

# **SRM VALLIAMMAI ENGINEERING COLLEGE**

(An Autonomous Institution)  
SRM Nagar, Kattankulathur – 603 203

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

## **LAB MANUAL**



**Regulation – 2023**

**EC3465 DIGITAL SIGNAL PROCESSING LABORATORY**

**ACADEMIC YEAR: 2025-2026 (EVEN SEM)**

**II YEAR ECE  
IV SEMESTER**

**Prepared by  
Dr. S. Manikandan, AP  
Dr. G. Sathishkumjar, AP  
Dr. R. Dhananjeyan, AP**

# **SRM VALLIAMMAI ENGINEERING COLLEGE**

(An Autonomous Institution)  
SRM Nagar, Kattankulathur – 603 203

## **DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

### **VISION OF THE INSTITUTE**

Educate to excel in social transformation

### **MISSION OF THE INSTITUTE**

- To contribute to the development of human resources in the form of professional engineers and managers of international excellence and competence with high motivation and dynamism, who besides serving as ideal citizen of our country will contribute substantially to the economic development and advancement in their chosen areas of specialization.
- To build the institution with international repute in education in several areas at several levels with specific emphasis to promote higher education and research through strong institute-industry interaction and consultancy.

### **VISION OF THE DEPARTMENT**

To excel in the field of electronics and communication engineering and to develop highly competent technocrats with global intellectual qualities.

### **MISSION OF THE DEPARTMENT**

- M1:** To educate the students with the state of art technologies to compete internationally, able to produce creative solutions to the society's needs, conscious to the universal moral values, adherent to the professional ethical code
- M2:** To encourage the students for professional and software development career
- M3:** To equip the students with strong foundations to enable them for continuing education and research.

# SRM VALLIAMMAI ENGINEERING COLLEGE

(An Autonomous Institution)

SRM Nagar, Kattankulathur – 603 203

## PROGRAM OUTCOMES

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

## PROGRAM SPECIFIC OUTCOME(PSOs)

- PSO1:** Ability to apply the acquired knowledge of basic skills, mathematical foundations, principles of electronics, modeling and design of electronics based systems in solving engineering Problems.
- PSO2:** Ability to understand and analyze the interdisciplinary problems for developing innovative sustained solutions with environmental concerns.
- PSO3:** Ability to update knowledge continuously in the tools like MATLAB, NS2, XILINIX and technologies like VLSI, Embedded, Wireless Communications to meet the industry requirements.
- PSO4:** Ability to manage effectively as part of a team with professional behavior and ethics.

# SYLLABUS

**EC3465**

**DIGITAL SIGNAL PROCESSING LABORATORY**

## **OBJECTIVES:**

1. To calculate the DFT using FFT
2. To design the IIR Filter.
3. To conceptualize the FIR Filter.
4. To implement the filters in standard structure.
5. To assess the finite word length effects.
6. To explore the architecture of a programmable digital signal processor.

## **LIST OF EXPERIMENTS:**

### **MATLAB / EQUIVALENT SOFTWARE PACKAGE/ DSP PROCESSOR BASED IMPLEMENTATION**

1. Generation of Elementary Discrete Time Signals.
2. Linear and Circular Convolution.
3. Computation of DFT.
4. Computation of FFT.
5. Design of FIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operation.
6. Design of Butterworth and Chebyshev IIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operations.
7. Study of architecture of Digital Signal Processor.
8. Perform MAC operation using various addressing modes.
9. Generation of various signals and random noise.
10. Implement an Up-sampling and Down-sampling operation in DSP Processor.

**TOTAL: 30 Periods**

## **COURSE OUTCOMES:**

On Completion of the course, the student will be able to

1. calculate the Fast Fourier Transforms by DIT and DIF techniques
2. design IIR Filters for the given specifications.
3. conceptualize FIR Filters with the given specifications
4. sketch the Filter in standard structures like Direct Form and Parallel Form
5. analyze performance degradation of digital signal processing systems due to finite precision
6. interpret the architectural details of fixed and floating digital signal processor

## **LIST OF EXPERIMENTS**

### **CYCLE I**

1. Generation of Elementary Discrete Time Signals.
2. Linear and Circular Convolution.
3. Computation of DFT.
4. Computation of FFT.
5. Design of FIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operation.

### **CYCLE II**

6. Design of Butterworth and Chebyshev IIR filters (LPF/HPF/BPF/BSF) and demonstrate the filtering operations.
7. Study of architecture of Digital Signal Processor.
8. Perform MAC operation using various addressing modes.
9. Generation of various signals and random noise.
10. Implement an Up-sampling and Down-sampling operation in DSP Processor

**Ex No:**

**Date:**

## **1. GENERATION OF ELEMENTARY DISCRETE TIME SIGNALS**

**Aim:**

To write a MATLAB program to generate unit step, unit impulse, unit ramp, exponential, sine waveform, cosine waveform, random signal and correlation.

**Prerequisite Lab Questions:**

1. Define Continuous Time and Discrete Time Signal?
2. How to represent Unit step, impulse and ramp function?
3. Compare stem & Plot functions.
4. Define Energy & Power signals.
5. Give an example for real time signals.

**Description:**

Input : Enter the sequence length for the generation of signals.

(a)The unit sample sequence (Unit Impulse) is

$$\begin{aligned}\delta(n) &= 1 \text{ for } n = 0 \\ &= 0 \text{ for } n \neq 0\end{aligned}$$

(b)The unit step sequence is

$$\begin{aligned}u(n) &= 1 \text{ for } n \geq 0 \\ &= 0 \text{ for } n < 0\end{aligned}$$

(c) The unit ramp sequence is denoted as  $u_r(n)$  is

$$\begin{aligned}u_r(n) &= n \text{ for } n \geq 0 \\ &= 0 \text{ for } n < 0\end{aligned}$$

(d)The exponential signal is a sequence of the form

$$x(n) = e^n \text{ for all 'n'}$$

(e)Sine signal  $x(t) = \sin\omega t$  , for all t values.

(f) Sine sequence  $x(n) = \sin\omega n$  , for all n values.

(g)Cosine signal  $x(t) = \cos\omega t$  , for all t values.

(h)Cosine signal  $x(t) = \cos\omega n$  , for all n values.

Output: Plot of all the above mentioned signals.

**Algorithm:**

1. Start the program.
2. Generate unit step, unit impulse, unit ramp, exponential, sine waveform, cosine waveform, random signal and correlation.
3. Execute the program.
4. Plot the output for each sequence.
5. Stop the program.

### Program:

```
%Program for Unit Impulse Signal
clc;
clear all;
t=-3:1:3;
y=[zeros(1,3),ones(1,1),zeros(1,3)];
figure(1);
subplot(2,2,1);
stem(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Unit impulse signal');
```

```
%Program for Unit Step Signal
t=0:5;
y=[ones(1,6)];
subplot(2,2,2);
stem(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Unit Step Signal');
```

```
%Program for Ramp Signal
t=0:5;
y=t;
subplot(2,2,3);
stem(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Ramp Signal');
```

```
%Program for Exponential Signal
t=0:0.1:5;
y=exp(t);
subplot(2,2,4);
stem(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Exponential Signal');
figure(2);
```

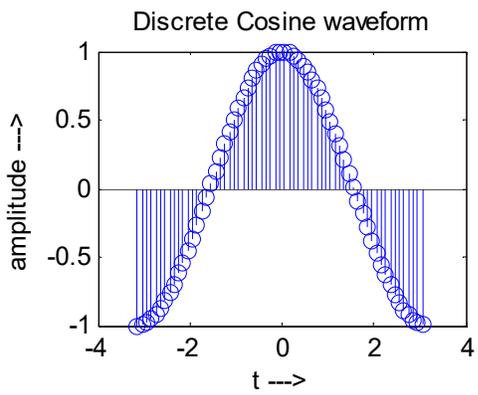
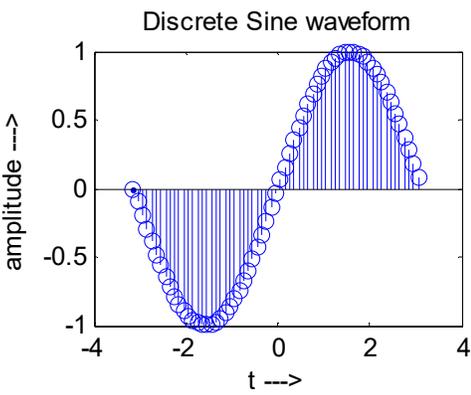
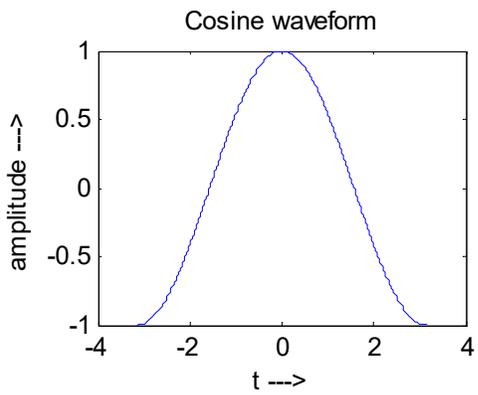
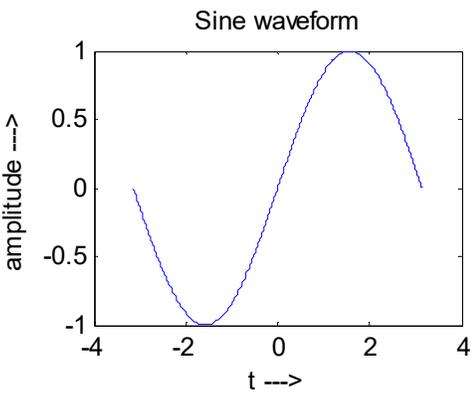
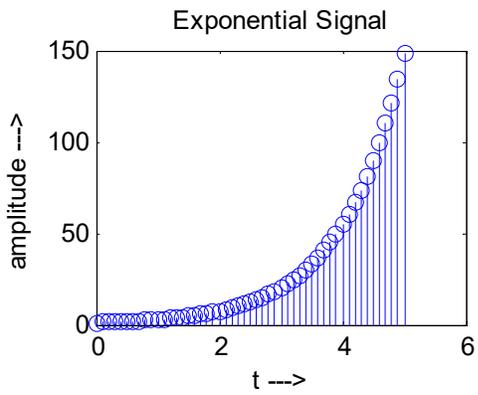
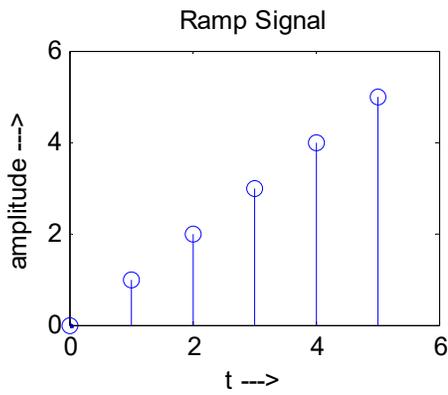
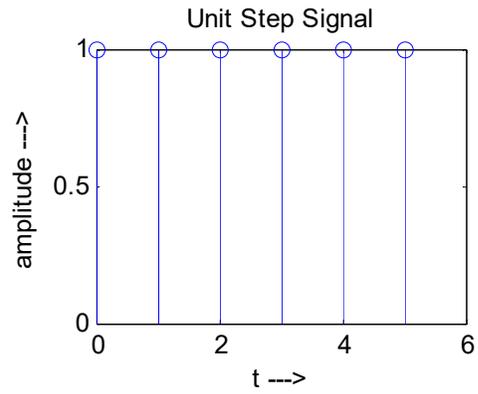
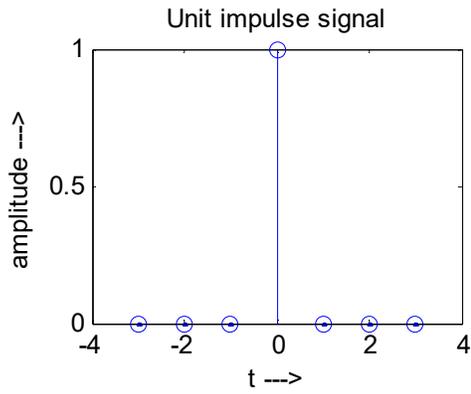
```
%Program for Sine waveform
t=-pi:0.01:pi;
y=sin(t);
subplot(2,2,1);
plot(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Sine waveform');
```

```
%Program for Cosine waveform
t=-pi:0.01:pi;
y=cos(t);
subplot(2,2,2);
plot(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Cosine waveform');
```

```
%Program for Discrete Sine waveform
t=-pi:0.1:pi;
y=sin(t);
subplot(2,2,3);
stem(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Discrete Sine waveform');
```

```
%Program for Discrete Cosine waveform
t=-pi:0.1:pi;
y=cos(t);
subplot(2,2,4);
stem(t,y);
ylabel('amplitude --->');
xlabel('t --->');
title('Discrete Cosine waveform');
```

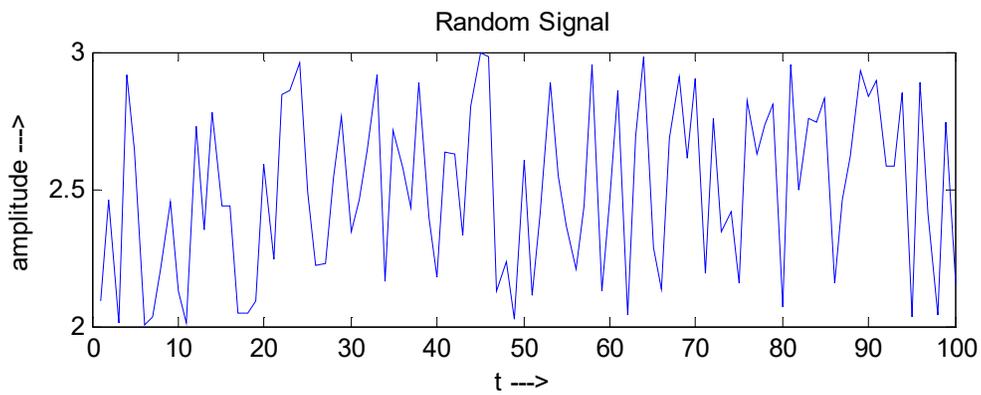
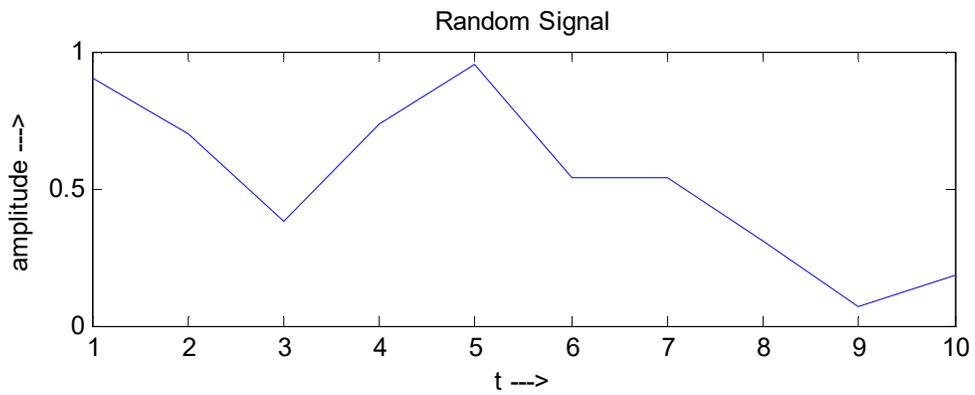
# Model Graph:



```

%Program for Random Signal
t = rand(1,10);
subplot(2,1,1);
plot(t);
ylabel('amplitude --->');
xlabel('t --->');
title('Random Signal');
disp('t=');
disp(t);
a=2;
b=3;
t1 = a + (b-a).*rand(100,1);
subplot(2,1,2);
plot(t1)
ylabel('amplitude --->');
xlabel('t --->');
title('Random Signal');
disp('t1=');
disp(t1);

```



### **Postlab Questions:**

1. Modify Program (b) to generate a delayed unit step sequence  $u[n]$  with an advance of “n” samples. Run the modified program and display the sequence generated.
2. What is the subplot command used to plot 4 figures in 2 rows and 2 columns?
3. How can you compute the average power of the generated sinusoidal sequence?
4. Write the syntax to determine auto correlation of the given signal  $x[n]$ .
5. Give the examples for 1-D, 2-D & 3-D signals.

### **Result:**

Thus a program to generate sequence for unit step, unit impulse, unit ramp, exponential, sine, cosine and correlation signal using MATLAB was written and executed and the corresponding simulation output is plotted.

**Ex No:**

**Date:**

## **2. LINEAR AND CIRCULAR CONVOLUTION**

**Aim:**

To write a program to find out the linear convolution of two sequence using Matlab.

**Prerequisite Lab Questions:**

1. Define Convolution.
2. What is zero padding ?
3. List the properties of linear convolution.
4. What is the MATLAB function used to perform linear & circular convolution?
5. How will you determine the maximum length of the convolution of the sequences?

**Description :**

**Input :** Give the input  $x(n)$  and  $h(n)$  in matrix form.

**Output :**  $y(n)$ , using the formula

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

**Algorithm:**

1. Start the program.
2. Get the two input sequence  $x(n)$  and  $h(n)$ .
3. Flip the  $h$  vector to the right or left direction.
4. Calculate the length of the input and output sequence.
5. Append zeros to the input to get the same length as output.
6. Sum the product of  $x(n)$  and shifted sequence of  $h(n-k)$ .
7. Execute the program, display the result and verify theoretically.
8. Plot the graph for the input and output sequence.
9. Stop the program.

**Program:**

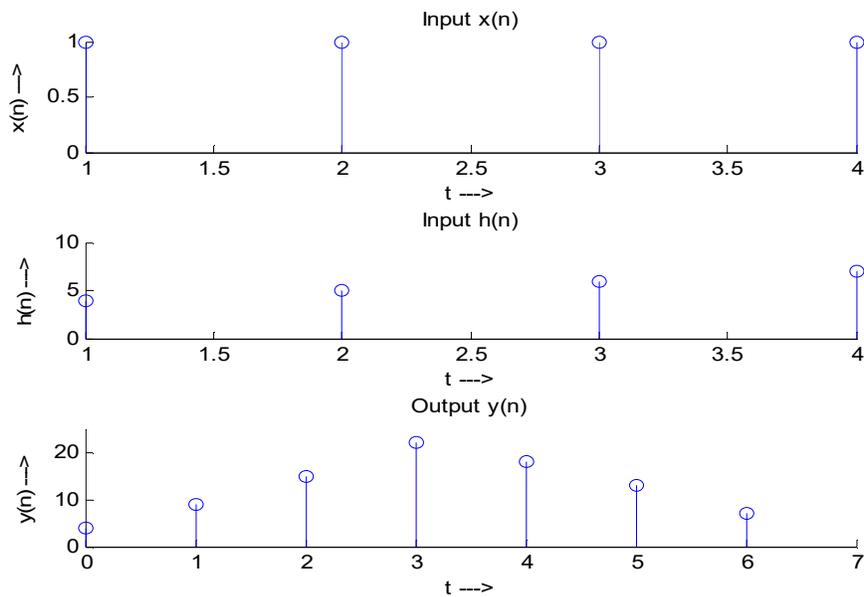
```
% Program for Linear Convolution
clc;
clear all;
close all;
%x=[1 1 1 1];
%h=[4 5 6 7];
x=input('enter the value x=');
h=input('enter the value h=');
subplot(3,1,1);
stem(x);
title('Input x(n)');
ylabel('x(n) --->');
xlabel('t --->');
subplot(3,1,2);
stem(h);
title('Input h(n)');
```

```

ylabel('h(n) --->');
xlabel('t --->');
h1=fliplr(h);
n1=length(x);
n2=length(h);
n=n1+n2-1;
x=[zeros(1,n2-1),x];
h1=[h1,zeros(1,n1-1)];
for i=1:n
    new=[zeros(1,i-1),h1(1:n-i+1)];
    out(i)=x*new';
end
disp('output=');
disp(out);
subplot(3,1,3);
t=0:1:n-1;
stem(t,out);
title('Output y(n)');
ylabel('y(n) --->');
xlabel('t --->');

```

### MODEL GRAPH



### THEORETICAL CALCULATION

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

Enter the value  $x=[1 \ 1 \ 1 \ 1]$

Enter the value  $h=[4 \ 5 \ 6 \ 7]$

Output= 4 9 15 22 18 13 7

## EX NO: 2 CIRCULAR CONVOLUTION

### **Aim:**

To write a program to find out the circular convolution of two sequences using MATLAB.

### **Description:**

**Input :** Give the input  $x(n)$  and  $h(n)$  in matrix form.

**Output :**  $y(n)$ , using the formula  
$$y(n) = x(n) \otimes h(n)$$

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k)$$

### **Algorithm:**

1. Start the program.
2. Get the two input sequence  $x(n)$  and  $h(n)$  and compute the length of the sequence.
3. Append zeros to  $x(n)$  or  $h(n)$  to make both sequence of same length.
4. Rotate the sequence  $h(-k)$ , 'n' times. If 'n' is positive shift the sequence in anticlockwise direction else in the clockwise direction.
5. Determine the product of sequence  $x(n)$  and shifted  $h(n)$  sequence and sum it.
6. Execute the program, display the result and verify it theoretically.
7. Plot the output graph for each sequence.
8. Stop the process.

### **Program:**

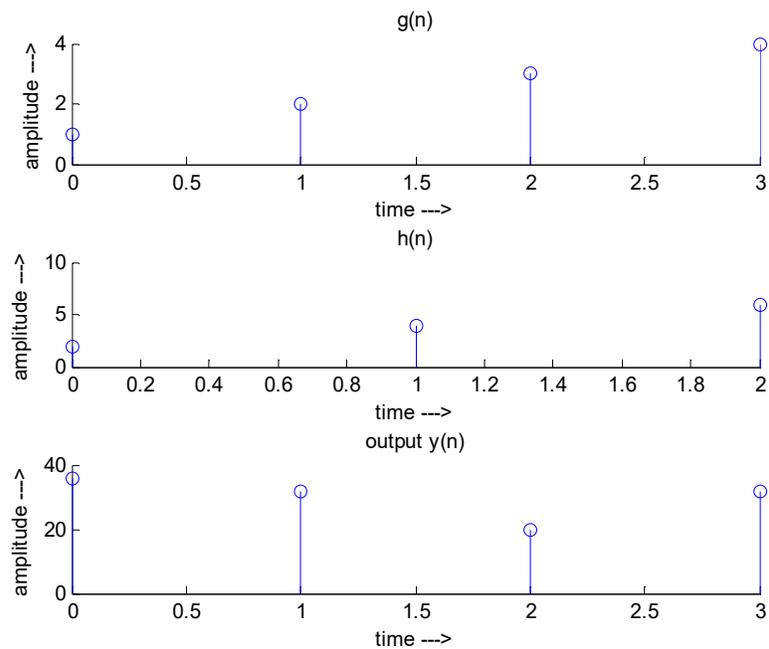
```
% Program for Circular Convolution
clc;
clear all;
close all;
%g=[1,2,3,4];
%h=[2,4,6];
x=input('enter the sequence');
h=input('enter the sequence');
N1=length(x);
N2=length(h);
subplot(3,1,1);
n=0:1:N1-1;
stem(n,x);
title('x(n)');
ylabel('amplitude --->');
xlabel('time --->');
subplot(3,1,2);
n=0:1:N2-1;
stem(n,h);
title('h(n)');
ylabel('amplitude --->');
xlabel('time --->');
```

```

N=max(N1,N2);
if(N1<=N);
    x1=[x,zeros(1,N-N1)];
end
if(N2<=N);
h1=[h,zeros(1,N-N2)];
end
for n=1:N
    y(n)=0;
for i=1:N
    j=n-i+1;
if(j<=0)
    j=N+j;
end
    y(n)=y(n)+x1(i)*h1(j);
end
end
disp('y=');
disp(y);
subplot(3,1,3);
n=0:1:N-1;
stem(n,y);
title('output y(n)');
ylabel('amplitude --->');
xlabel('time --->');

```

## MODEL GRAPH



THEORITICAL CALCULATION

Using graphical or Matrix method

$$y(n) = x(n) \otimes h(n)$$

$$y(n) = \sum_{k=0}^{N-1} x(k)h(n-k)$$

Enter the sequence g=[1,2,3,4]

Enter the sequence h=[2,4,6]

Output Sequence is = 36 32 20 32

**Matrix Method:**

$$\begin{pmatrix}
 x(0) & x(N-1) & x(N-2) & \dots & x(1) \\
 x(1) & x(0) & x(N-1) & \dots & x(2) \\
 x(2) & x(1) & x(0) & \dots & x(N-1) \\
 \vdots & \vdots & \vdots & \ddots & \vdots \\
 x(N-1) & x(N-2) & x(N-3) & \dots & x(0)
 \end{pmatrix}
 \begin{pmatrix}
 h(0) \\
 h(1) \\
 h(2) \\
 \vdots \\
 h(N-1)
 \end{pmatrix}
 =
 \begin{pmatrix}
 y(0) \\
 y(1) \\
 y(2) \\
 \vdots \\
 y(N-1)
 \end{pmatrix}$$

**Postlab Questions :**

1. What is the difference between Linear convolution & circular convolution?
2. State convolution property of a signal.
3. What are sectioned convolutions & what is the need of it?
4. Write the applications of convolution.
5. What is the relation between convolution in time domain and frequency domain ?

**Result:**

Thus a program to find out the linear and circular convolution of two sequences using Matlab was written and executed.

**Ex No:**

**Date:**

### **3. COMPUTATION OF DFT**

**Aim:**

To analyze the amplitude and phase response of the DFT and IDFT signal using MATLAB.

**Prerequisite Lab Questions:**

1. Write the analysis and synthesis equation for DFT?
2. What do you mean by the property of shifting of DFT?
3. Find the value of twiddle factor for  $N=8, K=2$ .
4. How many multiplications and additions are required to compute  $N$  point DFT ?
5. What is the relationship between DFT and Z-transform?

**Algorithm:**

1. Start the program.
2. Get the two input sequence  $x(n)$  and type of the DFT.
3. Compute the Discrete Fourier Transform (DFT) of the given sequence.
4. Find out the magnitude and phase response of the given sequence.
5. Execute the program, display the result and verify it.
6. Plot the output graph for magnitude and phase.
7. Compute the Inverse Discrete Fourier Transform (IDFT) of the given sequences.
8. Display the results verify it and plot the result.
9. Stop the process.

**Program:**

```
% Program for Discrete Fourier Transform
clc;
clear all;
close all;
%x=[1 2 3 4 5 6 7 8];
%N=[8];
x=input('enter the input x=');
N=input('enter the order of dft N=');
k=0:1:N-1;
disp('Input Sequence x=');
disp(x);
XK=fft(x,N);
disp('XK=');
disp(XK);
R=real(XK);
disp('Real part=');
disp(R);
I=imag(XK);
disp('Imaginary part=');
disp(I);
```

```

mag=abs(XK);
disp('Magnitude=');
disp(mag);
ang=angle(XK);
disp('Angle=');
disp(ang);
xn=ifft(XK,N);
disp('IDFT xn=');
disp(xn);

figure(1);
subplot(2,1,1);
stem(x);
xlabel('time -->');
ylabel('Amplitude -->');
title('Given sequence');

subplot(2,1,2);
stem(R,'r*');
hold on;
stem(I);
legend('Real Part','Imaginary Part');
xlabel('time -->');
ylabel('Amplitude -->');
title('Discrete Fourier Transform')

figure(3);
subplot(2,2,1);
stem(k,mag);
xlabel('time -->');
ylabel('x(k) --->');
title('Magnitude response');

subplot(2,2,2);
stem(k,ang);
xlabel('time -->');
ylabel('ang x(k) --->');
title('Phase response');

subplot(2,2,3);
stem(k,xn);
xlabel('time -->');
ylabel('x(n)');
title('Inverse Discrete Fourier transform');
%Spectrum analysis
Fs=[10];
xdft = XK(1:N/2+1);
psdx = (1/(Fs*N)) * abs(xdft).^2;
psdx(2:end-1) = 2*psdx(2:end-1);

```

```

freq = 0:Fs/N:Fs/2;
subplot(2,2,4);
plot(freq,psdx);
grid on;
xlabel('hz -->');
ylabel('db');
title('Spectrum analysis DFT');
disp('Spectrum analysis DFT=');
disp(psdx);

```

### Output:

enter the input x=[1 2 3 4 5 6 7 8]

enter the order of dft N=[8]

Input Sequence x=1 2 3 4 5 6 7 8

XK=

```

36.0000 + 0.0000i
-4.0000 - 9.6569i
-4.0000 - 4.0000i
-4.0000 - 1.6569i
-4.0000 + 0.0000i
-4.0000 + 1.6569i
-4.0000 + 4.0000i
-4.0000 + 9.6569i

```

Real part= 36 -4 -4 -4 -4 -4 -4 -4

Imaginary part= 0 9.6569 4.0000 1.6569 0 -1.6569 -4.0000 -9.6569

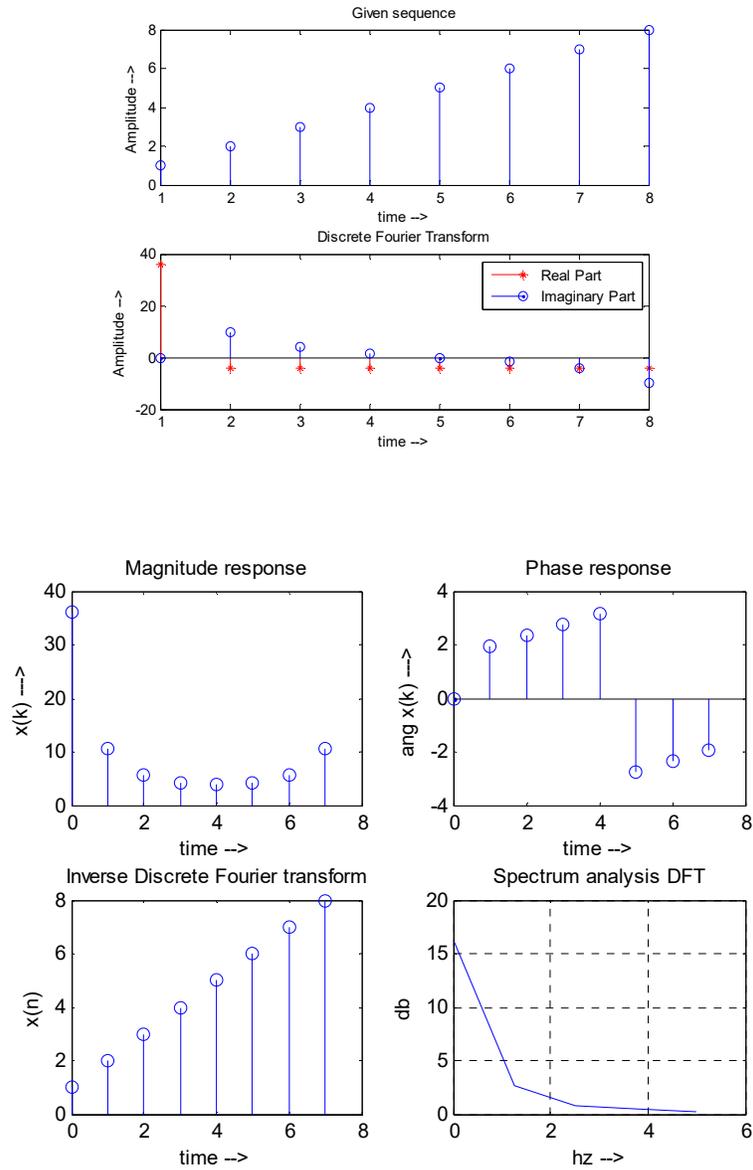
Magnitude= 36.0000 10.4525 5.6569 4.3296 4.0000 4.3296 5.6569 10.4525

Angle= 0 1.9635 2.3562 2.7489 3.1416 -2.7489 -2.3562 -1.9635

IDFT xn= [1 2 3 4 5 6 7 8]

Spectrum analysis DFT= 16.2000 2.7314 0.8000 0.4686 0.2000

## Model Graph



## Postlab Questions :

1. Mention the applications of DFT?
2. What is the drawback in Fourier transform and how it can be avoided?
3. Write the 4-point DFT matrix representation.
4. How many multiplications and additions are required to compute DFT for  $N=512$ ?
5. Distinguish between DFT and DTFT.

## **Result:**

Thus a MATLAB program for Spectrum analysis of the DFT and IDFT signal was written, executed and the graphs were plotted.

**Ex : No:**

**Date:**

## 4. COMPUTATION OF FFT

**Aim:**

To write a program to find out the Fast Fourier transform and Inverse Fast Fourier transform of the sequence using MATLAB and display the magnitude and phase response.

**Prerequisite Lab Questions:**

1. What do you mean by the term “bit reversal” as applied to FFT?
2. How many multiplications and additions are required to compute N point DFT using radix 2 FFT?
3. What are the advantages of FFT algorithm over direct computation of DFT?
4. State and periodicity and symmetry property of FFT?
5. Define in place computation?

**Description**

**Input** : Enter input x (n) as a sequence  
**Output** : X (k) can be obtained by using formula,

**FFT**

$$X(k) = \sum_{n=0}^{N-1} x(n) \omega_N^{kn}$$

**IFFT**

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \omega_N^{-kn}$$

$$\omega_N^{kn} = e^{-j \frac{2\pi kn}{N}}$$

**Algorithm:**

1. Start the program.
2. Get the two input sequence x (n) and type of the FFT.
3. Compute the fast Fourier transform (FFT) of the given sequence.
4. Find out the magnitude and phase response of the given sequence.
5. Execute the program, display the result and verify it.
6. Plot the output graph for magnitude and phase.
7. Compute the inverse fast Fourier transform (IFFT) of the given sequences.
8. Display the results verify it and plot the result.
9. Stop the process.

```

Program:
% Program for FFT & IFFT
clc;
clear all;
close all;
%x=[2 2 2 2 1 1 1 1];
%N=[8];
x=input('enter the input x=');
N=input('enter the order of fft N=');
k=0:1:N-1;

disp('Input Sequence x=');
disp(x);

XK=fft(x,N);
disp('XK=');
disp(XK);

R=real(XK);
disp('Real part=');
disp(R);

I=imag(XK);
disp('Imaginary part=');
disp(I);

mag=abs(XK);
disp('Mag=');
disp(mag);

ang=angle(XK);
disp('angle=');
disp(ang);

xn=ifft(XK,N);
disp('IFFT xn=');
disp(xn);

figure(1);
stem(R,'r*');
hold on;
stem(I);
legend('Real Part','Imaginary Part');
xlabel('time -->');
ylabel('Amplitude -->');
title('Fast Fourier Transform')

```

```

figure(2);
subplot(2,2,1);
stem(x);
xlabel('time -->');
ylabel('Amplitude -->');
title('Given sequence');
subplot(2,2,2);
stem(k,mag);
xlabel('time -->');
ylabel('x(k) --->');
title('Magnitude response');

subplot(2,2,3);
stem(k,ang);
xlabel('time -->');
ylabel('ang x(k) --->');
title('Phase response');

subplot(2,2,4);
stem(k,xn);
xlabel('time -->');
ylabel('x(n)');
title('Inverse Fast Fourier transform');

```

**Output:**

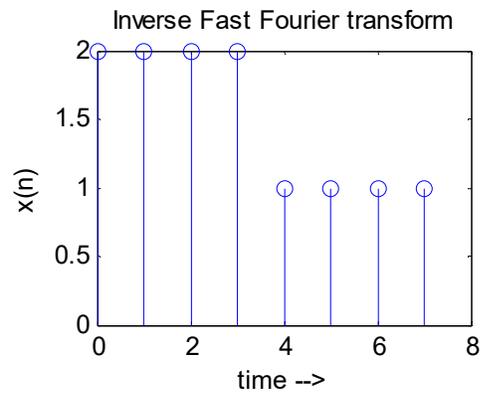
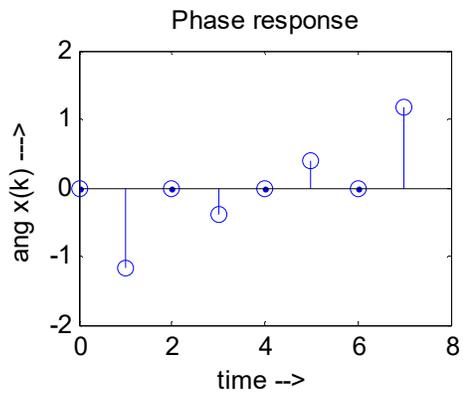
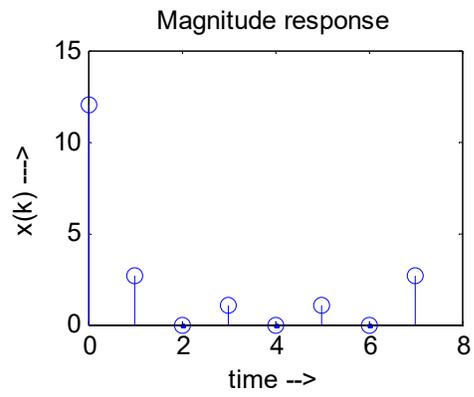
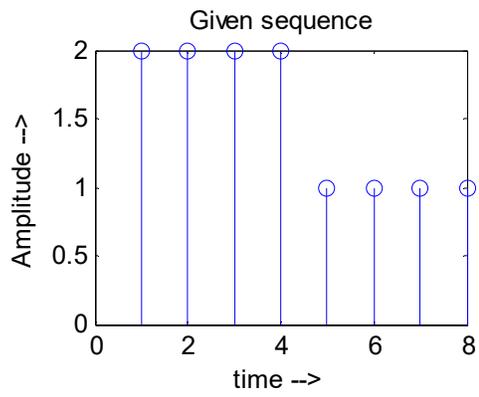
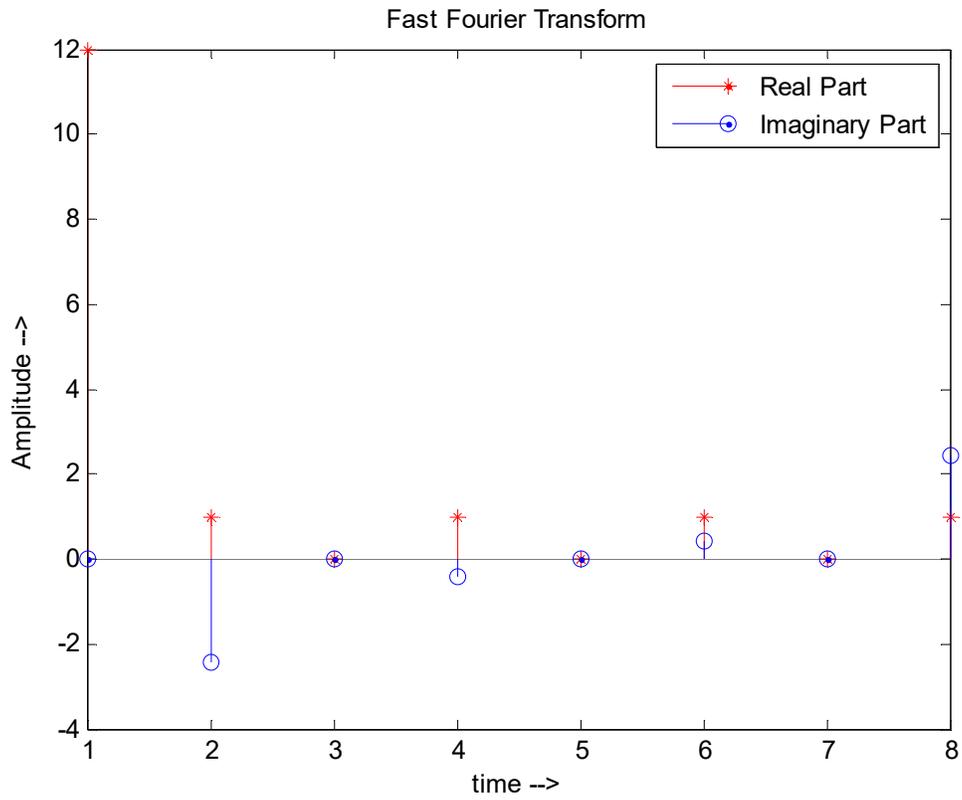
```

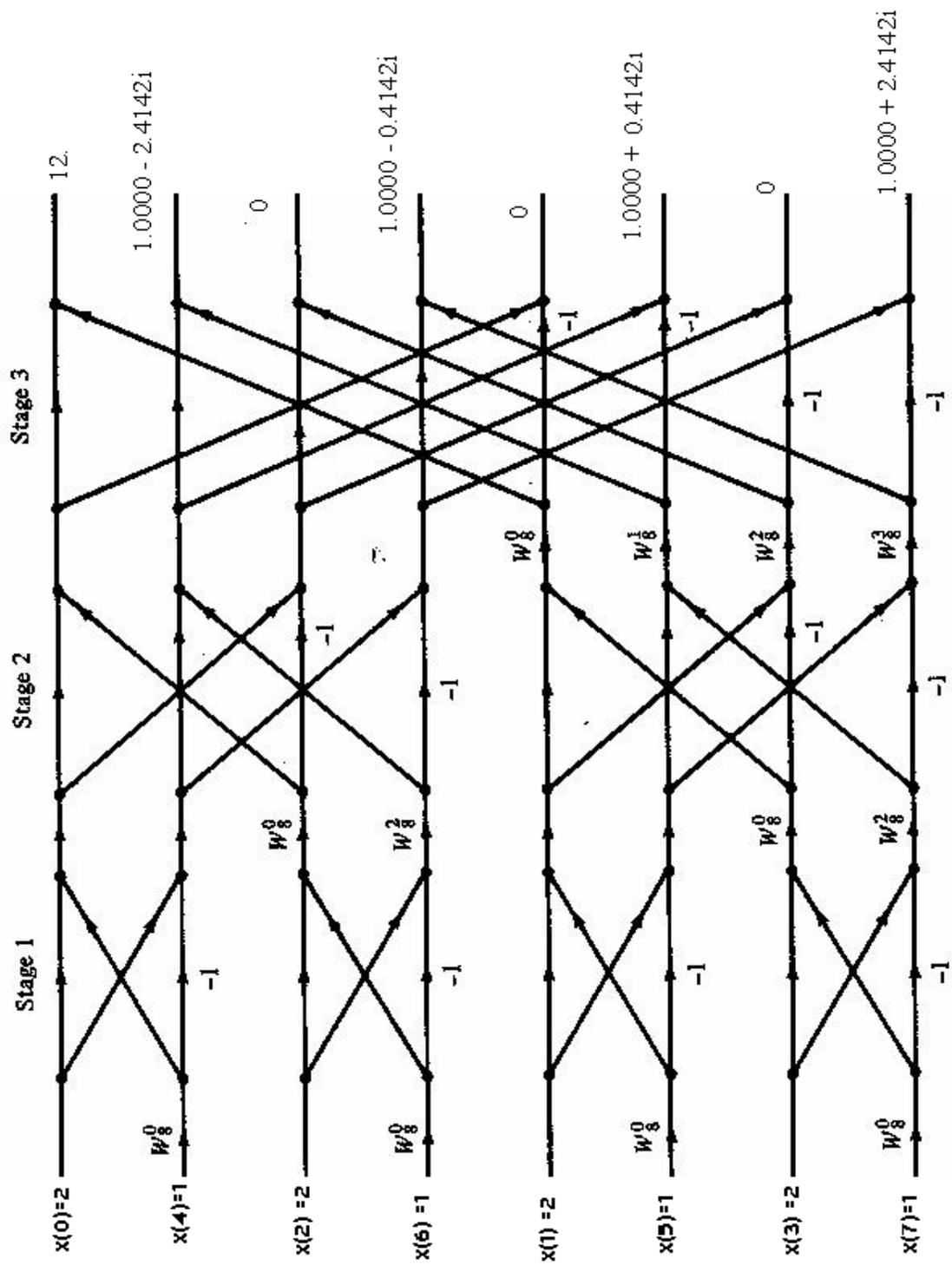
enter the input x=[2 2 2 2 1 1 1 1]
enter the order of fft N=[8]
Input Sequence x=  2   2   2   2   1   1   1   1
XK=
 12.0000 + 0.0000i
  1.0000 + 2.4142i
  0.0000 + 0.0000i
  1.0000 + 0.4142i
  0.0000 + 0.0000i
  1.0000 - 0.4142i
  0.0000 + 0.0000i
  1.0000 - 2.4142i

Real part=  12   1   0   1   0   1   0   1
Imaginary part=    0 -2.4142    0 -0.4142    0  0.4142    0  2.4142
Mag=  12.0000  2.6131    0  1.0824    0  1.0824    0  2.6131
Angle=    0 -1.1781    0 -0.3927    0  0.3927    0  1.1781
IFFT xn=  2   2   2   2   1   1   1   1

```

# Model Graph:





**PostLab Questions:**

1. How many multiplications and additions are involved in radix-2 FFT for  $N=1024$  when compared to DFT?
2. Draw the 2-point DIF-FFT butterfly structure.
3. Compare DIF and DIT FFT algorithm.
4. Why FFT is needed?
5. Mention the applications of FFT algorithm?

**Result:**

Thus a program to find out the fast Fourier (FFT) and inverse fast Fourier transform (IFFT) of the sequence using Matlab was written and executed.

**Ex:No:**

**Date:**

**5. DESIGN OF FIR FILTERS (LPF/HPF/BPF/BSF) AND  
DEMONSTRATE THE FILTERING OPERATION**

**Aim:**

To write a program to design the FIR low pass, high pass, band pass and band stop filters and obtain the frequency response of the filter using MATLAB.

**Prerequisite Lab Questions:**

1. Mention the advantages and disadvantages of FIR filters.
2. What are the classifications of filters?
3. List the techniques for designing FIR filters.
4. What are the features of windowing method?
5. What are the types of windows in FIR filter design?

**Description:**

**Input:** Give the cutoff frequency and order of the filter.

- Output:**
- 1) Find the impulse response  $h_d(n)$  for the given cut-off frequency and Order of the filter.
  - 2) Decide the window function  $w(n)$  and compute the window Coefficients.
  - 3) Find the filter coefficients  $h(n) = h_d(n) * w(n)$ .
  - 4) Plot the frequency response for Low pass, High pass, Band pass and Band stop filters

**Algorithm:**

1. Start the program.
2. Get the cutoff frequency and order of the filter as input...
3. Find the ideal impulse response of the filter  $h_d(n)$ .
4. Select a window function  $w(n)$ .
5. Find out the filter coefficients  $h(n)$  by multiplying  $h_d(n)$  and  $w(n)$ .
6. Compute the frequency response of the filter.
7. Repeat the process for high pass, band pass and band stop filters.
8. Execute the program, display the result and verify it.
9. Plot the output graph for  $h_d(n)$ ,  $w(n)$ ,  $h(n)$  and frequency response of LPF, HPF, BPF and BSF.
10. Stop the process.

**Program:**

```
% Ideal LowPass filter computation
% hd = ideal impulse response between 0 to M-1
% wc = cutoff frequency in radians
% M = length of the ideal filter
% Save as ideal_lp

% Function:
function [hd]=ideal_lp(wc,M)
alpha=(M-1)/2;
n=0:1:M-1;
N=n-alpha+eps; % add smallest number to avoid divide by zero
hd=sin(wc*N)./(pi*N);
```

**Program:**

```
%PROGRAM FOR LOW PASS FILTER
```

```
clc;
clear all;
close all;
M=7;
wc=1.2;
n=0:1:M-1;
hd=ideal_lp(wc,M);
w=hamming(M);
h=hd.*w';
disp('Filter coefficients for LPF=');
```

```
[M1,P]=freqz(h,1,2000);
mag=20*log10(abs(M1));
```

```
subplot(2,2,1);
stem(n,hd);
xlabel('n--->');
ylabel('hd(n)--->');
title('Ideal Impulse Response');
disp('hd=');
disp(hd);
subplot(2,2,2);
stem(n,w);
xlabel('n--->');
ylabel('w(n)--->');
title('Hamming Window');
disp('w=');
disp(w);
subplot(2,2,3);
```

```

stem(n,h);
xlabel('n--->');
ylabel('h(n)--->');
title('Actual Impulse Response');
disp('h=');
disp(h);
subplot(2,2,4);
plot(P/pi, mag);
grid on;
xlabel('Normalized frequency--->');
ylabel('Magnitude in db--->');
title('Frequency Response ');

```

**Output:**

Filter coefficients for LPF=

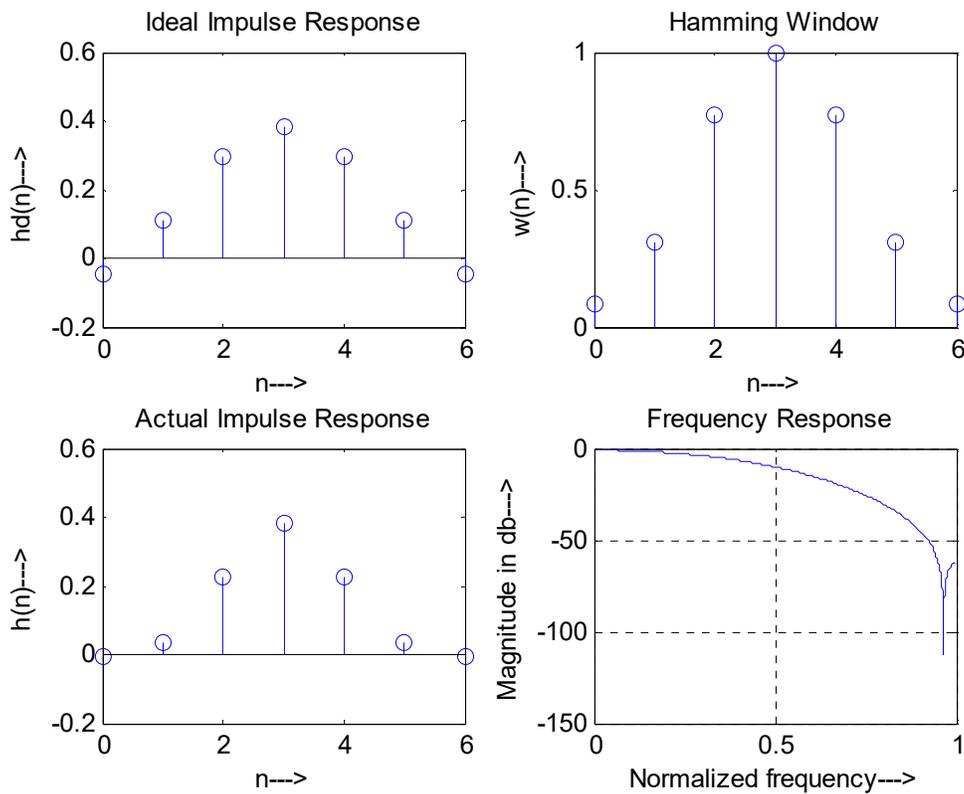
hd= -0.0470 0.1075 0.2967 0.3820 0.2967 0.1075 -0.0470

w = 0.0800 0.3100 0.7700 1.0000 0.7700 0.3100 0.0800

h = -0.0038 0.0333 0.2284 0.3820 0.2284 0.0333 -0.0038

**Model Graph:**

**LOW PASS FILTER**



```

%PROGRAM FOR HIGH PASS FILTER
clc;
clear all;
close all;
M=11;
wc=1.2;
n=0:1:M-1;
hd=ideal_lp(pi,M)-ideal_lp(wc,M);
w=hann(M);
h=hd.*w';
disp('Filter coefficients for HPF=');

[M1,P]=freqz(h,1,2000);
mag=20*log10(abs(M1));

subplot(2,2,1);
stem(n,hd);
xlabel('n--->');
ylabel('hd(n)--->');
title('Ideal Impulse Response');
disp('hd=');
disp(hd);

subplot(2,2,2);
stem(n,w);
xlabel('n--->');
ylabel('w(n)--->');
title('Hanning Window');
disp('w=');
disp(w);

subplot(2,2,3);
stem(n,h);
xlabel('n--->');
ylabel('h(n)--->');
title('Actual Impulse Response');
disp('h=');
disp(h);

subplot(2,2,4);
plot(P/pi, mag);
grid on;
xlabel('Normalized frequency--->');
ylabel('Magnitude in db--->');
title('Frequency Response ');

```

**Output:**

Filter coefficients for HPF=

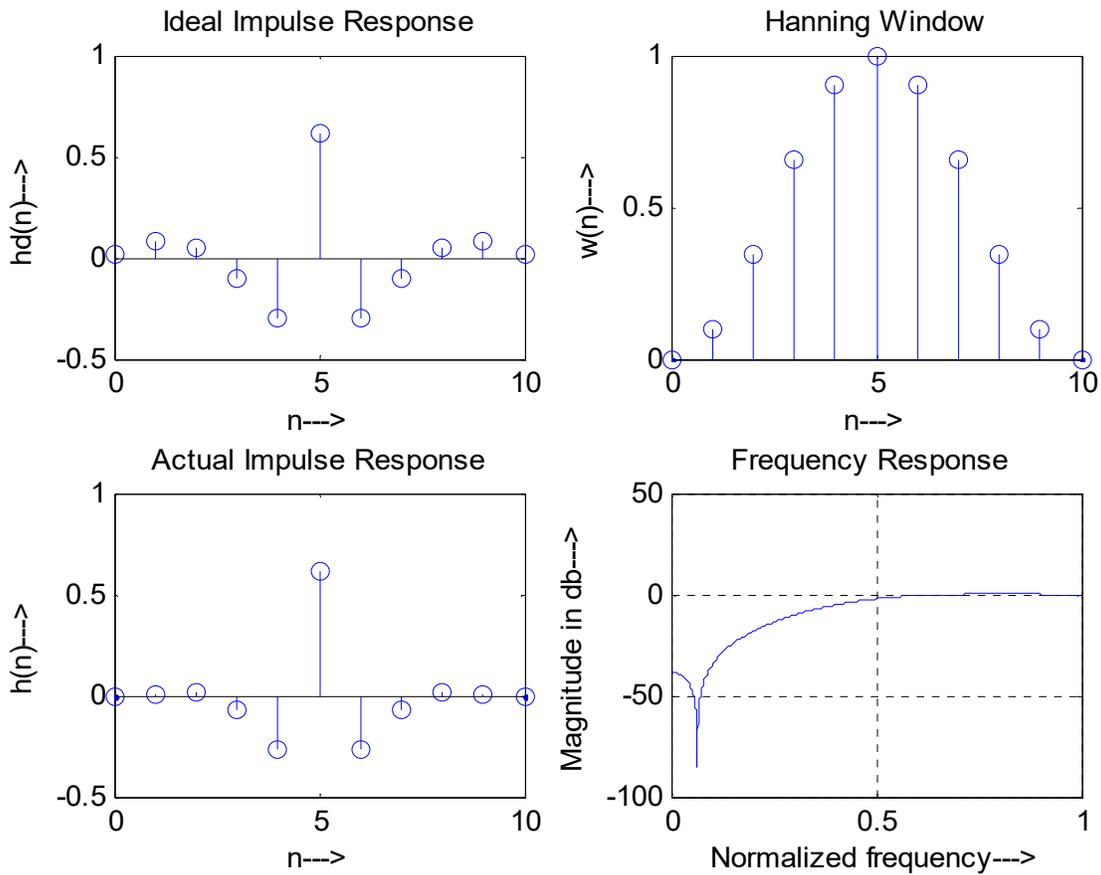
hd=0.0178 0.0793 0.0470 -0.1075 -0.2967 0.6180 -0.2967 -0.1075 0.0470  
0.0793 0.0178

w = 0 0.0955 0.3455 0.6545 0.9045 1.0000 0.9045 0.6545 0.3455 0.0955  
0

h = 0 0.0076 0.0162 -0.0704 -0.2683 0.6180 -0.2683 -0.0704 0.0162 0.0076  
0

**Model Graph:**

**HIGH PASS FILTER**



```

%PROGRAM FOR BAND PASS FILTER
    clc;
    clear all;
    close all;
    M=11;
    wc1=1.2;
    wc2=2;
    n=0:1:M-1;
    hd=ideal_lp(wc2,M)-ideal_lp(wc1,M);
    w=hann(M);
    h=hd.*w';
    disp('Filter coefficients for BPF=');

    [M1,P]=freqz(h,1,2000);
    mag=20*log10(abs(M1));

    subplot(2,2,1);
    stem(n,hd);
    xlabel('n--->');
    ylabel('hd(n)--->');
    title('Ideal Impulse Response');
    disp('hd=');
    disp(hd);

    subplot(2,2,2);
    stem(n,w);
    xlabel('n--->');
    ylabel('w(n)--->');
    title('Hanning Window');
    disp('w=');
    disp(w);

    subplot(2,2,3);
    stem(n,h);
    xlabel('n--->');
    ylabel('h(n)--->');
    title('Actual Impulse Response');
    disp('h=');
    disp(h);

    subplot(2,2,4);
    plot(P/pi, mag);
    grid on;
    xlabel('Normalized frequency--->');
    ylabel('Magnitude in db--->');
    title('Frequency Response ');

```

**Output:**

Filter coefficients for BPF=

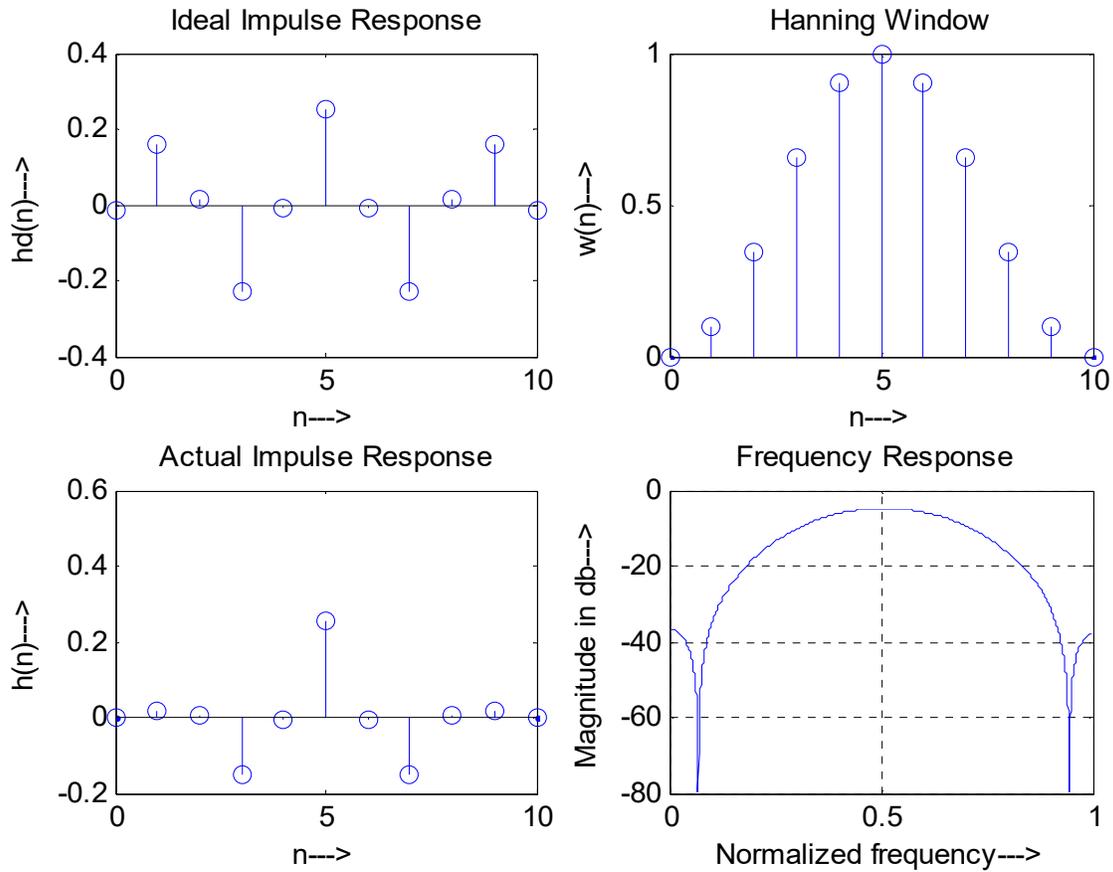
hd=-0.0168 0.1580 0.0173 -0.2280 -0.0072 0.2546 -0.0072 -0.2280 0.0173  
0.1580 -0.0168

w= 0 0.0955 0.3455 0.6545 0.9045 1.0000 0.9045 0.6545 0.3455 0.0955  
0

h= 0 0.0151 0.0060 -0.1492 -0.0065 0.2546 -0.0065 -0.1492 0.0060 0.0151  
0

**Model Graph:**

**BANDPASS FILTER**



```

%PROGRAM FOR BAND STOP FILTER
    clc;
    clear all;
    close all;
    M=7;
    wc1=1.2;
    wc2=2;
    n=0:1:M-1;
    hd=ideal_lp(pi,M)-ideal_lp(wc2,M)+ideal_lp(wc1,M);
    w=rectwin(M);
    h=hd.*w';
    disp('Filter coefficients for BSF=');

    [M1,P]=freqz(h,1,2000);
    mag=20*log10(abs(M1));

    subplot(2,2,1);
    stem(n,hd);
    xlabel('n--->');
    ylabel('hd(n)--->');
    title('Ideal Impulse Response');
    disp('hd=');
    disp(hd);

    subplot(2,2,2);
    stem(n,w);
    xlabel('n--->');
    ylabel('w(n)--->');
    title('Rectangular Window');
    disp('w=');
    disp(w);

    subplot(2,2,3);
    stem(n,h);
    xlabel('n--->');
    ylabel('h(n)--->');
    title('Actual Impulse Response');
    disp('h=');
    disp(h);

    subplot(2,2,4);
    plot(P/pi, mag);
    grid on;
    xlabel('Normalized frequency--->');
    ylabel('Magnitude in db--->');
    title('Frequency Response ');

```

**Output:**

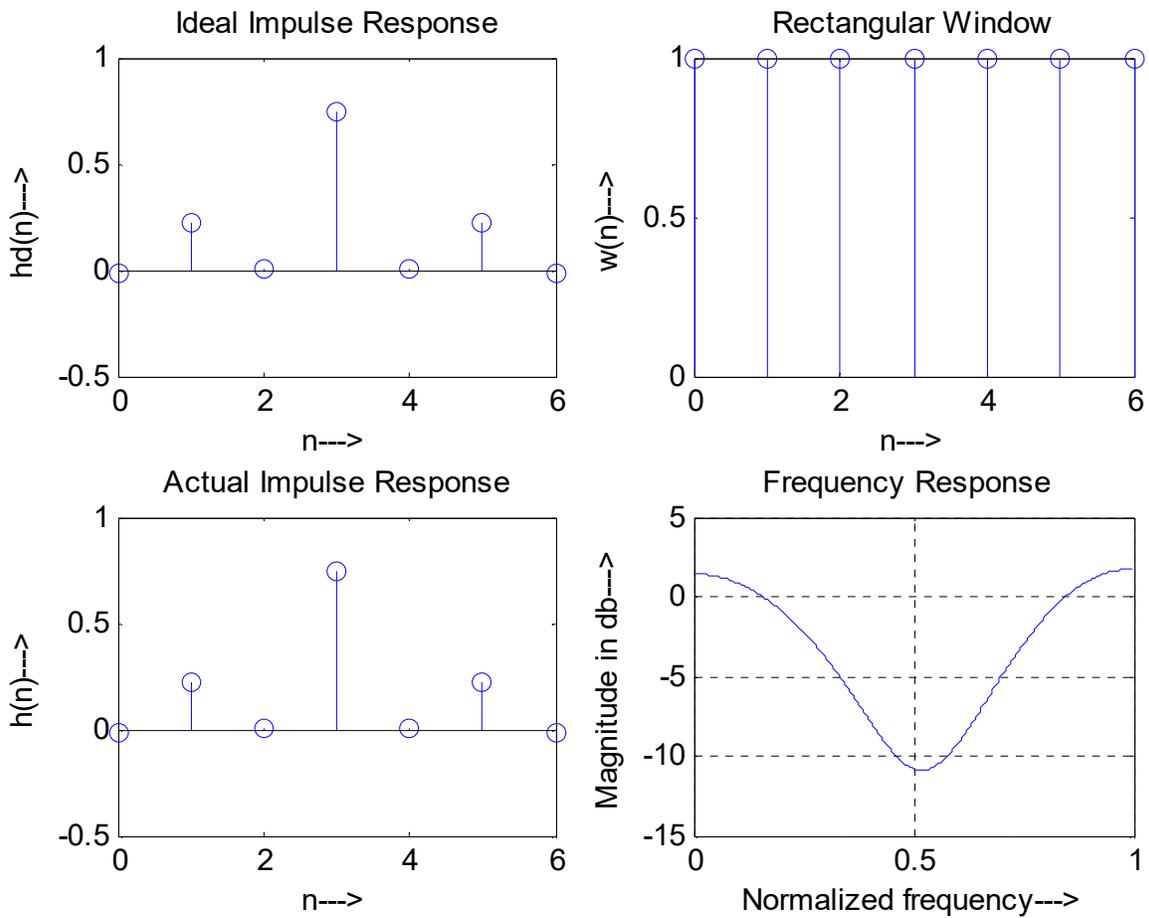
Filter coefficients for BSF=

hd= -0.0173 0.2280 0.0072 0.7454 0.0072 0.2280 -0.0173

w = 1 1 1 1 1 1 1

h = -0.0173 0.2280 0.0072 0.7454 0.0072 0.2280 -0.0173

**Model Graph:**  
**BANDSTOP FILTER**



**Windows:**

**Theoretical calculation:**

1.  $hd(n) = 1/2\pi \int Hd(w)e^{jwn}dw$
2. Select w(n)
3. Find  $h(n) = hd(n)w(n)$

### Rectangular Window

$$w(n) = \begin{cases} 1, & 0 \leq n \leq M-1 \\ 0, & \textit{otherwise} \end{cases}$$

### Hanning Window

$$w(n) = \begin{cases} 0.5 * \left[ 1 - \cos\left(\frac{2\pi n}{M-1}\right) \right], & 0 \leq n \leq M-1 \\ 0, & \textit{otherwise} \end{cases}$$

### Hamming Window

$$1. w(n) = \begin{cases} 0.54 - 0.46 * \cos\left(\frac{2\pi n}{M-1}\right), & 0 \leq n \leq M-1 \\ 0, & \textit{otherwise} \end{cases}$$

### Kaiser window

$$w(n) = \begin{cases} I_0\left(\beta\left(1 - \left(\frac{n-\alpha}{\alpha}\right)^2\right)^{1/2}\right) / I_0(\beta) & 0 \leq n \leq M-1 \\ 0 & \textit{otherwise} \end{cases}$$

Where  $\alpha = (M-1)/2$  and  $I_0 =$  the zeroth order Modified Bessel Function

If  $\omega_p$ ,  $\omega_s$ ,  $R_p$ , and  $A_s$  are given then the following equations are needed for design.

$$\text{Transition width} = \Delta f = \frac{\omega_s - \omega_p}{2\pi}$$

$$\text{Filter order } M \cong \frac{A_s - 7.95}{14.36\Delta f} + 1$$

$$\text{Parameter } \beta = \begin{cases} 0.1102(A_s - 8.7), & A_s \geq 50 \\ 0.5842(A_s - 21)^{0.4} + 0.07886(A_s - 21), & 21 > A_s > 50 \\ 0 & A_s < 21 \end{cases}$$

### **PostLab questions:**

1. What is meant by linear phase filters?
2. What is the condition for the impulse response of FIR filter to satisfy for constant and Phase delay and for only constant group delay?
3. What are Gibbs oscillations?
4. Give the equation for rectangular, hamming and Kaiser Windows.
5. State the condition for linear phase characteristics in FIR.

### **Result:**

Thus a program to design the FIR low pass, high pass, band pass and band stop filters was Written and response of the filter using MATLAB was executed.

## 6. DESIGN OF BUTTERWORTH AND CHEBYSHEV IIR FILTERS (LPF/HPF/BPF/BSF) AND DEMONSTRATE THE FILTERING OPERATIONS

**Aim:**

To write a program to design the Butterworth low pass, high pass filters and find out the response of the filter using MATLAB.

**Prerequisite Lab Questions:**

1. What are the requirements for an analog filter to be stable and causal?
2. What are the types of analog filters?
3. State the conditions to design a stable digital filters.
4. Write the relationship between analog and digital frequency in impulse invariant method?
5. Define Bilinear transformation.

**Description:**

**Input** : (a) Enter the pass band ripple and stop band attenuation  
(b) Enter the passband and stop band frequency

**Output** : (a) Find the order and cutoff frequency using bilinear and impulse Invariant transformation  
(b) Calculate the Analog filter coefficients H(s)  
(c) Transform the Analog filter to digital filter and find H (z)  
(d) Plot the responses for all the filters.

**Algorithm:**

1. Start the program.
2. Get the input sequences for pass band ripple, stop band ripple, pass band frequency, stop band frequency and sampling frequency.
3. Convert the frequency into radians/sec.
4. Compute the order and cutoff frequency of the filter and display it.
5. Design the analog filter and find the coefficient.
6. Convert the analog filter into digital using bilinear or impulse invariant transform.
7. Find out the frequency response of the filter.
8. Compute the magnitude and phase response of the filter.
9. Execute the program, display the result and verify it.
10. Plot the output graph for each sequence for low pass and high pass filters.
11. Stop the process.

**Program:**

```
% function to find order of butterworth filter using impulse invariant transformation
% Save as butt_impord
function [N,Wc]= butt_impord(w1,w2,a1,a2)
l=sqrt((1/a2^2)-1)
e=sqrt((1/a1^2)-1)
```

```

num=log10(1/e)
den=log10(w2/w1)
N=ceil((num/den))
Wc=(w1)/((e)^(1/N))
%function to find order of butterworth filter using bilinear transformationSave is as
butt_biord.
function [N1,Wc1]= butt_biord(w1,w2,a1,a2)
T=1;
wp1=(2/T)*tan(w1/2)
ws1=(2/T)*tan(w2/2)
lam=sqrt((1/a2^2)-1)
ep=sqrt((1/a1^2)-1)
num1=log10(lam/ep)
den1=log10(ws1/wp1)
N1=ceil((num1/den1))
Wc1=wp1/((ep)^(1/N1))

% Program for Butterworth Low Pass Filter
clc;
clear all;
close all;
%w1=0.2*pi;w2=0.6*pi;a1=0.8;a2=0.2;
w1=input('Enter the Passband frequency w1=');
w2=input('Enter the Stopband frequency w2=');
a1= input('Enter the Passband ripple a1=');
a2=input('Enter the Stopband ripple a2=');
[N,Wc]=butt_impord(w1,w2,a1,a2);
disp('LPF ANALOG FILTER TRANSFER FUNCTION');
[num,den]=butter(N,Wc,'s')
disp('LPF DIGITAL FILTER TRANSFER FUNCTION');
[b,a]=impinvar(num,den)
[mag,angle]=freqz(b,a,512);
magnitude=20*log10(abs(mag));
figure(1);
plot(angle/pi,magnitude);
grid on;
title('Low Pass Butterworth Filter');
xlabel('frequency --->');
ylabel('magnitude --->');

%Program for Butterworth High Pass Filter
[N1,Wc1]=butt_biord(w2,w1,a1,a2);
disp(' HPF ANALOG FILTER TRANSFER FUNCTION');
[num1,den1]=butter(N1,Wc1,'high','s')
disp('HPF DIGITAL FILTER TRANSFER FUNCTION');
[b1,a1]=bilinear(num1,den1,1)
[mag1,angle1]=freqz(b1,a1,512);
magnitude=20*log10(abs(mag1));
figure(2);

```

```

plot(angle1/pi,magnitude);
grid on;
title('High Pass Butterworth Filter');
xlabel('frequency --->');
ylabel('magnitude --->');

```

**Output:**

```

Enter the Passband frequency w1=[0.2*pi]
Enter the Stopband frequency w2=[0.6*pi]
Enter the Passband ripple a1=[0.8]
Enter the Stopband ripple a2=[0.2]

```

$l = 4.8990, e = 0.7500$

$num = 0.8150, den = 0.4771$

$N = 2, Wc = 0.7255$

LPF ANALOG FILTER TRANSFER FUNCTION

$num = 0 \quad 0 \quad 0.5264$

$den = 1.0000 \quad 1.0260 \quad 0.5264$

LPF DIGITAL FILTER TRANSFER FUNCTION

$b = 0 \quad 0.3015 \quad 0$

$a = 1.0000 \quad -1.0432 \quad 0.3584$

$T = 1, wp1 = 0.6498$

$ws1 = 2.7528, lam = 4.8990$

$ep = 0.7500$

$num1 = 0.8150, den1 = 0.6270$

$N1 = 2$

$Wc1 = 0.7504$

HPF ANALOG FILTER TRANSFER FUNCTION

$num1 = 1 \quad 0 \quad 0$

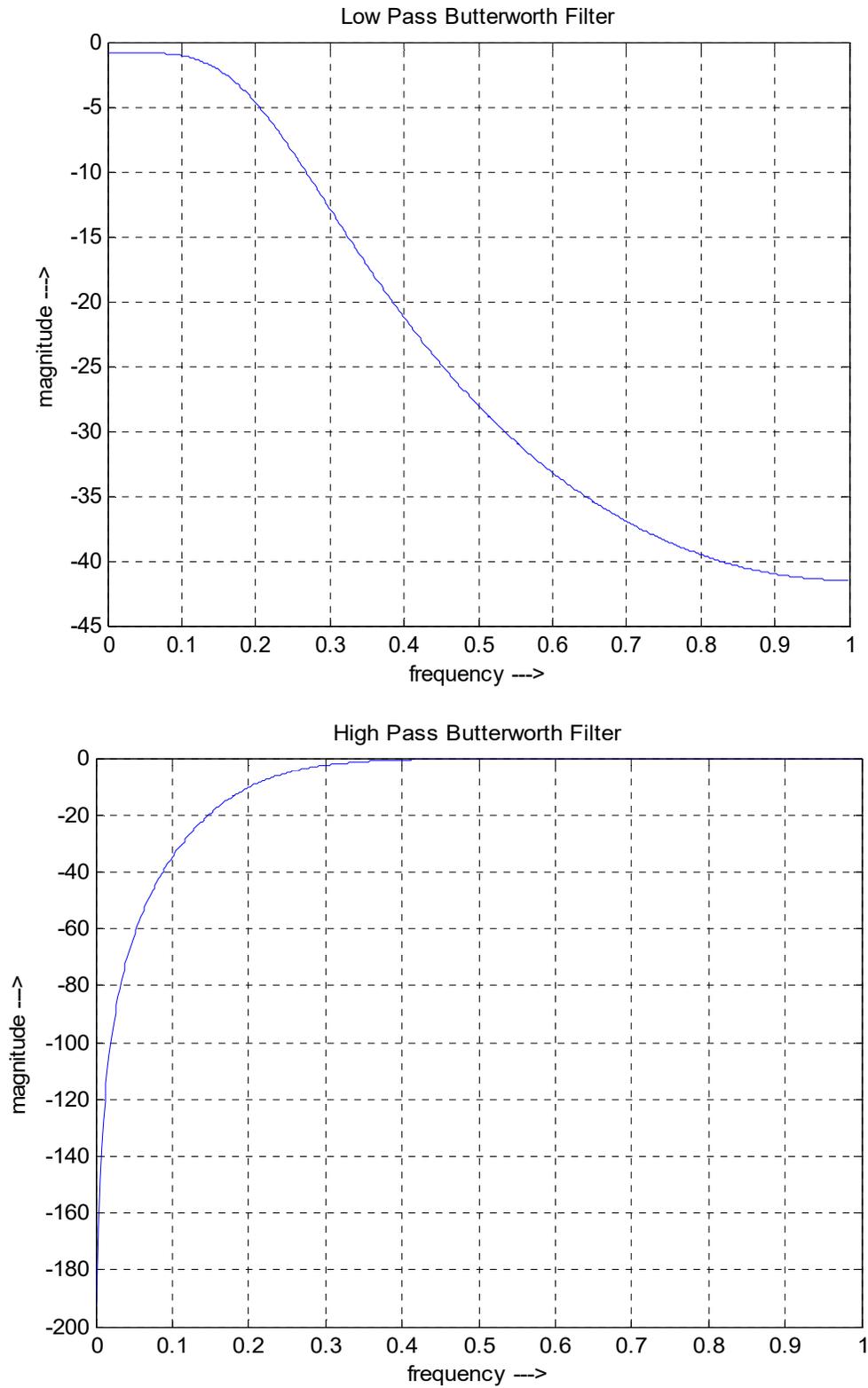
$den1 = 1.0000 \quad 1.0612 \quad 0.5631$

HPF DIGITAL FILTER TRANSFER FUNCTION

b1 = 0.5983 -1.1966 0.5983

a1 = 1.0000 -1.0282 0.3651

### Model graph



## Theoretical calculation

### Bilinear Transformation

$$wp1=(2/T)*\tan(w1/2)$$

$$ws1=(2/T)*\tan(w2/2)$$

$$lam=\sqrt{(1/a^2)-1}$$

$$ep=\sqrt{(1/a1^2)-1}$$

$$N=\log_{10}(lam/ep) / \log_{10}(ws1/wp1);$$

$$Wc1=wp1/((ep)^{(1/N)});$$

$$\text{N-Even} \quad H(s) = \prod_{k=1}^{\frac{N}{2}} \frac{\Omega_c^2}{s^2 + 2\Omega_c b_k s + \Omega_c^2}; \dots k = 1, 2, \dots, \frac{N}{2}$$

$$\text{N-Odd} \quad H(s) = \frac{\Omega_c}{s + \Omega_c} \prod_{k=1}^{\frac{N}{2} - 1} \frac{\Omega_c^2}{s^2 + 2\Omega_c b_k s + \Omega_c^2}; \dots k = 1, 2, \dots, \frac{N}{2} - 1$$

$$b_k = 2 \sin(2k-1)*\pi/4$$

To find H (Z) replace 's' by

$$S = \frac{2(Z-1)}{T(Z+1)}$$

$$T(Z+1)$$

### Impulse Invariant Transformation

$$l=\sqrt{(1/a^2)-1}$$

$$e=\sqrt{(1/a1^2)-1}$$

$$N=\log_{10}(l/e) / \log_{10}(w2/w1); ;$$

$$Wc=(w1)/((e)^{(1/N)});$$

H(S) is same

$$H_c(s) = \sum_{k=1}^N \frac{A_k}{s - s_k} H(z) = \sum_{k=1}^N \frac{A_k}{1 - e^{s_k T} z^{-1}}$$

**PostLab questions:**

1. What are the properties of Butterworth low pass filters?
2. Give the magnitude function of Butterworth filter.
3. What is meant by conformal mapping?
4. Define warping effect?
5. What are the advantages and disadvantages of bilinear transformations?

**Result:**

Thus a program to design the Butterworth low pass and high pass filter using Matlab was written and response of the filter was executed.

**Ex : No:**

**Date:**

## **7. STUDY OF ARCHITECTURE OF DIGITAL SIGNAL PROCESSOR**

**Aim:**

To Study the architecture of TMS320C5416 Digital Signal Processor Starter Kit (DSK).

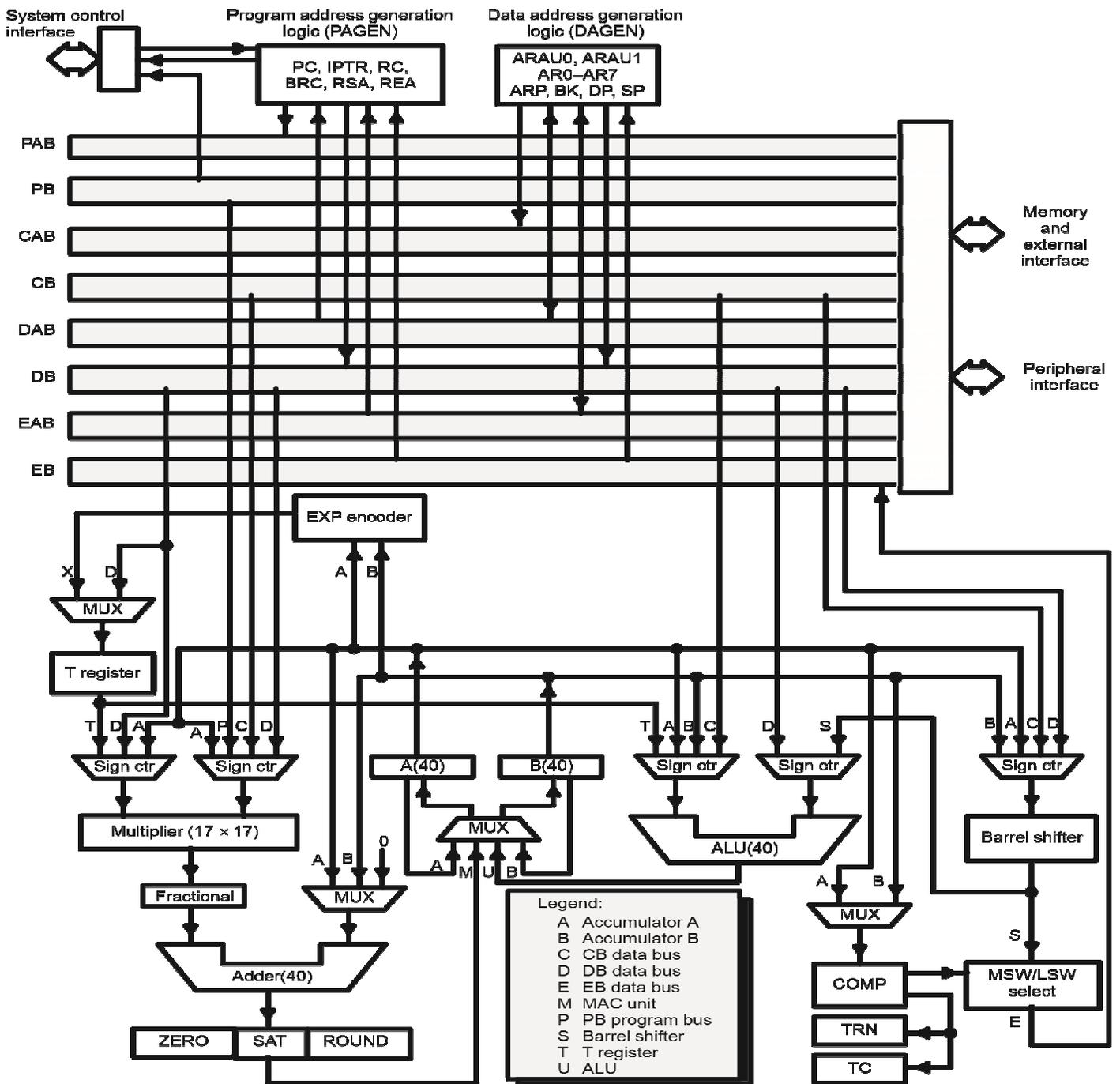
**Description:**

The TMS320VC5416 fixed-point, digital signal processor (DSP) (hereafter referred to as the device unless otherwise specified) is based on an advanced modified Harvard architecture that has one programMemory bus and three data memory buses. This processor provides an arithmetic logic unit (ALU) with aHigh degree of parallelism, application-specific hardware logic, on-chip memory, and additional on-chipPeripherals. The basis of the operational flexibility and speed of this DSP is a highly specialized instructionSet. Separate program and data spaces allow simultaneous access to program instructions and data, providing, a high degree of parallelism. Two read operations and one write operation can be performed in a single cycle. Instructions with parallel store and application-specific instructions can fully utilize this architecture. In addition, data can be transferred between data and program spaces. Such parallelism supports a powerful set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. The device also includes the control mechanisms to manage interrupts, repeated Operations, and function calls.

**Architecture:**

The “54x DSPs use an advanced, modified Harvard architecture that maximizes processing power by maintaining one program memory bus and three data memory buses. These processors also provide an arithmetic logic unit (ALU) that has a high degree of parallelism, application-specific hardware logic, on-chip memory, and additional on-chip peripherals. These DSP families also provide a highly specialized instruction set, which is the basis of the operational flexibility and speed of these DSPs. Separate program and data spaces allow simultaneous access to program instructions and data, providing the high degree of parallelism. Two reads and one write operation can be performed in a single cycle. Instructions with parallel store and application-specific instructions can fully utilize this architecture. In addition, data can be transferred between data and program spaces. Such parallelism supports a powerful set of arithmetic, logic, and bit-manipulation operations that can all be performed in a single machine cycle. Also included are the control mechanisms to manage interrupts, repeated operations, and function calls.

# Architecture of TMS320VC5416 Fixed-Point Digital Signal Processor:



## Central Processing Unit (CPU)

The CPU is common to all C54x™ devices. The C54x CPU contains:

- 40-bit arithmetic logic unit (ALU)
- Two 40-bit accumulators
- Barrel shifter
- 17 × 17-bit multiplier
- 40-bit adder
- Compare, select, and store unit (CSSU)
- Data address generation unit
- Program address generation unit

### Arithmetic Logic Unit (ALU)

The C54x DSP performs 2s-complement arithmetic with a 40-bit arithmetic logic unit (ALU) and two 40-bit accumulators (accumulators A and B). The ALU can also perform Boolean operations. The ALU uses these inputs:

- 16-bit immediate value
- 16-bit word from data memory
- 16-bit value in the temporary register, T
- Two 16-bit words from data memory
- 32-bit word from data memory
- 40-bit word from either accumulator

The ALU can also function as two 16-bit ALUs and perform two 16-bit operations simultaneously.

### Accumulators

Accumulators A and B store the output from the ALU or the multiplier/adder block. They can also provide a second input to the ALU; accumulator A can be an input to the multiplier/adder. Each accumulator is divided into three parts:

- Guard bits (bits 39–32)
- High-order word (bits 31–16)
- Low-order word (bits 15–0)

Instructions are provided for storing the guard bits, for storing the high- and the low-order accumulator words in data memory, and for transferring 32-bit accumulator words in or out of data memory. Also, either of the accumulators can be used as temporary storage.

### Barrel Shifter

The C54x DSP barrel shifter has a 40-bit input connected to the accumulators or to data memory (using CB or DB), and a 40-bit output connected to the ALU or to data memory (using EB). The barrel shifter can produce a left shift of 0 to 31 bits and a right shift of 0 to 16 bits on the input data. The shift requirements are defined in the shift count field of the instruction, the shift count field (ASM) of status register ST1, or in temporary register T (when it is designated as a shift count register).

The barrel shifter and the exponent encoder normalize the values in an accumulator in a single cycle. The LSBs of the output are filled with 0s, and the MSBs can be either zero filled or sign extended, depending on the state of the sign-extension mode bit (SXM) in ST1. Additional shift capabilities enable the processor to perform numerical scaling, bit extraction, extended arithmetic, and overflow prevention operations.

### **Multiplier/Adder Unit**

The multiplier/adder unit performs 17 17-bit 2s-complement multiplication with a 40-bit addition in a single instruction cycle. The multiplier/adder block consists of several elements: a multiplier, an adder, signed/unsigned input control logic, fractional control logic, a zero detector, a rounder (2s complement), overflow/saturation logic, and a 16-bit temporary storage register (T). The multiplier has two inputs: one input is selected from T, a data-memory operand, or accumulator A; the other is selected from program memory, data memory, accumulator A, or an immediate value.

The fast, on-chip multiplier allows the C54x DSP to perform operations efficiently such as convolution, correlation, and filtering. In addition, the multiplier and ALU together execute multiply/accumulate (MAC) computations and ALU operations in parallel in a single instruction cycle. This function is used in determining the Euclidian distance and in implementing symmetrical and LMS filters, which are required for complex DSP algorithms.

### **Compare, Select, and Store Unit (CSSU)**

The compare, select, and store unit (CSSU) performs maximum comparisons between the accumulator's high and low word, allows both the test/control flag bit (TC) in status register ST0 and the transition register (TRN) to keep their transition histories, and selects the larger word in the accumulator to store into data memory. The CSSU also accelerates Viterbi-type butterfly computations with optimized on-chip hardware.

### **Status Registers (ST0, ST1)**

The status registers ST0 and ST1 contain the status of the various conditions and modes for the C54x devices. ST0 contains the flags (OVA, OVB, C, and TC) produced by arithmetic operations and bit manipulations, in addition to the DP and the ARP fields. ST1 reflects the status of modes and instructions executed by the processor.

### **Auxiliary Registers (AR0–AR7)**

The eight 16-bit auxiliary registers (AR0–AR7) can be accessed by the CPU and modified by the auxiliary register arithmetic units (ARAUs). The primary function of the auxiliary registers is to generate 16-bit addresses for data space. However, these registers can also act as general-purpose registers or counters.

### **Transition Register (TRN)**

The 16-bit transition (TRN) register holds the transition decision for the path to new metrics to perform the Viterbi algorithm. The CMPS (compare select max and store) instruction updates the contents of TRN register on the basis of the comparison between the accumulator high word and the accumulator low word.

### **Temporary Register (T)**

The temporary (T) register has many uses. For example, it may hold:

One of the multiplicands for multiply and multiply/accumulate instructions (For more details about the T register and the processes of multiplication.

A dynamic (execution-time programmable) shift count for instructions with shift operation such as the ADD, LD, and SUB instructions

A dynamic bit address for the BITT instruction

Branch metrics used by the DADST and DSADT instructions for ACS operation of Viterbi decoding

In addition, the EXP instruction stores the exponent value computed into T register, and then the NORM instruction uses the T register value to normalize the number.

### **Stack-Pointer Register (SP)**

The 16-bit stack-pointer register (SP) contains the address of the top of the system stack. The SP always points to the last element pushed onto the stack. The stack is manipulated by interrupts, traps, calls, returns, and the PSHD, PSHM, POPD, and POPM instructions. Pushes and pops of the stack predecrement and postincrement, respectively, the 16-bit value in the stack pointer.

### **Circular-Buffer Size Register (BK)**

The ARAUs use 16-bit circular-buffer size register (BK) in circular addressing to specify the data block size. For information on BK and circular addressing.

### **Block-Repeat Registers (BRC, RSA, REA)**

The 16-bit block-repeat counter (BRC) register specifies the number of times a block of code is to repeat when a block repeat is performed. The 16-bit block-repeat start address (RSA) register contains the starting address of the block of program memory to be repeated. The 16-bit block-repeat end address (REA) register contains the ending address of the block of program memory to be repeated.

### **Memory-Mapped Registers**

The 64K words of data memory space include the device's memory-mapped registers, which reside in data page 0 (data addresses 0000h–007Fh).

The peripheral registers are used as control and data registers in peripheral circuits. These registers reside within addresses 0020h–005F and reside on a dedicated peripheral bus structure. For a list of peripherals on a particular C54x device.

The scratch-pad RAM block (60h–7Fh in data memory) includes 32 words of DARAM for variable storage that helps avoid fragmenting the large RAM block. The data memory space contains memory-mapped registers for the CPU and the on-chip peripherals. Simplifying access to them. The memory-mapped access provides a convenient way to save and restore the registers for context switches and to transfer information between the accumulators and the other registers.

### **Data Memory**

The data memory space addresses up to 64K of 16-bit words. The device automatically accesses the on-chip RAM when addressing within its bounds. When an address is generated outside the RAM bounds, the device automatically generates an external access.

The advantages of operating from on-chip memory are as follows:

- Higher performance because no wait states are required
- Higher performance because of better flow within the pipeline of the central arithmetic logic unit (CALU)

- Lower cost than external memory
- Lower power than external memory

The advantage of operating from off-chip memory is the ability to access a larger address space.

### **Program Memory**

Software can configure their memory cells to reside inside or outside of the program address map. When the cells are mapped into program space, the device automatically accesses them when their addresses are within bounds. When the program-address generation (PAGEN) logic generates an address outside its bounds, the device automatically generates an external access.

The advantages of operating from on-chip memory are as follows:

- Higher performance because no wait states are required
- Lower cost than external memory
- Lower power than external memory

The advantage of operating from off-chip memory is the ability to access a larger address space.

### **Overview of the TMS320VC5416 DSK**

The TMS320VC5416 DSK is a stand-alone development and evaluation module. It allows evaluators to examine certain characteristics of the C5416 digital signal processor (DSP) to determine if it meets their application requirements. Furthermore, the module is an excellent platform to develop and run software for the TMS320VC5416 family of processors.

The DSK allows full speed verification of VC5416 code. With 64K words of on board RAM memory, 256K words of on board Flash ROM, and a Burr Brown PCM 3002 stereo codec, the board can solve a variety of problems as shipped. Three expansion connectors are provided for interfacing to evaluation circuitry not provided on the as shipped configuration.

To simplify code development and shorten debugging time, a special version of Code Composer Studio is shipped with the board.

### **Key Features of the TMS320VC5416 DSK**

The VC5416 DSK has the following features:

- VC5416 operating at 16-160 MHz
- On board USB JTAG controller with plug and play drivers
- 64K words of on board RAM
- 256K words of on board Flash ROM
- 3 Expansion Connectors (Memory Interface, Peripheral Interface, and Host Port Interface)
- On board IEEE 1149.1 JTAG Connection for Optional Emulation Debug
- Burr Brown PCM 3002 Stereo Codec
- +5 volt operation

## **Code Composer Studio - 3.1v**

### **INTRODUCTION TO CODE COMPOSER STUDIO**

Code Composer is the DSP industry's first fully integrated development environment (IDE) with DSP-specific functionality. With a familiar environment like MS-based C++™, Code Composer lets you edit, build, debug, profile and manage projects from a single unified environment. Other unique features include graphical signal analysis, injection/extraction of data

signals via file I/O, multi-processor debugging, automated testing and customization via a C-interpretive scripting language and much more.

### **Initial setup to work on Code Composer Studio:**

- To execute using Simulator OpenSetup CCStudio\_v3.1 add C5416 Device Simulator Save and Quit.

(OR)

- To Execute using DSP Processor Kit ->DSK Diagnostic Utility->Start  
Check for PASS in the display and continue the Procedure
- Select 5416 dskCCStudio\_v3.1->Debug->Connect

### **Procedure to work on Code Composer Studio:**

#### **To create the New Project**

Project → New → (C:\CCStudio\_v3.1\MyProjects\projectname.pjt)

#### **To Create a Source file**

File → New → Type the code (Save & give file name, Eg: filename.cor  
filename.asm).  
(C:\CCStudio\_v3.1\MyProjects\projectname.pjt\filename.c)

#### **To Add Source files to Project**

Project → Add files to Project → filename.c

#### **To add library & cmd:**

Project → Add files to Project → rts\_EXT.lib  
Library files: (Path: C:\CCStudio\C5400\cgtools\lib\rts\_EXT.lib)  
Note: Select Object & Library from all files  
Project → Add files to Project → c5416dsk.cmd  
CMD file – Which is common for all non-real time programs.  
(Path: Path: C:\CCStudio\tutorial\dsk5416\shared\c5416dsk.cmd)  
Note: Select Linker Command file (\*.cmd) in Type of files

#### **To Compile:**

Project → Compile

#### **To Build:**

Project → Build

If there is more number of warnings after build, then

Project → Build Options → Advanced (in Category) → Use Far Calls (- mf)

#### **Procedure to Load and Run program:**

Load the program to DSK: **File** → Load program → project name. Out  
(C:\CCStudio\_v3.1\MyProjects\projectname.pjt\Debug\projectname.out)

**To execute project:**

Debug → Run.

**To Plot Graph:**

View → Graph (OR Right click in the program → Insert Graph)  
Select proper Properties and specify the address.

**Result:**

Thus the architecture of TMS320C5416 Digital Signal Processor Starter Kit (DSK) and programming steps for Code Composer Studio was studied.

**Ex : No:**

**Date:**

## **8. PERFORM MAC OPERATION USING VARIOUS ADDRESSING MODES**

### **Aim:**

To perform MAC operation using various addressing modes of TMS320C5416 DSP processor.

### **Prerequisite Lab Questions:**

1. What are the classification of DSP Processors?
2. Compare fixed and floating point Processors ?
3. Brief the different types of addressing mode present in the TMS320C5416?
4. Define program counter.
5. What is the architectural difference between conventional and DSP processors?

### **Algorithm:**

1. Start the program.
2. Store the first number from X memory to AR2.
3. Store the second number from Y memory to AR3.
4. Store the immediate address to AR4 to store the output.
5. Add both the numbers and store the result in accumulator A.
6. Store the lower order word to the memory location 1500 and higher order word to 1501 and increment the memory location
7. Load the content pointed by AR2 to the temporary register
8. Multiply the Content pointed by AR2 with the temporary register and store the result in B register.
9. Store the lower order word and higher order word of B register to the memory location pointed by AR4
10. Stop the program execution.

### **Program:**

```
; Program for Addition

                .global _main
                .mmregs
                .data
X:              .word 50h
Y:              .word 20h
                .text

_main
                STM X,AR2
                STM Y,AR3
                STM #1500h,AR4
                ADD *AR2,*AR3,A
                STL A,*AR4+
                STH A,*AR4
                .end
```

**Output:**

0x1500: 0x0000  
0x1501: 0x0070

```
; Program for Subtraction

                                .global _main
                                .mmregs
                                .data
X:                                .word 60h
Y:                                .word 20h
                                .text
_main
                                STM X,AR2
                                STM Y,AR3
                                STM #1500h,AR4
                                SUB *AR2,*AR3,A
                                STL A,*AR4+
                                STH A,*AR4
                                .end
```

**Output:**

0x1500: 0x0000  
0x1501: 0x0040

```
; Program for Multiplication

                                .global _main
                                .mmregs
                                .data
X:                                .word 30h
Y:                                .word 20h
                                .text
_main
                                STM X,AR2
                                STM Y,AR3
                                STM #1500h,AR4
                                MPY *AR2,*AR3,A
                                STL A,*AR4+
                                STH A,*AR4
                                .end
```

**Output:**

0x1500: 0x0600

```

; Program for Division

                .global _main
                .mmregs
                .data
X:              .word 90h
Y:              .word 30h
                .text

_main
                STM X,AR2
                STM Y,AR3
                STM #1500h,AR4
                LD *AR2+,A
                RPT #15
                SUBC *AR3,A
                STL A,*AR4+
                STH A,*AR4
                .end

```

SUBC performs binary division like long division. For 16-bit by 16-bit integer division, the dividend is stored in low part accumulator A. The program repeats the SUBC command 16 times to produce a 16-bit quotient in low part accumulator A and a 16-bit remainder in high part accumulator B. For each SUBC subtraction that results in a negative answer, you must left-shift the accumulator by 1 bit. This corresponds to putting a 0 in the quotient when the divisor does not go into the dividend. For each subtraction that produces a positive answer, you must left shift the difference in the ALU output by 1 bit, add 1, and store the result in accumulator A. This corresponds to putting a 1 in the quotient when the divisor goes into the dividend.

**Output:**

0x1500: 0x0003

**PostLab Questions:**

1. Give an example for immediate and indirect addressing modes?
2. What do you mean by bit reversal addressing mode?
3. What are the registers available for indirect addressing mode?
4. What are barrel shifters?
5. What is arithmetic and logical shift in DSP Processors?

**Result:**

Thus the assembly coding for MAC operation using various addressing modes of DSP processor TMS320C5416 was written and executed.

Ex : No:

Date:

## 9. GENERATION OF VARIOUS SIGNALS AND RANDOM NOISE

### Aim:

To write a program to generate Sine wave and Ramp wave form using TMS320C5416 DSP Processor.

### Algorithm:

1. Start the program.
2. Declare the output buffer to store the result.
3. Assign 0 to accumulator
4. Increment the variable by one and store the value in the output buffer
5. After the counter is incremented to 40, then decrement the value of the variable one by one from a negative going ramp
6. Again decrement the value from a negative going ramp in negative scale until the value becomes maximum negative value.
7. Decrement the maximum negative value one by one until the value of the variable becomes zero.

### **Program:**

; Program for Triangular Wave Generation

```
.global _main
.mmregs
.def start
.data
.text

_main

start          STM #2000h,AR4
               LD #0h,0,A

BEGIN          STM #40,BRC
               RPTB RAMP
               STL A,*AR4+
               LD #1h,0,B

RAMP           ADD B,0,A
               STM #40,BRC
               RPTB RAMP1
               STL A,*AR4+
               LD #1h,0,B
```

```

RAMP1      SUB B,0,A
           STM #40,BRC
           RPTB RAMP2
           STL A,*AR4+
           LD #1h,0,B

RAMP2      SUB B,0,A
           STM #40,BRC
           RPTB RAMP3
           STL A,*AR4+
           LD #1h,0,B

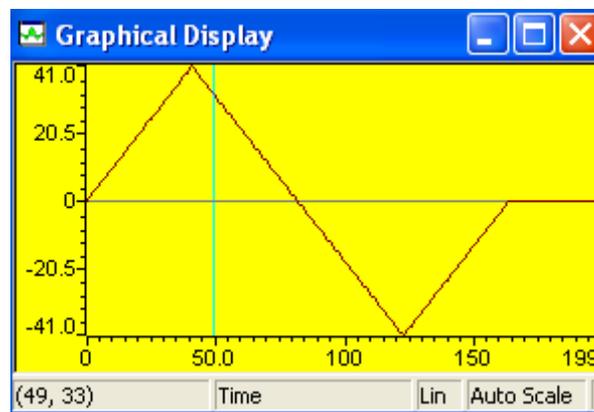
RAMP3      ADD B,0,A
           STL A,*AR4+
           LD #1h,0,A
           SUB #1h,0,A
           BC BEGIN,ANEQ
           .end

```

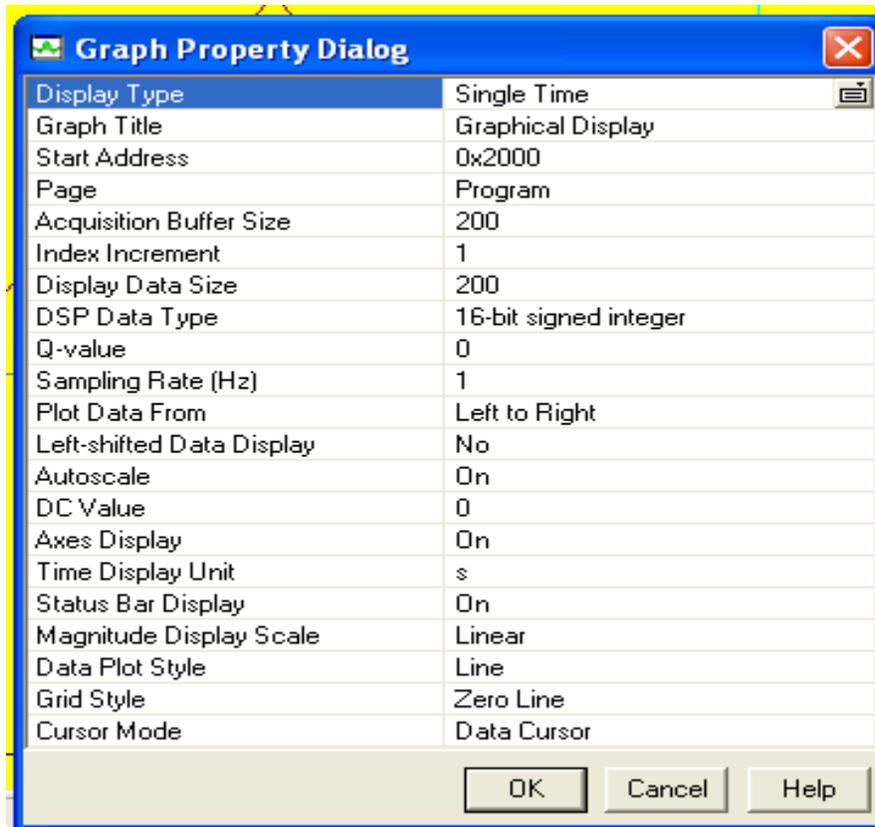
The repeat-block instructions are used to repeat a block of code  $N + 1$  times, where  $N$  is the value loaded into the block-repeat counter register (BRC).

BRC contains the value  $N$ , which is one less than the number of times the block is to be repeated. The block-repeat start address register (RSA) holds the address of the first instruction of the block of code to be repeated. The block-repeat end address register (REA) holds the address of the last instruction word of the block of code to be repeat

### Model Graph:



## Graph Property:



## Program:

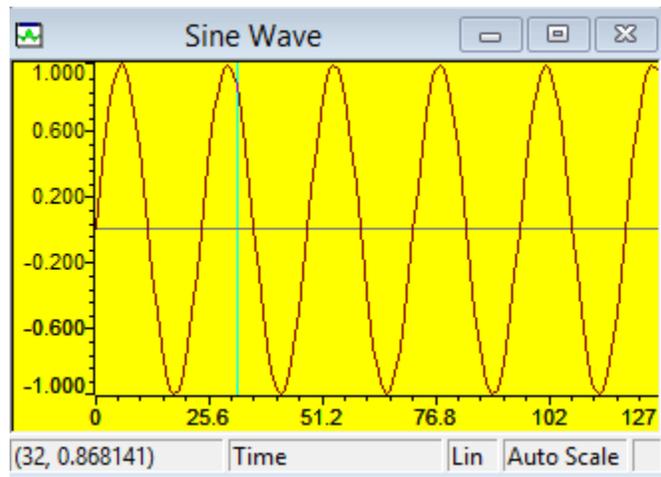
```
//Program for Sine Wave Generation

#include <stdio.h>
#include<math.h>
#define FREQ 1000

float a[128];
main()
{
    int i=0;

    for(i=0;i<128;i++)
    {
        a[i]=sin(2*3.14*FREQ*i/24000);
        printf("%f",a[i]);
    }
}
```

## Model Graph:



## Graph Property:

Property	Value
Display Type	Single Time
Graph Title	Sine Wave
Start Address	a
Page	Data
Acquisition Buffer Size	128
Index Increment	1
Display Data Size	128
DSP Data Type	32-bit floating point
Sampling Rate (Hz)	1
Plot Data From	Left to Right
Left-shifted Data Display	No
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	s
Status Bar Display	On
Magnitude Display Scale	Linear
Data Plot Style	Line
Grid Style	Zero Line
Cursor Mode	Data Cursor

## Result:

Thus the program was written to generate a Ramp signal, Sine signal and executed using DSP processor.

Ex : No:

Date:

## 10. IMPLEMENT AN UP-SAMPLING AND DOWN-SAMPLING OPERATION IN DSP PROCESSOR

**Aim:**

Write a program to perform the decimation and interpolation on the given input sequence operation using Matlab.

**Prerequisite Lab Questions:**

1. What are the need for multirate signal processing?
2. Define Decimator and Interpolator.
3. How aliasing effect is overcome in decimation?
4. Write the time domain relationship for upsampling and downsampling.
5. List the advantages of MDSP.

**Description:**

**Input:** Give the input signal

**Output:** Find the decimated and interpolated output signal for the given number of samples.

**Algorithm:**

1. Start the program.
2. Get the amplitude of the input signal.
3. Get the decimation and interpolation factor.
4. Perform the decimation and interpolation operations using Matlab functions.
5. Plot the output signals.
6. Stop the process.

**Program:**

```
%Program for Decimate signal

clc;
clear all;
close all;
t=0:0.00025:1; % Time vector
x=sin(2*pi*30*t)+sin(2*pi*60*t);
r=4; % Factor of 4
y=decimate(x,r);

%View the original signal
figure(1);
subplot(2,1,1);
stem(x(1:120));
xlabel('n--->');
```

```

ylabel('Amplitude--->');
title('Original Signal');

%View the decimated signal
subplot(2,1,2);
stem(y(1:30));
xlabel('n--->');
ylabel('Amplitude--->');
title('Decimated Signal');

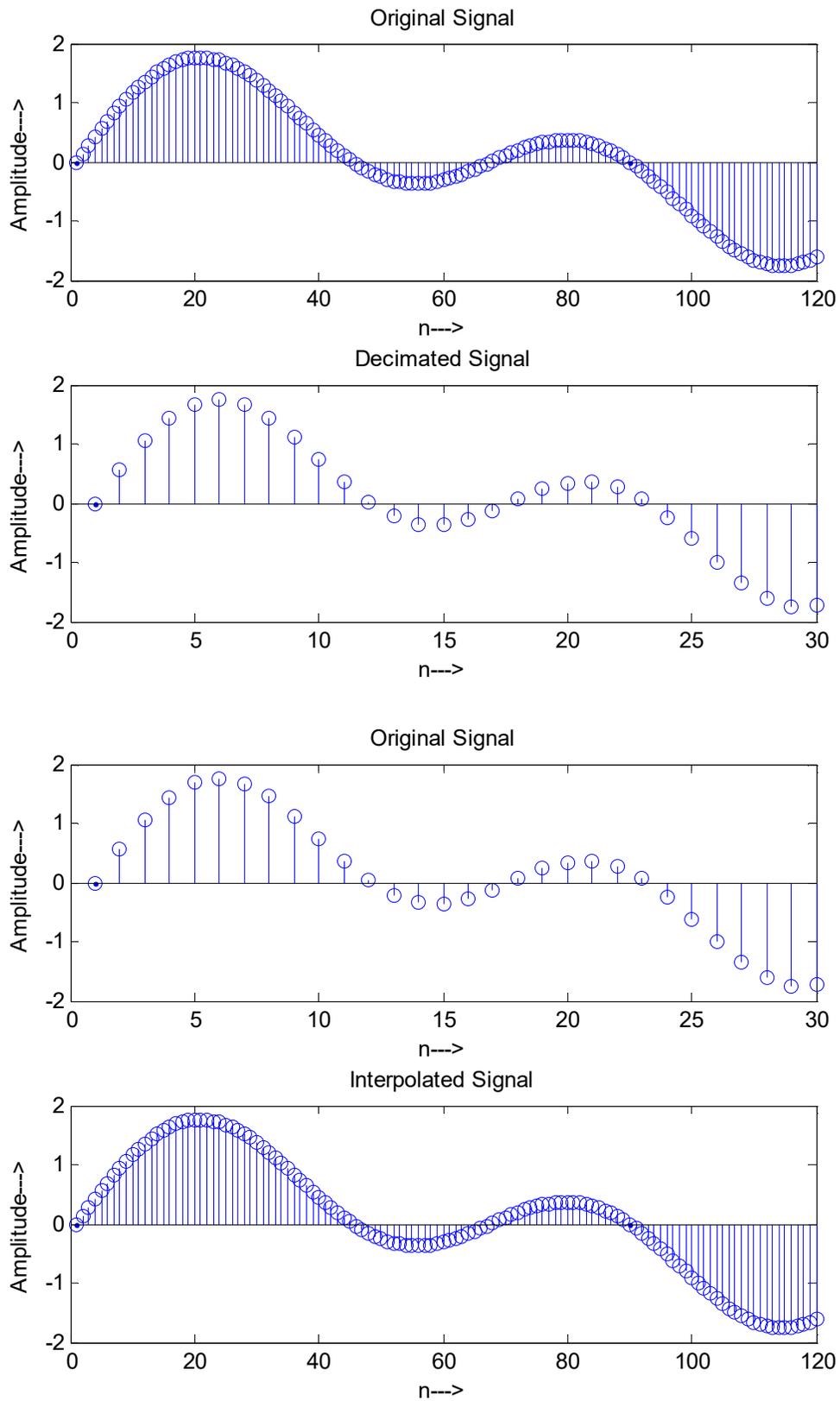
%Program for Interpolated signal

t=0:0.001:1; % Time vector
x=sin(2*pi*30*t)+sin(2*pi*60*t);
r=4; % Factor of 4
y=interp(x,r);
%View the Original signal
figure(2);
subplot(2,1,1);
stem(x(1:30));
xlabel('n--->');
ylabel('Amplitude--->');
title('Original Signal');

%View the Interpolated Signal
subplot(2,1,2);
stem(y(1:120));
xlabel('n--->');
ylabel('Amplitude--->');
title('Interpolated Signal');

```

**Model graph:**



**PostLab Questions:**

1. What are the effects of upsampling?
2. List the applications of multirate signal processing?
3. What are sampling rate converters?
4. Define sub band coding of signals.
5. For the given signal  $x[n] = \{1,3,4,5,6,7,8,9,1,-2\}$  . Decimate and interpolate by a factor of 2.

**Result:**

Thus a program to perform decimation and interpolation the input signal was written using MATLAB and plotted.