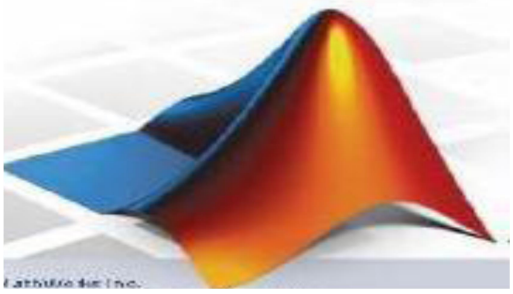
# MUFFAKHAM JAH COLLEGE OF ENGINEERING & TECHNOLOGY

Banjara Hills Road No 3, Hyderabad- 34
www.mjcollege.ac.in

## ELECTRICAL ENGINEERING DEPARTMENT

## LABORATORY MANUAL

# DIGITAL SIGNAL PROCESSING LAB



**For**

## B.E VI- SEMESTER EEE/EIE AICTE/MC
## 2021-22

**Prepared by**

**Dr. Mohd. Abdul Muqeet**
**Assoc. Professor, EED**

**WITH EFFECT FROM THE ACADEMIC YEAR 2020-2021**

# DSP LAB
# (COMMON TO EEE & EIE)

| | |
|---|---|
| Instruction | 2 Periods per week |
| Duration of University Examination | 2 Hours |
| University Examination (SEE) | 50 Marks |
| Sessional (CIE) | 25 Marks |

**PC463EE (EEE)**
**PC507EE (EIE)**

1. Generation of different discrete signal sequences and Waveforms.
2. Basic Operations On Discrete Time Signals
3. DFT Computation and FFT Algorithms.
4. Verification of Convolution Theorem.
5. Verification of sampling theorem.
6. Design of Butterworth and Chebyshev LP and HP filters.
7. Design of LPF using Rectangular, Hamming and Kaiser Windows.
8. To perform linear and circular convolution for the given sequences.
9. Design and implementation of FIR and IIR filter.
10. Computation of DFT using DIT and DIF algorithm.
11. Generation of basic waves.
12. Impulse response.

**At least ten experiments should be completed in the semester**

# Index

## Cycle –I

**[1].** Waveform generation -Square, Triangular and Trapezoidal.
**[2].** Verification of Convolution Theorem-comparison Circular and Linear Convolutions.
**[3].** Computation of DFT, IDFT using Direct and FFT methods.
**[4].** Verification of Sampling Theorem
**[5].** Design of Butterworth and Chebyshev of LP & HP filters.
**[6].** Design of LPF using rectangular and Hamming, Kaiser Windows.

## Cycle –II

**[7].** To verify Linear Convolution using TMS320C6713 DSK
**[8].** Generation of Sine wave and square wave using TMS320C6713 DSK and Code Composer Studio.
**[9].** Computation of DFT and DIT FFT using TMS320C6713 DSK and Code Composer Studio.
**[10].** Generating the Responses of Low Pass and High Pass IIR filters using DSP Trainer Kit (TMS320C6713)

# Cycle-I

# INTRODUCTION

MATLAB, which stands for **MAT**rix **LAB**oratory, is a state-of-the-art mathematical 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 and is used extensively in both academia and industry. It is an interactive program for *numerical* computation and data visualization, which along with its programming capabilities provides a very useful tool for almost all areas of science and engineering.

At its core ,MATLAB is essentially a set (a "toolbox") of routines (called "m files" or "mex files") that sit on your computer and a window that allows you to create new variables with names (e.g. voltage and time) and process those variables with any of those routines (e.g. plot voltage against time, find the largest voltage, etc).
It also allows you to put a list of your processing requests together in a file and save that combined list with a name so that you can run all of those commands in the same order at some later time. Furthermore, it allows you to run such lists of commands such that you pass in data.

## MATLAB Windows:

MATLAB works with through these 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

## The Current Directory Window

The Current Directory window displays a current directory with a listing of its contents. There is navigation capability for resetting the current directory to any directory among those set in the path. This window is useful for finding the location of particular files and scripts so that they can be edited, moved, renamed, deleted, etc. The default current directory is the Work subdirectory of the original MATLAB installation directory

## The Command History Window

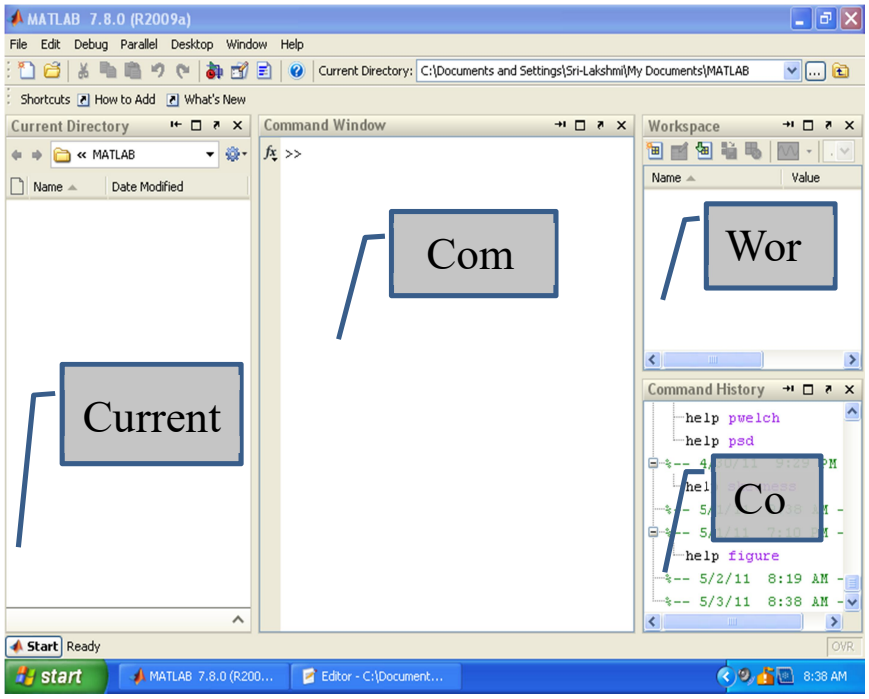
The Command History window, at the lower left in the default desktop, contains a log of commands that have been executed within the Command window. This is a convenient feature for tracking when developing or debugging programs or to confirm that commands were executed in a particular sequence during a multistep calculation from the command line.

## 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.
Typing quit in the command prompt>> quit, will close MATLAB Development Environment.
For any clarification regarding plot etc, which are built in functions type help topic i.e. help plot

## Basic Instructions in MATLAB

1. **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]
```

2. **F= 20: 1: 100**

```
F = [20 21 22 23 24 ..... 100]
```

3. **T= 0:1/ pi: 1**

```
T= [0, 0.3183, 0.6366, 0.9549]
```

4. **zeros (1, 3)** The above instruction creates a vector of one row and three columns whose values are zero Output= [0 0 0]

5. **Transpose a vector**

Suppose `T= [1 2 3],`
Then transpose

```
T'= 1
  2
    3
```

6. **Empty vector**

```
Y = []
Y =
    []
```

6. **Matrix Operation**

```
a)If a = [ 1 2 3] b = [4 5 6]
    a.*b = [4 10 18]
b)If v = [0:2:8]
          v = [0 2 4 6 8]
  v(1:3)
    ans [0 2 4]
 v(1:2:4)
    ans[ 0 4]
c) A = [1 2 3; 3 4 5; 6 7 8]
   A =
         1 2 3
         3 4 5
         6 7 8
   A(2,3)
      ans 5
   A(1:2,2:3)
       ans =
             2 3
             4 5
   A(:,2)
       ans =
             2
             4
             7
  A(3,:)
      ans =
             6 7 8
```

## Operations on vector and matrices in MATLAB

MATLAB utilizes the following arithmetic operators;

| | |
|---|---|
| + | Addition |
| - | Subtraction |
| * | Multiplication |
| / | Division |
| ^ | Power Operator |
| ‘ | transpose |

## Relational operators in MATLAB

| Operator | Description |
|---|---|
| < | Less than |
| <= | Less than or equal to |
| > | Greater |
| >= | Greater or equal to |
| == | Equal to |
| ~= | Not equal to |

## Control Flow in MATLAB

1) Syntax of the **for loop** is shown below

```
for k = array
commands
end
```

The commands between **for** and **end** statements are executed for all values stored in the **array**.

2) Syntax for the **if loop**

```
if expression
    commands
end
```

This construction is used if there is one alternative only.

Two alternatives requires the following construction

```
if expression
    commands (evaluated if expression is true)
else
    commands (evaluated if expression is false)
end
```

**3) Syntax of the switch-case construction is**

```
switch expression (scalar or string)
    case value1 (executes if expression evaluates to value1)
        commands
    case value2 (executes if expression evaluates to value2)
        commands
    ...
    otherwise
        statements
end
```

Switch compares the input expression to each case value. Once the match is found it executes the associated commands.

**Basic Functions in MATLAB**

1) **Plot** Syntax: plot (x,y)

Plots vector y versus vector x. If x or y is a matrix, then the vector is plotted versus the rows or columns of the matrix.

**2) Stem** Syntax: stem(Y)

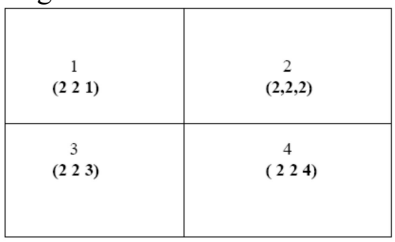
Discrete sequence or "stem" plot.

Stem (Y) plots the data sequence Y as stems from the x axis terminated with circles for the data value. If Y is a matrix then each column is plotted as a separate series.

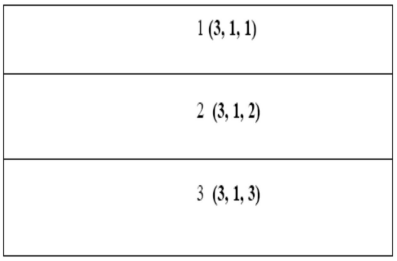
**3) Subplot** Syntax: Subplot (2 2 1)

This function divides the figure window into rows and columns.

Subplot (2 2 1) divides the figure window into 2 rows and 2 columns 1 represent number of the figure.




Subplot (3 1 2) divides the figure window into 3 rows and 1 column 2 represent number of the figure




4) **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.

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

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

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

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

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

8) **grid on** Syntax : grid on

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

# Experiment – 1

**Aim :-** To generate the waveform for the following signals using MATLAB.

1) Sine Wave signal
2) Cosine Wave signal
3) Saw Tooth Wave signal
4) Square Wave signal
5) Triangular Wave signal
6) Trapezoidal Wave signal

**Apparatus:** Matlab Software, PC

**Algorithm:-**

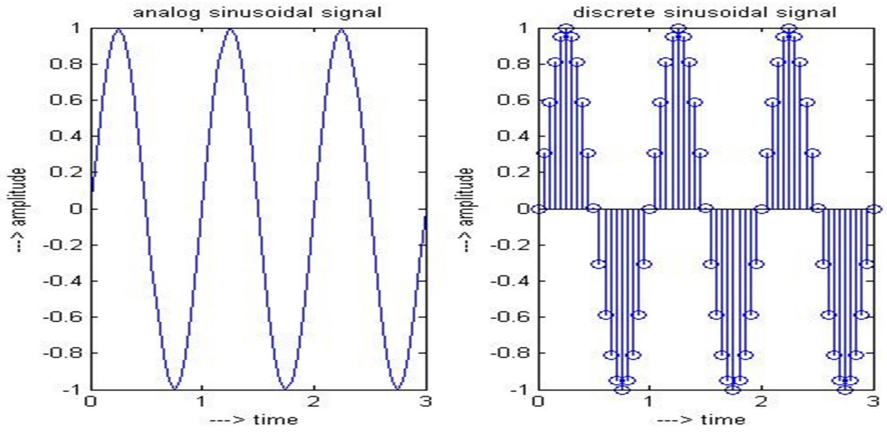
1) Enter the number of cycles, period and amplitude for respective waves.
2) Generate the signals using corresponding general formula.
3) Plot the graph.

**Program:**

```
1)% To generate a sinusoidal signal
clear all;
close all;clc;
N = input('enter the number of cycles....');
t = 0:0.05:N;
x = sin(2*pi*t);
subplot(121);
plot(t,x);
xlabel('---> time');
ylabel('---> amplitude');
title('analog sinusoidal signal');
subplot(122);
stem(t,x);
xlabel('---> time');
ylabel('---> amplitude');
title('discrete sinusoidal signal');
```

**Results:**

**enter the number of cycles....3**

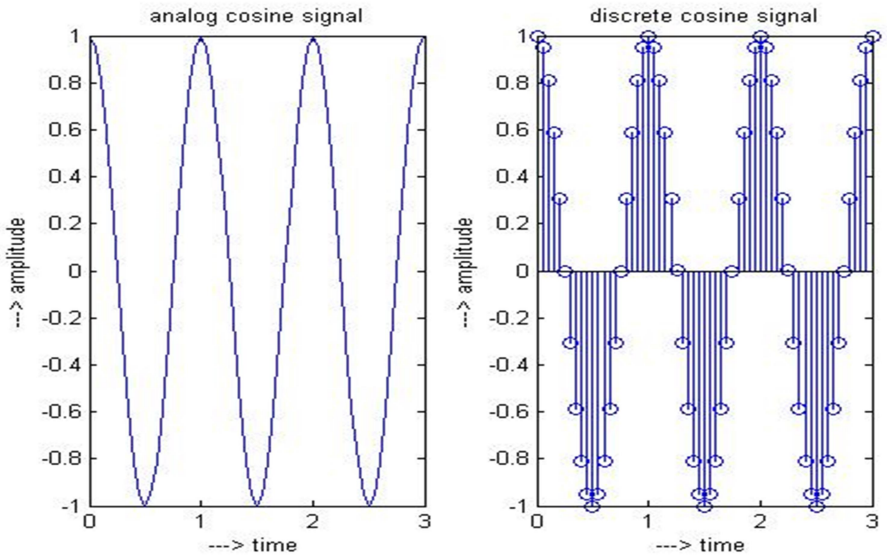



```
2)% To generate a Cosine Wave signal
```

```
clear all;
close all;
clc;
N = input('enter the number of cycles....');
t = 0:0.05:N;
x = cos(2*pi*t);
subplot(121);
plot(t,x);
xlabel('---> time');
ylabel('---> amplitude');
title('analog cosine signal');
subplot(122);
stem(t,x);
xlabel('---> time');
ylabel('---> amplitude');
title('discrete cosine signal');
```

**Results:**

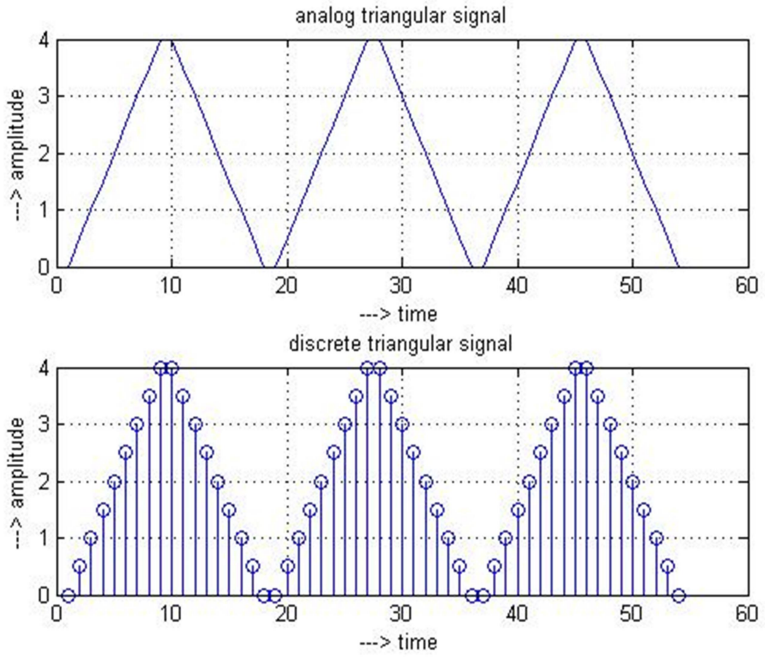
```
enter the number of cycles....3
```




3) % To generate a triangular signal

```
clc;
clear all;
close all;
N = input('enter the number of cycles....');
M = input('enter the amplitude....');
t1 = 0:0.5:M;
t2 = M:-0.5:0;
t = [];
for i = 1:N,
    t = [t,t1,t2];
end;
subplot(211);
plot(t); grid on;
xlabel('---> time');
ylabel('---> amplitude');
```

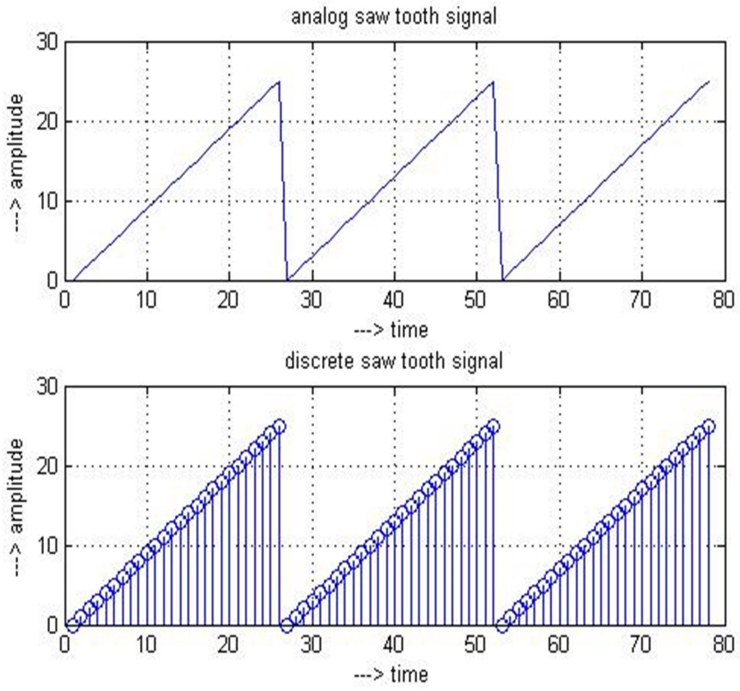
```
title('analog triangular signal');
subplot(212);
stem(t); grid on;
xlabel('---> time');
ylabel('---> amplitude');
title('discrete triangular signal');

Results:
enter the number of cycles....3
enter the amplitude....4
```




```
4) % To generate a saw tooth signal
clear all;
close all;
clc;
N = input('enter the number of cycles....');
t1 = 0:25;
t = [];
for i = 1:N,
    t = [t,t1];
end;
subplot(211);
plot(t); grid on;
xlabel('---> time');
ylabel('---> amplitude');
title('analog saw tooth signal');
subplot(212);
stem(t); grid on;
xlabel('---> time');

ylabel('---> amplitude');
title('discrete saw tooth signal');
```

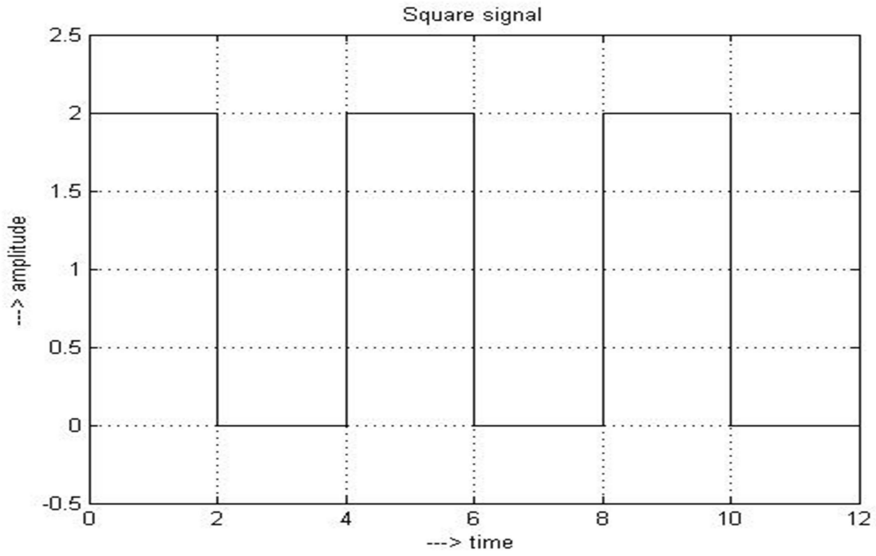
**Results:**

```
enter the number of cycles....3
```




```
5)% To generate a square signal
clear all;
close all; clc;
N = input('enter the number of cycles....');
M = input('enter the period....');
y = 0:0.001:2;
for j = 0:M/2:M*N;
    x = y;
    plot(j,x,'k'); grid on;
    hold on;
end;
for k = 0:M:M*N;
    x = k+y;
    m = 2;
    plot(x, m, 'k'); grid on;
    hold on;
end;
for k =2:M:M*N;
    x = k+y;
    m =0;
    plot(x, m, 'k'); grid on;
    hold on;
end;

hold off;
axis([0 12 -0.5 2.5])
xlabel('---> time');
```

```
ylabel('---> amplitude');
title('Square signal');

Results:
enter the number of cycles....4
enter the period....4
```



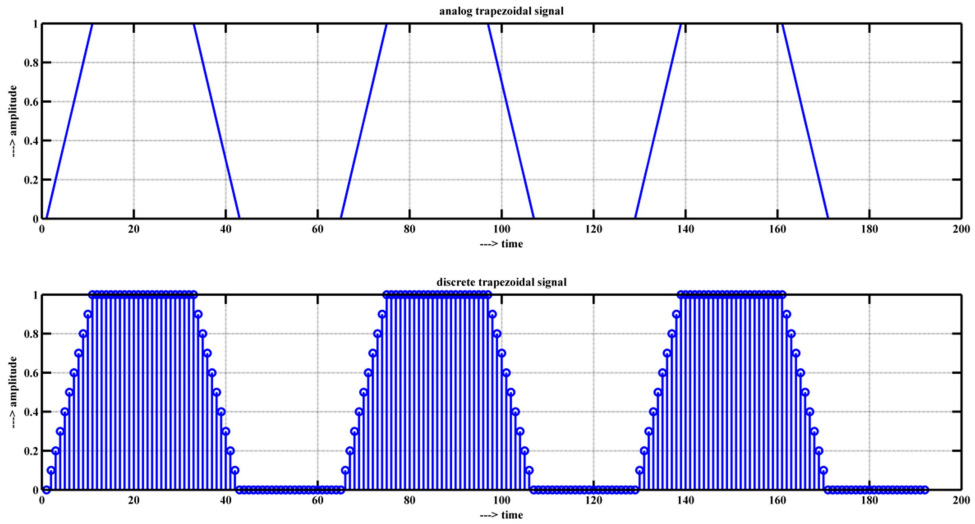

5)% To generate a Trapezoidal signal

```
clear all;
close all;
clc;
N = input('enter the number of cycles....');
LN=1;
x=0:0.1:LN; % 'x' is meant for linear rise %
a=length(x);
y=ones(1,a+10); % 'y' is meant for constancy %
z=LN:-0.1:0; % 'z' is meant for linear fall  %
y3=[x y z ];
%y4=[y3 y3 y3 y3];
y4=[];
for i = 1:N,
    y4=[y4,y3];
end;
subplot(211);
plot(y4); grid on;
xlabel('---> time');
ylabel('---> amplitude');
title('analog trapezoidal signal');
subplot(212);
stem(y4); grid on;

xlabel('---> time');
ylabel('---> amplitude');
```

```
title('discrete trapezoidal signal');
```

**Results:**

```
enter the number of cycles....3
```




**Discussions on results:**

Thus different waveforms have been generated in Matlab and plotted with respect to time.

By performing the experimentation the student will be to

1. Discuss the effect of change in number of cycles on waveform.
2. Discuss the effect of change in time duration on the waveform
3. Discuss the application and significance of each waveform in digital signal processing.

# Experiment – 2

**Aim:** Write a Matlab program to verify Convolution Theorem-comparison Circular and Linear Convolutions.

a) Write a Matlab program to implement and verify Linear Convolution.

**Apparatus:** Matlab Software, PC

**Theory:**

The mathematical definition of convolution in discrete time domain

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

where $x$[n] is input signal, $h$[n] is impulse response, and $y$[n] is output. * denotes convolution. Here we multiply the terms of $x$[k] by the terms of a time-shifted $h$[n] and add them up.

In this equation, x(k), h(n-k) and y(n) represent the input to and output from the system at time n. Here one of the input is shifted in time by a value every time it is multiplied with the other input signal. Linear Convolution is quite often used as a method of implementing filters of various types.

**Algorithm:**

1) Give input sequence x[n].
2) Give impulse response sequence h[n].
3) Find the convolution y[n] using the matlab command CONV.
4) Plot x[n],h[n],y[n].

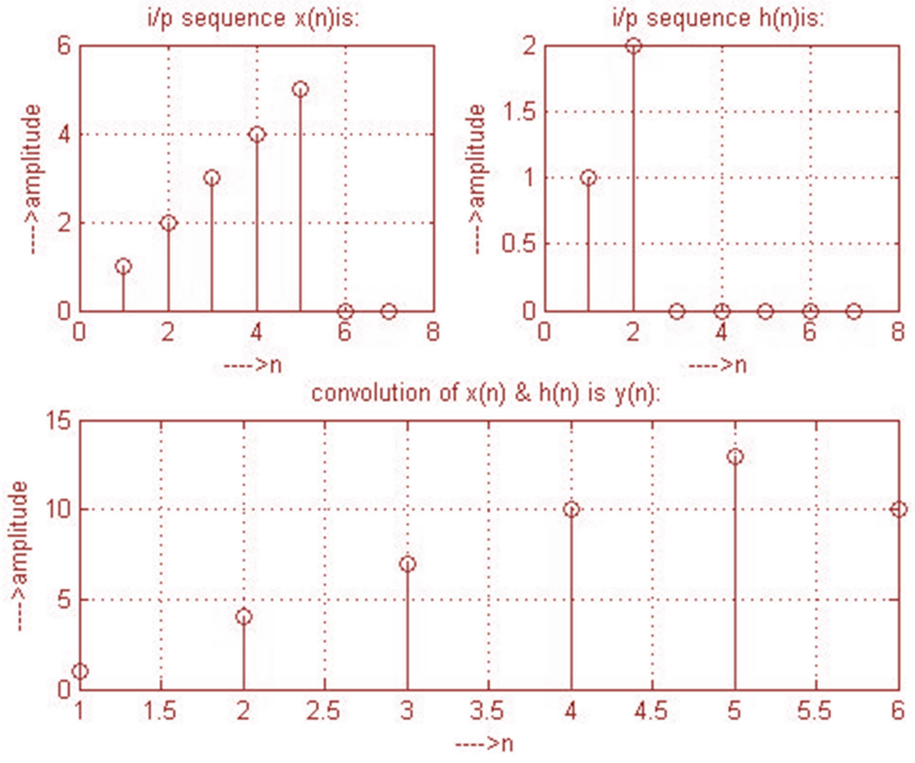
**Program:**

```
% MATLAB program for linear convolution
clc;
clear all;
close all;
disp('linear convolution program');
x=input('enter i/p x(n):');
m=length(x);
h=input('enter i/p h(n):');
n=length(h);
x=[x,zeros(1,n)];
subplot(2,2,1), stem(x);
title('i/p sequence x(n)is:');
xlabel('---->n');
ylabel('---->amplitude');grid;
h=[h,zeros(1,m)];
subplot(2,2,2), stem(h);
title('i/p sequence h(n)is:');
xlabel('---->n');
ylabel('---->amplitude');grid;
disp('convolution of x(n) & h(n) is y(n):');
y=zeros(1,m+n-1);
for i=1:m+n-1
```

```
y(i)=0;

for j=1:m+n-1
if(j<i+1)
     y(i)=y(i)+x(j)*h(i-j+1);
end
end
end
y
subplot(2,2,[3,4]),stem(y);
title('convolution of x(n) & h(n) is y(n):');
xlabel('---->n');
ylabel('---->amplitude');grid;
```

**Results:**

```
linear convolution program
enter i/p x(n):[1 2 3 4 5]
enter i/p h(n):[ 1 2]
convolution of x(n) & h(n) is y(n):

y =

     1     4     7    10    13    10
```



**b) Write a Matlab program to implement and verify Circular convolution of two given sequences.**

**Apparatus:** Matlab Software, PC

**Theory:**

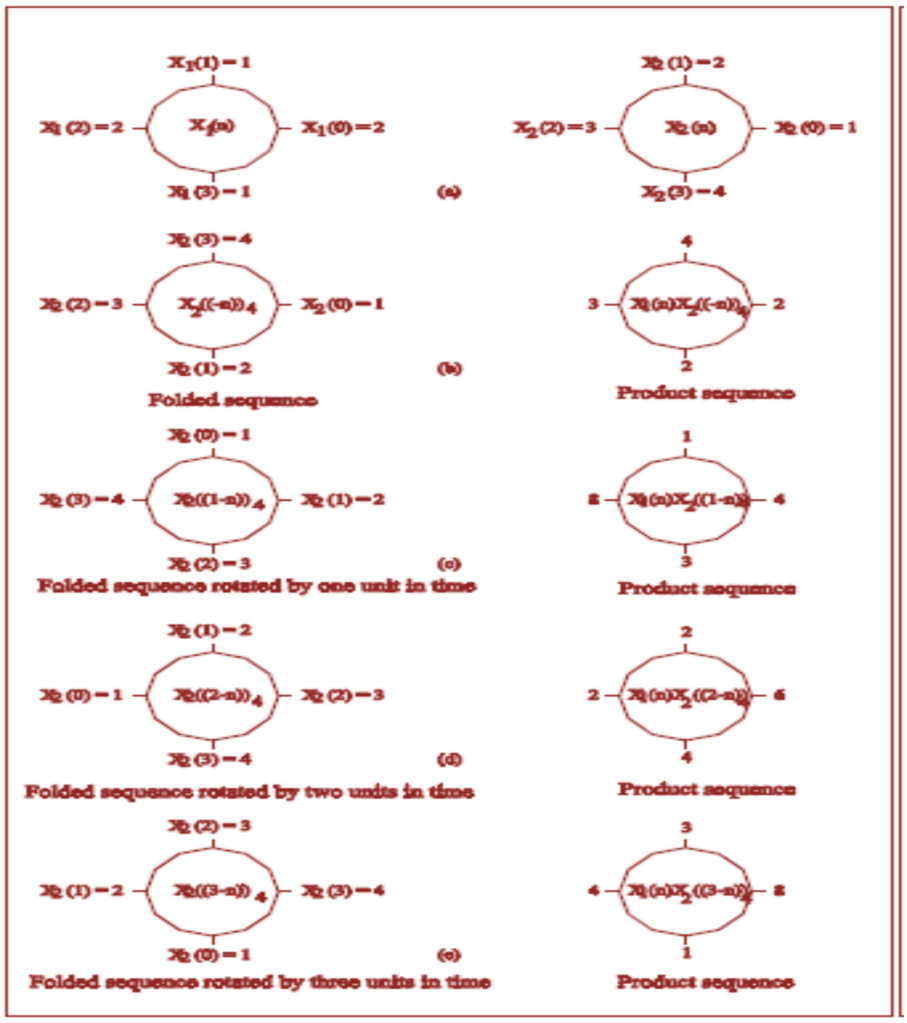
Circular convolution is another way of finding the convolution sum of two input signals. It resembles the linear convolution, except that the sample values of one of the input signals is folded and right shifted before the convolution sum is found. Also note that circular convolution could also be found by taking the DFT of the two input signals and finding the product of the two frequency domain signals. The Inverse DFT of the product would give the output of the signal in the time domain which is the circular convolution output. The two input signals could have been of varying sample lengths. But we take the DFT of higher point, which ever signals levels to. For eg. If one of the signal is of length 256 and the other spans 51 samples, then we could only take 256 point DFT. So the output of IDFT would be containing 256 samples instead of 306 samples, which follows N1+N2 – 1 where N1 & N2 are the lengths 256 and 51 respectively of the two inputs. Thus the output which should have been 306 samples long is fitted into 256 samples. The 256 points end up being a distorted version of the correct signal. This process is called circular convolution.
Circular convolution is explained using the following example.
The two sequences are

$$x1\ (n) = \{2,1,2,1\}$$
$$x2\ (n) = \{1,2,3,4\ \}$$

Each sequence consists of four nonzero points. For purpose of illustrating the operations involved in circular convolution it is desirable to graph each sequence as points on a circle. Thus the sequences x1 (n) and x2 (n) are graphed as illustrated in the **fig**.We note that the sequences are graphed in a counterclockwise direction on a circle.This stablishes the reference direction in rotating one of sequences relative to the other. Now, y (m) is obtained by circularly convolving x (n) with h (n).

**Algorithm:**

1) Give input sequence x[n].
2) Give impulse response sequence h[n].
3) Find the Circular Convolution y[n] using the DFT method.
4) Plot x[n],h[n],y[n].



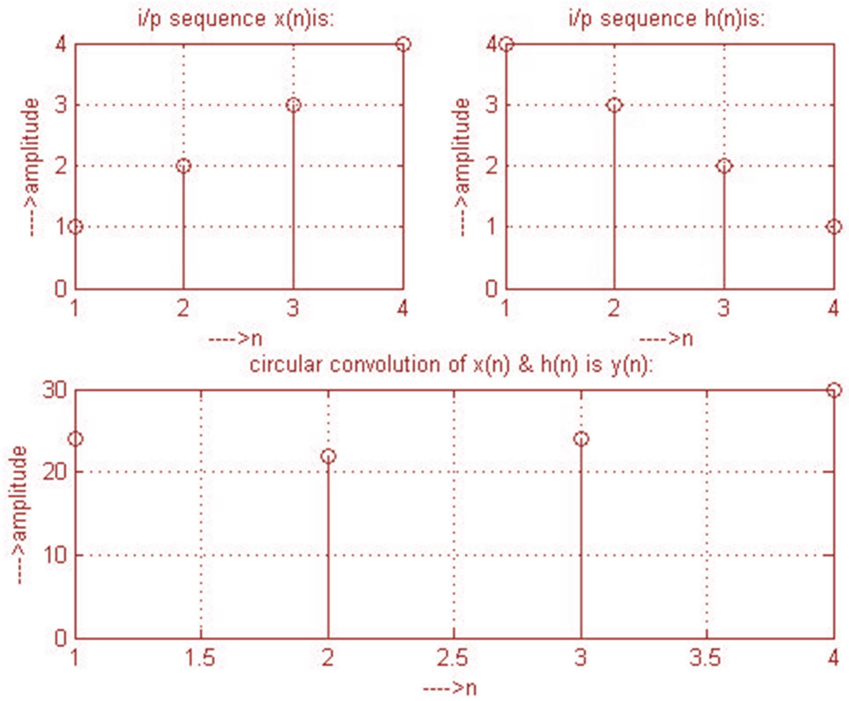

**Program:**

```
clc;
clear all;
close all;
disp('Circular convolution program');
x=input('enter i/p x(n):');
m=length(x);
h=input('enter i/p h(n):');
n=length(h);
subplot(2,2,1), stem(x);
title('i/p sequence x(n)is:');
xlabel('---->n');
ylabel('---->amplitude');grid;
subplot(2,2,2), stem(h);
title('i/p sequence h(n)is:');
xlabel('---->n');
ylabel('---->amplitude');grid;
disp('circular convolution of x(n) & h(n) is y(n):');
y1=fft(x,n);
y2=fft(h,n);
y3=y1.*y2;
```

```
y=ifft(y3,n);
y
subplot(2,2,[3,4]),stem(y);
title('circular convolution of x(n) & h(n) is y(n):');
xlabel('---->n');
ylabel('---->amplitude');grid;
```

**Result:**

```
Circular convolution program
enter i/p x(n):[1 2 3 4]
enter i/p h(n):[4 3 2 1]
circular convolution of x(n) & h(n) is y(n):

y =

    24    22    24    30
```




**Discussions on results:**

Thus the Linear convolution and circular convolution for discrete time signals are obtained mathematically and graphically .Through this experiment student will be able to

1) Discuss the effect on results if zero padding is used in the program.
2) Discuss the effect on results if zero padding is not used in the program.
3) Discuss the results in obtaining the circular convolution without using frequency domain technique.
4) Discuss the difference between linear convolution and circular convolution.

# Experiment – 3

**Aim:** Write a Matlab program for computation of DFT and IDFT using Direct and FFT method.

**Apparatus:** Matlab Software, PC

**Theory:**

**DFT:**

Discrete Fourier Transform (DFT) is used for performing frequency analysis of discrete time signals. DFT gives a discrete frequency domain representation whereas the other transforms are continuous in frequency domain. The N point DFT of discrete time signal x[n] is given by the equation

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

The inverse DFT allows us to recover the sequence x[n] from the frequency samples

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

**FFT:**

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. Evaluating the sums of DFT directly would take O($N$ 2) arithmetical operations. An FFT is an algorithm to compute the same result in only O($N$ log $N$) operations. In general, such algorithms depend upon the factorization of $N$, but there are FFTs with O($N$ log $N$) complexity for all $N$, even for prime $N$. Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/$N$ factor, any FFT algorithm can easily be adapted for it as well.

**Algorithm:**

1) Get the input sequence
2) Find the DFT of the input sequence using direct equation of DFT.
3) Find the IDFT using the direct equation.
4) Find the FFT of the input sequence using MATLAB function.
5) Find the IFFT of the input sequence using MATLAB function.
4) Display the above outputs using stem function.

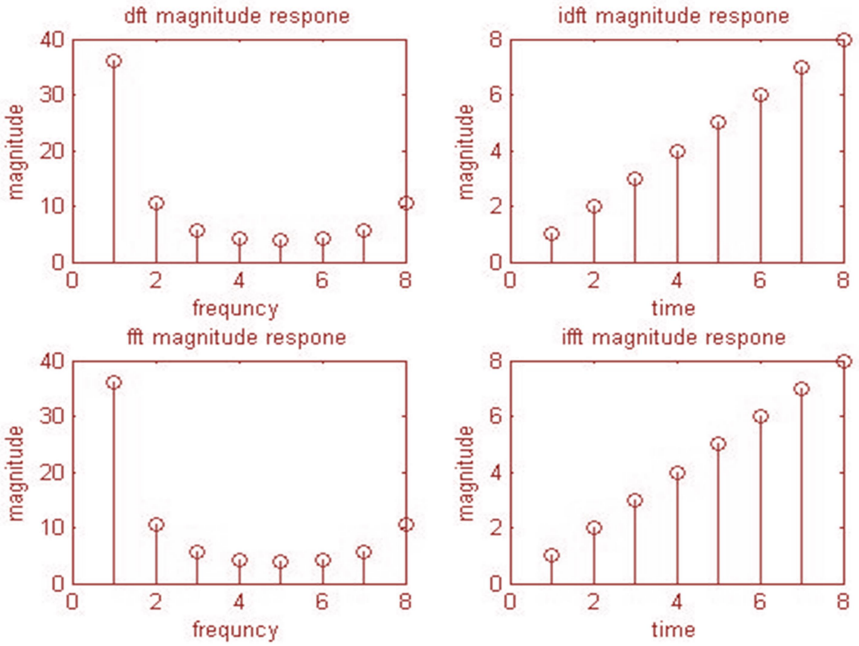
**Program:**

```
%********** Direct DFT ***********
clc;close all;clear all;
xn=input('enter 8 inputs');
N=length(xn);
n=0:N-1;
k=0:N-1;
wn=exp((-1i*2*pi*n'*k)/N);
xf=wn*xn';
subplot(2,2,1);
stem(abs(xf));
title('dft magnitude respone');
ylabel('magnitude');
xlabel('frequncy');
% ******* Direct IDFT **********
```

```
WN=exp((1i*2*pi*n'*k)/N);
pn=WN*xf/N;
subplot(2,2,2);
stem(abs(pn));
title('idft magnitude respone');
ylabel('magnitude');
xlabel('time');
%******* FFT Method**********
xp=fft(xn,N);
subplot(2,2,3);
stem(abs(xp));
title('fft magnitude respone');
ylabel('magnitude');
xlabel('frequncy');
%******** IFFT method *********
xw=ifft(xp,N);
subplot(2,2,4);
stem(abs(xw));
title('ifft magnitude respone');
ylabel('magnitude');
xlabel('time');
```

**Results:**

```
enter 8 inputs[1 2 3 4 5 6 7 8]
```




**Discussions on results:**

Thus from the results students will be able to

1. Discuss that the Fourier transform of a discrete time signal is also called as Signal Spectrum.
2. Discuss the changes in the results due to more number of inputs in the given sequences in finding the DFT and FFT.
3. Discuss that FFT performs faster and take less computational time compared to DFT.

# Experiment – 4

**Aim:** Write a Matlab program to verify Sampling Theorem

**Apparatus:** Matlab Software, PC

**Theory:**

**Sampling Theorem:** The sampling theorem, attributed to Nyquist, Shannon, Kotelnikov and Whittaker, is useful when calculating the sampling frequency required for use in the Analog-to-Digital converter.
The theorem states that a band limited signal can be reconstructed exactly if it is sampled at a rate at least twice the maximum frequency component in it.
The maximum frequency component of g(t) is fm. To recover the signal g(t) exactly from its samples it has to be sampled at a rate fs=2fm. The minimum required sampling rate fs = 2fm is called Nyquist rate.
Sampling is also a process of converting a continuous time signal (analog signal) x(t) into a discrete time signal x[n],which is represented as a sequence of numbers. (A/D Converter)
Converting back x[n] into analog (resulting in) x(t) is the process of reconstruction.(D/A Converter)

**Algorithm:**

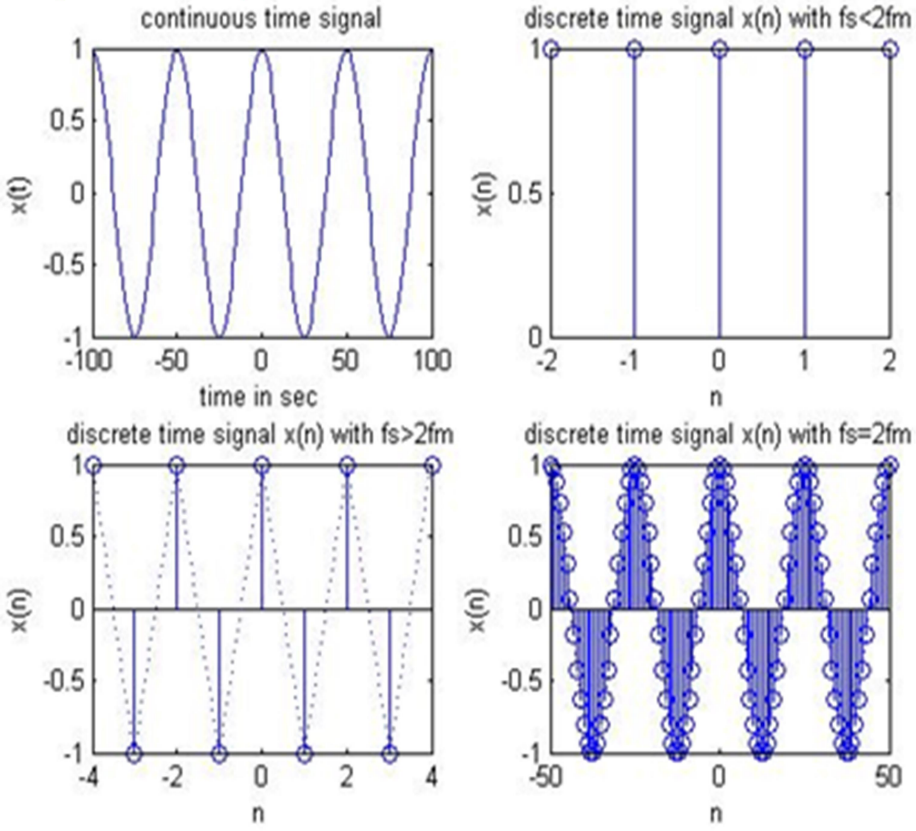
1) Input the desired frequency $f_m$ (for which sampling theorem is to be verified)
2) Generate the cosine wave, i.e a continuous-time signal given mathematically as, $x(t) = \cos(2\pi f_m t)$ where **f** represents the frequency and *t* the time.
3) Generate the discrete-time signals for Undersampling, Nyquist sampling andoversampling conditions.
   oversampled & under sampled conditions after sampling at instants n1, n2, n3 which are given as, ,.
   a. To do this for **under sampling**, choose sampling frequency **`fs1<2*fm`**. For this sampling rate T1=1/fs1,
   b. For **Nyquist Sampling**, choose sampling frequency **`fs2=2*fm`**. For this sampling rate T2=1/fs2.
   c. For **Over Sampling**, choose sampling frequency **`fs2>fd`**.
4) Plot the waveforms and hence prove sampling theorem.

**Program:**

```
clc;
clear all;
%define analog signal for comparison
t=-100:01:100;
fm=0.02;
x=cos(2*pi*t*fm);
subplot(2,2,1);
plot(t,x);
xlabel('time in sec');
ylabel('x(t)');
title('continuous time signal');
```

```
%simulate condition for undersamplingi.e.,fs1<2*fm
fs1=0.02;
n=-2:2;
x1=cos(2*pi*fm*n/fs1);
subplot(2,2,2);
stem(n,x1);
hold on
subplot(2,2,2);
plot(n,x1,':');
title('discrete time signal x(n) with fs<2fm');
xlabel('n');
ylabel('x(n)');
%condition for Nyquist plot
fs2=0.04;
n1=-4:4;
x2=cos(2*pi*fm*n1/fs2);
subplot(2,2,3);
stem(n1,x2);
hold on
subplot(2,2,3);
plot(n1,x2,':');
title('discrete time signal x(n) with fs>2fm');
xlabel('n');
ylabel('x(n)');
%condition for oversampling
n2=-50:50;
fs3=0.5;
x3=cos(2*pi*fm*n2/fs3);
subplot(2,2,4);
stem(n2,x3);
hold on
subplot(2,2,4);
plot(n2,x3,':');
xlabel('n');
ylabel('x(n)');
title('discrete time signal x(n) with fs=2fm');
```

**Results:**




## Discussions on Results:

This experiment verifies the sampling theorem in Matlab for undersampling, Nyquist sampling and oversampling.

Thus from the results students will be able to

1) Discuss the effect of undersampling for the given signal
2) Discuss the effect of Nyquist sampling for the given signal
3) Discuss the effect of oversampling for the given signal.

# Experiment – 5

**Aim: -**To Design and generate IIR Butterworth/ Chebyshev LP/HP Filter using MATLAB

**Apparatus Required: -** MATLAB Software, PC

**Theory:**

The Digital Filter Design problem involves the determination of a set of filter coefficients to meet a set of design specifications. These specifications typically consist of the width of the passband and the corresponding gain, the width of the stopband(s) and the attenuation therein; the band edge frequencies (which give an indication of the transition band) and the peak ripple tolerable in the passband and stopband(s).

The design of IIR filters is closely related to the design of analog filters, which is a widely studied topic. An analog filter is usually designed and a transformation is carried out into the digital domain. Two transformations exist – the **impulse invariant** transformation and the **bilinear** transformation.

**Analog to Digital Domain Mapping Techniques**

Digital Filters are designed by using the values of both the past outputs and the present input, an operation brought about by convolution. If such a filter is subjected to an impulse then its output need not necessarily become zero. The impulse response of such a filter can be infinite in duration. Such a filter is called an *Infinite Impulse Response* filter or **IIR** filter. The infinite impulse response of such a filter implies the ability of the filter to have an infinite impulse response. This indicates that the system is prone to feedback and instability.

The experiment studies two different types of IIR filters Butterworth Filter, and Chebyschev I type Filters.

IIR filters are designed essentially by the Impulse Invariance or the Bilinear Transformation method.

**1) Impulse Invariance**

This procedure involves choosing the response of the digital filter as an equi-spaced sampled version of the analog filter.

1. Decide upon the desired frequency response
2. Design an appropriate analogue filter
3. Calculate the impulse response of this analogue filter
4. Sample the analogue filter's impulse response
5. Use the result as the filter coefficients

**2) Bilinear Transformation**:

The Bilinear Transformation method overcomes the effect of aliasing that is caused to due the analog frequency response containing components at or beyond the *Nyquist* Frequency. The bilinear transform is a method of compressing the infinite, straight analogue frequency axis to a finite one long enough to wrap around the unit circle once only. This is also sometimes called frequency warping. This introduces a distortion in the frequency. This is undone by pre-warping the critical frequencies of the analog filter (cut-off frequency, center frequency) such that when the analog filter is transformed into the digital filter, the designed digital filter will meet the desired specifications.

## Filter Types

### Butterworth Filters

Butterworth filters are causal in nature and of various orders, the lowest order being the best (shortest) in the time domain, and the higher orders being better in the frequency domain. Butterworth or maximally flat filters have a monotonic amplitude frequency response which is maximally flat at zero frequency response and the amplitude frequency response decreases logarithmically with increasing frequency.
A Butterworth filter is characterized by its magnitude frequency response,

$$|H(j\Omega)| = \frac{1}{\left[1+(\frac{\Omega}{\Omega_c})^{2N}\right]^{\frac{1}{2}}}$$

Where N is the order of the filter and Ωc is defined as the cutoff frequency where the filter magnitude is 1/√2 times the dc gain (**Ω=0).**

### Chebyshev Filters

Chebyshev filters are equiripple in either the passband or stopband. Hence the magnitude response oscillates between the permitted minimum and maximum values in the band a number of times depending upon the order of filters. There are two types of chebyshev filters. The chebyshev I filter is equiripple in passband and monotonic in the stopband, where as chebyshev II is just the opposite.
The Chebyshev low-pass filter has a magnitude response given by

$$|H(j\Omega)|^2 = \left(1+\varepsilon^2 T_N^2(\frac{\Omega}{\Omega_c})\right)^{-1}$$

where ϵ is a parameter related to the ripple present in the passband
$T_N(x)$ is given by

$$C_N(x) = \begin{Bmatrix} \cos(N\cos^{-1}x)\, for\, |x| \le 1, passband \\ \cos(N\cosh^{-1}x)\, for\, |x| \le 1, stopband \end{Bmatrix}$$

The magnitude response has equiripple pass band and maximally flat stop band. By increasing the filter order N, the Chebyshev response approximates the ideal response. The phase response of the Chebyshev filter is more non-linear than the Butter worth filter for a given filter length N.

### Algorithm:

1) Enter the pass band ripple (rp) and stop band ripple (rs).
2) Enter the pass band frequency (wp) and stop band frequency (ws).
3) Get the sampling frequency (fs).
4) Calculate normalized pass band frequency, and normalized stop band frequency w1 and w2 respectively.

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

5) Make use of the following function to calculate order of filter

Butterworth filter order

[n,wn]=buttord(w1,w2,rp,rs)

Chebyshev filter order

[n,wn]=cheb1ord(w1,w2,rp,rs)

6) Design an nth order digital lowpass Butterworth or Chebyshev filter using the following statements.

Butterworth filter

[b, a]=butter (n, wn)

Chebyshev filter

[b,a]=cheby1(n,0.5,wn)

**OR**

Design an nth order digital high pass Butterworth or Chebyshev filter using the following statement.

Butterworth filter

[b,a]=butter (n, wn,'high')

Chebyshev filter

[b,a]=cheby1 (n, 0.5, wn,'high')

7) Find the digital frequency response of the filter by using 'freqz( )' function

[H,w]=freqz(b,a,512,fs)

8) Calculate the magnitude of the frequency response in decibels (dB)

mag=20*log10 (abs (H))

9) Plot the magnitude response [magnitude in dB Vs normalized frequency (Hz]]

10) Calculate the phase response using an = angle (H)

11) Plot the phase response [phase in radians Vs normalized frequency (Hz)].

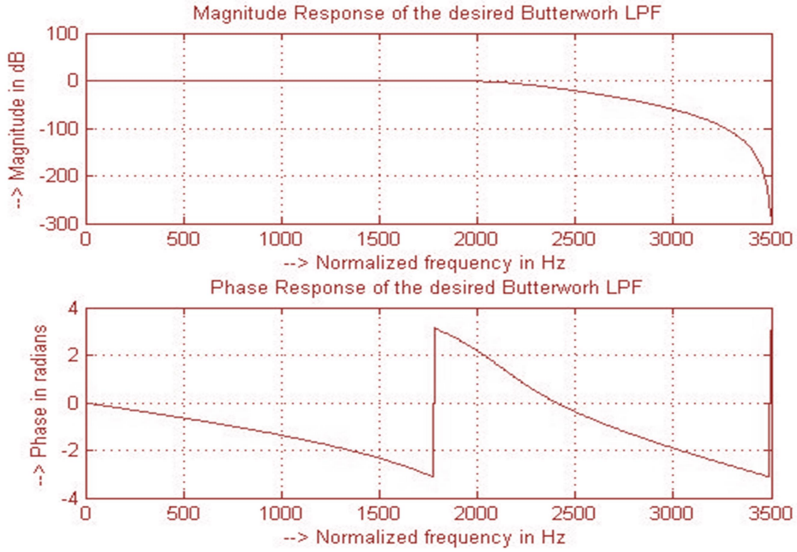
**Program:**

```
% IIR filters
clc; clear all; close all;
warning off;
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;%normalized pass band frequency
w2=2*ws/fs;%normalized stop band frequency
[n,wn]=buttord(w1,w2,rp,rs);% Find the order n and cut-
off frequency
ch=input('give type of filter 1:LPF,2:HPF');
switch ch
case 1
 disp('Frequency response of Butterworth IIR LPF is:');
 [b,a]=butter(n,wn); % Find the filter coefficient of LPF
 [H,w]=freqz(b,a,512,fs);% to get the transfer function
of the filter
 mag=20*log10(abs(H));
 phase=angle(H);
 subplot(211);
 plot(w,mag);grid on;
 ylabel('--> Magnitude in dB');
 xlabel('--> Normalized frequency in Hz');
```

```
title('Magnitude Response of the desired Butterworh
LPF');

subplot(212);

plot(w,phase);grid on;
 ylabel('--> Phase in radians');
 xlabel('--> Normalized frequency in Hz');
 title('Phase Response of the desired Butterworh LPF');
case 2
 disp('Frequency response of IIR Butterworth HPF is:');
 [b,a]=butter(n,wn,'high'); % Find the filter co-
efficients of HPF
 [H,w]=freqz(b,a,512,fs);% to get the transfer function
of the filter
 mag=20*log10(abs(H));
 phase=angle(H);
 subplot(211);
 plot(w,mag);grid on;
 ylabel('--> Magnitude in dB');
 xlabel('--> Normalized frequency in Hz');
 title('Magnitude Response of the desired Butterworh
HPF');
 subplot(212);
 plot(w,phase);grid on;
 ylabel('--> Phase in radians');
 xlabel('--> Normalized frequency in Hz');
 title('Phase Response of the desired Butterworh HPF');
end
```

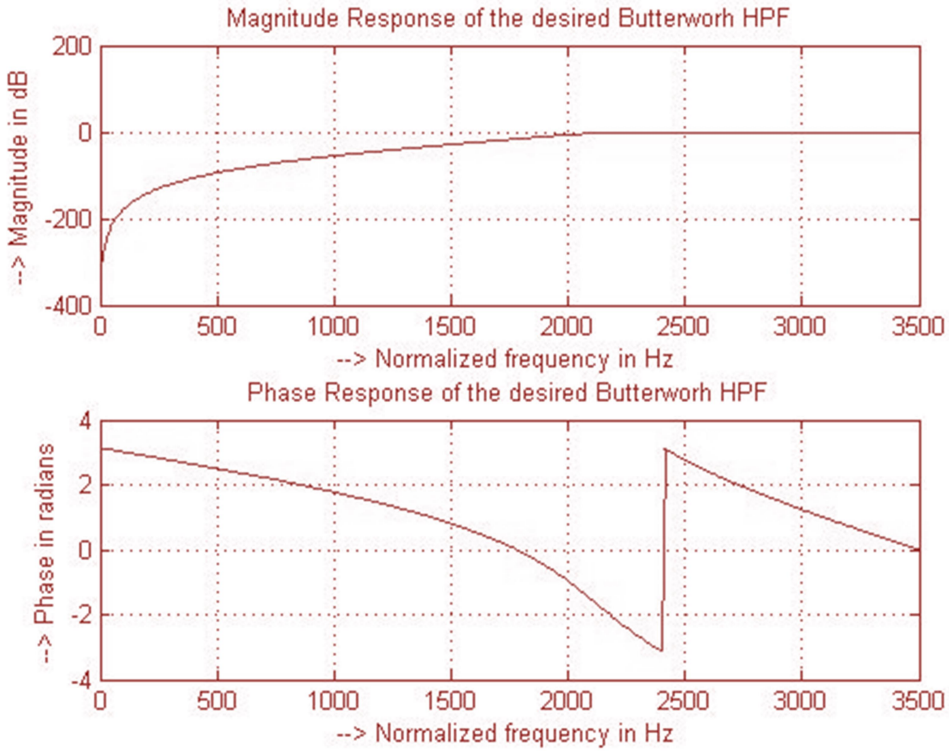
Results:

```
enter the IIR filter design specifications
enter the passband ripple   0.15
enter the stopband ripple    60
enter the passband freq     1500
enter the stopband freq     3000
enter the sampling freq     7000
give type of filter 1:LPF,2:HPF
1
Frequency response of Butterworth IIR LPF is:
```




## IIR HIGH PASS FILTER

```
enter the IIR filter design specifications
enter the passband ripple    0.15
enter the stopband ripple     60
enter the passband freq      1500
enter the stopband freq      3000
enter the sampling freq      7000
give type of filter 1:LPF,2:HPF
2
Frequency response of Butterworth IIR HPF is:
```

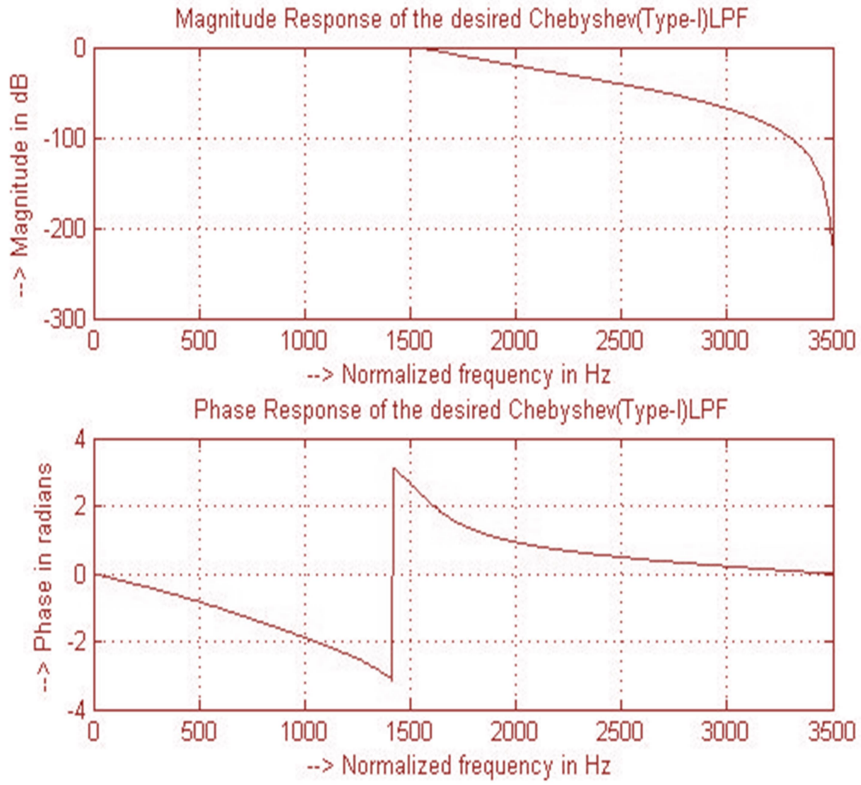


```
%To design a Chebyshev (Type-I) Low/High Pass Filter for the
given specifications
clc; clear all; close all;
disp('enter the IIR filter design specifications');
rp=input('enter the passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter the passband freq');
ws=input('enter the stopband freq');
fs=input('enter the sampling freq');

w1=2*wp/fs;%to get normalized pass band frequency
w2=2*ws/fs;% to get normalized stop band frequency

ch=input('give type of filter 1:LPF,2:HPF');
 % to get the order and cut-off frequency of the filter
[n,wn]=cheb1ord(w1,w2,rp,rs);
switch ch
 case 1

disp('Frequency response of Chebyshev IIR LPF is:');
  [b,a]=cheby1(n,0.5,wn);% to get the filter coefficients
  % to get the transfer function of the filter
  [H,w]=freqz(b,a,512,fs);
  mag=20*log10(abs(H));
  phase=angle(H);
  subplot(211);
  plot(w,mag);grid on;
  ylabel('--> Magnitude in dB');
  xlabel('--> Normalized frequency in Hz');
  title('Magnitude Response of the desired Chebyshev Type -I)
LPF');
  subplot(212);
  plot(w,phase);grid on;
  ylabel('--> Phase in radians');
  xlabel('--> Normalized frequency in Hz');
  title('Phase Response of the desired Chebyshev(Type-I)LPF');
 case 2
  disp('Frequency response of Chebyshev IIR HPF is:');
  % to get the filter coefficients
  [b,a]=cheby1(n,0.5,wn,'high');
  % to get the transfer function of the filter
  [H,w]=freqz(b,a,512,fs);
  mag=20*log10(abs(H));
  phase=angle(H);
  subplot(211);
  plot(w,mag);grid on;
  ylabel('--> Magnitude in dB');
  xlabel('--> Normalized frequency in Hz');
  title('Magnitude Response of the desired Chebyshev(Type-
I)HPF');
  subplot(212);
  plot(w,phase);grid on;
  ylabel('--> Phase in radians');

  xlabel('--> Normalized frequency in Hz');
  title('Phase Response of the desired Chebyshev(Type-I)HPF');
end
```

**Results:**

```
enter the IIR filter design specifications
enter the passband ripple      0.15
enter the stopband ripple      60
enter the passband freq        1500
enter the stopband freq        3000
enter the sampling freq        7000
give type of filter 1:LPF,2:HPF
1
```

**Frequency response of Chebyshev IIR LPF is:**

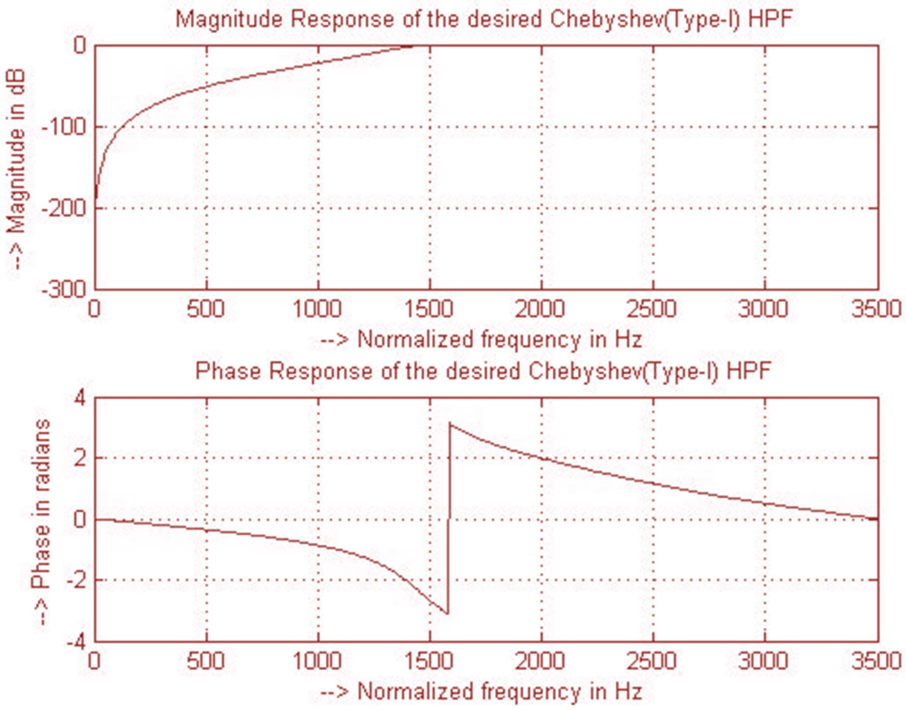



## High Pass Filter

**Result:**

```
enter the IIR filter design specifications
enter the passband ripple      0.15
enter the stopband ripple      60
enter the passband freq        1500
enter the stopband freq        3000
enter the sampling freq        7000
give type of filter 1:LPF,2:HPF
2
```

**Frequency response of Chebyshev IIR HPF is:**




## Discussions on results:

By this experiment we have studied the LP/HP IIR digital filter designing.

From the obtained results the students will be able to

1) Discuss the effect of order of the filer on magnitude response.
2) Discuss the effect of variation in pass band ripple, stop band ripple, pass band frequency, stop band frequency and sampling frequency respectively in designing the IIR Butterworth digital filter.
3) Discuss the effect of variation in pass band ripple, stop band ripple, pass band frequency, stop band frequency and sampling frequency respectively in designing the IIR Chebyshev digital filter.

# Experiment – 6

**Aim: -** Design and implementation of FIR Filter (LP/HP) to meet given specifications Using Windowing technique

a. Rectangular window
b. Hamming window
c. Kaiser window

**Apparatus:** Matlab Software, PC

**Theory:**

A linear-phase is required throughout the passband of the filter to preserve the shape of the given signal in the passband. A causal IIR filter cannot give linear-phase characteristics and only special types of FIR filters that exhibit center symmetry in its impulse response give the linear-space. A Finite Impulse Response (FIR) filter is a discrete linear time-invariant system whose output is based on the weighted summation of a finite number of past inputs.

A zero-phase frequency response of an ideal filter is given as

$$H_{LP}(e^{j\omega}) = \begin{cases} 1, |\omega| \le \omega_c, \\ 0, \omega_c < \omega \le \pi. \end{cases}$$

Hence time domain impulse response is

$$h_d[k] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d\left(e^{j\omega}\right)e^{j\omega k}\,d\omega = .. = \alpha\frac{\sin(\omega_c k)}{\omega_c k}$$

so the impulse response is doubly infinite, not absolutely summable, and therefore unrealizable.

By setting all impulse response coefficient outside the range $-M \le n \le M$ equal to zero, we arrival at a finite-length noncausal approximation of length $N = 2M+1$ which when shifted to the right yield the coefficients of a causal FIR lowpass filter:

$$h_{LP}[n] = \begin{cases} \dfrac{\sin(\omega_c(n-M))}{\pi(n-M)}, & 0 \le n \le N-1 \\ 0, otherwise \end{cases}$$

Gibbs phenomenon

The causal FIR filter obtained by simply truncating the impulse response coefficients of the ideal filters exhibit an oscillatory behavior in their respective magnitude responses which is more commonly referred to as the Gibbs phenomenon

Cause of Gibbs phenomenon:

The FIR filter obtained by truncation can be expressed as

$$h[n] = h_d[n]\cdot\omega[n]$$

$$H(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_d(e^{j\phi})\psi(e^{j(\omega-\phi)})d\phi$$

The window used to achieve simple truncation of the ideal filter is rectangular window

$$w_R[n] = \begin{cases} 1, 0 \le |n| \le M \\ 0, otherwise \end{cases}$$

Thus by applying windowing functions we can obtain FIR filters.
Available Fixed window functions Rectangular, Bartlett, Hamming, Hanning, Blackmann etc.
Hamming window function

$$w[n] = 0.54 + 0.46\cos(\frac{2\pi n}{2M+1}), -M \le n \le M$$

In Adjustable Window Functions, windows have been developed that provide control over ripple by means of an additional parameter.
Like Kaiser Window

$$w[n] = \frac{I_0\left\{\beta\sqrt{1-(n/M)^2}\right\}}{I_0(\beta)}, -M \le n \le M$$

Where $\beta$ is an adjustable parameter and $I_0(\beta)$ is a zero order Bessel function
To design a FIR filter order of the filter should be specified or can be calculated from the following equation

$$N = \frac{-20\log_{10}\left(\sqrt{r_p r_s}\right) - 13}{14.6\left(w_s - w_p\right)/2\pi}$$

rp=passband ripple, rs=stopband ripple, wp=passband frequency
ws=stopband frequency
Then from order of the filter we can find the length by which a window function can be applied.

## Algorithm:

## FIR Low Pass Filter design

1) Enter the pass band ripple (rp) and stop band ripple (rs).
2) Enter the pass band frequency (wp) and stop band frequency (ws).
3) Get the sampling frequency (fs), beta value for Kaiser window.
4) Calculate the analog pass band edge frequencies, w1 and w2.
   i. w1 = 2*wp/fs
   ii. w2 = 2*ws/fs
5) Calculate the order of the filter using the order equation.
6) Use switch condition and ask the user to choose either Rectangular Window or Hamming window or Kaiser window.

7) Use rectwin, hamming, Kaiser Commands
   Command **`fir1`** uses the window method of FIR filter design, If w(n) denotes a window, where $1 \le n \le N$, and the impulse response of the ideal filter is h(n), where $h_d(n)$ is the inverse Fourier transform of the ideal frequency response.

$$h[n] = h_d[n] \cdot \omega[n]$$

8) Calculate the digital frequency response using the command 'freqz()'
9) Calculate the magnitude of the frequency response in decibels
   m=20*log10 (abs(h))
10) Plot the magnitude response [magnitude in dB Vs normalized frequency (om/pi)]

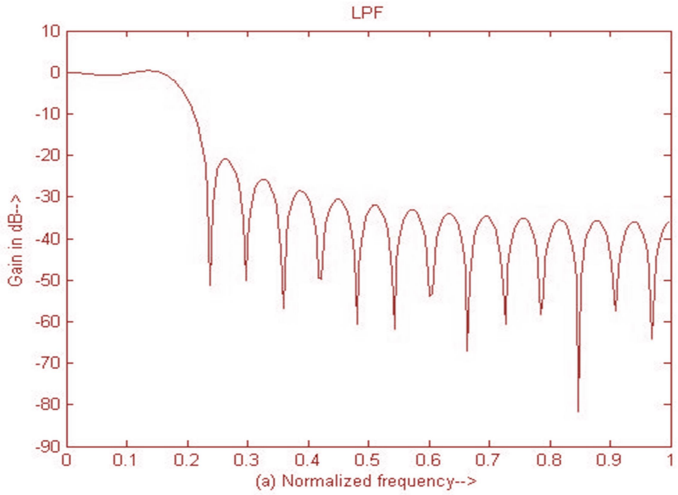
**Program:**

```
%FIR Low Pass/High pass filter design using
Rectangular/Hamming/Kaiser window
clc; clear all; close all;
rp=input('enter passband ripple');
rs=input('enter the stopband ripple');
wp=input('enter passband freq');
ws=input('enter stopband freq');
fs=input('enter sampling freq ');
beta=input('enter beta value');
w1=2*wp/fs;
w2=2*ws/fs;
num=-20*log10(sqrt(rp*rs))-13;
dem=14.6*(ws-wp)/fs;
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. Hamming 3.kaiser: \n ');
if(c==1)
    y=rectwin(n1);
    disp('Rectangular window filter response');
end
if (c==2)
    y=hamming(n1);
    disp('Hamming window filter response');
end
if(c==3)
    y=kaiser(n1,beta);
    disp('kaiser window filter response');
end

ch=input('give type of filter 1:LPF,2:HPF');
switch ch
    case 1
      b=fir1(n,w1,y);
      [h,o]=freqz(b,1,256);
      m=20*log10(abs(h));
      plot(o/pi,m);
      title('LPF');
      xlabel('(a) Normalized frequency-->');
      ylabel('Gain in dB-->');

    case 2
      b=fir1(n,w1,'high',y);
      [h,o]=freqz(b,1,256);
      m=20*log10(abs(h));
      plot(o/pi,m);
      title('HPF');
      xlabel('(b) Normalized frequency-->');
      ylabel('Gain in dB-->');
end
```

**Results:**

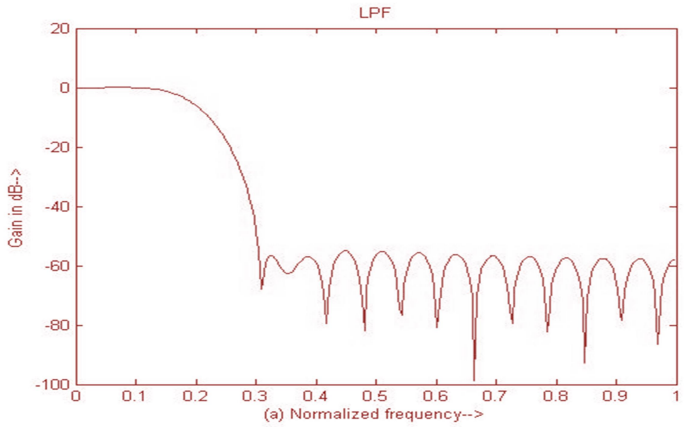
```
enter passband ripple          0.02
enter the stopband ripple      0.01
enter passband freq            1000
enter stopband freq            1500
enter sampling freq            10000
enter beta value                 5
enter your choice of window function 1. rectangular 2.
Hamming 3.kaiser:
 1
Rectangular window filter response
give type of filter 1:LPF,2:HPF
1:LPF
```

**Low pass FIR filter using Rectangular Window**



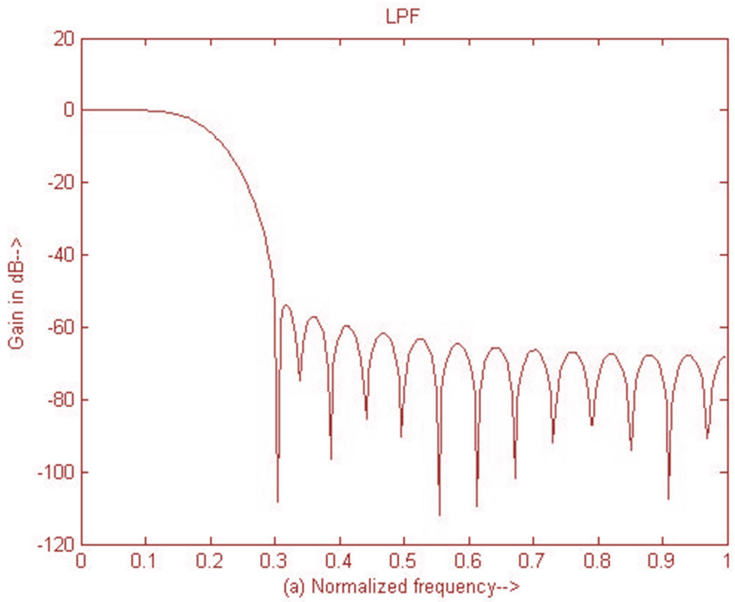

```
enter your choice of window function 1. rectangular 2.
Hamming 3.kaiser:
 2
Hamming window filter response
give type of filter 1:LPF,2:HPF
1:LPF
```

**Low pass FIR filter using Hamming Window**



```
enter your choice of window function 1. rectangular 2.
Hamming 3.kaiser:
 3
kaiser window filter response
give type of filter 1:LPF,2:HPF
1:LPF
```

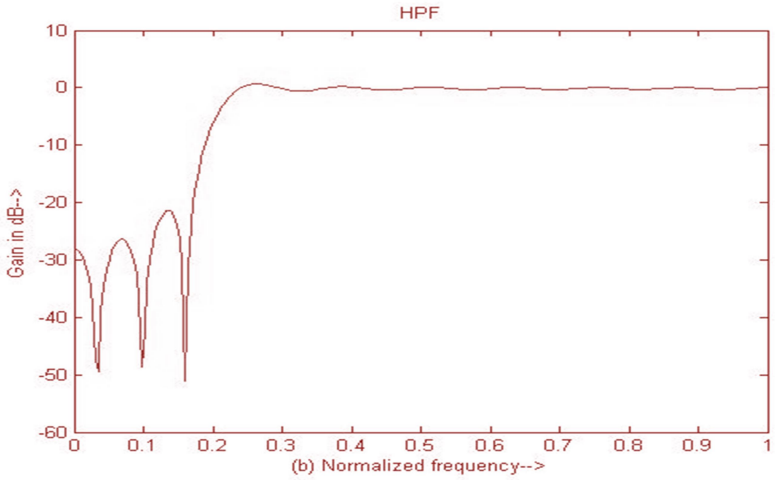
**Low pass FIR filter using Kaiser Window**




**FIR High pass Filter design**

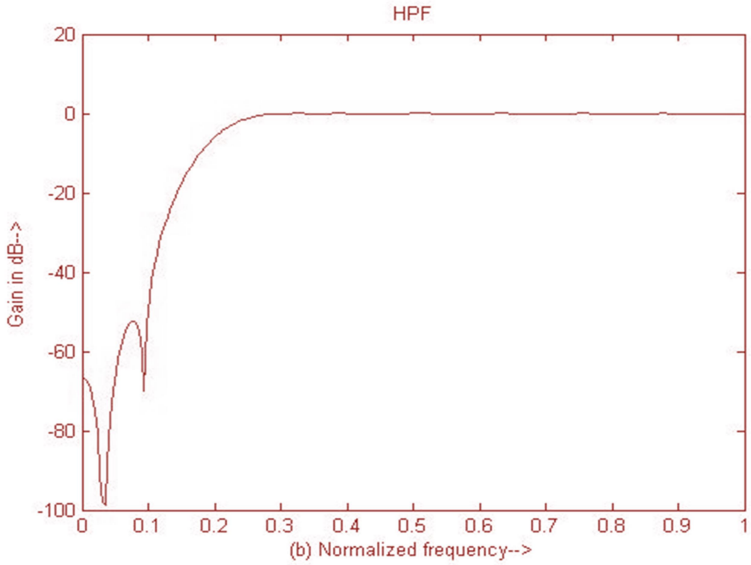
```
Results:
enter passband ripple          0.02
enter the stopband ripple      0.01
enter passband freq            1000
enter stopband freq            1500
enter sampling freq            10000
enter beta value                  5
enter your choice of window function 1. rectangular 2.
Hamming 3.kaiser:
 1
Rectangular window filter response
give type of filter 1:LPF,2:HPF
2:HPF
```

**High pass FIR filter using Rectangular Window**



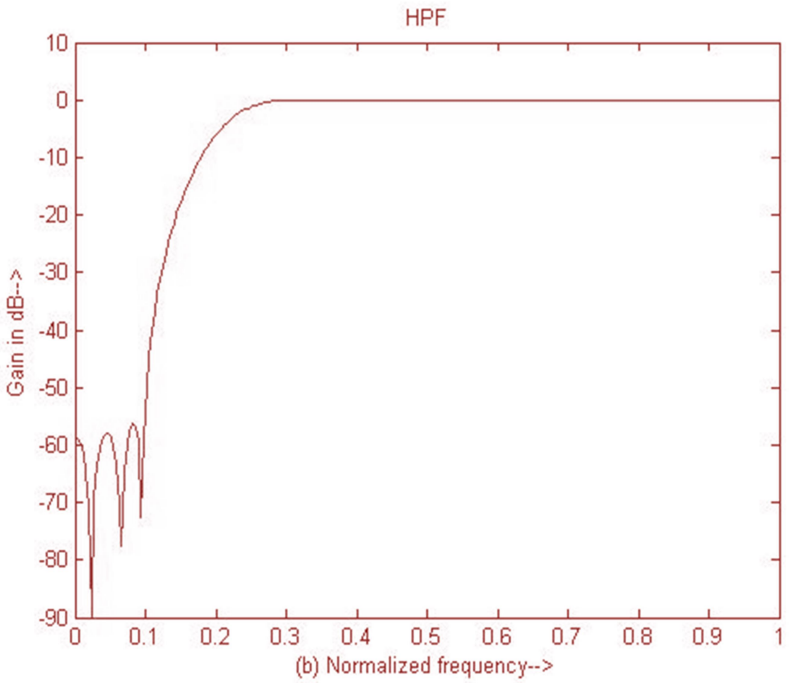

```
enter your choice of window function 1. rectangular 2.
Hamming 3.kaiser:
2
Hamming window filter response
give type of filter 1:LPF,2:HPF
2:HPF
```

**High pass FIR filter using Hamming Window**




```
enter your choice of window function 1. rectangular 2.
Hamming 3.kaiser:
 3
kaiser window filter response
give type of filter 1:LPF,2:HPF
2: HPF
```

**High pass FIR filter using Kaiser Window**




## Discussions on results:

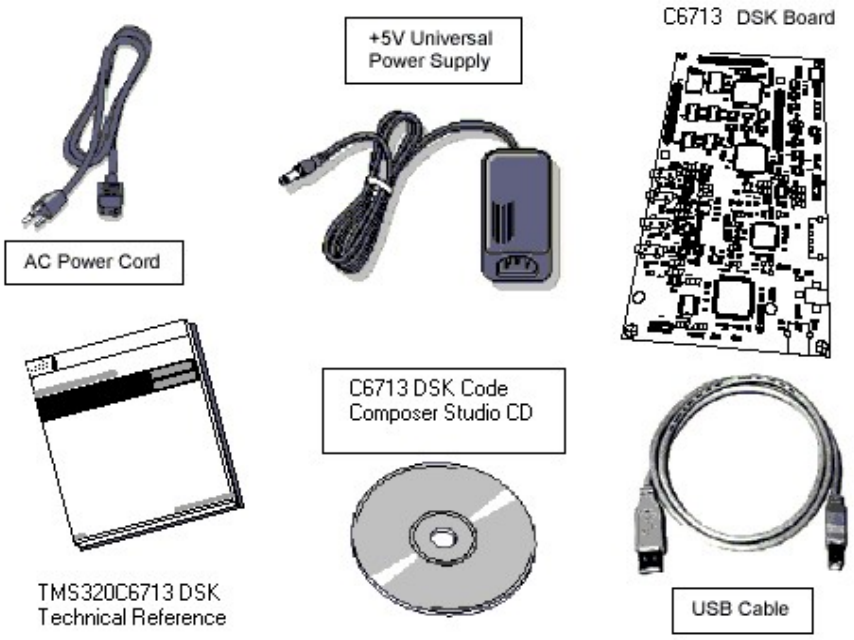
Thus FIR digital filter designing is experimented using Matlab software.

Thus from the results students will be able to

1) Discuss the effect of order of the filer on magnitude response.
2) Discuss the effect of variation in pass band ripple, stop band ripple, pass band frequency, stop band frequency and sampling frequency respectively in designing the FIR digital filter.
3) Discuss the difference between the Rectangular, Hamming and Kaiser Window functions.
4) Discuss the performance of FIR digital filter designed using Kaiser window over FIR digital filter designed with Rectangular and Hamming window functions.

# Cycle-II

# Introduction to TMS320C6713 DSK[ Courtesy: Texas Instrument]




The C6713™ DSK builds on TI's industry-leading line of low cost, easy-to-use DSP Starter Kit (DSK) development boards. The high-performance board features the TMS320C6713 floating-point DSP. Capable of performing 1350 million floating-point operations per second (MFLOPS), the C6713 DSP makes 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. The DSK consists of a C6713-based printed circuit board that will serve as a hardware reference design for TI's customers' products. With extensive host PC and target DSP software support, including bundled TI tools, the DSK provides ease-of-use and capabilities that are attractive to DSP engineers.

The following checklist details items that are shipped with the C6711 DSK kit.

- TMS320C6713 DSK TMS320C6713 DSK development board
- Other hardware External 5VDC power supply

  IEEE 1284 compliant male-to-female cable
- CD-ROM Code Composer Studio DSK tools

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

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

- Embedded JTAG emulation support

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

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

The C6713 DSK includes a stereo codec. This analog interface circuit (AIC) has the following characteristics:

High-Performance Stereo Codec

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

Software Control Via TI McBSP-Compatible Multiprotocol Serial Port

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

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

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

The C6713DSK has the following features:

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

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

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

A programmable logic device called a CPLD is used to implement glue logic that ties the board components together. The CPLD has a register based user interface that lets

the user configure the board by reading and writing to the CPLD registers. The registers reside at the midpoint of CE1.

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

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

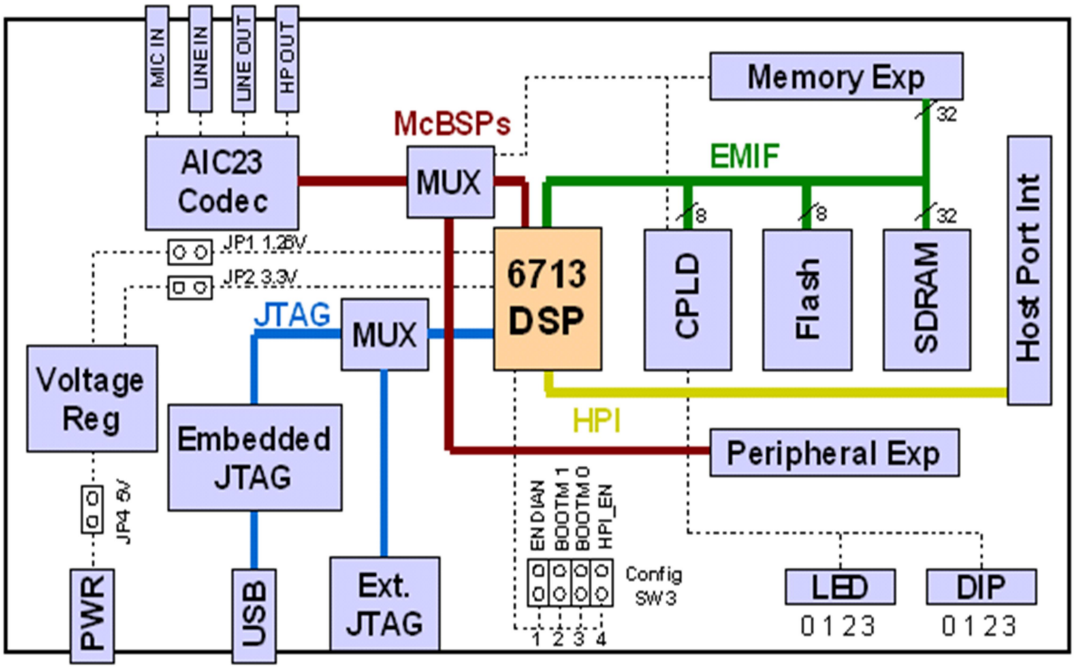
Code Composer communicates with the DSK through an embedded JTAG emulator with a USB host interface. The DSK can also be used with an external emulator through the external JTAG connector.

## TMS320C6713 DSP Features

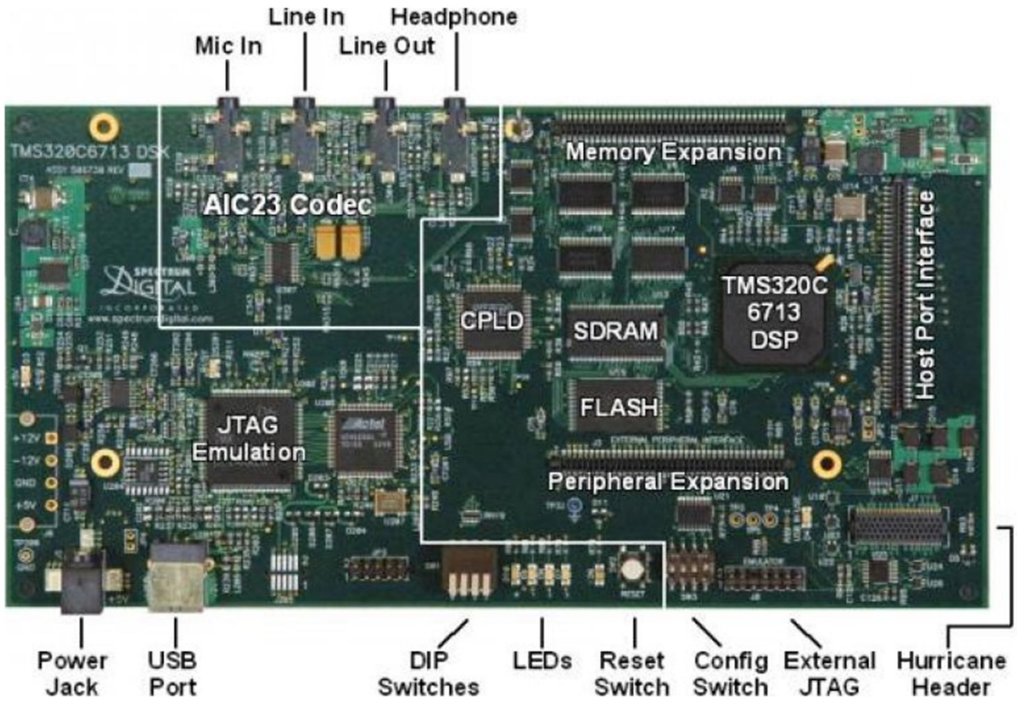
- Highest-Performance Floating-Point Digital Signal Processor (DSP):
  - Eight 32-Bit Instructions/Cycle
  - 32/64-Bit Data Word
  - 300-, 225-, 200-MHz (GDP), and 225-, 200-, 167-MHz (PYP) Clock Rates
  - 3.3-, 4.4-, 5-, 6-Instruction Cycle Times
  - 2400/1800, 1800/1350, 1600/1200, and 1336/1000 MIPS /MFLOPS
  - Rich Peripheral Set, Optimized for Audio
  - Highly Optimized C/C++ Compiler
  - Extended Temperature Devices Available
- Advanced Very Long Instruction Word (VLIW) TMS320C67x™ DSP Core
  - Eight Independent Functional Units:
    - Two ALUs (Fixed-Point)
    - Four ALUs (Floating- and Fixed-Point)
    - Two Multipliers (Floating- and Fixed-Point)
  - Load-Store Architecture With 32 32-Bit General-Purpose Registers
  - Instruction Packing Reduces Code Size
  - All Instructions Conditional
- Instruction Set Features
  - Native Instructions for IEEE 754
    - Single- and Double-Precision
  - Byte-Addressable (8-, 16-, 32-Bit Data)
  - 8-Bit Overflow Protection
  - Saturation; Bit-Field Extract, Set, Clear; Bit-Counting; Normalization
- L1/L2 Memory Architecture
  - 4K-Byte L1P Program Cache (Direct-Mapped)
  - 4K-Byte L1D Data Cache (2-Way)

- 256K-Byte L2 Memory Total: 64K-Byte L2 Unified Cache/Mapped RAM, and 192K-Byte Additional L2 Mapped RAM

- Device Configuration
  - Boot Mode: HPI, 8-, 16-, 32-Bit ROM Boot
  - Endianness: Little Endian, Big Endian
- 32-Bit External Memory Interface (EMIF)
  - Glueless Interface to SRAM, EPROM, Flash, SBSRAM, and SDRAM
  - 512M-Byte Total Addressable External Memory Space
- Enhanced Direct-Memory-Access (EDMA) Controller (16 Independent Channels)
- 16-Bit Host-Port Interface (HPI)
- Two Multichannel Audio Serial Ports (McASPs)
  - Two Independent Clock Zones Each (1 TX and 1 RX)
  - Eight Serial Data Pins Per Port:
    Individually Assignable to any of the Clock Zones
  - Each Clock Zone Includes:
    - Programmable Clock Generator
    - Programmable Frame Sync Generator
    - TDM Streams From 2-32 Time Slots
    - Support for Slot Size:
      8, 12, 16, 20, 24, 28, 32 Bits
    - Data Formatter for Bit Manipulation
  - Wide Variety of I2S and Similar Bit Stream Formats
  - Integrated Digital Audio Interface Transmitter (DIT) Supports:
    - S/PDIF, IEC60958-1, AES-3, CP-430 Formats
    - Up to 16 transmit pins
    - Enhanced Channel Status/User Data
  - Extensive Error Checking and Recovery
- Two Inter-Integrated Circuit Bus (I2C Bus™) Multi-Master and Slave Interfaces
- Two Multichannel Buffered Serial Ports:
  - Serial-Peripheral-Interface (SPI)
  - High-Speed TDM Interface
  - AC97 Interface
- Two 32-Bit General-Purpose Timers
- Dedicated GPIO Module With 16 pins (External Interrupt Capable)
- Flexible Phase-Locked-Loop (PLL) Based Clock Generator Module
- IEEE-1149.1 (JTAG †) Boundary-Scan-Compatible
- Package Options:
  - 208-Pin PowerPAD™ Plastic (Low-Profile) Quad Flatpack (PYP)
  - 272-BGA Packages (GDP and ZDP)
- 0.13-µm/6-Level Copper Metal Process
  - CMOS Technology
- 3.3-V I/Os, 1.2 ‡ -V Internal (GDP & PYP)
- 3.3-V I/Os, 1.4-V Internal (GDP)(300 MHz only)




TMS320C6713 DSK Overview Block Diagram [Courtesy: Texas Instrument]

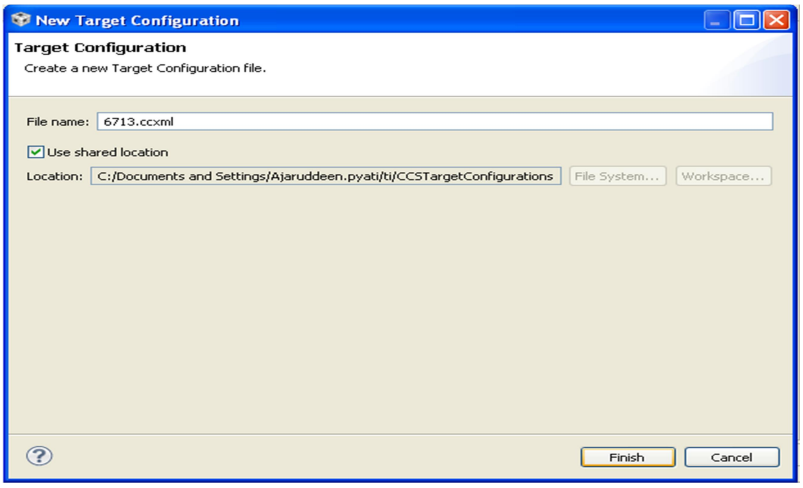



Procedure to work with CCS-V5 using TMS320C6713

Procedure to work with non real time program (Linear Convolution)

Step1: Creation of a Target configuration

File→new→target configuration , you will get window as shown in fig1. You can give any target name with extension .ccxml (for example here I have taken target name as 6713.ccxml) after giving the name click on finish.

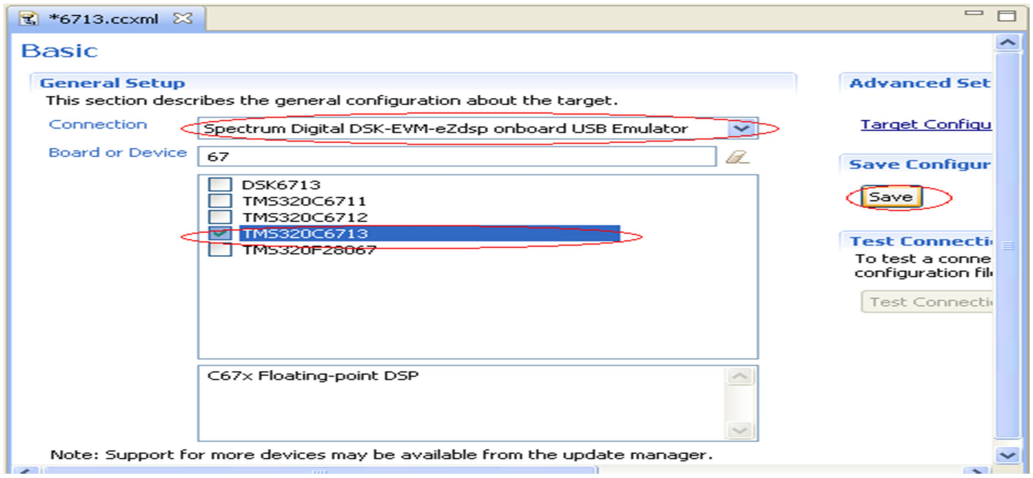



Once you click on finish you will get general setup window as shown below in fig2. In the general setup you have select device and connection as mentioned below.

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

Board or Device: TMS320C6713

Then click on SAVE

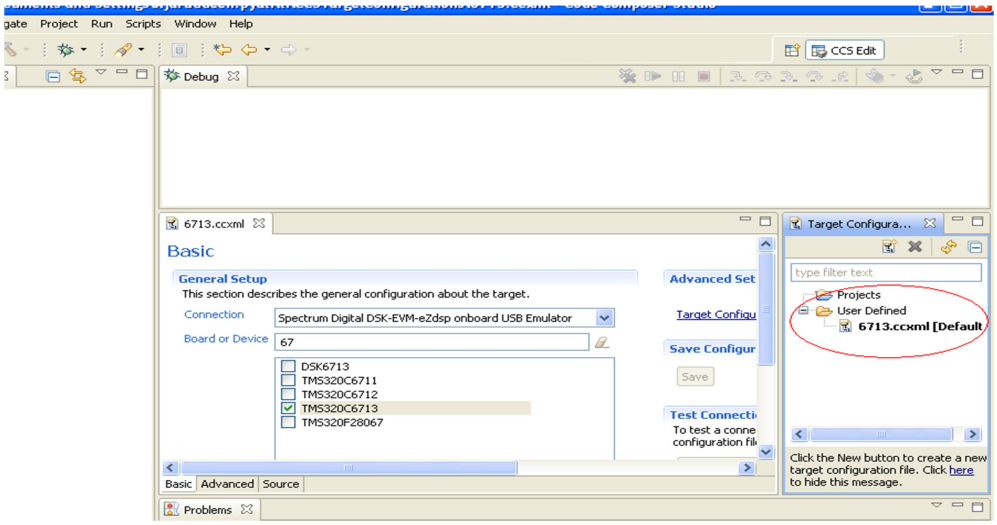


Step 2: Launch the Target Configuration

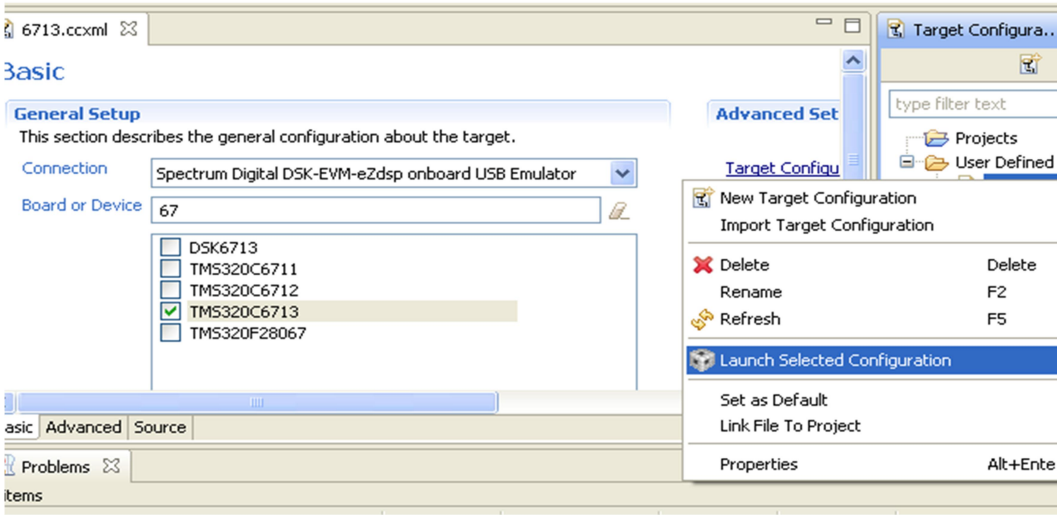
Go to view from toolbar

View→ target configuration , you can see your target name under user defined as circled in fig 3 .

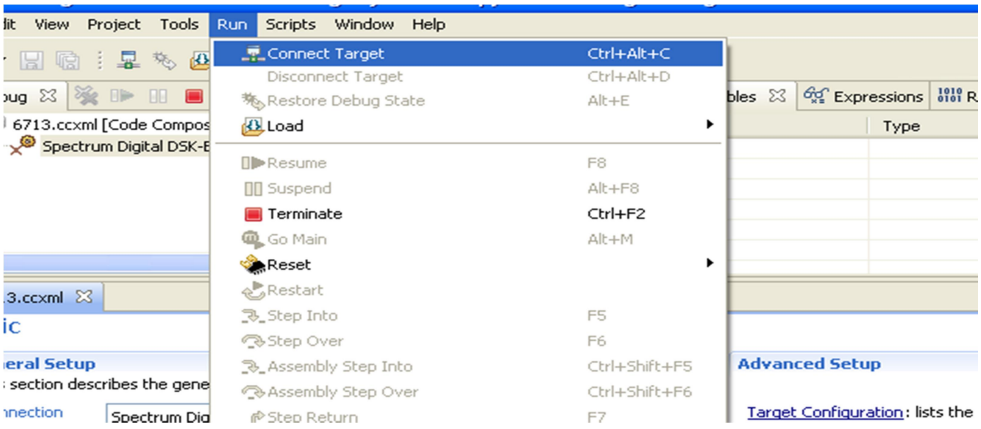
Here under user defined it is showing 6713.ccxml it is my target.



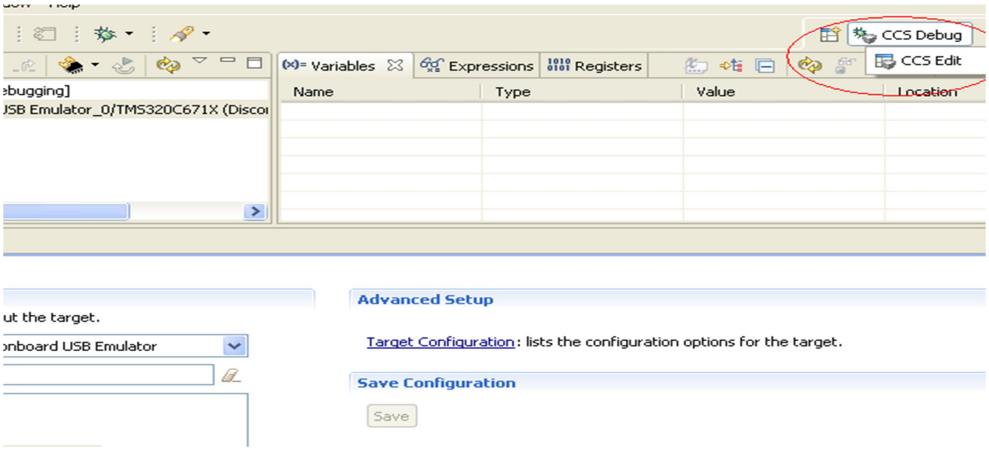

Right click on your target(6713.ccxml), then launch selected configuration as shown in fig4 below.



Step 3: Connecting the target
go to Run→connect target as shown in below.

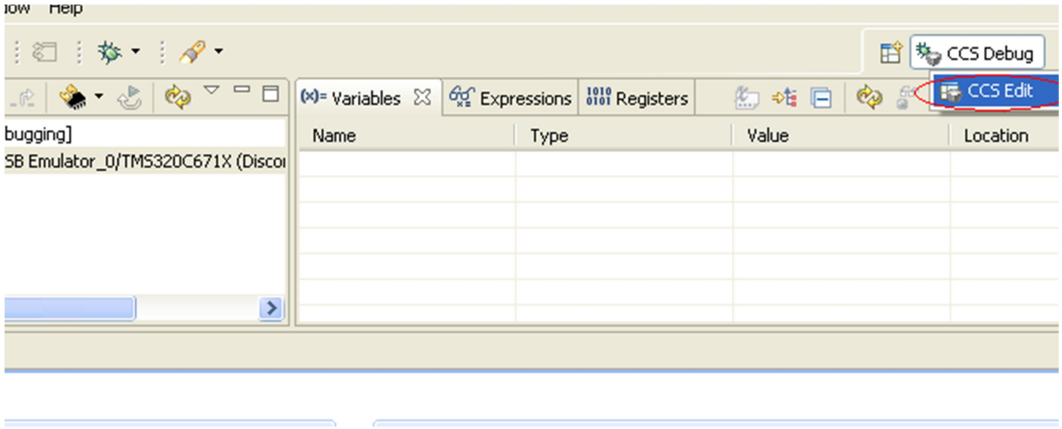



Note: on mail CCS window there are two prospective are there one is CCS Debug and one more is CCS edit . as shown in below diagram fig 6 . in CCS debug it contain all option related to target and in CCS edit it contain all options related to projects( source, lib etc)

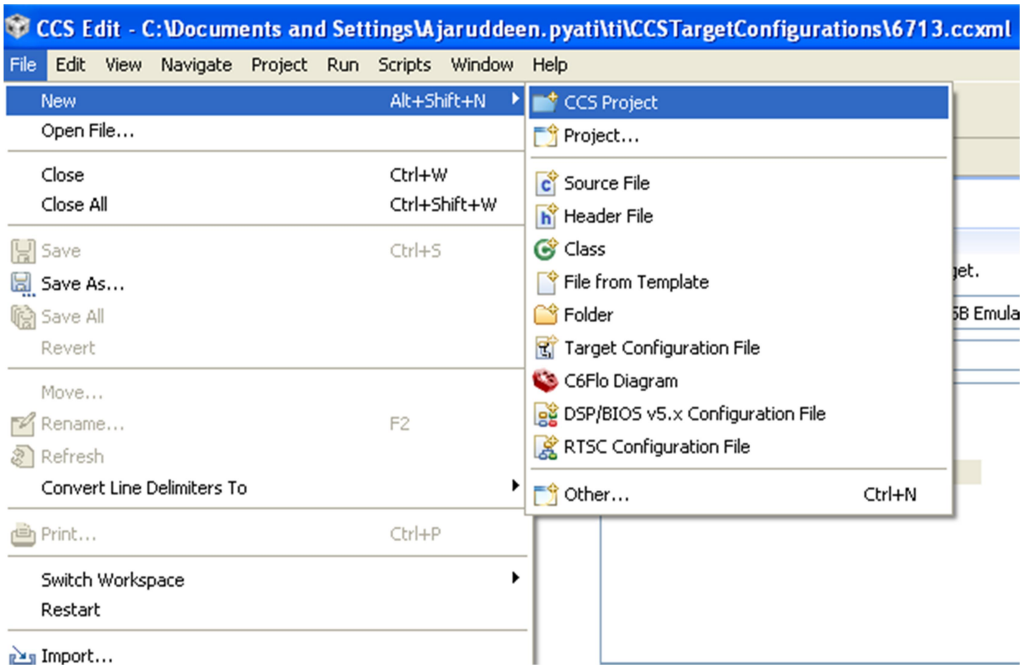


Step 4: Creation of a project

First click on CCS edit as shown below



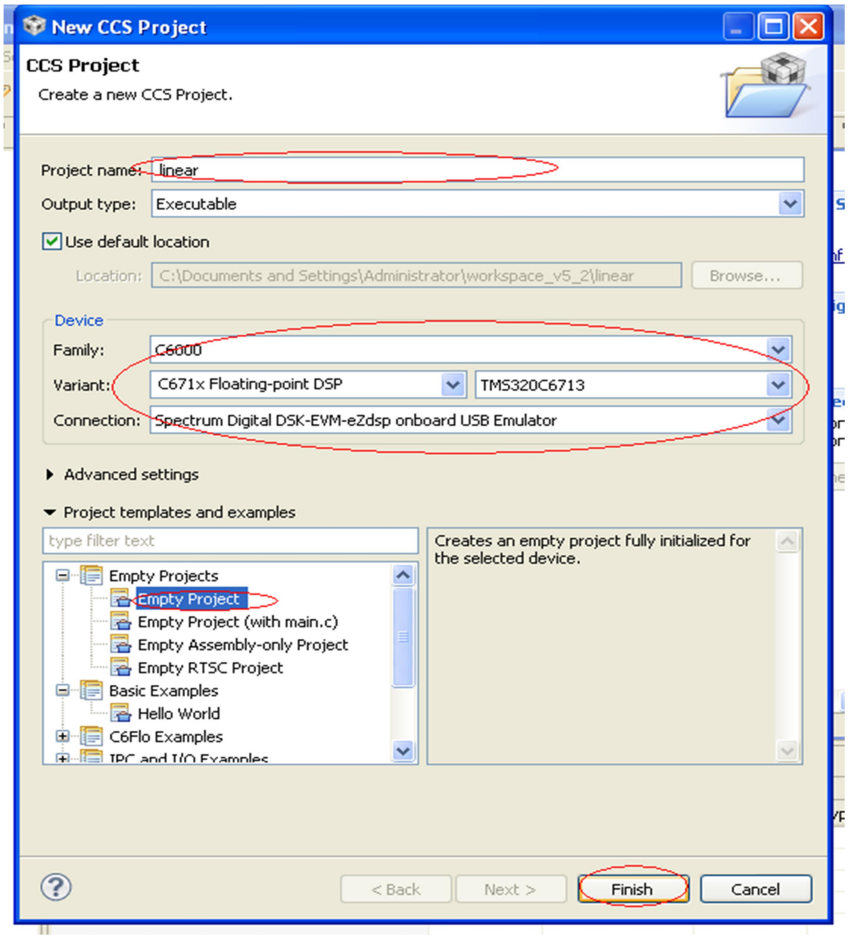

Then go to File→new→CCS Project as shown in below figure.




Then make as changes as mentioned.

Project name: any name (for example I have given as linear)
Family : C6000
Variant : C671xFloating-point DSP TMS320C6713
Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator

Then select empty project, Then click on finish .


New CCS Project
CCS Project
Create a new CCS Project.
Project name: linear
Output type: Executable
Use default location
Location: C:\Documents and Settings\Administrator\workspace_v5_2\linear
Browse...
Device
Family: C6000
Variant: C671x Floating-point DSP
TMS320C6713
Connection: Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
Advanced settings
Project templates and examples
type filter text
Empty Projects
Empty Project
Empty Project (with main.c)
Empty Assembly-only Project
Empty RTSC Project
Basic Examples
Hello World
C6Flo Examples
Creates an empty project fully initialized for the selected device.
< Back
Next >
Finish
Cancel

Step 5: Create a source file

File→new→ source file

Source file: give any name with extension .c(here I have given as lin.c) as sown in Then click on finish.

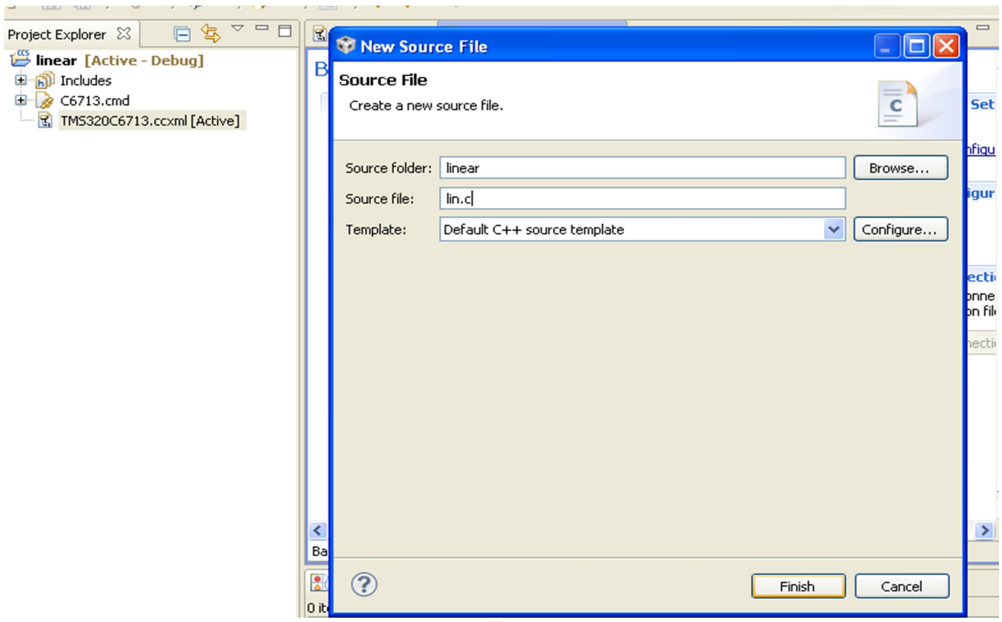
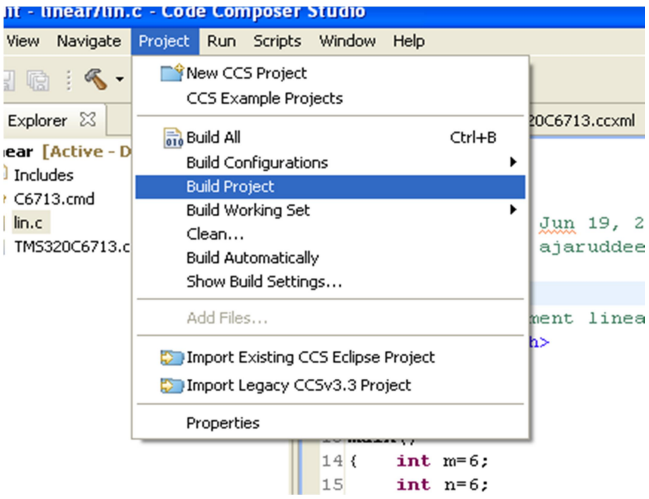



Fig 10

Now write a code on workspace, then go to file→save.

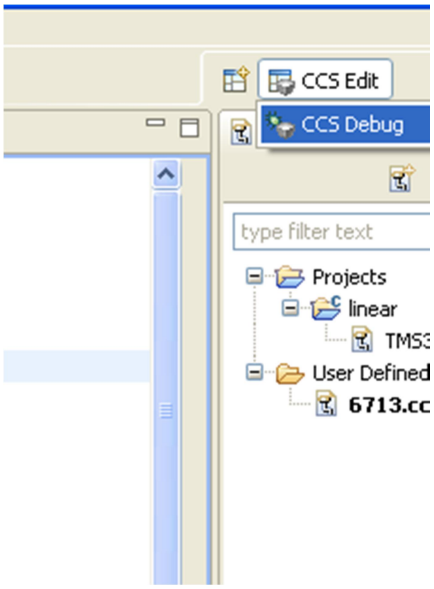
Step 6: Build the project

Go to Project→Build Project as shown in Fig 11below.

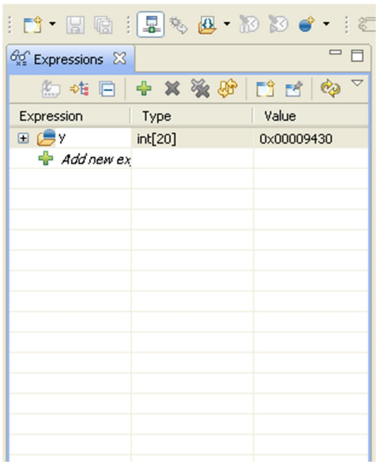


**Step 7: run the project**

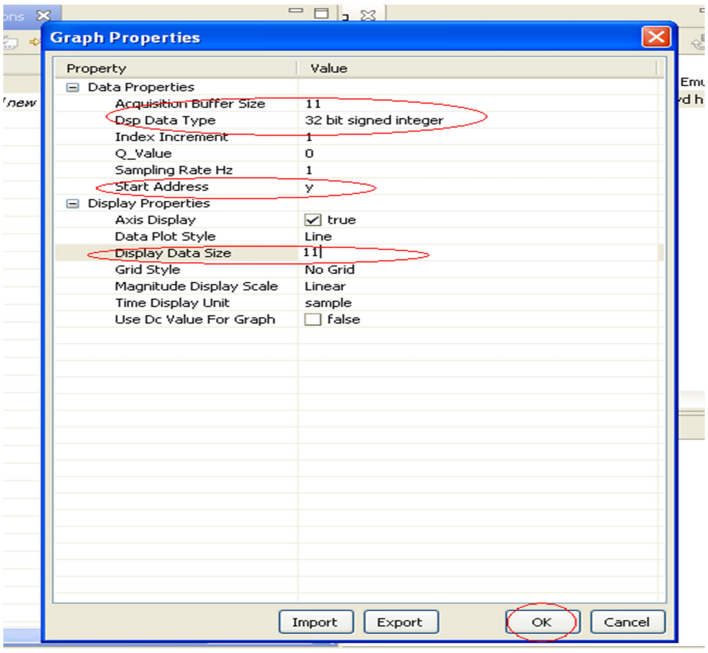
Switch to CCS debug prospective as shown in below figure.



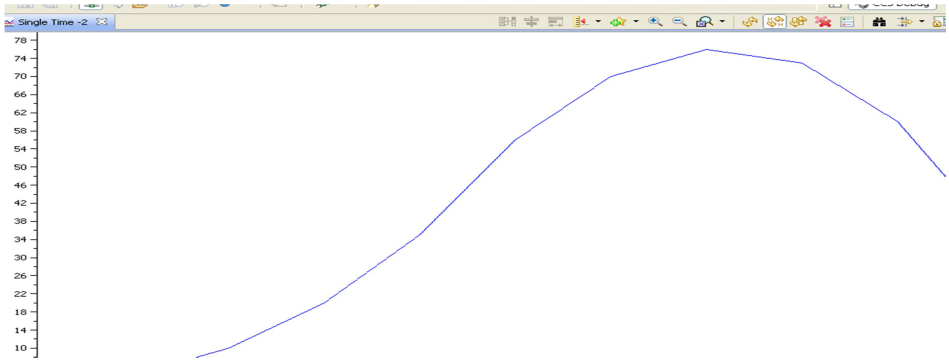

Then go to Run→Resume to run the program.
You can see output of linear convoluted samples as in watch window by typing in expression as mention below.
View→expression
Then type output variable name y then enter as shown below.



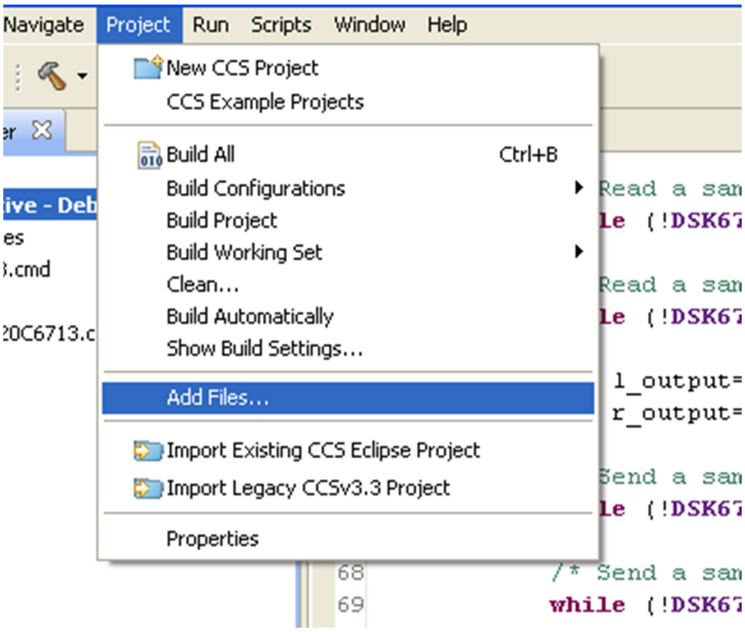

Step 8: Plotting a graph
Go to Tools→Graph→Single Time , then make graphical properties change as per program here in fig14 I had made changes which round up with red line . this changes applicable for linear convolution program. Then click ok.




In following figure my expected graph for linear con is displayed.




Procedure to work with real time program(FIR)
Note: Here again no need to create a new target configuration because we can use the same so now directly I will go to creating a project
Step 1: Create a Project
Please refer the step 4 of non real time program, give the project name as FIR.
Step 2: Create a source file
Please refer the step 5 of non real time program , give the source file name as FIR.C.
Step 3: Adding library file
Go to project→add files as shown in below figure.



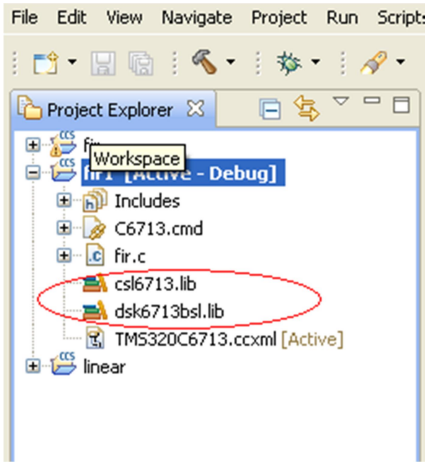

Then you have to add bsl and csl library files. These files you will get in below mentioned paths.

- BSL(Board support library file)

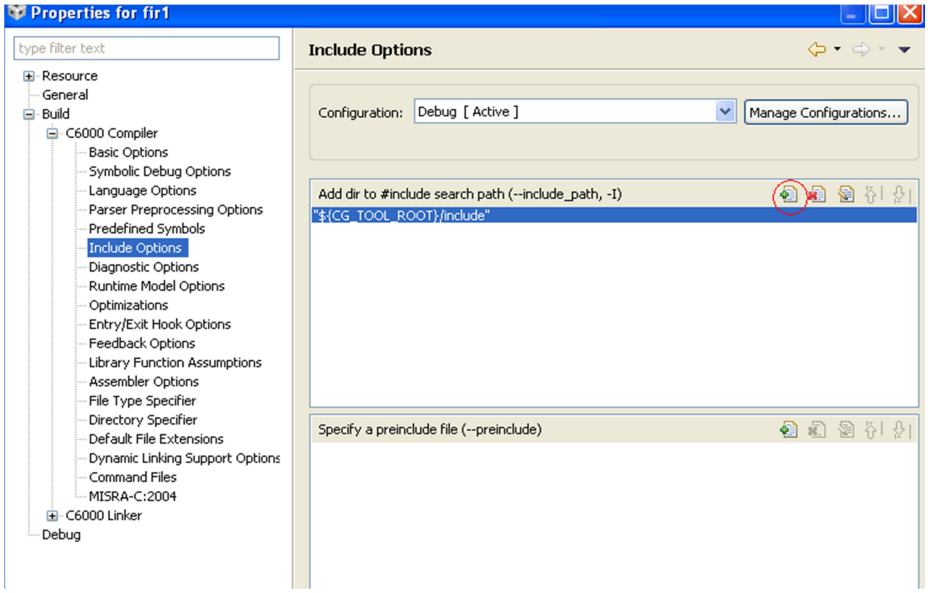
Path : C:\CCStudio_v3.1\C6000\dsk6713\lib\dsk6713bsl.lib

- CSL(Chip support library file )

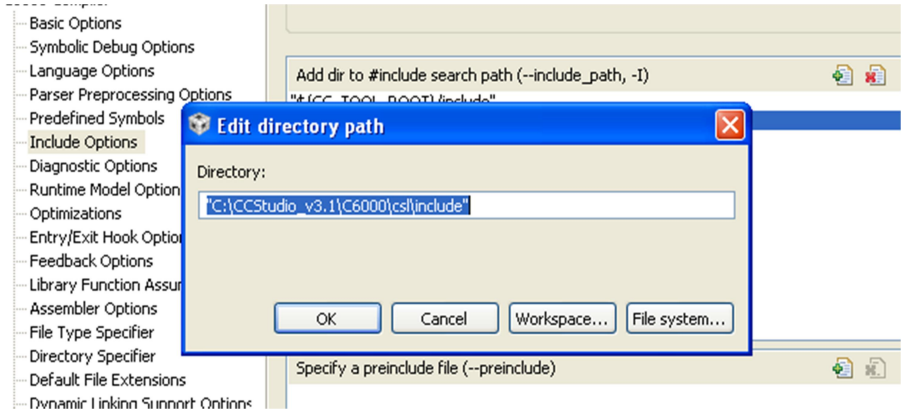
Path: C:\CCStudio_v3.1\C6000\csl\lib\csl6713.lib
You can see these files added to your project as shown below in figure.



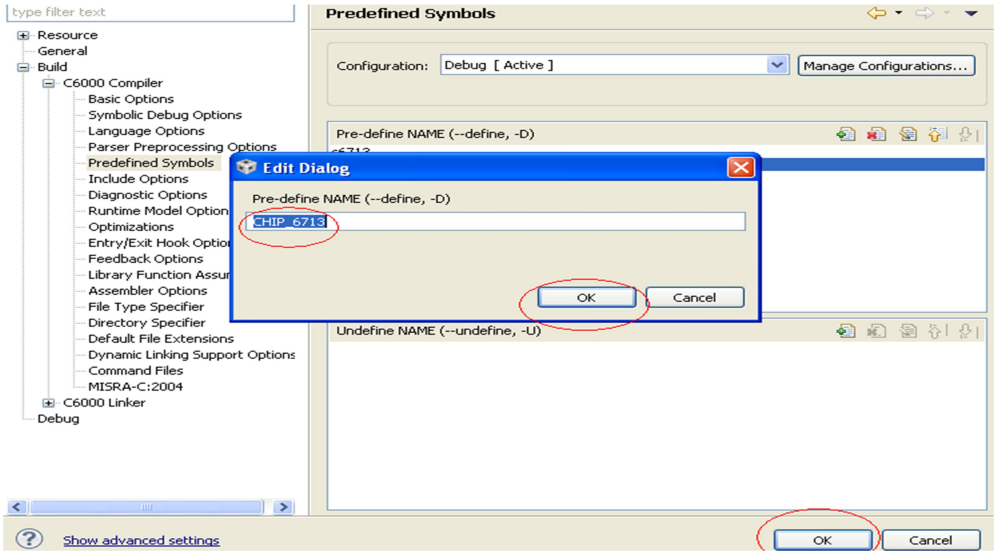
Step 4: Setting Build properties
Go to project→properties , under Build→ C6000 Compiler→include options now click on add as shown in below figure




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




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

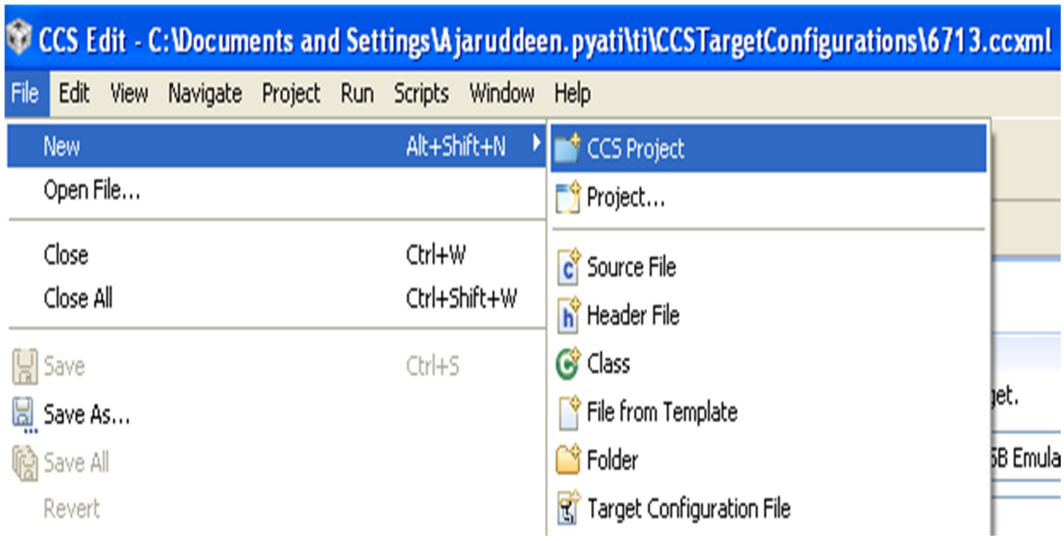



Step 5: Build the project
Go to Project→Build Project.
Step 6: run the project
Then go to Run→Resume to run the program.

# I. Procedure to work with NON REAL TIME programs (Linear Convolution):

**NR1: Creation of a project**

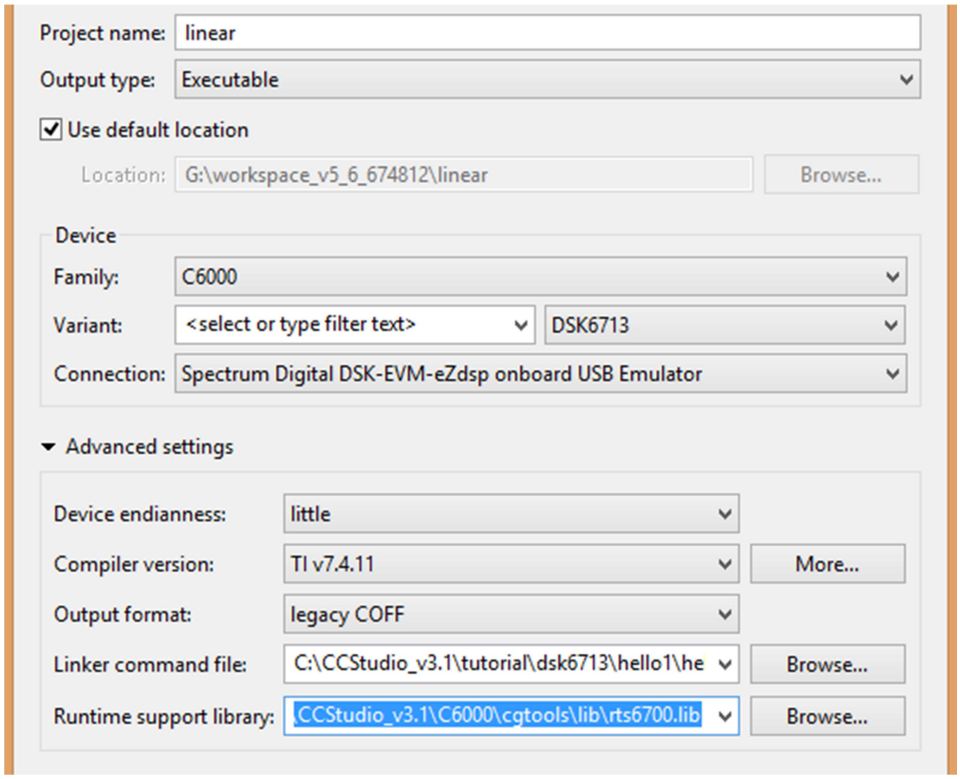
Go to **File→new→CCS Project** as shown in figure below

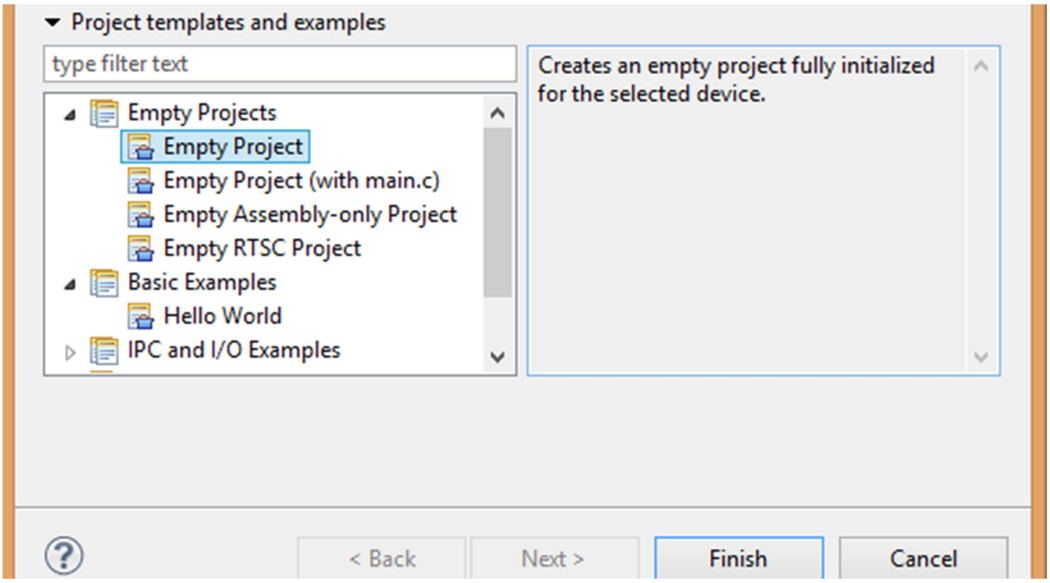



Then make changes as mentioned below(or refer the figure in next page):

**Project name:** any name (for example I have given as linear)
**Family** : C6000
**Variant** : <LEAVE IT BLANK> | DSK6713
**Connection** : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator (if using the kit)
Advanced settings:
**Linker Command file:** C:\CCStudio_v3.1\tutorial\dsk6713\hello1\hello.cmd
**Runtime Support Library**: C:\CCStudio_v3.1\C6000\cgtools\lib\rts6700.lib

Then select **empty project,** Then click on **finish .**

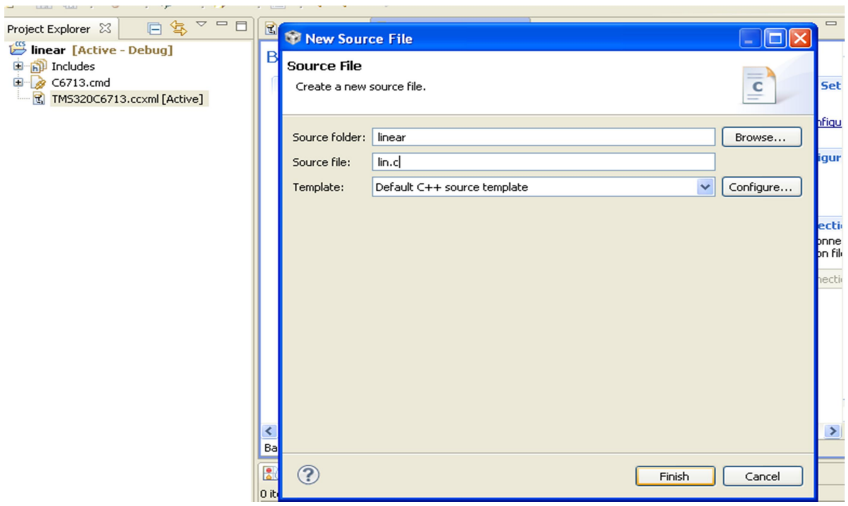

Project name: linear
Output type: Executable
Use default location
Location: G:\workspace_v5_6_674812\linear
Browse...
Device
Family: C6000
Variant: <select or type filter text>
DSK6713
Connection: Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
Advanced settings
Device endianness: little
Compiler version: TI v7.4.11
More...
Output format: legacy COFF
Linker command file: C:\CCStudio_v3.1\tutorial\dsk6713\hello1\he
Browse...
Runtime support library: CCStudio_v3.1\C6000\cgtools\lib\rts6700.lib
Browse...


Project templates and examples
type filter text
Empty Projects
Empty Project
Empty Project (with main.c)
Empty Assembly-only Project
Empty RTSC Project
Basic Examples
Hello World
IPC and I/O Examples
Creates an empty project fully initialized for the selected device.
< Back
Next >
Finish
Cancel

**NR2: Creation of source file (Use either a or b):**

**a. Create a source file**

**File->new-> source file**




**Source file:** give any name with extension .c (here I have given **as lin.c**) Then click on finish.
Now write a code (or copy the lin.c) on workspace, then go to **file->save.**

```
//lin.c
#include<stdio.h>
#define LENGHT1 6 /*Lenght of i/p samples sequence*/
#define LENGHT2 4 /*Lenght of impulse response Co-efficients
*/
int x[2*LENGHT1-1]={1,2,3,4,5,6,0,0,0,0,0,0}; /*Input Signal
Samples*/
int h[2*LENGHT1-1]={1,2,3,4,0,0,0,0,0,0,0}; /*Impulse
Response Coefficients*/
int y[LENGHT1+LENGHT2-1];
main()
{
int i=0,j;
for(i=0;i<(LENGHT1+LENGHT2-1);i++)
{
y[i]=0;
for(j=0;j<=i;j++)
y[i]+=x[j]*h[i-j];
}
for(i=0;i<(LENGHT1+LENGHT2-1);i++)
```

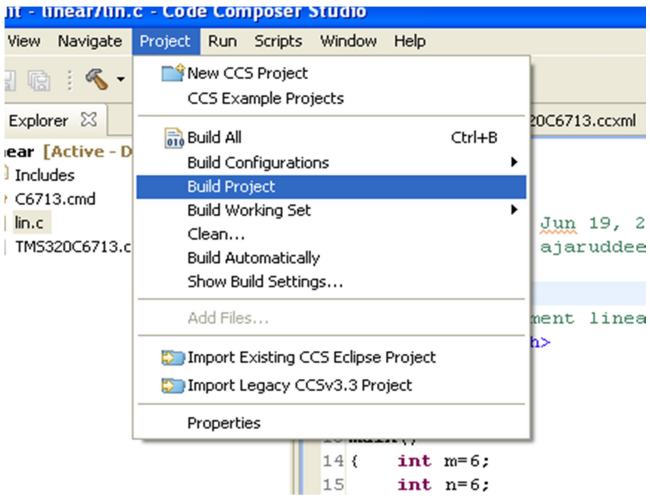
**b. Add existing C file:**

Right click on project name and choose **Add files..**

Now browse the lin.c from the system and click OK.

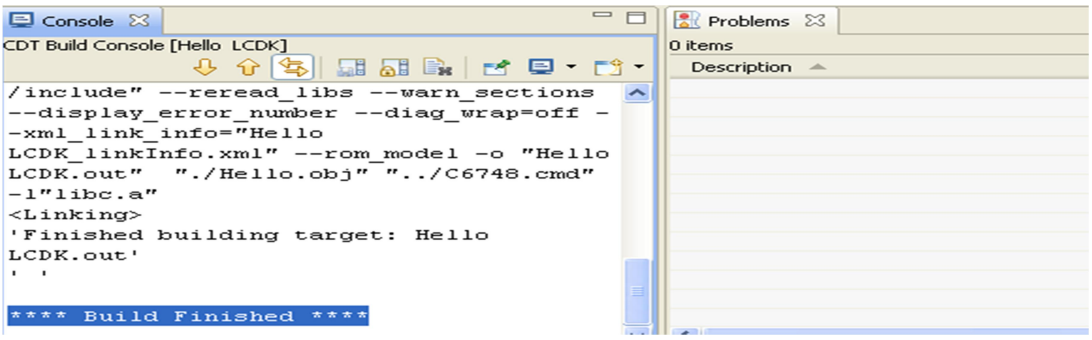
**NR3: Build the project**

Go to **Project→Build Project** as shown below.



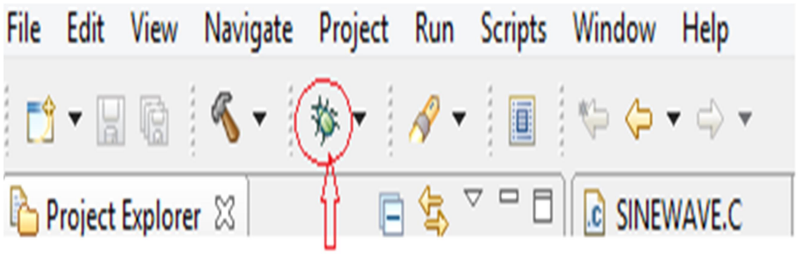

If your code doesn't have any errors and warnings, a message will be printed in the console window that "`**** Build Finished ****`"

**Problems** window display errors or warnings, if any.



**NR4. DEBUG**

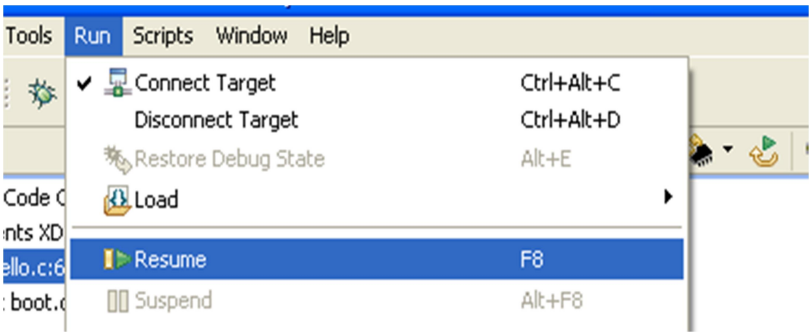
After successful Build, connect the kit with the system using the JTAG emulator and power the kit.
Click the **Debug** as shown in the below figure.




It will redirect to the Debug perspective automatically.

**NR5: Running the project**

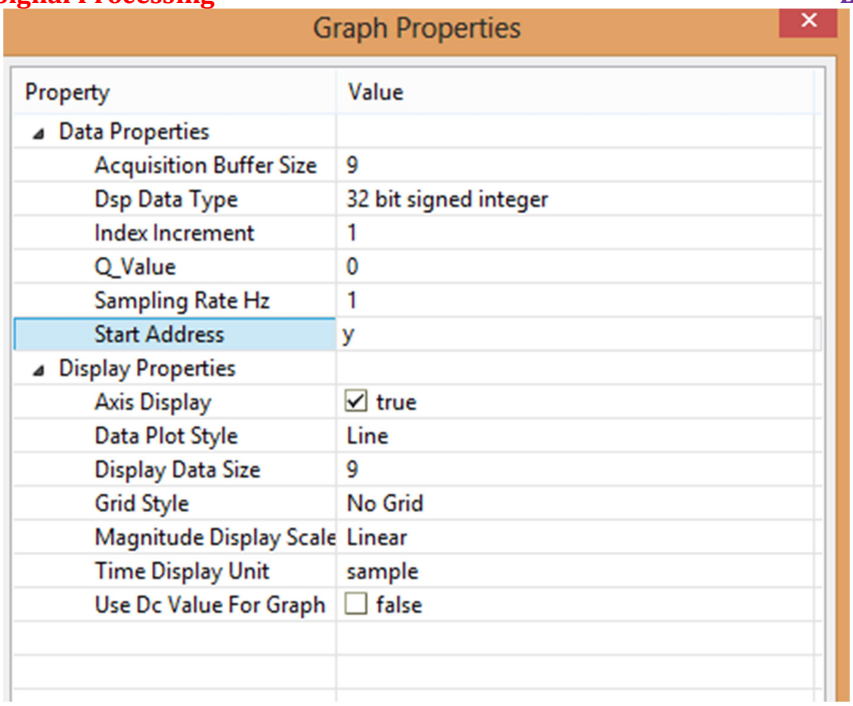
Wait until the program loaded to the hardware automatically.
Now you can run the code, by selecting Run-> **Resume**.



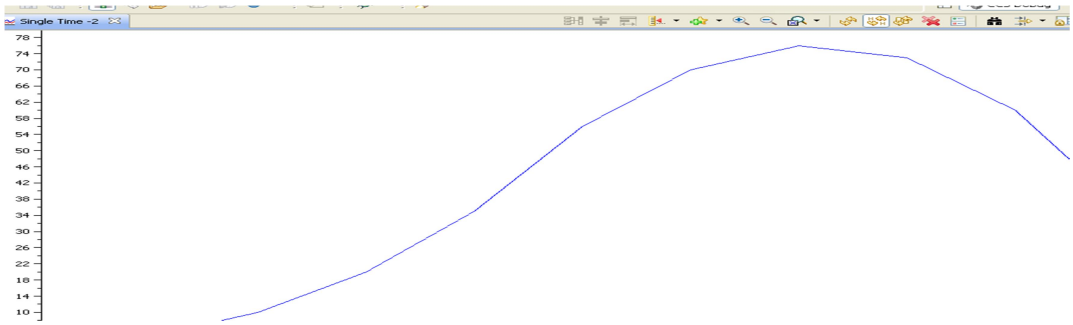

The linear convolution values will be displayed in the **Console window**.

**NR6: Plotting a graph**

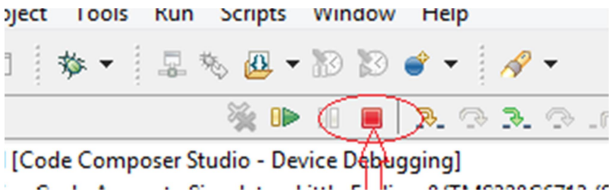
Go to **Tools→Graph→Single Time** , then make graphical properties change as per program (Graph property will vary according to your program).
These changes are applicable for linear convolution program. Then click **ok**




Expected graph for linear convolution:



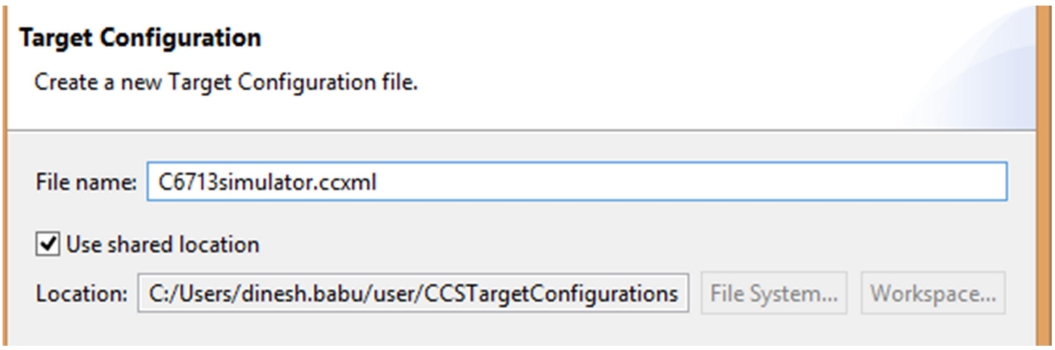
## Terminate the project after use:



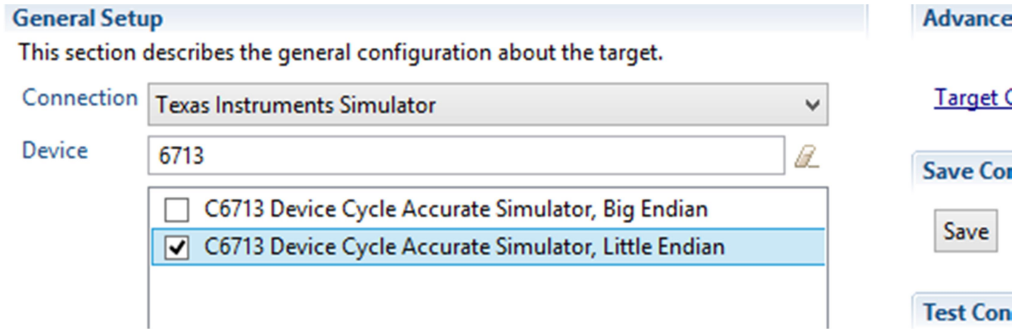

Note: In the same way all the Non Real-Time projects can be done by just replacing the corresponding source files.

## II. Procedure for using simulation mode (without the kit and assuming that you have already created a non-real time project):

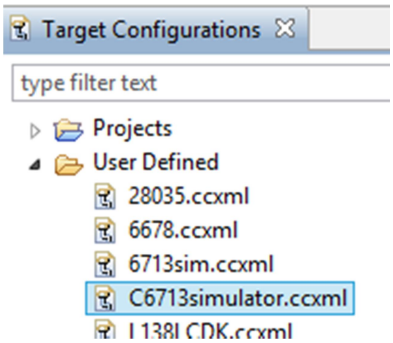
- Goto File-> New -> Target Configuration File
-



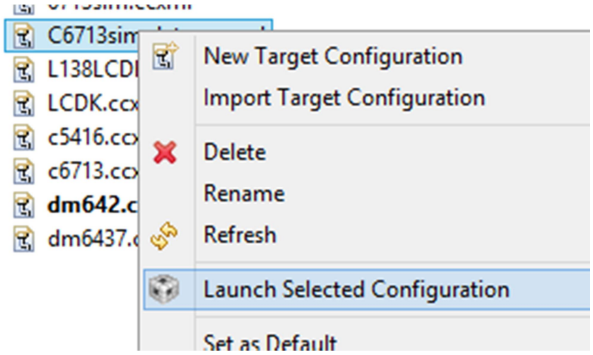

- Give a name with '.ccxml' extension.
- Connection: Texas Instruments Simulator
- Device: <just type 6713>
- Choose '…..little endian' (see the below figure)




- Click SAVE.
- View-> Target Configurations
- Target Configurations tab will be opened:




- Right on your file and click Launch Configuration:

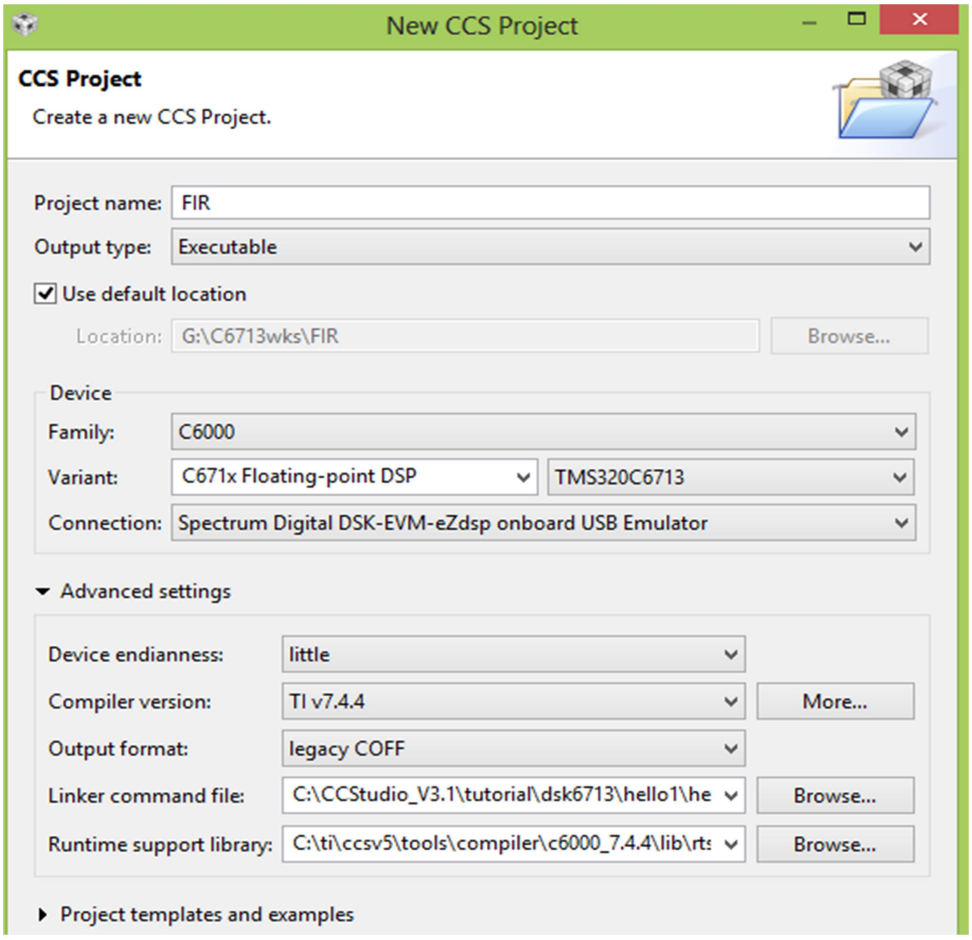



- Goto Run-> Load -> Load program-> Browse project
- Choose the '.out' file of your project from debug folder and **OK**.
- Now you can RUN and plot graphs in the same way as in previous procedure.

## III. Procedure to work with REAL TIME program (FIR filter):

**R1: Create a Project:**
**Project name:** any name (for example I have given as FIR)
**Family : C6000**
**Variant : <LEAVE IT BLANK> | DSK6713**
**Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator (if using the kit)**
Advanced settings:
**Linker Command file:** C:\CCStudio_v3.1\tutorial\dsk6713\hello1\hello.cmd
**Runtime Support Library**: C:\CCStudio_v3.1\C6000\cgtools\lib\rts6700.lib

Then select **empty project,** Then click on **finish .**



**R2: Create a source file**

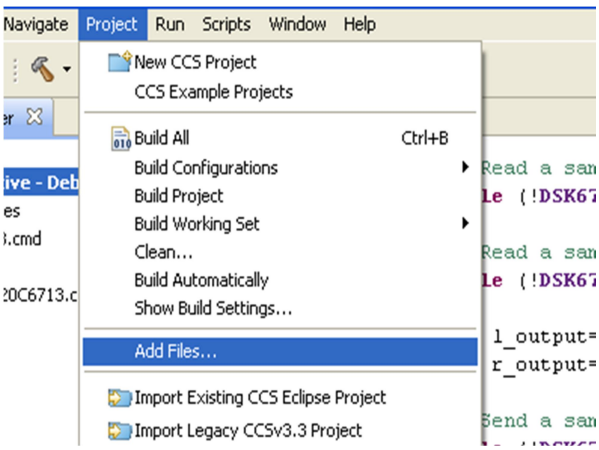
Please refer the **step 2** of non real time program, give the source file name as FIR.C.

```
//FIR.c
#include "C:\CCStudio_v3.1\C6000\dsk6713\include\dsk6713.h"
#include "C:\CCStudio_v3.1\C6000\dsk6713\include\dsk6713_aic23.h"

float filter_Coeff[] = {

  0.000000,-0.001591,-0.002423,0.000000,0.005728,
0.011139,0.010502,-0.000000,-0.018003,-0.033416,-0.031505,0.000000,
0.063010,0.144802,0.220534,0.262448,0.220534,0.144802,0.063010,0.000000,
-0.031505,-0.033416,-0.018003,-0.000000,0.010502,0.011139,0.005728,
0.000000,-0.002423,-0.001591,0.000000
};
static short in_buffer[100];
DSK6713_AIC23_Config config = { \
   0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */ \
   0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume */\
   0x00d8,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */  \
   0x00d8,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */ \
   0x0011,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */      \
   0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */     \
   0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */             \
   0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */ \
   0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */            \
   0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */   \
};
/*
 *  main() - Main code routine, initializes BSL and generates tone
 */

void main()
{
   DSK6713_AIC23_CodecHandle hCodec;

   Uint32 l_input, r_input,l_output, r_output;

   /* Initialize the board support library, must be called first */
   DSK6713_init();

   /* Start the codec */
   hCodec = DSK6713_AIC23_openCodec(0, &config);

   DSK6713_AIC23_setFreq(hCodec, 1);

     while(1)
      {    /* Read a sample to the left channel */
                              while (!DSK6713_AIC23_read(hCodec, &l_input));

                              /* Read a sample to the right channel */
                              while (!DSK6713_AIC23_read(hCodec, &r_input));

                                      l_output=(Int16)FIR_FILTER(&filter_Coeff ,l_input);
                                 r_output=(Int16)FIR_FILTER(&filter_Coeff ,r_input);

                              /* Send a sample to the left channel */
        while (!DSK6713_AIC23_write(hCodec, l_output));

        /* Send a sample to the right channel */
        while (!DSK6713_AIC23_write(hCodec, r_output));
      }
```

```
    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}
signed int FIR_FILTER(float * h, signed int x)
{
int i=0;
signed long output=0;

in_buffer[0] = x;  /* new input at buffer[0]  */

for(i=31;i>0;i--)
in_buffer[i] = in_buffer[i-1]; /* shuffle the buffer   */

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

return(output);

}
```

**R3: <u>Adding library files</u>**

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




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

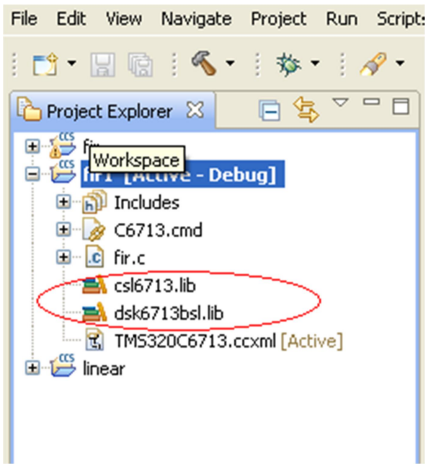
- BSL(Board support library file)

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

- CSL(Chip support library file )

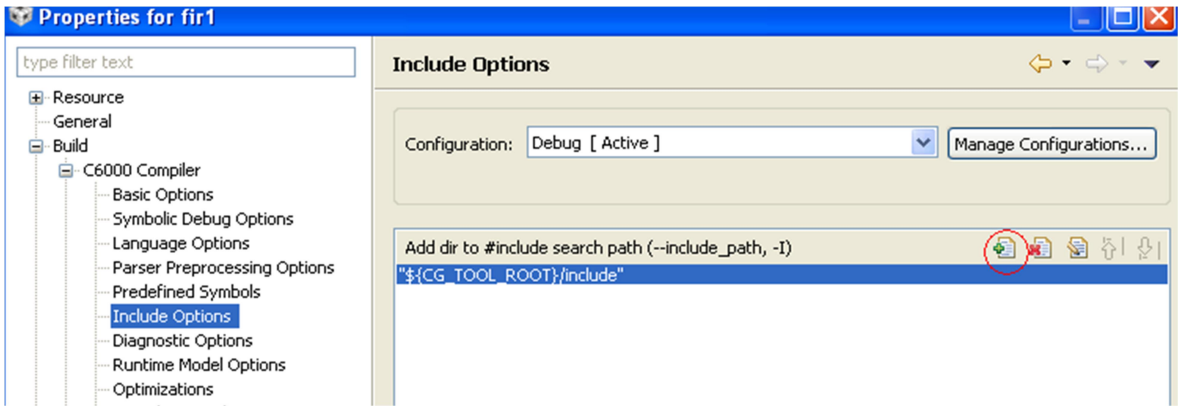
Path: ***C:\CCStudio_v3.1\C6000\csl\lib\csl6713.lib***

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

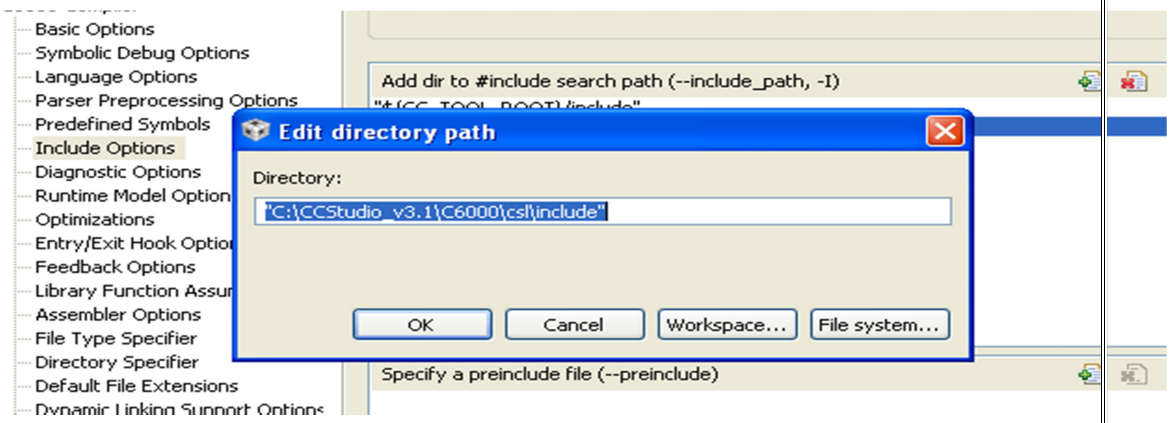



**R4: Setting Build properties**

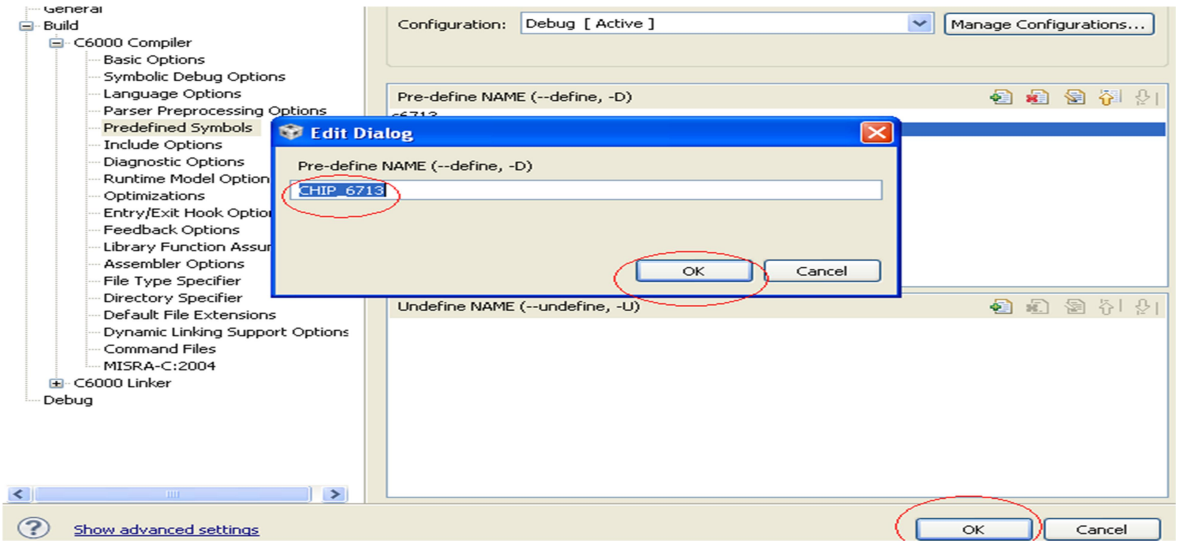
- Go to **project→properties** , under **Build→ C6000 Compiler→include options** now click on **add** as shown in below:




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



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




**R5:** Now Build, Debug and RUN the project as mentioned in NR3, NR4 and NR5.
Connect CRO through stereo cable to the LINE OUT.
Connect a Signal Generator through stereo cable to the LINE IN Socket.
The signal will be attenuated beyond 1 KHz for this program.

In the same way all the Real-Time projects can be done just by replacing the corresponding source files.

# Experiment -7

**Aim:** To verify Linear Convolution using TMS320C6713 DSK

**Equipments Required:** PC, TMS320C6713 DSK, Code composer Studiov5.5

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

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

x[ ]= Input signal Samples
h[ ]= Impulse response co-efficient.
y[ ]= Convolution output.
n = No. of Input samples
h = No. of Impulse response co-efficient.

**Algorithm:**

Algorithm to implement 'C' program for Convolution:

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

**Where: n=4, k=4.** **;Values of n & k should be a multiple of 4.**
**If n & k are not multiples of 4, pad with zero's to make multiples of 4**
**r= n+k-1** **; Size of output sequence.**
**= 4+4-1**
**= 7.**

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

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

**Procedure:**

1) Step1: Creation of a Target configuration
   a. File→new→target configuration , you will get window as shown in fig1. You can give any target name with extension .ccxml (for example here I have taken target name as 6713.ccxml) after giving the name click on finish.

   Once you click on finish you will get general setup window as shown below in fig2. In the general setup you have select device and connection as mentioned below.
   Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
   Board or Device: TMS320C6713

Then click on SAVE

2) Step 2: Launch the Target Configuration
   Go to view from toolbar
   View→ target configuration, you can see your target name under user defined as circled in fig 3 .
   Here under user defined it is showing 6713.ccxml it is my target.
3) Step 3: Connecting the target
   go to Run→connect target as shown in below.
   Note: on mail CCS window there are two prospective are there one is CCS Debug and one more is CCS edit . as shown in below diagram fig 6 . in CCS debug it contain all option related to target and in CCS edit it contain all options related to projects( source, lib etc).
4) Step 4: Creation of a project
   First click on CCS edit as shown below
   Then go to File→new→CCS Project as shown in below figure.
   Then make as changes as mentioned.
   Project name: any name (for example I have given as linear)
   Family : C6000
   Variant : C671xFloating-point DSP TMS320C6713
   Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
   Then select empty project, Then click on finish.
5) Step 5: Create a source file
   File→new→ source file
   Source file: give any name with extension .c(here I have given as lin.c) as sown in
   Then click on finish.
   Now write a code on workspace, then go to file→save.
6) Step 6: Build the project
   Go to Project→Build Project as shown in Fig 11below.
7) Step 7: Run the project
   Switch to CCS debug prospective as shown in below figure.
   Then go to Run→resume to run the program.
   You can see output of linear convoluted samples as in watch window by typing in expression as mention below.
   View→expression
   Then type output variable name y then enter as shown below.
8) Step 8: Plotting a graph
   Go to Tools→Graph→Single Time , then make graphical properties change as per program here in fig14 I had made changes which round up with red line . this changes applicable for linear convolution program. Then click ok.
   .

**Program:**

```
/* program to implement linear convolution */

#include<stdio.h>
#define LENGHT1 6   /*Length of i/p samples sequence*/
#define LENGHT2 4   /*Length of impulse response Co-
efficient */
```

```
int x[2*LENGHT1-1]={1,2,3,4,5,6,0,0,0,0,0};   /*Input
Signal Samples*/
int h[2*LENGHT1-1]={1,2,3,4,0,0,0,0,0,0,0};   /*Impulse
Response Co-efficient*/

int y[LENGHT1+LENGHT2-1];

main()
{
      int i=0,j;

      for(i=0;i<(LENGHT1+LENGHT2-1);i++)
      {
      y[i]=0;
      for(j=0;j<=i;j++)

            y[i]+=x[j]*h[i-j];

      }
      for(i=0;i<(LENGHT1+LENGHT2-1);i++)
      printf("%d\n",y[i]);

}
```

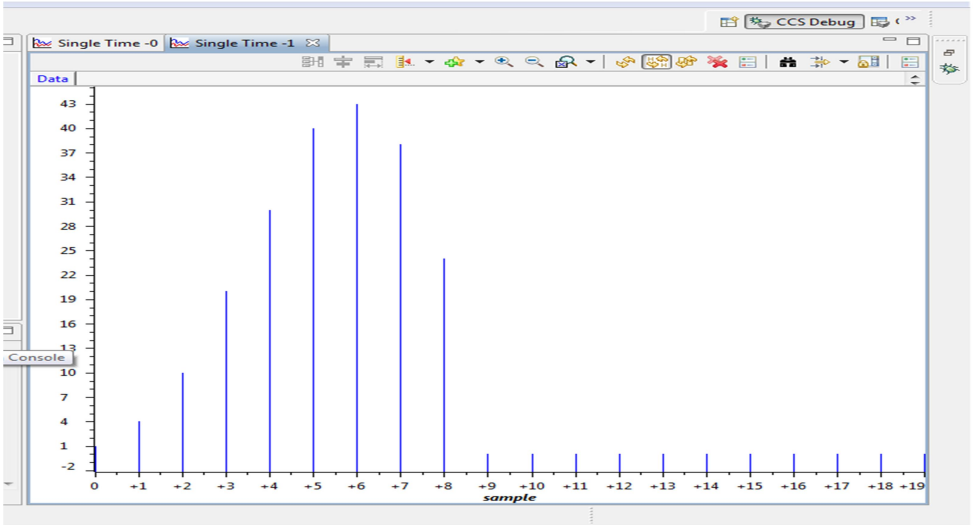
**Results:**

Thus, the Linear Convolution of two given discrete sequence has been performed. The input sequences are given in the program and the output will be displayed in the CCS software.

**Input** x[n] = {1, 2, 3, 4, 5, 6, 0, 0, 0, 0}
h[n] = {1, 2, 3, 4, 0, 0, 0, 0, 0, 0}

**Output:** y[n] = { 1, 4, 10, 20, 30, 40, 43, 38, 24, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

Now configure the Graphical window as shown below



**Discussions on results:**

Thus we have verified the Linear Convolution in Code composer studio environment by writing a C program.

From the results students will be able to

1) Discuss the steps required to interface TMS320C6713 Kit with Code composer studio environment.
2) Discuss the changes in the program to get the input sequences from user.
3) Discuss the steps required in graphical property dialog of the Code composer studio for graphical visualization of linear convolution output

# Experiment -8

**Aim:** Generation of Sine wave and square wave using TMS320C6713 DSK and Code Composer Studio.

**Equipments Required:** PC, TMS320C6713 DSK, Code composer Studiov5.5

**Theory:**

**Sinusoidal Wave:** The sine wave or sinusoidal wave is a mathematical curve that describes smooth repetitive oscillations. It can be represented in mathematical form as

$$y(t) = A\sin t(\omega t + \phi)$$

Where

A=Amplitude in V.

ω=angular frequency

ϕ=Phase in radins.

**Square Wave:** The square wave is a non sinusoidal periodic wave form which is represented as infinite summation of sinusoidal waves in which the amplitude alternates at a steady frequency between fixed minimum and maximum values with the same duration at maximum and minimum.

**Algorithm:**

1) Define frequency in C program.
2) Generate the signals using corresponding general formula.
3) Plot the graph in Code Composer Studio.

**Procedure for Sinusoidal wave form generation**

1) Step1: Creation of a Target configuration
   a. File→new→target configuration , you will get window as shown in fig1. You can give any target name with extension .ccxml (for example here I have taken target name as 6713.ccxml) after giving the name click on finish.

   Once you click on finish you will get general setup window as shown below in fig2. In the general setup you have select device and connection as mentioned below.

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

   Board or Device: TMS320C6713

   Then click on SAVE
2) Step 2: Launch the Target Configuration

   Go to view from toolbar

   View→ target configuration, you can see your target name under user defined as circled in fig 3 .

   Here under user defined it is showing 6713.ccxml it is my target.
3) Step 3: Connecting the target

   go to Run→connect target as shown in below.

Note: on mail CCS window there are two prospective are there one is CCS Debug and one more is CCS edit. In CCS debug it contain all option related to target and in CCS edit it contain all options related to projects( source, lib etc).

4) Step 4: Creation of a project
   First click on CCS edit as shown below
   Then go to File→new→CCS Project
   Then make as changes as mentioned.
   Project name: any name (for example SivewaveGen)
   Family : C6000
   Variant : C671xFloating-point DSP TMS320C6713
   Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
   Then select empty project, Then click on finish.
5) Step 5: Create a source file
   File→new→ source file
   Source file: give any name with extension .c(here I have given as SivewaveGen.c). Then click on finish. Now write a code on workspace, then go to file→save.
6) Step 6: Build the project
   Go to Project→Build Project
7) Step 7: Run the project
   Switch to CCS debug prospective as shown in below figure.
   Then go to Run→resume to run the program.
   You can see output of linear convoluted samples as in watch window by typing in expression as mention below.
   View→expression
   Then type output variable name y then enter as shown below.
8) Step 8: Plotting a graph
   Go to Tools→Graph→Single Time , then make graphical properties change as per program here in fig14 I had made changes which round up with red line . this changes applicable for linear convolution program. Then click ok.

.

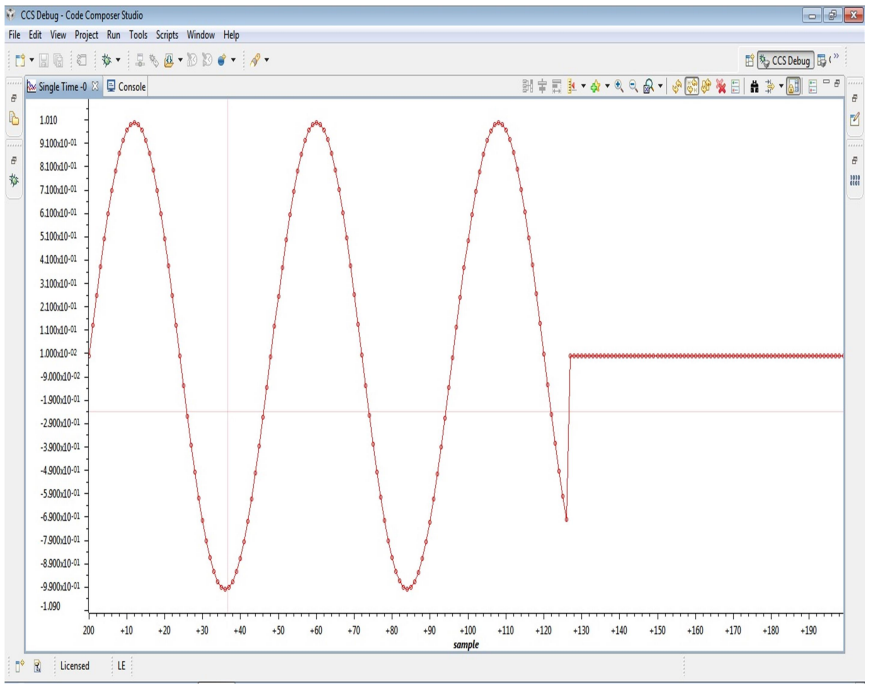
**Program:**

```
/* program for sinewave generation */

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

#define freq 400
 float m[128];

main()
{
      int n=0;
      for(n=0;n<127;n++)
      {
         m[n]=sin(2*3.14*freq*n/24000)
         printf("%f",m[n]);
      }

}
```

## Program for Square wave Generation

```
/* program for Squarewave generation */

#include<stdio.h>
#include<math.h>
#define freq 500
 float m[81];

main()
{
      int n=0;
      for(n=0;n<21;n++)
      {
         m[n]=5.0
      }
      for(n=21;n<41;n++)
      {
         m[n]=-5.0
      }
      for(n=41;n<61;n++)
      {
         m[n]=5.0
      }
      for(n=61;n<81;n++)
      {
         m[n]=-5.0
      }
}
```

## Plot of sine waveform



**Results:**

Thus, the waveform generation on sine wave and square wave is performed in Code composer environment by using a C program. Output will be displayed in the graphical window of CCS software**.**
Now configure the graphical window as shown below for Sine wave generation .the same procedure can be followed for square wave generation.

**Discussion on Results:**

Thus we have performed the waveform generation in Code composer studio environment by writing a C program.

From the results the student will be able to

1) Discuss the steps required to interface TMS320C6713 Kit with Code composer studio environment.
2) Discuss the changes in the program to get the input sequences from user.
3) Discuss the steps required in graphical property dialog of the Code composer studio for graphical visualization of the sine wave and square wave output.

# Experiment -9

**Aim:** Computation of DFT and DIT FFT using TMS320C6713 DSK and Code Composer Studio.

**Equipments Required:** PC, TMS320C6713 DSK, Code composer Studiov5.5

**Theory:**

Discrete Fourier Transform (DFT) is used for performing frequency analysis of discrete time signals. DFT gives a discrete frequency domain representation whereas the other transforms are continuous in frequency domain. The N point DFT of discrete time signal x[n] is given by the equation

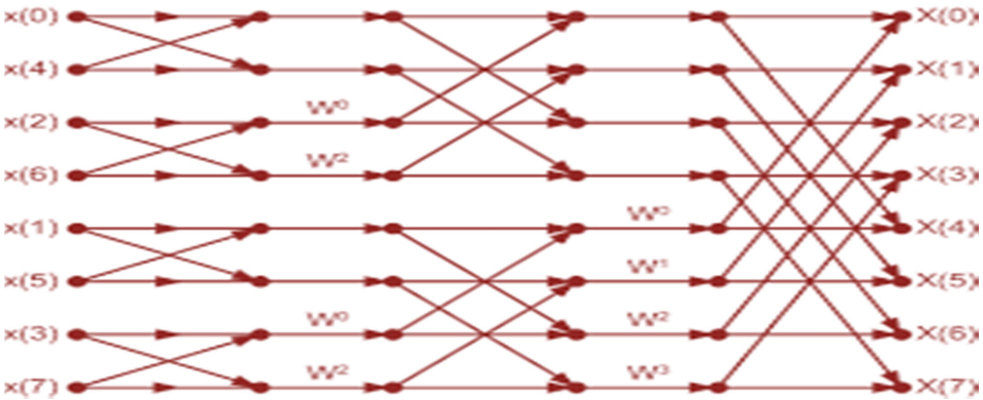
$$X(k) = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}; \quad k = 0,1,2,..N-1$$

The inverse DFT allows us to recover the sequence x[n] from the frequency samples

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

A Fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. There are many distinct FFT algorithms involving a wide range of mathematics, from simple complex number arithmetic to group theory and number theory.

A DFT decomposes a sequence of values into components of different frequencies. This operation is useful in many fields (see discrete Fourier transform for properties and applications of the transform) but computing it directly from the definition is often too slow to be practical. An FFT is a way to compute the same result more quickly: computing a DFT of N points in the naive way, using the definition, takes O(N2) arithmetical operations, while an FFT can compute the same result in only O(N log N) operations. The difference in speed can be substantial, especially for long data sets where N may be in the thousands or millions—in practice, the computation time can be reduced by several orders of magnitude in such cases, and the improvement is roughly proportional to N / log(N). This huge improvement made many DFT based algorithms practical; 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. The most well known FFT algorithms depend upon the factorization of N, but there are FFTs with O(N log N) complexity for all N, even for prime N.



## Algorithm:

1) Enter the length of the sequence for which DFT need to be computed.
2) Enter the sequence data as per the length of the sequence 4 or 8.
3) Compute the 4 point or 8 point DFT using the code.
4) Get the results in the Console window.

## Procedure for DFT Computation

1) Step1: Creation of a Target configuration
   a. File→new→target configuration , you will get window as shown in fig1. You can give any target name with extension .ccxml (for example here I have taken target name as 6713.ccxml) after giving the name click on finish.

   Once you click on finish you will get general setup window as shown below in fig2. In the general setup you have select device and connection as mentioned below.
   Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
   Board or Device: TMS320C6713
   Then click on SAVE
2) Step 2: Launch the Target Configuration
   Go to view from toolbar
   View→ target configuration, you can see your target name under user defined as circled in fig 3 .
   Here under user defined it is showing 6713.ccxml it is my target.
3) Step 3: Connecting the target
   go to Run→connect target as shown in below.
   Note: on mail CCS window there are two prospective are there one is CCS Debug and one more is CCS edit. In CCS debug it contain all option related to target and in CCS edit it contain all options related to projects( source, lib etc).
4) Step 4: Creation of a project
   First click on CCS edit as shown below
   Then go to File→new→CCS Project
   Then make as changes as mentioned.
   Project name: any name (for example DFTComput)
   Family : C6000
   Variant : C671xFloating-point DSP TMS320C6713
   Connection : Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
   Then select empty project, Then click on finish.
5) Step 5: Create a source file
   File→new→ source file
   Source file: give any name with extension .c (here I have given as DFTComput.c). Then click on finish. Now write a code on workspace, then go to file→save.
6) Step 6: Build the project
   Go to Project→Build Project
7) Step 7: Run the project
   Switch to CCS debug prospective as shown in below figure.
   Then go to Run→resume to run the program.

You can see output of linear convoluted samples as in watch window by typing in expression as mention below.
View→expression
Then type output variable name y then enter as shown below.

8) Step 8: Plotting a graph
Go to Tools→Graph→Single Time , then make graphical properties change as per program here in fig14 I had made changes which round up with red line . this changes applicable for linear convolution program. Then click ok.

## Program:

```
/* program for DFT Computation */

#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]);
}
}
```

## Results:

```
enter the length of the sequence
4
enter the sequence
1
2
3
4
X([0])=      10.000000    +      0.000000i
X([1])=      -1.999963    +      2.000022i
X([2])=      -2.000000    +      0.000059i
X([3])=      -2.000108    +      -1.999934i
```

## Program for the DIT FFT Algorithm:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define SWAP(a,b)var=(a);(a)=(b);(b)=var;

void main()
{
 int N,n,m,j,k,i,p;
 float data[200],real1,imag1,real2,imag2,var;
 float costheta,sintheta,t,Theta;

 clrscr();
 printf("\n\t\t Readix-2 DIT FFT algorithm\n\n");

 printf("\n\n Enter the number of samples in the sequence x(n),N=");
 scanf("%d",&N);

 printf("\n\n Enter the Samples of the Sequence x(n):\n");
 printf("\n Real part  Imaginary part");

 for(n=1;n<=N;n++)
 {
   printf("\n x(%d)=",n-1);
   scanf("%f%f",&data[2*n-1],&data[2*n]);
 }
 n=N<<1;
 j=1;

 for(i=1;i<n;i=i+2)
 {
  if(j>i)
  {
   SWAP(data[j],data[i]);
   SWAP(data[j+1],data[i+1]);
  }
  m=n>>1;
  while(m>=2 && j>m)
  {
   j-=m;
   m>>=1;
  }
  j+=m;
 }

 k=1;m=1;t=0.0;
 while((N/(2*k))>=1)
 {
  p=pow(2,m);
  n=1;
  Theta=((2*M_PI)/(float)p)*t;
  costheta=cos(Theta);
  sintheta=sin(Theta);

  for(i=1;i<=2*N;)
  {
   real1=data[i]+costheta*data[i+p]+sintheta*data[i+1+p];
   imag1=data[i+1]+costheta*data[i+1+p]-sintheta*data[i+p];
   real2=data[i]-costheta*data[i+p]-sintheta*data[i+1+p];
   imag2=data[i+1]-costheta*data[i+1+p]+sintheta*data[i+p];
```

```
   data[i]=real1;
   data[i+1]=imag1;
   data[i+p]=real2;
   data[i+p+1]=imag2;

   if(n<k)
   {
    t=t+1;
    Theta=((2*M_PI)/(float)p)*t;
    costheta=cos(Theta);
    sintheta=sin(Theta);
   }
   else
   {
    i=i+p+2;
    n=1;
    t=0;
    Theta=((2*M_PI)/(float)p)*t;
    costheta=cos(Theta);
    sintheta=sin(Theta);
   }
  }
  k=k<<1;
  m++;
 }

 printf("\n\n Output of DIT FFt is as follows:\n");
 printf("\n\n Real part of X[k]       Imaginary part of X[k]");
 for(n=1;n<=N;n++)
 {
  printf("\n%f\t\t %f ",data[2*n-1],data[2*n]);
 }
 }
```

## Results:

```
Enter the number of samples in the sequence x(n),N=8

 Enter the samples of the sequence x(n):

       Real Part   Imaginary Part

 x(0)=   0.5           0

 x(1)=   0.5           0

 x(2)=   0.5           0

 x(3)=   0.5           0

 x(4)=    0            0

 x(5)=    0            0

 x(6)=    0            0

 x(7)=    0            0


 Output of DIT FFT is as follows

       Real part of X(k)    Imaginary part of X(k)
```

```
2.000000        0.000000

0.500000        -1.207107

0.000000        0.000000

0.500000        -0.207107

0.000000        0.000000

0.500000        0.207107

0.000000        0.000000

0.500000        1.207107
```

**Discussion:**

Thus, the DFT computation is performed for N=4 using the Code composer environment by using a C program. Output will be displayed in the Console window of CCS software.

# Experiment -10

**Aim:** Generating the Responses of Low Pass and High Pass IIR filters using DSP Trainer Kit (TMS320C6713)

**Equipment Required:** PC Host (PC) with windows (95/98/Me/XP/NT/2000), TMS320C6713 DSP Starter Kit (DSK).Oscilloscope and Function generator, Code Composer Studio v3.0

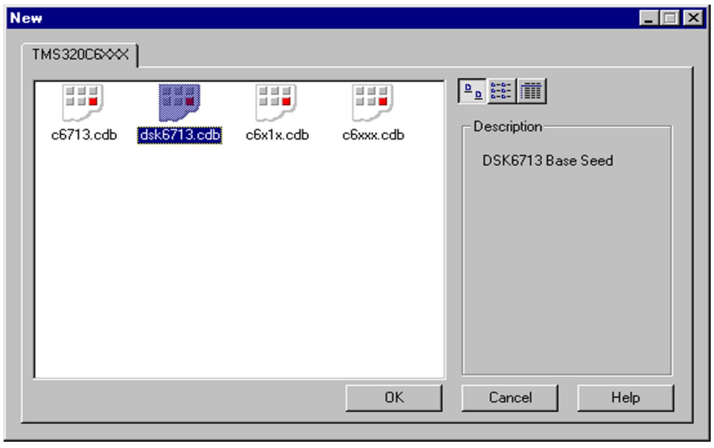
**Algorithm:**

We need to realize the Butter worth band pass IIR filter by implementing the difference equation y[n] = b0x[n] + b1x[n-1]+b2x[n-2]-a1y[n-1]-a2y[n-2] where b0 – b2, a0-a2 are feed forward and feedback word coefficients respectively [Assume 2nd order of filter].These coefficients are calculated using MATLAB.A direct form I implementation approach is taken.

1) Initialize the McBSP, the DSP board and the on board codec.
   "Kindly refer the Topic Configuration of 6713Codec using BSL"
2) Initialize the discrete time system , that is , specify the initial conditions. Generally zero initial conditions are assumed.
3) Take sampled data from codec while input is fed to DSP kit from the signal generator. Since Codec is stereo , take average of input data read from left and right channel . Store sampled data at a memory location.
4) Perform filter operation using above said difference equation and store filter Output at a memory location .
5) Output the value to codec (left channel and right channel) and view the output at Oscilloscope.
6) Step 6 - Go to step 3.

**Procedure for Real time Programs:**

1. Connect CRO to the Socket Provided for LINE OUT.
2. Connect a Signal Generator to the LINE IN Socket.
3. Switch on the Signal Generator with a sine wave of frequency 500 Hz. and Vp-p=1.5v
4. Now Switch on the DSK and Bring Up Code Composer Studio on the PC.
5. Create a new project with name codec.pjt.
6. From the File Menu ➔ new ➔ DSP/BIOS Configuration ➔select "dsk6713.cdb" and save it as "xyz.cdb"




7. Add "xyz.cdb" to the current project.
8. Add the given "codec.c" file to the current project which has the main function and calls all the other necessary routines.
9. Add the library file "dsk6713bsl.lib" to the current project
   Path → "C:\CCStudio\C6000\dsk6713\lib\dsk6713bsl.lib"
10. Copy files "dsk6713.h" and "dsk6713_aic23.h" from C:\CCStudio\C6000\dsk6713\include and paste it in current project.
11. Build, Load and Run the program.
12. You can notice the input signal of 500 Hz. appearing on the CRO verifying the codec configuration.
13. You can also pass an audio input and hear the output signal through the speakers.
14. You can also vary the sampling frequency using the DSK6713_AIC23_setFreq Function in the "codec.c" file and repeat the above steps.

## Procedure to execute IIR Filter Program

1) Switch on the DSP board.
2) Open the Code Composer Studio.
3) Create a new project
   Project → New (File Name. pjt , Eg: IIR.pjt)
4) Initialize on board codec.
5) Add the given above 'C' source file to the current project (remove codec.c source file from the project if you have already added).
6) Connect the speaker jack to the input of the CRO.
7) Build the program.
8) Load the generated object file (*.out) on to Target board.
9) Run the program
10) Observe the waveform that appears on the CRO screen.
11) Vary the frequency on function generator to see the response of filter.

## Program:

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

const signed int filter_Coeff[] =
{
     //12730,-12730,12730,2767,-18324,21137 /*HP 2500 */
     //312,312,312,32767,-27943,24367       /*LP 800 */
```

```
 //1455,1455,1455,32767,-23140,21735      /*LP 2500 */
     //9268,-9268,9268,32767,-7395,18367  /*HP 4000*/
       7215,-7215,7215,32767,5039,6171,         /*HP 7000*/
} ;

/* Codec configuration settings */
DSK6713_AIC23_Config config = { \
    0x0017,  /* 0 DSK6713_AIC23_LEFTINVOL  Left line input channel volume */
\
    0x0017,  /* 1 DSK6713_AIC23_RIGHTINVOL Right line input channel volume
*/\
    0x00d8,  /* 2 DSK6713_AIC23_LEFTHPVOL  Left channel headphone volume */
\
    0x00d8,  /* 3 DSK6713_AIC23_RIGHTHPVOL Right channel headphone volume */
\
    0x0011,  /* 4 DSK6713_AIC23_ANAPATH    Analog audio path control */
\
    0x0000,  /* 5 DSK6713_AIC23_DIGPATH    Digital audio path control */
\
    0x0000,  /* 6 DSK6713_AIC23_POWERDOWN  Power down control */
\
    0x0043,  /* 7 DSK6713_AIC23_DIGIF      Digital audio interface format */
\
    0x0081,  /* 8 DSK6713_AIC23_SAMPLERATE Sample rate control */
\
    0x0001   /* 9 DSK6713_AIC23_DIGACT     Digital interface activation */
\
};

/*
 *  main() - Main code routine, initializes BSL and generates tone
 */
void main()
{
    DSK6713_AIC23_CodecHandle hCodec;

    int l_input, r_input, l_output, r_output;

    /* Initialize the board support library, must be called first */
    DSK6713_init();

    /* Start the codec */
    hCodec = DSK6713_AIC23_openCodec(0, &config);

    DSK6713_AIC23_setFreq(hCodec, 3);

       while(1)
        {           /* Read a sample to the left channel */
             while (!DSK6713_AIC23_read(hCodec, &l_input));

             /* Read a sample to the right channel */
             while (!DSK6713_AIC23_read(hCodec, &r_input));

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

             /* Send a sample to the left channel */
             while (!DSK6713_AIC23_write(hCodec, l_output));

             /* Send a sample to the right channel */
             while (!DSK6713_AIC23_write(hCodec, r_output));
        }

    /* Close the codec */
    DSK6713_AIC23_closeCodec(hCodec);
}

signed int IIR_FILTER(const signed int * h, signed int x1)
{
```

```
static signed int x[6] = { 0, 0, 0, 0, 0, 0 };  /* x(n), x(n-1), x(n-2).
Must be static */
       static signed int y[6] = { 0, 0, 0, 0, 0, 0 };  /* y(n), y(n-1), y(n-
2). Must be static */
        int temp=0;

        temp = (short int)x1; /* Copy input to temp */

       x[0] = (signed int) temp; /* Copy input to x[stages][0] */

       temp =  ( (int)h[0] * x[0]) ;   /* B0 * x(n)   */

       temp += ( (int)h[1] * x[1]);    /* B1/2 * x(n-1) */
       temp += ( (int)h[1] * x[1]);    /* B1/2 * x(n-1) */
       temp += ( (int)h[2] * x[2]);    /* B2 * x(n-2) */

       temp -= ( (int)h[4] * y[1]);    /* A1/2 * y(n-1) */
       temp -= ( (int)h[4] * y[1]);    /* A1/2 * y(n-1) */
       temp -= ( (int)h[5] * y[2]);   /* A2 * y(n-2) */

    /* Divide temp by coefficients[A0] */

        temp >>= 15;

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

     /* Shuffle values along one place for next time */

     y[2] = y[1];   /* y(n-2) = y(n-1) */
     y[1] = y[0];   /* y(n-1) = y(n)   */

     x[2] = x[1];   /* x(n-2) = x(n-1) */
     x[1] = x[0];   /* x(n-1) = x(n)   */

     /* temp is used as input next time through */

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

## Results:

Thus the result can be observer on the CRO for various frequencies.

## Discussions on results:

The designing of IIR Low pass and High Pass Filters requires initialization of BSL codec.

From the results students will be able to

1) Discuss the real time interfacing of the TMS320C6713 Kit by using the full functionality of the board support library files and BSL Codec.
2) Discuss the effect of changing the value of coefficients for filters.
3) Discuss the steps required to do the connection of CRO and function generator to TMS320C6713 Kit.