THANGAL KUNJU MUSALIAR COLLEGE OF ENGINEERING

KOLLAM – 691 005

ELECTRONICS AND COMMUNICATION ENGINEERING

LABORATORY RECORD

YEAR 2024-25

Certified that this is a Bonafide Record of the work done by Smt. [SANDHYA M] of ***5th*** *Semester class (Roll No.* ***[B22ECB62] Electronics and Communication*** *Branch) in the* ***Digital Signal Processing*** *Laboratory during the year* ***2024-25***

Name of the Examination: ***Fifth Semester B.Tech Degree Examination 2024***

Register Number : ***[TKM22EC118]***

Staff Member in-charge *External Examiner*

Date:

INDEX

SL No.	DATE	NAME OF THE EXPERIMENTS	PAGE NO.	REMARKS
1	25/08/2024	Simulation of Basic Test Signals		
2	01/08/2024	Verification of Sampling Theorem		
3	08/08/2024	Linear Convolution		
4	15/08/2024	Circular Convolution		
5	22/08/2024	Linear Convolution using circular convulation and vice versa		
6	01/10/24	DFT and IDFT		
7	01/10/24	Properties of DFT		
8	08/10/24	Overlap Add and Overlap Save Method		
9	14/09/24	Familiarization of TMS320C6748		
10	14/10/24	Generation of Sine wave using DSP Processor		
11	14/10/24	Linear Convolution using DSP Processor		
12	19/10/24	FIR Filters		

Experiment No: **1** Date: 29/07/24

Experiment Name: Simulation of Basic Test Signals

Aim:

To generate continuous and discrete waveforms for the following:

1. Unit Impulse Signal
2. Bipolar Pulse Signal
3. Unipolar Pulse Signal
4. Ramp Signal
5. Triangular Signal
6. Sine Signal
7. Cosine Signal
8. Exponential Signal
9. Unit Step Signal

Theory:

1. Unit Impulse Signal:

- A signal that is zero everywhere except at one point, typically at t=0 where its value is 1.
- **Mathematically** $\delta(t) = \begin{cases} \infty; & t = 0 \\ 0; & t \neq 0 \end{cases}$

2. Bipolar Pulse Signal:

- A pulse signal that alternates between positive and negative values, usually rectangular in shape. It switches between two constant levels (e.g., -1 and 1) for a defined duration.
- **Mathematically $p(t) = A$ for $|t| \leq \tau/2$, $p(t) = 0$ otherwise**

3. Unipolar Pulse Signal:

- A pulse signal that alternates between zero and a positive value. It remains at zero for a specified duration and then jumps to a positive constant level (e.g., 0 and 1).
- **Mathematically $p(t) = A$ for $|t| \leq \tau/2$, $p(t) = 0$ otherwise (assuming A is positive)**

4. Ramp Signal:

- A signal that increases linearly with time.
- **Mathematically** $r(t) = \begin{cases} t; t \geq 0 \\ 0; t < 0 \end{cases}$

5. Triangular Signal:

- A periodic signal that forms a triangle shape, linearly increasing and decreasing with time, typically between a positive and negative peak.
- **Mathematically: $\Lambda(t) = 1 - |t|$ for $|t| \leq 1$, $\Lambda(t) = 0$ otherwise**

6. Sine Signal:

- A continuous periodic signal. It oscillates smoothly between -1 and 1.
- **Mathematically: $y(t)=A\sin(2\pi ft)$**

7. Cosine Signal:

- A continuous periodic signal like the sine wave but phase-shifted by π\2.
- **Mathematically: $y(t)=A\cos(2\pi ft)$**

8. Exponential Signal:

- A signal that increases or decreases exponentially with time. The rate of growth or decay is determined by the constant a.
- **Mathematically: e^(at)**

9. Unit Step Signal:

- A signal that is zero for all negative time values and one for positive time values.
- **Mathematically** $u(t) = \begin{cases} 1; t \geq 0 \\ 0; t < 0 \end{cases}$

Program:

```
clc;

clear all;

close all;
```

```
subplot(3,3,1);
t = -5:1:5;
y = [zeros(1,5),ones(1,1),zeros(1,5)];
stem(t,y);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Impulse Signal");

subplot(3,3,2);
t2 = 0:0.01:1;
f = 5;
y2 = square(2*pi*f*t2);
stem(t2,y2);
hold on;
plot(t2,y2);
xlabel("Time(s)");
ylabel("Amplitude");
title("Bipolar Pulse Signal");
legend("Discrete","Continuous");

subplot(3,3,3);
t3 = 0:0.1:1;
f = 5;
y3 = abs(square(2*pi*f*t3));
stem(t3,y3);
hold on;
plot(t3,y3);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unipolar Pulse Signal");
legend("Discrete","Continuous");
```

```
subplot(3,3,4);
t4 = -5:1:5;
y4 = t4 .*(t4>=0);
stem(t4,y4);
hold on;
plot(t4,y4);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Ramp Signal");
legend("Discrete","Continuous");

subplot(3,3,5);
t5 = 0:0.025:1;
f = 10;
y5 = sawtooth(2*pi*f*t5,0.5);
stem(t5,y5);
hold on;
plot(t5,y5);
xlabel("Time(s)");
ylabel("Amplitude");
title("Triangular Signal");
legend("Discrete","Continuous");

subplot(3,3,6);
t6 = 0:0.001:1;
f = 10;
y6 = sin(2*pi*f*t6);
stem(t6,y6);
hold on;
plot(t6,y6);
xlabel("Time(s)");
```

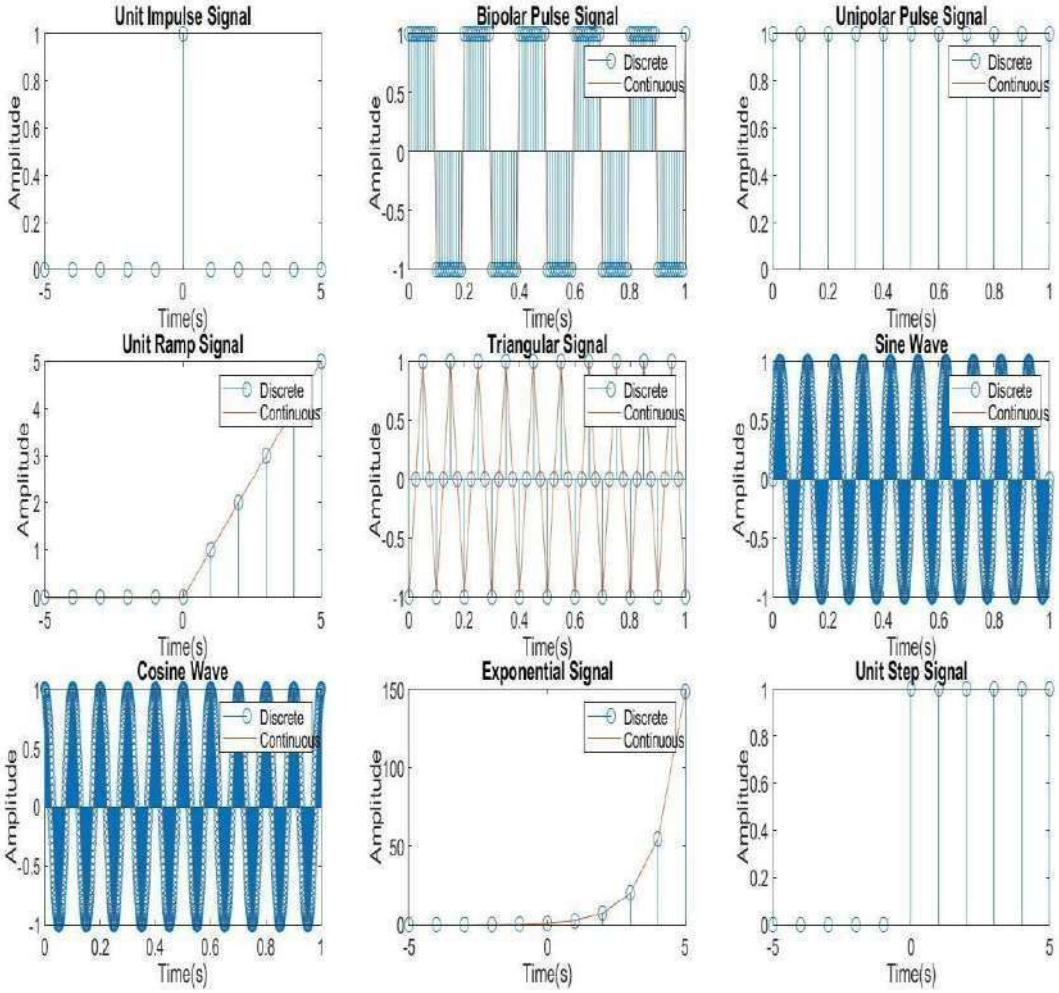
```
ylabel("Amplitude");
title("Sine Wave");
legend("Discrete","Continuous");

subplot(3,3,7);
t7 = 0:0.001:1;
f = 10;
y7 = cos(2*pi*f*t7);
stem(t7,y7);
hold  on;
plot(t7,y7);
xlabel("Time(s)");
ylabel("Amplitude");
title("Cosine Wave");
legend("Discrete","Continuous");

subplot(3,3,8);
t8 = -5:1:5;
y8 = exp(t8);
stem(t8,y8);
hold on;
plot(t8,y8);
xlabel("Time(s)");
ylabel("Amplitude");
title("Exponential Signal");
legend("Discrete","Continuous");

subplot(3,3,9);
t9 = -5:1:5;
y9 = [zeros(1,5),ones(1,6)];
 stem(t9,y9);
```

```
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Step Signal");
```

Result

Generated and Verified various Continuous and Discrete waveforms for basic test signals.

Observation



Experiment No: **2** Date: 06/08/24

Experiment Name: Verification of Sampling Theorem

Aim:

To verify Sampling Theorem.

Theory:

The Sampling Theorem, also known as the Nyquist-Shannon Sampling Theorem, states that a continuous signal can be completely reconstructed from its samples if the sampling frequency is greater than twice the highest frequency present in the signal. This critical frequency is known as the Nyquist rate.

$$-f_s > 2 \cdot f_{max}$$

Where:

- fs is the sampling frequency (rate at which the signal is sampled),
- fmax is the highest frequency present in the signal.

Applications:

- Digital audio and video processing
- Communication systems
- Image processing
- Medical imaging

Program:

```
clc;

clear all;

close all;


subplot(2,2,1);

t = 0:0.01:1;

f=10;
```

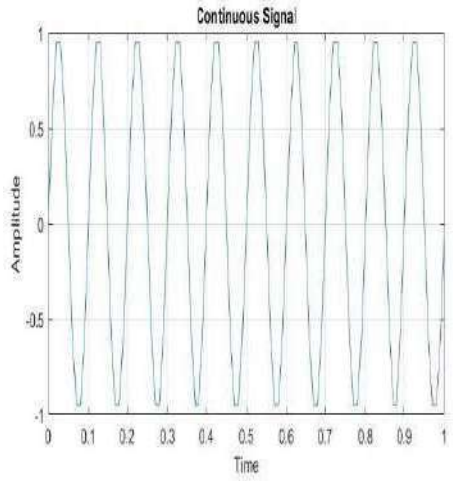
```
y = sin(2*pi*f*t);
plot(t,y);
grid(true);
xlabel("Time");
ylabel("Amplitude");
title("Continuous Signal");

 subplot(2,2,2);
 fs= 0.5*f; Undersampled
 t1 = 0:1/fs:1;
 y1 = sin(2*pi*f*t1);
 stem(t1,y1);
 hold on;
 plot(t1,y1);
 grid(true);
 xlabel("Time");
 ylabel("Amplitude");
 title("Under Sampled Signal");

 subplot(2,2,3);
 fs2= 3*f; Nyquist sampled
 t3 = 0:1/fs2:1;
 y2 = sin(2*pi*f*t3);
 stem(t3,y2);
 hold on;
 plot(t3,y2);
 xgrid(true);
 xlabel("Time");
 ylabel("Amplitude");
 legend("Discrete","Continuous")
```

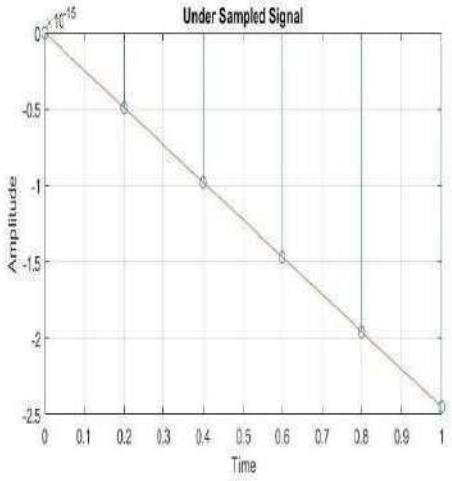
```
title("Nyquist Sampled Signal");
subplot(2,2,4);

fs2= 100*f; Oversampled
t3 = 0:1/fs2:1;
y2 = sin(2*pi*f*t3);
stem(t3,y2);
hold on;
plot(t3,y2);
grid(true);
xlabel("Time");
ylabel("Amplitude");
legend("Discrete","Continuous")
title("Over Sampled Signal");
```

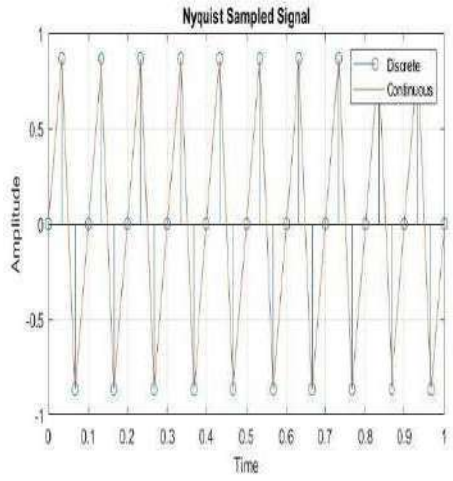
Result

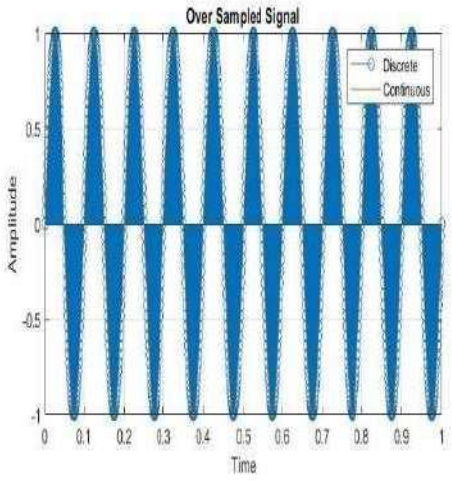
Verified Sampling Theorem using MATLAB.

Observation









Experiment No: **3** Date: 10/08/24

Experiment Name: Linear Convolution

Aim:

To find linear convolution of following sequences with and without built in function.

- 1. x(n) = [1 2 1 1]
  h(n) = [1 1 1 1]
- 2. x(n) = [1 2 1 2]
  h(n) = [3 2 1 2]

Theory:

Linear convolution is a mathematical operation used to combine two signals to produce a third signal. It's a fundamental operation in signal processing and systems theory.

Mathematical Definition:

Given two signals, *x(t)* and *h(t)*, their linear convolution is defined as:

$$\mathbf{y(t) = x(t) * h(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)\, d\tau}$$

Applications:

Filtering: Convolution is used to filter signals, removing unwanted frequencies or noise.

System Analysis: The impulse response of a system completely characterizes its behaviour, and convolution can be used to determine the output of the system given a known input.

Image Processing: Convolution is used for tasks like edge detection, blurring, and sharpening images.

Program:

1.with built-in function:

```
clc;

clear all;

close all;

x1 = input("Enter first Sequence");

h1 = input("Enter second Sequence");

y1 = conv(x1,h1);

disp("The convoluted sequence is: ");
```

```
disp(y1);
l = length(x1);
m = length(h1);
k = l+m-1;
n1 = 0:1:l-1;
n2 = 0:1:m-1;
n3 = 0:1:k-1;
 subplot(1,3,1);
 stem(n1,x1,"o");
 xlabel("n");
 ylabel("Amplitude");
 title("x(n)");
 grid on
 xlim([-1 l+1]);
 ylim([0 max(x1)+2]);

 subplot(1,3,2);
 stem(n2,h1,"o");
 xlabel("n");
 ylabel("Amplitude");
 title("h(n)");
 grid on
 xlim([-1 m+1]);
 ylim([0 max(h1)+2]);

 subplot(1,3,3);
 stem(n3,y1,"o");
 xlabel("n");
 ylabel("Amplitude");
 title("y(n)");
 grid on
 xlim([-1 k+1]);
```

```
ylim([0 max(y1)+2]);
```

2.**without built-in function:**

```
clc;
clear all;
close all;
x1 = input("Enter first Sequence");
h1 = input("Enter second Sequence");
l = length(x1);
m = length(h1);
k = l+m-1;
y1 = zeros(1,k);
for i=1:l
  for j=1:m
    y1(i+j-1) = y1(i+j-1) + x1(i)*h1(j);
  end
end
disp("The convoluted sequence is: ");
disp(y1);

n1 = 0:1:l-1;
n2 = 0:1:m-1;
n3 = 0:1:k-1;
subplot(1,3,1);
stem(n1,x1,"o");
xlabel("n");
ylabel("Amplitude");
title("x(n)");
grid on
xlim([-1 l+1]);
```

```
ylim([0 max(x1)+2]);

subplot(1,3,2);
stem(n2,h1,"o");
xlabel("n");
ylabel("Amplitude");
title("h(n)");



grid on
xlim([-1 m+1]);
ylim([0 max(h1)+2]);

subplot(1,3,3);
stem(n3,y1,"o");
xlabel("n");
ylabel("Amplitude");
title("y(n)");
grid on
xlim([-1 k+1]);
ylim([0 max(y1)+2]);
```

Result

Performed Linear Convolution using with and without built-in function.

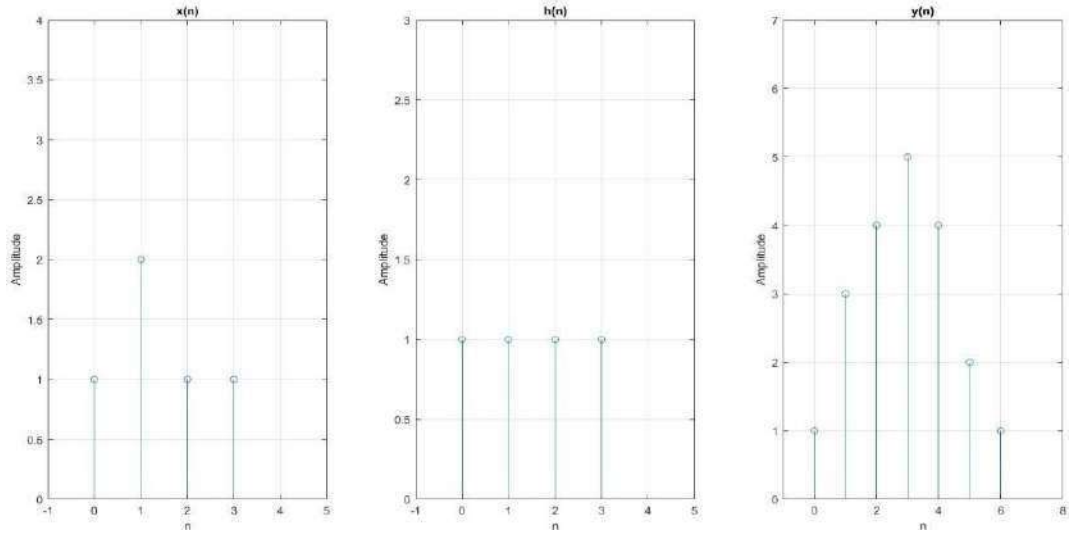
Observation

a) Enter first Sequence [1 2 1 1]

Enter second Sequence [1 1 1 1]

The convoluted sequence is:

1 3 4 5 4 2 1

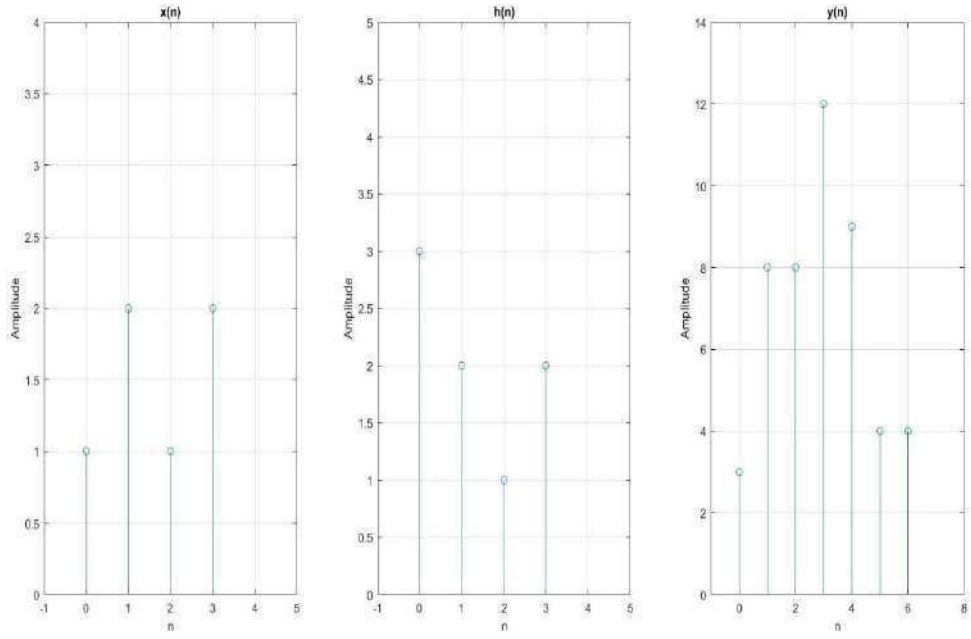


b) Enter first Sequence [1 2 1 2]

Enter second Sequence [3 2 1 2]

The convoluted sequence is:

3 8 8 12 9 4 4



Experiment No: **4** Date: 03/09/24

Experiment Name: Circular Convolution

Aim:

To find circular convolution

a. Using FFT and IFFT.
b. Using Concentric Circle Method.
c. Using Matrix Method.

Theory:

Circular convolution is a mathematical operation that is like linear convolution but is performed in a periodic or circular manner. This is particularly useful in discrete-time signal processing where signals are often represented as periodic sequences.

Mathematical Definition:

Given two periodic sequences *x[n]* and *h[n]*, their circular convolution is defined as:

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

Applications:

- Discrete-Time Filtering: Circular convolution is used for filtering discrete-time signals.
- Digital Signal Processing: It's a fundamental operation in many digital signal processing algorithms.
- Cyclic Convolution: In certain applications, such as cyclic prefix OFDM, circular convolution is used to simplify the implementation of linear convolution.

Program:

a. Using FFT and IFFT.

```
clc;

close all; c

lear all;


x1 = [1 2 1 2];

x2 = [1 2 3 4];

X1_k = fft(x1);

X2_k = fft(x2);
```

```
Y1_k = X1_k.*X2_k;
y1 =ifft(Y1_k);
disp("Using FFT and IFFT:")
disp(y1);
```

b. Using Concentric Circle Method.

```
clc;
close all;
clear all;

x = [1 2 1 2];
h = [1 2 3 4];
N = max(length(x),length(h));
y = zeros(1,N);
for n=1:N
  h_s = circshift(h,n-1); %shifting h(n) by 1 unit
  y(n) = sum(x.*h_s);
end

disp("Using Concentric Circle Method:");
disp(y);
```

c. Using Matrix Method.

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

```
 x = [1 2 1 2];
 h = [1 2 3 4];
N = max(length(x),length(h));
h_n = zeros(N,N);
for n=1:N h_s = circshift(h,n-1); %shifting h(n) by 1 unit
h_n(:,n) = h_s;
end
y = h_n *x';
disp("Using Concentric Circle Method:")
disp(y');
```

Result

Performed Circular Convolution using a) FFT and IFFT; b) Concentric Circle method; c) Matrix method and verified result.

Observation

a) USING FFT AND IFFT

Using FFT and IFFT:

16 14 16 14

b) USING Concentric Circle Method

Using Concentric Circle Method:

16 14 16 14

c) USING Matrix Method

Using Matrix Method.:

16 14 16 14

Experiment No: **5** Date: 10/09/24

Experiment Name: Linear Convolution using CircularConvolution and Vice versa.

Aim:

1. To perform Linear Convolution using Circular Convolution.
2. To perform Circular Convolution using Linear Convolution.

Theory:

Performing Linear Convolution Using Circular Convolution

Method:

1. **Zero-Padding:**
   - Pad both sequences *x[n]* and *h[n]* with zeros to a length of at least *2N-1*, where *N* is the maximum length of the two sequences. This ensures that the circular convolution will not wrap around and introduce artificial periodicity.
2. **Circular Convolution:**
   - Perform circular convolution on the zero-padded sequences.
3. **Truncation:**
   - Truncate the result of the circular convolution to the length *N1 + N2 - 1*, where *N1* and *N2* are the lengths of the original sequences *x[n]* and *h[n]*, respectively.

Example:

Consider the sequences *x[n] = [1, 2, 3]* and *h[n] = [4, 5]*.

1. Zero-padding:
   - Pad *x[n]* to [1, 2, 3, 0, 0] and *h[n]* to [4, 5, 0, 0].
2. Circular Convolution:
   - Perform circular convolution on the zero-padded sequences. The result will be [4, 13, 21, 15, 0].
3. Truncation:
   - Truncate the result to [4, 13, 21, 15].

This result is the same as the linear convolution of *x[n]* and *h[n]*.

Performing Circular Convolution Using Linear Convolution

Method:

1. **Zero-Padding:**
    - Pad both sequences *x[n]* and *h[n]* to a length of at least *2N-1*, where *N* is the maximum length of the two sequences.
2. **Linear Convolution:**
    - Perform linear convolution on the zero-padded sequences.
3. **Modulus Operation:**
    - Apply the modulus operation to the indices of the linear convolution result, using the period *N*. This effectively wraps around the ends of the sequence, making it circular.

Example:

Using the same sequences as before, *x[n] = [1, 2, 3]* and *h[n] = [4, 5]*.

1. Zero-padding:
    - Pad *x[n]* to [1, 2, 3, 0, 0] and *h[n]* to [4, 5, 0, 0].
2. Linear Convolution:
    - Perform linear convolution. The result will be [4, 13, 21, 15, 0].
3. Modulus Operation:
    - Apply the modulus operation to the indices: [4, 13, 21, 15, 0] becomes [4, 13, 2, 15, 0].

Program:

1. Linear Convolution using Circular Convolution

```
clc;

clear all;

close all;

x = [1 2 3 4];

h = [1 1 1 ];

l = length(x);

m = length(h);

k = l+m-1;

x = [x zeros(1,k-l)];

h = [h zeros(1,k-m)];
```

```
X_k = fft(x);
H_k = fft(h);
Y_k = X_k.*H_k;
y = ifft(Y_k);
disp("Linear Convolution using Circular Convolution :");
disp(y);
```

2.Circular convolution using Linear Convolution

```
clc;

close all;

clear all;

x = [1 2 3 4];
h = [1 1 1 ];
l = length(x);
m = length(h);
Lc = max(l,m);
Ll= l+m-1;
y = conv(x,h);
for i=1:Ll-Lc
  y(i) = y(i) + y(Lc+i);
end
for i=1:Lc
y1(i) = y(i);
end
disp("Circular convolution using Linear Convolution:")
disp(y1);
```

Result

Performed

a) Linear Convolution using Circular Convolution;

b) Circular Convolution usingLinear Convolution

and verified result.

Observation

1) Linear Convolution using Circular Convolution :

Linear Convolution using Circular Convolution:

1 3 6 9 7 4

2.Circular convolution using Linear Convolution

Circular convolution using Linear Convolution:

8 7 6 9

Experiment No: 6 Date: 01/10/24

DFT AND IDFT

Aim:

1. DFT using inbuilt function and without using inbuilt function. Also plot magnitude and phase plot of DFT

2. IDFT using inbuilt function and without using inbuilt function.

Theory:

Discrete Fourier Transform (DFT)

The **Discrete Fourier Transform (DFT)** is a mathematical transformation used to analyze the frequency content of discrete signals. For a sequence x[n] of length N, the DFT is defined as:

$$\boldsymbol{X[k] = \sum_{n=0}^{N} x[n] \cdot e^{-j\frac{2\pi}{N}nk}, k = 0, 1, 2, \ldots, N-1}$$

- X[k] is the DFT of the sequence x[n].
- The exponential factor represents $\boldsymbol{e^{-j\frac{2\pi}{N}nk}}$ the complex sinusoidal basis functions.
- The DFT maps the time-domain signal into the frequency domain.

Inverse Discrete Fourier Transform (IDFT)Method:

The **Inverse Discrete Fourier Transform (IDFT)** is used to convert a frequency-domain sequence X[k] back into its time-domain sequence x[n]. The IDFT is defined as:

$$\boldsymbol{x[n] = \frac{1}{N} \sum_{k=0}^{N} X[k] \cdot e^{j\frac{2\pi}{N}nk}, n = 0, 1, 2, \ldots, N-1}$$

- The IDFT takes the frequency components X[k] and reconstructs the original sequence x[n].
- The exponential factor $\boldsymbol{e^{j\frac{2\pi}{N}nk}}$ is the inverse of the DFT's complex sinusoidal basis functions

.

Application

- Spectrum (Analysis)
- Filtering
- Compression
- Modulation
- Convolution
- Demodulation

- Equalization
- Restoration
- Detection
- Estimation

Program:

1. Discrete Fourier Transform (DFT)

```
clc;
clear all;
close all;
x=input("enter sequence:");
N=input("enter the N point:");
l=length(x);
x=[x zeros(1,N-l)];
X=zeros(1,N);
for k=0:N-1
    for n=0:N-1
        X(k+1)=X(k+1)+x(n+1)*exp(-1j*2*pi*n*k/N);
    end
end
disp('X');
disp(X);
disp('round(X)');
disp(round(X));
%verification
disp('fft');
disp(fft(x));

k=0:N-1;
magX=abs(X);
```

```
phaseX=angle(X);
subplot(2,1,1);
stem(k,magX);
title("Magnitude Plot");
hold on;
plot(k,magX);
subplot(2,1,2);
stem(k,phaseX);
hold on;
title("Phase Plot");
plot(k,phaseX);
```

2. IDFT

```
clc;
clear all;
close all;
X=input("enter sequence:");
N=input("enter the n point:");
l=length(X);
X=[X zeros(1,N-l)];
x=zeros(N,1);
for k=0:N-1
    for n=0:N-1
        x(n+1)=x(n+1)+X(k+1)*exp(1j*2*pi*n*k/N);
    end
end
x=1/N.*x;
disp('x');
disp(x);
disp('round(x)');
```

```
disp(round(x));
disp('ifft');
disp(ifft(X));
```

Result:

Performed

1) DFT using inbuilt function and without using inbuilt function. Also plotted magnitude and phase plot of DFT.

2) IDFT using inbuilt function and without using inbuilt function.

and verified the result.

Observation:

1.DFT

enter sequence:[1 1 1 0]

enter the N point:8

X

Columns 1 through 7

3.0000 + 0.0000i 1.7071 - 1.7071i 0.0000 - 1.0000i 0.2929 + 0.2929i 1.0000 + 0.0000i 0.2929 - 0.2929i -0.0000 + 1.0000i

Column 8

1.7071 + 1.7071i

round(X)

Columns 1 through 7

3.0000 + 0.0000i 2.0000 - 2.0000i 0.0000 - 1.0000i 0.0000 + 0.0000i 1.0000 + 0.0000i 0.0000 + 0.0000i 0.0000 + 1.0000i

Column 8

2.0000 + 2.0000i

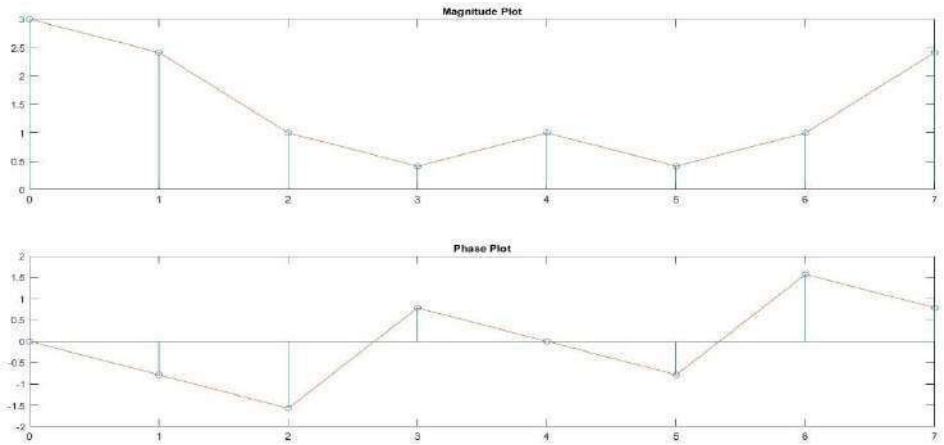
fft

Columns 1 through 7

3.0000 + 0.0000i 1.7071 - 1.7071i 0.0000 - 1.0000i 0.2929 + 0.2929i 1.0000 + 0.0000i 0.2929 - 0.2929i 0.0000 + 1.0000i

Column 8

1.7071 + 1.7071i



2.IDFT

enter sequence: [3.0000 + 0.0000i 1.7071 - 1.7071i 0.0000 - 1.0000i 0.2929 + 0.2929i 1.0000 + 0.0000i 0.2929 - 0.2929i 0.0000 + 1.0000i]
enter the n point:8
x
   0.7866 - 0.2134i
   0.6982 - 0.0000i
   0.7866 + 0.2134i
  -0.0000 + 0.3018i
   0.2134 + 0.2134i
   0.3018 - 0.0000i
   0.2134 - 0.2134i
   0.0000 - 0.3018i

round(x)
     1
     1
     1
     0
     0
     0
     0
     0

ifft
  Columns 1 through 7

   0.7866 - 0.2134i 0.6982 + 0.0000i   0.7866 + 0.2134i   0.0000 + 0.3018i   0.2134 + 0.2134i 0.3018 + 0.0000i   0.2134 - 0.2134i

  Column 8

   0.0000 - 0.3018i

Experiment No: 7 Date: 01/10/24

Properties of DFT

Aim:

Verify following properties of DFT using Matlab/Scilab.

1.Linearity Property

2.Parsevals Theorem

3.Convolution Property

4.Multiplication Property

Theory:

1. Linearity Property

The linearity property of the DFT states that if you have two sequences x1[n] and x2[n], and their corresponding DFTs are X1[k] and X2[k], then for any scalar a and b:

$$\mathbf{DFT\{a\cdot x1[n]+b\cdot x2[n]\} = a\cdot DFT\{x1[n]\}+b\cdot DFT\{x2[n]\}}$$

2. Parseval's Theorem

Parseval's theorem states that the total energy of a signal in the time domain is equal to the total energy in the frequency domain. For a sequence x[n] and its DFT X[k]:

$$\sum_{n=0}^{N-1}]x[n]^2=\frac{1}{N}\sum_{n=0}^{N-1}X[k]$$

3.Convolution Property

The convolution property of the DFT states that the circular convolution of two sequences in the time domain is equivalent to the element-wise multiplication of their DFTs in the frequency domain:

$$\mathbf{DFT\{x1[n]\circledast x2[n]\}=DFT\{x1[n]\}\cdot DFT\{x2[n]\}}$$

4. Multiplication Property

The multiplication property of DFT states that pointwise multiplication in the time domain corresponds to circular convolution in the frequency domain:

$$\mathbf{DFT\{x1[n]\cdot x2[n]\} =}\frac{1}{N}\mathbf{DFT\{x1[n]\}\circledast DFT\{x2[n]\}}$$

Program:

1. Linearity Property

```
clc;
```

```
clear all;
close all;
x=input("enter first sequence");
h=input("enter sequence sequence:");
lx=length(x);
lh=length(h);
if lx>lh
    h=[h zeros(1,lx-lh)]
else
    x=[x zeros(1,lh-lx)]
end
a=input("enter value of 'a':");
b=input("enter value of 'b':");
lhs=fft((a.*x)+(b.*h));
rhs=a.*fft(x)+b.*fft(h);
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp('Linearity property verified');
else
    disp('Linearity property not verified');
end
```

2. Parseval's Theorem

```
clc;
clear all;
close all;
x=input("enter first sequence:");
```

```
h=input("enter second sequence:");
N=max(length(x),length(h));
xn=[x zeros(1,N-length(x))];
hn=[h zeros(1,N-length(h))];
lhs=sum(xn.*conj(hn));
rhs=sum(fft(xn).*conj(fft(hn)))/N;
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp("Parseval's Theorem verified");
else
    disp("Parseval's Theorem not verified");
end
```

3.Convolution Property

```
clc;
clear all;
close all;
x=input("enter first sequence");
h=input("enter sequence sequence:");
N=max(length(x), length(h));
xn=[x zeros(N-length(x))];
hn=[h zeros(N-length(h))];
Xn=fft(xn);
Hn=fft(hn);
lhs=cconv(xn,hn,N);
rhs=ifft(Xn.*Hn);
disp('LHS');
```

```
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp('Circular Convolution verified')
else
    disp('Circular Convolution not verified');
end
```

4. Multiplication Property

```
clc;
clear all;
close all;
x=input("enter first sequence");
h=input("enter sequence sequence:");
N=max(length(x), length(h));
xn=[x zeros(N-length(x))];
hn=[h zeros(N-length(h))];
lhs=fft(xn.*hn);
Xn=fft(xn);
Hn=fft(hn);
rhs=(cconv(Xn,Hn,N))/N;
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp('Multiplication property verified');
else
    disp('Multiplication property not verified');
end
```

Result:

Performed and verified the following properties of DFT:

1.Linearity Property

2.Parsevals Theorem

3.Convolution Property

4.Multiplication Property

Observation:

1. Linearity Property

enter first sequence[1 2 3 4]

enter sequence sequence:[1 1 1 1]

x =

1 2 3 4

enter value of 'a':2

enter value of 'b':3

LHS

32.0000 + 0.0000i -4.0000 + 4.0000i -4.0000 + 0.0000i -4.0000 - 4.0000i

RHS

32.0000 + 0.0000i -4.0000 + 4.0000i -4.0000 + 0.0000i -4.0000 - 4.0000i

Linearity property verified

2. Parseval's Theorem

enter first sequence:[1 2 3 4]

enter second sequence:[1 1 1 1]

LHS

10

RHS

10

Parseval's Theorem verified

3.Convolution Property

enter first sequence[1 2 3 4]

enter sequence sequence:[1 1 1 1]

LHS

10 10 10 10

RHS

10 10 10 10

Circular Convolution verified

4.Multiplication Property

enter first sequence[1 2 3 4]

enter sequence sequence:[1 1 1 1]

LHS

Columns 1 through 3

10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i

Column 4

-2.0000 - 2.0000i

RHS

Columns 1 through 3

10.0000 + 0.0000i -2.0000 + 2.0000i -2.0000 + 0.0000i

Column 4

-2.0000 - 2.0000i

Multiplication property verified

Experiment No: 8 Date: 08/10/24

OVERLAP ADD AND OVERLAP SAVE METHOD

Aim:

Implement overlap add and overlap save method using Matlab/Scilab.

Theory:

Both the Overlap-Save and Overlap-Add methods are techniques used to compute the convolution of long signals using the Fast Fourier Transform (FFT). The direct convolution of two signals, especially when they are long, can be computationally expensive. These methods allow us to break the signals into smaller blocks and use the FFT to perform the convolution more efficiently.

Overlap-Save Method

The Overlap-Save method deals with circular convolution by discarding the parts of the signal that are corrupted by wrap-around effects. Here's how it works:

1. Block Decomposition: The input signal is divided into overlapping blocks. If the filter has length and we use blocks of length, the overlap is samples, so each block has new samples and samples from the previous block.

2. FFT and Convolution: Each block is convolved with the filter using FFT. However, because of circular convolution, the result contains artifacts due to the overlap.

3. Discard and Save: We discard the first samples from each block (the part affected by the wrap-around) and save the remaining samples. This gives us the correct linear convolution.

Overlap-Add Method

The Overlap-Add method, on the other hand, handles circular convolution by adding overlapping sections of the convolved blocks. Here’s how it works:

1. Block Decomposition: The input signal is split into non-overlapping blocks of size. Each block is then zero-padded to a size of , where is the length of the filter.

2. FFT and Convolution: Each block is convolved with the filter using FFT. Since the blocks are zero-padded, the convolution produces valid linear results, but the output blocks overlap.

3. Overlap and Add: After convolution, the results of each block overlap by samples. These overlapping regions are added together to form the final output.

Program:

1. Overlap Add

```
clc;
clear all;
close all;

x = input('Enter the input sequence x : ');
h = input('Enter the impulse response h : ');
L = length(h);  % Length of impulse response
N = length(x);
M = length(h);
x_padded = [x, zeros(1, L - 1)];
y = zeros(1, N + M +1);

num_sections = (N + L - 1) / L; % Calculate number of sections
for n = 0:num_sections-1
    start_idx = n * L + 1;
    end_idx = start_idx + L - 1;
    x_section = x_padded(start_idx:min(end_idx, end));

    conv_result = conv(x_section, h);
```

```
    y(start_idx:start_idx     +     length(conv_result)     -     1)
=y(start_idx:start_idx + length(conv_result) - 1) + conv_result;
end
y = y(1:N + M - 1);
y_builtin = conv(x, h);
% Display results
disp('Overlap-add convolution result:');
disp(y);
disp('Built-in convolution result:');
disp(y_builtin);
figure;
subplot(2, 1, 1);
stem(y, 'filled');
title('Overlap-add Convolution Result');
grid on;
subplot(2, 1, 2);
stem(y_builtin, 'filled');
title('Built-in Convolution Result');
grid on;
```

2.Overlap Save

```
clc;
clear all;
close all;
x = input("Enter 1st sequence: ");
h = input("Enter 2nd sequence: ");
```

```
N = input("Fragmented block size: ");
y = ovrlsav(x, h, N);
disp("Using Overlap and Save method");
disp(y);
disp("Verification");
disp(cconv(x,h,length(x)+length(h)-1));
function y = ovrlsav(x, h, N)
    if (N < length(h))
        error("N must be greater than the length of h");
    end
    Nx = length(x);  % Length of input sequence x
    M = length(h);   % Length of filter sequence h
    M1 = M - 1;      % Length of overlap
    L = N - M1;      % Length of non-overlapping part
    x = [zeros(1, M1), x, zeros(1, N-1)];
    h = [h, zeros(1, N - M)];
    K = floor((Nx + M1 - 1) / L);
    Y = zeros(K + 1, N);

    for k = 0:K
        xk = x(k*L + 1 : k*L + N);
        Y(k+1, :) = cconv(xk, h, N);
    end
```

```
    Y = Y(:, M:N)';
    y = (Y(:))';
end
```

Result:

Performed Overlap Save and Overlap Add method and verified the result.

Observation:

1. Overlap Add

Enter the input sequence x : [3 -1 0 1 3 2 0 1 2 1]

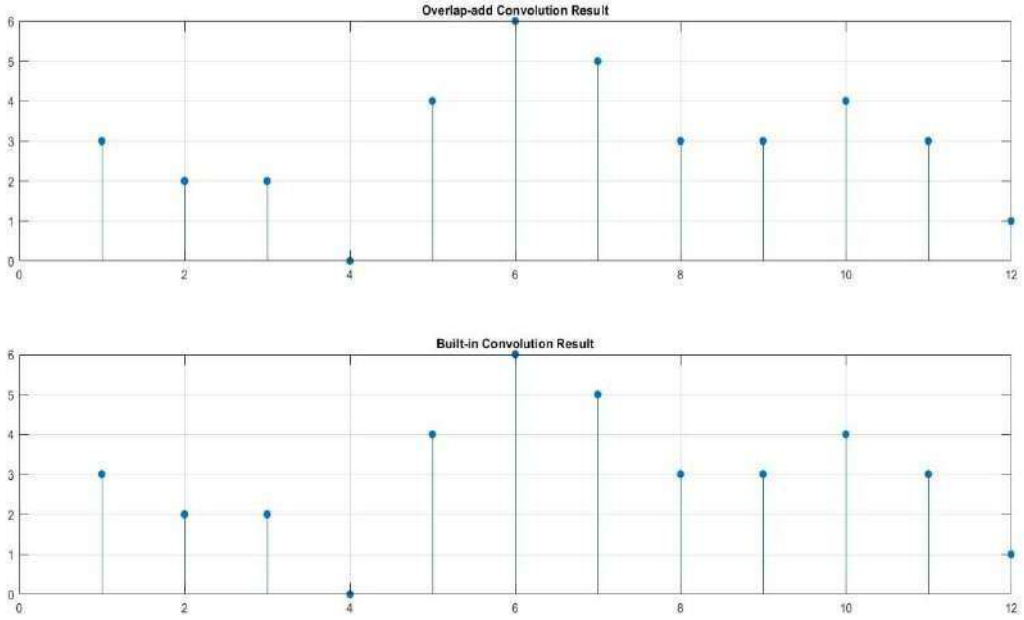
Enter the impulse response h : [1 1 1]

Overlap-add convolution result:

3 2 2 0 4 6 5 3 3 4 3 1

Built-in convolution result:

3 2 2 0 4 6 5 3 3 4 3 1



2.Overlap Save

Enter 1st sequence: [3 -1 0 1 3 2 0 1 2 1]

Enter 2nd sequence: [1 1 1]

Fragmented block size: 3

Using Overlap and Save method

3 2 2 0 4 6 5 3 3 4 3 1

Verification

3.0000 2.0000 2.0000 0 4.0000 6.0000 5.0000 3.0000 3.0000 4.0000 3.0000 1.0000

Experiment No:9 Date:14/09/2024

FAMILIARIZATION OF TMS 320C6748

Aim

To explore the architectural and functional capabilities of the TMS320C6748 DSP processor.

TMS 320C6748

Overview of the TMS320C6748 DSP Processor: The TMS320C6748 DSP processor is designed tohandle intensive digital signal processing tasks efficiently. At its core is the powerful C674x™ DSPCPU, optimized for real-time embedded applications, multimedia processing, and other high- computation requirements.

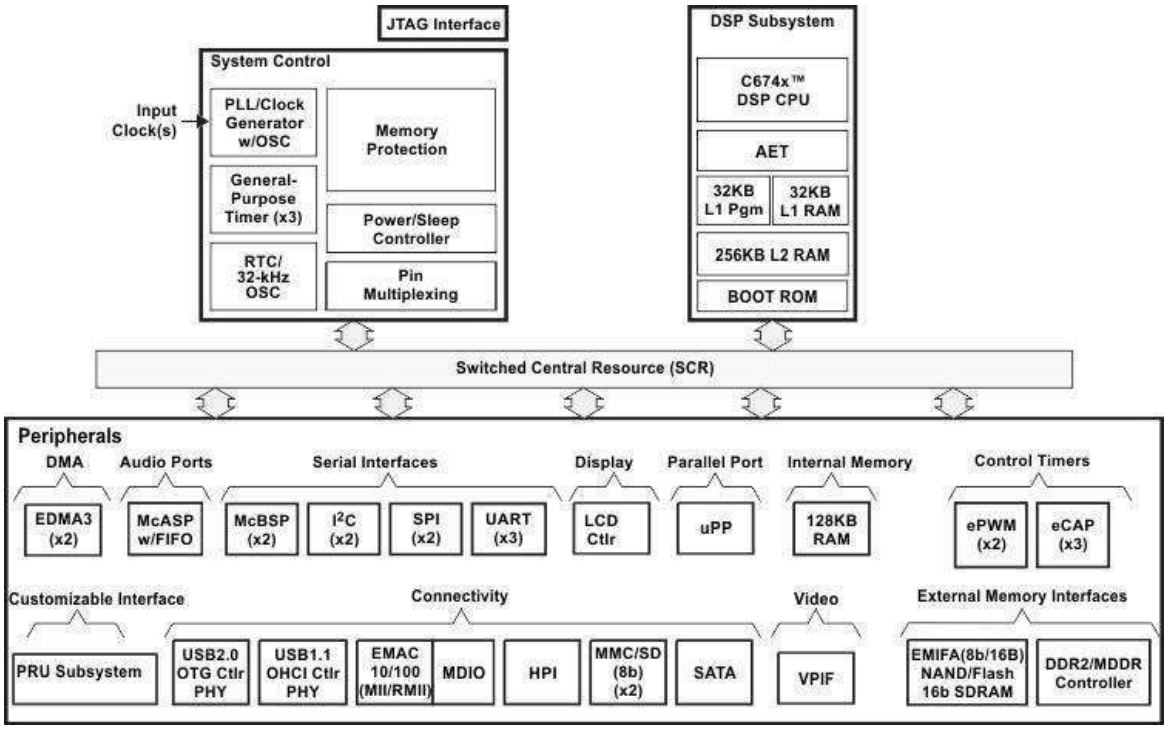
DSP Subsystem and Memory Components: The DSP subsystem includes several memory components for efficient storage and data handling. A 32 KB L1 Program Cache and 32 KBL1 RAMensure quick access to frequently used data, while the 256 KB L2 RAM provides storage for larger datasets. Additionally, a BOOT ROM is available to facilitate the processor's startup sequence.

System Control Features: Essential to managing the processor's operation, the System Control section includes a PLL/Clock Generator with Oscillator (OSC) to supply clock signals, a General-
Purpose Timer, and an RTC/32-kHz Oscillator for precise timing
functions. This section also provides Memory Protection for secure data handling, aPower/Sleep Controller to optimize power consumption, and Pin Multiplexing for I/O configuration flexibility.

Debugging and System Testing: For system testing and troubleshooting, the TMS320C6748includes aJTAG Interface. This interface supports connection to external debugging tools,
allowing developers to perform in-depth analysis and testing.

Switched Central Resource (SCR): At the heart of data management, the SCR is a high- speed interconnect linking the DSP subsystem, system control, and various peripherals. The SCR efficientlymanages data flow, facilitating smooth communication across the processor even when multiple subsystems are active.

Peripherals and Interfaces: The TMS320C6748 offers a versatile range of peripherals. Direct Memory Access (DMA) through EDMA3 with two channels enables rapid data transfers without CPU involvement. Audio Ports, including McASP and McBSP, support audio dataI/O, making the processor suitable for audio applications. Multiple Serial Interfaces (I2C, SPI, and UART) enable communication with devices like sensors and storage units, while the LCD Controllersupports direct display interfacing. Additionally, the 128 K Internal RAM provides further data storage.


JTAG Interface
System Control
Input Clock(s)
PLL/Clock Generator w/OSC
Memory Protection
General-Purpose Timer (x3)
Power/Sleep Controller
RTC/ 32-kHz OSC
Pin Multiplexing
DSP Subsystem
C674x™ DSP CPU
AET
32KB L1 Pgm
32KB L1 RAM
256KB L2 RAM
BOOT ROM
Switched Central Resource (SCR)
Peripherals
DMA
Audio Ports
Serial Interfaces
Display
Parallel Port
Internal Memory
Control Timers
EDMA3 (x2)
McASP w/FIFO
McBSP (x2)
I2C (x2)
SPI (x2)
UART (x3)
LCD Ctlr
uPP
128KB RAM
ePWM (x2)
eCAP (x3)
Customizable Interface
Connectivity
Video
External Memory Interfaces
PRU Subsystem
USB2.0 OTG Ctlr PHY
USB1.1 OHCI Ctlr PHY
EMAC 10/100 (MII/RMII)
MDIO
HPI
MMC/SD (8b) (x2)
SATA
VPIF
EMIFA(8b/16B) NAND/Flash 16b SDRAM
DDR2/MDDR Controller

PRU Subsystem for Real-Time Control: The Programmable Real-time Unit (PRU) subsystem addscustomization and flexibility, enabling the processor to handle specialized tasks in real-time.

Connectivity Options: For connectivity, the processor supports USB 2.0 OTG, USB 1.1, Ethernet(EMAC 10/100) for network connections, as well as MDIO and HPI interfaces. It also supports MMC/SD and SATA ports, enhancing its capability to interface with various storage devices.

Video Port Interface (VPIF): The VPIF feature makes the TMS320C6748 suitable for videoapplications by supporting video input and output functions.

Control Timers: The Control Timers section includes ePWM and eCAP timers, providing precise control over pulse-width modulation and capture events. These features are useful for motor control,sensor data acquisition, and other control-based applications.

External Memory Interfaces: To support additional memory, the processor includes interfaces for EMIFA (8b/16b), NAND/Flash 16b, and a DDR2/MDDR Controller, allowing theconnection of external DRAM and flash memory for data-intensive applications.

Application

The TMS320C6748 DSP processor is versatile, supporting applications across industries due to itspowerful DSP core, real-time control, and connectivity options. It is ideal for audio processing indigital audio workstations and industrial automation tasks like motor control and robotics. In the medical field, it handles real-time bio-signal processing for imaging and monitoring, while in video and image processing, it's used for surveillance and computer vision. Theprocessor also serves well in communication systems (e.g., modems and wireless networks), automotive ADAS, energy management, test and measurement equipment, and IoT applications, enabling smart devices and automation in home environments.

Result

Studied and obtained a comprehensive overview of the TMS320C6748 DSP processor, highlighting its robust DSP CPU core, high-speed data handling through the Switched Central Resource(SCR), and its extensive set of peripherals and connectivity option

Experiment : 10 Date:14/10/2024

GENERATION OF SINE WAVE USING DSP PROCESSOR

Aim

To generate a sine wave using DSP processor

Theory

Sinusoidal are the most smooth signals with no abrupt variation in their amplitude, the amplitude witnesses gradual change with time. Sinusoidal signals can be defined as a periodic signal with waveform as that of a sine wave. The amplitude of sine wave increase from a value of 0 at 0° angle to a maximum value of 1 at 90°, it further reaches its minimum valueof -1 at 270° and then return to 0 at 360°. After any angle greater than 360°, the sinusoidal signal repeats the values so we can say that time period of sinusoidal signal is 2π i.e. 360°.If we observe the graph, we can see that the amplitude varying gradually with a maximum valueof 1 and a minimum value of -1. We can also observe that the wave begins to repeat its value after a time period or angle value of 2π hence periodicity of sinusoidal signal is 2π.

These are sinusoidal signal parameters:

- **Graph:** It is a plot used to depict the relation between quantities. Depending upon the number of variables, we can decide to number of axes each perpendicular to the other.
- **Time period: The** period for a signal can be defined as the time taken by a periodic signal to complete one cycle.
- **Amplitude:** Amplitude can be defined as the maximum distance between the horizontal axis and the vertical position of any signal.
- **Frequency:** This can be defined as the number of times a signal oscillates in one second. It can be mathematically defined as the reciprocal of a period.
- **Phase:** It can be defined as the horizontal position of a waveform in one oscillation. The symbol θ is used to indicate the phase.

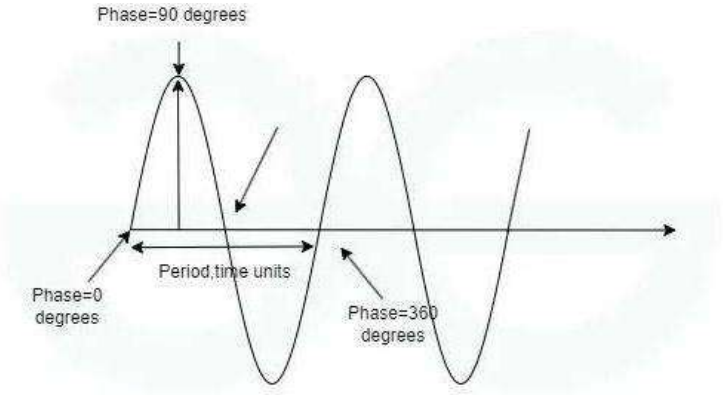
If we consider a sinusoidal signal y(t) having an amplitude A, frequency f, and phase of quantity then we can represent the signal as

$$y(t) = A\ sin(2\pi ft + \theta)$$

If we denote 2πf as an angular frequency ω the we can re-write the signal as

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

Output


Phase=90 degrees
Period,time units
Phase=0
degrees
Phase=360
degrees

PROCEDURE

1. Open Code Composer Studio,
   Click on File - New – CCS Project
   Select the Target – C674X Floating point DSP , TMS320C6748 , and
   Connection – Texas Instruments XDS 100v2 USB Debug Probe and Verify.
   Give the project name and select Finish.

2. Type the code program for generating the sine wave and choose
   File – Save As and then save the program with a name including 'main.c'. Delete the already existing main.c program.

3. Select Debug and once finished, select the Run option.

4. From the Tools Bar, select Graphs – Single Time.
   Select the DSP Data Type as 32-bit Floating point and time display unit as second(s).
   Change the Start address with the array name used in the program(here,a).

5. Click OK to apply the settings and Run the program or clock Resume in CCS.

PROGRAM

```
#include<stdio.h>
#include<math.h>
#define pi 3.1415625
float a[200];
main()
{
    int i=0;
    for(i=0;i<200;i++)
        a[i]=sin(2*pi*5*i/200);
}
```

RESULT

Generated a sine wave using DSP processor

Experiment : 11 Date:14/10/2024

LINEAR CONVOLUTION USING DSP PROCESSOR

Aim

To perform linear convolution of two sequences using DSP processor.

Theory

Linear convolution is one of the fundamental operations used extensively in signal and system in electrical engineering. It has applications in areas like audio processing, signal filtering, imaging, communication systems and more.

In simple terms, linear convolution is the process of combining two signals or functions to produce a third signal or function. Formally, the linear convolution of two functions f(t) and g(t) is defined as:

The formula for linear convolution of two discrete signals x[n] and h[n] is given by:

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

where:

- x[n] is the input signal.
- h[n] is the impulse response of the system.
- y[n] is the output signal.

In the context of linear convolution in DSP, this operation is applied to digital signals. DSP systems utilize algorithms to perform convolution efficiently, often leveraging Fast Convolution methods to handle large datasets and real-time processing.

Applications of Linear Convolution :

- Filtering: Used in digital filters to process signals.
- Image Processing: Applied for edge detection and blurring.
- System Analysis: Helps in analyzing LTI systems in response to inputs.

Output

Linear Convolution Output

4
11
20
30
20
11
4

Procedure

1. Open Code Composer Studio,
   Click on File - New – CCS Project
   Select the Target – C674X Floating point DSP , TMS320C6748 , and
   Connection – Texas Instruments XDS 100v2 USB Debug Probe and Verify.
   Give the project name and select Finish.

2. Type the code program for generating the sine wave and choose
   File – Save As and then save the program with a name including 'main.c'. Delete the already existing main.c program.

3. Select Debug and once finished, select the Run option.

4. In the Debug perspective, click Resume to run the code on DSP.
   Observe the console output to verify the convolution result.

Program

```
#include<stdio.h>
int y[7];
void main()
{
   int m=4;                    //Length of input sample sequence
   int n=4;                   // Length of impulse response Co-efficient
   int i,j;
   int x[7]={1,2,3,4};         //Input signal sample
   int h[7]={4,3,2,1};          //Impulse Response Co-efficient
         for(i=0;i<m+n-1;i++)
         {
            y[i]=0;
            for(j=0;j<=i;j++)
            {
               y[i]+=x[j]*h[i-j];
            }

         }
  printf("Linear Convolution output\n:");
         for(i=0;i<m+n-1;i++)
         {
            printf("%d\n",y[i]);
         }
}
```

Result

Performed linear convolution of two sequences using DSP processor.

Experiment : 12 Date:19/10/2024

FIR FILTERS

Aim

To implement a low pass filter, high pass filter, bandpass filter, and band reject filter using rectangular window.

Theory

A low-pass FIR filter is a filter that allows signals with lower frequencies to pass through while attenuating signals with higher frequencies. The exact frequency response of the filter is dependent on the filter design. Low-pass filters are used in many applications, including audio, image blurring, and data smoothing. For example, in audio, low-pass filters are sometimes called high-cut or treble-cut filters. Low-pass filters are the opposite of high-pass filters, which allow high-frequency signals to pass through. Filter designers often use the low-pass form as a prototype filter, which has unity bandwidth and impedance.

- Purpose: A low-pass filter allows frequencies below a specified cutoff frequency fc to pass through while attenuating frequencies above fc.
- Ideal Frequency Response: The ideal frequency response

$$H_{LPF}(e^{j\omega}) = \begin{cases} 1 & \text{if } |\omega| \leq \omega_c \\ 0 & \text{if } |\omega| > \omega_c \end{cases}$$

where $\omega_c = 2\pi f_c / f_s$ is the normalized cutoff frequency.

High-pass filters allow signals with frequencies above a certain cutoff to pass through, while attenuating signals with lower frequencies. The amount of attenuation depends on the filter's design.

High-pass filters are used in many applications, including:

- Blocking DC from circuitry
- Removing low-frequency noise from audio signals
- Removing low-frequency trends from time-series data
- Redirecting higher frequency signals to speakers in sound systems

- High-pass filters can be designed using the Kaiser window, least squares, or equiripple methods. High-pass filters are often implemented as an RC circuit, with a

capacitor in series with the signal source and a resistor in parallel. FIR filters are inherently stable and can be designed to have linear phase. However, they can have long transient responses and may be computationally expensive in some applications.

- Purpose: A high-pass filter allows frequencies above a specified cutoff frequency fc to pass through while attenuating frequencies below fc.
- Ideal Frequency Response: The ideal frequency response

$$H_{HPF}(e^{j\omega}) = \begin{cases} 0 & \text{if } |\omega| \leq \omega_c \\ 1 & \text{if } |\omega| > \omega_c \end{cases}$$

A band-pass finite impulse response (FIR) filter is a type of filter that can be designed to pass frequencies within a specific range and reject other frequencies. FIR filters are used in a variety of applications, including wireless transmitters and receivers, and in noisy environments. Band-pass filters allow frequencies within a certain range to pass through, while blocking frequencies that are too high or too low. In wireless transmitters, band-pass filters limit the bandwidth of the output signal to prevent interference with other stations. In receivers, band-pass filters allow signals within a selected range of frequencies to be heard or decoded. Band-pass FIR filters can be designed using MATLAB. MATLAB can be used to select a suitable window function and an ideal filter, and then truncate the impulse response to obtain a FIR filter.

FIR filters often require a higher filter order than IIR filters to achieve a given level of performance. This can result in a greater delay than for an equal performance IIR filter.

- Purpose: A band-pass filter allows frequencies within a specific range [fc1,fc2] to pass through while attenuating frequencies outside this range.

Ideal Frequency Response: The ideal frequency response

$$H_{BPF}(e^{j\omega}) = \begin{cases} 1 & \text{if } \omega_1 \leq |\omega| \leq \omega_2 \\ 0 & \text{otherwise} \end{cases}$$

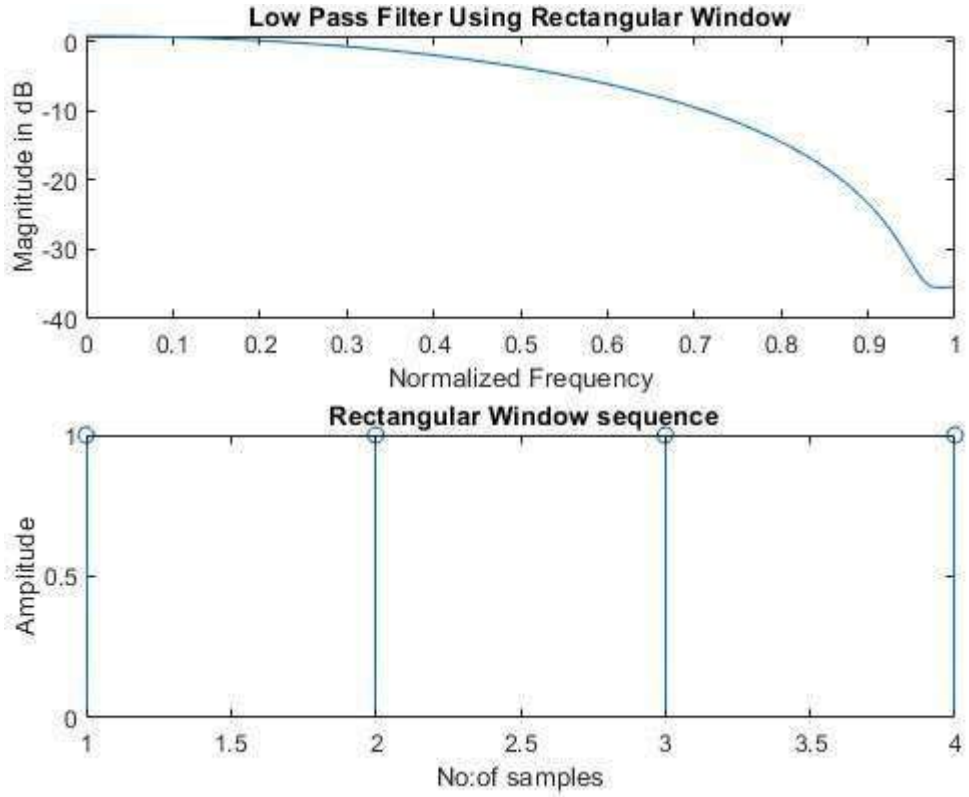
where $\omega_1 = 2\pi f_{c1}/f_s$ and $\omega_2 = 2\pi f_{c2}/f_s$.

A band-stop filter, also known as a band-rejection or notch filter, is a filter that passes most frequencies while attenuating a specific range of frequencies to very low levels. The range of frequencies that a band-stop filter attenuates is called the stopband. We can create a band-stop filter by connecting a low-pass filter in parallel with a high-pass filter. This combination is often called a “Twin-T” filter. A notch filter is a type of band-stop filter with a narrow stopband.

- Purpose: A band-stop filter attenuates frequencies within a specific range [fc1, fc2] while allowing frequencies outside this range to pass through.
- Ideal Frequency Response: The ideal frequency response

$$H_{BSF}(e^{j\omega}) = \begin{cases} 0 & \text{if } \omega_1 \leq |\omega| \leq \omega_2 \\ 1 & \text{otherwise} \end{cases}$$

Output

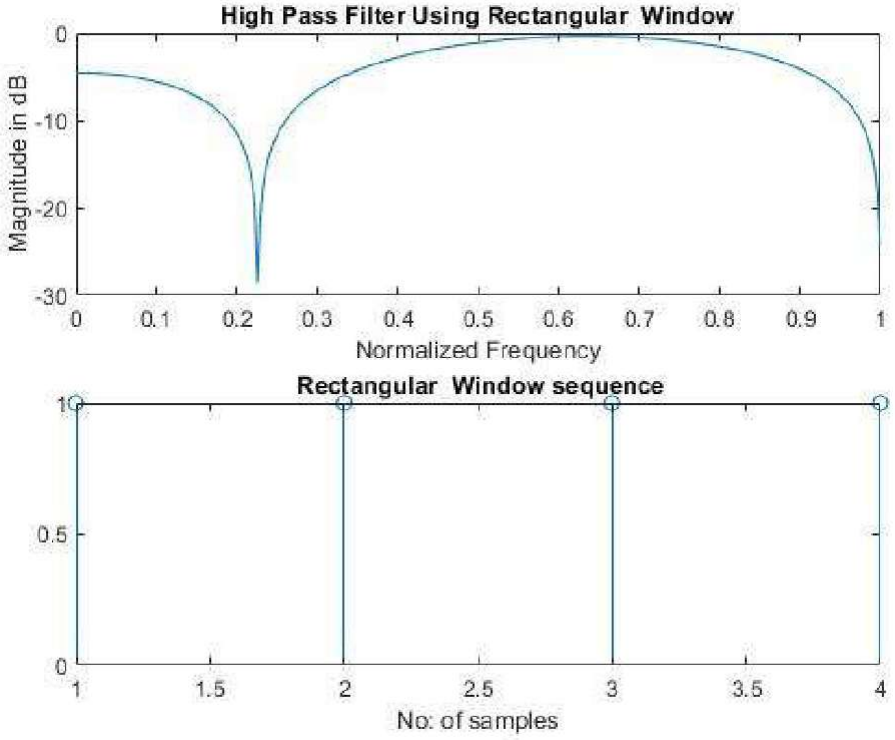

Low Pass Filter Using Rectangular Window
Magnitude in dB
0
-10
-20
-30
-40
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Normalized Frequency
Rectangular Window sequence
Amplitude
1
0.5
0
1
1.5
2
2.5
3
3.5
4
No:of samples

Program

Low pass filter :

```
clc;
clear all;
wc=0.5*pi;
N=input('enter the value of N');
alpha=(N-1)/2;
eps=0.001;
n=0:1:N-1;
hd=sin(wc*(n-alpha+eps))./(pi*(n-alpha+eps));
wr=boxcar(N);
hn=hd.*wr';
w=0:0.01:pi;
h=freqz(hn,1,w);

subplot(2,1,1);
plot(w/pi,10*log10(abs(h)));
title('Low Pass Filter Using Rectangular Window');
xlabel('Normalized Frequency');
ylabel('Magnitude in dB');\

subplot(2,1,2);
stem(wr);
title(' Rectangular Window sequence');
xlabel('No:of samples');
ylabel('Amplitude');

N=input('enter the value of N');

wc=0.5*pi;
alpha=(N-1)/2;
eps=0.001;
n=0:1:N-1;
hd=(sin(pi*(n-alpha+eps))-sin(wc*(n-alpha+eps)))./(pi*(n-alpha+eps));
wr=boxcar(N);
hn=hd.*wr';
w=0:0.01:pi
h=freqz(hn,1,w);
subplot(2,1,1);
plot(w/pi,10*log10(abs(h)));
title('High Pass Filter Using Rectangular Window');
xlabel('Normalized Frequency');
ylabel('Magnitude in dB');
subplot(2,1,2);
stem(wr);
title(' Rectangular Window sequence');
xlabel('No: of samples');ylabel('Amplitude);
```

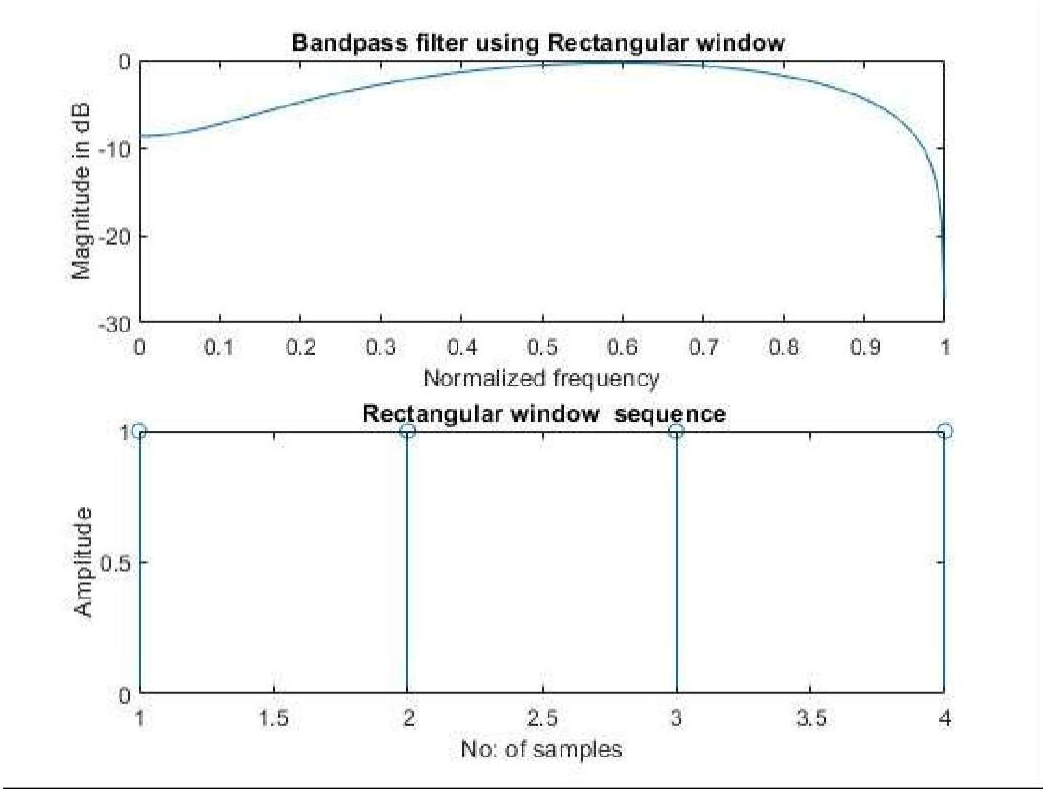
Output


High Pass Filter Using Rectangular Window
Magnitude in dB
0
-10
-20
-30
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Normalized Frequency
Rectangular Window sequence
1
0.5
0
1
1.5
2
2.5
3
3.5
4
No: of samples

High pass filter:

```
N=input('enter the value of N');
wc=0.5*pi;
alpha=(N-1)/2;
eps=0.001;
n=0:1:N-1;
 hd=(sin(pi*(n-alpha+eps))-sin(wc*(n-alpha+eps)))./(pi*(n-alpha+eps)); wr=boxcar(N);
 hn=hd.*wr';
w=0:0.01:pi
h=freqz(hn,1,w);

 subplot(2,1,1);
 plot(w/pi,10*log10(abs(h)));
title('High Pass Filter Using Rectangular Window');xlabel('Normalized
Frequency');
 ylabel('Magnitude in dB');

 subplot(2,1,2);
stem(wr);
 title(' Rectangular Window sequence');
xlabel('No: of samples'); ylabel('Amplitude);
```

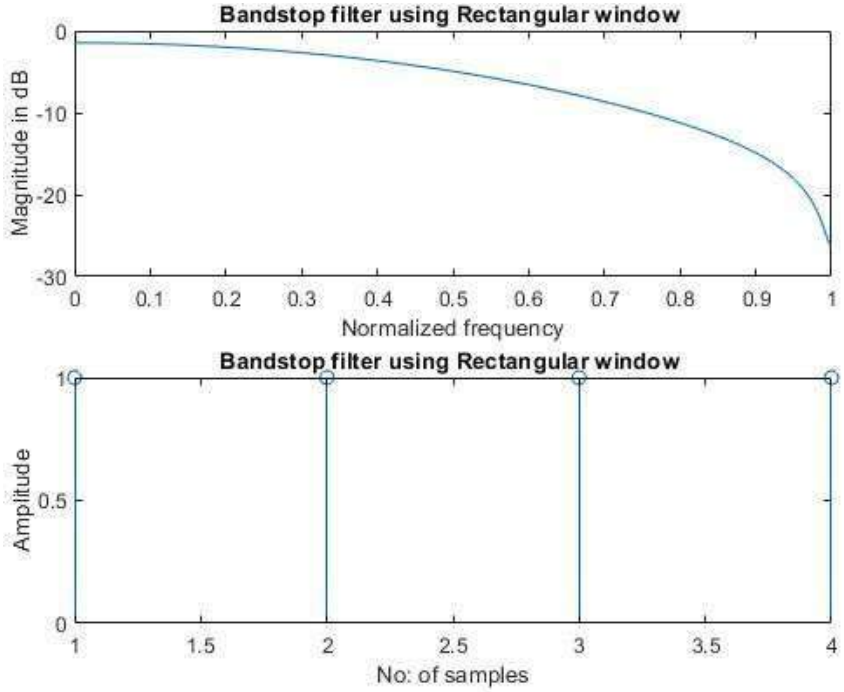
Output


Bandpass filter using Rectangular window
Magnitude in dB
0
-10
-20
-30
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Normalized frequency
Rectangular window sequence
Amplitude
1
0.5
0
1
1.5
2
2.5
3
3.5
4
No: of samples

Band pass filter :

```
N=input('enter the value of N');
wc1=0.25*pi;
wc2=0.75*pi;
alpha=(N-1)/2;
eps=0.001;
n=0:1:N-1;
hd=(sin(wc1*(n-alpha+eps))-sin(wc2*(n-alpha+eps))./(pi*(n-alpha+eps));
 wr=boxcar(N);
 hn=hd.*wr';
 w=0:0.01:pi;
h=freqz(hn,1,w);

 subplot(2,1,1);
plot(w/pi,10*log10(abs(h)));
title('Bandpass filter using Rectangular window ');
 xlabel('Normalized frequency');
 ylabel('Magnitude in dB');

 subplot(2,1,2);
 stem(wr);
 title(' Rectangular window sequence');
 xlabel('No: of samples');
 ylabel('Amplitude');
```

Output


Bandstop filter using Rectangular window
Magnitude in dB
0
-10
-20
-30
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
Normalized frequency
Bandstop filter using Rectangular window
Amplitude
1
0.5
0
1
1.5
2
2.5
3
3.5
4
No: of samples

Band stop filter:

```
N=input('enter the value of N');
wc1=0.25*pi;
wc2=0.75*pi;
alpha=(N-1)/2;
eps=0.001;
n=0:1:N-1;
 hd=(sin(wc1*(n-alpha+eps))-sin(wc2*(n-alpha+eps))+
sin(pi*(n-alpha+eps))).(pi*(n- alpha+eps));
wr=boxcar(N);
 hn=hd.*wr';
 w=0:0.01:pi;
h=freqz(hn,1,w);

subplot(2,1,1);
plot(w/pi,10*log10(abs(h)));
title('Bandstop filter using Rectangular window ');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');

subplot(2,1,2);
stem(wr);
title('Bandstop filter using Rectangular window ');
xlabel('No: of samples');
ylabel('Amplitude');
```

Result

Implemented a low pass filter, high pass filter, bandpass filter, and band reject filter using rectangular window.