



SVR ENGINEERING COLLEGE

Approved by AICTE & Permanently Affiliated to JNTUA

Ayyalurmetta, Nandyal – 518503. Website: www.svrec.ac.in

Department of Electronics and Communication Engineering



DIGITAL SIGNAL PROCESSING LABORATORY MANUAL

III B.Tech (ECE) II Semester (R - 19)

2021-2022



INDEX

<u>S.No.</u>	<u>Name of the Experiment</u>	<u>Page No.</u>
	The following experiments shall be conducted using MATLAB / Lab View / CProgramming/ Equivalent software.(PART A)	
1	Generation of sinusoidal waveform / signal based on recursive difference equations	1-5
2	Find DFT / IDFT of given discrete time signal.	6-9
3	Find frequency response of a system given in transfer function/ differential equation form.	10-11
4	Implementation of FFT of given Sequence.	12-15
5	Design and implementation of IIR filter using bilinear transformation and impulse invariantmethod.	16-18
6	Design and implementationof IIR Butterworth (LP/HP) filter.	19-20
7	Design and implementationof IIR Chebyshev(LP/HP) filter.	21-22
8	Design and implementation of FIR with low pass filter using any three windowingtechniques. Plot its magnitude and phase responses.	23-26
9	Design and implementation of FIR filter with high passfilter using any three windowingtechniques. Plot its magnitude and phase responses.	27-30
10	Design and implementation of FIR filter with band pass / band stopfilter using any three windowing techniques. Plot its magnitude and phase responses.	31-38
	The following experiments shall be conducted using (TI / Analog Devices / Motorola /Equivalent DSP processors). (PART – B)	
1	Study the architecture of DSP chips – TMS 320C 5X/6X Instructions.	53-61
2	Find DFT / IDFT of given discrete time signal.	62-63
3	Implementation of FFT of given Sequence.	64-67
4	Design and implementation of IIR Butterworth / Chebyshev (LP/HP) filter.	68-70
5	Design and implementation of FIR with low pass / high pass filter using any three windowingtechniques. Plot its magnitude and phase responses.	71-72

ECE DEPT VISION & MISSION PEOs and PSOs

Vision: To produce highly skilled, creative and competitive Electronics and Communication Engineers to meet the emerging needs of the society.

Mission:

- Impart core knowledge and necessary skills in Electronics and Communication Engineering through innovative teaching and learning.
- Inculcate critical thinking, ethics, lifelong learning and creativity needed for industry and society
- Cultivate the students with all-round competencies, for career, higher education and self-employability

I. PROGRAMME EDUCATIONAL OBJECTIVES (PEOS)

- PEO1: Graduates apply their knowledge of mathematics and science to identify, analyze and solve problems in the field of Electronics and develop sophisticated communication systems.
- PEO2: Graduates embody a commitment to professional ethics, diversity and social awareness in their professional career.
- PEO3: Graduates exhibit a desire for life-long learning through technical training and professional activities.

II. PROGRAM SPECIFIC OUTCOMES (PSOS)

- PSO1: Apply the fundamental concepts of electronics and communication engineering to design a variety of components and systems for applications including signal processing, image processing, communication, networking, embedded systems, VLSI and control system
- PSO2: Select and apply cutting-edge engineering hardware and software tools to solve complex Electronics and Communication Engineering problems.

III. PROGRAMME OUTCOMES (PO'S)

- 1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- 2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- 3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- 4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- 5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

- 6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- 7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- 8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- 9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- 10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- 11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- 12. Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

IV. COURSE OBJECTIVES

Students can learn the basics of using DSP chips to perform real-time digital signal processing.

Ability to apply knowledge of mathematics, science and engineering: Construction of tools for visualizing the basic concepts of discrete signal representation such as Fourier transforms, discrete time representations.

Students will learn numerous programming tools for design and implementations of filtering algorithms.

Understand the concept of Multi-rate signal processing and sample rate conversion.

Develop and Implement DSP algorithms in software using CCS with DSP floating pointProcessor.

V. COURSE OUTCOMES

After the completion of the course students will be able to

Course Outcomes	Course Outcome statements	BTL
CO1	Ability to design-test, to verify, to evaluate, and to benchmark a real-time DSP system.	L1
CO2	Ability to calculate discrete time domain and frequency domain of signals using discreteFourier series and Fourier transform.	L3
CO3	Ability to design, using MATLAB-based filter design techniques, FIR and IIR digital filters and Determine the frequency response of filters.	L4
CO4	Implementation of basic signal processing algorithms such as convolution, difference equation implementation and application of them in the construction of FIR and IIR filters.	L2
CO5	Design DSP based real time processing systems to meet desired needs of the society.	L5

VI.COURSE MAPPING WITH PO'S AND PEO'S

Course Title	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PE 01	PE 02
DSP Lab	2.2	2.6	2.2	2.4	2.6	2.2	1.4	2	2.0	2.2	2.2	2.6	2.6	2.4

V MAPPING OF COURSE OUTCOMES WITH PEO'S AND PO'S

Course Title	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 11	PO 12	PE 01	PE 02
CO1	2	3	2	2	3	2	2	1	3	3	3	3	3	3
CO2	1	2	3	2	2	2	1	2	1	2	3	2	2	3
CO3	3	3	2	3	2	3	2	2	2	1	1	3	2	2
CO4	3	2	2	2	3	2	1	3	2	2	2	2	3	2
CO5	2	3	2	3	3	2	1	2	2	3	2	3	3	2

LABORATORY INSTRUCTIONS

1. While entering the Laboratory, the students should follow the dress code. (Wear shoes and White apron, Female Students should tie their hair back).
2. The students should bring their observation book, record, calculator, necessary stationery items and graph sheets if any for the lab classes without which the students will not be allowed for doing the experiment.
3. All the Equipment and components should be handled with utmost care. Any breakage or damage will be charged.
4. If any damage or breakage is noticed, it should be reported to the concerned in charge immediately.
5. The theoretical calculations and the updated register values should be noted down in the observation book and should be corrected by the lab in-charge on the same day of the laboratory session.
6. Each experiment should be written in the record note book only after getting signature from the lab in-charge in the observation notebook.
7. Record book must be submitted in the successive lab session after completion of experiment.
8. 100% attendance should be maintained for the laboratory classes.

WORKING PROCEDURE WITH MATLAB:

1) Double click on Matlab icon. -> Then Matlab will be opened

2) To write the Matlab Program

Goto file menu-> New -> Script(Mfile) -> In the opened Script file write the Matlab code and save the file with an extension of .m

Ex: “linear.m”

3) To execute Matlab Program Select the all lines in matlab program(ctrl+A) of mfile and press “F9” to execute the matlab code

4) Entering the inputs in command window

☐ If the command window is displaying the message like “enter the input sequence” then

enter the sequence with square brackets and each sample values is spaced with single space

Ex: Enter input sequence [1 2 3 4] If it is asking a value input write the value without brackets

Ex: “enter length of sequence 4” After entering inputs It displays the Output Graphs.

PROCEDURE TO WORK ON CODE COMPOSER STUDIO

Test the USB port by running DSK Port test from the start menu

Use Start ☐ Programs ☐ Texas Instruments ☐ Code Composer Studio ☐ Code Composer Studio

CDSK6713 Tools ☐ DSK6713 Diagnostic Utilities

☐ *Select ☐ Start ☐ Select DSK6713 Diagnostic Utility Icon from Desktop*

☐ *Select **Start** Option*

☐ *Utility Program will test the board*

☐ *After testing Diagnostic Status you will get **PASS***

To create the New Project

Project ☐ New (File Name. pj1 , Eg: **Vectors.pj1**)

To Create a Source file

File ☐ New ☐ Type the code (Save & give file name, Eg: **sum.c**).

To Add Source files to Project

Project ☐ Add files to Project ☐ c/ccs studio3.1/my projects/your project name/

sum.c(select the file type as c/c++ source files)

To Add rts.lib file & hello.cmd:

Project ☐ Add files to Project ☐ rts6700.lib

(Path:c/ccs studio3.1/cg tools/c6000/lib/ rts6700.lib)

Note: Select Object & Library in(*.o,*.l) in Type of files

Project ☐ Add files to Project ☐ hello.cmd

CMD file – Which is common for all non real time programs.

(Path: c/ccs studio3.1\tutorial\dsk 6713 \hello1\hello.cmd)

Note: Select Linker Command file(*.cmd) in Type of files

Compile:

To Compile: Project ☐ Compile project

To Build: Project ☐ build project,

To Rebuild: Project ☐ rebuild,

Which will create the final .out executable file.(Eg. Vectors.out).

Procedure to Load and Run program:

Load the program to DSK: File ☐ Load program ☐ Vectors. out

To Execute project: Debug ☐ Run.

1. Execution should halt at break point.
2. Now press F10. See the changes happening in the watch window.
3. Similarly go to view & select CPU registers to view the changes happening in CPU registers.

Configure the graphical window as shown below

INPUT

$x[n] = \{1, 2, 3, 4, 0, 0, 0, 0\}$ $h[k] = \{1, 2, 3, 4, 0, 0, 0, 0\}$

CONNECTING DSP PROCESSOR TO PC

- ☐ Connect the dsp processor to the pc using usb cable connector.
- ☐ Check the DSK6713 diagnostics (IF you get the “pass”then click on ok).
- ☐ Click on ccs studio3.1 desk top icon. Then the window will be opened.
- ☐ Go to debug click on connect (then target device will be connected to pc)

TO CREATE PROJECT

- ☐ Project new given project name and select the family'TMS320C67XX'Then click ok
- ☐ File new source file write down the 'c'program and save it with.'c' extension in current project file
- ☐ File new dsp/bios.config file select dsk67xx click on dsk6713 and save it in current project.
- ☐ Project add files to project add source file
- ☐ Project add files to project add library file by following the given path

- ☐ *c/ccs studio3.1/cgtools/c6000/dsk6713/DSK6713.bs/file.*
- ☐ Project add files to the project .Add the configuration file.
- ☐ Now files are generated and included in generated files . in that open the 3rd file, and copy the header file and paste it in source file. Copy the include files named as "**dsk6713.h**" and "**dsk6713_aic23.h**" paste it in current project folder.
- ☐ Now compile project.(project compile)
- ☐ Project build. Project rebuild all.
- ☐ File load program project name.pjt debug "project name .out" file click on open debug click on run
- ☐ Now apply the input sine wave to line in of dsk6713 kit.
- ☐ Observe the output at line out of dsk6713 by using CRO.

INTRODUCTION

MATLAB: MATLAB is a software package for high performance numerical computation and visualization provides an interactive environment with hundreds of built in functions for technical computation, graphics and animation .The MATLAB name stands for Matrix Laboratory.

MATLAB Windows : MATLAB works with through three basic windows

Command Window : This is the main window .it is characterized by MATLAB command prompt >> when you launch the application program MATLAB puts you in this window all commands including those for user-written programs ,are typed in this window at the MATLAB prompt

Graphics window: the output of all graphics commands typed in the command window are flushed to the graphics or figure window, a separate gray window with white background color the user can create as many windows as the system memory will allow

Edit window: This is where you write edit, create and save your own programs in files called M files.

Input-output:

MATLAB supports interactive computation taking the input from the screen and flushing, the output to the screen. In addition it can read input files and write output files

Data Type: the fundamental data –type in MATLAB is the array. It encompasses several distinct data objects- integers, real numbers, matrices, character strings, structures and cells. There is no need to declare variables as real or complex,

MATLAB automatically sets the variable to be real.

Dimensioning: Dimensioning is automatic in MATLAB. No dimension statements are required for vectors or arrays .we can find the dimensions of an existing matrix or a vector with the size and length commands.

Where to work in MATLAB?

All programs and commands can be entered either in the a) Command window

b) As an M file using Matlab editor

Note: Save all M files in the folder 'work' in the current directory. Otherwise you have to locate the file during compiling.

BASIC INSTRUCTIONS

T = 0: 1:10

This instruction indicates a vector T which as initial value 0 and final value 10 with an increment of 1 Therefore
T = [0 1 2 3 4 5 6 7 8 9 10]

F = 20: 1: 100 Therefore F = [20 21 22 23 24 100]

T = 0:1/pi: 1 Therefore T= [0, 0.3183, 0.6366, 0.9549]

zeros (1, 3)

The above instruction creates a vector of one row and three columns whose values are zero

Output= [0 0 0]

Syntax: w = conv(u,v)

Description: w = conv(u,v) convolves vectors u and v. Algebraically, convolution is the same operation as multiplying the polynomials whose coefficients are the elements of u and v.

Disp Syntax: disp(X)

Description: disp(X) displays an array, without printing the array name. If X contains a text string, the string is displayed. Another way to display an array on the screen is to type its name, but this prints a leading "X=," which is not always desirable. Note that disp does not display empty arrays.

xlabel

Syntax: **xlabel('string')**

Description: xlabel('string') labels the x-axis of the current axes.

ylabel

Syntax : **ylabel('string')**

Description: ylabel('string') labels the y-axis of the current axes.

Title

Syntax : **title('string')**

Description: title('string') outputs the string at the top and in the center of the current axes. .grid on

Syntax : **grid on** Description: grid on adds major grid lines to the current axes.

FFT

FFT(X) is the discrete Fourier transform (DFT) of vector X. For matrices, the FFT operation is applied to each column. For N-D arrays, the FFT operation operates on the first non-singleton dimension.

FFT(X,N) is the N-point FFT, padded with zeros if X has less than N points and truncated if it has more.

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

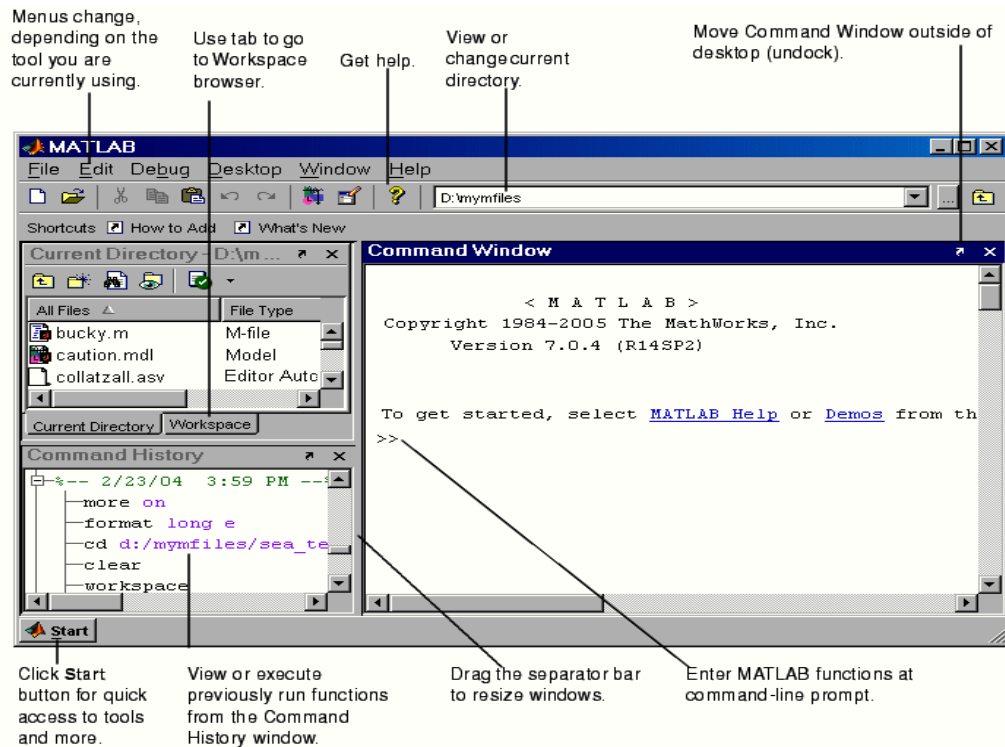


Figure : The graphical interface to the MATLAB workspace

When MATLAB is started for the first time, the screen looks like the one that shown in the Figure. 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.

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

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, $1 + 2 \times 3$. 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 $1 + 2 \times 3 = 7$. 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
```

3 Quitting MATLAB

To end your MATLAB session, type `quit` in the Command

Window, or select `FileMATLAB` in the desktop main menu.

1. GENERATION OF SINUSOIDAL WAVEFORM / SIGNAL BASED ON RECURSIVE DIFFERENCE EQUATIONS.

AIM: To generate sin signal and plot the same as a waveform showing all the specifications.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

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.

PROGRAM:

```
% Generation of signals and sequences
clc; clear all;

close all;

disp('SINE SIGNAL ');

N= input(' Enter Number of Samples : ');

n= 0:0.1:N;

x=sin(n) ;

subplot(2,1,1);

plot(n,x);

xlabel('Time') ;

ylabel(' Am plitude');

title('Continuous sine Signal');

subplot(2,1,2);

stem(n,x);

xlabel('Time') ;

ylabel(' Am plitude');

title('Discrete sine Signal');
```

```
%-----Generation of signals-----%
y5=sawtooth(2*pi*50*t,.5);
figure;
subplot(2,2,1);
plot(t,y5);
axis([0 0.1 -2 2]);
xlabel('time');
ylabel('amplitude');
title(' triangular wave signal');
%generation of triangular wave sequence
subplot(2,2,2);
stem(t,y5);
axis([0 0.1 -2 2]);
xlabel('n');
ylabel('amplitude');
title('triangular wave sequence');
%generation of sinsoidal wave signal
y6=sin(2*pi*40*t);
subplot(2,2,3);
plot(t,y6);
axis([0 0.1 -2 2]);
xlabel('time');
ylabel('amplitude');
title(' sinsoidal wave signal');
%generation of sin wave sequence
subplot(2,2,4);
stem(t,y6);
axis([0 0.1 -2 2]);
xlabel('n');
ylabel('amplitude');
title('sin wave sequence');
```

%To find the impulse response of the following difference equation

% $y(n)-y(n-1)+0.9y(n-2)=x(n)$

```
clc; clear all; close all;
```

```
disp('Difference Equation of a digital system');
```

```
N=input('Desired Impulse response length = ');
```

```
b=input('Coefficients of x[n] terms = ');
```

```
a=input('Coefficients of y[n] terms = ');
```

```
h=impz(b,a,N);
```

```
disp('Impulse response of the system is h = ');
```

```
disp(h);
```

```
n=0:1:N-1;
```

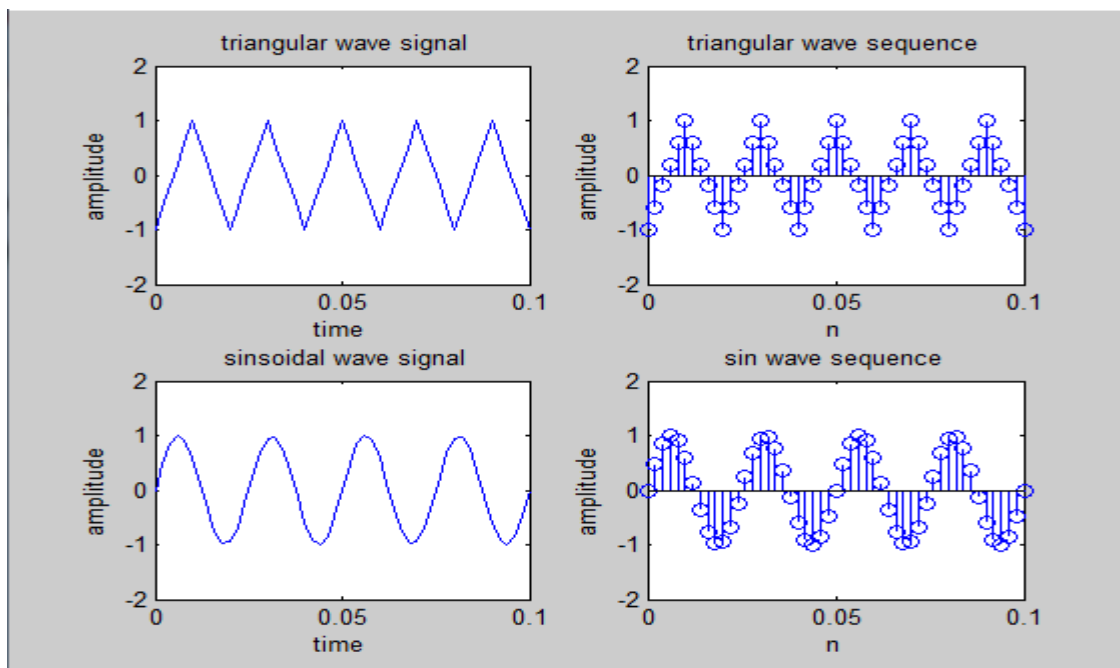
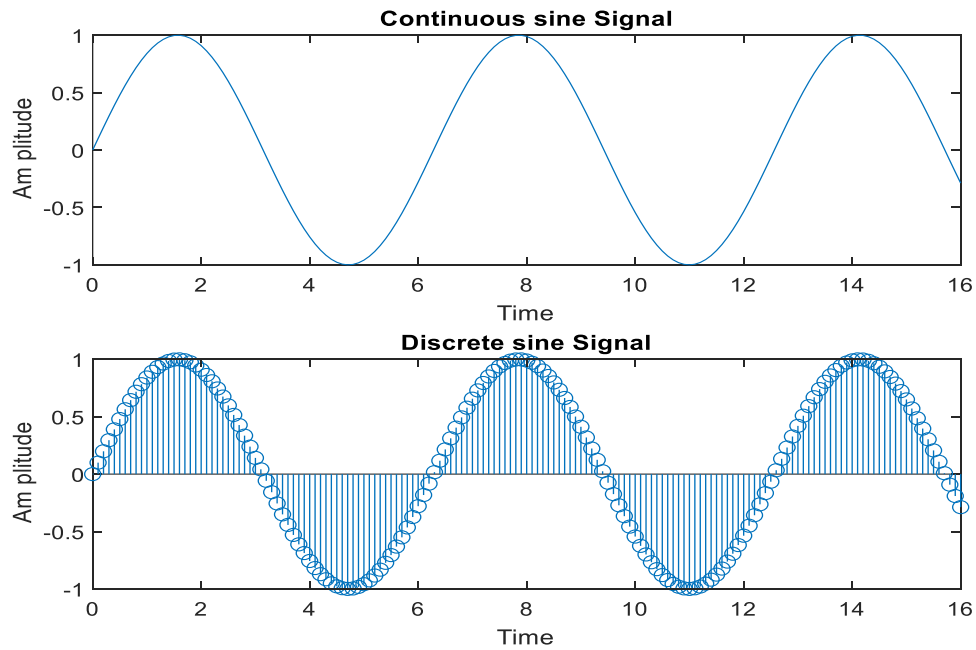
```
stem(n,h); xlabel('time index'); ylabel('h[n]');
```

```
title('Impulse response')
```

OUTPUT :

SINE SIGNAL

Enter Number of Samples : 16

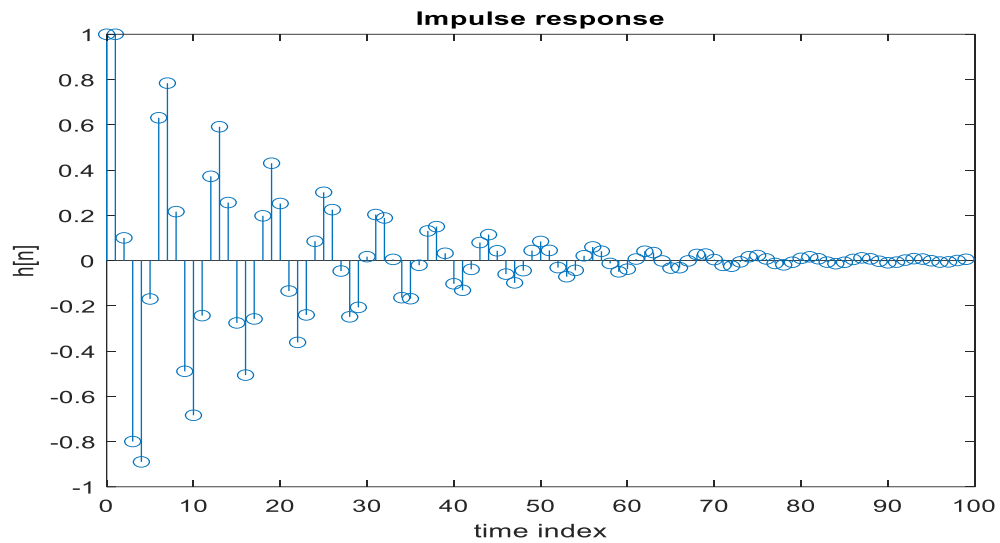


Difference Equation of a digital system

Desired Impulse response length = 100

Coefficients of $x[n]$ terms = 1

Coefficients of $y[n]$ terms = [1 -1 0.9]



RESULT: Various signals & sequences generated using Matlab software.

2) TO FIND DFT / IDFT OF GIVEN DT SIGNAL

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

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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) PROGRAM:WITH BUILTIN FUNCTION.

```
clc;
x1 = input('Enter the sequence:');
n = input('Enter the length:');
m = fft(x1,n);
disp('N-point DFT of a given sequence:');
disp(m);
N = 0:1:n-1; subplot(1,2,1);
stem(N,m);
xlabel('Length');
ylabel('Magnitude of X(k)');
title('Magnitude spectrum:');
an = angle(m);
subplot(1,2,2);
stem(N, an);
xlabel('Length');
ylabel('Phase of X(k)');
title('Phase spectrum:');
```

2) PROGRAM :WITHOUT BUILTIN FUNCTION.

```
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;
```

```
disp(Xk);  
MagX=abs(Xk) % Magnitude of calculated DFT  
PhaseX=angle(Xk)*180/pi % Phase of the calculated DFT  
figure(1);  
subplot(2,1,1);  
title('MAGNITUDE PLOT ');  
plot(k,MagX);  
subplot(2,1,2);  
plot(k,PhaseX)  
title('PHASE PLOT ')
```

OUTPUT-1 :-

Enter the sequence:[1 1 0 0]

Enter the length:4

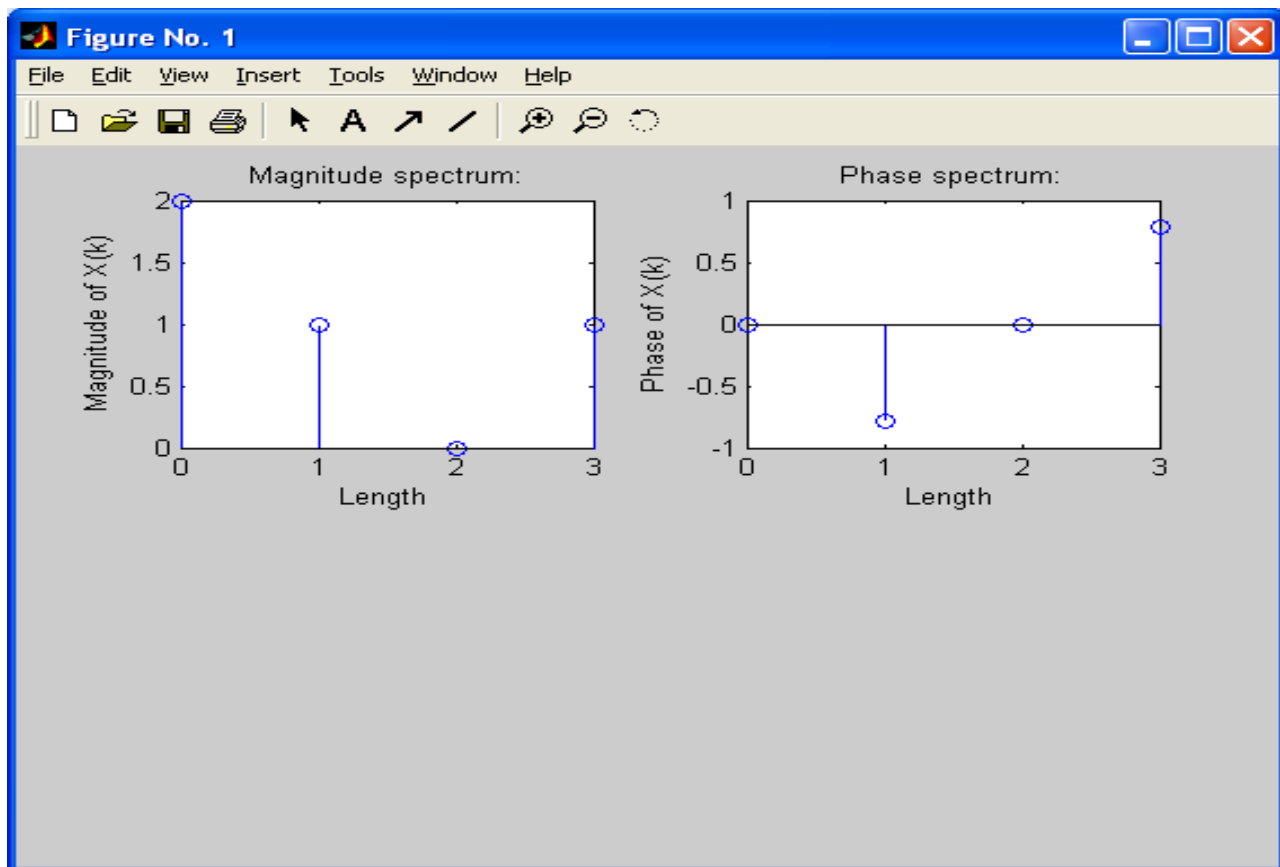
N-point DFT of a given sequence:

Columns 1 through 3

2.0000 1.0000 - 1.0000i 0

Column 4

1.0000 + 1.0000i



RESULT:- Thus Discrete Fourier Transform is Performed using Matlab.

OUTPUT -2 :

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 1 0 0]

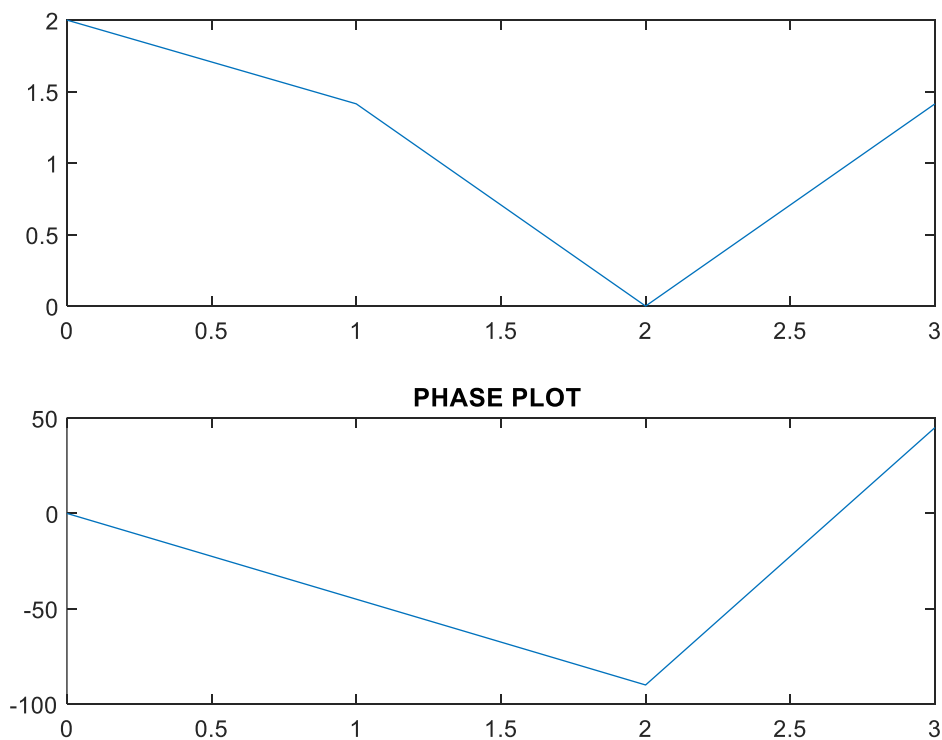
2.0000 + 0.0000i 1.0000 - 1.0000i 0.0000 - 0.0000i 1.0000 + 1.0000i

MagX =

2.0000 1.4142 0.0000 1.4142

PhaseX =

0 -45.0000 -90.0000 45.0000



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

3 FREQUENCY RESPONSE OF A GIVEN SYSTEM GIVEN IN(TRANSFER FUNCTION/ DIFFERENCE EQUATION FORM)

AIM :- To find the frequency response of the following difference equation

$$y(n) - 5y(n-1) = x(n) + 4x(n-1)$$

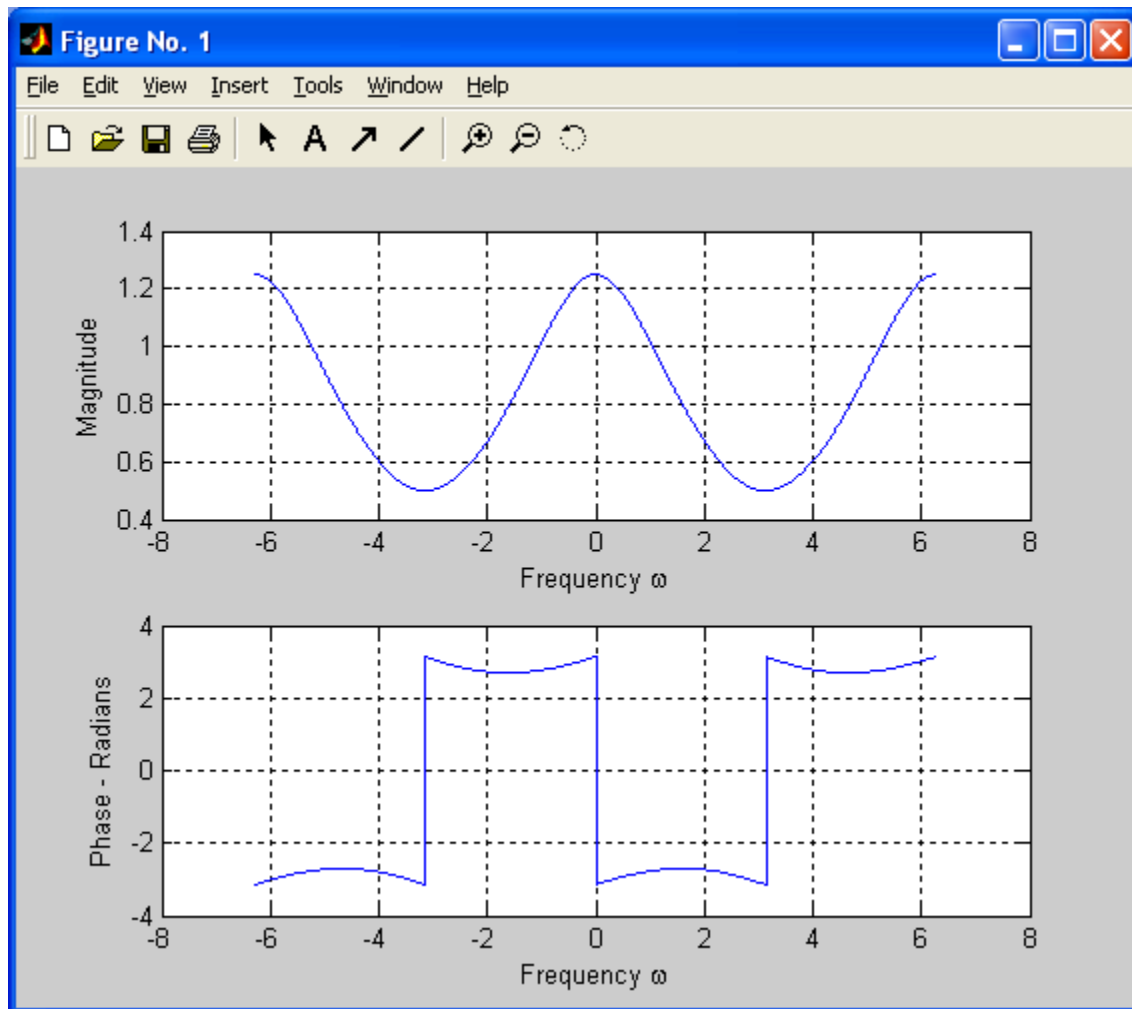
APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

Program:-

```
b = [1, 4]; %Numerator coefficients
a = [1, -5]; %Denominator coefficients
w = -2*pi: pi/256: 2*pi;
[h] = freqz(b, a, w);
subplot(2, 1, 1), plot(w, abs(h));
xlabel('Frequency \omega');
ylabel('Magnitude');
grid
subplot(2, 1, 2);
plot(w, angle(h));
xlabel('Frequency \omega');
ylabel('Phase - Radians');
grid
```

OUTPUT:-

RESULT:- Thus the MATLAB program for Frequency Response of Difference Equation was performed and the output was verified.

Note:- If the transfer function is given instead of difference equation then perform the inverse Z-Transform and obtain the difference equation and then find the frequency response for the difference equation using Matlab.

4) IMPLEMENTATION OF FFT OF GIVEN SEQUENCE

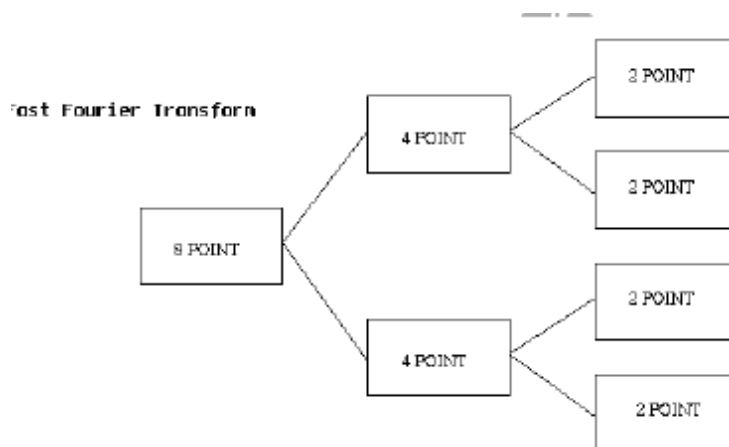
AIM: To perform the FFT of signal $x(n)$ using Mat lab.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

THEORY:- A fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. FFTs are of great importance to a wide variety of applications, from digital signal processing and solving partial differential equations to algorithms for quick multiplication of large integers.



ALGORITHM:

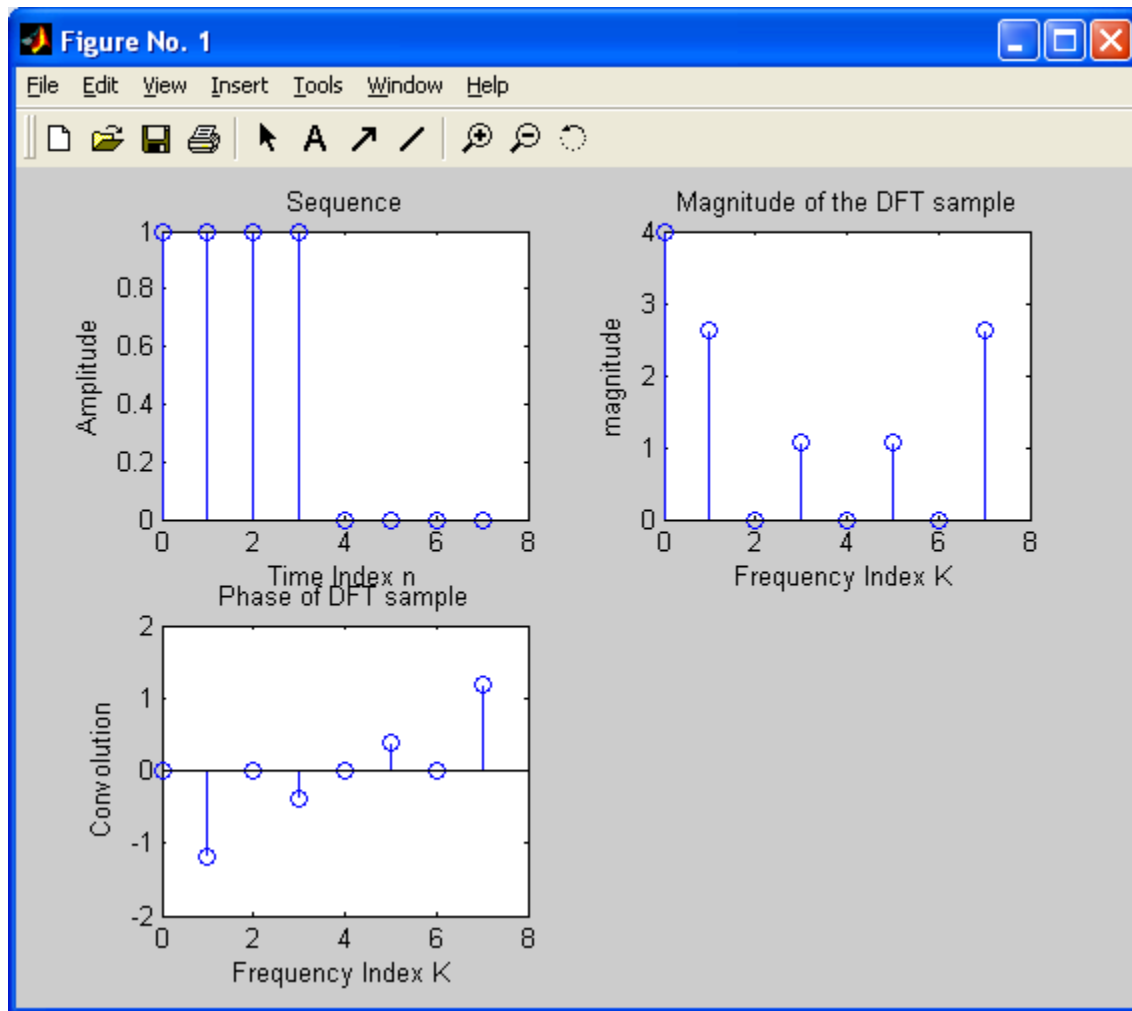
- 1) Get the input sequence
- 2) Number of DFT point(m) is 8
- 3) Find out the FFT function using MATLAB function.
- 4) Display the input & outputs sequence using stem function
- 5) Compile and Run the program
- 6) For the output see command window\ Figure window

1) PROGRAM:

```
clear all;
N=8;
m=8;
a=input('Enter the input sequence');
n=0:1:N-1;
subplot(2,2,1);
stem(n,a);
xlabel('Time Index n');
ylabel('Amplitude');
title('Sequence');
x=fft(a,m);
k=0:1:N-1;
subplot(2,2,2);
stem(k,abs(x));
ylabel('magnitude');
xlabel('Frequency Index K');
title('Magnitude of the DFT sample');
subplot(2,2,3);
stem(k,angle(x));
xlabel('Frequency Index K');
ylabel('Phase');
title('Phase of DFT sample');
ylabel('Convolution');
```

OUTPUT:-

Enter the input sequence[1 1 1 1 0 0 0 0]



RESULT:- Thus Fast Fourier Transform is Performed using Matlab.

2) PROGRAM:

```
clc;
clear all;
close all;
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;

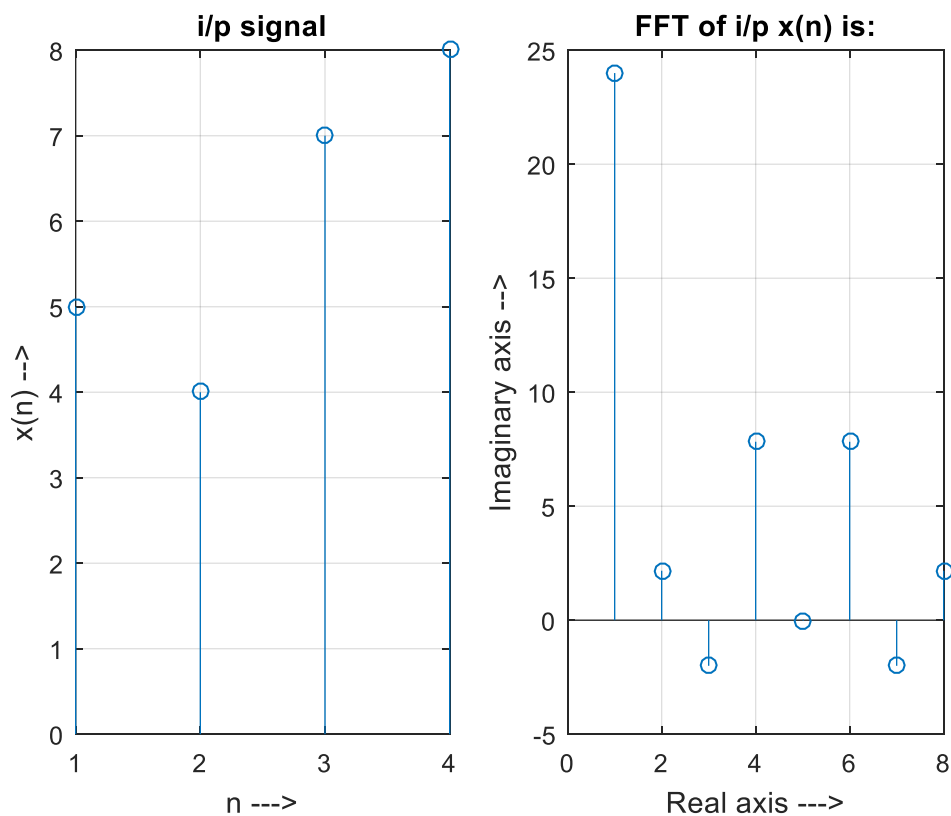
```

OUTPUT :

Enter the sequence:[5 4 7 8]

Enter the length of fft: 8

Fourier transformed signal



RESULT :-- N point FFT algorithm using Matlab is designed.

5) DESIGN AND IMPLEMENTATION OF IIR FILTER USING BILINEAR TRANSFORMATION AND IMPULSE INVARIANT METHOD.

AIM: To design and implementation of iir filter using bilinear transformation and impulse invariant method using matlab.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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

PROGRAM:

% Design using both bilinear and impulse invariant transformations an IIR
% digital low-pass Butterworth filter having following specifications:

% $0.8 \leq |H(e^{j\omega})| \leq 1, |\omega| \leq 0.2\pi$
% $|H(e^{j\omega})| \leq 0.2, 0.6\pi \leq |\omega| \leq \pi$
% Assume $T = 1$ sec.

% Without using 'buttord' inbuilt function.

```
clear all;
close all;
clc;
alphap = 20*log10(1) - 20*log10(0.8); %Calculating the passband attenuation
alphas = 20*log10(1) - 20*log10(0.2); %Calculating the stopband attenuation
op = 2* tan(0.1*pi) %analog passband frequency
os = 2* tan(0.3*pi) %analog stopband frequency
k1 = abs (os/op);
A1 = (10^(0.1*alphas) - 1 )^0.5;
A2 = (10^(0.1*alphap) - 1 )^0.5;
```

```
A = A1/A2;

N = ceil(A/k1) % order of the transfer function

k3 = 1/(2*N);

k4 = (10^(0.1*alphap) - 1)^ k3;

oc = op/k4 % cutoff frequencny calculation

[b,a] = butter(N,oc)

% b and a return the numerator and denominator of the transfer function

w = 0:.01:pi;

[h,om] = freqz (b,a,w,'whole');

m = abs (h);

an = angle (h);

subplot (2,1,1)

plot (om/pi, m),

grid, ylabel ('Normalized Magnitude'),

xlabel ('Normalized Frequency');

subplot (2,1,2)

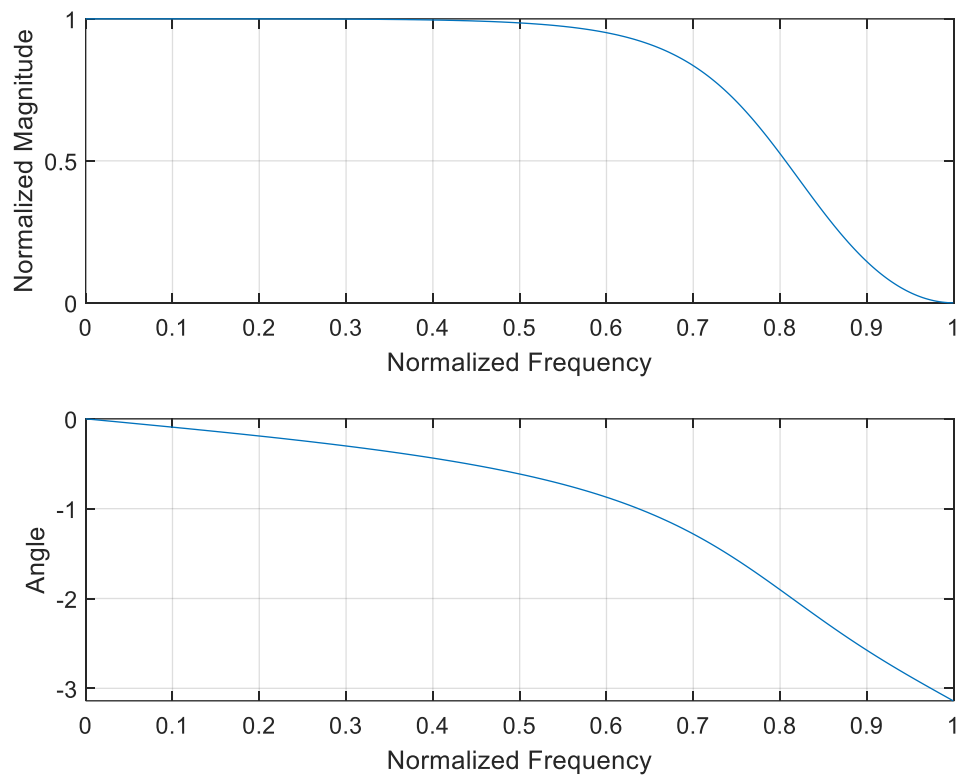
plot (om/pi, an),

grid, ylabel ('Angle'),

xlabel ('Normalized Frequency');

[bz1,az1]=impinvar(b,a,1) % impulse invariant transform of the function

[bz2,az2]=bilinear(b,a,1) % Bilinear transform of the function
```

OUTPUT:

RESULT: IIR filter using bilinear transformation and impulse invariant method using matlab is implemented.

6) DESIGN AND IMPLEMENTATION OF IIR BUTTERWORTH (LP/HP) FILTER.

AIM: To write a matlab program for design and implementation of iir butterworth (lp/hp) filter.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

PROGRAM:

```
clc;

clear all;

close all;

disp('Enter the Analog filter design specifications');

N=input('Enter the order of the filter');

c=input('Enter the choice of filter 1. LPF 2. HPF 3.BPF 4.BSF \n ');

if(c==1)

disp('Frequency response of Analog LPF is:');

Cf=100;

[b,a]=butter(N,Cf,'S');

freqs(b,a);

end

if(c==2)

disp('Frequency response of Analog HPF is:');

Cf=100;

[b,a]=butter(N,Cf,'HIGH','S');

freqs(b,a);

end

if(c==3)
```

```
disp('Frequency response of Analog BPF is:');
```

```
Cf1=[10 100];
```

```
[b,a]=butter(N,Cf1,'S');
```

```
freqs(b,a);
```

```
end
```

```
if(c==4)
```

```
disp('Frequency response of Analog BPF is:');
```

```
Cf1=[10 100];
```

```
[b,a]=butter(N,Cf1,'STOP','S');
```

```
freqs(b,a);
```

```
end
```

OUTPUT :

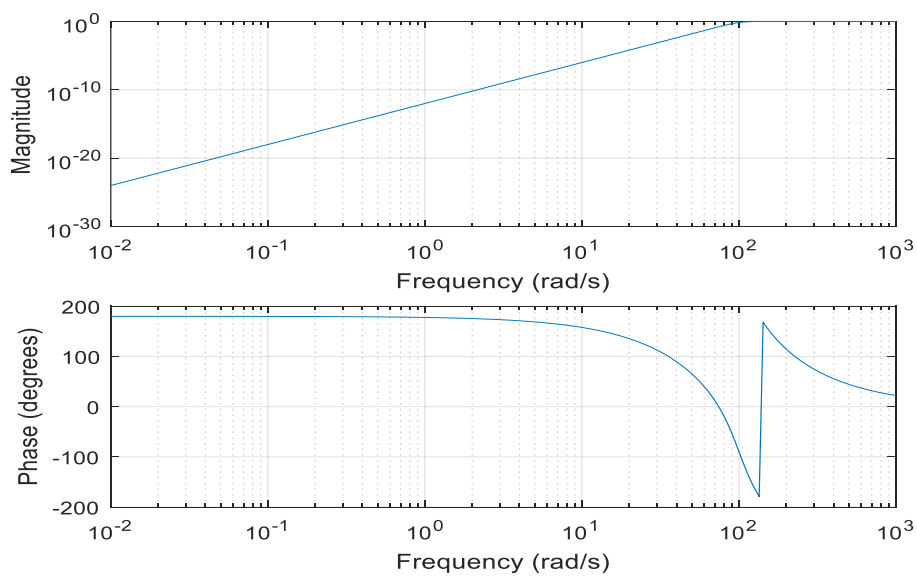
Enter the Analog filter design specifications

Enter the order of the filter 6

Enter the choice of filter 1. LPF 2. HPF 3.BPF 4.BSF

2

Frequency response of Analog HPF is:



RESULT :-- Analog filters using Matlab is designed.

7) DESIGN AND IMPLEMENTATION OF IIR CHEBYSHEV (LP/HP) FILTER.

AIM: To design and implementation of IIR Chebyshev (LP/HP) filter.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

PROGRAM :

```
% 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(1,2,1);

plot(abs(h)); xlabel('frequency'); ylabel('amplitude');

title('low pass chebyshev filter response');

%%% chebyshev high pass filter

wp=0.7;

ws=0.5;

rp=1;

rs=30;

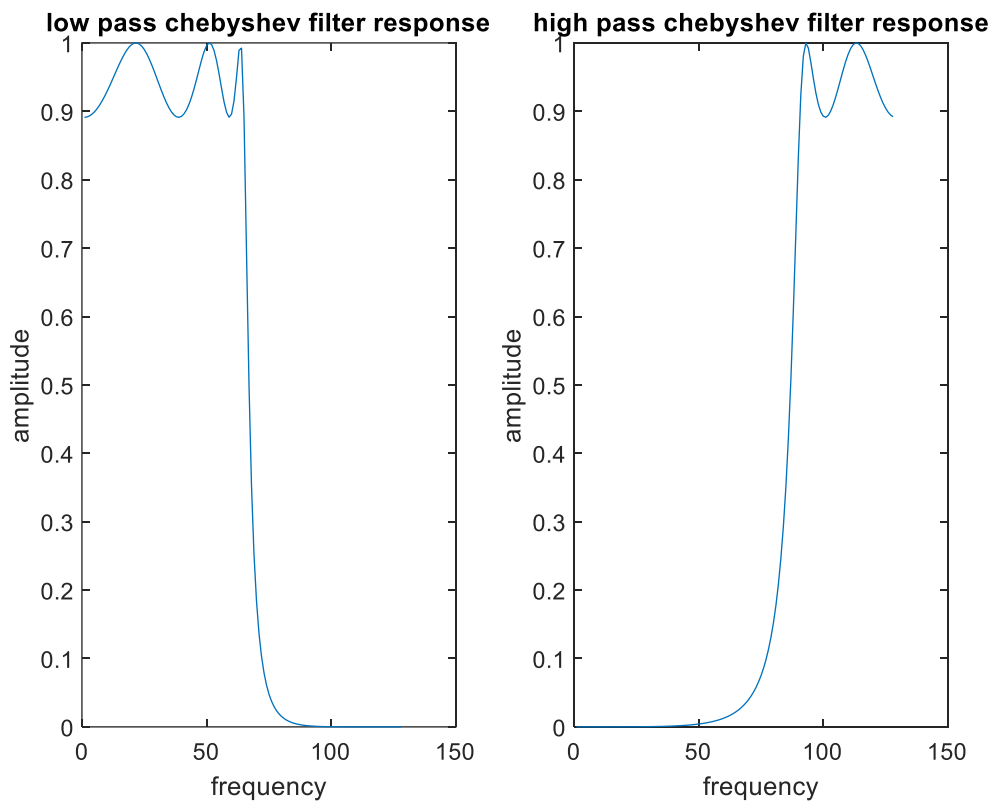
[n,wn]=cheb1ord(wp,ws,rp,rs);

[b,a]=cheby1(n,rp,wn, 'high');
```



```
[h,w]=freqz(b,a,128);  
subplot(1,2,2);  
plot(abs(h));  
xlabel('frequency');  
ylabel('amplitude');  
title('high pass chebyshev filter response')
```

Output :



RESULT : Design and implementation of iir chebyshev(lp/hp) filter using matlab is verified.

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) DESIGN AND IMPLEMENTATION OF FIR WITH LOW PASS FILTER USING ANY THREE WINDOWING TECHNIQUES. PLOT ITS MAGNITUDE AND PHASE RESPONSES.

AIM :- To Design FIR LP Filter using Rectangular/Triangular/kaiser Windowing Technique.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

ALGORITHM:-

- 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), beta value.
- 4) Calculate the analog pass band edge frequencies, w1 and w2.
$$w1 = 2*fp/f$$
$$w2 = 2*fs/f$$
- 5) calculate the numerator and denominator
- 6) Use an If condition and ask the user to choose either Rectangular Window or Triangular window or Kaiser window..
- 7) use rectwin, triang, kaiser commands
- 8) Calculate the magnitude of the frequency response in decibels (dB)
$$m = 20 * \log_{10}(\text{abs}(h))$$
- 9) Plot the magnitude response [magnitude in dB Vs normalized frequency (ω/π)]
- 10) Give relevant names to x and y axes and give an appropriate title for the plot.
- 11) Plot all the responses in a single figure window. [Make use of subplot]

PROGRAM:-

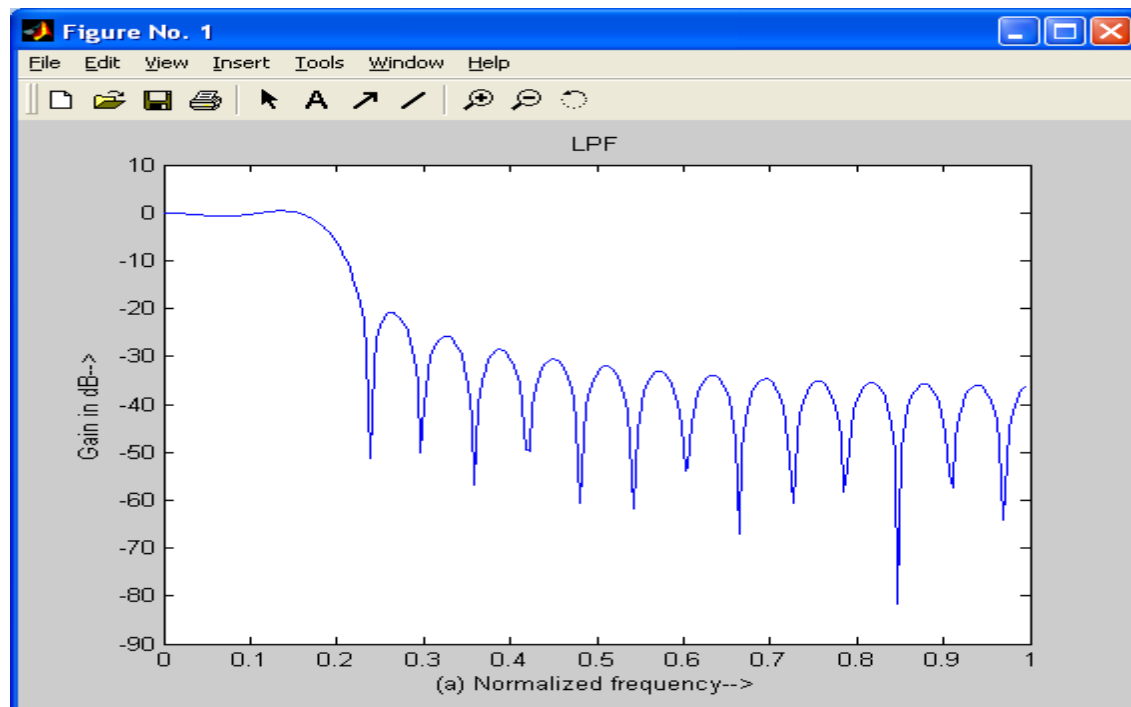
```
%FIR Filter design window techniques
clc;
clear all;
close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter passband freq');
fs=input('enter stopband freq');
f=input('enter sampling freq ');
beta=input('enter beta value,');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs- fp)/f;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
n1=n;
n=n-1;
end
c=input('enter your choice of window function 1. rectangular 2. triangular 3.kaiser: \n ');
if(c==1)
y=rectwin(n1);
disp('Rectangular window filter response');
end
if (c==2)
y=triang(n1);
disp('Triangular window filter response');
end
if(c==3)
y=kaiser(n1,beta);
disp('kaiser window filter response');
end
```

```
%LPF
b=fir1(n,wp,y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
plot(o/pi,m);
title('LPF');
ylabel('Gain in dB-->');
xlabel('(a) Normalized frequency-->');
```

Output:-

enter passband ripple 0.02
enter the stopband ripple 0.01
enter passband freq 1000
enter stopband freq 1500
enter sampling freq 10000
enter beta value
enter your choice of window function 1. rectangular 2. triangular 3.kaiser:
1

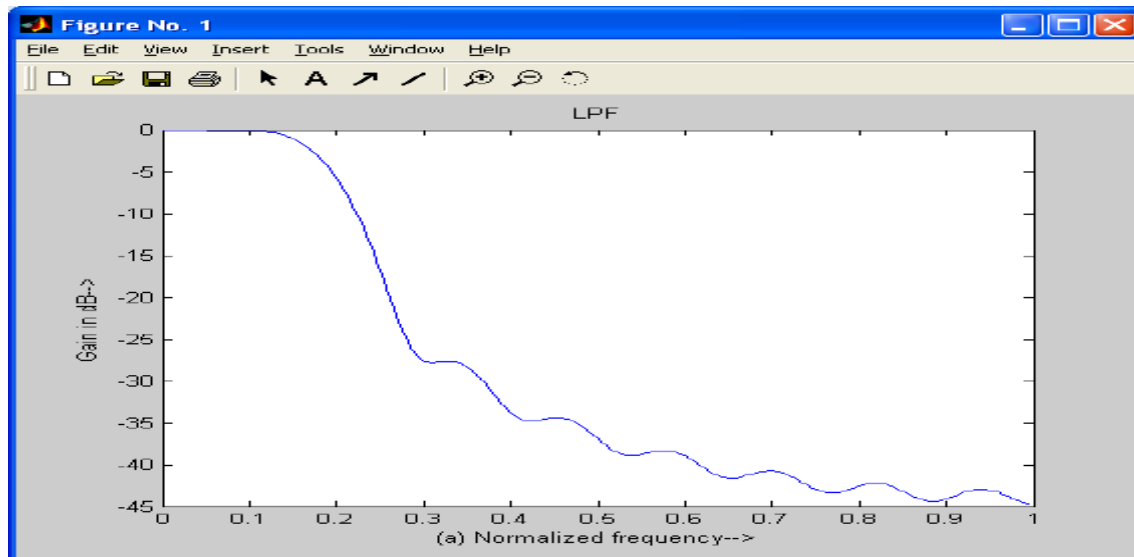
Rectangular window filter response



enter your choice of window function 1. rectangular 2. triangular 3.kaiser:

2

triangular window filter response

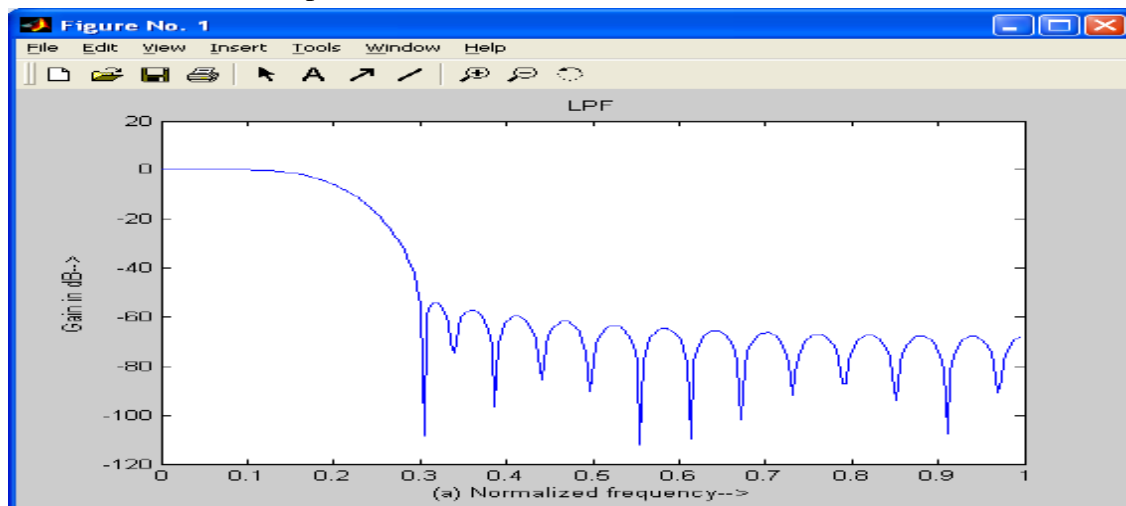


enter beta value 5

enter your choice of window function 1. rectangular 2. triangular 3.kaiser:

3

kaiser window filter response



RESULT:- Thus FIR LP Filter is designed for Rectangular/triangular/kaiser windowing techniques using MATLAB.

9) DESIGN AND IMPLEMENTATION OF FIR FILTER WITH HIGH PASSFILTER USING ANY THREE WINDOWINGTECHNIQUES. PLOT ITS MAGNITUDE AND PHASE RESPONSES.

AIM :- To Design FIR HP Filter using Rectangular/Triangular/kaiser Windowing Technique.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

ALGORITHM:-

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

$$w1 = 2*fp/f$$

$$w2 = 2*fs/f$$

- 5) calculate the numerator and denominator
- 6) Use an If condition and ask the user to choose either Rectangular Window or Triangular window or Kaiser window..
- 7) use rectwin, triang, kaiser commands
- 8) Calculate the magnitude of the frequency response in decibels (dB)

$$m=20*\log_{10}(\text{abs}(h))$$

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

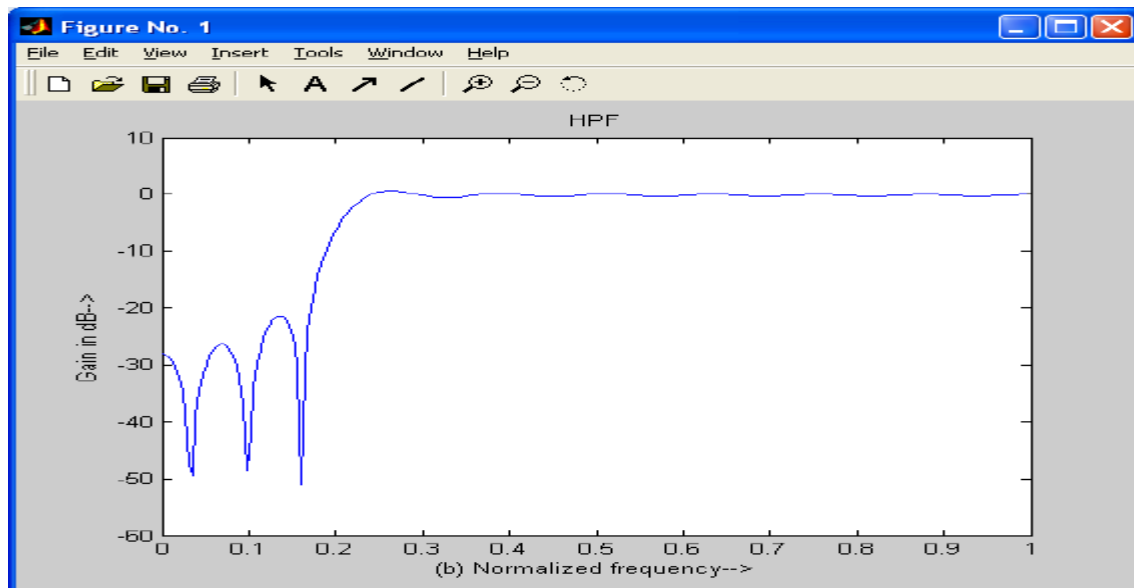
PROGRAM:-

```
%FIR Filter design window techniques
clc;
clear all;
close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
fp=input('enter passband freq');
fs=input('enter stopband freq');
f=input('enter sampling freq ');
beta=input('enter beta value');
wp=2*fp/f;
ws=2*fs/f;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(fs- fp)/f;
n=ceil(num/dem);
n1=n+1;
if(mod(n,2)~=0)
n1=n;
n=n-1;
end
c=input('enter your choice of window function 1. rectangular 2. triangular 3.kaiser: \n ');
if(c==1)
y=rectwin(n1);
disp('Rectangular window filter response');
end
if (c==2)
y=triang(n1);
disp('Triangular window filter response');
end
if(c==3)
y=kaiser(n1,beta);
disp('kaiser window filter response');
end
%HPF
b=fir1(n,wp,'high',y);
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
plot(o/pi,m);
title('HPF');
ylabel('Gain in dB-->');
xlabel('(b) Normalized frequency-->');
```

OUTPUT:-

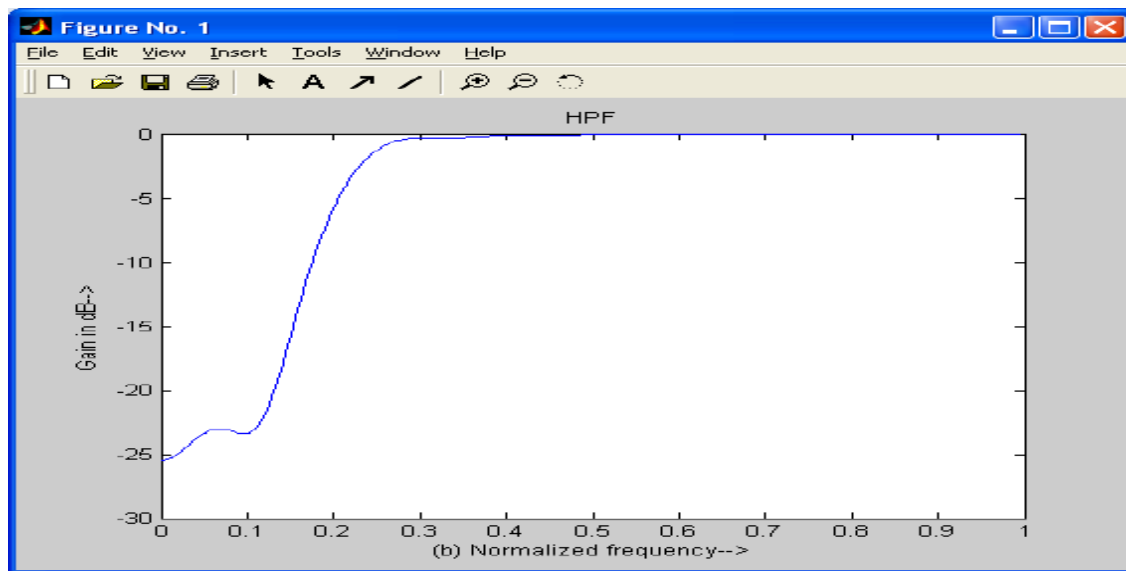
enter passband ripple 0.02
enter the stopband ripple 0.01
enter passband freq 1000
enter stopband freq 1500
enter sampling freq 10000
enter beta value
enter your choice of window function 1. rectangular 2. triangular 3.kaiser:
1

Rectangular window filter response



enter your choice of window function 1. rectangular 2. triangular 3.kaiser:
2

triangular window filter response

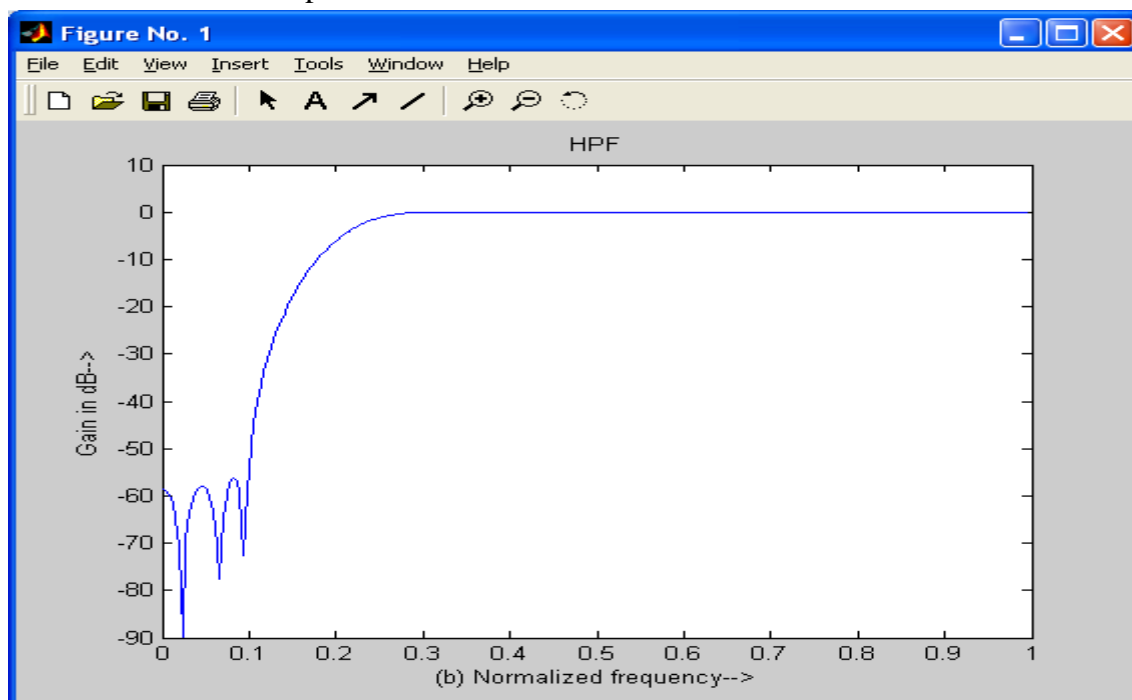


enter beta value 5

enter your choice of window function 1. rectangular 2. triangular 3.kaiser:

3

kaiser window filter response



RESULT:- Thus FIR HP Filter is designed for Rectangular/triangular/kaiser windowing techniques using MATLAB.

10) DESIGN AND IMPLEMENTATION OF FIR FILTER WITH BAND PASS / BAND STOPFILTER USING ANY THREE WINDOWING TECHNIQUES. PLOT ITS MAGNITUDE AND PHASE RESPONSES.

AIM : To design and implementation of fir filter with band pass / band stopfilter using any three windowing techniques. plot its magnitude and phase responses using matlab.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

PROGRAM :

MATLAB CODE FOR KAISER WINDOW:

```
clc;

clear all;

close all;

passripple=0.04; %pass band ripple
stopripple=0.05; %stop band ripple
passfreq=2000; %pass band ripple
stopfreq=3000; %pass band ripple
freq=9000; %sampling freq

wp=2*passfreq/freq;
ws=2*stopfreq/freq;

s=passripple*stopripple;

r=stopfreq-passfreq;

p=r/freq;

N=-20*log10(sqrt(s))-13;

D=14.6*p;

n=ceil(N/D);
```

```
n1=n+1;

if(rem(n,2)~=0)

n1=n;

n=n-1;

end

beta=2;

for beta=2:2:4

y=kaiser(n1,beta);

b=fir1(n,wp,'low',y); %low pass filter

[h,o]=freqz(b,1,256);

m=20*log10(abs(h));

figure(beta)

subplot(2,2,1);

plot(o/pi,m,'b');

title('Low pass filter');

ylabel('gain in db');

xlabel('(a) normalised frequency');

b=fir1(n,wp,'high',y); %high pass filter

[h,o]=freqz(b,1,256);

m=20*log10(abs(h));

subplot(2,2,2);

plot(o/pi,m,'b');

title('high pass filter');

ylabel('gain in db');

xlabel('(b) normalised frequency');

wn=[wp,ws]; %band pass filter

b=fir1(n,wn,y);

[h,o]=freqz(b,1,256);

m=20*log10(abs(h));

subplot(2,2,3);
```

```
plot(o/pi,m,'b');  
  
title('Band pass filter');  
  
ylabel('gain in db');  
  
xlabel('(c)normalised frequency');  
  
b=fir1(n,wn,'stop',y); %band stop filter  
  
[h,o]=freqz(b,1,256);  
  
m=20*log10(abs(h));  
  
subplot(2,2,4);  
  
plot(o/pi,m,'b');  
  
title('band stop filter');  
  
ylabel('gain in db');  
  
xlabel('(d) normalised frequency');  
  
end
```

OUTPUT : FIGURE 1: When beta value is 2.

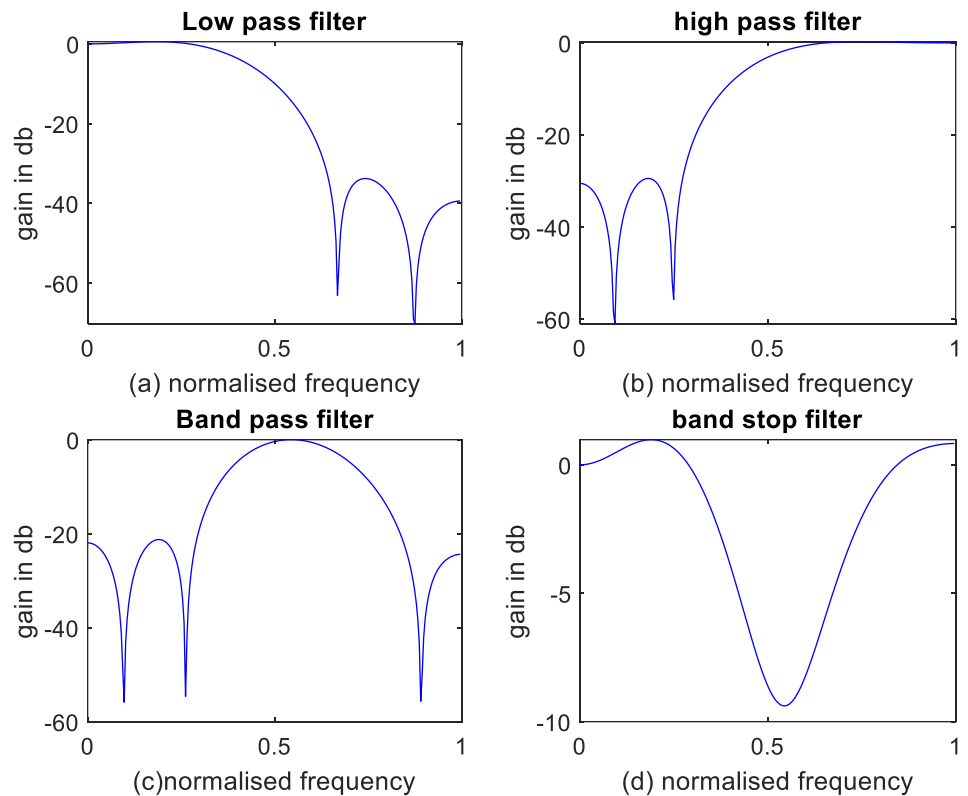
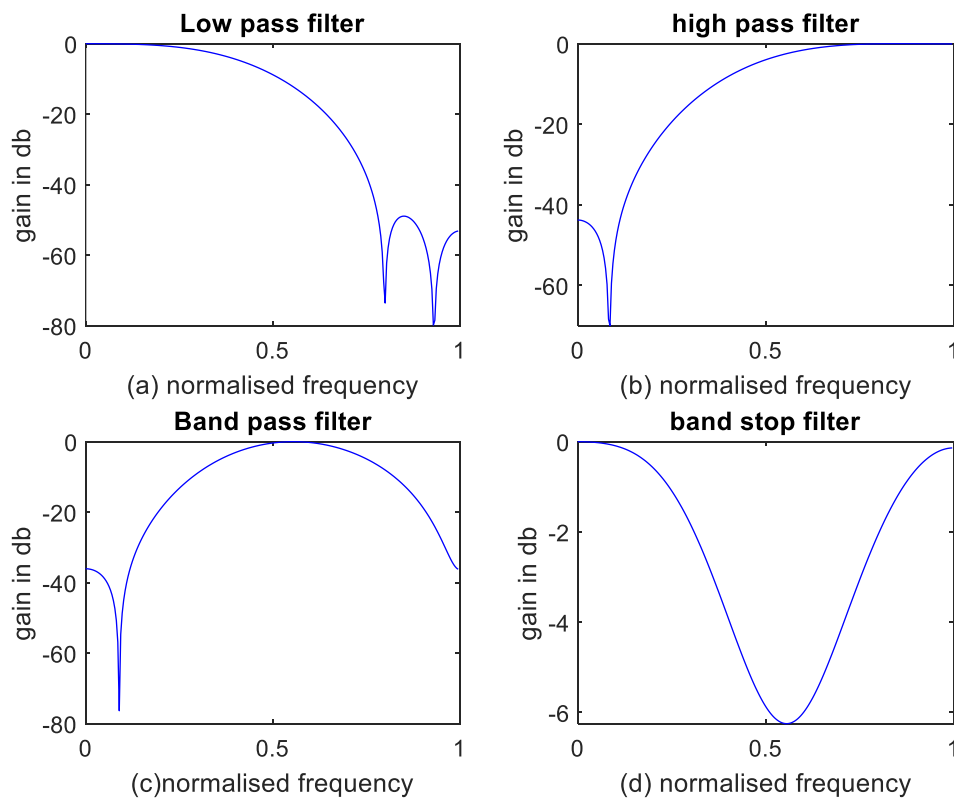


FIGURE 2 : When beta value is 4



MATLAB CODE FOR COMPARISON OF WINDOW METHODS

```
clc;

clear all;

close all;

passripple=0.04; %pass band ripple
stopripple=0.05; %stop band ripple
passfreq=2000; %pass band ripple
stopfreq=3000; %pass band ripple
freq=9000; %sampling freq

wp=2*passfreq/freq;
ws=2*stopfreq/freq;

s=passripple*stopripple;
r=stopfreq-passfreq;
p=r/freq;
num=-20*log10(sqrt(s))-13;
dem=14.6*p;
n=ceil(num/dem);
n1=n+1;
if(rem(n,2)~=0)
    n1=n;
    n=n-1;
end
beta=4;
y=kaiser(n1,beta);
b=fir1(n,wp,'low',y); %low pass filter
[h,o]=freqz(b,1,256);
m=20*log10(abs(h));
figure(2)
subplot(2,2,1);
plot(o/pi,m,'b');

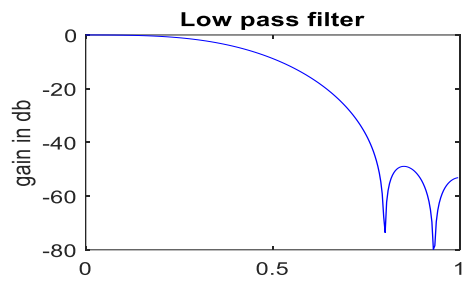
title('Low pass filter');
```

```
ylabel('gain in db');  
  
xlabel('(a) normalised frequency');  
  
b=fir1(n,wp,'high',y); %high pass filter  
  
[h,o]=freqz(b,1,256);  
  
m=20*log10(abs(h));  
  
subplot(2,2,2);  
  
plot(o/pi,m,'b');  
  
title('high pass filter');  
  
ylabel('gain in db');  
  
xlabel('(b) normalised frequency');  
  
wn=[wp,ws]; %band pass filter  
  
b=fir1(n,wn,y);  
  
[h,o]=freqz(b,1,256);  
  
m=20*log10(abs(h));  
  
subplot(2,2,3);  
  
plot(o/pi,m,'b');  
  
title('Band pass filter');  
  
ylabel('gain in db');  
  
xlabel('(c)normalised frequency');  
  
b=fir1(n,wn,'stop',y); %band stop filter  
  
[h,o]=freqz(b,1,256);  
  
m=20*log10(abs(h));  
  
subplot(2,2,4);  
  
plot(o/pi,m,'b');  
  
title('band stop filter');  
  
ylabel('gain in db');  
  
xlabel('(d) normalised frequency');  
  
hold on  
  
ys=rectwin(n1);  
  
b=fir1(n,wp,'low',ys); %low pass filter  
  
[h,o]=freqz(b,1,256);
```

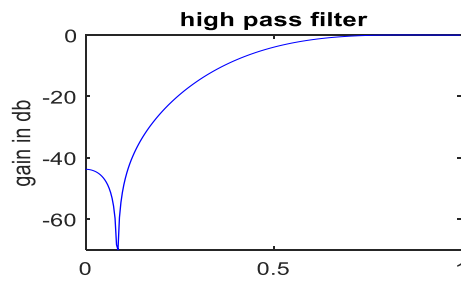
```
m=20*log10(abs(h));  
  
figure(1)  
subplot(2,2,1);  
plot(o/pi,m,'r');  
title('Low pass filter');  
ylabel('gain in db');  
xlabel('(a) normalised frequency');  
  
b=fir1(n,wp,'high',ys); %high pass filter  
[h,o]=freqz(b,1,256);  
m=20*log10(abs(h));  
subplot(2,2,2);  
plot(o/pi,m,'r');  
title('high pass filter');  
ylabel('gain in db');  
xlabel('(b) normalised frequency');  
  
wn=[wp,ws]; %band pass filter  
b=fir1(n,wn,ys);  
[h,o]=freqz(b,1,256);  
m=20*log10(abs(h));  
subplot(2,2,3);  
plot(o/pi,m,'r');  
title('Band pass filter');  
ylabel('gain in db');  
xlabel('(c)normalised frequency');  
  
b=fir1(n,wn,'stop',ys); %band stop filter  
[h,o]=freqz(b,1,256);  
m=20*log10(abs(h));  
subplot(2,2,4);  
plot(o/pi,m,'r');  
title('band stop filter');  
ylabel('gain in db'); xlabel('(d) normalised frequency');
```


OUTPUT :

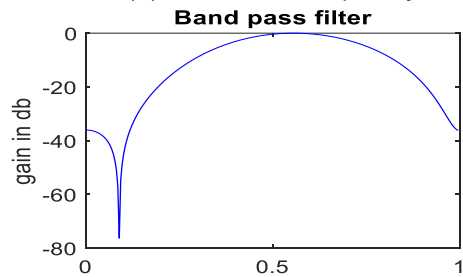
Blue color- Kaiser Window, Red color-Rectangular window



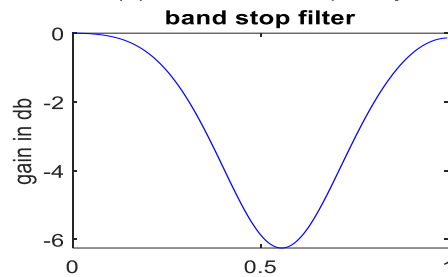
(a) normalised frequency



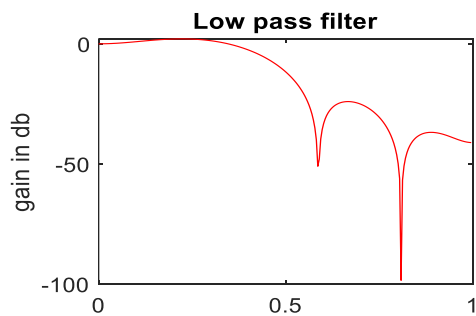
(b) normalised frequency



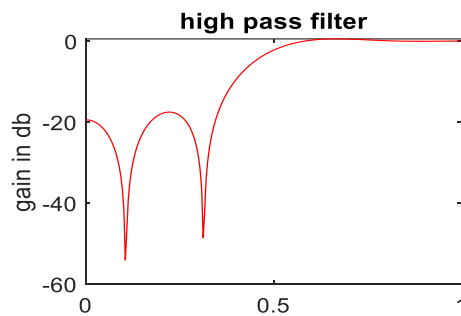
(c) normalised frequency



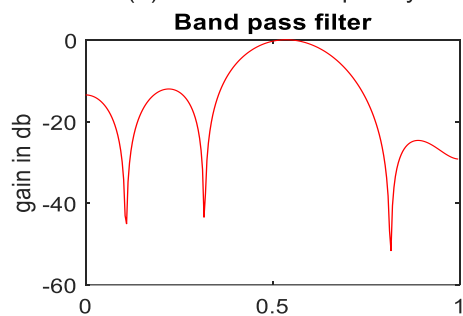
(d) normalised frequency



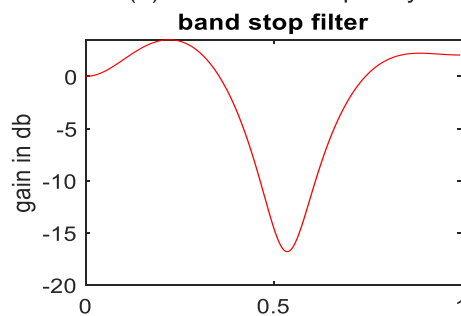
(a) normalised frequency



(b) normalised frequency



(c) normalised frequency



(d) normalised frequency

ADDITIONAL PROGRAM**11) DESIGN OF IIR FILTER USING ANY OF THE AVAILABLE METHODS AND VERIFY THE FREQUENCY RESPONSE OF THE FILTER**

AIM: To write a matlab program for design of IIR filter using any of the available methods and verify the frequency response of the filter.

APPARATUS: PC with MATLAB Software.

PROCEDURE:

- 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.

PROGRAM:

% IIR FILTERS LPF & HPF USING MATLAB

```
clc;

clear all;

close all;

disp('enter the IIR filter design specifications');

rp=input('enter the passband ripple');

rs=input('enter the stopband ripple');

wp=input('enter the passband freq');

ws=input('enter the stopband freq');

fs=input('enter the sampling freq');

w1=2*wp/fs;w2=2*ws/fs;

[n,wn]=buttord(w1,w2,rp,rs,'s');
```

% IIR FILTERS LPF & HPF USING MATLAB

```
clc;

clear all;

close all;

disp('enter the IIR filter design specifications');
```

```
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband freq');
ws=input('enter the stopband freq');
fs=input('enter the sampling freq');
w1=2*wp/fs;w2=2*ws/fs;
[n,wn]=buttord(w1,w2,rp,rs,'s');
c=input('enter choice of filter 1. LPF 2. HPF \n ');
if(c==1)
    disp('Frequency response of IIR LPF is:');
    [b,a]=butter(n,wn,'low','s');
end
if(c==2)
    disp('Frequency response of IIR HPF is:');
    [b,a]=butter(n,wn,'high','s');
end
w=0:.01:pi;
[h,om]=freqs(b,a,w);
m=20*log10(abs(h));
an=angle(h);
figure,subplot(2,1,1);plot(om/pi,m);
title('magnitude response of IIR filter is:');
xlabel('(a) Normalized freq. -->');
ylabel('Gain in dB-->');
subplot(2,1,2);plot(om/pi,an);
title('phase response of IIR filter is:');
xlabel('(b) Normalized freq. -->');
ylabel('Phase in radians-->');
```

OUTPUT :--

enter the IIR filter design specifications

enter the passband ripple.5

enter the stopband ripple.8

enter the passband freq1000

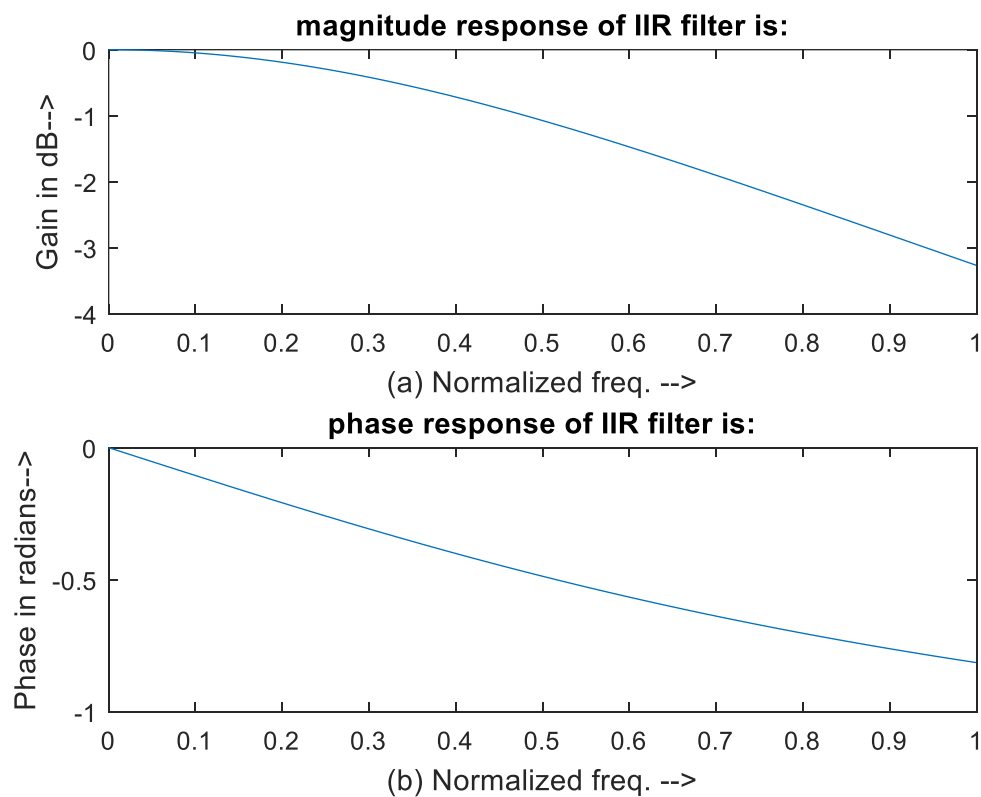
enter the stopband freq2000

enter the sampling freq3000

enter choice of filter 1. LPF 2. HPF

1

Frequency response of IIR LPF is:



RESULT:-- IIR filters using matlab is designed.

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?

CC STUDIO PROGRAMS

INTRODUCTION TO DSP PROCESSORS

A digital signal processor (DSP) is an integrated circuit designed for high-speed data manipulations, and is used in audio, communications, image manipulation, and other data-acquisition and data-control applications. The microprocessors used in personal computers are optimized for tasks involving data movement and inequality testing. The typical applications requiring such capabilities are word processing, database management, spread sheets, etc. When it comes to mathematical computations the traditional microprocessor are deficient particularly where real-time performance is required. Digital signal processors are microprocessors optimized for basic mathematical calculations such as additions and multiplications.

Fixed versus Floating Point:

Digital Signal Processing can be divided into two categories, fixed point and floating point which refer to the format used to store and manipulate numbers within the devices. Fixed point DSPs usually represent each number with a minimum of 16 bits, although a different length can be used. There are four common ways that these 2^{16} i.e., 65,536 possible bit patterns can represent a number. In unsigned integer, the stored number can take on any integer value from 0 to 65,535, signed integer uses two's complement to include negative numbers from -32,768 to 32,767. With unsigned fraction notation, the 65,536 levels are spread uniformly between 0 and 1 and the signed fraction format allows negative numbers, equally spaced between -1 and 1.

The floating point DSPs typically use a minimum of 32 bits to store each value. This results in many more bit patterns than for fixed point, 2^{32} i.e., 4,294,967,296 to be exact. All floating point DSPs can also handle fixed point numbers, a necessity to implement counters, loops, and signals coming from the ADC and going to the DAC. However, this doesn't mean that fixed point math will be carried out as quickly as the floating point operations; it depends on the internal architecture.

C versus Assembly:

DSPs are programmed in the same languages as other scientific and engineering applications, usually assembly or C. Programs written in assembly can execute faster, while programs written in C are easier to develop and maintain. In traditional applications, such as programs run on PCs

and mainframes, C is almost always the first choice. If assembly is used at all, it is restricted to short subroutines that must run with the utmost speed.

How fast are DSPs?

The primary reason for using a DSP instead of a traditional microprocessor is speed: the ability to move samples into the device and carry out the needed mathematical operations, and output the processed data. The usual way of specifying the fastness of a DSP is: fixed point systems are often quoted in MIPS (million integer operations per second). Likewise, floating point devices can be specified in MFLOPS (million floating point operations per second).

TMS320 Family:

The Texas Instruments TMS320 family of DSP devices covers a wide range, from a 16-bit fixed-point device to a single-chip parallel-processor device. In the past, DSPs were used only in specialized applications. Now they are in many mass-market consumer products that are continuously entering new market segments. The Texas Instruments TMS320 family of DSP devices and their typical applications are mentioned below.

C1x, C2x, C2xx, C5x, and C54x: The width of the data bus on these devices is 16 bits. All have modified Harvard architectures. They have been used in toys, hard disk drives, modems, cellular phones, and active car suspensions.

C3x: The width of the data bus in the C3x series is 32 bits. Because of the reasonable cost and floating-point performance, these are suitable for many applications. These include almost any filters, analyzers, hi-fi systems, voice-mail, imaging, bar-code readers, motor control, 3D graphics, or scientific processing.

C4x: This range is designed for parallel processing. The C4x devices have a 32-bit data bus and are floating-point. They have an optimized on-chip communication channel, which enables a number of them to be put together to form a parallel-processing cluster. The C4x range devices have been used in virtual reality, image recognition, telecom routing, and parallel-processing systems.

C6x: The C6x devices feature VelociTI™, an advanced very long instruction word (VLIW) architecture developed by Texas Instruments. Eight functional units, including two multipliers and six arithmetic logic units (ALUs), provide 1600 MIPS of cost-effective performance. The C6x DSPs are optimized for multi-channel, multi function applications, including wireless base stations, pooled modems, remote-access servers, digital subscriber loop systems, cable modems, and multi-channel telephone systems.

Typical Applications for the TMS320 Family

The TMS320 DSPs offer adaptable approaches to traditional signal-processing problems and support complex applications that often require multiple operations to be performed simultaneously.

Automotive	Consumer	Control
Adaptive ride control	Digital radios/TVs	Disk drive control
Cellular telephones	Music synthesizers	Laser printer control
Digital radios	Pagers	Motor control
Navigation	Radar detectors	Servo control
Vibration analysis	Solid-state answering machines	

General-Purpose	Graphics/Imaging	Industrial
Adaptive filtering	3-D transformations	Numeric control
Convolution	Animation/digital maps	Power-line monitoring
Correlation	Homomorphic processing	Robotics
Digital filtering	Image compression/transmission	Security access
Fast Fourier transforms	Image enhancement	Military
Instrumentation	Medical	
Digital filtering	Diagnostic equipment	Image processing
Function generation	Fetal monitoring	Missile guidance
Pattern matching	Hearing aids	Navigation
Phase-locked loops	Patient monitoring	Radar processing
Seismic processing	Prosthetics	Radio frequency modems
Spectrum analysis	Ultra sound equipment	Secure communications
Transient analysis	Sonar processing	GPS
Telecommunications		Voice/Speech
1200- to 56 600-bps modems	Faxing	Speaker verification
Adaptive equalizers	Future terminals	Speech enhancement
ADPCM transcoders	Line repeaters	Speech recognition
Echo cancellation	X.25 packet switching	Speech synthesis
Channel multiplexing	PDA	Speech coding
Data encryption	Speaker phones	Text-to-speech
Digital PBXs	Spread spectrum	Voice mail
Digital speech interpolation		
DTMF encoding/decoding	Video conferencing	

INTRODUCTION TO TMS 320 C6713 DSK

The high-performance board features the TMS320C6713 floating-point DSP. Capable of performing 1350 million floating point operations per second, the C6713 DSK the most powerful DSK development board.



The DSK is USB port interfaced platform that allows to efficiently develop and test applications for the C6713. With extensive host PC and target DSP software support, the DSK provides ease-of-use and capabilities that are attractive to DSP engineers.

The 6713 DSP Starter Kit (DSK) is a low-cost platform which lets customers evaluate and develop applications for the Texas Instruments C67X DSP family.

The primary features of the DSK are:

- 225 MHz TMS320C6713 Floating Point DSP
- AIC23 Stereo Codec
- Four Position User DIP Switch and Four User LEDs
- On-board Flash and SDRAM

TI's Code Composer Studio development tools are bundled with the 6713DSK providing the user with an industrial-strength integrated development environment for C and assembly programming.

Code Composer Studio communicates with the DSP using an on-board JTAG emulator through a USB interface.

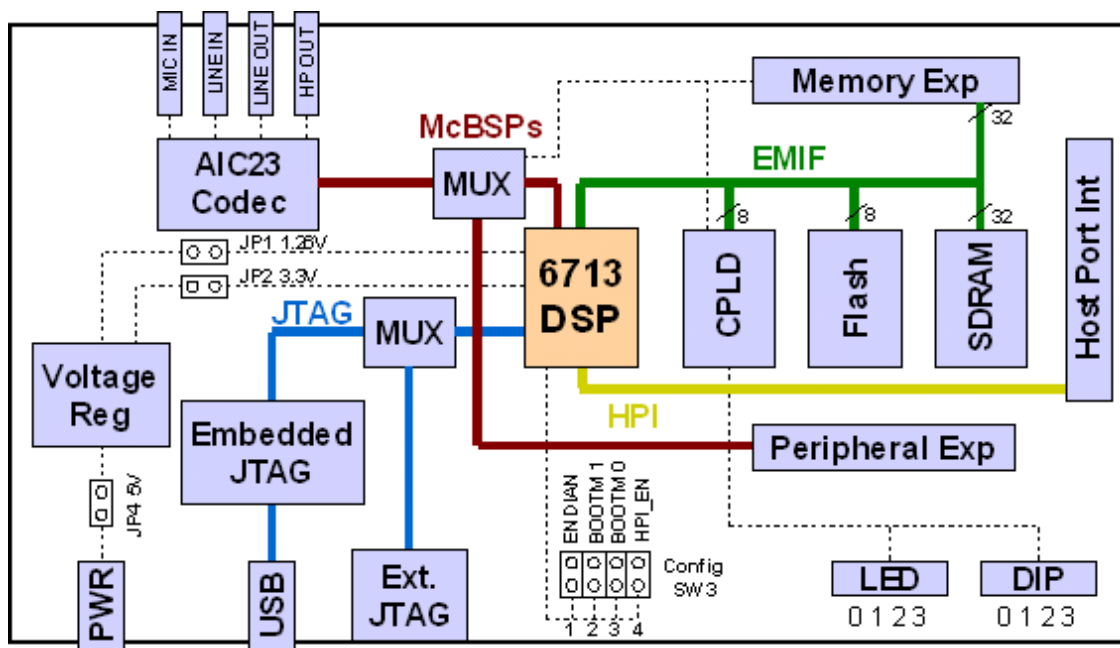
The TMS320C6713 DSP is the heart of the system. It is a core member of Texas Instruments' C64X line of fixed point DSPs whose distinguishing features are an extremely high performance 225MHz VLIW DSP core and 256Kbytes of internal memory. On-chip peripherals include a 32-bit external memory interface (EMIF) with integrated SDRAM controller, 2 multi-channel buffered serial ports (McBSPs), two on-board timers and an enhanced DMA controller (EDMA). The 6713 represents the high end of TI's C6700 floating point DSP line both in terms of computational performance and on-chip resources.

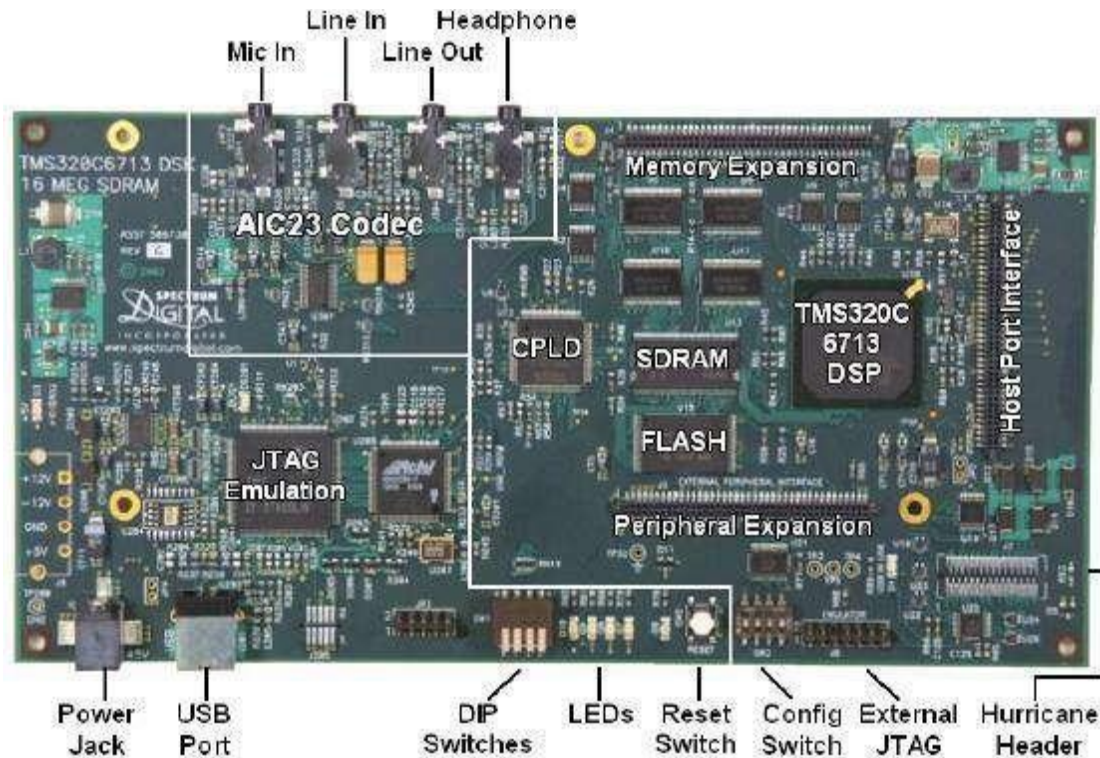
The 6713 has a significant amount of internal memory so many applications will have all code

and data on-chip. External accesses are done through the EMIF which can connect to both synchronous and asynchronous memories. The EMIF signals are also brought out to standard TI expansion bus connectors so additional functionality can be added on daughter card modules.

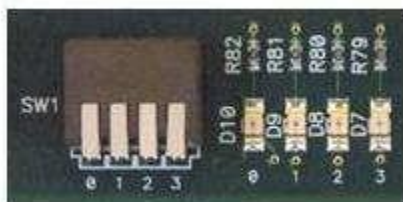
DSPs are frequently used in audio processing applications so the DSK includes an on-board codec called the AIC23. Codec stands for coder/decoder, the job of the AIC23 is to code analog input samples into a digital format for the DSP to process, then decode data coming out of the DSP to generate the processed analog output. Digital data is sent to and from the codec on McBSP1.

TMS320C6713 DSK Overview Block Diagram





The DSK has 4 light emitting diodes (LEDs) and 4 DIP switches that allow users to interact with programs through simple LED displays and user input on the switches. Many of the included examples make use of these user interfaces Options.



The DSK implements the logic necessary to tie board components together in a programmable logic device called a CPLD. In addition to random glue logic, the CPLD implements a set of 4 software programmable registers that can be used to access the on-board LEDs and DIP switches as well as control the daughter card interface.

DSK hardware installation

- Shut down and power off the PC
- Connect the supplied USB port cable to the board
- Connect the other end of the cable to the USB port of PC
- Plug the other end of the power cable into a power outlet
- Plug the power cable into the board
- The user LEDs should flash several times to indicate board is operational
- When you connect your DSK through USB for the first time on a Windows loaded PC the new hardware found wizard will come up. **So, Install the drivers** (The CCS CD contains the required drivers for C6713 DSK).
- Install the CCS software for C6713 DSK.

Troubleshooting DSK Connectivity

If Code Composer Studio IDE fails to configure your port correctly, perform the following steps:

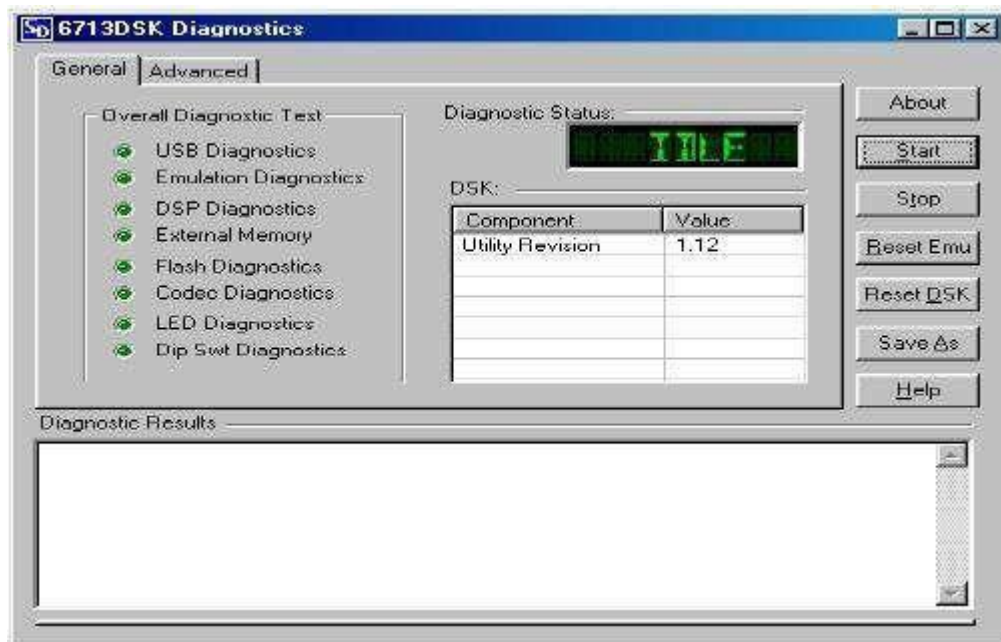
- Test the USB port by running DSK Port test from the start menu

Use Start → Programs → Texas Instruments → Code Composer Studio → Code Composer Studio C6713 DSK Tools → C6713 DSK Diagnostic Utilities

- The below screen will appear
- Select 6713 DSK Diagnostic Utility Icon from Desktop
- The screen look like as below

Select Start Option

- Utility Program will test the board
- After testing Diagnostic Status you will get **PASS**



If the board still fails to detect Go to CM OS setup Enable the USB Port Option
(The required Device drivers will load along with CCS Installation)

SOFTWARE INSTALLATION

You must install the hardware before you install the software on your system.

The requirements for the operating platform are;

- Insert the installation CD into the CD-ROM drive
An install screen appears; if not, goes to the windows Explorer and run setup.exe
- Choose the option to install Code Composer Studio
If you already have C6000 CC Studio IDE installed on your PC, do not install DSK software. CC Studio IDE full tools supports the DSK platform

Respond to the dialog boxes as the installation program runs

The Installation program automatically configures CC Studio IDE for operation with your DSK and creates a CCStudio IDE DSK icon on your desktop. To install, follow these instructions:

1) STUDY THE ARCHITECTURE OF DSP CHIPS – TMS 320C 5X/6X INSTRUCTIONS.

INTRODUCTION

The TMS320C6713 (C6713) is based on the VLIW architecture, which is very well suited for numerically intensive algorithms. The internal program memory is structured so that a total of eight instructions can be fetched every cycle. For example, with a clock rate of 225MHz, the C6713 is capable of fetching eight 32-

bit instructions every $1/(225 \text{ MHz})$ or 4.44 ns. Features of the C6713 include 264 kB of internal memory (8kB as L1P and L1D Cache and 256kB as L2 memory shared between program and data space), eight functional or execution units composed of six arithmetic-logic units (ALUs) and two multiplier units, a 32-bit address bus to address 4 GB (gigabytes), and two sets of 32-bit general-purpose registers.

The C67xx (such as the C6701, C6711, and C6713) belong to the family of the C6x floating-point processors, whereas the C62xx and C64xx belong to the family of the C6x fixed-point processors. The C6713 is capable of both fixed- and floating point processing.

An application-specific integrated circuit (ASIC) has a DSP core with customized circuitry for a specific application. A C6x processor can be used as a standard general-purpose DSP programmed for a specific application. Specific purpose digital signal processors are the modem, echo canceler, and others. A

fixed-point processor is better for devices that use batteries, such as cellular phones, since it uses less power than does an equivalent floating-point processor. The fixed-point processors, C1x, C2x, and C5x, are 16-bit processors with limited dynamic range and precision. The C6x fixed-point processor is a 32-bit processor with improved dynamic range and precision. In a fixed-point processor, it is necessary to scale the data. Overflow, which occurs when an operation such as the addition of two numbers produces a result with more bits than can fit within a processor's register, becomes a concern.

A floating-point processor is generally more expensive since it has more "real estate" or is a larger chip because of additional circuitry necessary to handle integer as well as floating-point arithmetic. Several factors, such as cost, power consumption, and speed, come into play when choosing a specific DSP.

The C6x processors are particularly useful for applications requiring intensive computations. Family members of the C6x include both fixed-point (e.g., C62x, C64x) and floating-point (e.g., C67x) processors. Other DSP's are also available from companies such as Motorola and Analog Devices.

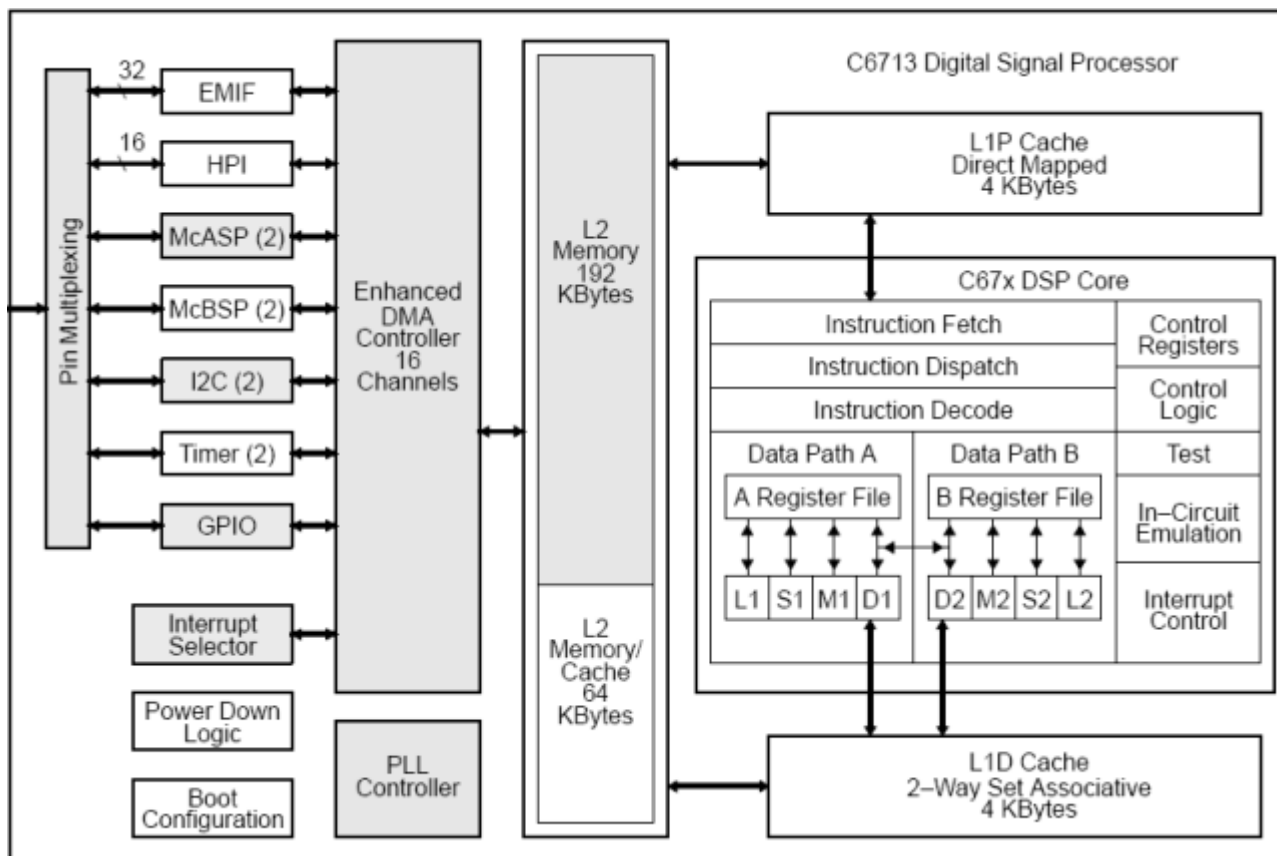


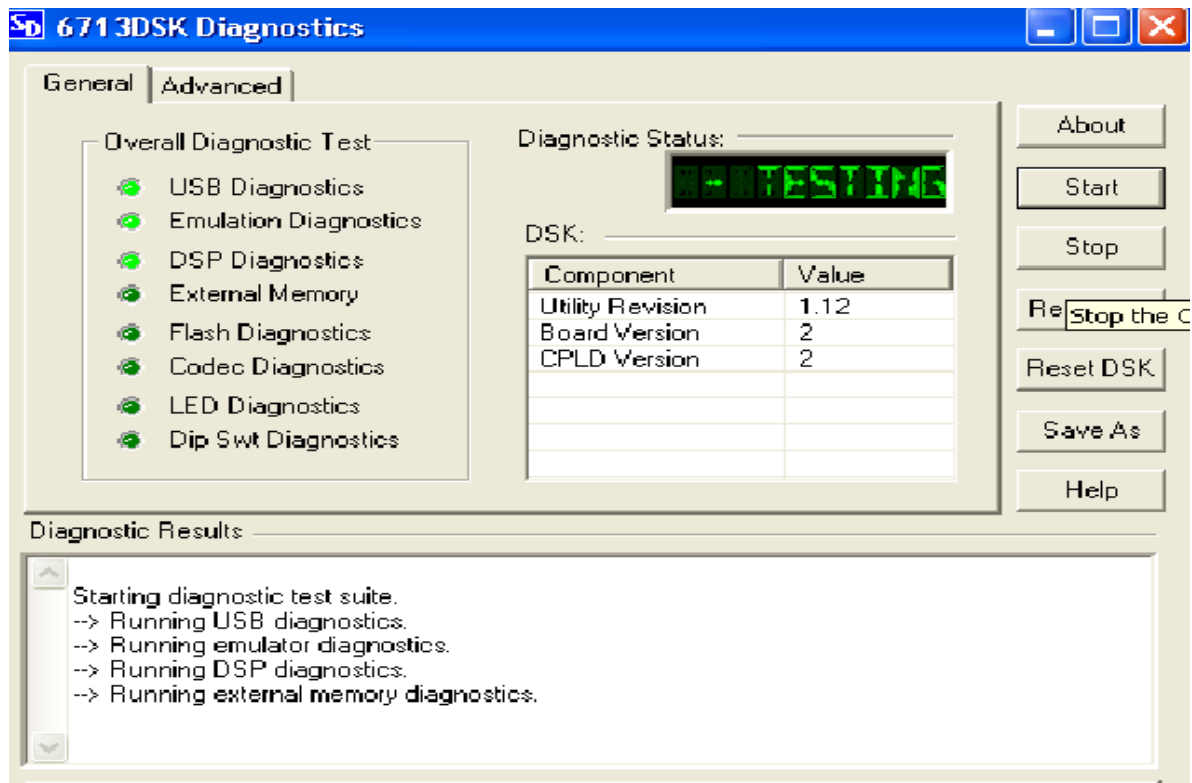
Figure : Functional block diagram of TMS320C6713

The TMS320C6713 onboard the DSK is a floating-point processor based on the VLIW architecture [6–10]. Internal memory includes a two-level cache architecture with 4 kB of level 1 program cache (L1P), 4 kB of level 1 data cache (L1D), and 256 kB of level 2 memory shared between program and data space.

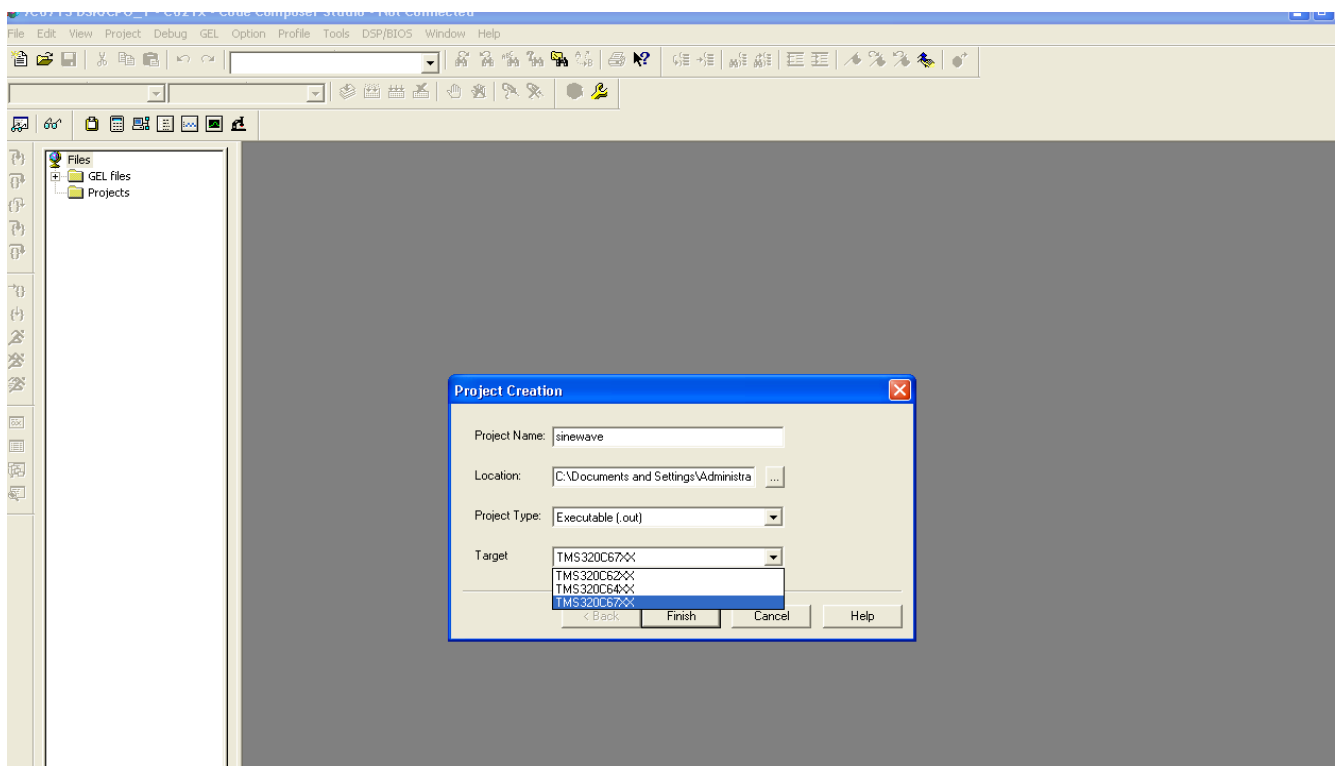
It has a glue less (direct) interface to both synchronous memories (SDRAM and SBSRAM) and asynchronous memories (SRAM and EPROM). Synchronous memory requires clocking but provides a compromise between static SRAM and dynamic DRAM, with SRAM being faster but more expensive than DRAM. On-chip peripherals include two McBSPs, two timers, a host port interface (HPI), and a 32-bit EMIF. It requires 3.3 V for I/O and 1.26 V for the core (internal). Internal buses include a 32-bit program address bus, a 256-bit program data bus to accommodate eight 32-bit instructions, two 32-bit data address buses, two 64-bit data buses, and two 64-bit store data buses. With a 32-bit address bus, the total memory space is $2^{32} = 4\text{GB}$, including four external memory spaces: CE0, CE1, CE2, and CE3. Figure shows a functional block diagram of the C6713 processor included with CCS.

TMS320C6X- PROCEDURE FOR USING C60 DEBUGGER

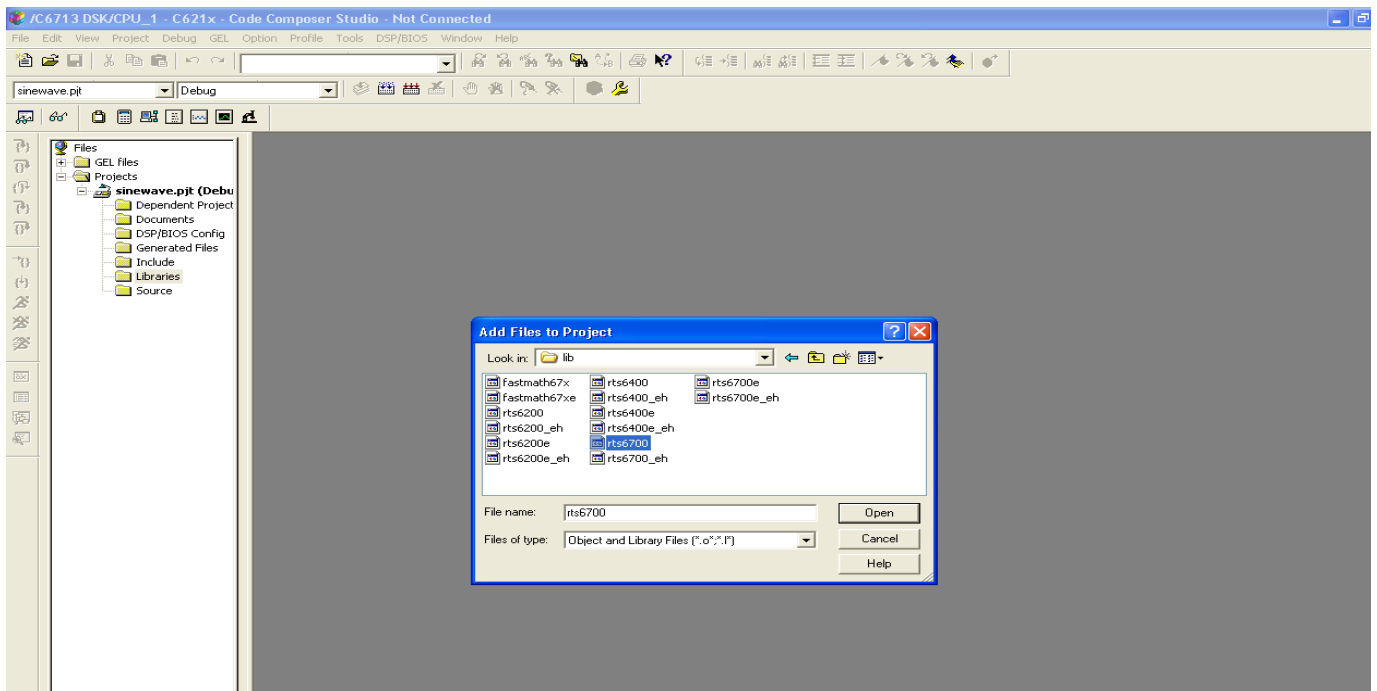
Step1: Open → 6177 Diagnostic → start → Stop → Close



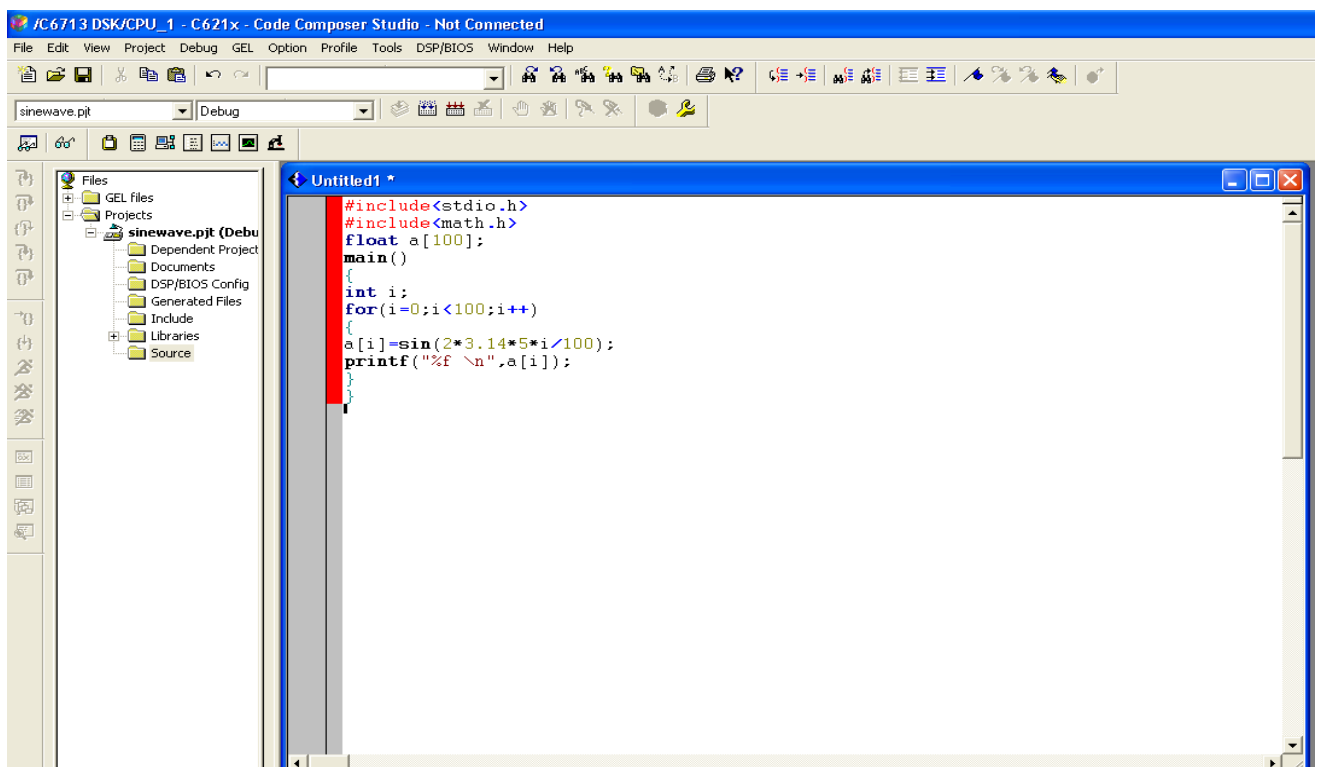
Step 2: Project → New → Project Name → Target → TMS320C67XX



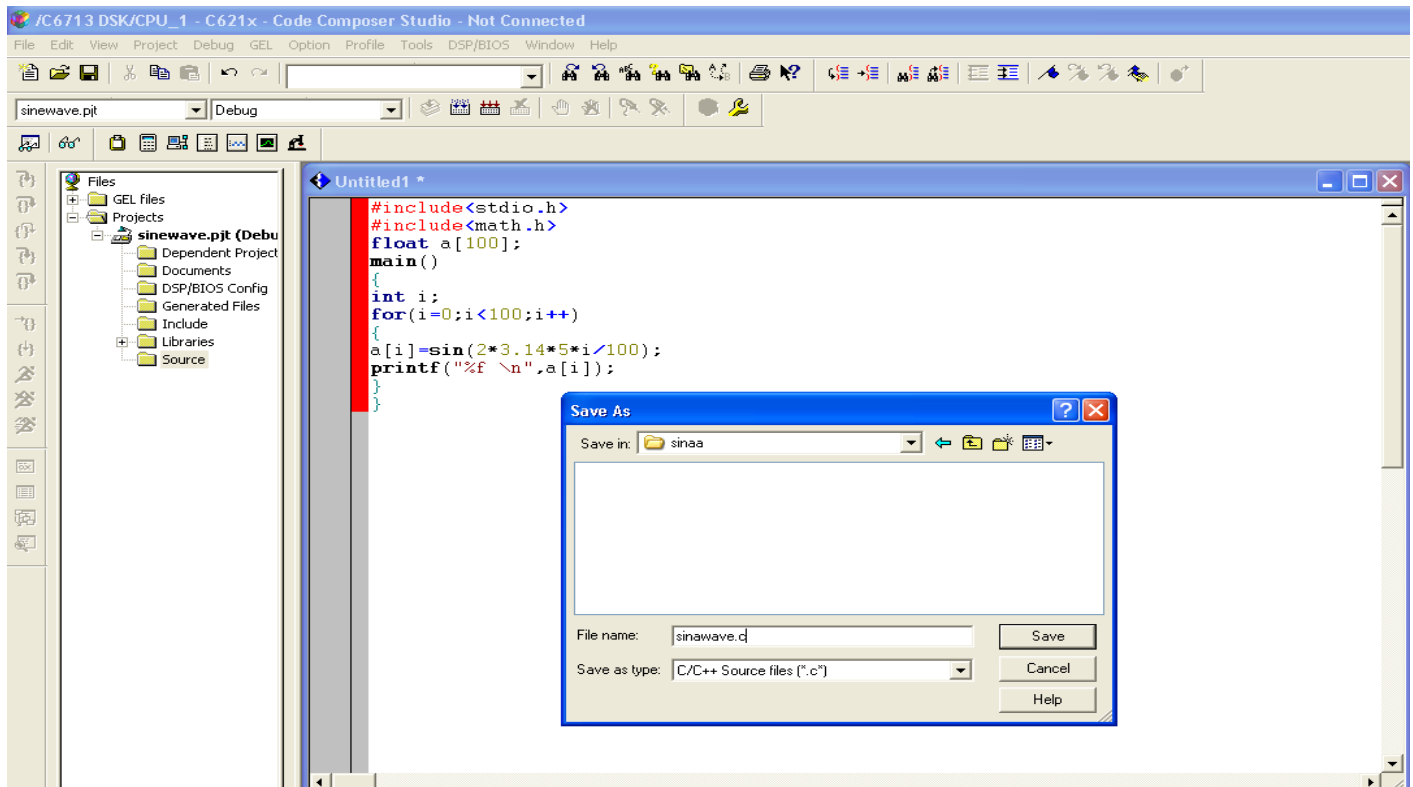
Step 3: Libraries → Add Files to Project → My Project → C600 → CG Tools → Library → rts6700.lib



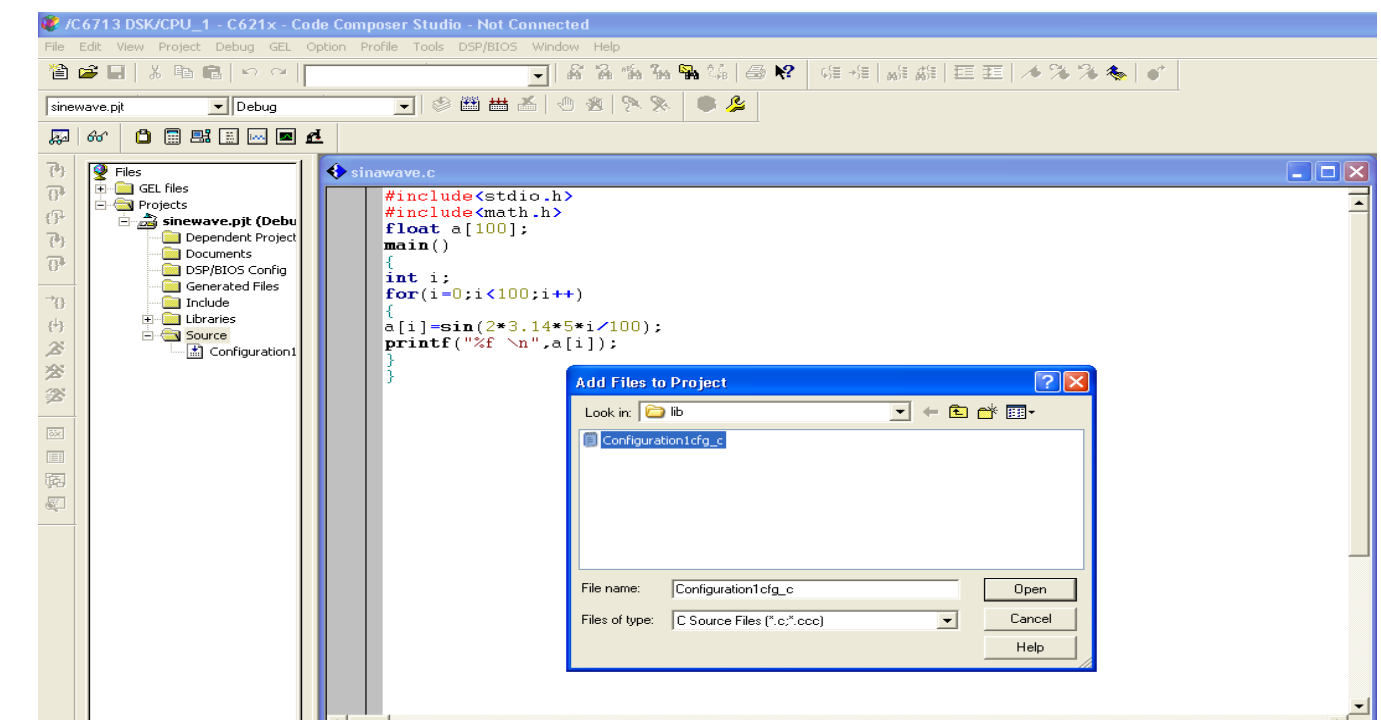
Step 4: File → New → Source File → Write Program in window



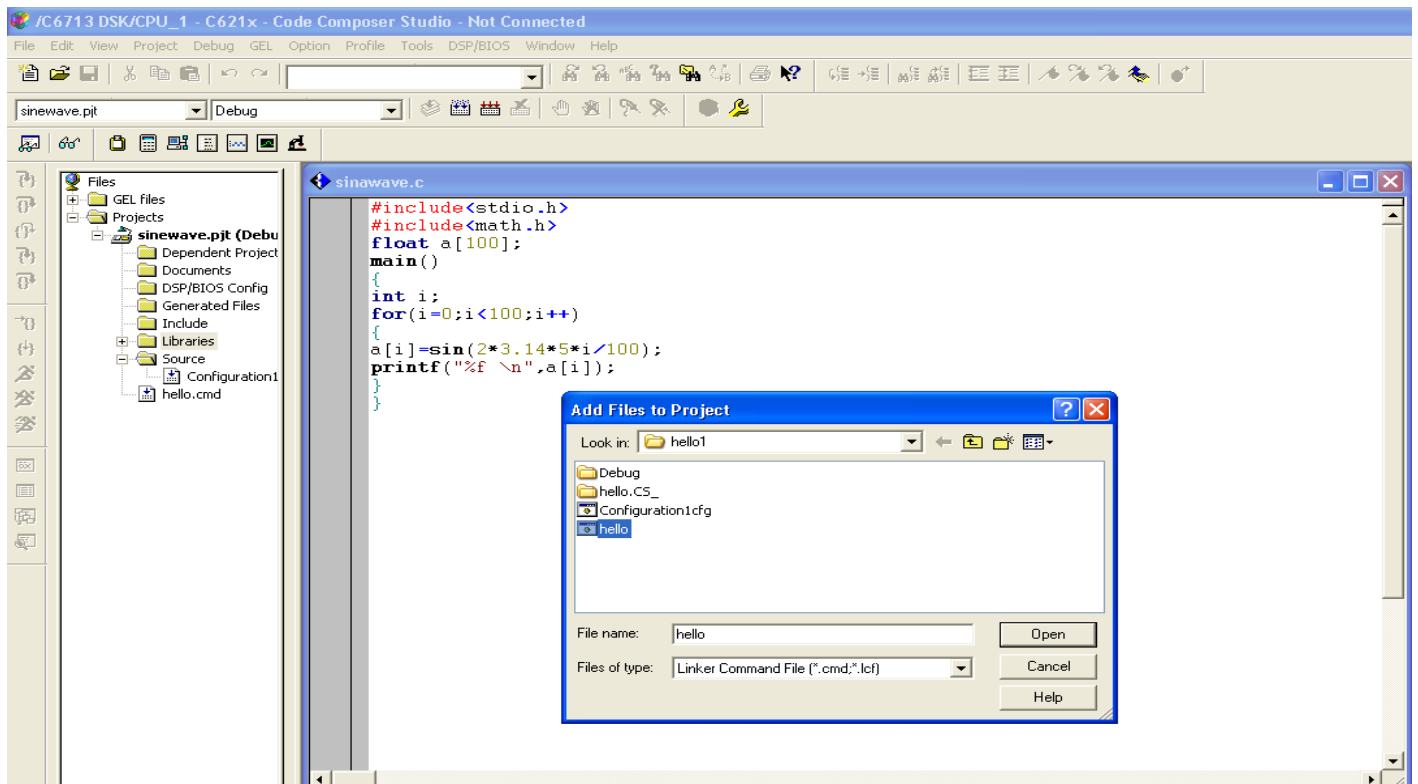
Step 5: File → Save as → CStudio → my project → sss.c



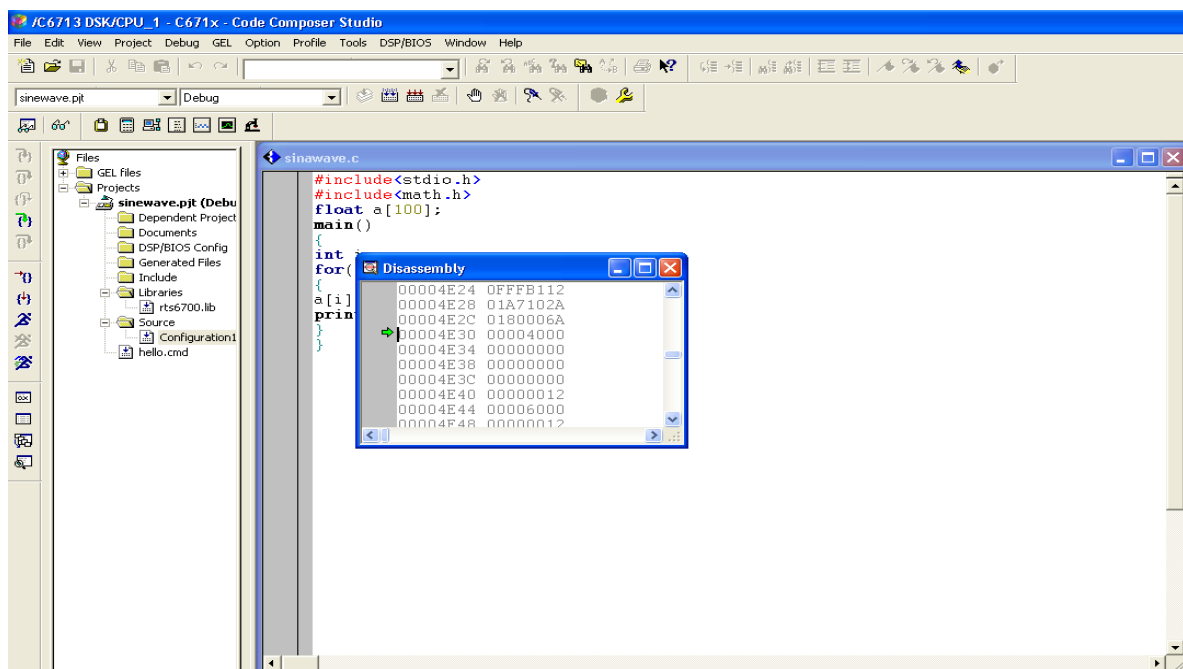
Step 6: Source → Add Files to project → CStudio → lib → confi.c



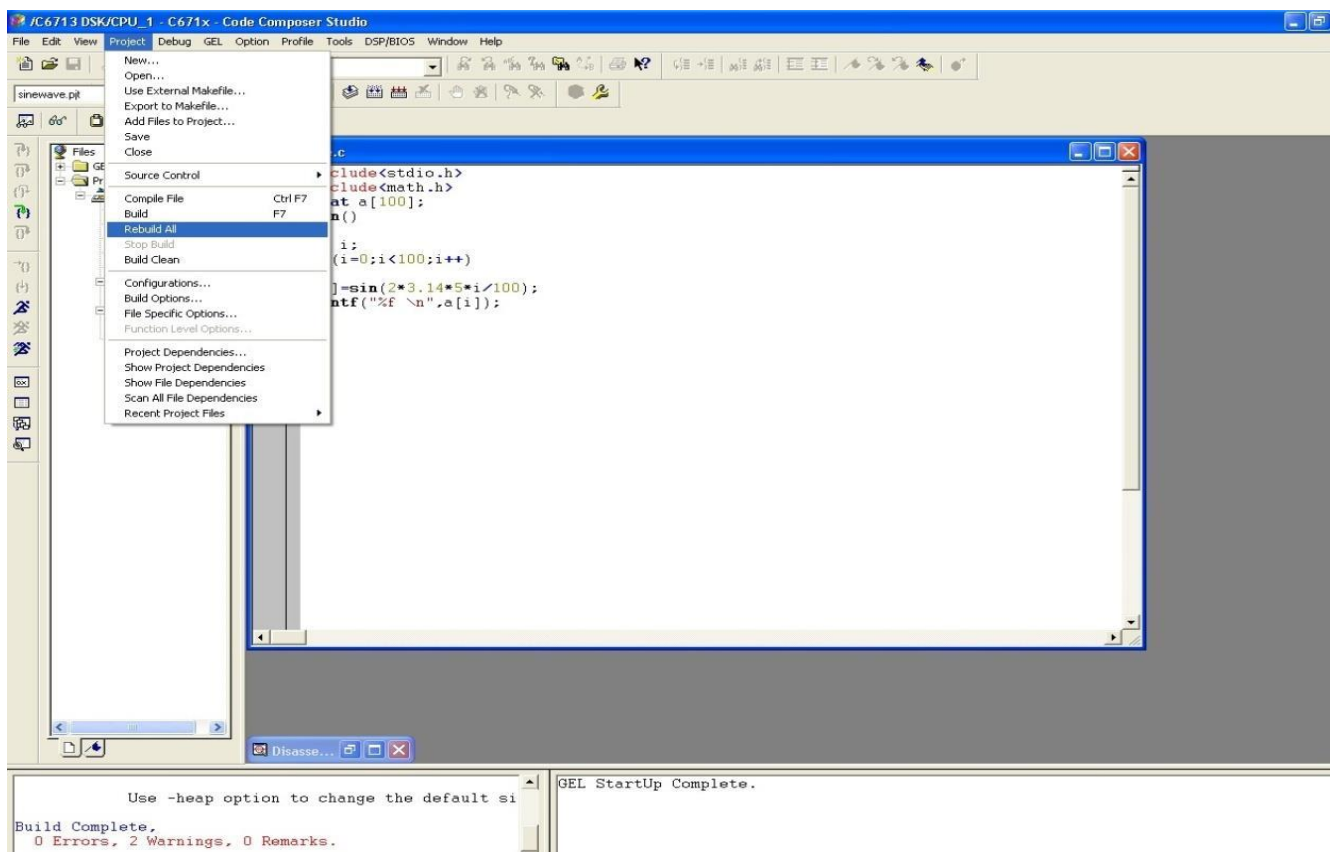
Step 7: Source → Add Files to Project → CStudio → C600 → Tutorial →
Dsk6713 → hello1 → link cmd → hello.cmd



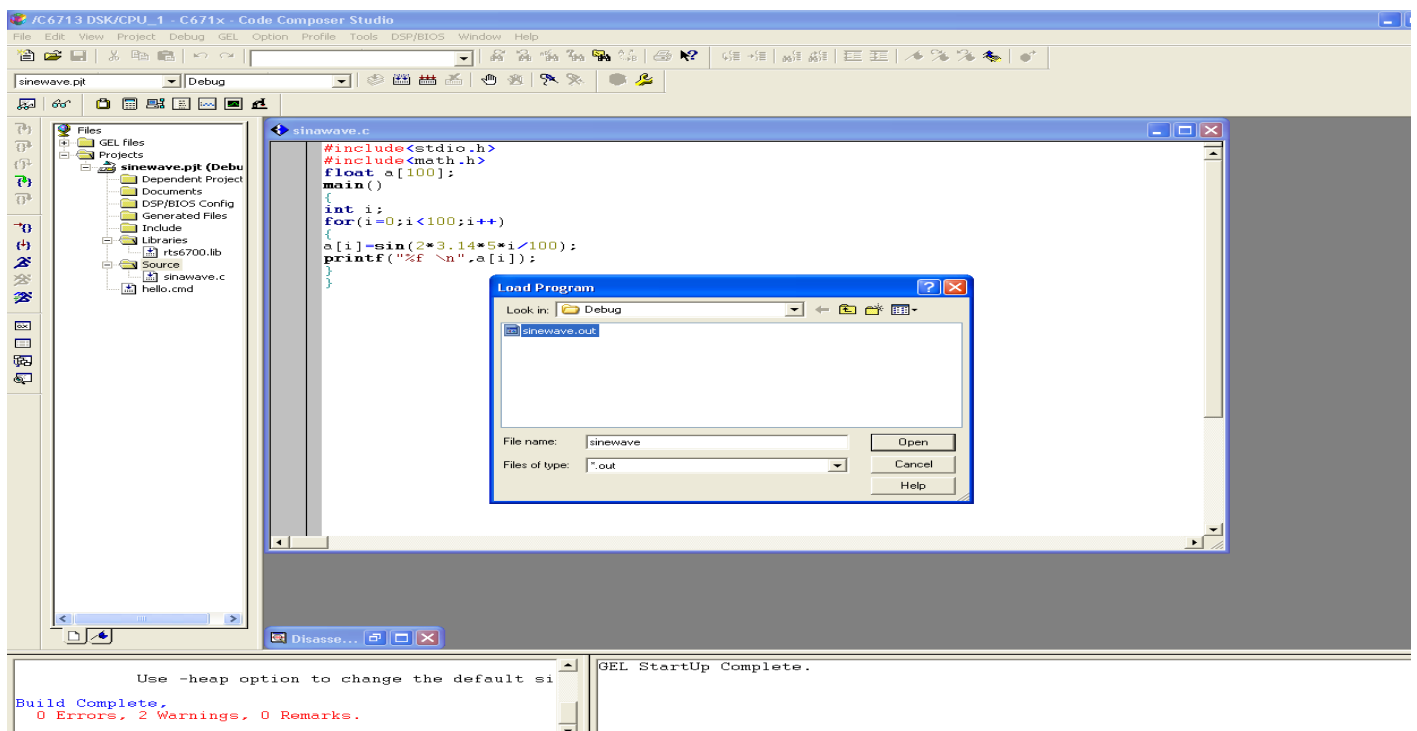
Step 8: Project → Compile File



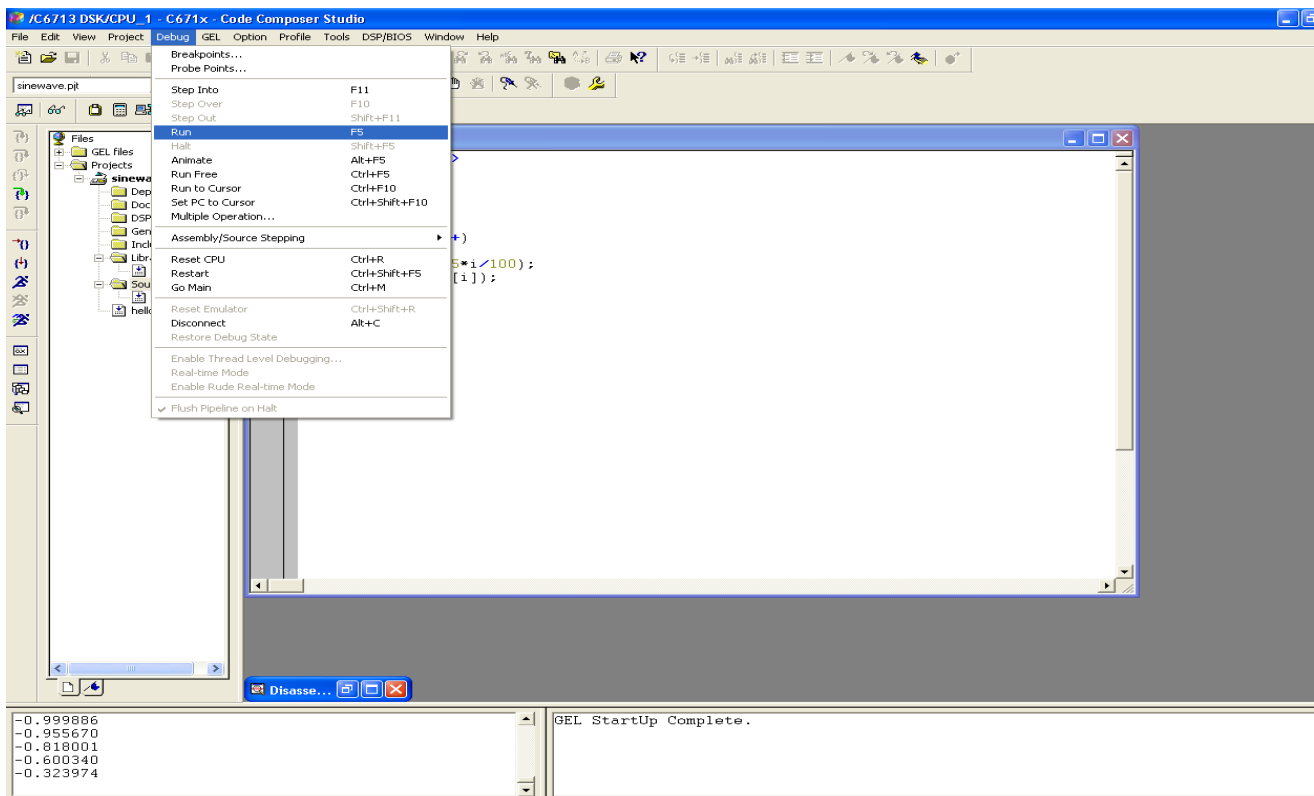
Step 9: Project → Build all



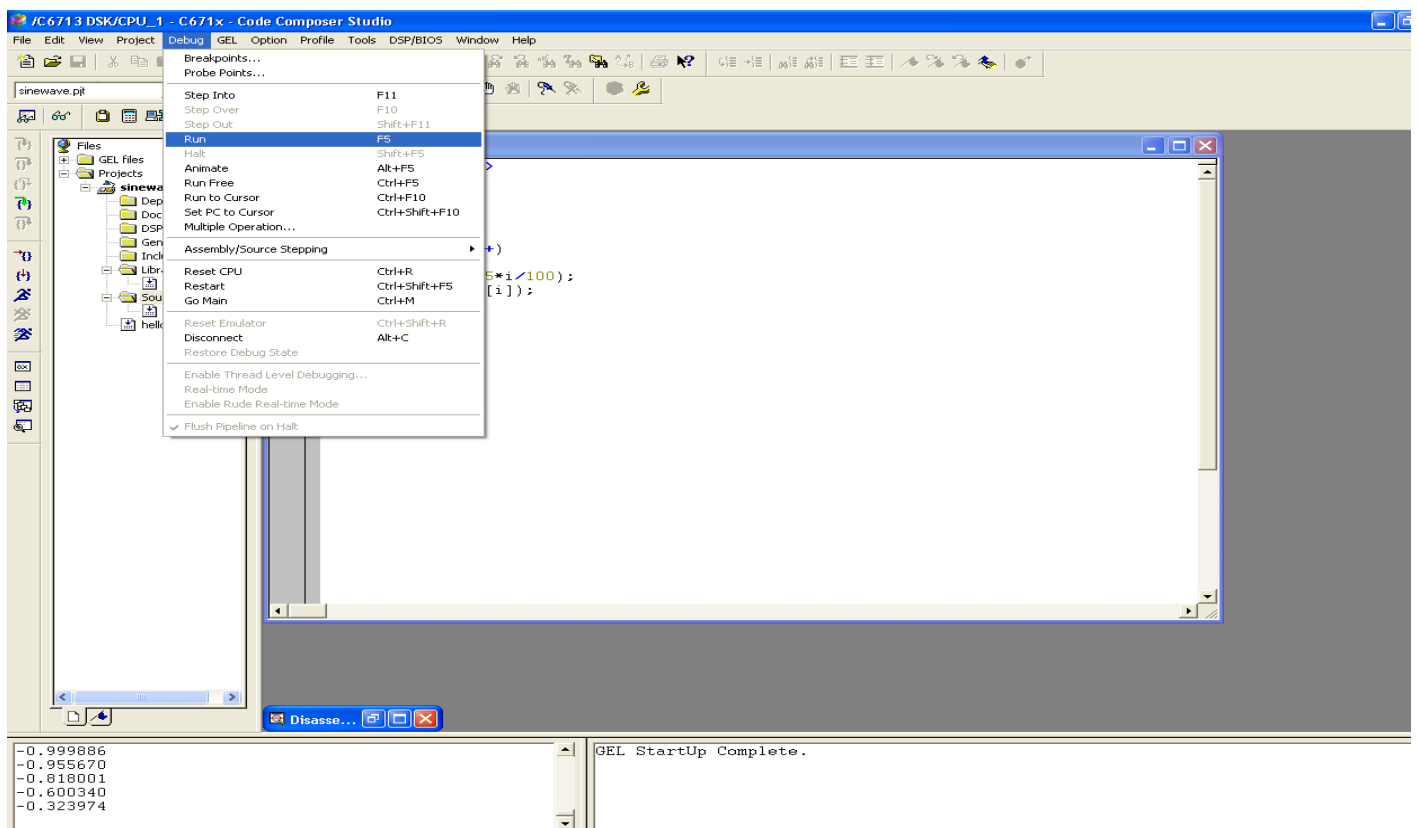
Step 10: File → Load Program → Debug → Output File



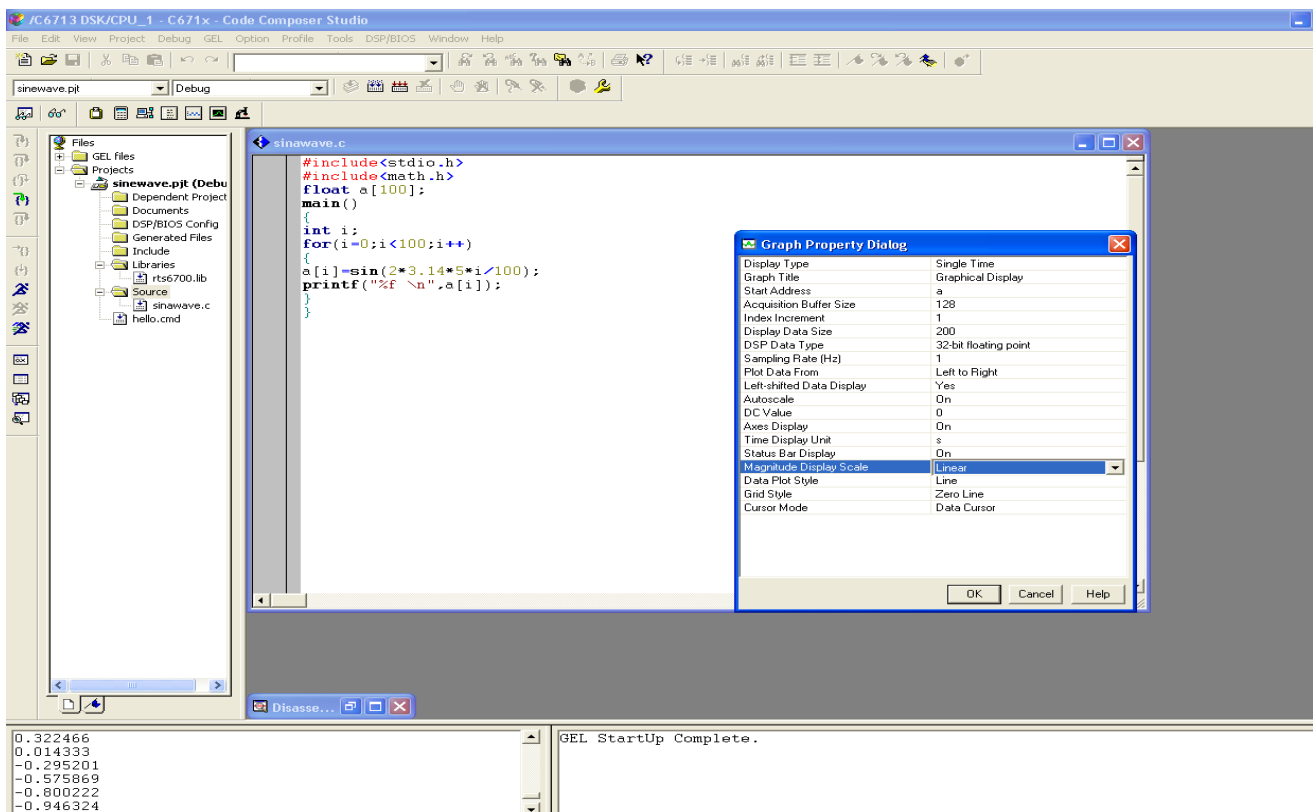
Step 11: Debug → Run



Step 12: View → Graph → Time/Frequency



Step 13: Start Address 'a' and DSP data type- 32 bit floating point



Step 14: We will get th required output!!!!

2) FIND DFT / IDFT OF GIVEN DISCRETE TIME SIGNAL.

PROGRAM :

```
#include<stdio.h>

#include<math.h>

int N,k,n,i;

float pi=3.1416,sumre=0, sumim=0,out_real[8]={0.0}, out_imag[8]={0.0};

int x[32];

void main(void)

{

printf(" enter the length of the sequence\n");

scanf("%d",&N);

printf(" enter the sequence\n");

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

scanf("%d",&x[i]);

for(k=0;k<N;k++)

{

sumre=0;

sumim=0;

for(n=0;n<N;n++)

{

sumre=sumre+x[n]* cos(2*pi*k*n/N);

sumim=sumim-x[n]* sin(2*pi*k*n/N);

}

out_real[k]=sumre;

out_imag[k]=sumim;

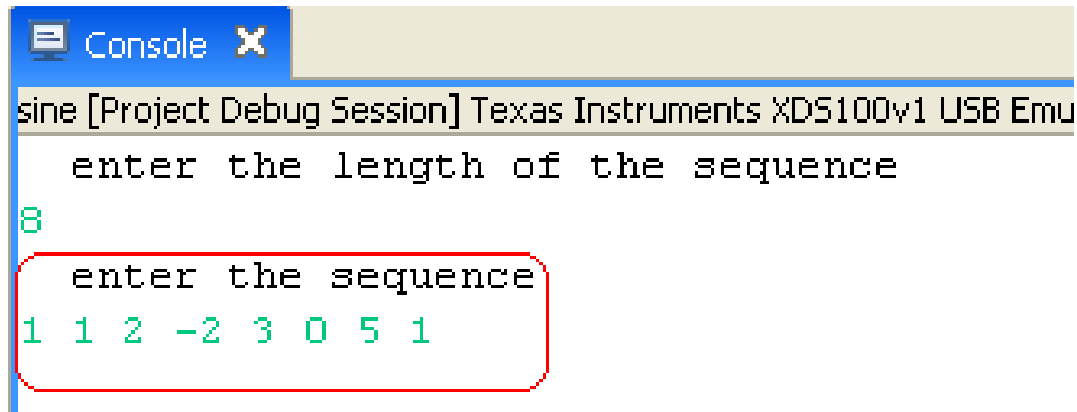
printf("X([%d])=\t%f\t+\t%fi\n",k,out_real[k],out_imag[k]);

}
```

}

RESULT :

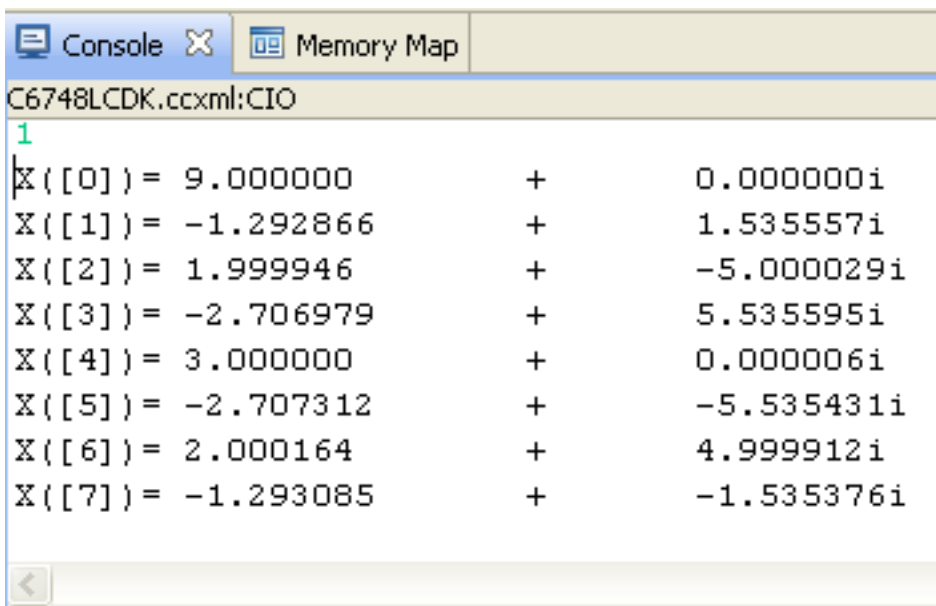
INPUT ;



The screenshot shows a console window titled "Console" with a close button. The text inside the console reads: "sine [Project Debug Session] Texas Instruments XDS100v1 USB Emu", followed by the prompt "enter the length of the sequence", the input "8", another prompt "enter the sequence", and the input "1 1 2 -2 3 0 5 1". The input sequence is highlighted with a red rounded rectangle.

```
sine [Project Debug Session] Texas Instruments XDS100v1 USB Emu
enter the length of the sequence
8
enter the sequence
1 1 2 -2 3 0 5 1
```

OUTPUT :



The screenshot shows a console window titled "Console" with a close button and a "Memory Map" button. The text inside the console reads: "C6748LCDK.ccxml:CIO", followed by the input "1", and then a list of complex numbers: "X([0]) = 9.000000 + 0.000000i", "X([1]) = -1.292866 + 1.535557i", "X([2]) = 1.999946 + -5.000029i", "X([3]) = -2.706979 + 5.535595i", "X([4]) = 3.000000 + 0.000006i", "X([5]) = -2.707312 + -5.535431i", "X([6]) = 2.000164 + 4.999912i", and "X([7]) = -1.293085 + -1.535376i".

```
C6748LCDK.ccxml:CIO
1
X([0]) = 9.000000 + 0.000000i
X([1]) = -1.292866 + 1.535557i
X([2]) = 1.999946 + -5.000029i
X([3]) = -2.706979 + 5.535595i
X([4]) = 3.000000 + 0.000006i
X([5]) = -2.707312 + -5.535431i
X([6]) = 2.000164 + 4.999912i
X([7]) = -1.293085 + -1.535376i
```

3) IMPLEMENTATION OF FFT OF GIVEN SEQUENCE

PROGRAM :

```
//fft.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, 64 -> sampling 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*samples[i].imag);
}
} //end of main

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]).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)

{

templ.real = (Y[j]).real;

templ.imag = (Y[j]).imag;

(Y[j]).real = (Y[i]).real;

(Y[j]).imag = (Y[i]).imag;

(Y[i]).real = templ.real;

(Y[i]).imag = templ.imag;

}

}

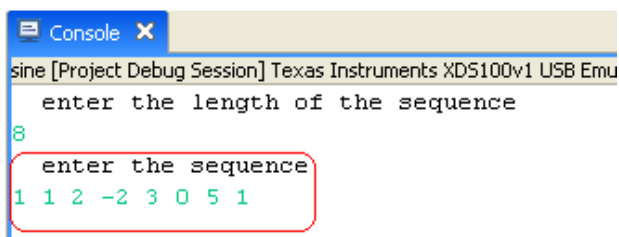
return;

}

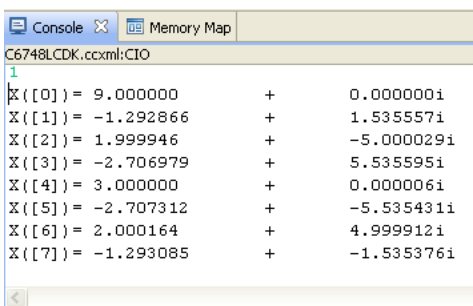
```

RESULT :

INPUT :



OUTPUT :



4) DESIGN AND IMPLEMENTATION OF IIR BUTTERWORTH / CHEBYSHEV (LP/HP) FILTER.

PROGRAM :

```
// L138_iir_intr.c
// IIR filter implemented using second order sections
// integer coefficients read from file
#include "L138_LCDK_aic3106_init.h"
// bs1800int.cof
#define NUM_SECTIONS 3

int b[NUM_SECTIONS][3] = {
{11538, -4052, 11538},
{32768, 599, 32768},
{32768, -22852, 32768} };

int a[NUM_SECTIONS][3] = {
{32768, -5832, 450},
{32768, 17837, 26830},
{32768, -35076, 27634} };

int w[NUM_SECTIONS][2] = {0};

interrupt void interrupt4() //interrupt service routine
{
    int section;    // index for section number
    int input;      // input to each section
    int wn, yn;     // intermediate and output values in each stage
    input = input_left_sample();
//    input = (int)prbs();

    for (section=0 ; section< NUM_SECTIONS ; section++)
```

```

{
    wn = input - ((a[section][1]*w[section][0])>>15) -
((a[section][2]*w[section][1])>>15);

    yn = ((b[section][0]*wn)>>15) + ((b[section][1]*w[section][0])>>15) +
((b[section][2]*w[section][1])>>15);

    w[section][1] = w[section][0];

    w[section][0] = wn;

    input = yn;                // output of current section will be input to next
}

output_left_sample((int16_t)(yn)); // before writing to codec

return;                        //return from ISR
}

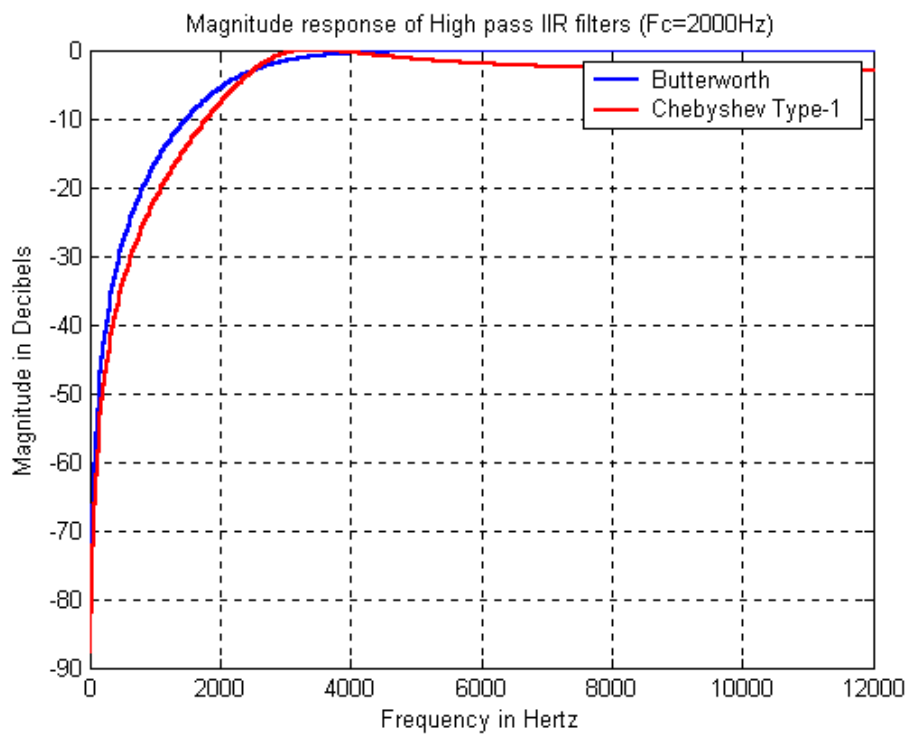
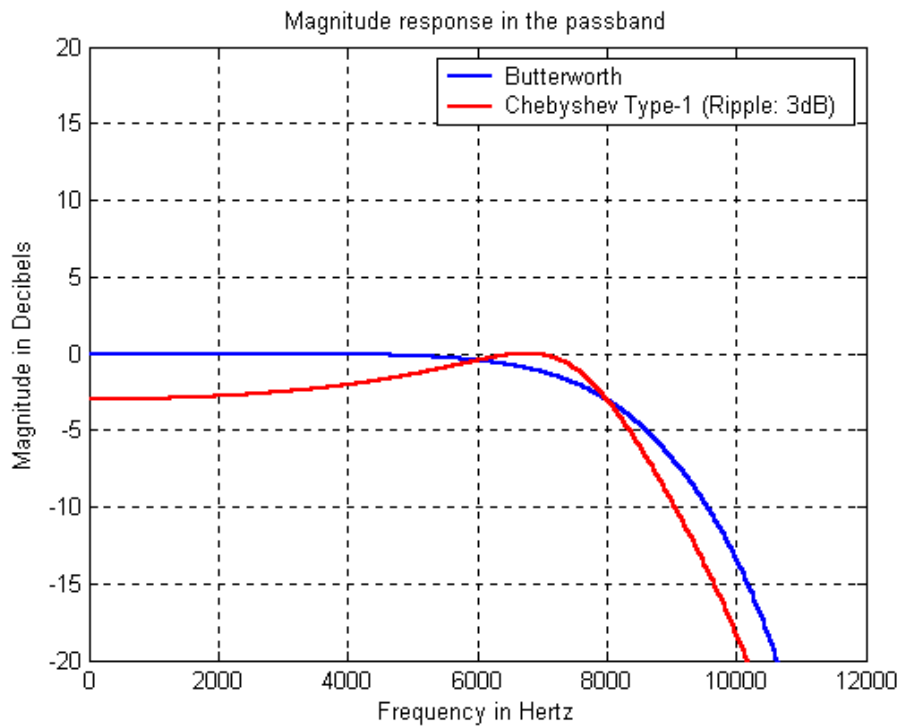
int main(void)

{
    L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);

    while(1);
} // end of main()

```


RESULT :



5) DESIGN AND IMPLEMENTATION OF FIR WITH LOW PASS / HIGH PASS FILTER USING ANY THREE WINDOWING TECHNIQUES. PLOT ITS MAGNITUDE AND PHASE RESPONSES.

PROGRAM :

```
// L138_fir_intr.c

#include "L138_LCDK_aic3106_init.h"

#define N 5

float h[N] = {

2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001,2.0000E-001

};float x[N]; // filter delay line

interrupt void interrupt4(void)

{

    short i;

    float yn = 0.0;

    x[0] = (float)(input_left_sample()); // input from ADC

    for (i=0 ; i<N ; i++)                // compute filter output

        yn += h[i]*x[i];

    for (i=(N-1) ; i>0 ; i--)            // shift delay line

        x[i] = x[i-1];

    output_left_sample((uint16_t)(yn)); // output to DAC

    return;

}

int main(void)

{

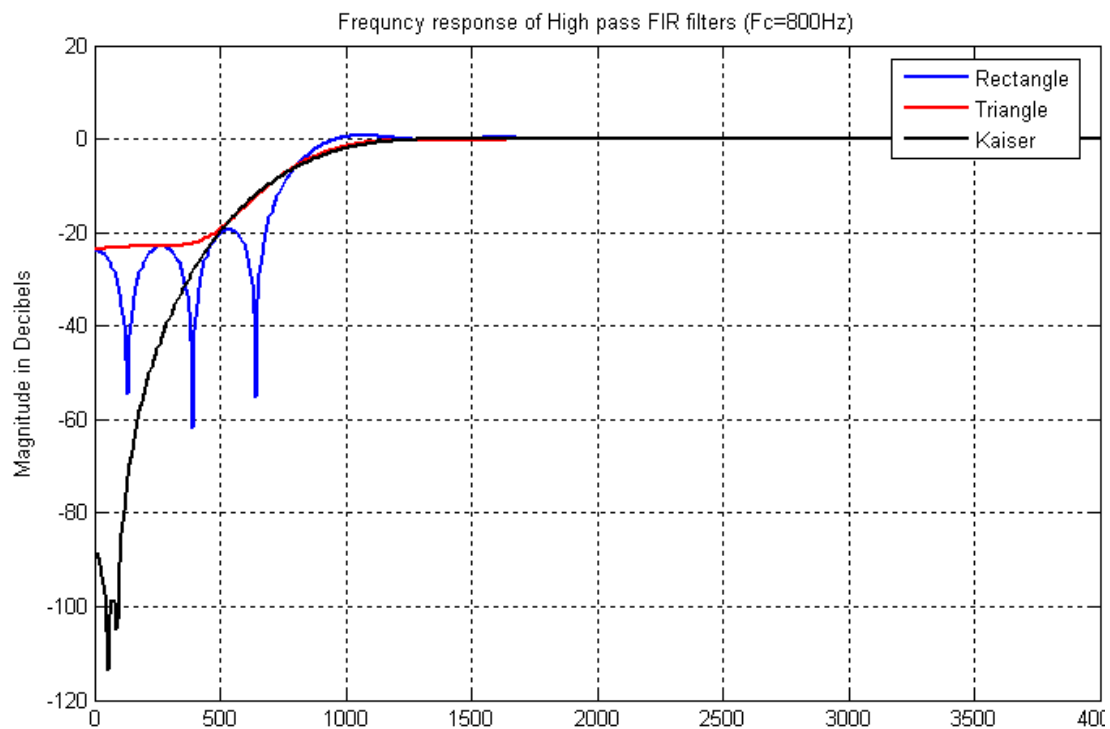
    L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPUT);

    while(1);

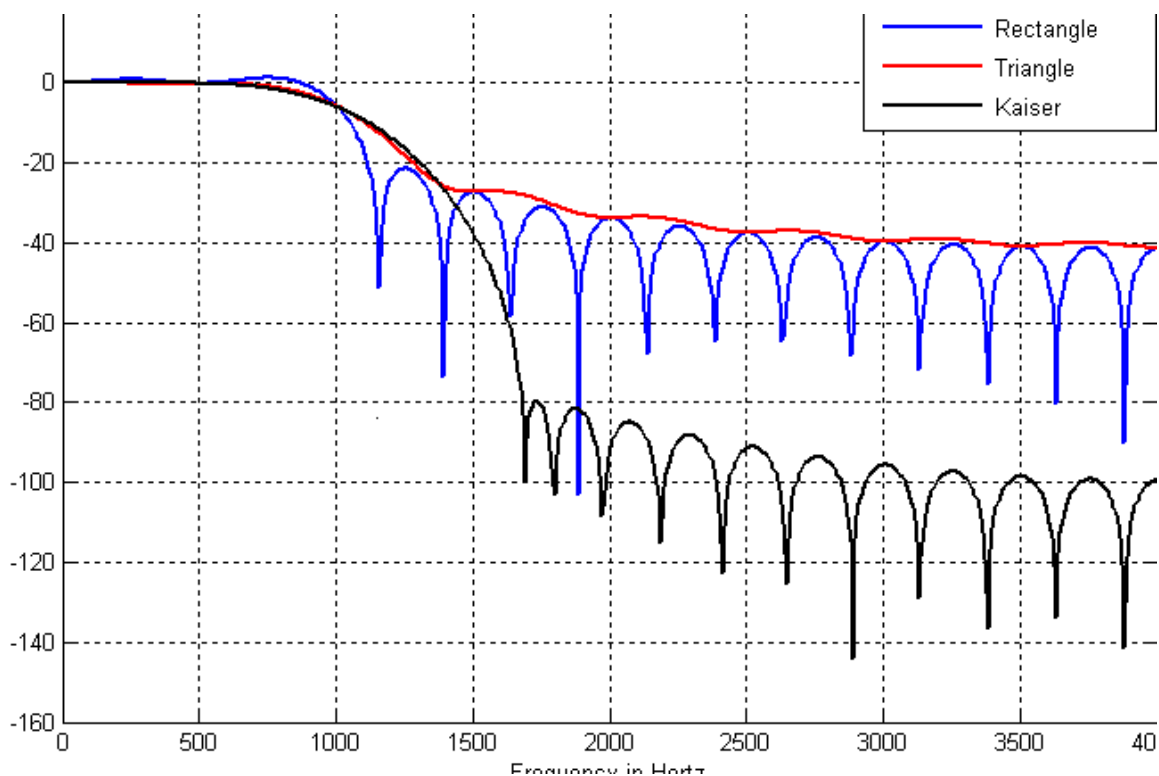
}
```

RESULT :

High Pass FIR filter($F_c = 800\text{Hz}$).



Low Pass FIR filter ($F_c = 1000\text{Hz}$)



ADDITIONAL PROGRAM

6) Design of FIR filter using windowing technique and verify the frequency response of the filter

Program:

```
#include<stdio.h>

#include<math.h>

#define pi 3.1415

int n,N,c;

float wr[64],wt[64];

void main()

{ printf("\n enter no. of samples,N= :"); scanf("%d",&N);

printf("\n enter choice of window function\n 1.rect \n 2. triang \n c= :"); scanf("%d",&c);

printf("\n elements of window function are:");

switch(c)

{

case 1:

for(n=0;n<=N-1;n++)

{

wr[n]=1;

printf(" \n wr[%d]=%f",n,wr[n]);

}

break;

case 2:

for(n=0;n<=N-1;n++)

{

wt[n]=1-(2*(float)n/(N-1));

printf("\n wt[%d]=%f",n,wt[n]);

}

break; }}
```