Geethanjali College of Engineering and Technology

Cheeryal (v), Keesara (M), Ranga Reddy District.

DIGITAL SIGNAL PROCESSING LABORATORY STUDENTS'MANUAL

For III year II semester ECE

A.Y.2015-16

…striving toward perfection

DEPARTMENT OF

ELECTRONICS & COMMUNICATION ENGINEERING

INCHARGES	**HOD**
Mr. R.Odaiah	**Dr. P.Srihari**

Geethanjali College of Engineering and Technology

Cheeryal (v), Keesara (M), Ranga Reddy District.

…striving toward perfection

LABORATORY MANUAL
FOR
DIGITAL SIGNAL PROCESSING

Prepared by:
Ms.S.Jyothirmaye

Checked by:

Approved by:
Dr. P.Srihari, HOD
Dept., of ECE

Revision No:4 **Date: 20/11/2015**

GEETHANJALI COLLEGE OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF

Electronics And Communications Engineering

(Name of the Subject / Lab Course) : DSP Lab

(JNTU CODE - A60493)	Programme : UG
Branch: ECE-A, B, C & D	Version No : 04
Year: III	Updated on :20/11/2015
Semester: II	No. of pages :116

Classification status (Unrestricted / Restricted)

Distribution List : Department , Lab, Library, Lab incharge

Prepared by : 1) Names :Ms. S.Jyothirmaye Ms. Kavya	1) Name : Ms. M. Umarani Mr. R.Odaiah
2) Sign :	2) Sign :
3) Design : Assoc. Prof	3) Design :
4) Date :	4) Date :

Verified by : 1) Name :	* For Q.C Only.
	1) Name :
2) Sign :	2) Sign :
3) Design :	3) Design :
4) Date :	4) Date :

Approved by : (HOD) 1) Name :Dr.P.Srihari

2) Sign :

3) Date :

List of Contents

Sl.No.	Title	Page No.
1.	JNTU syllabus.	i
2.	Dos and Don'ts.	ii
3.	Vision and mission of the Department.	iii
4.	Course objectives and outcomes.	v
5.	Basic Introduction	1
6.	List of Experiments	

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

III Year B.Tech. ECE - II Sem

L	T/P/D	C
0	-/3/-	2

JNTUH Syllabus

DIGITAL SIGNAL PROCESSING LAB

The programs shall be implemented in software (Using MATLAB / Lab view / C programming/ Equivalent) and hardware (Using TI / Analog devices / Motorola / Equivalent DSP processors).

1. Generation of Sinusoidal waveform / signal based on recursive difference equations.
2. To find DFT / IDFT of given DT signal.
3. To find frequency response of a given system given in (Transfer Function/ Differential equation form).
4. Implementation of FFT of given sequence.
5. Determination of Power Spectrum of a given signal(s).
6. Implementation of LP FIR filter for a given sequence.
7. Implementation of HP FIR filter for a given sequence.
8. Implementation of LP IIR filter for a given sequence.
9. Implementation of HP IIR filter for a given sequence.
10. Generation of Sinusoidal signal through filtering.
11. Generation of DTMF signals.
12. Implementation of Decimation Process.
13. Implementation of Interpolation Process.
14. Implementation of I/D sampling rate converters.
15. Audio application such as to plot a time and frequency display of microphone plus a cosine using DSP. Read a .wav file and match with their respective spectrograms.
16. Noise removal: Add noise above 3 KHz and then remove, interference suppression using 400 Hz tone.
17. Impulse response of first order and second order systems.

Note: - Minimum of 12 experiments has to be conducted.

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY HYDERABAD

III Year B.Tech. ECE - II Sem

DIGITAL SIGNAL PROCESSING LAB

Cycle I

1. Generation of Sinusoidal waveform / signal based on recursive difference equations.
2. To find DFT / IDFT of given DT signal.
3. To find frequency response of a given system given in (Transfer Function/ Differential equation form).
4. Implementation of FFT of given sequence.
5. Determination of Power Spectrum of a given signal(s).
6. Implementation of LP FIR filter for a given sequence.
7. Implementation of LP IIR filter for a given sequence.
8. Implementation of HP IIR filter for a given sequence.

Cycle II:

9. Implementation of HP IIR filter for a given sequence.
10. Implementation of Decimation Process.
11. Implementation of Interpolation Process.
12. Implementation of I/D sampling rate converters.

Additional experiments:

1. Determination of Power Spectrum of a given signal(s).
2. Converting CD DATA TO DVD DATA

Design Experiments

1. Audio application such as to plot a time and frequency display of microphone plus a cosine using DSP. Read a .wav file and match with their respective spectrograms.

2. Noise removal: Add noise above 3 KHz and then remove interference suppression using 400 Hz tone.

Open Experiment

1. Impulse response of first order and second order systems.
2. Implementation of I/D sampling rate converters

INSTRUCTIONS / Do's and Don't

Instruction to Students:-

1. Do not handle any equipment without reading the instructions /Instruction manuals.
2. Observe type of sockets of equipment power to avoid mechanical damage.
3. Do not insert connectors forcefully in the Sockets.
4. Strictly observe the instructions given by the Teacher/ Lab Instructor.
5. After the experiment is over, the students must hand over the Bread board, Trainer kits, wires, CRO Probes and other Components to the lab assistant / teacher.
6. It is mandatory to come to lab in a formal dress (Shirts, Trousers, ID card, and Shoes for boys). Strictly no Jeans for both Girls and Boys.
7. It is mandatory to come with observation book and lab record in which previous experiment should be written in Record and the present lab's experiment in Observation book.
8. Observation book of the present lab experiment should be get corrected on the same day and Record should be corrected on the next scheduled lab session.
9. Mobile Phones should be Switched OFF in the lab session.
10. Students have to come to lab in-time. Late comers are not allowed to enter the lab.
11. Prepare for the viva questions. At the end of the experiment, the lab faculty will ask the viva Questions and marks are allotted accordingly.
12. Bring all the required stationery like graph sheets, pencil & eraser, different color pens etc. for the lab class.

Instructions to Laboratory Teachers:-

1. Observation book and lab records submitted for the lab work are to be checked and signed before the next lab session.
2. Students should be instructed to switch ON the power supply after the connections are checked by the lab assistant / teacher.

3. The promptness of submission should be strictly insisted by awarding the marks accordingly.
4. Ask viva questions at the end of the experiment.
5. Do not allow students who come late to the lab class.
6. Encourage the students to do the experiments innovatively.

ECE DEPARTMENT

Vision of the Department

To impart quality technical education in Electronics and Communication Engineering emphasizing analysis, design/synthesis and evaluation of hardware/embedded software using various Electronic Design Automation (EDA) tools with accent on creativity, innovation and research thereby producing competent engineers who can meet global challenges with societal commitment.

Mission of the Department

i. To impart quality education in fundamentals of basic sciences, mathematics, electronics and communication engineering through innovative teaching-learning processes.
ii. To facilitate Graduates define, design, and solve engineering problems in the field of Electronics and Communication Engineering using various Electronic Design Automation (EDA) tools.
iii. To encourage research culture among faculty and students thereby facilitating them to be creative and innovative through constant interaction with R & D organizations and Industry.
iv. To inculcate teamwork, imbibe leadership qualities, professional ethics and social responsibilities in students and faculty.

Program Educational Objectives of B. Tech (ECE) Program :

I. To prepare students with excellent comprehension of basic sciences, mathematics and engineering subjects facilitating them to gain employment or pursue postgraduate studies with an appreciation for lifelong learning.

II. To train students with problem solving capabilities such as analysis and design with adequate practical skills wherein they demonstrate creativity and innovation that would enable them to develop state of the art equipment and technologies of multidisciplinary nature for societal development.

III. To inculcate positive attitude, professional ethics, effective communication and interpersonal skills which would facilitate them to succeed in the chosen profession exhibiting creativity and innovation through research and development both as team member and as well as leader.

Program Outcomes of B.Tech ECE Program:

1. An ability to apply knowledge of Mathematics, Science, and Engineering to solve complex engineering problems of Electronics and Communication Engineering systems.
2. An ability to model, simulate and design Electronics and Communication Engineering systems, conduct experiments, as well as analyze and interpret data and prepare a report with conclusions.
3. An ability to design an Electronics and Communication Engineering system, component, or process to meet desired needs within the realistic constraints such as economic, environmental, social, political, ethical, health and safety, manufacturability and sustainability.
4. An ability to function on multidisciplinary teams involving interpersonal skills.
5. An ability to identify, formulate and solve engineering problems of multidisciplinary nature.
6. An understanding of professional and ethical responsibilities involved in the practice of Electronics and Communication Engineering profession.
7. An ability to communicate effectively with a range of audience on complex engineering problems of multidisciplinary nature both in oral and written form.
8. The broad education necessary to understand the impact of engineering solutions in a global, economic, environmental and societal context.
9. Recognition of the need for, and an ability to engage in life-long learning and acquire the capability for the same.

10. A knowledge of contemporary issues involved in the practice of Electronics and Communication Engineering profession
11. An ability to use the techniques, skills and modern engineering tools necessary for engineering practice.
12. An ability to use modern Electronic Design Automation (EDA) tools, software and electronic equipment to analyze, synthesize and evaluate Electronics and Communication Engineering systems for multidisciplinary tasks.
13. Apply engineering and project management principles to one's own work and also to manage projects of multidisciplinary nature.

COURSE OBJECTIVES AND OUTCOMES

Objectives:

1. To generate the elementary signals/ waveforms.

2. To Calculate and Plot DFT / IDFT of given DT signal and prove it theoretical.

3. To plot frequency response of a given LTI system.

4. To Implement FFT of a given sequence.

5. To determine and plot the Power Spectrum of a given signal(s).

6. To Plot Magnitude and Phase of LP FIR filter for any given sequence.

7. To Plot Magnitude and Phase of HP FIR filter for a given sequence.

8. Plot Magnitude and Phase of LP IIR filter for a given sequence.

9. To Plot Magnitude and Phase of HP IIR filter for a given sequence.

10. To generate Sinusoidal signal through filtering.

11. To generate DTMF signals.

12. To Implement Decimation Process of any given sequence.

13. To Implement Interpolation Process of any given sequence.

14. To Implement I/D sampling rate converters.

15. To plot a time and frequency display of microphone plus a cosine using DSP.

16. To do Noise removal: Add noise above 3 KHz and then remove, interference suppression using 400Hz tone.

17. To plot Impulse response of first order and second order systems.

Learning Outcomes:

The student will be:

1. Able to generate elementary signals/ waveforms and perform arithmetic operations on signals.

2. Able to Calculate and Plot DFT / IDFT of given DT signal.

3. Able to plot frequency response of a given system and verify the properties of LTI system.

4. Able to Implement FFT of given sequence and identify the reduction of computations using FFT.

5. Able to Implement LP FIR filter for a given sequence and calculate the filter coefficients.

6. Able to Implement HP FIR filter for a given sequence and plot the response of the same.

7. Able to Implement and design LP IIR filter for a given sequence.

8. Able to Implement HP IIR filter for a given sequence.

9. Able to generate Sinusoidal signal through filtering.

10. Able to Implement Decimation Process and vary (decrease) the sampling rate.

11. Able to Implement Interpolation Process and vary (increase) the sampling rate.

12. Able to Implement I/D sampling rate converters and identify the importance of multi rate sampling.

13. Able to generate DTMF signals.

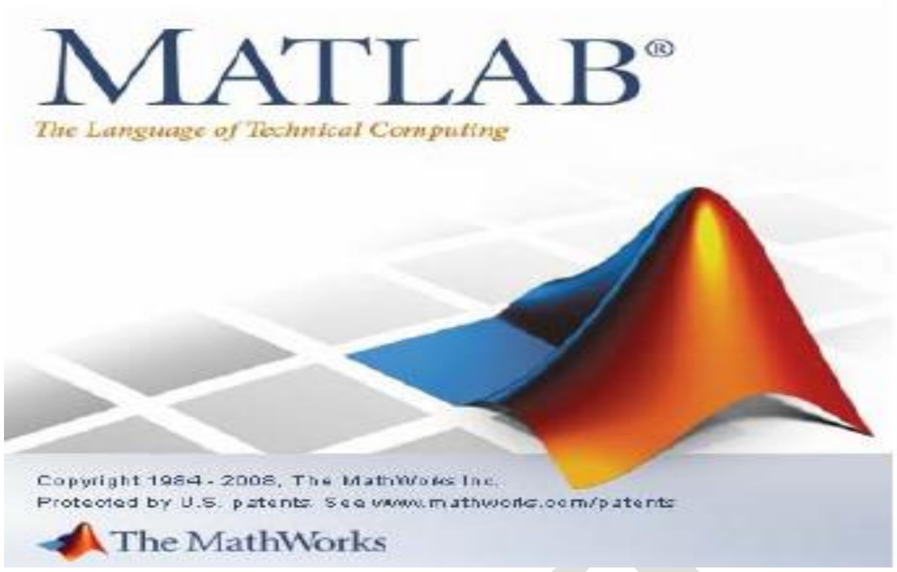
14. Able to determine the Power Spectrum of a given signal(s) and demonstrate the importance of frequency domain.

15. Able to plot a time and frequency display of microphone plus a cosine using DSP.

16. Able to do Noise removal: Add noise above 3 KHz and then remove, interference suppression using 400Hz tone.

17. Able to plot Impulse response of first order and second order systems and calculate the damping factor.

Introduction

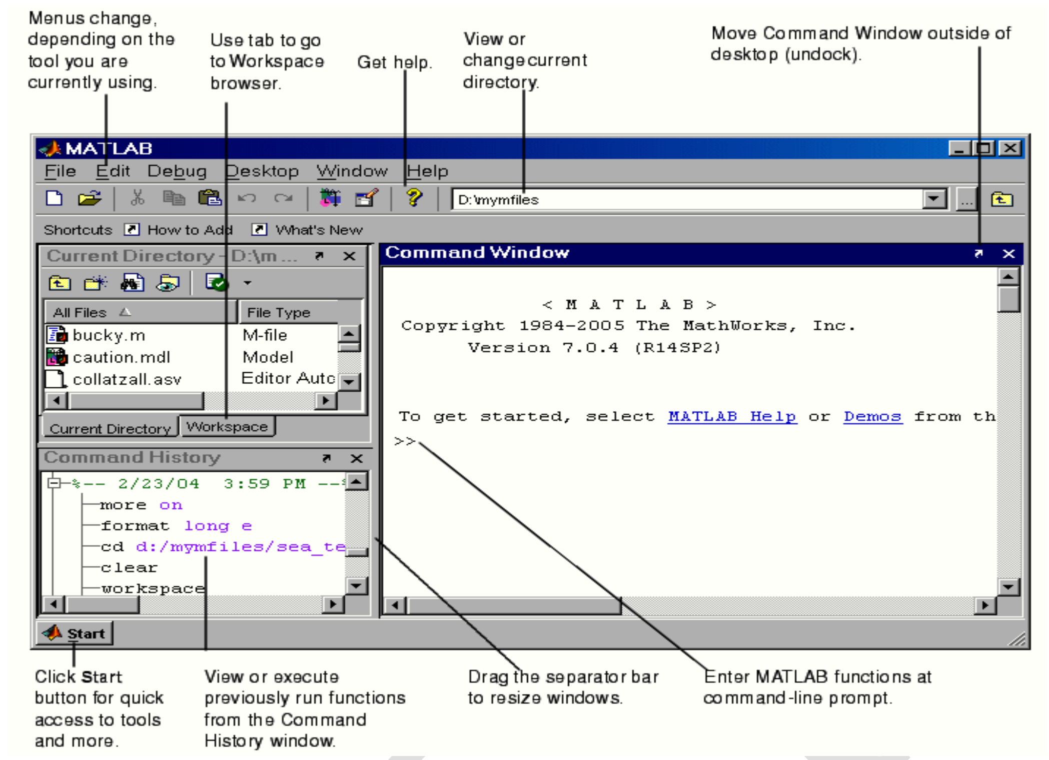

MATLAB®
The Language of Technical Computing
Copyright 1984 - 2008, The MathWorks Inc.
Protected by U.S. patents. See www.mathworks.com/patents
The MathWorks

Starting MATLAB:

After logging into your account, you can enter MATLAB by double-clicking on the MATLAB shortcut *icon* (MATLAB 7.0.4) on your Windows desktop. When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains *other* windows. The major tools within or accessible from the desktop are:

- The Command Window
- The Command History
- The Workspace
- The Current Directory
- The Help Browser

- The Start button


Menus change, depending on the tool you are currently using.
Use tab to go to Workspace browser.
Get help.
View or change current directory.
Move Command Window outside of desktop (undock).
MATLAB
File Edit Debug Desktop Window Help
D:\mymfiles
Shortcuts How to Add What's New
Current Directory - D:\m ...
All Files
File Type
bucky.m M-file
caution.mdl Model
collatzall.asv Editor Auto
Current Directory
Workspace
Command History
Command Window
Start
Click Start button for quick access to tools and more.
View or execute previously run functions from the Command History window.
Drag the separator bar to resize windows.
Enter MATLAB functions at command-line prompt.

```
%-- 2/23/04  3:59 PM --
  more on
  format long e
  cd d:/mymfiles/sea_te
  clear
  workspace
```

```
                 < M A T L A B >
     Copyright 1984-2005 The MathWorks, Inc.
          Version 7.0.4 (R14SP2)


  To get started, select MATLAB Help or Demos from th
>>
```

Figure 1.1: The graphical interface to the MATLAB workspace When MATLAB is started for the first time, the screen looks like the one that shown

in the Figure 1.1. This illustration also shows the default configuration of the MATLAB desktop. You can customize the arrangement of tools and documents to suit your needs.

Now, we are interested in doing some simple calculations. We will assume that you have sufficient understanding of your computer under which MATLAB is being run. You are now faced with the MATLAB desktop on your computer, which contains the prompt (>>) in the Command Window. Usually, there are 2 types of prompt:

>> For full version

EDU> for educational version

Note: To simplify the notation, we will use this prompt, >>, as a standard prompt sign,

Though our MATLAB version is for educational purpose.

Using MATLAB as a calculator

As an example of a simple interactive calculation, just type the expression you want to

evaluate. Let's start at the very beginning. For example, let's suppose you want to calculate the expression, 1 + 2 × 3. You type it at the prompt command (>>) as follows,

```
>> 1+2*3
ans = 7
```

You will have noticed that if you do not specify an output variable, MATLAB uses a

default variable ans, short for answer, to store the results of the current calculation. Note that the variable ans is created (or overwritten, if it is already existed). To avoid this, you may assign a value to a variable or output argument name. For example

```
>> x = 1+2*3

x =7
```

Will result in x being given the value 1 + 2*3=7. This variable name can always

be used to refer to the results of the previous computations. Therefore, computing 4 result in

```
>> 4*x

ans =28.0000
```

Before we conclude this minimum session, Table 1.1 gives the partial list of arithmetic

Operators.

Basic arithmetic operators

Symbol	Operation	Example
+	Addition	2 + 3
-	Subtraction	2 - 3
*	Multiplication	2*3
/	Division	2/3

3 Quitting MATLAB

To end your MATLAB session, type quit in the Command Window, or select File

MATLAB in the desktop main menu.

Getting started

After learning the minimum MATLAB session, we will now learn to use some additional operations.

1 Creating MATLAB variables

MATLAB variables are created with an assignment statement. The syntax of variable assignment is

Variable name = a value (or an expression)

For example,

```
>> x = expression
```

Where expression is a combination of numerical values, mathematical operators, variables, and function calls. On other words, expression can involve:

- manual entry
- built-in functions
- user-defined functions

Overwriting variable

Once a variable has been created, it can be reassigned. In addition, if you do not wish to see the intermediate results, you can suppress the numerical output by putting a semicolon (;) at the end of the line. Then the sequence of commands looks like this:

```
>> t = 5;

>> t = t+1

t =6
```

Error messages

If we enter an expression incorrectly, MATLAB will return an error message. For example, in the following, we left out the multiplication sign, *, in the following expression

```
>> x = 10;

>> 5x
```

Making corrections

To make corrections, we can, of course retype the expressions. But if the expression is

Lengthy, we make more mistakes by typing a second time. A previously typed command can be recalled with the up-arrow key When the command is displayed at the command prompt, it can be modified if needed and executed.

Controlling the hierarchy of operations or precedence

Let’s consider the previous arithmetic operation, but now we will include

example, 1 + 2×3 will become (1 + 2) ×3

```
>> (1+2)*3

ans =9
```

and, from previous example

```
>> 1+2*3
```

```
ans =7
```

By adding parentheses, these two expressions give di errant results: 9 and 7 The order in which MATLAB performs arithmetic operations is exactly that taught in high school algebra courses Exponentiations are done first, followed by multiplications and divisions, and finally by additions and subtractions. However, the standard order of precedence of arithmetic operations can be changed by inserting parentheses. For example, the result of 1 +2×3 is quite deferent than the similar expression with parentheses (1+2) ×3. The results are 7 and 9 respectively. Parentheses can always be used to overrule *priority* and their use is recommended in some complex expressions to avoid ambiguity.

Therefore, to make the evaluation of expressions unambiguous, MATLAB has established a series of rules. The order in which the arithmetic operations are evaluated is given in Table 1.2. MATLAB arithmetic operators obey the same precedence rules as those in Hierarchy of arithmetic operations Precedence Mathematical operations

First The contents of all parentheses are evaluated first, starting from the innermost parentheses and working outward Second All exponentials are evaluated, working from left to right Third All multiplications and divisions are evaluated, working

from left to right Fourth All additions and subtractions are evaluated, starting

from left to rightmost computer programs. For operators of equal precedence, evaluation is from left to right Now, consider another example:

In MATLAB, it becomes

```
>> 1/(2+3^2)+4/5*6/7

ans =0.7766.
```

or, if parentheses are missing,

```
>> 1/2+3^2+4/5*6/7

ans =10.1857.
```

So here what we get: two different results. Therefore, we want to emphasize the importance of precedence rule in order to avoid ambiguity.

Controlling the appearance of floating point number

MATLAB by default displays only 4 decimals in the result of the calculations, for example -163. 6667, as shown in above examples. However, MATLAB does numerical calculations in *double* precision, which is 15 digits. The command format controls how the results of computations are displayed. Here are some examples of the different formats together with the resulting outputs

```
>> Format short

>> x=-163.6667
```

If we want to see all 15 digits, we use the command format long

```
>> Format long
```

```
>> x= -1.636666666666667e+002
```

To return to the standard format, enter format short, or simply format there are several other formats. For more details, see the MATLAB documentation, or type help format Note - Up to now, we have let MATLAB repeat everything that we enter at the prompt (>>). Sometimes this is not quite useful, in particular when the output is pages en length. To prevent MATLAB from echoing what we type, simply enter a semicolon (;) at the end of the command. For example,

```
>> x=-163.6667;
```

and then ask about the value of x by typing,

```
>> x

        x =-163.6667
```

Managing the workspace

The contents of the workspace persist between the executions of separate commands. There-fore, it is possible for the results of one problem to have aneect on the next one. To avoid this possibility, it is a good idea to issue a clear command at the start of each new independent calculation

```
>> Clear
```

The command clear or clear all removes all variables from the workspace. This

frees up system memory. In order to display a list of the variables currently in the memory type

```
>> Who
```

While, whose will give more details which include size, space allocation, and class of the variables

Keeping track of your work session

It is possible to keep track of everything done during a MATLAB session with the

diary command.

```
>> Diary
```

Or give a name to a created file

```
>> Diary Filename
```

Where Filename could be any arbitrary name you choose

The function diary is useful if you want to save a complete MATLAB session.

They Save all input and output as they appear in the MATLAB window. When you want to stop the recording, enter diary off. If you want to start recording again, enter diary on. The file that is created is a simple text file. It can be opened by an editor or a word processing program and edited to remove extraneous material, or to add your comments. You can use the function type to view the diary file or you can edit in a text editor or print. This command is useful, for example in the process of preparing a homework or lab submission.

Entering multiple statements per line

It is possible to enter multiple statements per line. Use commas (,) or semicolons (;) to

Enter more than one statement at once. Commas (,) allow multiple statements per line

Without suppressing output

```
>> a=7; b=cos(a), c=cash (a)
```

```
b =0.6570
```

```
c =548.3170
```

Miscellaneous commands

Here are few additional useful commands:

•To clear the Command Window, type clc

•To abort a MATLAB computation, type ctrl-c

• To continue a line, type . . .

Getting help

To view the online documentation, select MATLAB Help from Help menu or MATLAB Help directly in the Command Window. The preferred method is to use the *Help Browser*. The Help Browser can be started by selecting the? Icon from the desktop toolbar. On the other hand, information about any command is available by typing

```
>> help Command
```

Another way to get help is to use the look for command. The look for commandeers

from the help command. The help command searches for an exact function name match, while the look for command searches the quick summary information in each function for a match. For example, suppose that we were looking for a function to take

The inverse of a matrix. Since MATLAB does not have a function named inverse, the command help inverse will produce nothing. On the other hand, the command look for inverse will produce detailed information, which includes the function of interest, inv

```
>> look for inverse
```

Note - At this particular time of our study, it is important to emphasize one main point.

Because MATLAB is a huge program; it is impossible to cover all the details

of each function one by one. However, we will give you information how to get help. Here are some examples

• Use on-line help to request info on a specific function

```
>> help sqrt
```

• In the current version (MATLAB version 7), the doc function opens the on-line version of the help manual. This is very helpful for more complex commands

```
>> Doc plot
```

• Use look for to find functions by keywords. The general form is

```
>>look for Function Name
```

Common Procedure to all Programs in MATLAB

1. Click on the MATLAB Icon on the desktop.

2. MATLAB window open.

3. Click on the ‘FILE’ Menu on menu bar.

4. Click on NEW M-File from the file Menu.

5. An editor window open, start typing commands.

6. Now SAVE the file in directory.

7. Then Click on DEBUG from Menu bar and Click Run.

1.GENERATION OF BASIC SIGNALS USING MATLAB

AIM : To generate basic signals like unit impulse, unit step, unit ramp signal and Exponential signals.

Objective: To generate basic signals like unit impulse, unit step, unit ramp signal and Exponential signals using MATlab.

Requirements : Computer with MATLAB software

(a). **Program for the generation of UNIT impulse signal**

```
clc;  close all;  clear all;

t=-2:1:2;

y=[zeros(1,2),ones(1,1),zeros(1,2)] figure(1)

subplot(2,2,1); stem(t,y);

title('unit impulse');
```

(b). **Program for the generation of UNIT step signal**

```
clc;  close all;  clear all;

n=input('enter the n value');

t=0:1:n-1;

y=ones(1,n);

figure(2)

subplot(2,2,2);

 stem(t,y);

title('unit step');
```

(c).**Program for the generation of unit RAMP   signal**

```
clc;  close all;  clear all;

n=input('enter the n value');

t=0:n;

y=ones(1,n);

figure(3)

subplot(2,2,3);

 stem(t,t);

title('unit ramp');
```

(d).**Program for the generation of Exponential signal**

```
clc;  close all;  clear all;

n=input('the length of i/p sequency');

t=0:n

a=input('enter the a value');

y=exp(a*t);       figure(4)

subplot(2,2,4);

stem(t,y);

xlabel('x-axis');   ylabel('y-axis'); title('unit exponential');
```

2. AIM : To Generate continuous time sinusoidal signal, Discrete time cosine signal.

Requirements : Computer with MATLAB software

```
%   Program for Continuous time signal

clc;

close all;

clear all;

t=0:.01:pi;

y= sin(2*pi*t);

subplot(4,1,1);

plot(t,y);

ylabel('amp...');xlabel('(a)n...');title('sin signal')


%Program for Discrete time cosine signal :

t=0:.03:pi/3;

y= cos(2*pi*t);

subplot(4,1,2);

stem(t,y);

xlabel('a(n)');ylabel('amplitude');title('cosinusoidal');
```

Results: Continuous time sinusoidal signal, discrete time cosine signal and sum of sinusoidal signal is designed.

Outcomes: After finishing this experiment the students are able to

1. Generate elementary signals/ waveforms.
2. Perform arithmetic operations on signals.

\

VIVA QUESTIONS:

1. Define impulse signal

2. Define ramp signal

3. Define unit step signal

4. Define exponent ional signal

5. Define sinusoidal signal

6. Define C.T.S

7. Define D.T.S.

8. Compare C.T.S & D.T.S

9. Define Stem, Plot, Plot3,fplot, ezplot, linspace, flyplr, grid,mesh and legend

10. Draw the C.T.S & D.T.S diagrams

FREQUENCY RESPONSE:

3. To find frequency response of a given system given in (Transfer Function/ Differential equation form).

AIM:- To write a MATLAB program to evaluate the Frequency response of the system .

Objective: To write a MATLAB program to evaluate the Frequency response of the system .

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

THEORY:-

The Difference equation is given as

$$y[n]-0.25y[n-1]+0.45y[n-2]=1.55x[n]+1.95x[n-1]+ 2.15x[n]$$

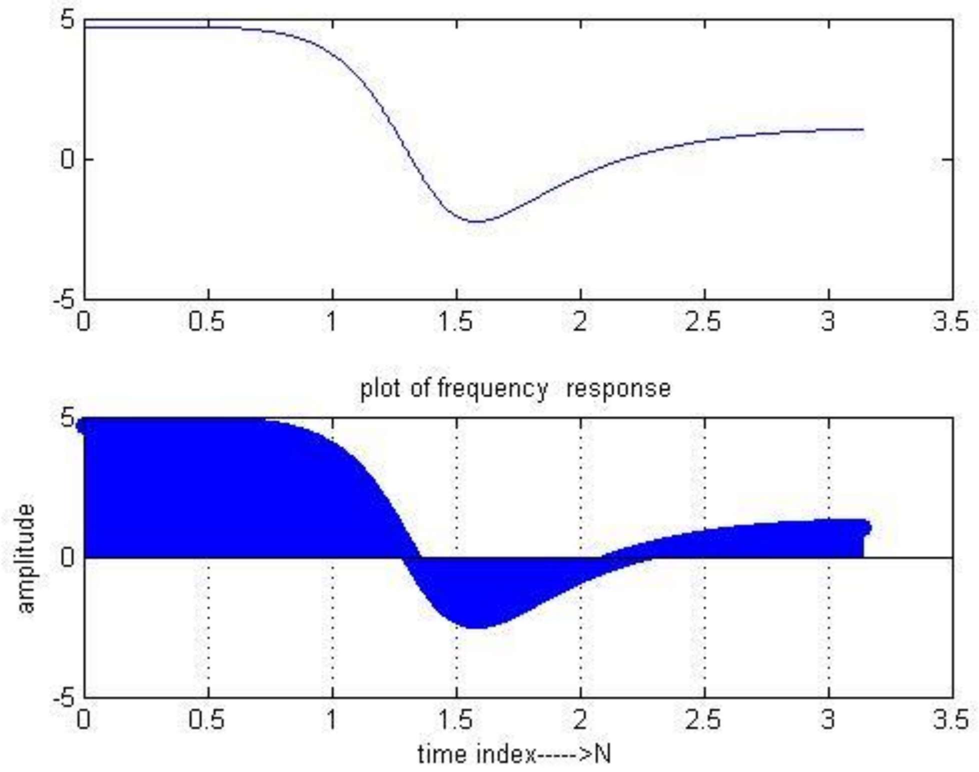
The frequency response is a representation of the system's response to sinusoidal inputs at varying frequencies. The output of a linear system to a sinusoidal input is a sinusoid of the same frequency but with a different magnitude and phase. Any linear system can be *completely* described by how it changes the amplitude and phase of cosine waves passing through it. This information is called the system's frequency response. Since both the impulse response and the frequency response contain complete information about the system, there must be a one-to-one correspondence between the two. Given one, you can calculate the other. The relationship between the impulse response and the frequency response is one of the foundations of signal processing: A system's frequency response is the Fourier Transform of its impulse response Since *h* [ ] is the common symbol for the impulse response, *H* [ ] is used for the frequency response.

PROGRAM:

```
clc;

clear all;

close all;

% Difference equation of a second order system

% y[n]-0.25y[n-1]+0.45y[n-2]=1.55x[n]+1.95x[n-1]+ 2.15x[n-2]

b=input('enter the coefficients of x(n),x(n-1)-----');

a=input('enter the coefficients of y(n),y(n-1)----');

N=input('enter the number of samples of frequency  response ');

[h,t]=freqz(b,a,N);

subplot(2,1,1);
% figure(1);
plot(t,h);
subplot(2,1,2);
% figure(2);
stem(t,h);
title('plot of frequency  response');
ylabel('amplitude');
 xlabel('time index----->N');
disp(h);
grid on;
```

OUTPUT:

```
enter the coefficients of x(n),x(n-1)-----[1.55 1.95 2.15]
enter the coefficients of y(n),y(n-1)----[1 -.25 .45]
enter the number of samples of frequency  response 1500
```



RESULT: The frequency response of given Differential equation is obtained. Hence the theory and practical value are proved.

Objective: After finishing this experiment the students are able to
calculate and plot a frequency response of given Differential equation is obtained. Hence the theory and practical value are proved.

VIVA QUESTIONS:

1. Which built in function is used to solve a given difference equation?
2. What is frequency response? Give equation for first order system and second order system?
3. What is an LTI system?
4. What is steady state response?
5. Suppose we have a system with transfer function H(z) = 1 / ((z – 1.1)*(z – 0.9)). Is the system stable or unstable?
6. What is Auto Regressive Model? How is the order of auto regressive model is decided?

2. IMPULSE RESPONSE OF A GIVEN SYSTEM

AIM:- To write a MATLAB program to evaluate the impulse response of the system .

OBJECTIVE:- To write a MATLAB program to evaluate the impulse response of the system using MATlab.

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

The Difference equation is given as

$$y(n) = x(n)+0.5x(n-1)+0.85x(n-2)+y(n-1)+y(n-2)$$

THEORY:-

LTI Discrete time system is completely specified by its impulse response i.e. knowing the impulse response we can compute the output of the system to any arbitrary input. Let h[n] denotes the impulse response of the LTI discrete time systems. Since discrete time system is time invariant, its response to [n-1] will be h[n-1] .Likewise the response to [n+2] , [n-4] and [n-6] will be h[n+2], h[n-4] and h[n-6] .

From the above result arbitrary input sequence x[n] can be expressed as a weighted linear combination of delayed and advanced unit sample in the form k=+

X[n] = x[k][n-k]

k=-

where weight x[k] on the right hand side denotes specifically the k th sample value of the sequence. The response of the LTI discrete time system to the sequence

x[k] [n-k] will be x[k] h [n-k].

As a result, the response y[n] of the discrete time system to x[n] will be given by

k=+

y[n] = x[k] h [n-k] …………..(1)

k=-

Which can be alternately written as

k=+

y[n] = x[n-k] h [k]…………(2)

k=-

The above equation (1) and (2) is called the convolution sum of the sequences x[n]

and h[n] and represented compactly as y[n]=x[n] * h[n] Where the notation * denotes the convolution sum.

Structure for Realization of Linear Time Invariant systems:

Let us consider the first order system Y(n)=-a 1y(n-1)+b0 x(n) +b1 x(n-1)

This realization uses separate delays(memory) for both the input and output samples and it is called as Direct form one structure.

A close approximation reveals that the two delay elements contain the same input w(n) and hence the same output w(n-1).consequently these two elements can be merged into one delay. In contrast to the direct form I structure , this new realization requires only one delay for auxiliary quantity w(n) ,and it is more efficient in terms of memory requirements. It is called the direct form II structure and it is used extensively.

PROGRAM:-

```
clc;
clear all;
close all;
% Difference equation of a second order system
% y(n) = x(n)+0.5x(n-1)+0.85x(n-2)+y(n-1)+y(n-2)
b=input('enter the coefficients of x(n),x(n-1)-----');
a=input('enter the coefficients of y(n),y(n-1)----');
N=input('enter the number of samples of imp response ');
[h,t]=impz(b,a,N);
subplot(2,1,1);
% figure(1);
plot(t,h);
title('plot of impulse response');
ylabel('amplitude');
 xlabel('time index----->N');
subplot(2,1,2);
% figure(2);
stem(t,h);
title('plot of impulse response');
ylabel('amplitude');
 xlabel('time index----->N');
 disp(h);
grid on;
```

Output

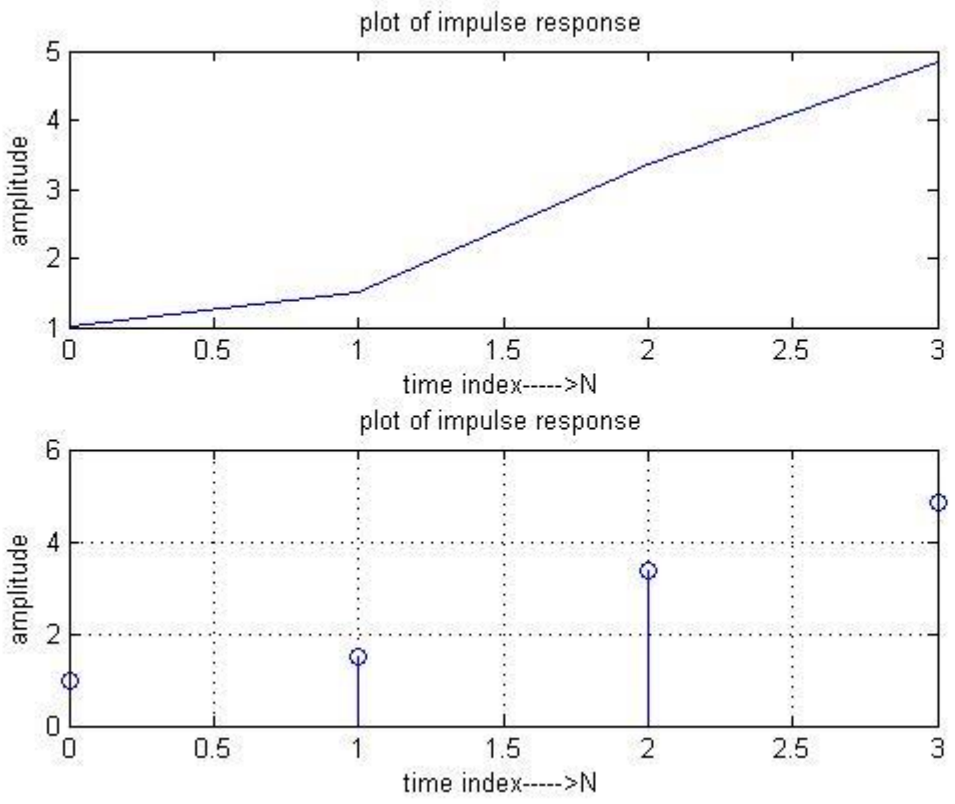
```
enter the coefficients of x(n),x(n-1)-----[1 0.5 0.85] enter the coefficients of y(n),y(n-1)-----[1 -1 -1]
enter the number of samples of imp respons 4

1.0000
```

1.5000

3.3500

4.8500

GRAPH:-



CALCULATIONS:-

y(n) = x(n)+0.5x(n-1)+0.85x(n-2)+y(n-1)+y(n-2)

y(n) - y(n-1) - y(n-2) = x(n) + 0.5x(n-1) + 0.85x(n-2) Taking Z transform on both sides,

Y(Z) - Z-1 Y(Z)- Z-2 Y(Z) = X(Z) + 0.5 Z-1 X(Z) + 0.85 Z-2 X(Z) Y(Z)[1 - Z-1 - Z-2] = X(Z)[1 + 0.5 Z-1 + 0.85 Z-2]

But, H(Z) = Y(Z)/X(Z)

= [1 + 0.5 Z-1 + 0.85 Z-2]/ [1 - Z-1 - Z-2] By dividing we get

H(Z) = 1 + 1.5 Z-1 + 3.35 Z-2 + 4.85 Z-3

h(n) = [1 1.5 3.35 4.85]

RESULT: The impulse response of given Differential equation is obtained. Hence the theory and practical value are proved

Outcome: The students are able to write the MATlab code to find the impulse response of given Differential equation is obtained and the theory and practical value are proved

Discussion /Viva questions:-

1) Differentiate between linear and circular convolution.
2) Determine the unit step response of the linear time invariant system with impulse response h(n)=a ^{n} u(n) a<1&-a<1
3) Determine the range of values of the parameter a for which linear time invariant system with impulse response h(n)=a ^{n} u(n) is stable.
4) Consider the special case of a finite duration sequence given as X(n)={2 4 0 3}, resolve the sequence x(n) into a sum of weighted sequences.
5) . Describe impulse response of a function?
6) . Where to use command filter or impz, and what is the difference between these two?

3. TO FIND DFT / IDFT OF GIVEN DT SIGNAL

AIM: To find the DFT / IDFT of given signal.

Objective: **T**o wite the MATlab code to find the DFT / IDFT of given signal.

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

THEORY:

Discrete Time Fourier Transform:

The discrete-time Fourier transform (DTFT) X(ejω) of a sequence x[n] is defined

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

In general *X*(*ejω*) is a complex function of the real variable *ω* and can be written as

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

where *Xre*(*ejω*) and *Xim*(*ejω*) are, respectively, the real and imaginary parts of *X*(*ejω*), and are real functions of *ω*. *X*(*ejω*) can alternately be expressed in the form

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

$$\theta(\omega) = \arg\{X(e^{j\omega})\}.$$

The quantity |X(ejω)| is called the magnitude function and the quantity θ(ω) is called the phase function

In many applications, the Fourier transform is called the Fourier spectrum and, likewise, |X(ejω)| and θ(ω) are referred to as the magnitude spectrum and phase spectrum, respectively.

The DTFT X(ejω) is a periodic continuous function in ω with a period 2π. The DTFT satisfies a number of useful properties that are often uitilized in a number of applications.

MATLAB COMMANDS:

For complex Z, the magnitude R and phase angle theta are given by:

R = abs (Z)

Theta = angle (Z)

Y = fft(X) returns the discrete Fourier transform of vector X, computed with a fast Fourier transform (FFT) algorithm.

Y = fft(X)

Y = fft(X,n) returns the n-point FFT.

Y = fft(X, n)

Compute the discrete Fourier transform of the following function analytically and Then plot the magnitude and phase:

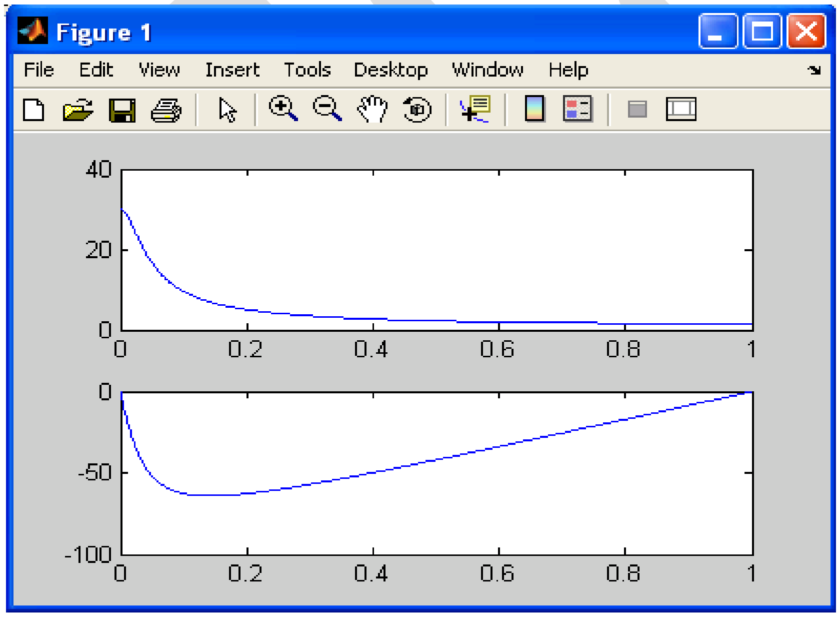
$$x(n) = 3(0.9)^n u(n)$$

Its DTFT is given as:

$$\frac{3}{1 - 0.9\, e^{-jW}}$$

MATLAB Code:

```
w = [0:500]*pi/500;

z = exp(-j*w);

x = 3*(1-0.9*z).^(-1);

a = abs(x);

b = angle(x)*180/pi;

subplot(2,1,1);

plot(w/pi,a);

subplot(2,1,2);

plot(w/pi,b);
```



Evaluate the DTFT of the given coefficients.

num=[2 1]

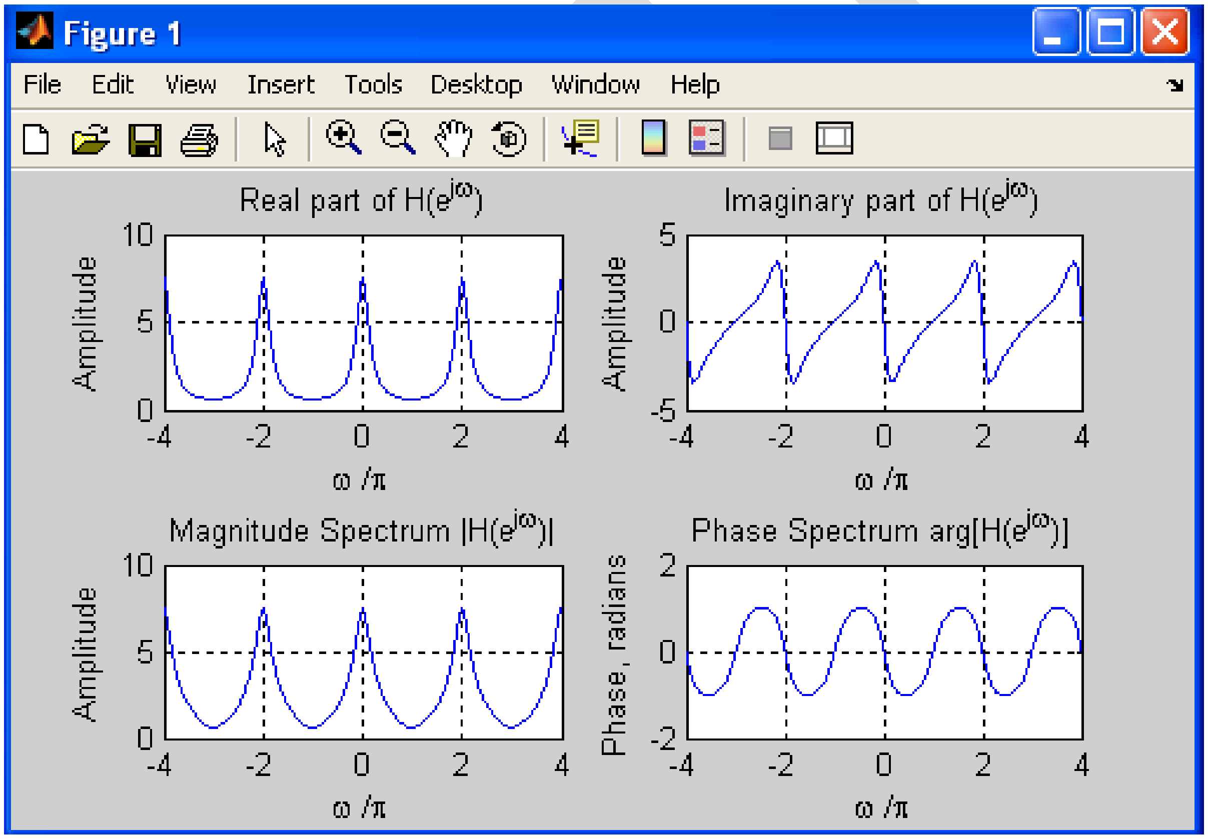
den=[1 –0.6]

- Plot real and imaginary parts of Fourier spectrum.
- Also plot the magnitude and phase spectrum.

MATLAB CODE:

```
% Evaluation of the DTFT

clc;

% Compute the frequency samples of the DTFT

w = -4*pi:8*pi/511:4*pi;

num = [2 1];

den = [1 -0.6];

h = freqz(num, den, w);

% Plot the DTFT

subplot(2,2,1)

plot(w/pi,real(h));

grid on;

title('Real part of H(e^{j\omega})')

xlabel('\omega /\pi');

ylabel('Amplitude');

subplot(2,2,2)

plot(w/pi,imag(h));

grid on;

title('Imaginary part of H(e^{j\omega})')

xlabel('\omega /\pi');

ylabel('Amplitude');

subplot(2,2,3)
```

```
plot(w/pi,abs(h));

grid on;

title('Magnitude Spectrum |H(e^{j\omega})|')

xlabel('\omega /\pi');

ylabel('Amplitude');

subplot(2,2,4)

plot(w/pi,angle(h));

grid on;

title('Phase Spectrum arg[H(e^{j\omega})]')

xlabel('\omega /\pi');

ylabel('Phase, radians');
```



PROGRAM:

Computation of N point DFT of a given sequence and to plot magnitude and phase spectrum.

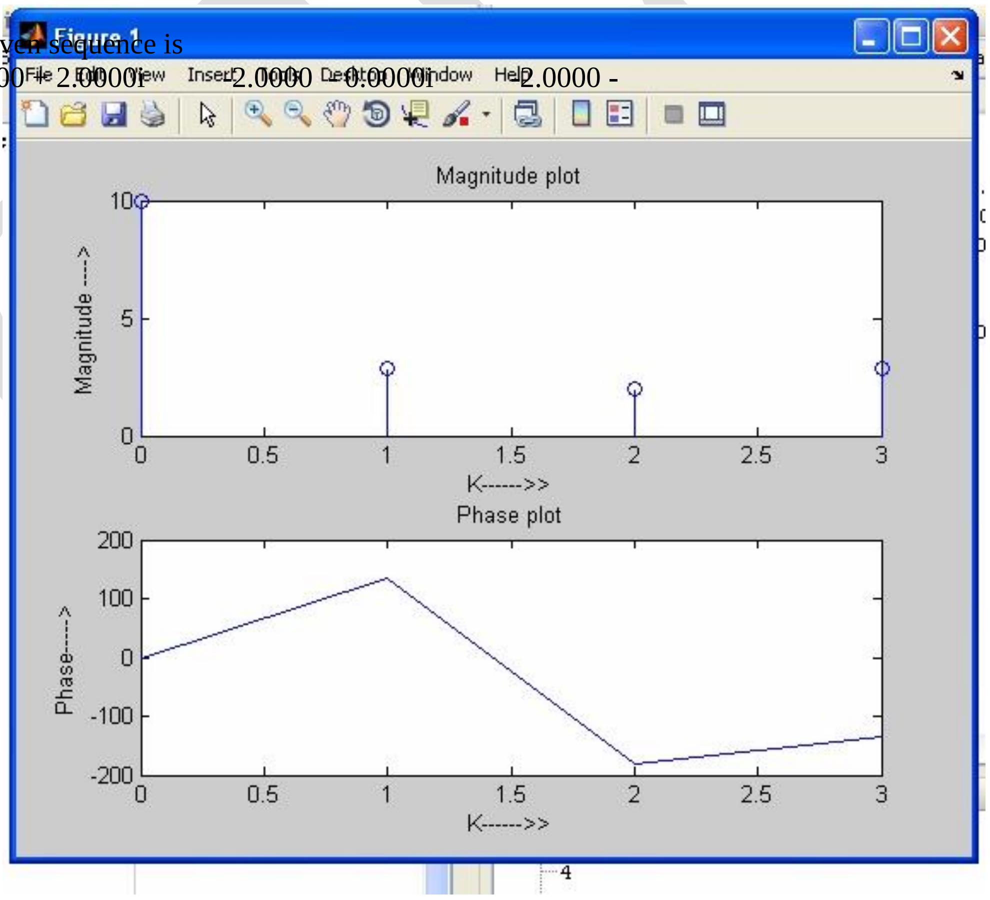
```
N = input('Enter the the value of N(Value of N in N-Point DFT)');
x = input('Enter the sequence for which DFT is to be calculated');
n=[0:1:N-1];
k=[0:1:N-1];
WN=exp(-1j*2*pi/N);      % twiddle factor
nk=n'*k;
WNnk=WN.^nk;
Xk=x*WNnk;
MagX=abs(Xk) % Magnitude of calculated DFT
PhaseX=angle(Xk)*180/pi % Phase of the calculated DFT figure(1);
subplot(2,1,1);
plot(k,MagX);
subplot(2,1,2);
plot(k,PhaseX);
```

-------------*******--------------

OUTPUT

```
Enter the the value of N(Value of N in N-Point DFT)4 Enter the sequence for which DFT is to be
calculated [1 2 3 4]

MagX =          10.00002.8284  2.0000  2.8284



PhaseX =        0         135.0000 -180.0000 -135.0000

DFT of the given sequence is
10.0000-2.0000 + 2.0000i       -2.0000 - 0.0000i       -2.0000 -
2.0000i
```

Output:



RESULT: The DFT of given sequence is obtained . Hence the theory and practical value are proved.

Outcomes:

After finishing this experiment the students are able to :

1. Calculate DFT /IDFT of any given signal.
2. Plot DFT / IDFT of given DT signal.

VIVA QUESTIONS:

1. How to calculate output of DFT using MATLAB?
2. Where DFT is used?
3. What is the difference between DFT and IDFT?
4. How to compute maximum length N for a circular convolution using DFT and IDFT.(what is command).
5. Explain the function of twiddle factor?
6. Give the practical application DFT & IDFT?
7. Explain the role of DFT & IDFT when the signal converted from the time domain to frequency domain?
8. Differentiate between time variant and time invariant system. If x 1(n)={1,2,3,4} and x 2(n)={1,2,3} Find the convolution using tabular representation.
9. Draw all elementary standard discrete time signals.
10. Differentiate between causal and Non causal system.
11. If x 1(n)={1,2,3,4} and x 2(n)={5,6,7,8} Find the circular representation for the above sequences.
12. How can you compute Fourier transform form Z-transform ?

4. IMPLEMENTATION OF FFT OF GIVEN SEQUENCE

AIM: To implementation of Fast Fourier Transform.

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

THEORY: The Fast Fourier Transform is useful to map the time-domain sequence into a continuous function of a frequency variable. The FFT of a sequence {x(n)} of length N is given by a complex-valued sequence X(k).

$$X(k) = \sum_{k=0}^{M} x(n) \quad e^{-j2\pi \frac{nk}{n}} \;;0 < k < N -1$$

The above equation is the mathematical representation of the DFT. As the number of computations involved in transforming a N point time domain signal into its corresponding frequency domain signal was found to be N^2 complex multiplications, an alternative algorithm involving lesser number of computations is opted.

When the sequence x(n) is divided into 2 sequences and the DFT performed separately, the resulting

$$x(k) = \sum_{n=0}^{\frac{N^2}{2-1}} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N^2}{2-1}} \quad x(2n+1) \quad W_N^{(2n+1)k} \quad (6)$$

Consider x(2n) be the even sample sequences and x(2n+1) be the odd sample sequence derived form x(n).

$$\sum_{n=0}^{\frac{N^2}{2-1}} x(2n) \quad W_N^{2nk}$$

$$(\text{N/2})^2\text{multiplication's} \sum_{n=0}^{\frac{N^2}{2-1}} x(2n+1) \quad W_N^{(2n+1)k} \qquad (8)$$

an other $(N/2)^2$ multiplication's finally resulting in $(N/2)^2 + (N/2)^2$

$$= \frac{N^2}{4} + \frac{N^2}{4} = \frac{N^2}{2} \textit{ Computations}$$

Further solving Eg. (2)

$$x(k) = \sum_{n=0}^{\frac{N^2}{2-1}} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N}{2-1}} \quad x(2n+1) \quad W_N^{(2nk)} \underset{N}{\overset{k}{W}} \; (9)$$

$$= \sum_{n=0}^{\frac{N}{2-1}} x(2n) \quad W_N^{2nk} + \underset{N}{\overset{k}{W}} \sum_{n=0}^{\frac{N}{2-1}} \quad x(2n+1) \quad W_N^{(2nk)} \; (10)$$

Dividing the sequence x(2n) into further 2 odd and even sequences would reduce the computations.

$W_N \rightarrow$ is the twiddle factor

$$= e^{\frac{-j2\pi}{n}}$$

$$W_N^{nk} = e^{\left(\frac{-j2\pi}{n}\right)nk}$$

$$W_N^{\left(K+\frac{N}{2}\right)} = W_N \quad W_N^{\left(K+\frac{N}{2}\right)} \qquad (11)$$

$$= e^{\frac{-j2\pi}{n}k} \quad e^{\frac{-j2\pi}{n} \frac{n}{2}}$$

$$= W_N^k \quad e^{\frac{-j2\pi}{n}k}$$

$$= W_N^k \quad (\cos\pi - j\sin\pi)$$

$$= W_N^{\left(K+\frac{N}{2}\right)} = W_N^k(-1)$$

$$= W_N^{\left(K+\frac{N}{2}\right)} = W_N^k \qquad (12)$$

Employing this equation, we deduce

$$x(k) = \sum_{n=0}^{\frac{N^2}{2-1}} x(2n) \quad W_N^{2nk} + \sum_{n=0}^{\frac{N}{2-1}} \quad x(2n+1) \quad W_N^{(2nk)} \quad (13)$$

$$x(k+\frac{N}{2}) = \sum_{n=0}^{\frac{N}{2-1}} x(2n) \quad W_N^{2nk} - W\sum_{N}^{K} \quad x(2n+1)^{\frac{N}{2-1}} \quad W_N^{(2nk)} \quad (14)$$

The time burden created by this large number of computations limits the usefulness of DFT in many applications. Tremendous efforts devoted to develop more efficient ways of computing DFT resulted in the above explained Fast Fourier Transform algorithm. This mathematical shortcut reduces the number of calculations the DFT requires drastically. The above mentioned radix-2 decimation in time FFT is employed for domain transformation.

Dividing the DFT into smaller DFTs is the basis of the FFT. A radix-2 FFT divides the DFT into two smaller DFTs, each of which is divided into smaller DFTs and so on, resulting in a combination of two-point DFTs. The Decimation -In-Time (DIT) FFT divides the input (time) sequence into two groups, one of even samples and the other of odd samples. N/2 point DFT are performed on the these sub-sequences and their outputs are combined to form the N point DFT.

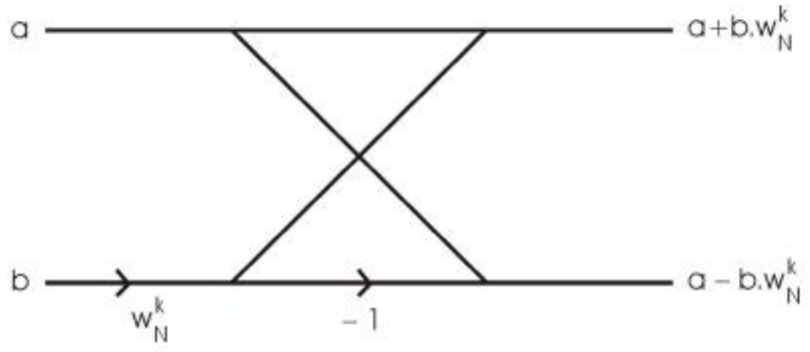
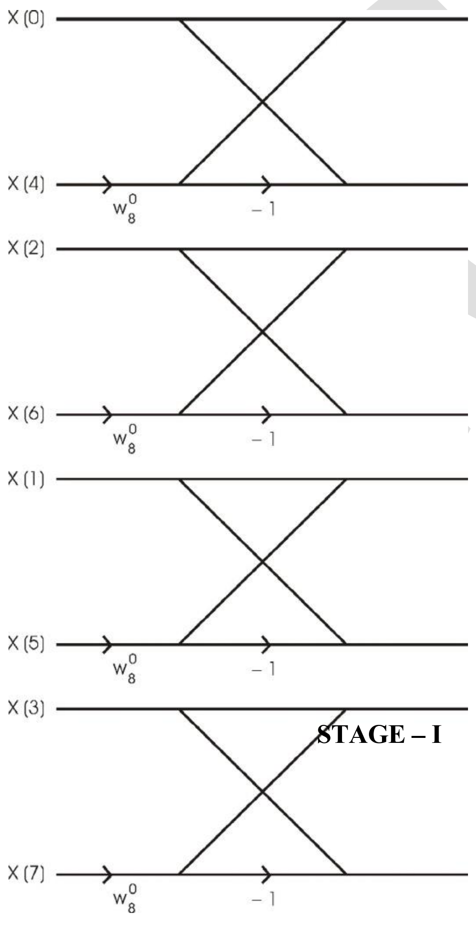
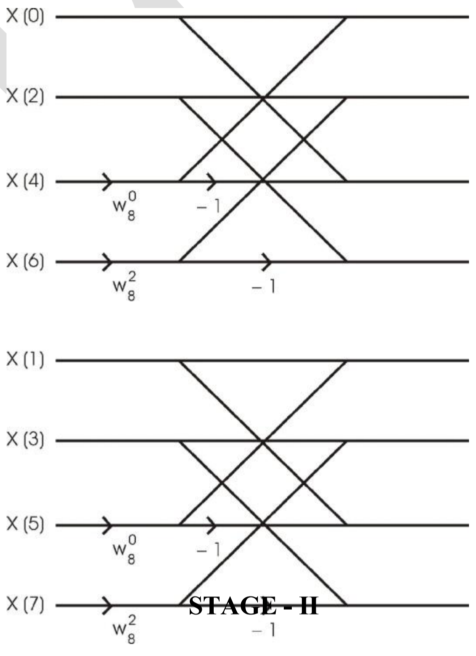


FIG. 3A.1

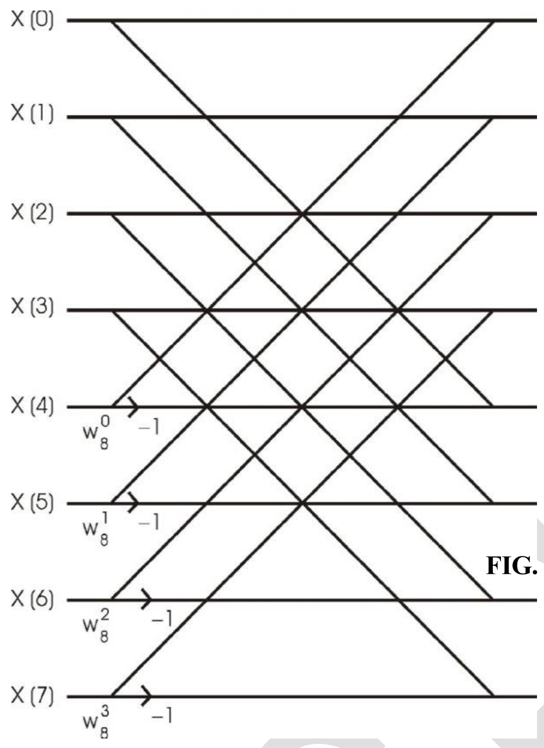
The above shown mathematical representation forms the basis of N point FFT and is called the **Butterfly Structure**.



STAGE – I



STAGE - II



STAGE - III

FIG. 3A.2 - 8 POINT DIT

Fast Fourier Transform:

The Fast Fourier Transform (FFT) is just a computationally fast way to calculate the DFT.

<u>Question No 03</u>: **USING INBUILT FUNCTION**

Determine the Fourier transform of the following sequence. Use the FFT (Fast Fourier Transform) function.

x (n) = {4 6 2 1 7 4 8}

MATLAB Code: for defined sequence

```
n = 0:6;

x = [4 6 2 1 7 4 8];

a = fft(x);
```

```
mag = abs(a);
pha = angle(a);
subplot(2,1,1);
plot(mag);
grid on
title('Magnitude Response');
subplot(2,1,2);
plot(pha);
grid on
title('phase Response');
```

IMPLEMENTATION OF FFT OF GIVEN SEQUENCE IN MATLAB

N-POINT FFT WITHOUT USING INBUILT FUNCTION

PROGRAM:

```
clear all;
close all;
clc;
x=input('Enter the sequence x[n]= ');
N=input('Enter the value N point= ');
L=length(x);
x_n=[x,zeros(1,N-L)];
for i=1:N
   for j=1:N
      temp=-2*pi*(i-1)*(j-1)/N;
      DFT_mat(i,j)=exp(complex(0,temp));
   end
end
X_k=DFT_mat*x_n';
disp('N point DFT is X[k] = ');
disp(X_k);

mag=abs(X_k);
phase=angle(X_k)*180/pi;
subplot(2,1,1);
stem(mag);
xlabel('frequency index k');
```

```
ylabel('Magnitude of X[k]');
axis([0 N+1 -2 max(mag)+2]);
subplot(2,1,2);
stem(phase);
xlabel('frequency index k');
ylabel('Phase of X[k]');
axis([0 N+1 -180 180]);
```

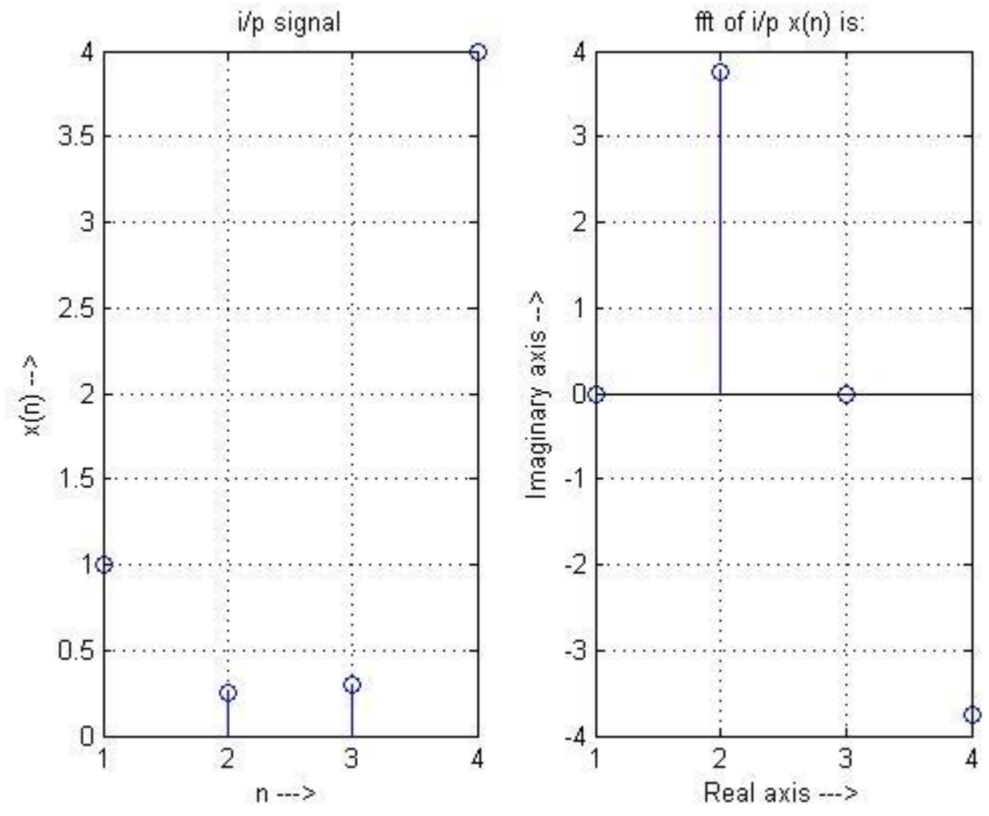
PROGRAM: for user defined sequence

```
%fast fourier transform
clc;
clear all;
close all;
tic;
x=input('enter the sequence');
n=input('enter the length of fft'); %compute fft
disp('fourier transformed signal');
 X=fft(x,n)
subplot(1,2,1);stem(x); title('i/p signal');
xlabel('n --->');
ylabel('x(n) -->');grid;
subplot(1,2,2);stem(X);
title('fft of i/p x(n) is:');
xlabel('Real axis --->');
ylabel('Imaginary axis -->');grid;
```

OUTPUT:-

```
enter the sequence[1 .25 .3 4]
enter the length of fft4
fourier transformed signal

X =

   5.5500          0.7000 + 3.7500i  -2.9500          0.7000 - 3.7500i
```

GRAPH FOR OUTPUT:-



RESULT: The Fast Fourier Transform of given sequence is obtained. Hence the theory and practical value are proved.

Outcomes:

After finishing this experiment the students are able to:

1. Able to Implement FFT of given sequence
2. Identify the reduction of computations using FFT.

VIVA QUESTIONS:

1. FFT is in complex domain how to use it in real life signals optimally?
2. What is the difference between FFT and IFFT?
3. Explain using convolution the effects of taking an FFT of a sample with no windowing
4. What is the need of FFT ?
5. What's the difference between FFT and DFT?
6. Why do we need Fourier transform in DSP?
7. Give any practical application of fft in daily life?
8. What is the importance of fft in OFDMA technology?
9. In STB DVB how many point fft is currently using?

10. Give FFT & IFFT formulae and calculate for any sequence?

11. What is "decimation-in-time" versus "decimation-in-frequency"?

12. What is "bit reversal"?

13. How does the FFT work?

5. Power Spectral Density

AIM: To verify Power Spectral Density

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

THEORY: The power spectral density(P.S.D) is a measurement of the energy at various frequencies

In the previous section we saw how to unwrap the FFT and get back the sine and cosine coefficients. Usually we only care how much information is contained at a particular frequency and we don't really care whether it is part of the sine or cosine series. Therefore, we are interested in the absolute value of the FFT coefficients. The absolute value will provide you with the total amount of information contained at a given frequency, the square of the absolute value is considered the power of the signal. Remember that the absolute value of the Fourier coefficients are the distance of the complex number from the origin. To get the power in the signal at each frequency (commonly called the power spectrum) you can try the following commands.

```
>> N = 8;            %% number of points
>> t = [0:N-1]'/N;        %% define time
>> f = sin(2*pi*t);       %%define function
>> p = abs(fft(f))/(N/2); %% absolute value of the fft
>> p = p(1:N/2).^2        %% take the positve frequency half, only
```

This set of commands will return something much easier to understand, you should get 1 at a frequency of 1 and zeros everywhere else. Try substituting cos for sin in the above commands, you should get the same result. Now try making >>f = sin(2*pi*t) + cos(2*pi*t). This change should result in twice the power contained at a frequency of 1.

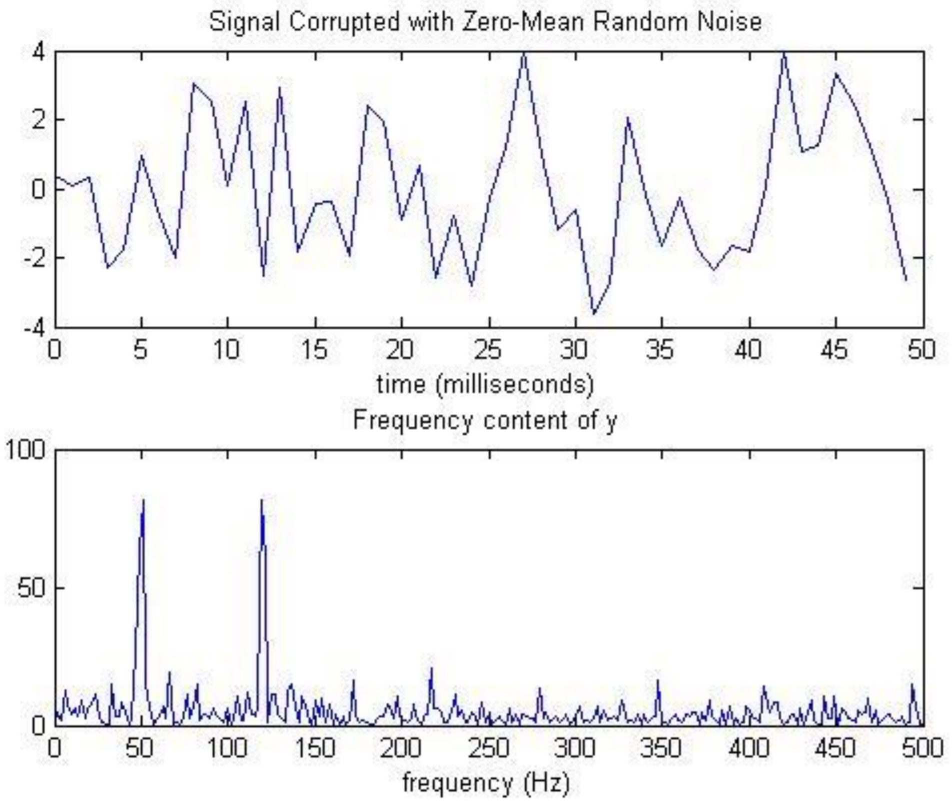
Thus far we have looked at data that is defined on the time interval of 1 second, therefore we could interpret the location in the number list as the frequency. If the data is taken over an arbitrary time interval we need to map the index into the Fourier series back to a frequency in Hertz. The following m-file script will create something that might look like data that we would obtain from a sensor. We will

.

PROGRAM:

```
%Power spectral density
t = 0:0.001:0.6;
x = sin(2*pi*50*t)+sin(2*pi*120*t);
y = x + 2*randn(size(t));
subplot(2,1,1);
% figure(1);
plot(1000*t(1:50),y(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('time (milliseconds)');
Y = fft(y,512);
%The power spectral density, a measurement of the energy at various frequencies, is:
Pyy = Y.* conj(Y) / 512;
f = 1000*(0:256)/512;
subplot(2,1,2);
% figure(2);
plot(f,Pyy(1:257))
title('Frequency content of y');
 xlabel('frequency (Hz)');
```

OUTPUT:-



RESULT: The power spectral density of given sequence is obtained. Hence the theory and practical value are proved.

Outcome: At the end of the experiment the students are able to find the power spectral density of given sequence is obtained.

VIVA QUESTION:

1. What do you mean by phase spectrum and magnitude spectrum/ give comparison?
2. How do you reduce spectral leakage?
3. What do you mean by spectral resolution?

EXPERIMENT # 6 & 7

AIM : Design of FIR filters of Low pass and high pass filter using Matlab commands

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

.

DESCRIPTION: Digital filters refers to the hard ware and software implementation of the mathematical algorithm which accepts a digital signal as input and produces another digital signal as output whose wave shape, amplitude and phase response has been modified in a specified manner. Digital filter play very important role in DSP. Compare with analog filters they are preferred in number of application due to following advantages.

- Truly linear phase response
- Better frequency response
- Filtered and unfiltered data remains saved for further use.

There are two type of digital filters.

1. FIR (finite impulse response) filter
2. IIR (infinite impulse response) filter

Description Of The Commands Used In FIR Filter Design

FIR1: FIR filters design using the window method.

B = **FIR1**(N,Wn) designs an N'th order low

pass 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. The normalized gain of the filter

at Wn is -6 dB.

B = **FIR1**(N,Wn,'high') designs an N'th order highpass filter.

You can also use B = **FIR1**(N,Wn,'low') to design a lowpass filter.

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,'stop') is a bandstop filter if Wn = [W1 W2]. You can also specify If Wn is a multi-element vector, Wn = [W1 W2 W3 W4 W5 ... WN], FIR1 returns an order N multiband filter with bands 0 < W < W1, W1 < W < W2, ..., WN < W < 1.

B = **FIR1**(N,Wn,'DC-1') makes the first band a passband.

B = **FIR1**(N,Wn,'DC-0') makes the first band a stopband.

By default FIR1 uses a Hamming window. Other available windows, including Boxcar, Hann, Bartlett, Blackman, Kaiser and Chebwin can be specified with an optional trailing argument.

For example, B = **FIR1**(N,Wn,kaiser(N+1,4)) uses a Kaiser window with beta=4.

B = **FIR1**(N,Wn,'high', chebwin(N+1,R)) uses a Chebyshev window.

For filters with a gain other than zero at Fs/2, e.g., highpass and bandstop filters, N must

be even. Otherwise, N will be incremented by one. In this case the window length

should be specified as N+2.

By default, the filter is scaled so the center of the first pass band has magnitude exactly one after windowing. Use a trailing 'noscale' argument to prevent this scaling, e.g.

B = **FIR1**(N,Wn,'noscale')

B = **FIR1**(N,Wn,'high','noscale')

B = **FIR1**(N,Wn,wind,'noscale').

You can also specify the scaling explicitly, e.g. **FIR1**(N,Wn,'scale'), etc.

FREQZ Digital Filter Frequency Response.

[H,W] = **FREQZ**(B,A,N) returns the N-point complex frequency response vector H and

the N-point frequency vector W in radians/sample of the filter: given numerator and

denominator coefficients 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.

[H,W] = **FREQZ**(B,A,N,'whole') uses N points around the whole unit circle.

H = **FREQZ**(B,A,W) returns the frequency response at frequencies designated in vector W, in radians/sample (normally between 0 and pi).

[H,F] = **FREQZ**(B,A,N,Fs) and [H,F] = FREQZ(B,A,N,'whole',Fs) return frequency vector F (in Hz), where Fs is the sampling frequency (in Hz).

H = **FREQZ**(B,A,F,Fs) returns the complex frequency response at the frequencies designated in vector F (in Hz), where Fs is the sampling frequency (in Hz).

[H,W,S] = **FREQZ**(...) or [H,F,S] = FREQZ(...) returns plotting information to be used with FREQZPLOT. S is a structure whose fields can be altered to obtain different frequency response plots. For more information see the help for FREQZPLOT.

FREQZ(B,A,...) with no output arguments plots the magnitude and unwrapped phase of the filter in the current figure window.

DESIGNING   A LOW PASS FILTER:

Suppose the target is to pass all frequencies below 1200 Hz

```
fs=8000; % sampling frequency
n=50;   % order of the filter
w=1200/ (fs/2);
b=fir1(n,w,'low'); % Zeros of the filter
freqz(b,1,128,8000); % Magnitude and Phase Plot of the filter figure(2)
[h,w]=freqz(b,1,128,8000);
subplot(2,1,1);
plot(w,abs(h));% Normalized Magnitude Plot
title('Normalized Magnitude Plot');
grid
subplot(2,1,2);
 figure(2)
 zplane(b,1);       %  the plot in lab
 title('zplane');
```

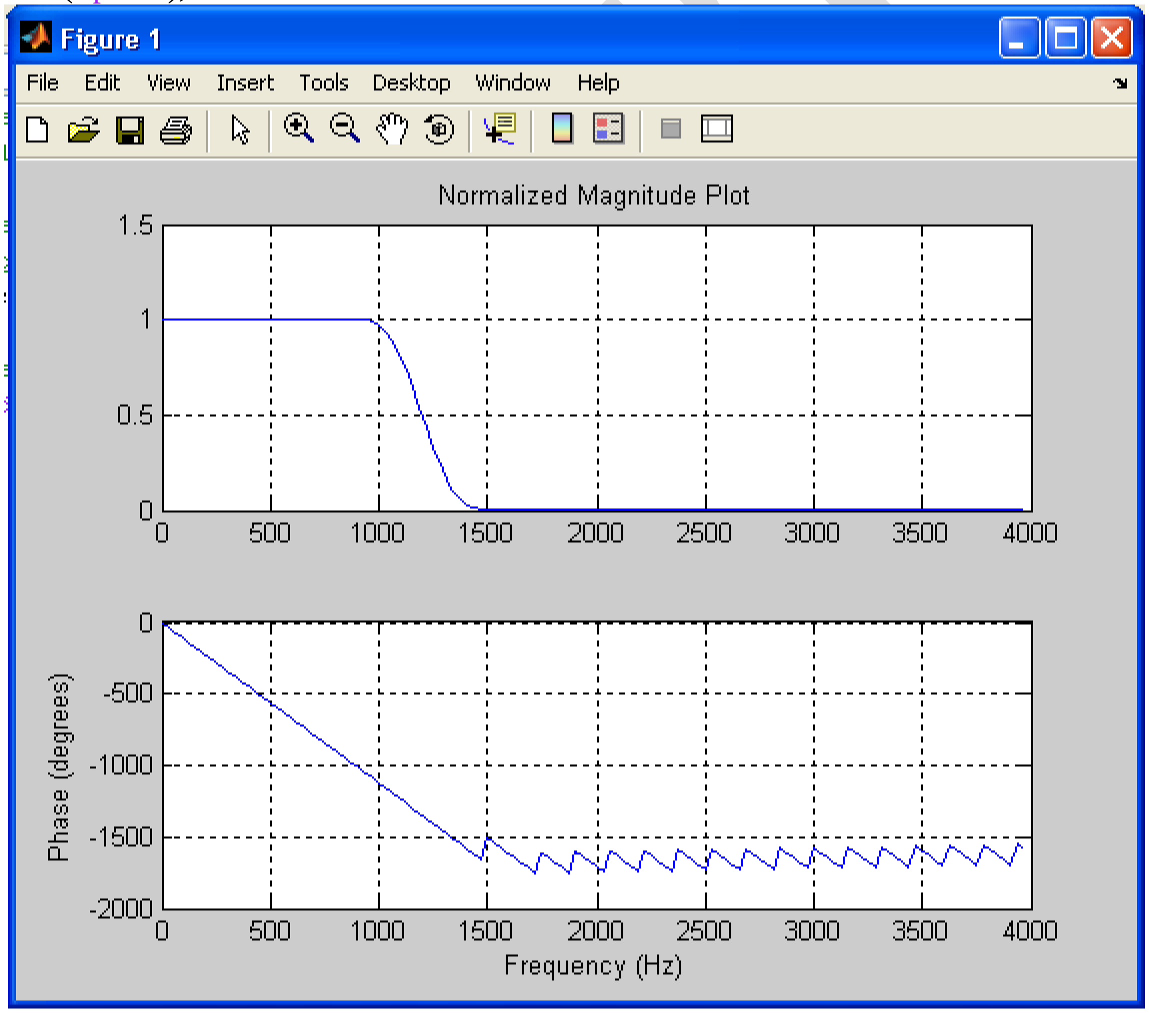
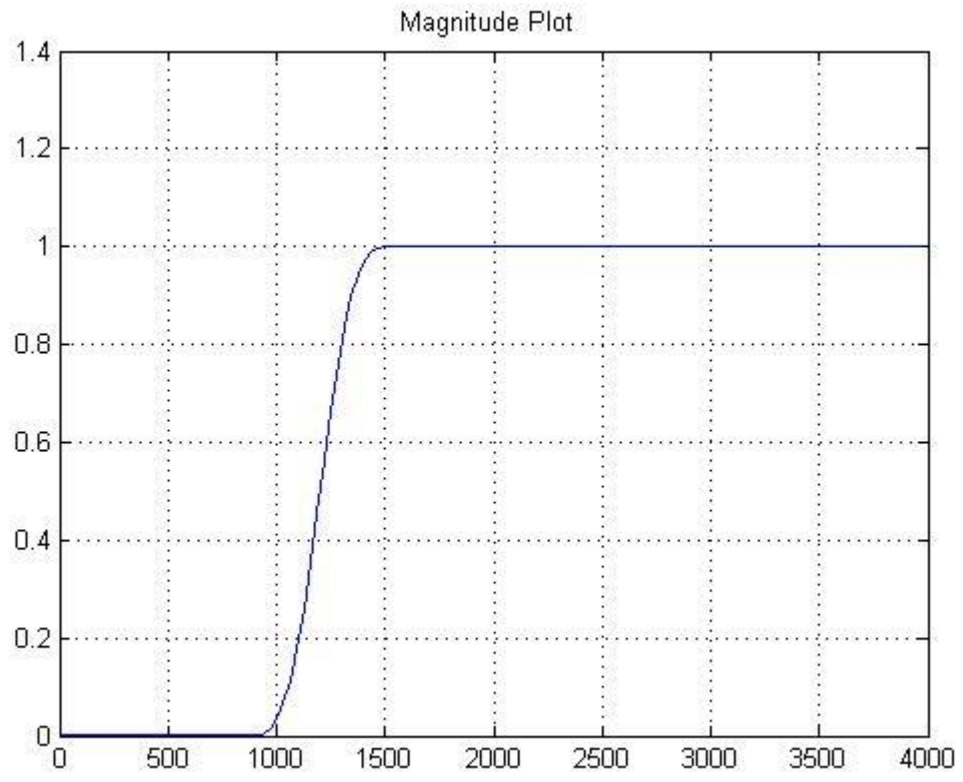


Figure 1
File Edit View Insert Tools Desktop Window Help
Normalized Magnitude Plot
Phase (degrees)
Frequency (Hz)

DESIGNING HIGH PASS FILTER:

Now the target is to pass all frequencies above 1200 Hz

```
fs=8000;
n=50;
w=1200/ (fs/2);
 b=fir1(n,w,'high');
 freqz(b,1,128,8000); % this function plots the phase(degree)and magnitude in db
subplot(2,1,2)
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
title('Magnitude Plot ');
grid
 figure(3)
 zplane(b,1); % this function plots fiq in zplane
```



Designing High Pass Filter:

```
fs=8000;

n=50;

w=1200/ (fs/2);
```

```
b=fir1(n,w,'high');
freqz(b,1,128,8000);
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing Band Pass Filter:

```
fs=8000;
n=40;
b=fir1(n,[1200/4000 1800/4000],’bandpass’); freqz(b,1,128,8000)
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
zplane(b,1);
```

Designing Notch Filter

```
fs=8000;
n=40;
b=fir1(n,[1500/4000 1550/4000],'stop'); freqz(b,1,128,8000)
figure(2)
[h,w]=freqz(b,1,128,8000);
plot(w,abs(h)); % Normalized Magnitude Plot
grid
figure(3)
```

```
zplane(b,1);
```

Designing Multi Band Filter

```
n=50;

w=[0.2 0.4 0.6];

b=fir1(n,w);

freqz(b,1,128,8000) figure(2)

[h,w]=freqz(b,1,128,8000);

plot(w,abs(h)); % Normalized Magnitude Plot

grid

figure(3)

zplane(b,1);
```

RESULT: The FIR low pass & high pass filter for given values is obtained. Hence the ideal and practical response of FIR filter are proved.

Outcomes:

After finishing this experiment the students are able to:

1. Able to Implement LP FIR filter for a given sequence
2. Calculate the filter coefficients.

VIVA QUESTION:

1. What is filter?
2. What is FIR and IIR filter define, and distinguish between these two?
3. What is window method? How you will design an FIR filter using window method?
4. What are low-pass and band-pass filter and what is the difference between these two?
5. What is the matlab command for Hamming window? Explain.
6. What do you mean by built in function ‘abs’ and where it is used?
7. Explain how the FIR filter are stable?
8. Why is the impulse response "finite"?
9. What does "FIR" mean?
10. What are the advantages of FIR Filters (compared to IIR filters)?
11. What are the disadvantages of FIR Filters (compared to IIR filters)?
12. What terms are used in describing FIR filters?
13. What is the delay of a linear-phase FIR?

14. What is the Z transform of a FIR filter?
15. What is the frequency response formula for a FIR filter?
16. How Can I calculate the frequency response of a FIR using the Discrete Fourier Transform (DFT)?
17. What is the DC gain of a FIR filter?

EXPERIMENT # 8 & 9

IMPLEMENTATION OF ANALOG IIR LOW PASS AND HIGH PASS FILTER FOR A GIVEN SEQUENCE

AIM: To implementation of IIR low pass and high pass filter.
To verify Frequency response of analog IIR filter using MATLAB (LP/HP).

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

THEORY: Matlab contains various routines for design and analyzing digital filter IIR. Most of these are part of the signal processing tool box. A selection of these filters is listed below.

- Buttord (for calculating the order of filter)
- Butter (creates an IIR filter)
- Ellipord (for calculating the order of filter) Ellip (creates an IIR filter)
- Cheb1ord (for calculating the order of filter) Cheyb1 (creates an IIR filter)
- Explanation Of The Commands For Filter Design: 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 pass band and has at least Rs dB of attenuation in the stop band. Wp and Ws are the pass band and stop band edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample).

For example
Low pass: Wp = .1, Ws = .2
High pass: Wp = .2, Ws = .1
Band pass: Wp = [.2 .7], Ws = [.1 .8]
Band stop: 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.

ELLIPORD: ELLIPTIC FILTER ORDER SELECTION.

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

For example

Low pass: Wp = .1,   Ws = .2
High pass: Wp = .2,   Ws = .1
Band pass: Wp = [.2 .7], Ws = [.1 .8]
Band stop: Wp = [.1 .8], Ws = [.2 .7]

ELLIPORD also returns Wn, the elliptic natural frequency to use with ELLIP to achieve the specifications.

[N, Wn] = **ELLIPORD**(Wp, Ws, Rp, Rs, 's') does the computation for an analog filter, in which case Wp and Ws are in radians/second. NOTE: If Rs is much greater than Rp, or Wp and Ws are very close, the estimated order can be infinite due to limitations of numerical precision.

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 pass band and has at least Rs dB of attenuation in the stop band. Wp and Ws are the pass band and stop band edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample).

For example,

Low pass: Wp = .1,   Ws = .2
High pass: Wp = .2,   Ws = .1
Band pass: Wp = [.2 .7], Ws = [.1 .8]
Band stop: Wp = [.1 .8], Ws = [.2 .7]

CHEB1ORD also returns Wn, the Chebyshev natural frequency to use with CHEBY1 to achieve the specifications. [N, Wn] = **CHEB1ORD**(Wp, Ws, Rp, Rs, 's') does the computation for an analog filter, in which case Wp and Ws are in radians/second.

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].

When used with three left-hand arguments, as in [Z,P,K] = BUTTER(...), the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K. When used with four left-hand arguments, as in [A,B,C,D] = BUTTER(...), state-space matrices are returned.

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.

ELLIP: ELLIPTIC OR CAUER DIGITAL AND ANALOG FILTER DESIGN.

[B,A] **= ELLIP**(N,Rp,Rs,Wn) designs an Nth order low pass digital elliptic filter with Rp decibels of peak-to-peak ripple and a minimum stop band attenuation of Rs decibels.
ELLIP 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 Rp = 0.5 and Rs = 20 as starting points, if you are unsure about choosing them. If Wn is a two-element vector, Wn = [W1 W2], ELLIP returns an order 2N band pass filter with pass band W1 < W < W2. [B,A] = ELLIP(N,Rp,Rs,Wn,'high') designs a high pass filter.

[B,A] = **ELLIP**(N,Rp,Rs,Wn,'stop') is a band stop filter if Wn = [W1 W2].

When used with three left-hand arguments, as in [Z,P,K] = ELLIP(...), the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K. When used with four left-hand arguments, as in [A,B,C,D] = ELLIP(...), state-space matrices are returned.

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

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.
If Wn is a two-element vector, Wn = [W1 W2], CHEBY1 returns an order 2N bandpass filter with passband W1 < W < W2.

[B,A] = **CHEBY1**(N,R,Wn,'high') designs a highpass filter.

[B,A] = **CHEBY1**(N,R,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

When used with three left-hand arguments, as in [Z,P,K] = CHEBY1(...), the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K.
When used with four left-hand arguments, as in [A,B,C,D] = CHEBY1(...), state-space matrices are returned.

CHEBY1(N,R,Wn,'s'), **CHEBY1**(N,R,Wn,'high','s') and **CHEBY1**(N,R,Wn,'stop','s') design analog Chebyshev Type I filters.In this case, Wn is in [rad/s] and it can be greater than 1.0.

ALGORITHM:

1. Get the pass band and stop band ripples.
2. Get the pass band and stop band edge frequencies.
3. Get the sampling frequency.
4. Calculate the order of the filter.
5. Find the window coefficients.
6. Draw the magnitude and phase responses.

PROCEDURE:

1) Enter the pass band ripple (rp) and stop band ripple (rs).

2) Enter the pass band frequency (fp) and stop band frequency (fs).

3) Get the sampling frequency (f).

4) Calculate the analog pass band edge frequencies, w1 and w2.

w1 = 2*fp/f   w2 = 2*fs/f

5) Calculate the order and 3dB cutoff frequency of the analog filter. [Make use of the following function]   [n,wn]=buttord(w1,w2,rp,rs,'s')

6) Design an nth order analog lowpass Butter worth filter using the following statement. [b,a]=butter(n,wn,'s')

7) Find the complex frequency response of the filter by using 'freqs( )' function [h,om]=freqs(b,a,w) where, w = 0:.01:pi

This function returns complex frequency response vector 'h' and frequency vector 'om' in radians/samples of the filter.

$$H(s) = \frac{b(s)}{a(s)} = \frac{b(1)S^{nb-1}+b(2)S^{nb-2}+\ldots\ldots\ldots\ldots b(nb)}{a(1)S^{na-1}+a(2)S^{na-2}+\ldots\ldots\ldots\ldots a(na)}$$

Where a,b are the vectors containing the denominator and numerator coefficients.

8) Calculate the magnitude of the frequency response in decibels (dB) m=20*log10(abs(h))

9) Plot the magnitude response [magnitude in dB Vs normalized frequency (om/pi)]

10)Give relevant names to x and y axes and give an appropriate title for the plot.

11)Repeat the procedure for highpass, bandpass, bandstop filters.

12)Plot all the responses in a single figure window.[Make use of subplot(   )].

9) Repeat the program for Chebyshev type-I and Chebyshev type-II filters.

PROGRAM CODE:

A. CHEBYSHEV LOW PASS FILTER

```
clc;
clear all;
wp=0.5; %%% chebyshev low pass filter;
ws=0.7;
rp=1;
rs=50;
[n,wn]=cheb1ord(wp,ws,rp,rs);
[b,a]=cheby1(n,rp,wn);
[h,w]=freqz(b,a,128);
subplot(2,2,1)
plot(abs(h));
xlabel('frequency');
ylabel('amplitude');
title('low pass chebyshev filter response');
```

B. BUTTER WORTH LOW PASS FILTER

```
wp=0.5;  % butter worth low pass filter
ws=0.7;
rp=1;
rs=50;
[n,wn]=buttord(wp,ws,rp,rs);
n
[b,a]=butter(n,wn);
[h,w]=freqz(b,a,128);
subplot(2,2,3)
```

```
plot(abs(h));
xlabel('frequency');6
ylabel('amplitude');
title('low pass butterworth filter response');
```

9.A CHEBYSHEV HIGH PASS FILTER

```
wp=0.7;   %%% chebyshev high pass filter
ws=0.5;
rp=1;
rs=30;
[n,wn]=cheb1ord(wp,ws,rp,rs);
[b,a]=cheby1(n,rp,wn, 'high');
[h,w]=freqz(b,a,128);
subplot(2,2,2)
plot(abs(h));
xlabel('frequency');
ylabel('amplitude');
title('high pass chebyshev filter response');
```

9.B BUTTER WORTH HIGH PASS FILTER

```
wp=0.7;   %butter worth high pass filter
ws=0.5;
rp=1;
rs=30;
[n,wn]=buttord(wp,ws,rp,rs);
n
[b,a]=butter(n,wn,'high');
[h,w]=freqz(b,a,128);
```

```
subplot(2,2,4)
plot(abs(h));
xlabel('frequency');
ylabel('amplitude');
title('high pass butterworth filter response');
```

OUTPUT:

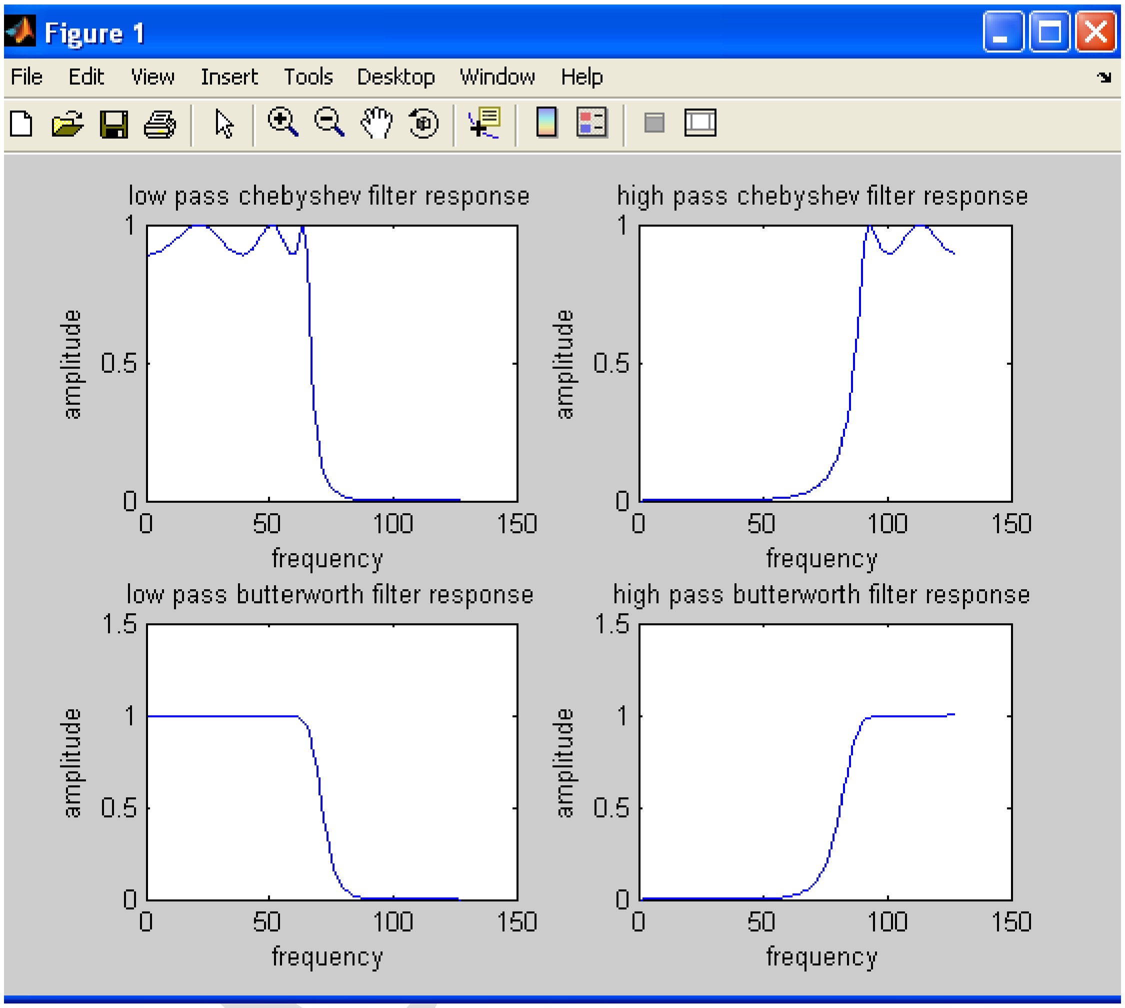


Figure 1
File Edit View Insert Tools Desktop Window Help
low pass chebyshev filter response
high pass chebyshev filter response
low pass butterworth filter response
high pass butterworth filter response
amplitude
frequency
0
0.5
1
1.5
0
50
100
150

RESULT: The IIR low pass & high pass IIR filter for given values is obtained. Hence the ideal and practical response of IIR filter are proved.

Outcomes:

After finishing this experiment the students are able to:

1. Implement HP FIR filter for a given sequence
2. Plot the response of the same.

VIVA QUESTION:

1. What do you mean by cut-off frequency?
2. Give the difference between analog and digital filter?
3. What is the difference between type 1 and type 2 filter structure?
4. what is the role of delay element in filter design?
5. Explain how the frequency is filter in filters?
6. Differences between Butterworth chebyshev filters?
7. Can IIR filters be Linear phase? how to make it linear Phase?
8. What is the special about minimum phase filter?

10. Generation of Sinusoidal signal through filtering

AIM : To generate the sinusoidal signal using filter.

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

PROGRAME:

```
clear all
close all
echo on
t0=.2; % signal duration
ts=0.001; % sampling interval
fc=250; % carrier frequency
fs=1/ts; % sampling frequency
t=[-t0/2:ts:t0/2]; % time vector
kf=100; % deviation constant
m=cos(2*pi*10*t); % the message signal
int_m(1)=0;
for i=1:length(t)-1 % integral of m
  int_m(i+1)=int_m(i)+m(i)*ts;
  echo off ;
```

```
end
echo on ;
u=cos(2*pi*fc*t+2*pi*kf*int_m); % modulated signal

%now lets filter the signal
[z,p] = butter(1,2*50/fs,'low');
filter_out = filter(50,[1 50],u); %this damn filter doesn't work!

subplot(2,1,1);
plot(t,u);
hold on;
plot(t,m,'r');

subplot(2,1,2);
plot(t,filter_out);
hold on;
plot(t,m,'r')
```

RESULT: The sinusoidal signal is generated from filter response for given values .

Outcome : At the end of the experiment the students are able to generate a sinusoidal signal is generated from filter response for given values

VIVA QUESTION:

1. What is the special about minimum phase filter?
2. In signal processing, why we are much more interested in orthogonal transform?
3. How is the non-periodic nature of the input signal handled?
4. If a have two vectors how will i check the orthogonality of those vectors?

12. IMPLEMENTATION OF DECIMATION PROCESS

AIM: To implementation of decimation of given sequence by factor M.

EQUIPMENTS:

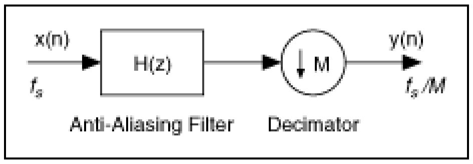
Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

THEORY :

Decimation is the process of reducing the sampling frequency of a signal to a lower sampling frequency that differs from the original frequency by an integer value. Decimation also is known as down-sampling. The lowpass filtering associated with decimation removes high-frequency content from the signal to accommodate the new sampling frequency.

Decimation is useful in applications in which the Nyquist frequency of a signal is much higher than the highest frequency of the signal. Decimation filters help you remove the excess bandwidth and reduce the sampling frequency of the signal. Decimation filters also help you reduce the computational resources required for processing and storing the signal. During the analog-to-digital (A/D) conversion process, decimation filters also can reduce the variance of quantization noise in a signal and maintain the signal power, thus improving the signal-to-noise ratio (SNR).

The following figure shows a typical M-fold decimation filter, where M is the integer value by which you want to decrease the sampling frequency. This filter contains a lowpass FIR filter $H(z)$. This lowpass FIR filter is an anti-aliasing filter followed by an$M$-fold decimator. The decimator passes every M^{th} sample and discards the other samples. After this operation, the decimation filter changes the sampling frequency f_s of the input signal $x(n)$ to a new sampling frequency f_s/M. The decimation filter then returns an output signal $y(n)$ with the new sampling frequency.



$y(n) = x(Mn)$ equation for decimation

To prevent aliasing, this system uses the lowpass filter $H(z)$ before the M-fold decimator to suppress the frequency contents above the frequency $f_s/(2M)$, which is the Nyquist frequency of the output signal. This system produces the same results as an analog anti-aliasing filter with a cutoff frequency of $f_s/(2M)$ followed by an analog-to-digital (A/D) converter with a sampling frequency of f_s/M. Because the system shown in the figure above is in the digital domain, $H(z)$ is a digital anti-aliasing filter.

PROGRAM:

```
Clc;
Close all;
Clear all;

N=input(‘ input length of the sine seq’);
M= input(‘ down sampling factor’);
fi= input(‘ input signal freq’);  % generation of sine sequence
t=0:N-1;
L=input(‘ enter the factor value’);
m=0:N*L-1;
x=sin(2*pi*fi*m);
%generate the down sample signal
Y=x([1:m:length(x)]);
Y([1:L:length(x)])=x;
Subplot(2,1,1);
Stem(n,x(1:N));
Title(‘ input sequence’);
Xlable(‘time’);
Ylabel(‘amplitude’);
Subplot(2,1,2);
Stem(n, Y);
Title([‘ output sequence ‘,num2str(m)]);

Xlable(‘time’);
Ylabel(‘amplitude’);
```

RESULT: The decimator for a given sequence is observed for chosen factor **M**.

Hence the theory and practical is verified.

Outcome : At the end of the experiment the student is able to perform decimator for a given sequence is observed for chosen factor **M**.

VIVA QUESTION:

1. What is the importance of decimation for a given signal/sequence?
2. What do you mean Aliasing? What is the condition to avoid aliasing for sampling?
3. How does poly phase filtering save computations in a decimation filter?
4. Give any practical application of decimation?
5. Which signals can be down sampled?
6. What happens if I violate the Nyquist criteria in down sampling or decimating?
7. Can we do decimate in multiple stages?
8. What are "decimation" and "downsampling"?
9. What is the "decimation factor"?

13. IMPLEMENTATION OF INTERPOLATION PROCESS

AIM: To implementation of interpolation for given sequence by factor L.

EQUIPMENTS:

Operating System	- Windows XP
Constructor	- Simulator
Software	- CCStudio 3 & MATLAB 7.5

THEORY:

The process of increasing the sampling rate is called interpolation. Interpolation is upsampling followed by appropriate filtering. $y(n)$ obtained by interpolating $x(n)$, is generally represented as:

$$y(n) = x(\frac{n}{L})$$

The simplest method to interpolate by a factor of L is to add L-1 zeros in between the samples, multiply the amplitude by L and filter the generated signal, with a so-called anti-imaging low pass filter at the high sampling frequency.

PROGRAME:

```
Clc;
Close all;
Clear all;

N=input(‘ input length of the sine seq’);
L= input(‘ up sampling factor’);
Fi= input(‘ input signal freq’);
t=0:N-1;
X=sin(2*pi*fi*t);

Y=zeros(1,L*length(x));
Y([1:L:length(x)])=x;
Subplot(2,1,1);
Stem(n,x);
Title(‘ input sequence’);
Xlable(‘time’);
Ylabel(‘amplitude’);
Subplot(2,1,2);
Stem(n, Y(1:length(x));
Title([‘ output sequence ‘,num2str(L)]);

Xlable(‘time’);
Ylabel(‘amplitude’);
```

RESULT: The interpolation of given sequence is observed for factor value **L**. Hence the theory and practical are verified.

Outcome : At the end of the experiment the student is able to calculate the interpolation of given sequence is observed for factor value **L** and prove the theory and practical same.

VIVA QUESTION:

1. How does polyphase filtering save computations in an interpolation filter?
2. Why do we need I&Q signals?
3. What is Interpolation and decimation filters and why we need it?
4. What are "upsampling" and "interpolation"?
5. Why interpolate,i needed for any signal/sequence?
6. What is the "interpolation factor"?
7. Which signals can be interpolated?
8. Can interpolate will happens in multiple stages? If yes give reason?
9. Give any example of a FIR interpolator?

Hardware Interfacing

INTERFACING WITH DSKTMS320C6713

SOFTARE REQUIREMENTS:

Operating System – Windows XP
Constructor **-** Simulator
Software – Code Composer 3.1v,6713DSK Diagnostics

HARDARE REQUIREMENTS:

TMS320C6713DSP KIT
USB cable
Power Supply +5v

PROCEDURE:

1. Open code composer studio

2. Create a new project, give a meaningful name ,save into working folder, select target as TMS320C6713,and select output file type is execution. out.

3. Click on file open→new→source file save it with .c extension in working folder and write .c-code for desired task.

4. Add c-file to project as a source [path source→add files to project→add c. file].

5. Add the linker command file→hello .cmd.[path c.\ccstudio-v3.1\tutorial\dsk67/3\HELLO1\Hello.cmd]

6. Add the run time support library file rts6700.lib.[path c.\ccstudio-v.3.1\c600\cg tools\lib\rts6700.lib].

7. Compile the program [path project→compile file]

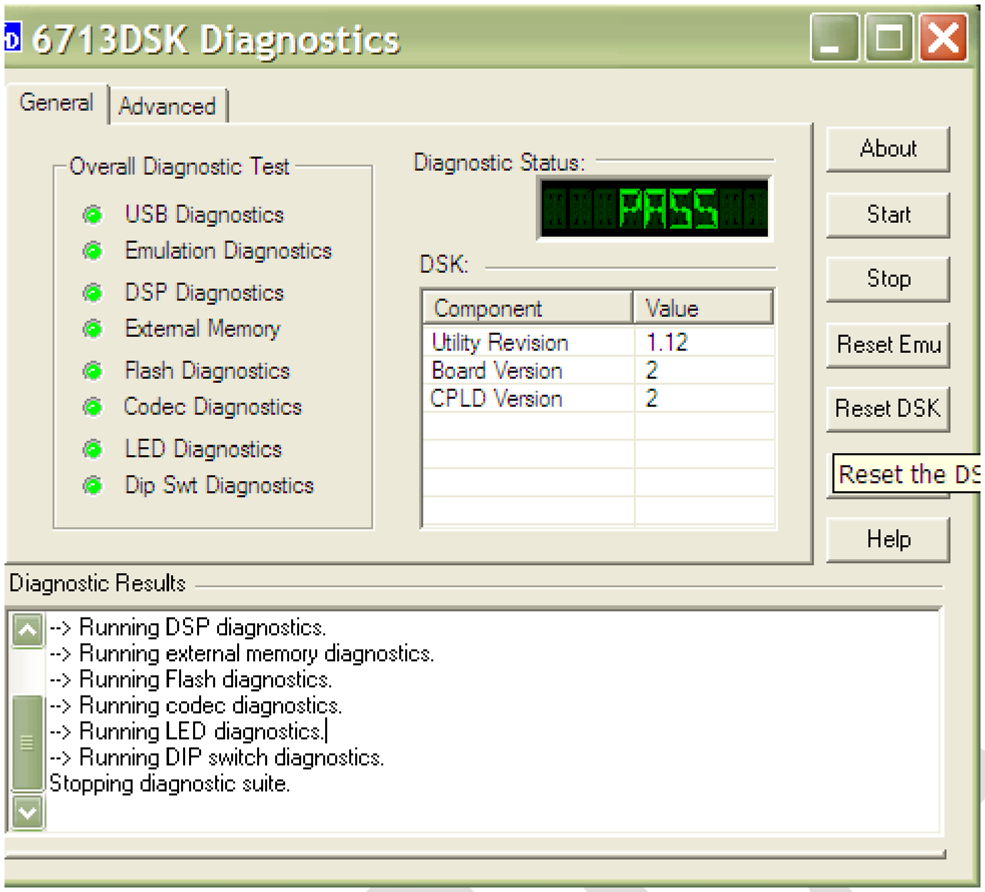
10. Build the program [path project→build]

11.Connect the kit to the system through the USB cable

12. Then switch on the kit

13. And now check the DSK6713 diagnostics kit whether all the connections are

Made correctly or not



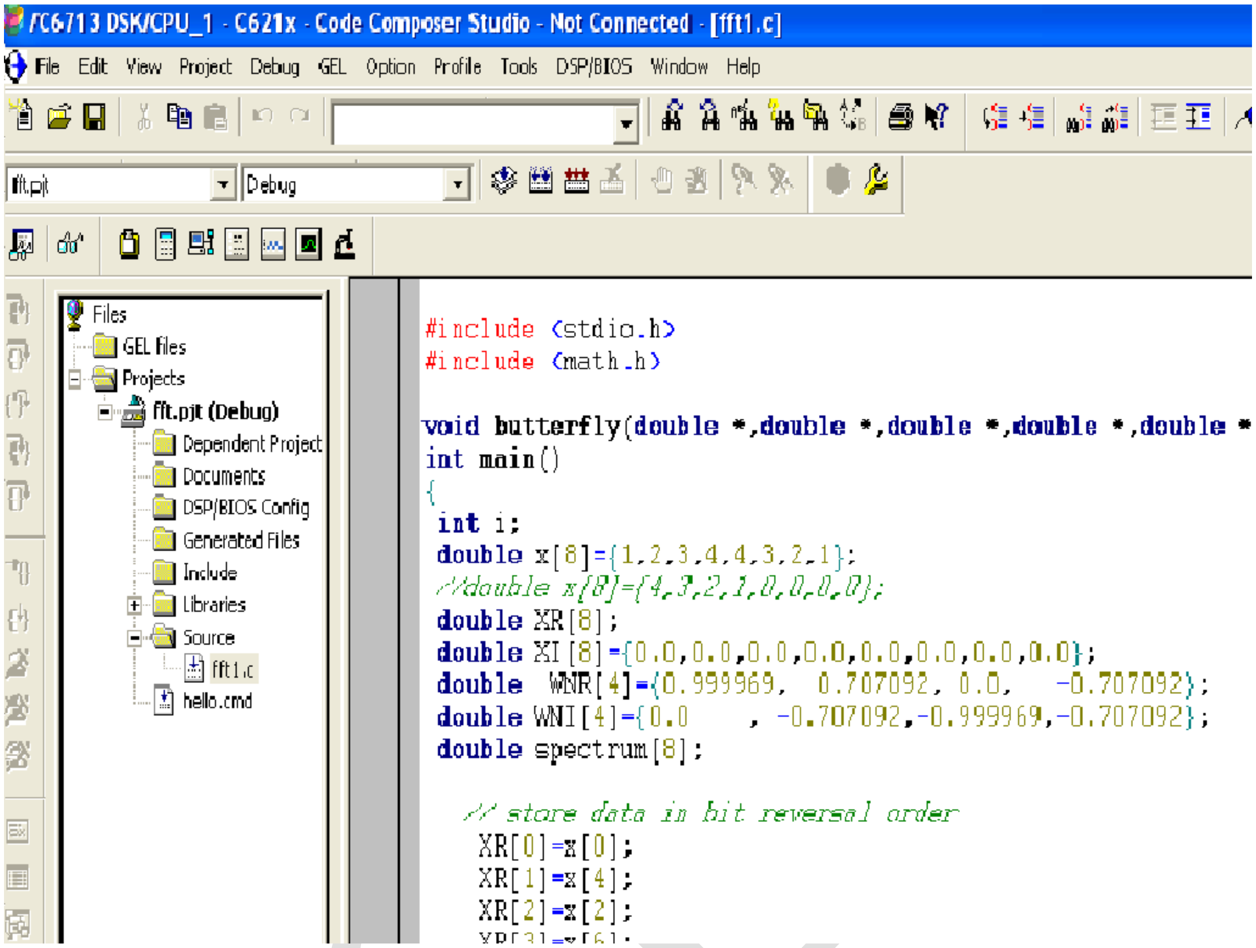
Component	Value
Utility Revision	1.12
Board Version	2
CPLD Version	2

```
--> Running DSP diagnostics.
--> Running external memory diagnostics.
--> Running Flash diagnostics.
--> Running codec diagnostics.
--> Running LED diagnostics.
--> Running DIP switch diagnostics.
Stopping diagnostic suite.
```

14. Then select [debug→connect] on the cc studio window

15. Load the program [file→load program]

16. Run the program [debug→run].


/C6713 DSK/CPU_1 - C621x - Code Composer Studio - Not Connected - [fft1.c]
File Edit View Project Debug GEL Option Profile Tools DSP/BIOS Window Help
fft.pjt
Debug
Files
GEL files
Projects
fft.pjt (Debug)
Dependent Project
Documents
DSP/BIOS Config
Generated Files
Include
Libraries
Source
fft1.c
hello.cmd

```
#include <stdio.h>
#include <math.h>

void butterfly(double *,double *,double *,double *,double *
int main()
{
 int i;
 double x[8]={1,2,3,4,4,3,2,1};
 //double x[8]={4,3,2,1,0,0,0,0};
 double XR[8];
 double XI [8] ={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
 double  WNR[4]={0.999969,  0.707092, 0.0,   -0.707092};
 double WNI[4]={0.0     , -0.707092,-0.999969,-0.707092};
 double spectrum[8];

   // store data in bit reversal order
    XR[0]=x[0];
    XR[1]=x[4];
    XR[2]=x[2];
```

4. N-POINT FFT ALGORITHM

AIM: To verify Linear Convolution using Code Composer Studio.

PROGRAM CODE:

```
#include <stdio.h>
#include <math.h>
void butterfly(double *,double *,double *,double *,double *,double *);
int main( )
{
int i;
double x[8]={1,2,3,4,4,3,2,1};
//double x[8]={4,3,2,1,0,0,0,0};
 double XR[8];
 double XI[8]={0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0};
 double  WNR[4]={0.999969,  0.707092, 0.0,   -0.707092};
 double WNI[4]={0.0    , -0.707092,-0.999969,-0.707092};
 double spectrum[8];

   // store data in bit reversal order
    XR[0]=x[0];
    XR[1]=x[4];
    XR[2]=x[2];
    XR[3]=x[6];
    XR[4]=x[1];
    XR[5]=x[5];
    XR[6]=x[3];
    XR[7]=x[7];
  //  clrscr( );

// FIRST STAGE OF BUTTERFLY
   butterfly(&XR[0],&XI[0],&XR[1],&XI[1],&WNR[0],&WNI[0]);
   butterfly(&XR[2],&XI[2],&XR[3],&XI[3],&WNR[0],&WNI[0]);
   butterfly(&XR[4],&XI[4],&XR[5],&XI[5],&WNR[0],&WNI[0]);
   butterfly(&XR[6],&XI[6],&XR[7],&XI[7],&WNR[0],&WNI[0]);

  // SECOND STAGE OF BUTTERFLY

   butterfly(&XR[0],&XI[0],&XR[2],&XI[2],&WNR[0],&WNI[0]);
   butterfly(&XR[1],&XI[1],&XR[3],&XI[3],&WNR[2],&WNI[2]);
   butterfly(&XR[4],&XI[4],&XR[6],&XI[6],&WNR[0],&WNI[0]);
   butterfly(&XR[5],&XI[5],&XR[7],&XI[7],&WNR[2],&WNI[2]);

  // THIRD STAGE OF BUTTERFLY
   butterfly(&XR[0],&XI[0],&XR[4],&XI[4],&WNR[0],&WNI[0]);
   butterfly(&XR[1],&XI[1],&XR[5],&XI[5],&WNR[1],&WNI[1]);
   butterfly(&XR[2],&XI[2],&XR[6],&XI[6],&WNR[2],&WNI[2]);
   butterfly(&XR[3],&XI[3],&XR[7],&XI[7],&WNR[3],&WNI[3]);

    for(i=0;i<8;i++)
     {
       XR[i]*=64.0;
       XI[i]*=64.0;
      }
    for(i=0;i<8;i++)
```

```
        {
        printf("\n FFT XR[%d] = %lf XI[%d]=  %lf",i,XR[i],i,XI[i]);
        }
        return(0);
        }
    void butterfly(double *ar,double *ai,double *br,double *bi,double *wr,double *wi)
        {
         double tr,ti;
         *ar/=4.0;
         *ai/=4.0;
          tr=*ar*2;
          ti=*ai*2;
         *br/=4.0;
         *bi/=4.0;
         *ar+=*br * *wr - *bi * *wi;
         *ai+=*br * *wi + *bi * *wr;
         *br= tr - *ar;
         *bi= ti - *ai;
          }
```

Expected Output:

```
XR[0]=19.9999070; XI[0]=0.000000;
XR[1]=-5.828134;    XI[1]=-2.414237;
XR[2]=0.000813;     XI[2]=-0.000093;
XR[3]=-0.171618;    XI[3]=-0.414175;
XR[4]=0.000248;     XI[4]=0.000000;
```

5&6 . FIR Filter (LP/HP) Using Windowing technique

Theory:

DESIGNING AN FIR FILTER :

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

I. Clearly specify the filter specifications.

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

II. Compute the cut-off frequency W_c

Eg: W_c = 2*pie* f_c / F_s
= 2*pie* 400/8000
= 0.1*pie

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

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

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

AIM: To verify FIR filters using Code Composer Studio.

1. Rectangular window (High pass)

PROGRAM CODE:

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


//float filter_coeff[ ]={0.021665,0.022076,0.020224,0.015918,0.009129,
//                      -0.000000,-0.011158,-0.023877,-0.037558,-0.051511,
//                       -0.064994,-0.077266,-0.087636,-0.095507,-0.100422,
//                       0.918834,-0.100422,-0.095507,-0.087636,-0.077266,
//                      -0.064994,-0.051511,-0.037558,-0.023877,-0.011158,
//                      -0.000000,0.009129,0.015918,0.020224,0.022076,
//                      0.021665};
                    //FIR High pass Rectangular filter pass band range 400Hz- 3.5KHz


//float filter_coeff[ ]={0.000000,-0.013457,-0.023448,-0.025402,-0.017127,
//                      -0.000000,0.020933,0.038103,0.043547,0.031399,
//                      0.000000,-0.047098,-0.101609,-0.152414,-0.188394,
//                      0.805541,-0.188394,-0.152414,-0.101609,-0.047098,
//                      0.000000,0.031399,0.043547,0.038103,0.020933,
//                      -0.000000,-0.017127,-0.025402,-0.023448,-0.013457,
//                      0.000000};
                   /|/FIR High pass Rectangular filter pass band range 800Hz-3.5KHz


float filter_coeff[ ]={-0.020798,-0.013098,0.007416,0.024725,0.022944,
                    -0.000000,-0.028043,-0.037087,-0.013772,0.030562,
                                  0.062393,0.045842,-0.032134,-0.148349,-0.252386,
                                  0.686050,-0.252386,-0.148349,-0.032134,0.045842,
                                  0.062393,0.030562,-0.013772,-0.037087,-0.028043,
                                 -0.000000,0.022944,0.024725,0.007416,-0.013098,
                                  -0.020798}:
                             //FIR High pass Rectangular filter pass band range 1200Hz-3.5KHz


//static short int  in_buffer[100];

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};

void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init( );
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 1);

  while(1)
  {
       while(!DSK6713_AIC23_read(hCodec, &l_input));

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

                  l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
                           r_output=l_output;

                   while(!DSK6713_AIC23_write(hCodec, l_output));

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

             DSK6713_AIC23_closeCodec(hCodec);
}

signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
static short int  in_buffer[100];
in_buffer[0] = x;

for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1];

for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
//output = x;
return(output);
}
```

Expected Output: observed the output in CRO with respect to input signal, it pass the signal at range of frequency 1200-3.5KHz,and attenuate at other than pass band frequency

2. Rectangular window (Lowpass)

PROGRAM CODE:

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

//float filter_coeff[ ]={-0.008982,-0.017782,-0.025020,-0.029339,-0.029569,
//                        -0.024895,-0.014970,0.000000,0.019247,0.041491,
//                         0.065053,0.088016,0.108421,0.124473,0.134729,
//                         0.138255,0.134729,0.124473,0.108421,0.088016,
//                         0.065053,0.041491,0.019247,0.000000,-0.014970,
//                        -0.024895,-0.029569,-0.029339,-0.025020,-0.017782,
//                        -0.008982};
                    //FIR Low pass Rectangular Filter pass band range 0-500Hz

//float filter_coeff[ ]={-0.015752,-0.023869,-0.018176,0.000000,0.021481,
//                         0.033416,0.026254,-0.000000,-0.033755,-0.055693,
//                        -0.047257,0.000000,0.078762,0.167080,0.236286,
//                         0.262448,0.236286,0.167080,0.078762,0.000000,
//                        -0.047257,-0.055693,-0.033755,-0.000000,0.026254,
//                         0.033416,0.021481,0.000000,-0.018176,-0.023869,
//                        -0.015752};
                    //FIR Low pass Rectangular Filter pass band range 0-1000Hz


float filter_coeff[ ]={-0.020203,-0.016567,0.009656,0.027335,0.011411,
                        -0.023194,-0.033672,0.000000,0.043293,0.038657,
                       -0.025105,-0.082004,-0.041842,0.115971,0.303048,
                         0.386435,0.303048,0.115971,-0.041842,-0.082004,
                        -0.025105,0.038657,0.043293,0.000000,-0.033672,
                        -0.023194,0.011411,0.027335,0.009656,-0.016567,
                     -0.020203};
                    //FIR Low pass Rectangular Filter pass band range 0-1500Hz

//static short int  in_buffer[100];

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};

void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);
  DSK6713_AIC23_setFreq(hCodec, 1);

  while(1)
  {
      while(!DSK6713_AIC23_read(hCodec, &l_input));

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

               l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
                      r_output=l_output;

                while(!DSK6713_AIC23_write(hCodec, l_output));

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

           DSK6713_AIC23_closeCodec(hCodec);
}

signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
static short int  in_buffer[100];
in_buffer[0] = x;

for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1];

for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
//output = x;
return(output);
}
```

Expected Output: observed the output in CRO for low pass filter with respect to input signal, it pass the signal at range of pass band frequency 0-1500KHz,and attenuate at other than pass band frequency

3. **Triangular window (Highpass)**

PROGRAM CODE:

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

```
//float filter_coeff[]={0.000000,0.001445,0.002648,0.003127,0.002391,
                    //                                    -0.000000,-0.004383,-0.010943,-0.019672,-0.030353,
//                                                        -0.042554,-0.055647,-0.068853,-0.081290,-0.092048,
//                                                         0.902380,-0.092048,-0.081290,-0.068853,-0.055647,
//                                                        -0.042554,-0.030353,-0.019672,-0.010943,-0.004383,
//                                                    -0.000000,0.002391,0.003127,0.002648,0.001445,
//                                                        0.000000};
```
//FIR High pass Triangular filter pass band range 400Hz-3.5KHz

```
//float filter_coeff[]={0.000000,-0.000897,-0.003126,-0.005080,-0.004567,
//                                                 -0.000000,0.008373,0.017782,0.023225,0.018839,
//                                                  0.000000,-0.034539,-0.081287,-0.132092,-0.175834,
//                                                  0.805541,-0.175834,-0.132092,-0.081287,-0.034539,
//                                                  0.000000,0.018839,0.023225,0.017782,0.008373,
//                                                 -0.000000,-0.004567,-0.005080,-0.003126,-0.000897,
//                                                   0.000000};
```
//FIR High pass Triangular filter pass band range 800Hz-3.5KHz

```
float filter_coeff[]={0.000000,-0.000901,0.001021,0.005105,0.006317,
                                                        -0.000000,-0.011581,-0.017868,-0.007583,0.018931,
                                                         0.042944,0.034707,-0.026541,-0.132736,-0.243196,
                                                         0.708287,-0.243196,-0.132736,-0.026541,0.034707,
                                                       0.042944,0.018931,-0.007583,-0.017868,-0.011581,
                                                      -0.000000,0.006317,0.005105,0.001021,-0.000901,
                                                         0.000000};
```
//**FIR High pass Triangular filter pass band range 1200Hz-3.5KHz**

```
//static short int  in_buffer[100];

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};

void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 1);

  while(1)
  {
       while(!DSK6713_AIC23_read(hCodec, &l_input));

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

                  l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
                           r_output=l_output;

                   while(!DSK6713_AIC23_write(hCodec, l_output));
```

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

        DSK6713_AIC23_closeCodec(hCodec);
}

signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
static short int  in_buffer[100];
in_buffer[0] = x;

for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1];

for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
//output = x;
return(output);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 400-3.5KHz,and attenuate at other than pass band frequency

4. Triangular window (Lowpass)

PROGRAM CODE:

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

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

//FIR Low pass Triangular Filter pass band range 0-500Hz

```
//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};
```

//**FIR Low pass Triangular Filter pass band range 0-1000Hz**

```
float filter_coeff[ ]={0.000000,-0.001104,0.001287,0.005467,0.003043,
                                         -0.007731,-0.013469,0.000000,0.023089,0.023194,
                                         -0.016737,-0.060136,-0.033474,0.100508,0.282844,
                                          0.386435,0.282844,0.100508,-0.033474,-0.060136,
                                         -0.016737,0.023194,0.023089,0.000000,-0.013469,
                                         -0.007731,0.003043,0.005467,0.001287,-0.001104,
                                          0.000000};
                              //FIR Low pass Triangular Filter pass band range 0-1500Hz

//static short int  in_buffer[100];

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};

void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 1);

  while(1)
  {
      while(!DSK6713_AIC23_read(hCodec, &l_input));

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

                l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
                        r_output=l_output;

                 while(!DSK6713_AIC23_write(hCodec, l_output));

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

            DSK6713_AIC23_closeCodec(hCodec);
}

signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
static short int  in_buffer[100];
in_buffer[0] = x;

for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1];

for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
//output = x;
return(output);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 0-1500KHz,and attenuate at other than pass band frequency

5. Kaiser window (High pass)

PROGRAM CODE:

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


//float filter_coeff[ ]={0.000050,0.000223,0.000520,0.000831,0.000845,
//                      -0.000000,-0.002478,-0.007437,-0.015556,-0.027071,
//                      -0.041538,-0.057742,-0.073805,-0.087505,-0.096739,
//                       0.899998,-0.096739,-0.087505,-0.073805,-0.057742,
//                      -0.041538,-0.027071,-0.015556,-0.007437,-0.002478,
 //                     -0.000000,0.000845,0.000831,0.000520,0.000223,
 //                    0.000050};
```
//FIR High pass Kaiser filter pass band range 400Hz-3.5KHz

```
//float filter_coeff[ ]={0.000000,-0.000138,-0.000611,-0.001345,-0.001607,
//                      -0.000000,0.004714,0.012033,0.018287,0.016731,
//                       0.000000,-0.035687,-0.086763,-0.141588,-0.184011,
//                       0.800005,-0.184011,-0.141588,-0.086763,-0.035687,
//                                                      0.000000,0.016731,0.018287,0.012033,0.004714,
//                                                     -0.000000,-0.001607,-0.001345,-0.000611,-0.000138,
//                                                      0.000000};
                                         //FIR High pass Kaiser filter pass band range 800Hz-3.5KHz

float filter_coeff[ ]={-0.000050,-0.000138,0.000198,0.001345,0.002212,-0.000000,
                                          -0.006489,-0.012033,-0.005942,0.016731,0.041539,0.035687,
                                          -0.028191,-0.141589,-0.253270,0.700008,-0.253270,-0.141589,
                                          -0.028191,0.035687,0.041539,0.016731,-0.005942,-0.012033,
                                           -0.006489,-0.000000,0.002212,0.001345,0.000198,-0.000138,
                                           -0.000050};
                                      //FIR High pass Kaiser filter pass band range 1200Hz-3.5KHz



//static short int  in_buffer[100];

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};

void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 1);

  while(1)
  {
       while(!DSK6713_AIC23_read(hCodec, &l_input));

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

                l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
                       r_output=l_output;

                 while(!DSK6713_AIC23_write(hCodec, l_output));

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

            DSK6713_AIC23_closeCodec(hCodec);
}

signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
static short int  in_buffer[100];
in_buffer[0] = x;

for(i=30;i>0;i--)
```

```
in_buffer[i] = in_buffer[i-1];

for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
//output = x;
return(output);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 1200-3.5KHz,and attenuate at other than pass band frequency

6. Kaiser window (Low pass)

PROGRAM CODE:

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

//float filter_coeff[]={-0.000019,-0.000170,-0.000609,-0.001451,-0.002593,
//                          -0.003511,-0.003150,0.000000,0.007551,0.020655,
//                           0.039383,0.062306,0.086494,0.108031,0.122944,
//                          0.128279,0.122944,0.108031,0.086494,0.062306,
//                           0.039383,0.020655,0.007551,0.000000,-0.003150,
//                          -0.003511,-0.002593,-0.001451,-0.000609,-0.000710,
//                          -0.000019};
                        // kaiser low pass fir filter pass band range 0-500Hz

//float filter_coeff[]={-0.000035,-0.000234,-0.000454,0.000000,0.001933,
//                     0.004838,0.005671,-0.000000,-0.013596,-0.028462,
//                    -0.029370,0.000000,0.064504,0.148863,0.221349,
//                     0.249983,0.221349,0.148863,0.064504,0.000000,
//                    -0.029370,-0.028462,-0.013596,-0.000000,0.005671,
//                     0.004838,0.001933,0.000000,-0.000454,-0.000234,
//                    -0.000035};
                    // kaiser low pass fir filter pass band range 0-1000Hz
```

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

//Kaiser low pass fir filter pass band range 0-1500Hz

```
//static short int  in_buffer[100];

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};
void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 1);

  while(1)
  {
       while(!DSK6713_AIC23_read(hCodec, &l_input));

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

                  l_output=(Int16)FIR_FILTER(&filter_coeff ,l_input);
                           r_output=l_output;

                   while(!DSK6713_AIC23_write(hCodec, l_output));

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

             DSK6713_AIC23_closeCodec(hCodec);
}



signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;
static short int  in_buffer[100];
in_buffer[0] = x;

for(i=30;i>0;i--)
in_buffer[i] = in_buffer[i-1];

for(i=0;i<32;i++)
output = output + h[i] * in_buffer[i];
//output = x;
return(output);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 0-1500KHz, and attenuate at other than pass band frequency.

8&9. IIR FILTER (LP/HP) ON DSP PROCESSORS

AIM: To verify IIR Filter (LP/HP) using Code Composer Studio.

SOFTARE REQUIREMENTS:

Operating System – Windows XP
Constructor **-** Simulator
Software – Code Composer Studio3.1v, 6713DSK Diagnostics.

HARDARE REQUIREMENTS:

TMS320C6713DSP KIT
USB cable
Power Supply +5v The
CRO (0-20MHz),
Function Generator (1-1MHz), Cables

PROCEDURE:

1. Set 1v peak to peak in function generator, and connect this to 6713kit at Line in pin, connect CRO to kit at Line out pin.

2. Open code composer studio

3. create a new project ,give a meaningful name ,save into working folder, select target as TMS320C6713,and select output file type is execution. Out.

4. Click on file open→new→source file save it with .c extension in working folder and write C-code for desired task.

4. File→new→ dsp \bios configuration file select DSK6713.cdB. save it with .cdb extension in working folder.

5. Add .cdb file to project [path→right click on dsp\ Bios configuration→ select .cdb file] and choose .cdb file.

6. Add the library file ‘dsk6713bsl.lib’ to the current project. [Path c.\ccstudio\c600\dsk6713\lib\dsk6713bst.lib]

7. Under working folder include 3-header files

“dsk6713.h”

“dsk6713_aic23.h”

“Filtercfg.h”(pathc:\ccstudio\c6000\dsk6713\include.)

8. Compile the program

9. Build the program

10. Connect the kit and load the program then run

11. In the CRO we can verify the output signal with input signal

1. Butterworth (High pass)

PROGRAM CODE:

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

//const signed int filter_coeff[ ] = {20539,-20539,20539,32767,-18173,13406};
              //IIR_BUTTERWORTH_HP FILTER  pass band range 2.5kHz-11KHz
//const signed int filter_coeff[ ] = {15241,-15241,15241,32761,-10161,7877};
               //IIR_BUTTERWORTH_HP FILTER pass band range 4kHz-11KHz
const signed int filter_coeff[ ] = { 7215,-7215,7215,32767,5039,6171};
                 //IIR_BUTTERWORTH_HP FILTER pass band range 7kHz-11Khz

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};


void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);
  DSK6713_AIC23_setFreq(hCodec, 3);

  while(1)
  {
     while(!DSK6713_AIC23_read(hCodec, &l_input));

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

           l_output=IIR_FILTER(&filter_coeff ,l_input);
                r_output=l_output;

           while(!DSK6713_AIC23_write(hCodec, l_output));

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

        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};

        static signed int y[6] = {0,0,0,0,0,0};

        int temp=0;

        temp = (short int)x1;

        x[0] = (signed int) temp;

        temp = ( (int)h[0] * x[0]);

        temp += ( (int)h[1] * x[1]);
        temp += ( (int)h[1] * x[1]);
        temp += ( (int)h[2] * x[2]);

        temp -= ( (int)h[4] * y[1]);
        temp -= ( (int)h[4] * y[1]);
        temp -= ( (int)h[5] * y[2]);

   temp >>=15;

  if ( temp > 32767 )
  {
   temp = 32767;
  }
   else if ( temp < -32767)
   {
    temp = -32767;
   }
   y[0] = temp;
   y[2] = y[1];
   y[1] = y[0];

   x[2] = x[1];
   x[1] = x[0];

   return (temp<<2);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 7-11KHz, and attenuate at other than pass band frequency.

2. Butterworth(Low pass)

PROGRAM CODE:

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

//const signed int filter_coeff[] = {2366,2366,2366,32767,-18179,13046};
              //IIR_BUTTERWORTH_LP FILTER pass band range 0-2.5kHZ
//const signed int filter_coeff[] = {312,312,312,32767,-27943,24367};
             //IIR_BUTTERWORTH_LP FILTER pass band range 0-800Hz
const signed int filter_coeff[] = {15241,15241,15241,32761,10161,7877};
                 //IIR_BUTERWORTH_LP FILTER pass band range 0-8kHz



DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};


void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 3);

  while(1)
  {
       while(!DSK6713_AIC23_read(hCodec, &l_input));
```

```
                while(!DSK6713_AIC23_read(hCodec, &r_input));

                 l_output=IIR_FILTER(&filter_coeff ,l_input);
                        r_output=l_output;

                  while(!DSK6713_AIC23_write(hCodec, l_output));

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

            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};

            static signed int y[6] = {0,0,0,0,0,0};

            int temp=0;

            temp = (short int)x1;

            x[0] = (signed int) temp;

            temp = ( (int)h[0] * x[0]);

            temp += ( (int)h[1] * x[1]);
            temp += ( (int)h[1] * x[1]);
            temp += ( (int)h[2] * x[2]);

            temp -= ( (int)h[4] * y[1]);
            temp -= ( (int)h[4] * y[1]);
            temp -= ( (int)h[5] * y[2]);

   temp >>=15;

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

   y[2] = y[1];
   y[1] = y[0];

   x[2] = x[1];
   x[1] = x[0];

   return (temp<<2);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 0-8KHz, and attenuate at other than pass band frequency.

3. Chebyshev (High pass)

PROGRAM CODE:

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

//const signed int filter_coeff[] = {12730,-12730,12730,32767,-18324,21137};
                         //IIR_CHEB_HP FILTER pass band range 2.5kHz-11KHz
//const signed int filter_coeff[] = {9268,-9268,9268,32767,-7395,18367};
                         //IIR_CHEB_HP FILTER pass band range 4kHz-11KHz
const signed int filter_coeff[] = { 3842,-3842,3842,32767,12360,19289};
                          //IIR_CHEB_HP FILTER pass band range 7kz-11KHz


DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};


void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 3);

  while(1)
  {
      while(!DSK6713_AIC23_read(hCodec, &l_input));

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

                l_output=IIR_FILTER(&filter_coeff ,l_input);
                       r_output=l_output;
```

```
                    while(!DSK6713_AIC23_write(hCodec, l_output));

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

             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};

             static signed int y[6] = {0,0,0,0,0,0};

             int temp=0;

             temp = (short int)x1;

             x[0] = (signed int) temp;

             temp = ( (int)h[0] * x[0]);

             temp += ( (int)h[1] * x[1]);
             temp += ( (int)h[1] * x[1]);
             temp += ( (int)h[2] * x[2]);

             temp -= ( (int)h[4] * y[1]);
             temp -= ( (int)h[4] * y[1]);
             temp -= ( (int)h[5] * y[2]);


    temp >>=15;

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

   y[2] = y[1];
   y[1] = y[0];

   x[2] = x[1];
   x[1] = x[0];

   return (temp<<2);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 7-11KHz, and attenuate at other than pass band frequency.

4. Chebyshev (Low pass)

PROGRAM CODE:

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

//const signed int filter_coeff[] = {1455,1455,1455,32767,-23410,21735};
                    //IIR_CHEB_LP FILTER pass band range 0-2.5kHz
//const signed int filter_coeff[] = {168,168,168,32767,-30225,28637};
                      //IIR_CHEB_LP FILTER pass band range 0-800Hz
const signed int filter_coeff[] = {11617,11617,11617,32767,8683,15506};
                       //IIR_CHEB_LP FILTER pass band range 0-8kHz

DSK6713_AIC23_Config
config={0x0017,0x0017,0x00d8,0x00d8,0x0011,0x0000,0x0000,0x0043,0x0081,0x0001};


void main( )
{
  DSK6713_AIC23_CodecHandle hCodec;
  Uint32 l_input, r_input,l_output, r_output;
  DSK6713_init();
  hCodec = DSK6713_AIC23_openCodec(0, &config);

  DSK6713_AIC23_setFreq(hCodec, 3);

  while(1)
  {
       while(!DSK6713_AIC23_read(hCodec, &l_input));

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

                  l_output=IIR_FILTER(&filter_coeff ,l_input);
                           r_output=l_output;

                   while(!DSK6713_AIC23_write(hCodec, l_output));

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

```
            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};

            static signed int y[6] = {0,0,0,0,0,0};

            int temp=0;

            temp = (short int)x1;

            x[0] = (signed int) temp;

            temp = ( (int)h[0] * x[0]);

            temp += ( (int)h[1] * x[1]);
            temp += ( (int)h[1] * x[1]);
            temp += ( (int)h[2] * x[2]);

            temp -= ( (int)h[4] * y[1]);
            temp -= ( (int)h[4] * y[1]);
            temp -= ( (int)h[5] * y[2]);


    temp >>=15;

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

   y[2] = y[1];
   y[1] = y[0];

   x[2] = x[1];
   x[1] = x[0];

   return (temp<<2);
}
```

Expected Output: Observed the output in CRO with respect to input signal, it pass the signal at range of pass band frequency 0-8KHz, and attenuate at other than pass band frequency.

5. N-POINT FAST FOURIER TRANSFORM (FFT) ALGORITHM [USING TMS320C6713 DSP PROCESSOR].

AIM: To find the DFT of a sequence using FFT algorithm**.**

EQUIPMENTS NEEDED:

Host (PC) with windows(95/98/Me/XP/NT/2000).

TMS320C6713 DSP Starter Kit (DSK).

Oscilloscope and Function generator.

ALGORITHM TO IMPLEMENT FFT:

- **Step 1 -** Select no. of points for FFT (Eg: 64).
- **Step 2 –** Generate a sine wave of frequency 'f ' (eg: 10 Hz with a sampling rate = No. of Points of FFT(eg. 64)) using **math library function**.
- **Step 3 -** Take sampled data and apply FFT algorithm .
- **Step 4 –** Use Graph option to view the Input & Output.
- **Step 5 -** Repeat **Step-1 to 4** for different no. of points & frequencies.

C PROGRAM TO IMPLEMENT FFT :

Main.c (fft 256.c):

```
#include <math.h>
#define PTS 64                    //# of points for FFT
#define PI 3.14159265358979

typedef struct {float real,imag;} COMPLEX;
```

```
void FFT(COMPLEX *Y, int n);          //FFT prototype
float iobuffer[PTS];                    //as input and output buffer
float x1[PTS];                          //intermediate buffer
short i;                        //general purpose index
variable short buffercount = 0;    //number of new
samples in iobuffer
short flag = 0;                         //set to 1 by ISR when
iobuffer full COMPLEX w[PTS];           //twiddle constants
stored in w COMPLEX samples[PTS];             //primary
working buffer

main()
{
  for (i = 0 ; i<PTS ; i++)// set up twiddle constants in w
  {
  w[i].real = cos(2*PI*i/(PTS*2.0)); //Re component of twiddle
  constants w[i].imag =-sin(2*PI*i/(PTS*2.0)); //Im component
  of twiddle constants
  }
  for (i = 0 ; i < PTS ; i++)    //swap buffers
   {

  iobuffer[i] = sin(2*PI*10*i/64.0);/*10- >
  freq,
   samples[i].real=0.0;
   samples[i].imag=0.0;
   }
 for (i = 0 ; i < PTS ; i++)    //swap buffers
   {
   samples[i].real=iobuffer[i]; //buffer with
   new data
   }
  for (i = 0 ; i < PTS ; i++)
   samples[i].imag = 0.0;         //imag
   components = 0

FFT(samples,PTS);       //call function
```

```
FFT.c for (i = 0 ; i < PTS ; i++)

//compute magnitude
{
 x1[i] =
 sqrt(samples[i].real*samples[i].
 real
          +
          samples[i].imag*sample
          s[i].imag);
```

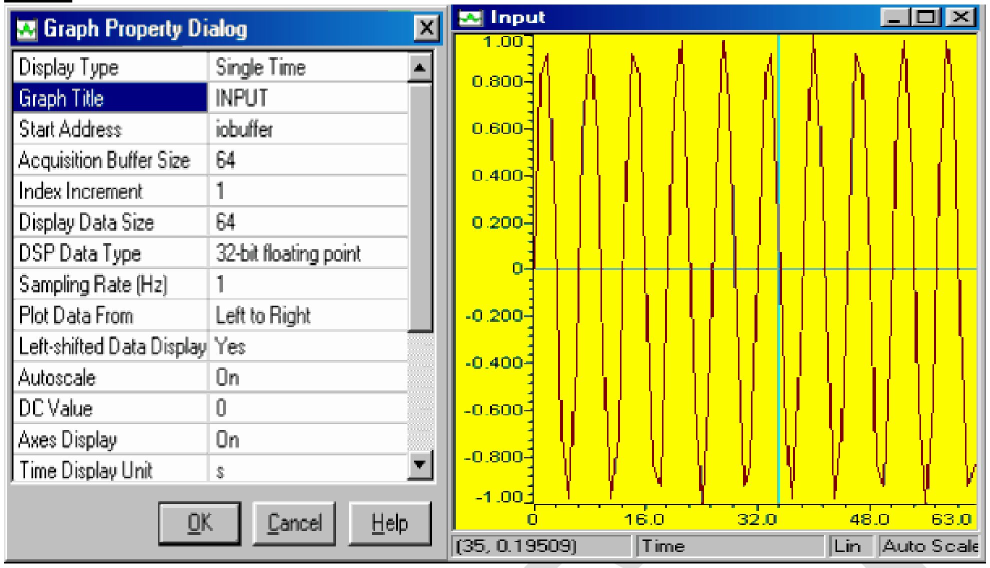
```
                                                                  64 -> sampling freq*/
    }
}                                         //end of main
```

fft.c:

```
#define PTS 64                        //# of points for FFT
typedef struct {float real,imag;} COMPLEX;
extern COMPLEX w[PTS];       //twiddle constants stored in w

void FFT(COMPLEX *Y, int N)   //input sample array, # of points
{
 COMPLEX temp1,temp2;  //temporary storage
 variables int i,j,k;               //loop counter variables
 int upper_leg, lower_leg;        //index of upper/lower
 butterfly leg int leg_diff;        //difference between
 upper/lower leg
 int num_stages = 0;    //number of FFT stages (iterations)
 int index, step;            //index/step through twiddle
 constant i = 1; //log(base2) of N points= # of stages
 do
  {
   num_stages +=1;
   i = i*2;
  }while (i!=N);
 leg_diff = N/2;             //difference between
 upper&lower legs step = (PTS*2)/N;   //step between
 values in twiddle.h
 for (i = 0;i < num_stages; i++)  //for N-point FFT
  {
   index = 0;
   for (j = 0; j < leg_diff; j++)
    {
  for (upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
   {
      lower_leg = upper_leg+leg_diff;
            temp1.real = (Y[upper_leg]).real +
      (Y[lower_leg]).real; temp1.imag =
      (Y[upper_leg]).imag + (Y[lower_leg]).imag;
      temp2.real = (Y[upper_leg]).real -
      (Y[lower_leg]).real; temp2.imag =
      (Y[upper_leg]).imag - (Y[lower_leg]).imag;
      (Y[lower_leg]).real = temp2.real*(w[index]).real
                                                -
      temp2.imag*(w[index]).imag; (Y[lower_leg]).imag =
      temp2.real*(w[index]).imag
                                              +temp2.imag*(w[index]).r
                                              eal;
      (Y[upper_leg]).real =
      temp1.real;
      (Y[upper_leg]).imag =
      temp1.imag;
    }
    index += step;
     }
     leg_diff = leg_diff/2;
     step *= 2;
      }
      j = 0;
      for (i = 1; i < (N-1); i++)    //bit reversal for resequencing data
      {
     k = N/2;
     while (k <= j)
      {
      j = j - k;
      k = k/2;
      }
     j = j + k;
  if
  (i<j)
       {
      temp1.real =
      (Y[j]).real;
      temp1.imag =
      (Y[j]).imag; (Y[j]).real
      = (Y[i]).real;
      (Y[j]).imag =
      (Y[i]).imag; (Y[i]).real
      = temp1.real;
      (Y[i]).imag =
        temp1.imag;
        }
       }
      return;
    }
```

RESULT:

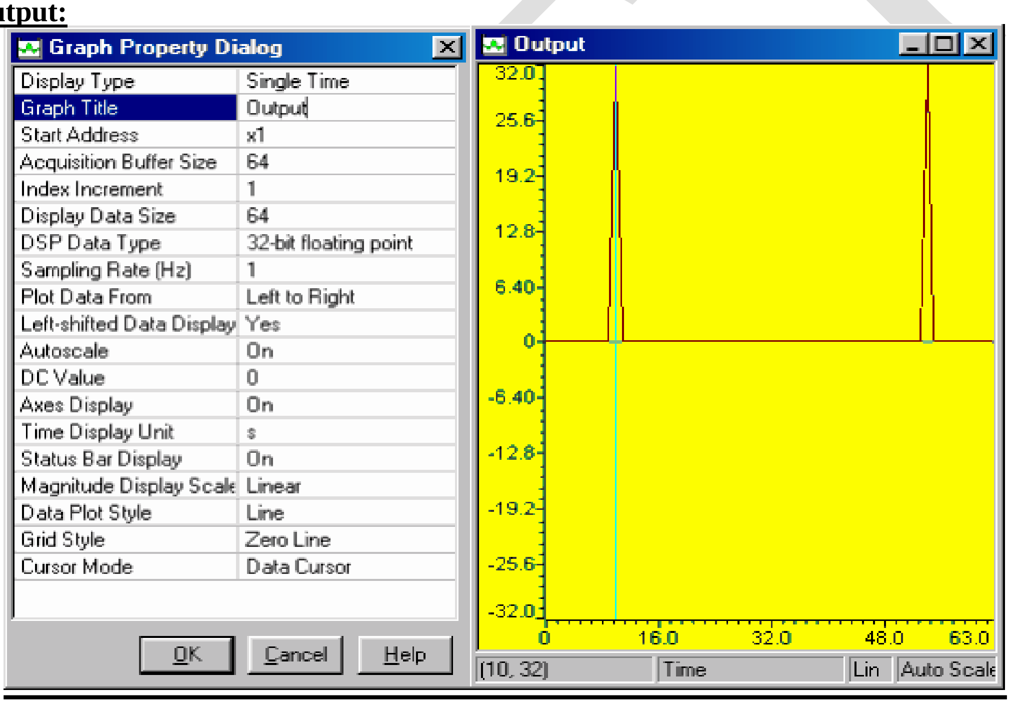
Input:



Graph Property Dialog

Display Type	Single Time
Graph Title	INPUT
Start Address	iobuffer
Acquisition Buffer Size	64
Index Increment	1
Display Data Size	64
DSP Data Type	32-bit floating point
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	s

Output:



Graph Property Dialog

Display Type	Single Time
Graph Title	Output
Start Address	x1
Acquisition Buffer Size	64
Index Increment	1
Display Data Size	64
DSP Data Type	32-bit floating point
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	s
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Line
Grid Style	Zero Line
Cursor Mode	Data Cursor

17. IMPULSE RESPONSE OF FIRST ORDER AND SECOND ORDER SYSTEM

```
/* Impulse response of the system defined by the difference equation */
//x(n) = y(n)-y(n-1)+0.9y(n-2)
/*Impulse response of the system
y[n] + a1 y[n-1] + a2 y[n-2] + .. = b0 x[n] + b1 x[n-1] + b2 y[n-2] + ..
Example :
1 y[n] + 1 y[n-1] + 1 y[n-2] = 1 x[n] + 2 x[n-1] + 1 y[n-2]
*/
```

12.IMPULSE RESPONSE USING TMS320C6713 KIT

Aim: To obtain impulse response.

Apparatus: DSK 6713 DSP Trainer kit. USB Cable Power supply

Program:

```
#include <stdio.h>
#define Order    2      /*Order of the system*/
#define Len 10          /*Length of the output pulses*/

float y[Len]={0,0,0},sum;
main()
{
int j,k;
float a[Order+1]={0.1311, 0.2622, 0.1311};
/* y coefficient may change in accordance with the difference equation */

float b[Order+1]={1, -0.7478, 0.2722};
/* x coefficients may change in accordance with the difference equation */

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

```
sum=0;
for(k=1;k<=Order;k++)
            {
if((j-k)>=0)
sum=sum+(b[k]*y[j-k]);
}
if(j<=Order)
{
y[j]=a[j]-sum;
}
else
{
y[j]=-sum;
}
printf("Respose[%d] = %f\n",j,y[j]);
}
}
```

Procedure:

1 Open code composer studio, make sure the DSP kit is turned on. Start a new project using 'project-new' pull down menu, save it in a separate directory (c:\ccstudio\myprojects) with the name 'file name. pjt'.

2 Add the source file of linear convolution to the project using 'project-add files to project' pull down menu.

3 Add the linker command file 'hello. cmd'(c\ccstudio\tutorials\dsk6713\hello1\hello.cmd)

4 Add the run time support library file rts6700.lib (c-ccstudio\c6000\cgtools\lib\rts6700.lib)

5 Compile program using the 'project-compile' pull down menu or by clicking the short cut icon on the left side of program window.

6 Build the program using ‘project-build’ pull down menu or by clicking the icon on the left side of the program window.

7 Load the program in program memory of DSP chip using the ‘file-load program’ pull down menu.

8 To view o/p graphically, select View -graph-time and frequency.

Precautions:

1 Switch ON the computer only after connecting USB cable and make sure the DSP kit is ON.

2 Perform the diagnostic check before opening code composer studio.

3 All the connections must be tight.

Result : The impulse response is obtained and the graphs are plotted.

Viva questions:

1. Define impulse response.
2. Give me one example for impulse response.
3. Write the Formula for impulse response.
4. What are major role in order & length?

2.COMPUTATION OF N- POINT DFT OF A GIVEN SEQUENCE

```
#include<stdio.h>
#include<math.h>

void main()
{
short N = 8;
short x[8] = {1,2,3,4,5,6,7,0}; // test data float pi = 3.1416;
float sumRe = 0, sumIm = 0; // init real/imag components
float cosine = 0, sine = 0; // Initialise cosine/sine components

// Output Real and Imaginary components

float out_real[8] = {0.0}, out_imag[8] = {0.0}; int n = 0, 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_real[k] = sumRe;
out_imag[k] = sumIm;
printf("[%d] %7.3f %7.3f \n", k, out_real[k], out_imag[k]);
}

}
```

Output

```
[0]     28.000  0.000
[1]     -9.657  4.000
[2]     -4.000  -4.000
[3]     1.657   -4.000
[4]     4.000   -0.000
[5]     1.657   4.000
[6]     -4.000  4.000
[7]     -9.657  -3.999
```

Verification in matlab

```
x = [1,2,3,4,5,6,7,0] fft(x)
```

Output

```
Columns 1 through 4

28.0000-3.5000 + 7.2678i         -3.5000 + 2.7912i         -3.5000 + 0.7989i

Columns 5 through 7

-3.5000 - 0.7989i          -3.5000 - 2.7912i          -3.5000 - 7.2678i
```

5. POWER SPECTRUM DENSITY (PSD) USING TMS320C6713 KIT

AIM: To compute power density spectrum of a sequence

Apparatus: DSK 6713 DSP Trainer kit. USB Cable Power supply

PROGRAM:

```
# include <math.h>
#define PTS128//# of points for FFT
#define PI 3.14159265358979
typedef struct    {float real,imag;}COMPLEX
 void FFT(COMPLEX Y, int n); float iobuffer[PTS];
float x1[PTS],x[PTS]; short i;
short buffercount = 0; shaort flag = 0;
COMPLEX w [PTS];
w*/
COMPLEX samples[PTS];
main( )
{
float j,sum=0.0; int n,k,i,a;
for (I = 0 ; i<PTS ; i++)
            {
 /*FFT prototype*/
```

```
/*as input and output buffer*/
            /*intermediate buffer*/
/*general purpose index variable */
/*number of new samples in iobuffer*/
/*set to 1 by ISR when iobuffer full*/
            /*twiddle constants stored in
/*primary working buffer*/
/*set up twiddle constants in w */
 w[i].real = cos(2*PI*i/(PTS*2.0)); /*Re component of twiddle constants*/
w[i].imag =-sin(2*PI*i/(PTS*2.0)); /*Im component of twiddle constants*/
}
/*Input signal X(n) */
for(i=0,j=0;i<PTS;i++)
{
x[i] = sin(2*PI*5*i/PTS);          /*Signal x(Fs)=sin(2*PI*f*i/Fs);*/
samples[i].real=0.0;
samples[i].imag=0.0;
}
/*Auto Correlation of X(n)=R(t) */
for(n=0;n<PTS;n++)
{
sum=0;
for(k=0;k<PTS-n;k++)
{
sum=sum+(x[k]*x[n+k]);
}
```

```
iobuffer[n] = sum;
}
/*FFT of R(t) */
for(i = 0 ; i < PTS ; i++)
        {
 /*Auto Correlation R(t)*/
/*swap buffers*/
 samples[i].real=iobuffer[i]; /*buffer with new data*/
}
for(i = 0 ; I < PTS ; i++)
 samples[i].imag = 0.0; FFT(samples,PTS);
/*Power Spectral Density */
for (i = 0 ; i < PTS ; i++)
        {
 /*imag components = 0*/ /*call function FFT.c*/
/*compute magnitude*/
 x1[i] = sqrt(samples[i].real*samples[i].real
+samples[i].imag*samples[i].imag);
}
}       /*end of main*/
```

FFT SUBPROGRAM

```
#define PTS 64 /*number of points for FFT*/
typedef struct {float real,imag; }COMPLEX;
extern COMPLEX w[PTS];     /*twiddle constants stored in w*/
void FFT(COMPLEX Y, int  N) /*input sample array, # of points*/
{
```

```
COMPLEX temp1,temp2;          /*Temporary storage variables*/
int i,j,k; /*loop counter variables */
int upper_leg, lower_leg; int leg_diff;
int num_stages = 0; int index, step;
i = 1;
do
            {
num_stages += 1;
            i= i*2;
}
while (i! = N);
            leg_diff = N/2;
lower legs*/
step = (PTS*2)/N;
twiddle.h */
 /*index of upper/lower butterfly leg*/
/*difference between upper/lower leg */
/* number of FFT stages (iterations)  */
            /*index/step through twiddle constant */
/* log(base2) of N points= # of stages */
/*          difference between upper &
/* step between values in
 for (i= 0;i < num_stages; i++)
            {
index = 0;
for (j = 0; j < leg_diff; j++)
```

```
            {


/*for N-point FFT*/
 for( upper_leg = j; upper_leg < N; upper_leg += (2*leg_diff))
{
lower_leg = upper_leg+leg_diff;
temp1.real = (Y[upper_leg]).real + (Y[lower_leg]).real;
temp1.imag = (Y[upper_leg]).imag + (Y[lower_leg]).imag;
temp2.real = (Y[upper_leg]).real - (y[lower_leg]).real;
temp2.imag = (Y[upper_leg]).imag - (Y[lower_leg]).imag;
(Y[lower_leg]).real = temp2.real*(w[index]).real
-temp2.imag*(w[index]).imag;
(Y[lower_leg]).imag = temp2.real*(w[index]).imag
+temp2.imag*(w[index]).real;
(Y[upper_leg]).real = temp1.real;
(Y[upper_leg]).imag = temp1.imag;
}
index += step;
}
leg_diff = leg_diff/2; step *= 2;
}
j           = 0;
for (i= 1 ; i< (N-1 ); i++)            /*bit reversal for resequencing data*/
{
k = N/2;
while (k <= j)
```

```
{
j= j - k;
k = k/2;
}
j= j + k;
if (i<j)
{
temp1.real = (Y[j]).real;
temp1.imag = (Y[j]).imag;
(Y[j]).real = (Y[i]).real;
(Y[j]).imag = (Y[i]).imag;
(Y[i]).real = temp1.real;
(Y(i]).imag = temp1.imag;
}
}
return;
}
```

PROCEDURE:

1. Open code composer studio, make sure the DSP kit is turned on.
2. Start a new project using 'project-new' pull down menu, save it in a separate directory (c:\ccstudio\myprojects) with the name 'file name. pjt'.
3. Add the source file of linear convolution to the project using 'project-add files to project' pull down menu.
4. Add the linker command file 'hello. cmd' (c\ccstudio\tutorials\dsk6713\hello1\hello.cmd)
5. Add the run time support library file rts6700.lib c-ccstudio\c6000\cgtools\lib\rts6700.lib)
6. Compile program using the 'project-compile' pull down menu or by clicking the short cut icon on the left side of program window.

7 Build the program using ‘project-build’ pull down menu or by clicking the icon on the left side of the program window.

8 Load the program in program memory of DSP chip using the ‘file-load program’ pull down menu.

9 To view o/p graphically, select View -graph-time and frequency.

PRECAUTIONS:

1 Switch ON the computer only after connecting USB cable and make sure the DSP kit is ON.

2 Perform the diagnostic check before opening code composer studio.

3 All the connections must be tight.

Result : The power density spectrum is obtained and the graphs are plotted.

VIVA QUESTIONS:

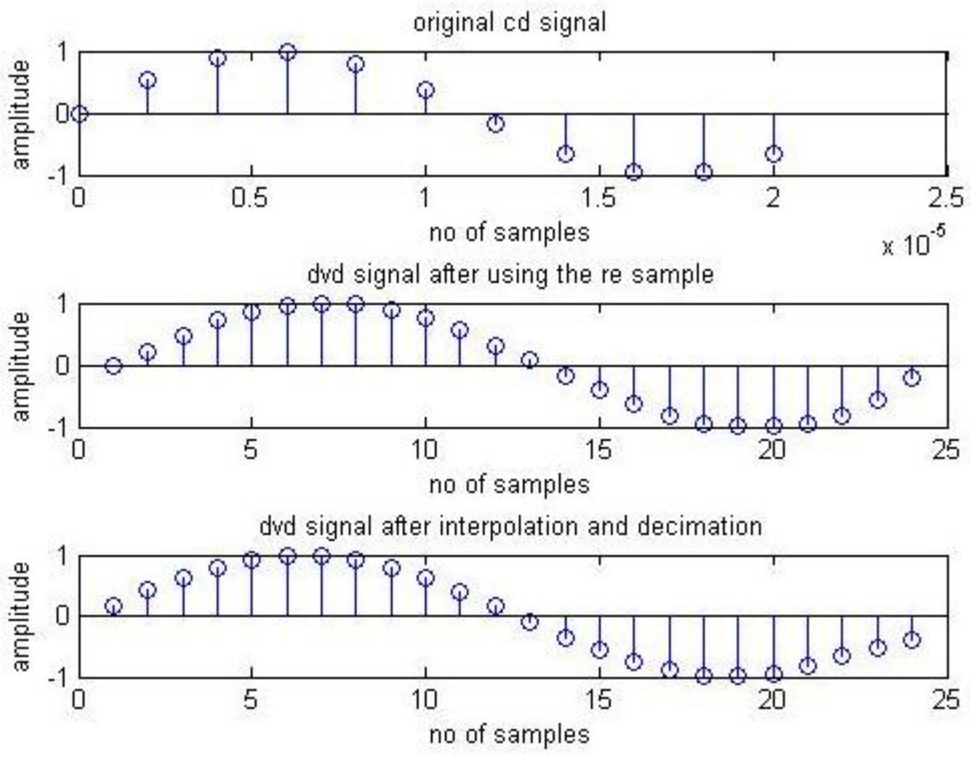
1. Define power spectral Density?
2. What is the need for spectral estimation?
3. Determine the power spectrum density?
4. What is the relation between auto correlation & spectral density?
5. Give the estimation of auto correlation function & power density for random Signals?
6. Explain power spectrum estimation using the Bartlett window?
7. Give the formula for PSD?

ADDITIONAL PROGRAMS

1. CD DATA TO DVD DATA

clc;

```
clear all;
fc=44100;
t=0:0.000002:0.00002;
x=sin(2*pi*fc*t);
subplot(3,1,1);
stem(t,x);
title('original cd signal');
xlabel('no of samples');
ylabel('amplitude');
i=13;
d=6;
y=resample(x,i,d);
subplot(3,1,2);
stem(y);
title('dvd signal after using the re sample');
xlabel('no of samples');
ylabel('amplitude');
in=interp(x,i);
de=decimate(in,d);
subplot(3,1,3);
stem(de);
title('dvd signal after interpolation and decimation');
xlabel('no of samples');
ylabel('amplitude');
```



2. ESTIMATION OF PSD NOISY SIGNAL

```
clc ;
clear all;
close all;
f1=input('enter the frequency of first sinosuidal signal');
f2=input('enter the frequency of second sinosuidal signal');
fs=input('enter the frequency of sampling signal');
%generation of input sequence
t=0:1/fs:1;
x=2*sin(2*pi*f1*t)+3*sin(2*pi*f2*t)-rand(size(t));
%generation of psd for 2 different fft lengths
pxx1=(abs(fft(x(1:256))).^2+abs(fft(x(257:512))).^2+abs(fft(x(257:512))).^2+abs(fft(x(513:768)))).^2/(2
56*6);
pxx2=(abs(fft(x(1:256))).^2+abs(fft(x(129:384))).^2+abs(fft(x(257:512))).^2+abs(fft(x(385:640))).^2+ab
s(fft(x(513:768))).^2+abs(fft(x(641:896)))).^2/(256*6);
figure(1);
plot((0:255)/256*fs,10*log10(pxx1));
xlabel('frequency in hertz');
ylabel('power spectrum in db');
title('psd with non overlapping segment');
grid;
figure(2);
```

```
plot((0:255)/256*fs,10*log10(pxx1));
xlabel('frequency in hertz');
ylabel('power spectrum in db');
title('psd with non overlapping segment');
grid;
figure(3);
plot((0:255)/256*fs,10*log10(pxx2));
xlabel('frequency in hertz');
ylabel('power spectrum in db');
title('psd with overlapping segment');
grid;
```