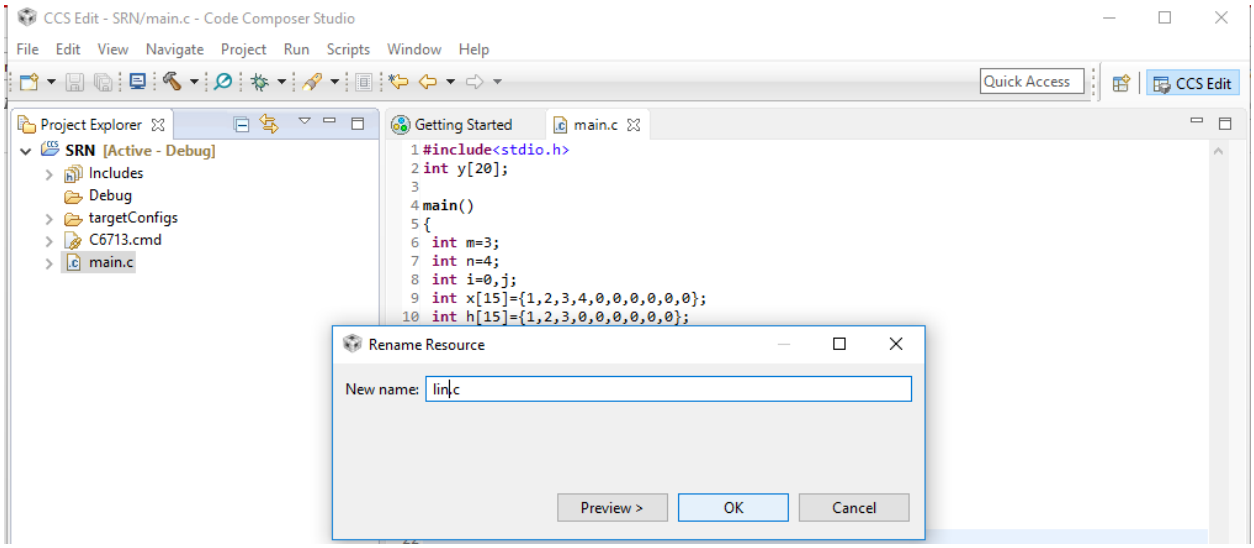
4. Once you click on finish button you will get a CCS editor window, in that type the C program and at the end click on SAVE button.

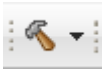


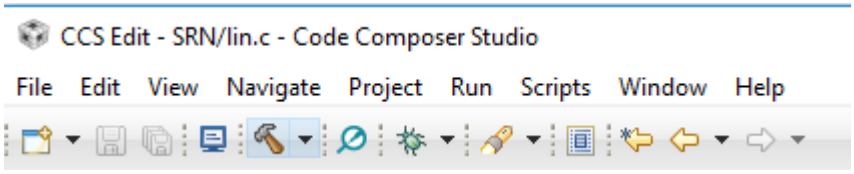
5. To rename the file name instead of main.c Right click on main.c which is in project explorer window and select Rename icon or press F2 button.



6. You will get a new Rename Resource window there you give a new file name with extension '.C' and click 'OK'.

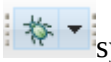


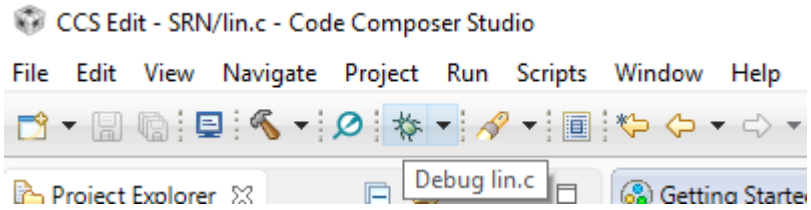
7. Now to build the project click on HAMMER  symbol which is on the top left of editor window.



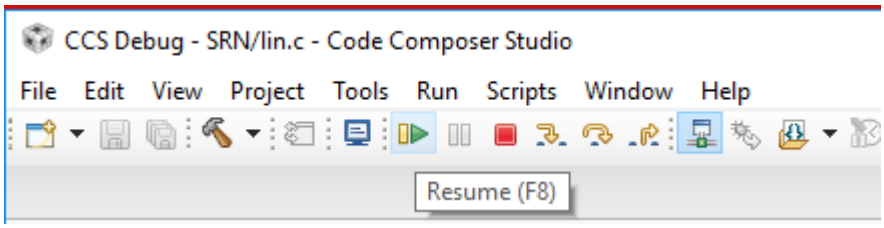
Note:

If any error occurs in the program that will be displayed in the console window which is at the bottom of editor window. Clear that error again click on save button and repeat the same above step.

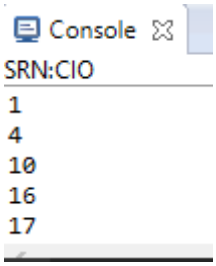
8. After building the project to debug the program click on BUG  symbol which is on the top left of editor window.



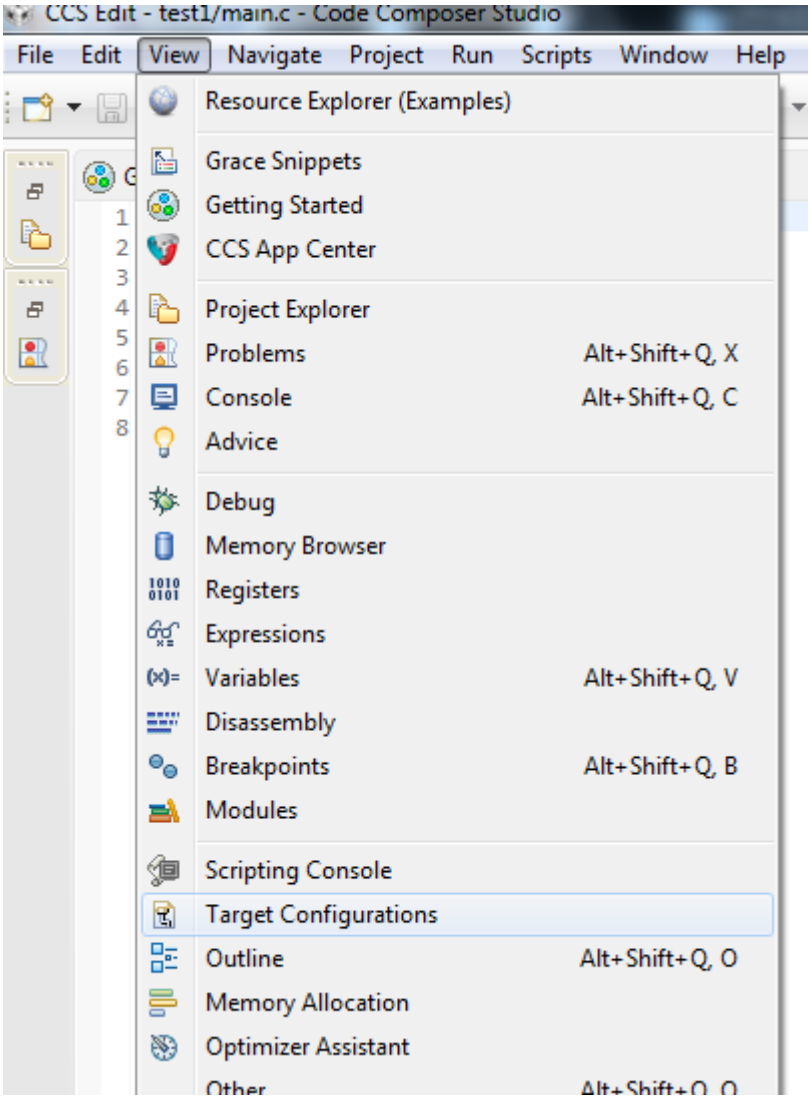
9. Now to RUN the program click on Resume  or F8 button which is on the top left of editor window.

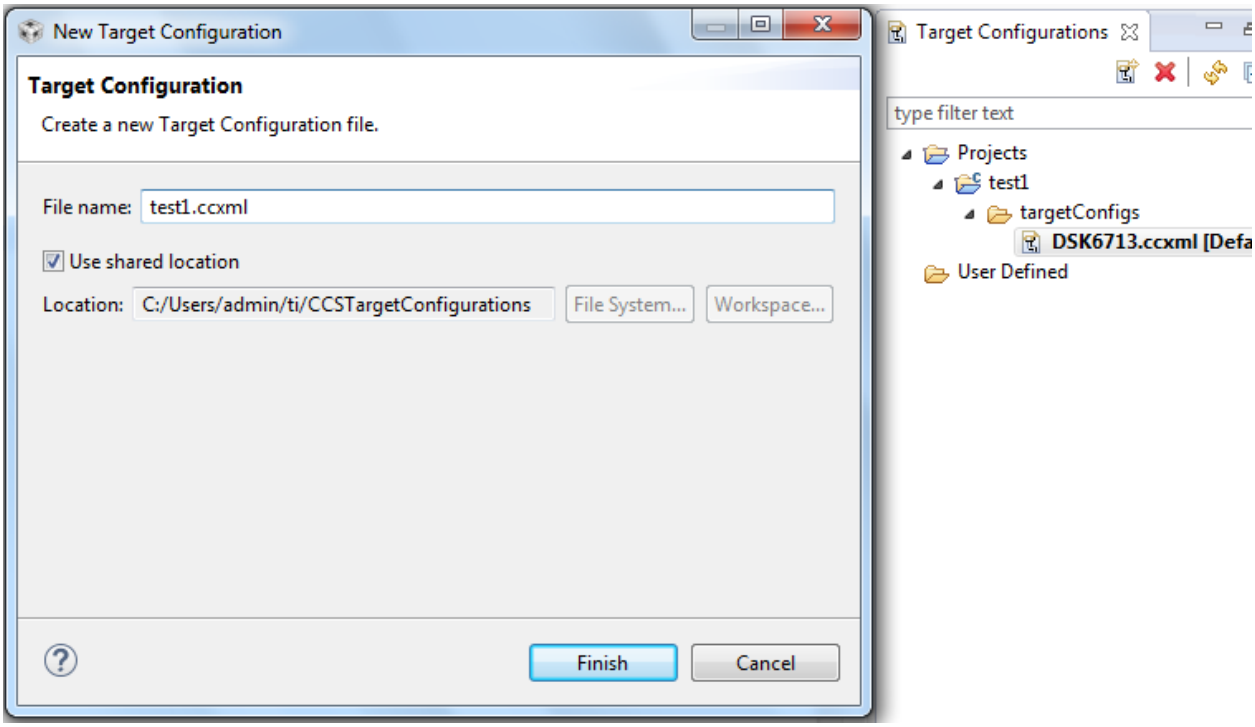


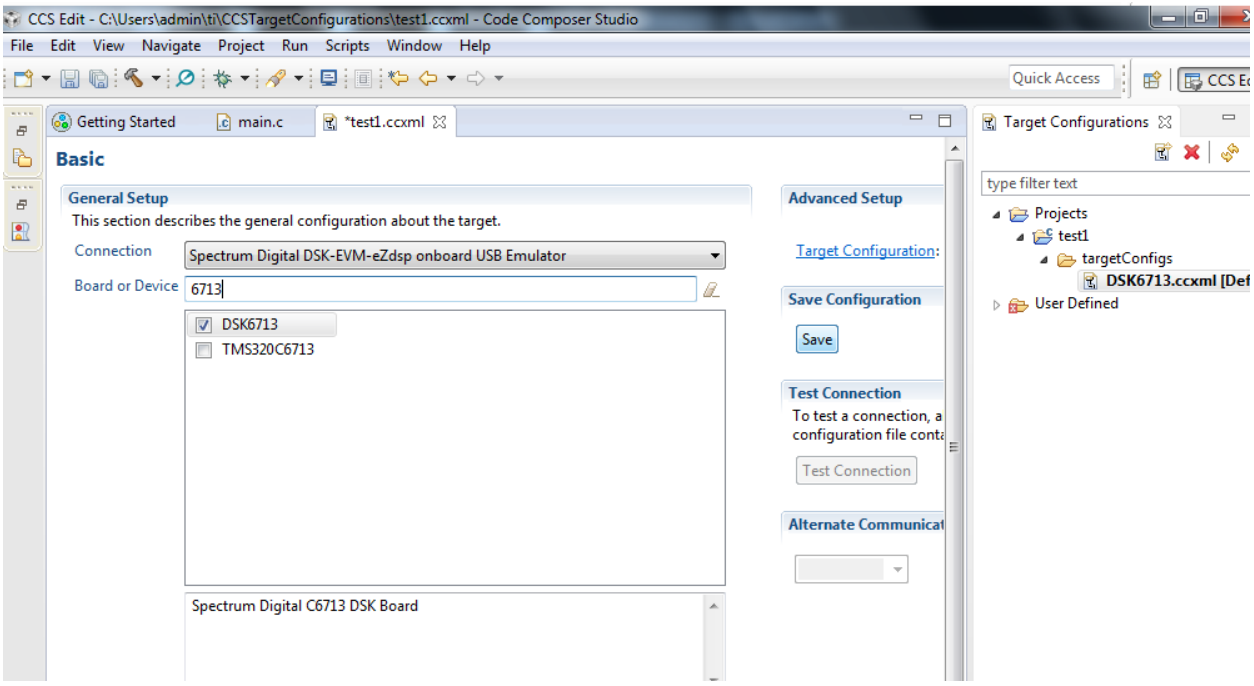
10. In the console window you can observe the output values.



For the very first time after installation of CCS software the following properties has to be modified.




New Target Configuration
Target Configuration
Create a new Target Configuration file.
File name: test1.ccxml
Use shared location
Location: C:/Users/admin/ti/CCSTargetConfigurations
File System...
Workspace...
Finish
Cancel
Target Configurations
type filter text
Projects
test1
targetConfigs
DSK6713.ccxml [Defa
User Defined


CCS Edit - C:\Users\admin\ti\CCSTargetConfigurations\test1.ccxml - Code Composer Studio
File Edit View Navigate Project Run Scripts Window Help
Quick Access
Getting Started
main.c
*test1.ccxml
Basic
General Setup
This section describes the general configuration about the target.
Connection
Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
Board or Device
6713
DSK6713
TMS320C6713
Spectrum Digital C6713 DSK Board
Advanced Setup
Target Configuration:
Save Configuration
Save
Test Connection
To test a connection, a
configuration file conta
Test Connection
Alternate Communicat
Target Configurations
type filter text
Projects
test1
targetConfigs
DSK6713.ccxml [Def
User Defined

EXPERIMENT-01

AIM:- TO COMPUTE THE LINEAR CONVOLUTION OF THE GIVEN INPUT SEQUENCE x(n) & THE IMPULSE RESPONSE OF THE SYSTEM h(n) USING "C-PROGRAMMING".

Linear Convolution involves the following operations.

1. Folding 2. Multiplication 3. Addition 4. Shifting

These operations can be represented by a Mathematical Expression as follows:

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

x[ ]= Input signal Samples

h[ ]= Impulse response co-efficient.

y[ ]= Convolution output.

n = No. of Input samples

h = No. of Impulse response co-efficient.

Algorithm to implement 'C' or Assembly program for Convolution:

Eg: x[n] = {1, 2, 3, 4}

h[k] = {1, 2, 3, 4}

Where: n=4, k=4. : Values of n & k should be a multiple of 4.

If n & k are not multiples of 4, pad with zero's to make multiples of 4

r= n+k-1 : Size of output sequence.

= 4+4-1

= 7.

r=	**0**	**1**	**2**	**3**	**4**	**5**	**6**
n= 0	x[0]h[0]	x[0]h[1]	x[0]h[2]	x[0]h[3]			
1		x[1]h[0]	x[1]h[1]	x[1]h[2]	x[1]h[3]		
2			x[2]h[0]	x[2]h[1]	x[2]h[2]	x[2]h[3]	
3				x[3]h[0]	x[3]h[1]	x[3]h[2]	x[3]h[3]

Output: y[r] = { 1, 4, 10, 20, 25, 24, 16}.

NOTE: At the end of input sequences pad 'n' and 'k' no. of zero's

/* program to implement linear convolution */

```
#include<stdio.h>
int y[20];
main()
{
 int m=3;
 int n=4;
 int i=0,j;
 int x[15]={1,2,3,4,0,0,0,0,0,0};
 int h[15]={1,2,3,0,0,0,0,0,0};
for(i=0;i<m+n-1;i++)
{
y[i]=0;
for(j=0;j<=i;j++)
y[i]+=x[j]*h[i-j];
}
for(i=0;i<m+n-1;i++)
printf("%d\n",y[i]);
}
```

Calculations:

x[n] = [1 2 3 4]—> L1=4 h[n] = [1 2 3]—> L2=3

L=L1+L2-1 => 4+3-1=>6 L=6

$$y[n]=x[n].h[n]$$

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

Now,

$$y[n]=\sum_{K=0}^{5} x[n].h[n]$$

$$n=0,\quad y[0] = \sum_{K=0}^{5} x[k].h[0-k]$$

=>x[0]h[0]+x[1]h[-1]+x[2]h[-2]+x[3]h[-3]+x[4]h[-4]+x[5]h[-5]

y[0] = 1

$$n=1,\quad y[1] = \sum_{K=0}^{5} x[0].h[1-k]$$

=> x[0]h[1]+x[1]h[0]+x[2]h[-1]+x[3]h[-2]+x[4]h[-3]+x[5]h[-4]

=>(1)(2)+(2)(1) =>2+2

y[1] = 4

$$n=2,\quad y[2] = \sum_{K=0}^{5} x[0].h[2-k]$$

=> x[0]h[2]+x[1]h[1]+x[2]h[0]+x[3]h[-1]+x[4]h[-2]+x[5]h[-3]

=> (1)(3)+(2)(2)+(3)(1) => 3+4+3

y[2]= 10

$$n=3,\quad y[3] = \sum_{K=0}^{5} x[0].h[3-k]$$

=> x[0]h[3]+x[1]h[2]+x[2]h[1]+x[3]h[0]+x[4]h[-1]+x[5]h[-2]

=>(2)(3)+(3)(2)+(4)(1) => 6+6+4

y[3]= 16

$$n=4,\quad y[4] = \sum_{K=0}^{5} x[0].h[4-k]$$

=> x[0]h[4]+x[1]h[3]+x[2]h[2]+x[3]h[1]+x[4]h[0]+x[5]h[-1]

=>(3)(3)+(4)(2) =>9+8 => 17

y[4] = 17

$$n=5,\quad y[5] = \sum_{K=0}^{5} x[0].h[5-k]$$

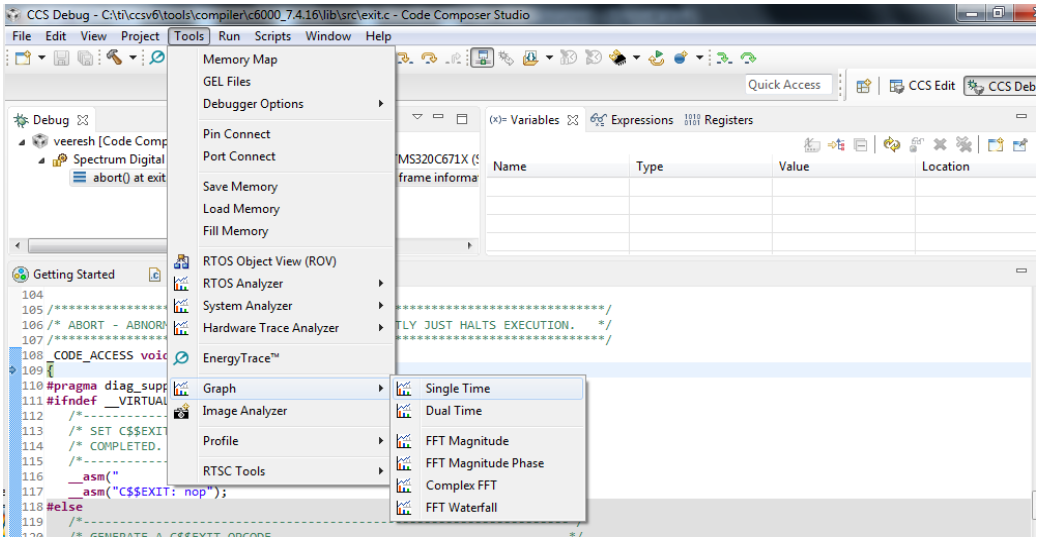
$$\Rightarrow x[0]h[5]+x[1]h[4]+x[2]h[3]+x[3]h[2]+x[4]h[1]+x[5]h[0]$$

$$\Rightarrow (4)(3) \Rightarrow 12$$

y[5]= 12

y[n] = [1 4 10 16 17 12]

LINEAR CONVOLUTION GRAPH PROPERTIES:



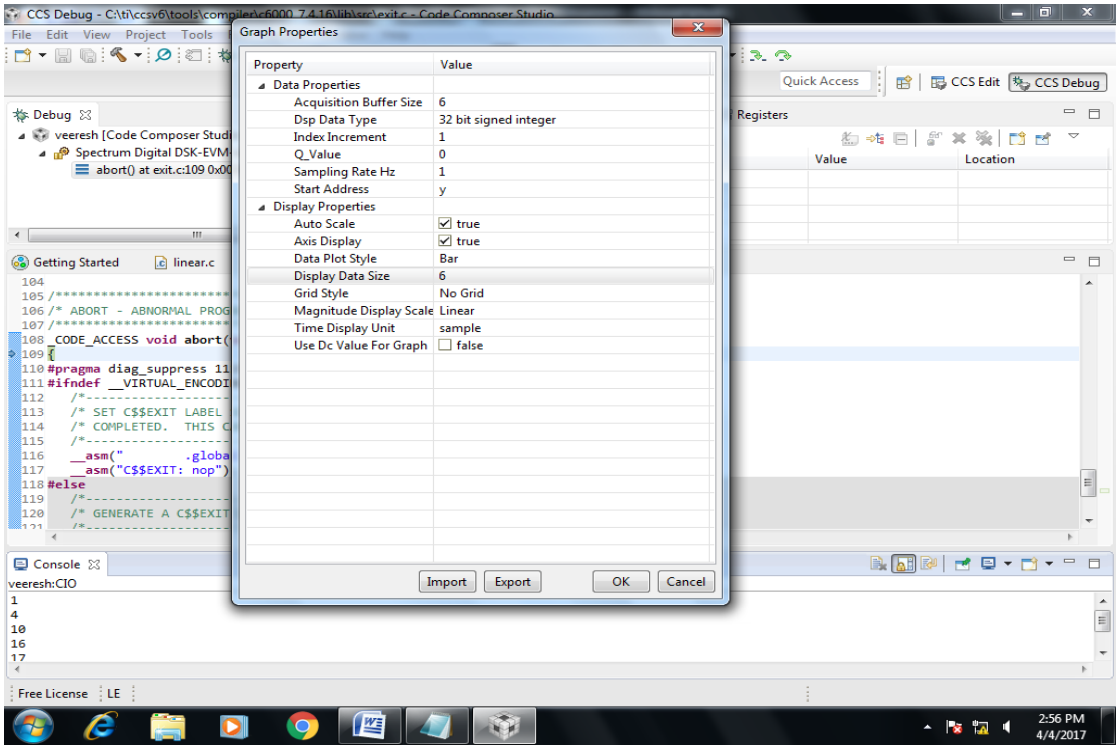
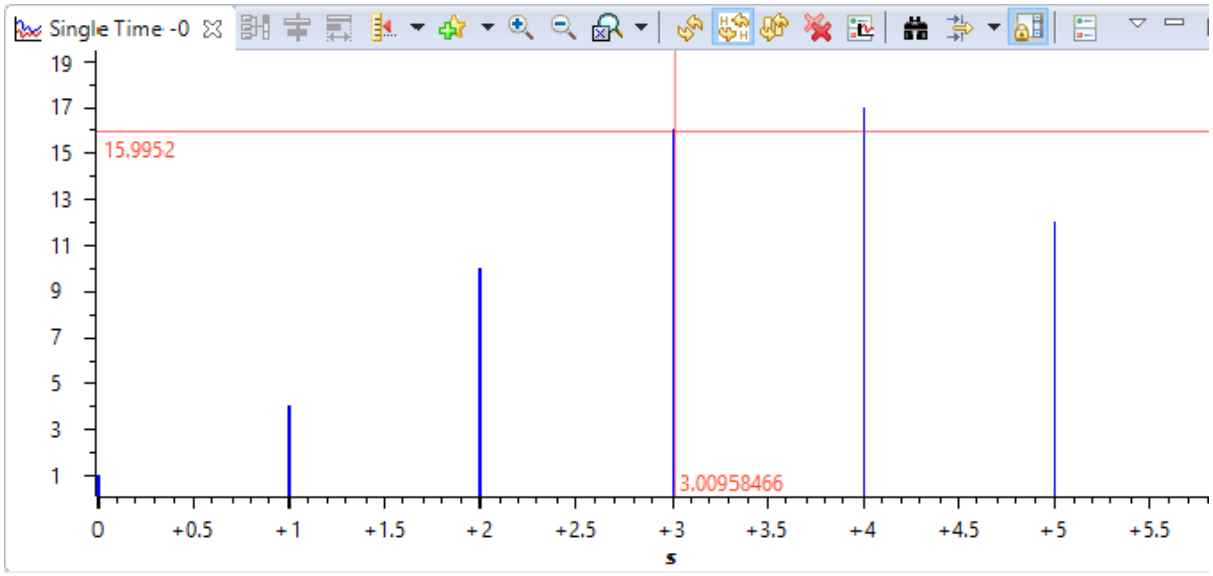


Figure:



Results and Inference:

y(n) = {1, 4, 10, 16, 17, 12}

Linear Convolution for Two Sequences is Calculated and Verified using Code Composer Studio and DSP Hardware Kit.

EXPERIMENT-02

AIM: - TO COMPUTE THE CIRCULAR CONVOLUTION OF THE GIVEN TWO SEQUENCES USING "C-PROGRAMMING".

Steps for Cyclic Convolution

Steps for cyclic convolution are the same as the usual convolution, except all index calculations are done "mod N" = "on the wheel"

Steps for Cyclic Convolution

Step1: "Plot f[m] and h[−m]

Subfigure 1.1 Subfigure 1.2

Step 2: "Spin" h[−m] *n* times Anti Clock Wise (counter-clockwise) to get h[n-m] (i.e. Simply rotate the sequence, *h*[*n*], clockwise by *n* steps)

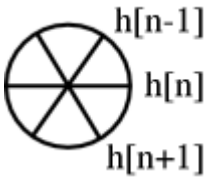
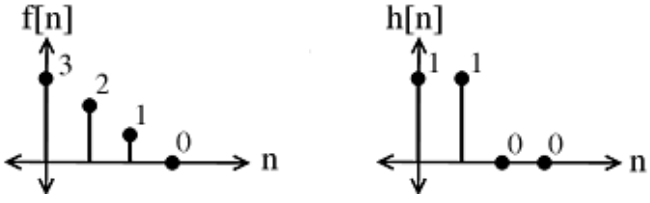


Figure 2: Step 2

Step 3: Pointwise multiply the *f*[*m*] wheel and the *h*[n−m] wheel. sum=*y*[*n*]

Step 4: Repeat for all $0 \le n \le N-1$

Example 1: Convolve (n = 4)



Subfigure 3.1 Subfigure 3.2

Figure 3: Two discrete-time signals to be convolved.

- h[−*m*] =

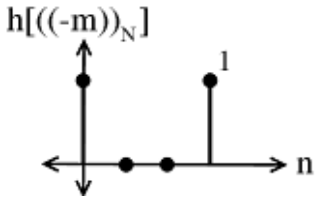


Figure 4

Multiply *f*[*m*] and sum to yield: *y*[0] =3

- *h*[1−*m*]

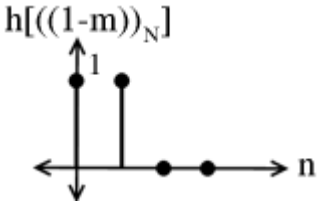


Figure 5

Multiply *f*[*m*] and sum to yield: *y*[1] =5

- *h*[2−*m*]

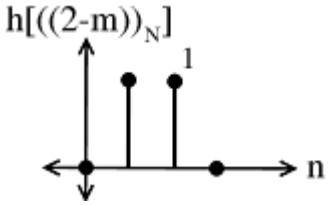


Figure 6

Multiply *f*[*m*] and sum to yield: *y*[2] =3

- *h*[3−*m*]

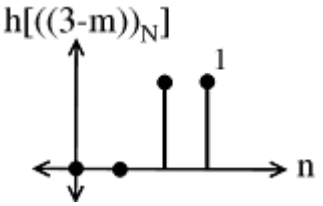


Figure 7

Multiply *f*[*m*] and sum to yield: *y*[3] =1

/* program to implement circular convolution */

```
#include<stdio.h>
int m,n,x[30],h[30],y[30],i,j,k;
void main()
{
  printf(" Enter the length of the first sequence\n");
  scanf("%d",&m);

  printf("  Enter the length of the second sequence\n");
  scanf("%d",&n);

  printf("  Enter the first sequence\n");

  for(i=0;i<m;i++)
  scanf("%d",&x[i]);

  printf("  Enter the second sequence\n");
  for(j=0;j<n;j++)
  scanf("%d",&h[j]);

  if(m-n!=0)
  {
        if(m>n)
        {
        for(i=n;i<m;i++)
        h[i]=0;
        n=m;
        }

        for(i=m;i<n;i++)
        x[i]=0;
        m=n;
  }
printf("the circular convolution is\n");
```

```
for(i=0;i<n;i++)
{
y[i]=0;
for(j=0;j<n;j++)
{
k=i-j;
if(k<0)
k=k+n;
y[i]=y[i]+x[j]*h[k];
}
printf("%d \t",y[i]);
}
}
```

Calculations:-

$$y[n] = \sum_{K=-\infty}^{\infty} x(k)\, h(n-k)_N \quad \Rightarrow \quad y[n] = \sum_{K=-\infty}^{\infty} x(k)h(n-k+N)$$

$x[n] = [1\ 2\ 3\ 4]$

$h[n] = [1\ 2\ 3\ 4]$

$y(n)= x(0)h(n)_N + x(1)h(n-1)_N + x(2)h(n-2)_N + x(3)h(n-3)_N$

$n=0, y[0] = x(0)h(0)_4 + x(1)h(-1)_4 + x(2)h(-2)_4 + x(3)h(-3)_4$

$= (1)(1) + (2)(4) + (3)(3) + (4)(2)$

y[0] = 26

$n=1, y[1] = x(0)h(1-0)_4 + x(1)h(1-1)_4 + x(2)h(1-2)_4 + x(3)h(1-3)_4$

$= x(0)h(1)_4 + x(1)h(0)_4 + x(2)h(-1)_4 + x(3)h(-2)_4$

$= (1)(2) + (2)(1) + (3)(4) + (4)(3)$

y[1] = 28

n=2, $y[2] = x(0)h(2-0)_4+x(1)h(2-1)_4+x(2)h(2-2)_4 +x(3)h(2-3)_4$

$=x(0)h(2)_4 +x(1)h(1)_4 +x(2)h(0)_4 +x(3)h(-1)_4$

$= (1)(3) +(2)(2)+(3)(1)+(4)(4)$

y[2] = 26

n=3, $y[3] = x(0)h(3-0)_4 +x(1)h(3-1)_4 +x(2)h(3-2)_4 +x(3)h(3-3)_4$

$=x(0)h(3)_4+x(1)h(2)_4 +x(2)h(1)_4 +x(3)h(0)_4$

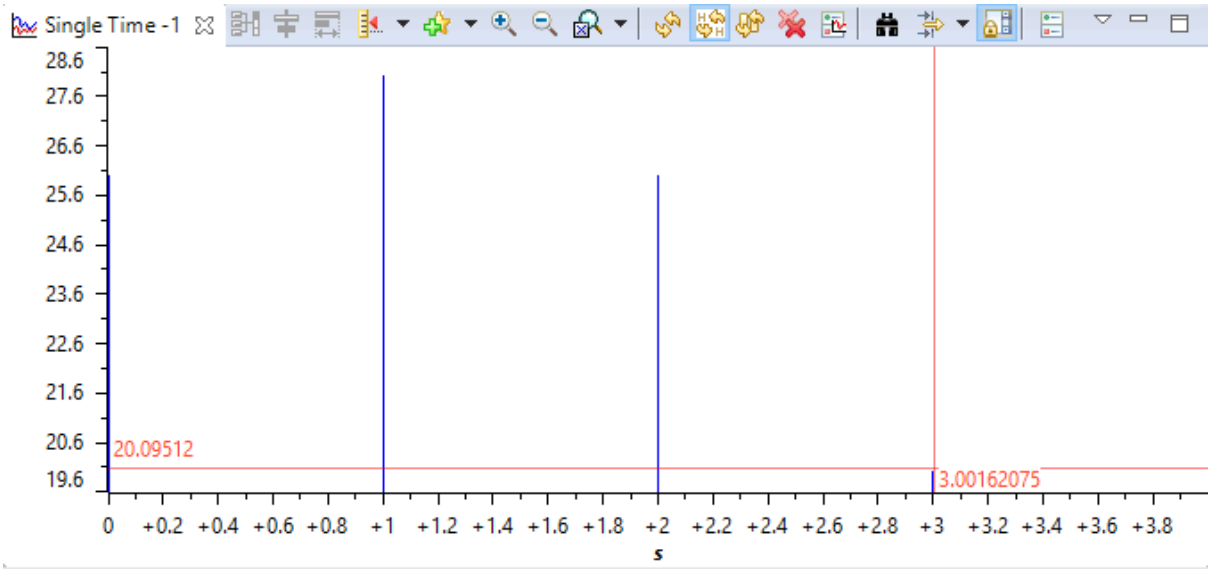
$= (1)(4) + (2)(3) +(3)(2) +(4)(1)$

y[3] = 20

Note:-

Follow same procedure of page no.80 and 81 to view graph.

Figure:



Results and Inference:

```
Enter the length of the first sequence
 4
Enter the length of the second sequence
 4
Enter the first sequence
 1 2 3 4
Enter the second sequence
 1 2 3 4
The circular convolution is
 26   28    26    20
```

y[n] = [26 28 26 20]

Circular Convolution for Two Sequences is Calculated and Verified using Code Composer Studio and DSP Hardware Kit.

EXPERIMENT-03

AIM: - TO COMPUTE THE N (=4/8/16) POINT DFT OF THE GIVEN SEQUENCE USING "C- PROGRAMMING".

Theory:

The N point DFT of discrete time signal x[n] is given by the equation:

$$X(k)=\sum_{n=0}^{N-1} x[n]e^{\frac{-j2\pi kn}{N}} \quad ; \quad k=0,1,2,\ldots N-1$$

Where N is chosen such that $N \geq L$, where L=length of x[n]. To implement using C program we use the expression:

$$e^{\frac{-j2\pi kn}{N}}=\cos\left(\frac{2\pi kn}{N}\right)-j\sin\left(\frac{2\pi kn}{N}\right)$$

and allot memory space for real and imaginary parts of the DFT X(k).

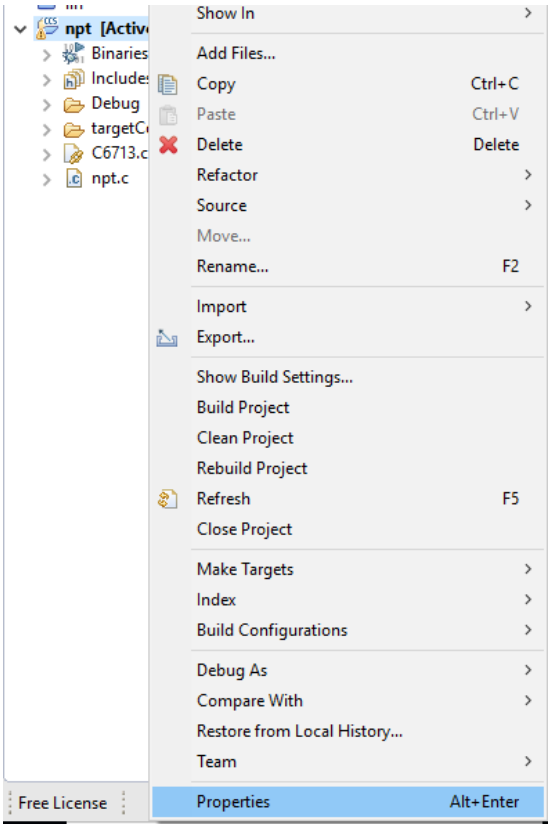
/* program to implement N point DFT */

```
#include<stdio.h>
#include<math.h>
short N=4;
short x[4]={2,3,4,5};
float pi=3.1416;
float sumre=0;
float sumim=0;
float cosine=0;
float sine=0;
float out_sumre[8]={0.0};
float out_sumim[8]={0.0};
void main()
{
int n,k=0;
for(k=0;k<N;k++)
{
sumre=0;
sumim=0;
```

```
 for(n=0;n<N;n++)
{
cosine=cos(2*pi*k*n/N);
sine=sin(2*pi*k*n/N);
sumre=sumre+x[n]*cosine;
sumim=sumim-x[n]*sine;
}
out_sumre[k]=sumre;
out_sumim[k]=sumim;
printf("[%d] %7.3f %7.3f\n\n",k,out_sumre[k],out_sumim[k]);
}
}
```

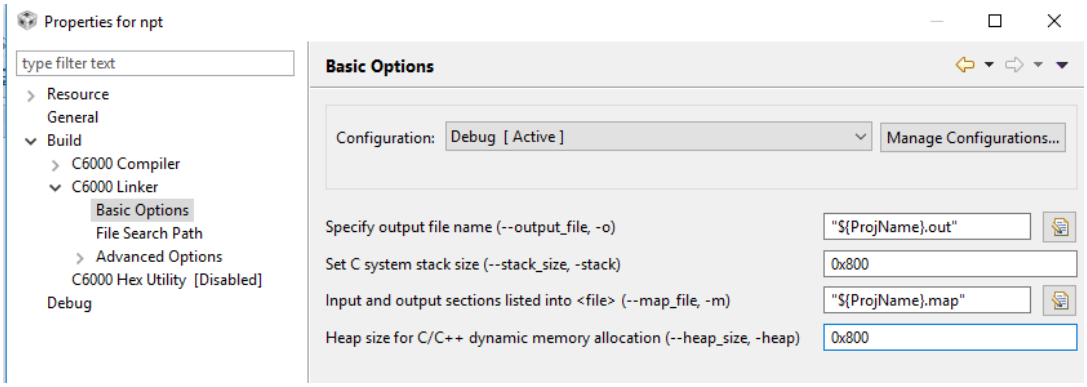
NOTE: Since the output is in decimal / complex value the following settings has to be done in the properties.

STEP 01: Right click on project name and select properties



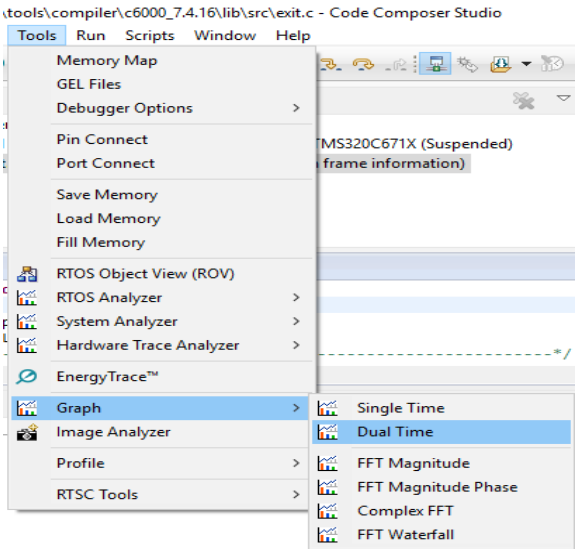
Step 02: Now select C6000 Linker and in that select basic options. Now enter 0x800 in Set C system stack size(--stack, size, -stack) : **0x800**

Heap size for C/C++ dynamic memory allocation(--heap_size, -heap) : **0x800**

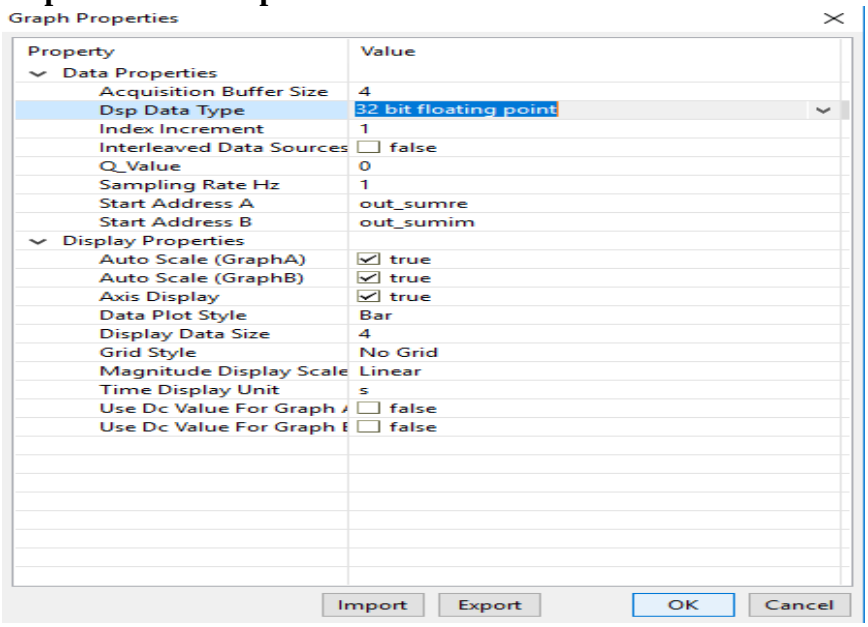


N-POINT DFT GRAPH PROPERTIES:

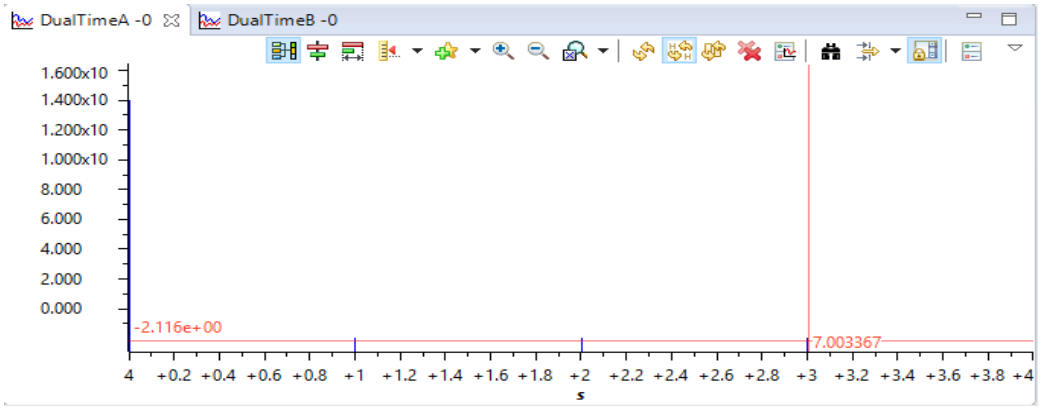
Step 01: Tools→Graph→Dual Time



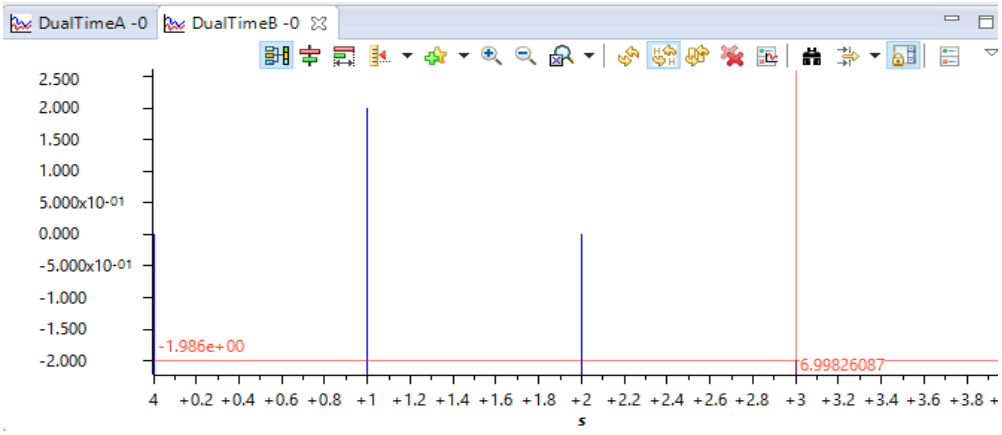
Step 02: Select the parameters



Figure(1):Sum of real part (out_sumre)



Figure(2):Sum of Imaginary part (out_sumim)



Results and Inference:-

[0] 14.000, 0.000, [1] -2.000, 2.000, [2] -2.000, 0.000, [3] -2.000, -2.000

N-Point DFT of a Sequence is Calculated and Verified using Code Composer Studio and DSP Hardware Kit.

EXPERIMENT-04

AIM: - SOLVING A LINEAR CONSTANT COEFFICIENT FIRST/SECOND ORDER DIFFERENCE EQUATION (zero initial conditions). TO FIND IMPULSE RESPONSE OF A SYSTEM USING TMS320C6713 DSP.

Program:

```
#include<stdio.h>

#define Order  2

#define Len     5

float h[Len] = {0.0,0.0,0.0,0.0,0.0}, sum;

void main()

{

int j, k;

float a[Order+1] = {1, 0.5, 0.0};

float b[Order+1] = {1, -0.63, 0.72};

for(j=0; j<Len; j++)

{

sum = 0.0;

for(k=1; k<=Order; k++)

{

if ((j-k) >= 0)

sum = sum+(b[k]*h[j-k]);

}

if (j <= Order)

h[j] = a[j]-sum;

else

h[j] = -sum;

printf (" %f ",h[j]);

}

}
```

Calculations:-

y[n]-0.63y[n-1]+0.72y[n-2] = x[n]+0.5x[n-1]

y[n] = 0.63y[n-1]-0.72y[n-2]+ x[n]+0.5x[n-1]

n=0; y[0] = 0.63y[0-1]-0.72y[0-2]+x[0]+0.5x[0-1]

= 0.63y[-1]-0.72y[-2]+x[0]+0.5x[-1]

y[0] = 1

n=1; y[1] = 0.63y[1-1]-0.72y[1-2]+x[1]+0.5x[1-1]

= 0.63y[0]-0.72y[-1]+x[1]+0.5x[0]

= (0.63)(1)+(0.5)(1)

y[1] = 1.13

n=2; y[2] = 0.63y[2-1]-0.72y[2-2]+x[2]+0.5x[2-1]

= 0.63y[1]-0.72y[0]+x[2]+0.5x[1]

= (0.63)(1.13)-(0.72)(1)

y[2] = -0.008100

n=3; y[3] = 0.63y[3-1]-0.72y[3-2]+x[3]+0.5x[3-1]

= 0.63y[2]-0.72y[1]+x[3]+0.5x[2]

= (0.63)(-0.008100)-(0.72)(1.13)

y[3] = -0.818703

n=4; y[4] = 0.63y[4-1]-0.72y[4-2]+x[4]+0.5x[4-1]

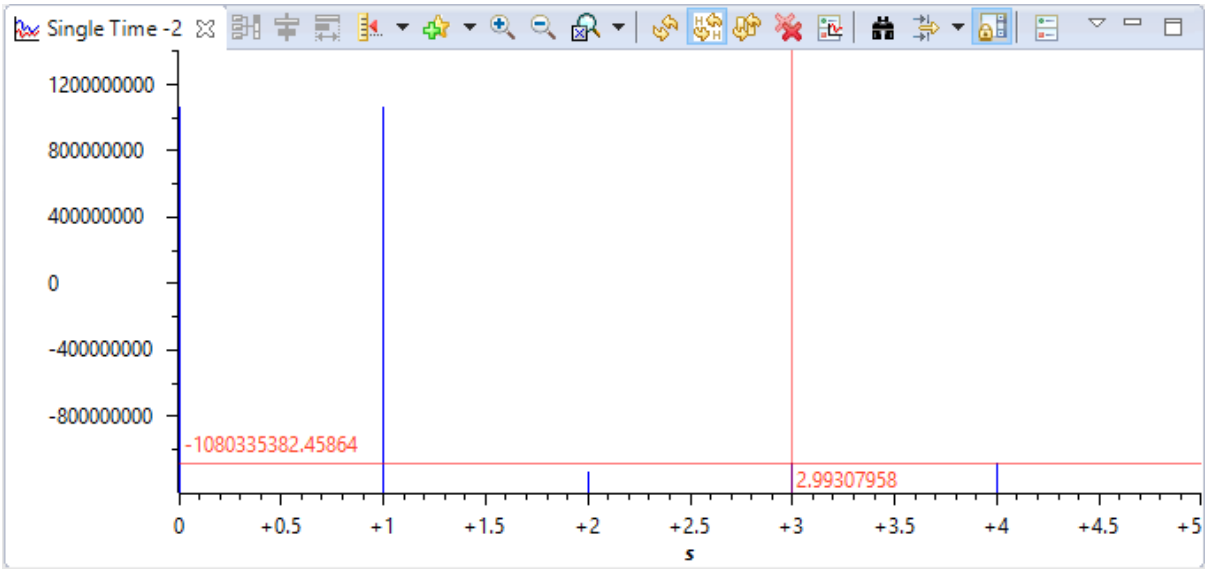
= 0.63y[3]-0.72y[2]+x[4]+0.5x[3]

= (0.63)(-0. 8187)-(0.72)(-0.0081)

y[4] = -0.509949

Note: Follow the same procedure from page no. 75 and 76.

Figure:



Results and Inference:

1, 1.13, -0.008100, -0.818703, -0.509949

Impulse Response for a given Difference Equation with and without initial conditions is Calculated and Verified using Code Composer Studio and DSP Hardware Kit.

IIR FILTER

AIM:- Design and Implementation of IIR digital filter for audio signals.

IIR Filter Theory:

IIR filter Designing Experiments

GENERAL CONSIDERATIONS:

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

IMPLEMENTATION OF DISCRETE-TIME SYSTEMS:

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

UNIT OBJECTIVE:

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

BACKGROUND CONCEPTS:

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

'C' PROGRAM TO IMPLEMENT IIR FILTER

```
#include "dsk6713.h"
#include "dsk6713_aic23.h"
const signed int filter_Coeff[] =
{
//12730,-12730,12730,2767,-18324,21137 /*HP 2500 */
//312,312,312,32767,-27943,24367 /*LP 800 */
//1455,1455,1455,32767,-23140,21735 /*LP 2500 */
//9268,-9268,9268,32767,-7395,18367 /*HP 4000*/
7215,-7215,7215,32767,5039,6171, /*HP 7000*/
} ;
/* Codec configuration settings */
DSK6713_AIC23_Config config = { \
0x0017, /* 0 DSK6713_AIC23_LEFTINVOL Left line input channel volume */
\
0x0017, /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume
*/\
0x00d8, /* 2 DSK6713_AIC23_LEFTHPVOL Left channel headphone volume */
\
0x00d8, /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */
\
0x0011, /* 4 DSK6713_AIC23_ANAPATH Analog audio path control */
\
0x0000, /* 5 DSK6713_AIC23_DIGPATH Digital audio path control */
\
0x0000, /* 6 DSK6713_AIC23_POWERDOWN Power down control */
\
0x0043, /* 7 DSK6713_AIC23_DIGIF Digital audio interface format */
\
0x0081, /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */
\
0x0001 /* 9 DSK6713_AIC23_DIGACT Digital interface activation */
\
};
/*
* main() - Main code routine, initializes BSL and generates tone
*/
void main()
{
DSK6713_AIC23_CodecHandle hCodec;
int l_input, r_input, l_output, r_output;
/* Initialize the board support library, must be called first */
DSK6713_init();
/* Start the codec */
                hCodec = DSK6713_AIC23_openCodec(0, &config);
DSK6713_AIC23_setFreq(hCodec, 3);
while(1)
{ /* Read a sample to the left channel */
while (!DSK6713_AIC23_read(hCodec, &l_input));
```

```
/* Read a sample to the right channel */
while (!DSK6713_AIC23_read(hCodec, &r_input));
l_output=IIR_FILTER(&filter_Coeff ,l_input);
r_output=l_output;
/* Send a sample to the left channel */
while (!DSK6713_AIC23_write(hCodec, l_output));
/* Send a sample to the right channel */
while (!DSK6713_AIC23_write(hCodec, r_output));
}
/* Close the codec */
DSK6713_AIC23_closeCodec(hCodec);
}
signed int IIR_FILTER(const signed int * h, signed int x1)
{
static signed int x[6] = { 0, 0, 0, 0, 0, 0 }; /* x(n), x(n-1), x(n-
2). Must be static */
static signed int y[6] = { 0, 0, 0, 0, 0, 0 }; /* y(n), y(n-1), y(n-
2). Must be static */
int temp=0;
temp = (short int)x1; /* Copy input to temp */
x[0] = (signed int) temp; /* Copy input to x[stages][0] */
temp = ( (int)h[0] * x[0]) ; /* B0 * x(n) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[1] * x[1]); /* B1/2 * x(n-1) */
temp += ( (int)h[2] * x[2]); /* B2 * x(n-2) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[4] * y[1]); /* A1/2 * y(n-1) */
temp -= ( (int)h[5] * y[2]); /* A2 * y(n-2) */
/* Divide temp by coefficients[A0] */
temp >>= 15;
if ( temp > 32767 )
{
temp = 32767;
}
else if ( temp < -32767)
{
                                        temp = -32767;

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

 }
```

PROCEDURE:

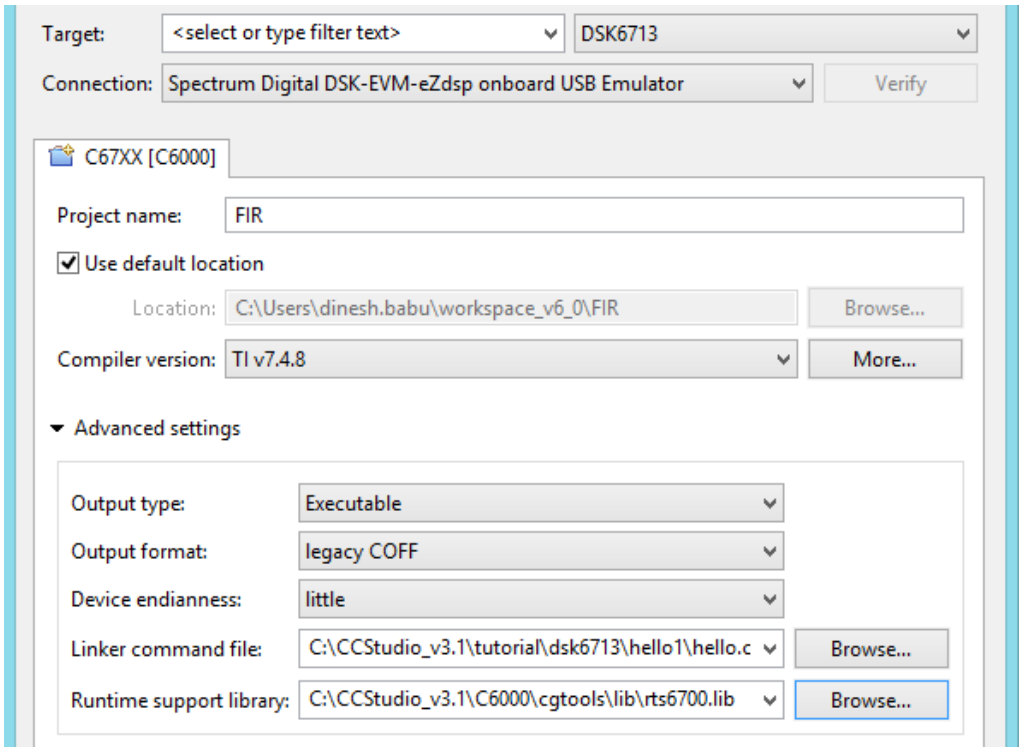
Create a Project (FIR filter)
Go to project in tool bar select new CCS project a project window will appear as shown below
Target : ---------- | DSK6713

Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator

Project name: any name (for example I have given as linear)

Linker Command File: C:\CCStudio_V3.1\tutorial\dsk6713\hello1**hello.cmd**

Run Time Support Library: C:\CCStudio_v3.1\C6000\cgtools\lib**rts6700.lib**



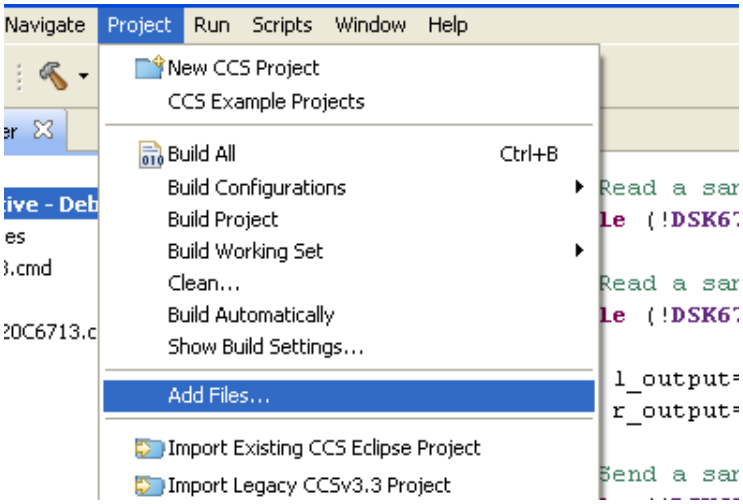
Then click on **finish .**

Then write the code in editor window and save it as iir.c

Now add supportive files:

Add BSL and CSL lib files as shown below

Go to **project→add files** as shown in below:



Then you have to add **bsl** and **csl** library files. These files you will get in below mentioned paths.

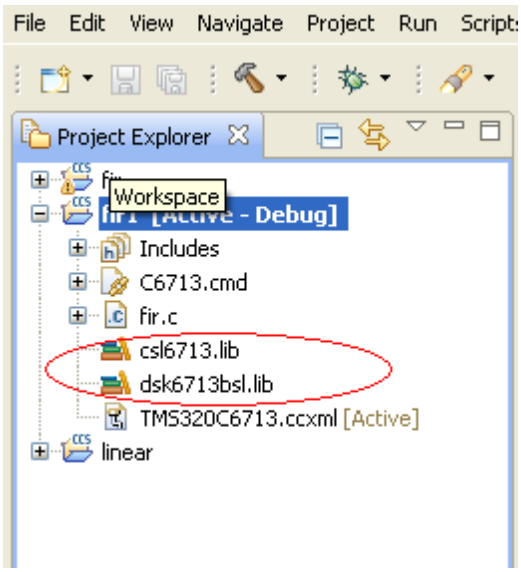
- BSL(Board support library file)

  Path: ***C:\CCStudio_v3.1\C6000\dsk6713\lib\dsk6713bsl.lib***

- CSL(Chip support library file)

  Path: ***C:\CCStudio_v3.1\C6000\csl\lib\csl6713.lib***

You can see these files added to your project as shown below:



FIR FILTER

AIM:- Design and Implementation of FIR digital filter for audio signals.

FIR Filter Theory:

Finite Impulse Response Filter
DESIGNING AN FIR FILTER:

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

1. Clearly specify the filter specifications.
   Eg: Order = 30;
   Sampling Rate = 8000 samples/sec
   Cut off Freq. = 400 Hz.

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

3. Compute the desired Impulse Response h d (n) using particular Window
   Eg: b_rect1=fir1(order, Wc , 'high',boxcar(31));

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

C PROGRAM TO IMPLEMENT FIR FILTER:
fir.c

```
#include "dsk6713.h"
#include "dsk6713_aic23.h"

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

static short in_buffer[100];

DSK6713_AIC23_Config config = {\
0x0017, /* 0 DSK6713_AIC23_LEFTINVOL Leftline input channel volume */\
0x0017, /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume*/\
0x00d8, /* 2 DSK6713_AIC23_LEFTHPVOL Left channel headphone volume */\
0x00d8, /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */\
0x0011, /* 4 DSK6713_AIC23_ANAPATH Analog audio path control */\
0x0000, /* 5 DSK6713_AIC23_DIGPATH Digital audio path control */\
0x0000, /* 6 DSK6713_AIC23_POWERDOWN Power down control */\
0x0043, /* 7 DSK6713_AIC23_DIGIF Digital audio interface format */\
```

```
0x0081, /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */\
0x0001 /* 9 DSK6713_AIC23_DIGACT Digital interface activation */ \
};
/*
* main() - Main code routine, initializes BSL and generates tone
*/
void main()
{

DSK6713_AIC23_CodecHandle hCodec;
Uint32 l_input, r_input,l_output, r_output;
/* Initialize the board support library, must be called first */
DSK6713_init();

/* Start the codec */

hCodec = DSK6713_AIC23_openCodec(0, &config);
DSK6713_AIC23_setFreq(hCodec, 1);
while(1)

{ /* Read a sample to the left channel */

while (!DSK6713_AIC23_read(hCodec, &l_input));

/* Read a sample to the right channel */

while (!DSK6713_AIC23_read(hCodec, &r_input));
l_output=(Int16)FIR_FILTER(&filter_Coeff ,l_input);
r_output=l_output;

/* Send a sample to the left channel */

while (!DSK6713_AIC23_write(hCodec, l_output));

/* Send a sample to the right channel */

while (!DSK6713_AIC23_write(hCodec, r_output));
}

/* Close the codec */

DSK6713_AIC23_closeCodec(hCodec);
}
signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
in_buffer[0] = x; /* new input at buffer[0] */
for(i=29;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer */
```

```
for(i=0;i<31;i++)
output = output + h[i] * in_buffer[i];
return(output);
}
```

PROCEDURE:

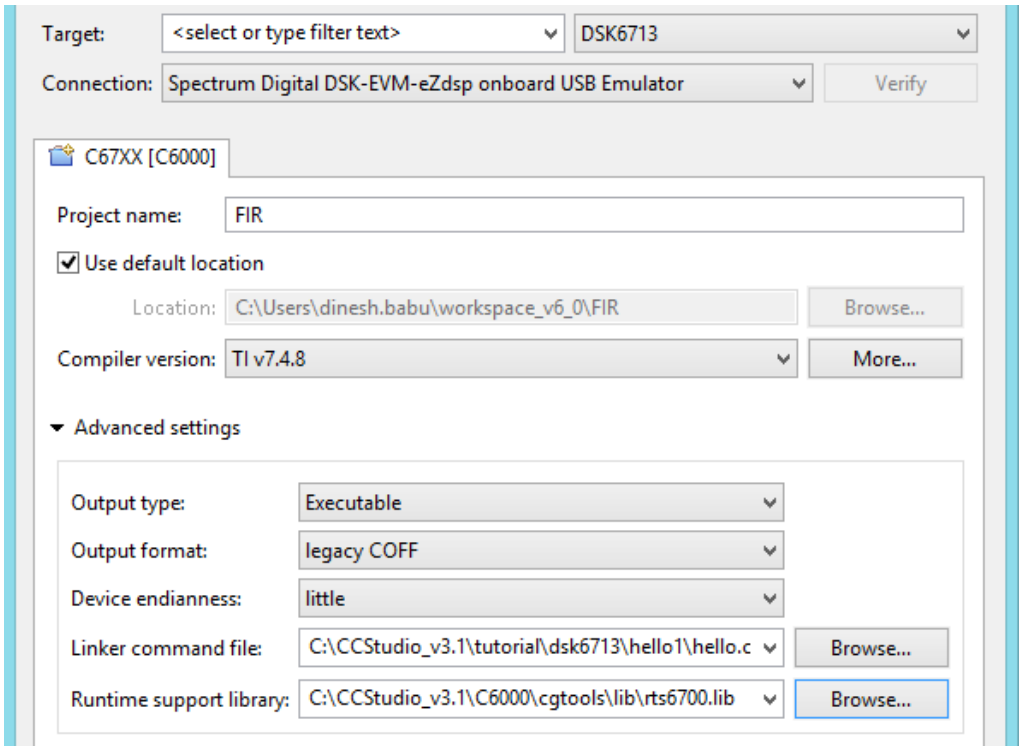
Create a Project (FIR filter)
Go to project in tool bar select new CCS project a project window will appear as shown below
Target : ---------- | DSK6713

Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator

Project name: any name (for example I have given as linear)

Linker Command File: C:\CCStudio_V3.1\tutorial\dsk6713\hello1**hello.cmd**

Run Time Support Library: C:\CCStudio_v3.1\C6000\cgtools\lib**rts6700.lib**



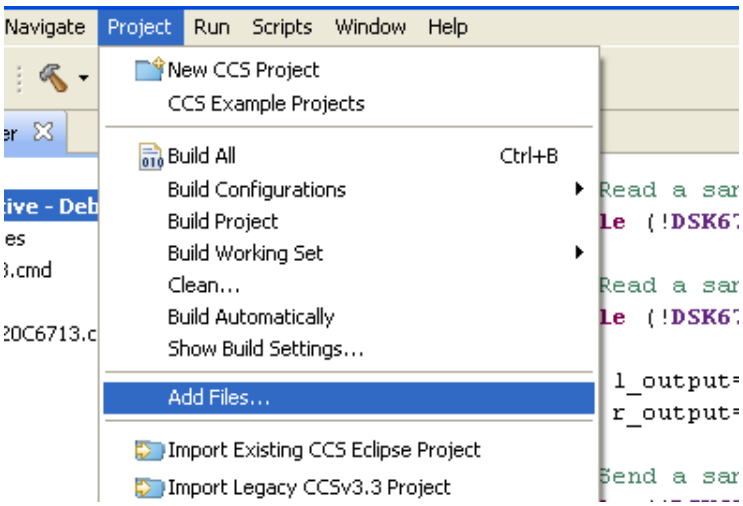
Then click on **finish .**

Then write the code in editor window and save it as fir.c

Now add supportive files:

Add BSL and CSL lib files as shown below

Go to **project→add files** as shown in below:

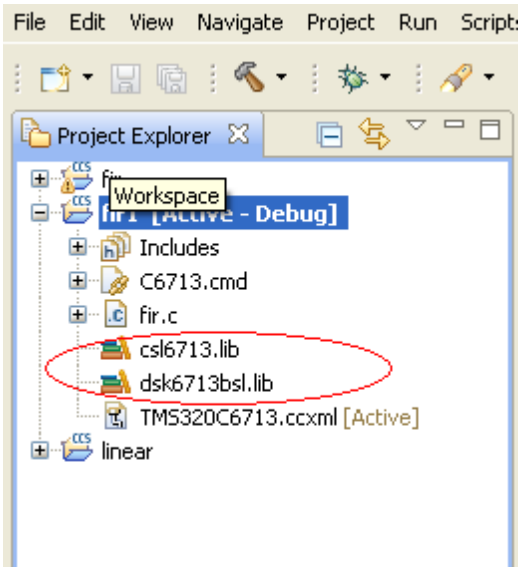


Then you have to add **bsl** and **csl** library files. These files you will get in below mentioned paths.

- BSL(Board support library file)
  Path: ***C:\CCStudio_v3.1\C6000\dsk6713\lib\dsk6713bsl.lib***

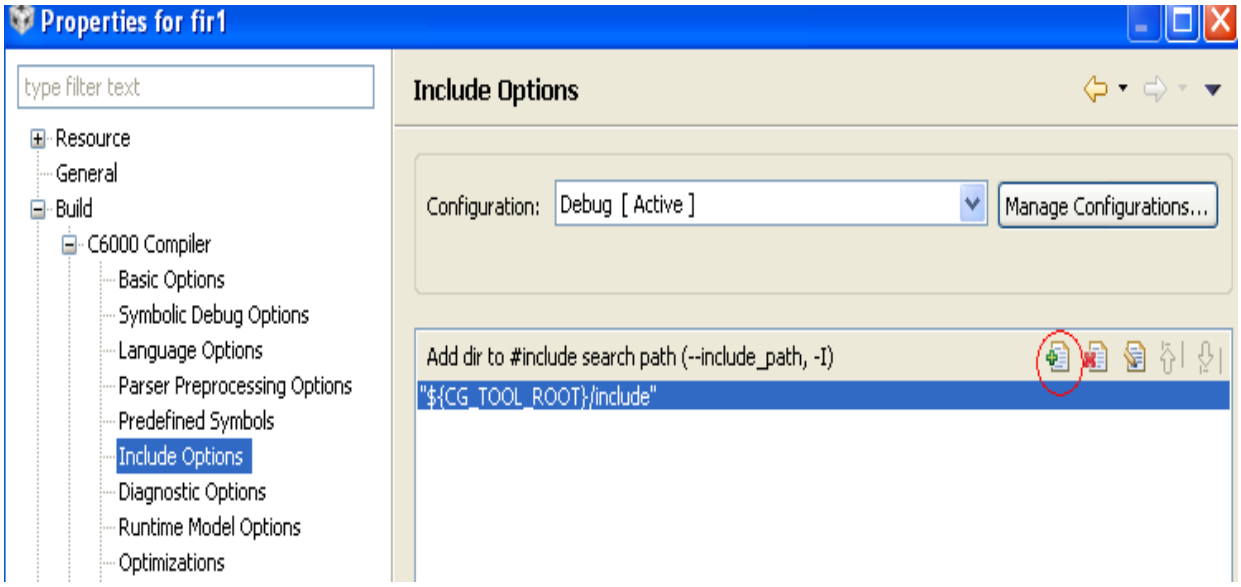
- CSL(Chip support library file)
  Path: ***C:\CCStudio_v3.1\C6000\csl\lib\csl6713.lib***

You can see these files added to your project as shown below:

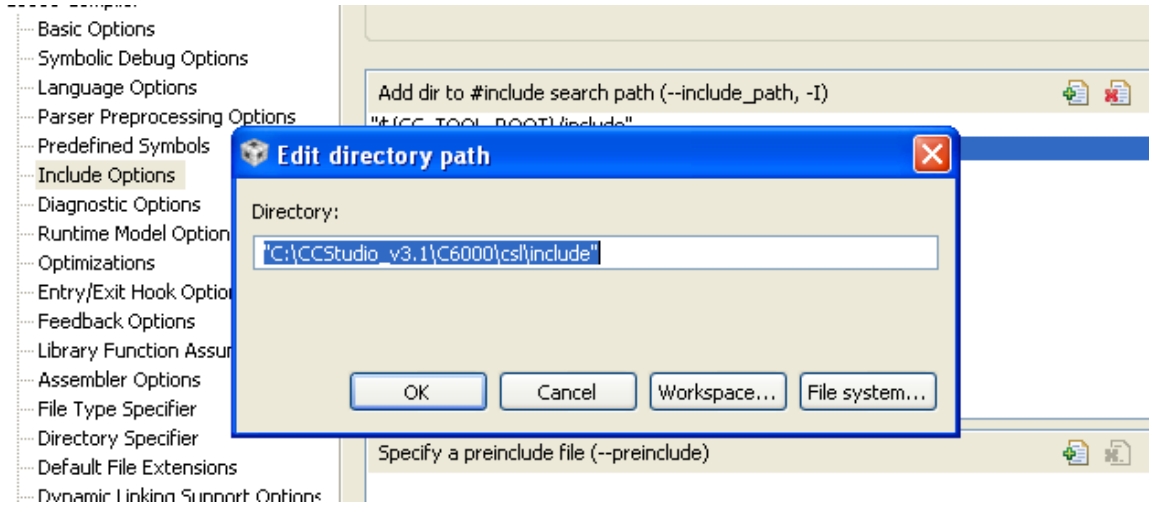


Next step is to add path and pre-defined names as shown below

- Go to **project-→properties** , under **Build→C6000 Compiler→include options** now click on **add** as shown in below:

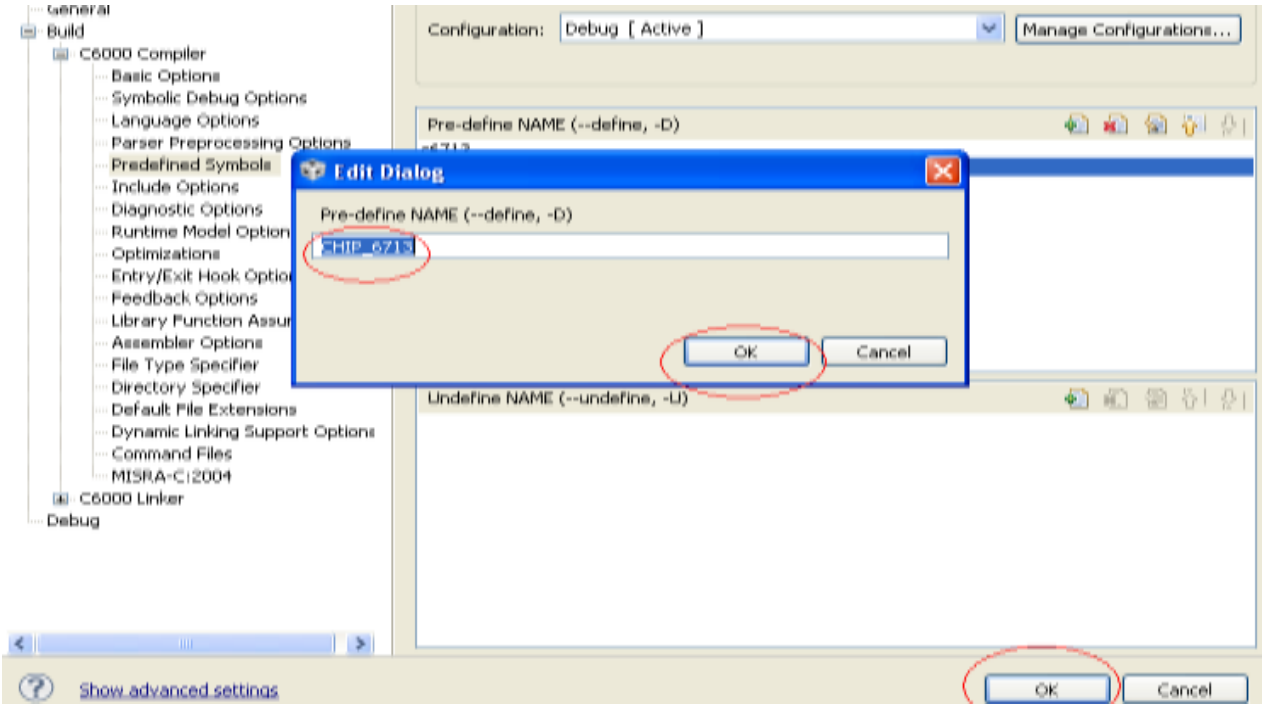


➢Then click on file system and go to this path *C:\CCStudio_v3.1\C6000\csl\include*,then ok.



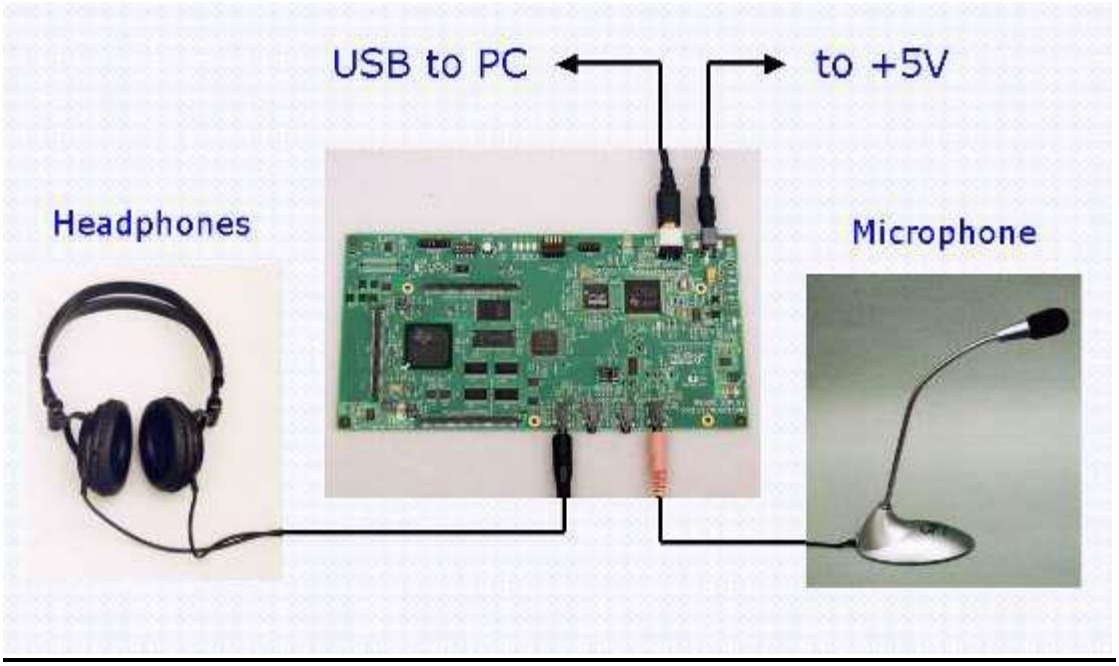
- Click again on add icon in include option window go to the path ***C:\CCStudio_v3.1\C6000\dsk6713\include,*** then ok.

➢ Now again under same build go to **Build→C6000 Compiler→Predefined symbols,** Then again click on **add** (On top right side Plus indication in green color symbol). Then type as "**CHIP_6713"** then apply ok as shown in below:



Save Project, Debug and run Project

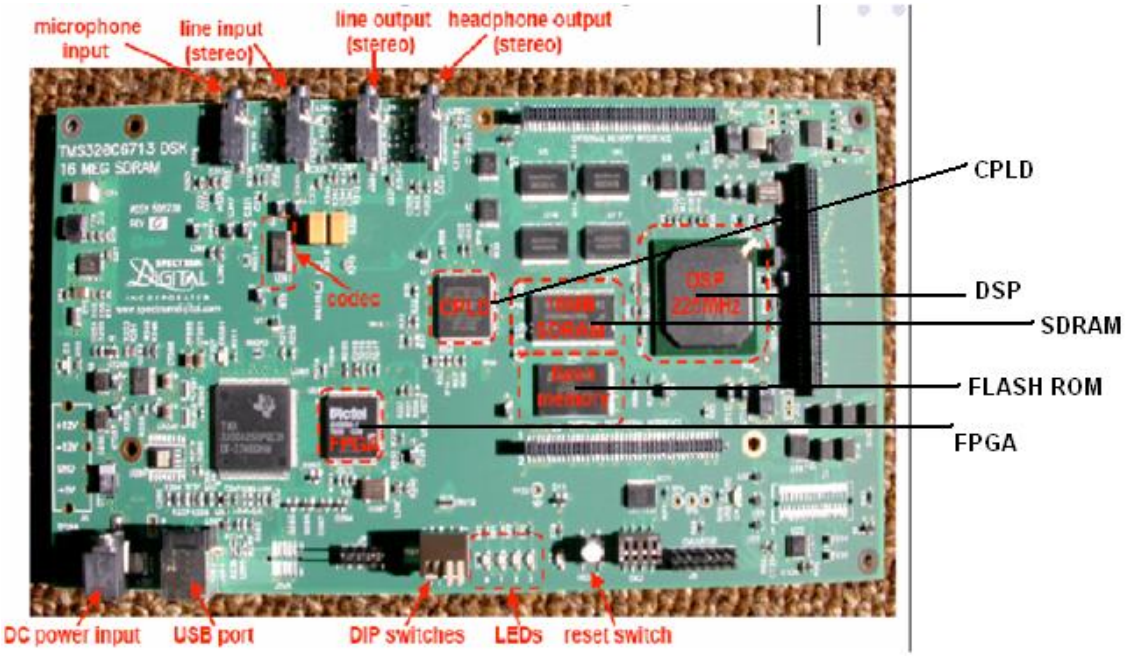
Connect CRO through stereo cable to the **LINE OUT**.
Connect a Signal Generator through stereo cable to the **LINE IN** Socket. As shown below:



VIVA QUESTION AND ANSWERS FOR DSP LAB [B20EN0605]

1. Features of TMS320C6713 Starter kit :-

- TMS320C6713 DSP operating at 225MHz device delivering up to 1800 million instructions per second (MIPs) and 1350 MFLOPS.
- Embedded USB JTAG controller with plug and play drivers, USB cable included
- TI TLV320AIC23 codec → This IC is used to transmit and receive analog signals.
- 16MB SDRAM **I.C No. 48LC4M32B2 → 4MEGX32 (1MEGX32X4banks)**
- 512K bytes of on board Flash ROM **I.C No. M29W400T**
- Expansion connectors (Memory Interface, Peripheral Interface, and Host Port Interface)
- On board IEEE 1149.1 JTAG connection for optional emulator debug
- +5V universal power supply
- High-quality 24-bit stereo codec
- user definable LEDs
- position dip switch, user definable
- Size: 8.25" x 4.5" (210 x 115 mm), 0.062" thick, 6 layers
- Compatible with Spectrum Digital's DSK Wire Wrap Prototype Card
- RoHS Compliant
- CPLD **I.C.No. MAX EPM3128A TC100-10N** {MAX3000A, 128 macro cells, 100TQFP, 8logic array blocks, 96 I/O pins and 10ns.}
- FPGA **I.C.No. ACTEL A54SX08A-TQG100** {FPGA SX-A Family, 8K gates, 512 cells, 238MHz,0.25um/0.22um(CMOS) tech.,2.5v and 100 pin TQFP}



2. **Expansion of TMS320C6713 DSK ?**

T	→	TEXAS INSTRUMENTS
MS	→	MIXED SIGNALS
32	→	It's a 32 bit processor
0	→	Floating point
C	→	CMOS Technology
6713	→	The series of the processor kit.
DSK	→	Digital Starter Kit.

3. Difference between DSP processor and microprocessor?

Functions of DSP:-

DSPs are designed specifically to perform large numbers of complex arithmetic calculations as quickly as possible, usually in applications such as image processing, speech recognition and telecommunications. DSPs are more efficient than general-purpose microprocessors at performing basic arithmetic operations, especially multiplication. They are found in devices that require rapid processing of audio or video data in real time, such as cell phones, DVD players and digital cameras.

Functions of microprocessor:-

In the computing world, faster is always better. However, microprocessors are general-purpose devices. They are designed to run software applications such as word processors, spreadsheets and web browsers. These are applications where speed is important but not critical. Microprocessors are at the core of desktops, laptops, netbooks and tablet PCs.

4. Difference between fixed point and floating point?

Digital signal processing can be separated into two categories - fixed point and floating point. These designations refer to the format used to store and manipulate numeric representations of data. Fixed-point DSPs are designed to represent and manipulate integers – positive and negative whole numbers – via a minimum of 16 bits, yielding up to 65,536 possible bit patterns (2^{16}). Floating-point DSPs represent and manipulate rational numbers via a minimum of 32 bits in a manner similar to scientific notation, where a number is represented with a mantissa and an exponent (e.g., A x 2^B, where 'A' is the mantissa and 'B' is the exponent), yielding up to 4,294,967,296 possible bit patterns (2^{32}).

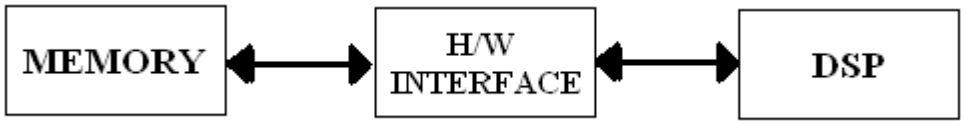
The term 'fixed point' refers to the corresponding manner in which numbers are represented, with a fixed number of digits after, and sometimes before, the decimal point. With floating-point representation, the placement of the decimal point can 'float' relative to the significant digits of the number. For example, a fixed-point representation with a uniform decimal point placement convention can represent the numbers 123.45, 1234.56, 12345.67, etc, whereas a floating-point representation could in addition represent 1.234567, 123456.7, 0.00001234567, 1234567000000000, etc. As such, floating point can support a much wider range of values than fixed point, with the ability to represent very small numbers and very large numbers.

5. What is the use of FPFA and CPLD in TMS320C6713 PROCESSOR?

DSPs often have to interface with external memory, typically shared with host processors or with other DSPs. The two main mechanisms available to implement the memory interfacing are to use hardware interfaces already existing on the DSP chip or to provide external hardware that carries out the memory interfacing. These two methods are briefly mentioned below.

Hardware interfaces are often available on TI as well as on ADI DSPs. An example is TI External Memory Interface (EMIF) [38], which is a glue less interface to memories such as SRAM,EPROM, Flash, Synchronous Burst SRAM (SBSRAM) and Synchronous DRAM (SDRAM). On theTMS320C6713 DSP, for instance, the EMIF provides 512 Mbytes of addressable external memory space. Additionally, the EMIF supports memory width of 8 bits, 12 bits and 32 bits, including read/write of both big- and little-endian devices.

When no dedicated on-chip hardware is available, the most common solution for interfacing a DSP to an external memory is to add external hardware between memory and DSP, as shown in below block diagram. Typically this is done by using a CPLD or an FPGA which implements address decoding and access arbitration. Care must be taken when programming the access priority and/or interleaved memory access in the CPLD/FPGA. This is essential to preserve the data integrity. Synchronous mechanisms should be preferred over asynchronous ones to carry out the data interfacing.



6. Which software is used to run TMS320C6713 processor and note about its features?

The software used is Code Composer Studio V3.1

CCS provides an IDE to incorporate the software tools. CCS includes tools for code generation, such as a C compiler, an assembler, and a linker. It has graphical capabilities and supports real-time debugging. It provides an easy-to-use software tool to build and debug programs.

The C compiler compiles a C source program with extension .c to produce an assembly source file with extension.asm. The assembler assembles an.asm source file to produce a machine language object file with extension.obj. The linker combines object files and object libraries as input to produce an executable file with extension .out.

7. What is compiler?

A program that translates a high level language into machine level language program is Called compiler.

8. What is an assembler?

Assembler is a software that translates assembly language program to a machine language program.

9. What is linker?

Linker is a software that joins together several object files and library functions into one large executable file.

10. What is the function of linker command file in TMS320C6713?

The function of linker command file is that it maps sections to memory.

11. What is debugger?

Debugger is a program that allows the execution of a program in single step mode under control of user.

12. What does the building process does in TMS320C6713?

The building process causes all the dependent files to be included.

13. What is convolution and mention its properties?

Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in digital signal processing. Using the strategy of impulse decomposition, systems are described by a signal called the *impulse response*. Convolution is important because it relates the three signals of interests: the input signals, the output signals and the impulse response.

The three basic properties of convolution as an algebraic operation are that it is commutative, associative, and distributive over addition. The commutative property means simply that x convolved with *h* is identical with *h* convolved with x. The consequence of this property for LTI systems is that for a system with a specified input and impulse response, the output will be the same if the roles of the input and impulse response are interchanged. The associative property specifies that while convolution is an operation combining two signals, we can refer unambiguously to the convolution of three signals without concern about how they are grouped pair wise.

The associative property combined with the commutative property leads to an extremely important property of LTI systems. Specifically, if we have several LTI systems cascaded together, the output generated by an input to the overall cascade combination does not depend on the order in which the systems are cascaded. This property of LTI systems plays an extremely important role in system design, implementation, and analysis. It is generally not true for arbitrary systems that are not linear and time-invariant, and it represents one very important consequence of exploiting the properties of linearity and time invariance.

The distributive property states that a signal convolved with the sum of two signals is identical to the result of carrying out the convolution with each signal in the sum individually and then summing the result.

14. What is the function of 'clc' in matlab tool?

CLC clears the command window and homes the cursor.

15. What is the function of 'clear all' in matlab tool?

CLEAR removes all variables from the workspace.

16. What is the function of 'close all' in matlab tool?

CLOSE ALL closes all the open figure windows.

17. What is the function of 'figure' in matlab tool?

FIGURE, by itself, creates a new figure window, and returns its handle.

18. What is the function of 'subplot' in matlab tool?

Subplot divides the figure window into rows, columns and position For example subplot(3,1,1) means the figure window divides into 3 rows, 1column and 1st position.

19. What is the function of 'plot' in matlab tool?

PLOT Linear plot.

PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix, then the vector is a

plotted versus the rows or columns of the matrix, which ever line up. If X is a scalar and Y is vector, length (Y) disconnected points are plotted.

20. What is the function of 'stem' in matlab tool?

STEM Discrete sequence or "stem" plot.
STEM(Y) plots the data sequence Y as stems from the x axis terminated with circles for the data value. STEM(X,Y) plots the data sequence Y at the values specified in X.

21. What is the function of 'xlabel' in matlab tool?

XLABEL X-axis label. XLABEL('text') adds text beside the X-axis on the current axis.

22. What is the function of 'ylabel' in matlab tool?

YLABEL Y-axis label. YLABEL('text') adds text beside the Y-axis on the current axis.

23. What is the function of 'title' in matlab tool?

TITLE Graph title. TITLE('text') adds text at the top of the current axis.

24. What is the function of 'conv' in matlab tool?

CONV Convolution and polynomial multiplication.
C = CONV(A, B) convolves vectors A and B. The resulting vector is LENGTH(A)+LENGTH(B)-1.
If A and B are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

25. What is the function of 'FFT' in matlab tool?

FFT Discrete Fourier transform.
FFT(X) is the discrete Fourier transform (DFT) of vector X. For matrices, the FFT Operation is applied to each column. For N-D arrays, the FFT operation operates on the first non-singleton dimension. FFT(X,N) is the N-point FFT, padded with zeros if X has less than N points and truncated if it has more.

26. What is the function of 'IFFT' in matlab tool?

IFFT Inverse discrete Fourier transform.
IFFT(X) is the inverse discrete Fourier transform of X.
IFFT(X,N) is the N-point inverse transform.

27. What is the function of 'xcorr' in matlab tool?

XCORR Cross-correlation function estimates.
C = XCORR(A,B), where A and B are length M vectors (M>1), returns the length 2*M-1 cross- correlation sequence C. If A and B are of different length, the shortest one is zero-padded. C will be a row vector if A is a row vector, and a column vector if A is a column vector. XCORR(A), when A is a vector, is the auto-correlation sequence.

28. What is the function of 'conj' in matlab tool?

CONJ Complex conjugate. CONJ(X) is the complex conjugate of X. For a complex X, CONJ(X) = REAL(X) - i*IMAG(X).

29. What is the function of 'ceil' in matlab tool?

CEIL Round towards plus infinity. CEIL(X) rounds the elements of X to the nearest integers towards infinity. For example: ceil([-2.7, 2.7]) => -2 3

30. What is the function of 'filter' in matlab tool?

FILTER One-dimensional digital filter.

Y = FILTER(B,A,X) filters the data in vector X with the filter described by vectors A and B to create the filtered data Y. The filter is a "Direct Form II Transposed" Implementation of the standard difference equation:

a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) + ... + b(nb+1)*x(n-nb)

- a(2)*y(n-1) - ... - a(na+1)*y(n-na)

If a(1) is not equal to 1, FILTER normalizes the filter coefficients by a(1).

31. What is the function of 'filtic' in matlab tool?

FILTIC Make initial conditions for 'filter' function.

Z = filtic(B, A, Y, X) converts past input X and output Y into initial conditions for the state variables Z needed in the TRANSPOSED DIRECT FORM II filter structure.

Z = filtic(B, A, Y) assumes that X = 0 in the past.

32. What is the function of 'fliplr' in matlab tool?

FLIPLR Flip matrix in left/right direction.

FLIPLR(X) returns X with row preserved and columns flipped in the left/right direction.

Example:

X =[1 2 3 becomes [3 2 1

4 5 6] 6 5 4]

33. What is the function of 'butter' in matlab tool?

BUTTER Butterworth digital and analog filter design.

[B,A] = BUTTER(N,Wn) designs an Nth order lowpass digital Butterworth filter and returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cutoff frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate.

34. What is the function of 'buttord' in matlab tool?

BUTTORD Butterworth filter order selection.

[N, Wn] = BUTTORD(Wp, Ws, Rp, Rs) returns the order N of the lowest order digital Butterworth filter that loses no more than Rp dB in the passband and has at least Rs dB of attenuation in the stopband.

Wp and Ws are the passband and stopband edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample). For example,

Lowpass: Wp = .1, Ws = .2

Highpass: Wp = .2, Ws = .1

Bandpass: Wp = [.2 .7], Ws = [.1 .8]

Bandstop: Wp = [.1 .8], Ws = [.2 .7]

35. What is the function of 'cheby1' in matlab tool?

CHEBY1 Chebyshev Type I digital and analog filter design.
[B,A] = CHEBY1(N,R,Wn) designs an Nth order lowpass digital Chebyshev filter with R decibels of peak-to-peak ripple in the passband. CHEBY1 returns the filter Coefficients in length N+1 vectors B (numerator) and A (denominator). The cutoff - frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. Use R=0.5 as a starting point, if you are unsure about choosing R.

36. What is the function of 'cheb1ord' in matlab tool?

CHEB1ORD Chebyshev Type I filter order selection.
[N, Wn] = CHEB1ORD(Wp, Ws, Rp, Rs) returns the order N of the lowest order digital Chebyshev Type I filter that loses no more than Rp dB in the passband and has at least Rs dB of attenuation in the stopband. Wp and Ws are the passband and stopband edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample). For example,

Lowpass: Wp = .1, Ws = .2
Highpass: Wp = .2, Ws = .1
Bandpass: Wp = [.2 .7], Ws = [.1 .8]
Bandstop: Wp = [.1 .8], Ws = [.2 .7]

37. What is meant by DFT ?

The discrete Fourier transform (DFT) is the primary transform used for numerical computation in digital signal processing. It is very widely used for spectrum analysis, fast convolution, and many other applications. The DFT transforms N discrete-time
samples to the same number of discrete frequency samples, and is defined as:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, e^{-j2\pi nk/N}$$

38. What are the applications of DFT?

1. **Spectral analysis**

When the DFT is used for spectral analysis, the $\{x_n\}$ sequence usually represents a finite set of uniformly spaced time-samples of some signal $x(t)$, where *t* represents time. The conversion from continuous time to samples (discrete-time) changes the underlying fourier transform of x(t) into a discrete-time Fourier transform (DTFT), which generally entails a type of distortion called *aliasing*.

2. Data compression

The field of digital signal processing relies heavily on operations in the frequency domain (i.e. on the Fourier transform). For example, several lossy image and sound compression methods employ the discrete Fourier transform: the signal is cut into short segments, each is transformed, and then the Fourier coefficients of high frequencies, which are assumed to be unnoticeable, are discarded. The decompressor computes the inverse transform based on this reduced number of Fourier coefficients. (Compression applications often use a specialized form of the DFT, the discrete cosine transform or sometimes the modified discrete cosine transform.)

3. Partial differential equations

Discrete Fourier transforms are often used to solve partial differential equations, where again the DFT is used as an approximation for the Fourier series (which is recovered in the limit of infinite *N*). The advantage of this approach is that it expands the signal in complex exponentials e^{inx}, which are Eigen functions of differentiation: $d/dx\ e^{inx} = in\ e^{inx}$.

39. What is meant by FFT?

Fast Fourier Transform is an algorithm used to compute DFTs.

40. What is meant by IDFT?

The inverse DFT (IDFT) transforms N discrete-frequency samples to the same number of discrete- time samples. The IDFT has a form very similar to the DFT,

$$x(n) = 1/N \sum_{k=0}^{N-1} X(k)\, e^{j2\pi nk/N}$$

41. What is meant by IFFT?

The IFFT block computes the inverse fast Fourier transform (IFFT) of each row of a sample-based 1-by-P input vector, or across the first dimension (P) of an N-D input array.

42. Difference between analog filter and digital filters?

Digital Filters:

1. It operates on the digital samples of the signals.
2. These kinds of filters are defined using linear difference equations.
3. While implementing the digital filters in hardware or software (for simulation), we need adders, subtractors, delays, etc which are classified under digital logic components.
4. In this filter, the filter coefficients are designed to meet the desired or expected frequency response.
5. Mathematically the transfer function ***H(z)*** is required to be a rational function of **z**, where the coefficients of z are real to meet the stability and causality requirement.

6. In order to be stable and causal, the poles of the transfer function should lie inside the unit circle in z-plane.

Analog Filters:

1. Unlike digital, analog filters works on analog signals or the so called actual signals.
2. It is defined by linear differential equations.
3. While implementing the analog filters in hardware or software simulation, electrical components like resistors, capacitors and inductors are used.
4. To achieve the required frequency response, approximation problem is solved.
5. To be stable and causal, the transfer function ***H(s)*** must be a rational function of ***s***, whose coefficients are real.
6. For stability and causality, the poles should lie on the left half of s-plane.

43. How do you convert analog filter prototype to a digital filter?

There are five methods to convert analog filter prototype to a digital filter, they are:

(1). Impulse-invariant method
(2). Frequency mapping method
(3). Bilinear transformation method
(4). Matched Z transform
(5). Backward difference method

44. What are the steps to be taken while designing a digital filter?

When designing a digital filter using an analog approximation and the bilinear transform, we follow these steps:

1. Prewarp the cutoff frequencies
2. Design the necessary analog filter
3. apply the bilinear transform to the transfer function
4. Normalize the resultant transfer function to be monotonic and have a unity passband gain (0dB).

Alternatively, if we have an inverse bilinear transform, we can follow these steps:

a. Use the inverse bilinear transform on the filter specifications in the digital domain to produce equivalent specifications in the analog domain.
b. Construct the analog filter transfer functions to meet those specifications.
c. Use the bilinear transform to convert the resultant analog filter into a digital filter.

45. What is prewarping?

Frequency warping follows a known pattern, and there is a known relationship between the warped frequency and the known frequency. We can use a technique called prewarping to account for the nonlinearity, and produce a more faithful mapping.

$$\omega_p = \frac{2}{T}\tan\left(\frac{\omega}{2}\right)$$

The *p* subscript denotes the prewarped version of the same frequency.

46. What is bilinear transform?

The Bilinear transform is a mathematical relationship which can be used to convert the transfer function of a particular filter in the complex Laplace domain into the z-domain, and vice-versa. The resulting filter will have the same characteristics of the original filter, but can be implemented using different techniques. The Laplace Domain is better suited for designing analog filter components, while the Z-Transform is better suited for designing digital filter components.

The bilinear transform is the result of a numerical integration of the analog transfer function into the digital domain. We can define the bilinear transform as:

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

47. Differentiate between Butterworth and Chebyshev filter?

Some of the important differences are as follows:

Magnitude response vs frequency curve: The magnitude response |H(jw)| of the butterworth filter decreases with increase in frequency from 0 to infinity, while the magnitude response of the Chebyshev filter fluctuates or show ripples in the passband and stopband depending on the type of the filter.

Width of Transition band: The width of the transition band is more in Butterworth filter compared to the Chebyshev filter.

Location of the poles: The poles of a Butterworth filter lies only on a circle while that of the Chebyshev filter lies on an ellipse, which can be easily concluded on looking at the poles formula for both types of filters.

No. Of Components required for implementing the filter: The number of poles in Butterworth filter is more compared to that of the Chebyshev filter of same specifications, this means that the order of Butterworth filter is more than that of a Chebyshev filter. This fact can be used for practical implementation, since the number of components required to construct a filter of same specification can be reduced significantly.

48. What is correlation?

Correlation is a mathematical operation that is very similar to convolution. Just as with convolution, correlation uses two signals to produce a third signal. This third signal is called the **cross-correlation** of the two input signals. If a signal is correlated with *itself,* the resulting signal is instead called the **autocorrelation**.

49. What are the applications of autocorrelation?

a) One application of autocorrelation is the measurement of optical spectra and the measurement of very-short-duration light pulses produced by lasers, both using optical autocorrelators.

b) Autocorrelation is used to analyze Dynamic light scattering data, which notably enables to determine the particle size distributions of nanometer-sized particles or micelles suspended in a fluid.

c) In signal processing, autocorrelation can give information about repeating events like musical beats (for example, to determine tempo) or pulsar frequencies, though it cannot tell the position in time of the beat.

d) In music recording, autocorrelation is used as a pitch detection algorithm prior to vocal processing, as a distortion effect or to eliminate undesired mistakes and inaccuracies.

50. What are the applications of cross correlation?

The cross-correlation function is used extensively in *pattern recognition* and *signal detection*. We know that projecting one signal onto another is a means of measuring how much of the second signal is present in the first. This can be used to ``detect'' the presence of known signals as components of more complicated signals. As a simple example, suppose we record *x(n)* which we think consists of a signal *s(n)* which we are looking for plus some additive measurement noise *e(n)*. Then the projection of x onto s is

$$\mathcal{P}_s(x) = \mathcal{P}_s(s) + \mathcal{P}_s(e) \approx s$$

since the projection of any specific signal s onto random, zero-mean noise is close to zero. Another term for this process is called *matched filtering*. The impulse response of the ``matched filter'' for a signal x is given by *Flip(x)*. By time reversing x, we transform the convolution implemented by filtering into a cross-correlation operation.

51. What are the properties of autocorrelation?

a) A fundamental property of the autocorrelation is even symmetry,

b) The continuous autocorrelation function reaches its peak at the origin, where it takes a real value, i.e. for any delay.

c) The autocorrelation of a periodic function is, itself, periodic with the same period.

52. What are the properties of cross correlation?

a. The cross-correlation of functions $f(t)$ and $g(t)$ is equivalent to the convolution of $f^*(-t)$ and

$g(t)$. I.e.: $f \star g = f^*(-t) * g.$

b. Analogous to the convolution theorem, the cross-correlation satisfies:

$$\mathcal{F}\{f \star g\} = (\mathcal{F}\{f\})^* \cdot \mathcal{F}\{g\},$$

53. What is the difference between cross correlation and auto correlation?

Sl No.	Cross correlation	Auto Correlation
01	When two independent signals are compared, the procedure is known as cross-correlation	When the same signal is compared to phase shifted copies of itself, the procedure is known as auto correlation
02	Cross-correlation is the method which basically underlies implementations of the Fourier transformation: signals of varying frequency and phase are correlated with the input signal, and the degree of correlation in terms of frequency and phase represents the frequency and phase spectrums of the input signal.	Autocorrelation is a method which is frequently used for the extraction of fundamental frequency, : if a copy of the signal is shifted in phase, the distance between correlation peaks is taken to be the fundamental period of the signal (directly related to the fundamental frequency)

54. What is impulse response?

In signal processing, the **impulse response**, or **impulse response function (IRF)**, of a dynamic system is its output when presented with a brief input signal, called an **impulse**. More generally, an impulse response refers to the reaction of any dynamic system in response to some external change. In both cases, the impulse response describes the reaction of the system as a function of time (or possibly as a function of some other independent variable that parameterizes the dynamic behavior of the system).

55. What is sampling?

Sampling is a process of converting a continuous time signal (analog signal) i.e., x(t) into a discrete time signal i.e., x[n] which is represented as a sequence of numbers (Analog to Digital converter).

56. What is quantization?

Quantization is the process of converting a discrete time continuous amplitude signal x(n) into a dicrete-time discrete amplitude signal $z_q(n)$.

57. What is aliasing?

Aliasing is an effect that causes different signals to become indistinguishable (or *aliases* of one another) when sampled. It also refers to the distortion or artifact that results when the signal reconstructed from samples is different from the original continuous signal.

58. State sampling theorem?

The sampling theorem states that a set of samples of a signal can be reconstructed into the original signal if and only if the original system is band limited and the sampling frequency is greater than twice the maximum frequency for non-zero values of the original function.

or

It states that, while taking the samples of a continuous signal, it has to be taken care that the sampling rate is equal to or greater than twice the cut off frequency and the minimum sampling rate is known as the Nyquist rate.

59. What is a system?

A system is defined as a physical device that generates a response or an output signal for a given input signal.

60. What are causal and non-causal system?

A system is said to be causal if the output of the system at any time 'n' depends only at present and past inputs but does not depend on future inputs.

If the output of a system depends on future inputs then the system is called non-causal system.

61. What are linear and non-linear system?

A system that satisfies the superposition principle is said to be linear system. Super position principle states that the response of the system to a weighted sum of signals should be equal to the corresponding weighted sum of the output's of the system to each of the individual input signals.

A relaxed system that does not satisfy the superposition principle is called non-linear system.

62. What is an FIR system?

If the impulse response of the system is of finite duration then the system is called **F**inite **I**mpulse **R**esponse.

63. What is a *linear phase* filter?

"Linear Phase" refers to the condition where the phase response of the filter is a linear (straight-line) function of frequency (excluding phase wraps at +/- 180 degrees). This results in the *delay* through the filter being the same at all frequencies. Therefore, the filter does not cause "phase distortion" or "delay distortion". The lack of phase/delay distortion can be a critical advantage of FIR filters over IIR and analog filters in certain systems, for example, in digital data modems.

64. What is the condition for linear phase?

FIR filters are usually designed to be linear-phase (but they don't have to be.) A FIR filter is linear-phase if (and only if) its coefficients are symmetrical around the center coefficient, that is, the first coefficient is the same as the last; the second is the same as the next-to-last, etc. (A linear-phase FIR filter having an odd number of coefficients will have a single coefficient in the center which has no mate.)

65. What is the delay of a linear-phase FIR?

The formula is simple: given a FIR filter which has N taps, the delay is: (N - 1) / (2 * Fs), where Fs is the sampling frequency. So, for example, a 21 tap linear-phase FIR filter operating at a 1 kHz rate has delay: (21 - 1) / (2 * 1 kHz)=10 milliseconds.

66. What is the Z transform of a FIR filter?

For an N-tap FIR filter with coefficients h(k), whose output is described by:

$y(n)=h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + ... h(N-1)x(n-N-1)$,

The filter's Z transform is:

$H(z)=h(0)z-0 + h(1)z-1 + h(2)z-2 + ... h(N-1)z-(N-1)$, or $H(z) \sum_{n=0}^{N-1} h(n)z^{-n}$

67. Can I calculate the frequency response of a FIR using the Discrete Fourier Transform (DFT)?

Yes. For an N-tap FIR, you can get N evenly-spaced points of the frequency response by doing a DFT on the filter coefficients.

68. How do i scale the gain of FIR filter?

Simply multiply all coefficients by the scale vector.

69. Are FIR filters inherently stable?

Yes. Since they have no feedback elements, any bounded input results in a bounded output.

70. What makes the numerical properties of FIR filters "good"?

The key is the lack of feedback. The numeric errors that occur when implementing FIR filters in computer arithmetic occur separately with each calculation; the FIR doesn't "remember" its past numeric errors. In contrast, the feedback aspect of IIR filters can cause numeric errors to compound with each calculation, as numeric errors are fed back.

71. Why are FIR filters generally preferred over IIR filters in multirate (decimating and interpolating) systems?

Because only a fraction of the calculations that would be required to implement a decimating or interpolating FIR in a literal way actually needs to be done.

Since FIR filters do not use feedback, only those outputs which are actually going to be used have to be calculated. Therefore, in the case of decimating FIRs (in which only 1 of N outputs will be used), the other N-1 outputs don't have to be calculated. Similarly, for interpolating filters (in which zeroes are inserted between the input samples to raise the sampling rate) you don't actually have to multiply the inserted zeroes with their corresponding FIR coefficients and sum the result; you just omit the multiplication-additions that are associated with the zeroes (because they don't change the result anyway.)

In contrast, since IIR filters use feedback, every input must be used, and every input must be calculated because all inputs and outputs contribute to the feedback in the filter.

72. What is an IIR System?

If the impulse response of the system is of infinite duration then the system is called **I**nfinite **I**mpulse **R**esponse.

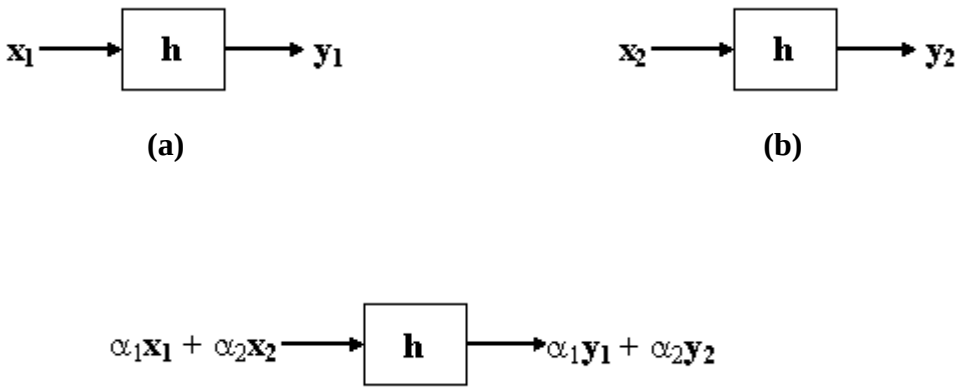
73. Compare FIR and IIR filters?

Sl. No.	IIR	FIR
01	IIR filters are difficult to control and have no particular phase,	FIR filters make a linear phase always possible.
02	IIR can be unstable	FIR is always stable.
03	IIR, when compared to FIR, can have limited cycles,	FIR has no limited cycles.
04	IIR is better for lower-order tapping	The FIR filter is used for higher-order tapping

74. Explain the properties of circular convolution?

The properties of circular convolution are:

a) Circular convolution (CC) is ***linear***



(a) **(b)**

(c)

b) Circular Convolution is ***shift invariant***

(a) **(b)**

c) Circular Convolution is ***commutative***

75. What is the difference between convolution and correlation?

Correlation is a metric for similarity between two different signals. When normalized, so that each of the two signals to be correlated have unitary power and null mean value, the correlation operation shifts to the computation of the correllation coefficient, of the two signals you are comparing. The correlation coefficient, for a given time lag t between the two signals, is always between -1 and +1, clearly giving a measure of the similarity of the shapes of the two signals at that time lag.

Convolution, instead, is the common operation a Linear and Time Invariant system can perform on a given input signal. It is clear that, in specific cases, correlation anc convolution are very similar: the matched filter case makes correlation and convolution identical.

Convolution is a technique to find the output of a system of impulse response h(n) for an input x(n) so basically it is used to calculate the output of a system, while correlation is a process to find the degree of similarity between two signals.

76. What is the difference between linear and circular convolution?

Linear convolution takes two functions of an independent variable, which is called **time**, and convolves them using the convolution sum formula. Basically it is a correlation of one function with the time-reversed version of the other function. I think of it as flip, multiply, and sum while shifting one function with respect to the other. This holds in continuous time, where the convolution sum is an integral, or in discrete time using vectors, where the sum is truly a sum. It also holds for functions defined from -Inf to Inf or for functions with a finite length in time.

Circular convolution is only defined for finite length functions (usually, maybe always, equal in length), continuous or discrete in time. In circular convolution, it is as if the finite length functions repeat in time, periodically. Because the input functions are now periodic, the convolved output is also periodic and so the convolved output is fully specified by one of its periods.

77. How to convert degrees into radians and vice-versa?

To convert degrees into radians:

Example- 200^0 $200^0 * \pi / 180^0 \Rightarrow 10\pi/9 \Rightarrow 3.49$ *rad*

To convert radians into degrees:

Example- $4\pi/9 \Rightarrow \frac{4\pi}{9} * \frac{180^0}{\pi} \Rightarrow \frac{720^0\pi}{9\pi} \Rightarrow 80^0$

78. What are imaginary numbers?

An imaginary number is a number that can be written as a real number multiplied by the imaginary unit i, which is defined by its property $i^2 = -1$.

For example, $5i$ is an imaginary number, and its square is -25.

79. What are complex numbers?

The values which contain both real and imaginary numbers.

For example, $x + iy$, where x and y are real numbers and i is the imaginary unit equal to the square root of -1, $\sqrt{-1}$.

REVA UNIVERSITY
Bengaluru, India

Rukmini Knowledge Park, Kattigenahalli
Yelahanka, Bengaluru - 560 064
Karnataka, India.

Ph: +91- 90211 90211, +91 80 4696 6966
E-mail: admissions@reva.edu.in

Follow us on

/REVAUniversity