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 Sri. **NANDANA S BABU** of **5th** Semester class (Roll No. **[B22ECB52] 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 : **TKM22EC096**

Staff Member in-charge *External Examiner*

Date:

INDEX

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

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
- **Mathematically $\delta(t)= \infty; t=0$ and $0; t \neq 0$**

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)*** **=*t***;***t***≥**0** and**0**;***t***<**0**

5.<u>Triangular Signal:</u>

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

6.<u>Sine Signal:</u>

- A continuous periodic signal. It oscillates smoothly between -1 and 1.
- **Mathematically: y(t)=Asin(2πft)**

7. <u>Cosine Signal:</u>

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

8. <u>Exponential Signal:</u>

- 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. <u>Unit Step Signal:</u>

- A signal that is zero for all negative time values and one for positive time values.
- **Mathematically** ***u(t)*** **=1**;***t***≥**0** and**0**;***t***<**0**

<u>Program:</u>

```
clc;

clear all;

close all;


% 1. unit impulse signal

t1=-5:1:5;

y1= [zeros (1,5), ones (1,1), zeros(1,5)];

subplot (3,3,1);
```

```
hold on;
stem (t1, y1);
xlabel('time');
ylabel('amplitude');
title('unit impulse signal');
grid on;
hold off;

% 2. biploar pulse
t2=0:0.01:1;
f2=5;
y2=square(2*pi*f2*t2);
subplot(3,3,2)
plot(t2,y2);
hold on;
stem(t2,y2);
xlabel('time');
ylabel('amplitide');
title('biploar pulse');
grid on;
axis([0 1 -2 2]);
legend('continuous','discrete');
hold off;

% 3. unipolar pulse
t3=0:0.01:1;
f3=5;
y3=sqrt(square(2*pi*f3*t3));
subplot(3,3,3);
plot(t3,y3);
hold on;
```

```
stem(t3,y3);
xlabel('time');
ylabel('amplitude');
title('unipolar pulse');
grid on;
axis([0 1 -2 2]);
legend('continuous','discrete','Location','best');
hold off;

% 4. ramp signal
t4=0:1:5;
y4=t4;
subplot(3,3,4);
plot(t4,y4);
hold on;
stem(t4,y4);
xlabel('time');
ylabel('amplitude');
title('ramp signal');
legend('continuous','discrete','best');
hold off;

% 5. triangular signal
f5=10;
t5=0:0.025:1;
y5=sin(2*pi*f5*t5);
subplot(3,3,5);
plot(t5,y5);
hold on;
stem(t5,y5);
axis([0 1 -2 2]);
```

```
xlabel('time');
ylabel('amplitude');
title('triangular signal');
legend('continuous','discrete','best');
hold off;
grid on;

% 6. sine wave
t6=0:0.01:1;
f6=5;
y6=sin(2*pi*f6*t6);
subplot(3,3,6);
hold on;
plot(t6,y6);
stem(t6,y6);
grid on;
xlabel('time');
ylabel('amplitude');
title('sine wave');
legend('continuous','discrete');
hold off;

% 7. cosine wave
t7=0:0.01:1;
f7=5;
y7=cos(2*pi*f7*t7);
subplot(3,3,7);
plot(t7,y7);
hold on;
stem(t7,y7);
grid on;
```

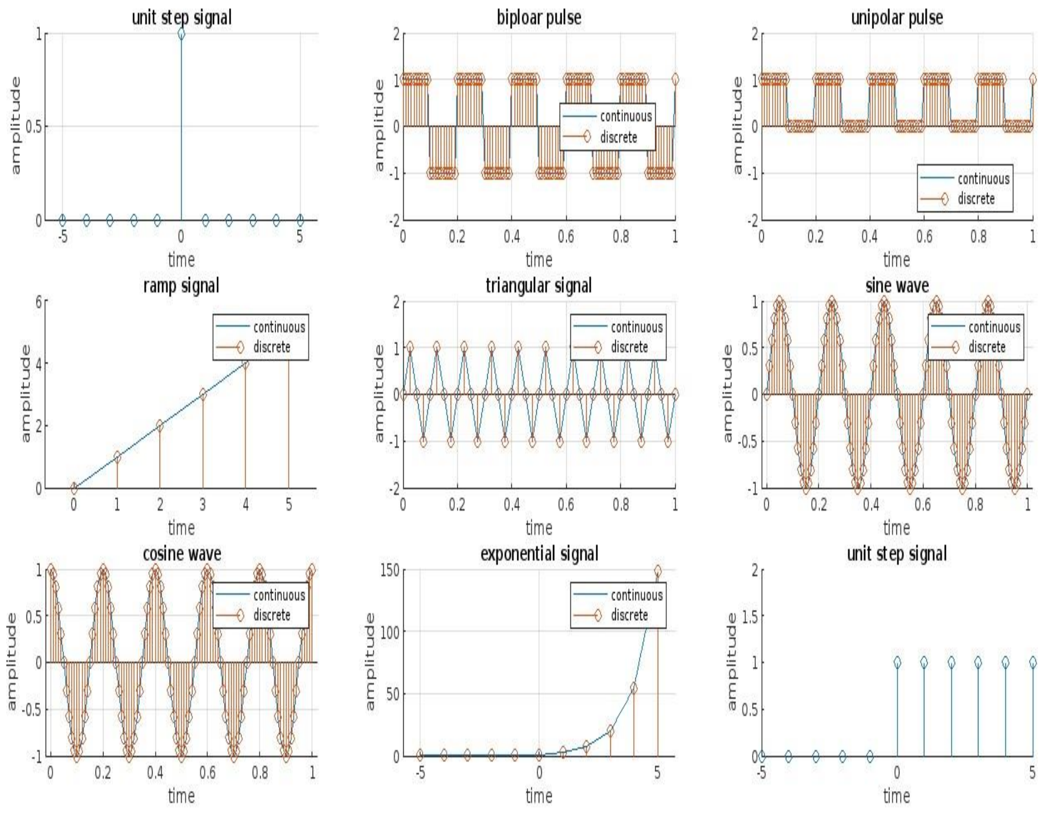
```
xlabel('time');

ylabel('amplitude');
title('cosine wave');
legend('continuous','discrete');
grid on;
hold off;

% 8. exponential signal
t8=-5:1:5;
y8=exp(t8);
subplot(3,3,8);
plot(t8,y8);
hold on;
stem(t8,y8);
xlabel('time');
ylabel('amplitude');
grid on;
legend('continuous','discrete')
title('exponential signal');
hold off;

% 9. unit step signal
t9=-5:1:5;
y9=[zeros(1,5),ones(1,6)];
subplot(3,3,9);
hold on;
stem(t9,y9);
xlabel('time');
ylabel('amplitude');
axis([-5 5 0 2]);
```

```
title('unit step signal');
hold off;
```

Result:

Generated and verified various continuous and discrete waveforms of basic test signals.

Observation:



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

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.

fs ≥ 2·fmax

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;
f=input("Enter frequency of sine wave in Hz:");
t1= 0:0.01:1;
y1=sin(2*pi*f*t1);
subplot(2,2,1);
hold on;
plot(t1,y1);
xlabel('time');
ylabel('amplitide');
```

```
title('Continuous Signal');
grid on;
hold off;

% undersampling
fs1=0.5*f;
t2=0:1/fs1:1;
y2=sin(2*pi*f*t2);
subplot(2,2,2);
plot(t2,y2);
hold on;
stem(t2,y2);
legend('continuous','discrete');
grid on;
xlabel('time');
ylabel('amplitude');
title('Under sampled signal');
hold off;

% Nyquist sampling
fs2=3*f;
t3=0:1/fs2:1;
y3=sin(2*pi*f*t3);
subplot(2,2,3);
plot(t3,y3);
hold on;
stem(t3,y3);
legend('continuous','discrete');
grid on;
xlabel('time');
ylabel('amplitude');
```

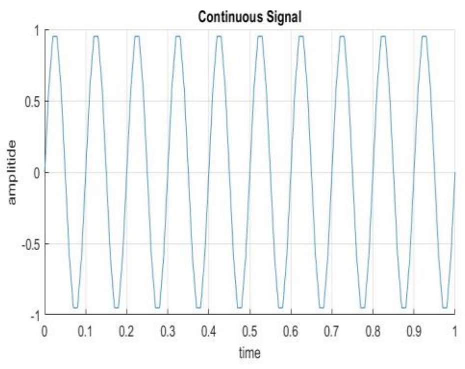
```
title('Nyquist sampled signal');
hold off;

% over sampling
fs3=10*f;
t4=0:1/fs3:1;
y4=sin(2*pi*f*t4);
subplot(2,2,4);
plot(t4,y4);
hold on;
stem(t4,y4);
legend('continuous','discrete');
grid on;
xlabel('time');
ylabel('amplitude');
title('Over sampled signal');
hold off;
```

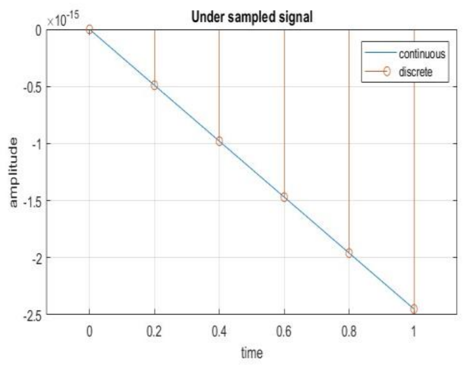
Result :

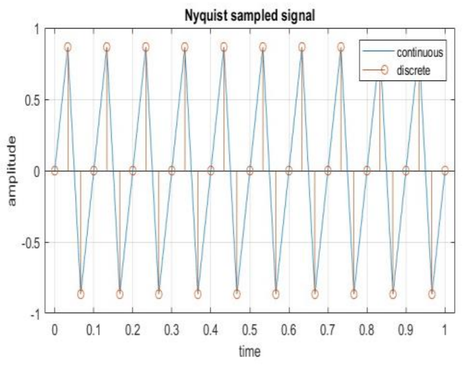
Verified Sampling Theorem using MATLAB.

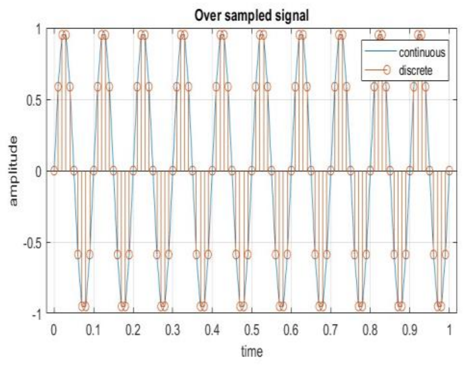
Observation:

Enter frequency of sine wave in Hz:10









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

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:

y(t) = x(t) * h(t) = ∫**x(τ)h(t−τ) dτ**∞−∞

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:

a)Using built-in function

```
clc;

clear all;

close all;


x=input('enter first sequence:');

h=input("enter second sequence:");

lx=length(x);

lh=length(h);

lt=lx+lh-1;

sx=input("enter starting index of first sequence:");

sh=input("enter starting index of second sequence:");
```

```
rx=sx:1:lx+sx-1;
rh=sh:1:lh+sh-1;
rt=sx+sh:1:lt+sx+sh-1;
y=conv(x,h);
disp(" convoluted output:")
disp(y);

%ploting
subplot(1,3,1);
stem(rx,x);
xlabel('n');
ylabel('amplitude');
grid on;
title('x(n)');

subplot(1,3,2);
stem(rh,h);
xlabel('n');
ylabel('amplitude');
grid on;
title('h(n)');

subplot(1,3,3);
stem(rt,y);
xlabel('n');
ylabel('amplitude');
grid on;
title('y(n)');
```

b)Without using built-in function

```
clc;
```

```
clear all;
close all;

x=input('enter first sequence:');
h=input("enter second sequence:");
lx=length(x);
lh=length(h);
lt=lx+lh-1;
sx=input("enter starting index of first sequence:");
sh=input("enter starting index of second sequence:");
rx=sx:1:lx+sx-1;
rh=sh:1:lh+sh-1;
rt=sx+sh:1:lt+sx+sh-1;
y=[zeros(1,lt)];
for i=1:lx
    for j=1:lh;
        y(i+j-1)=y(i+j-1)+x(i)*h(j);
    end
end
disp(“convoluted output:”);
disp(y);
%ploting
subplot(1,3,1);
stem(rx,x);
xlabel('n');
ylabel('amplitude');
grid on;
title('x(n)');

subplot(1,3,2);
stem(rh,h);
```

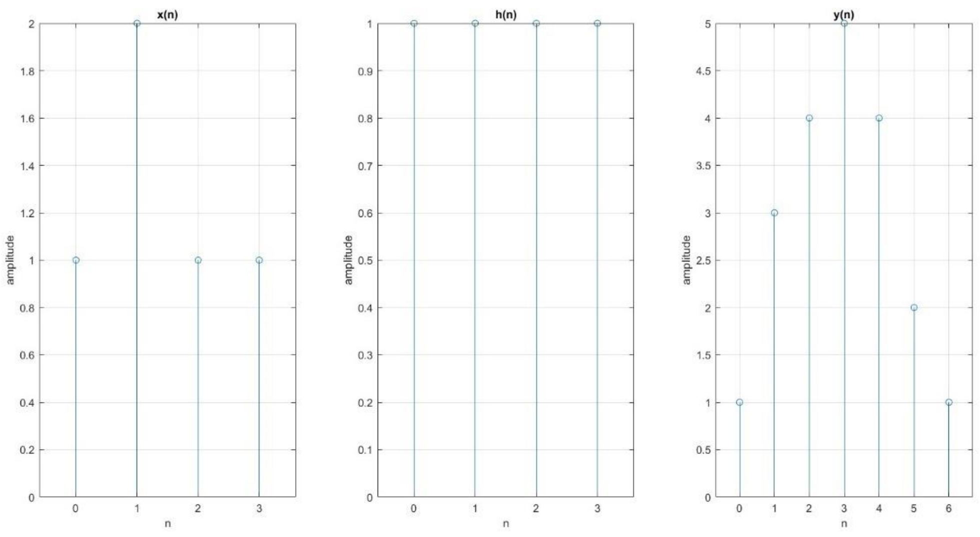
```
xlabel('n');
ylabel('amplitude');
grid on;
title('h(n)');

subplot(1,3,3);
stem(rt,y);
xlabel('n');
ylabel('amplitude');
grid on;
title('y(n)');
```

<u>Result:</u>

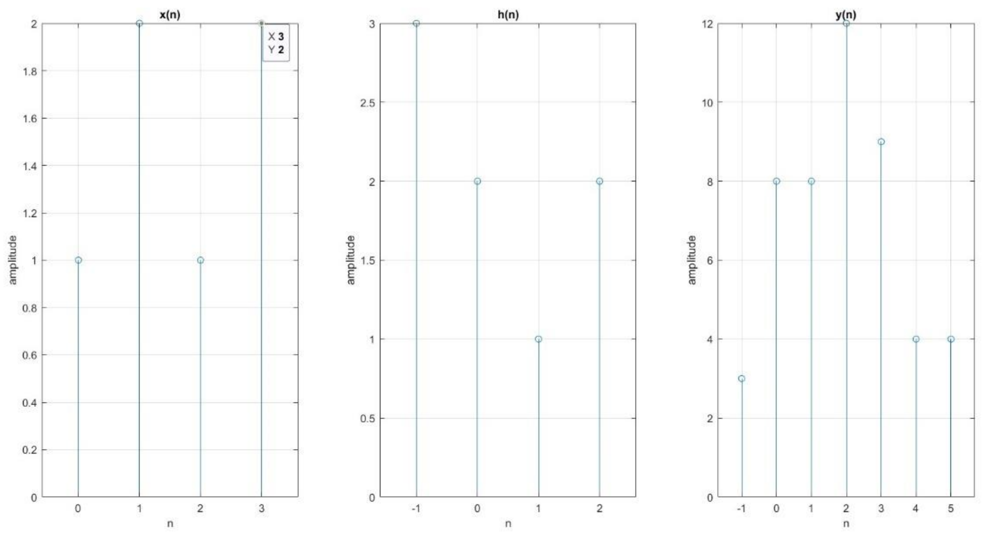
Performed Linear Convolution using with and without built-in function.

Observation:

1.
```
enter first sequence:[1 2 1 1]
enter second sequence:[1 1 1 1]
enter starting index of first sequence:0
enter starting index of second sequence:0
convoluted output:

1    3    4    5    4    2    1
```



2.
```
enter first sequence:[1 2 1 2]
enter second sequence:[3 2 1 2]
enter starting index of first sequence:0
 enter starting index of second sequence:-1


convoluted output:
3    8    8   12    9    4    4
```


x(n)
X 3
Y 2
amplitude
n
h(n)
amplitude
n
y(n)
amplitude
n

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

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] ⊛ h[n]) = Σ_{k=0} ^{N-1} x[k]h[(n-k) mod 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;

clear all;

close all;

x=input('enter first sequence:');

h=input("enter second sequence:");

lx=length(x);

lh=length(h);

k=max(lx,lh);

sx=input("enter starting index of first sequence:");

sh=input("enter starting index of second sequence:");

rx=sx:1:lx+sx-1;
```

```
rh=sh:1:lh+sh-1;
rk=sx+sh:1:k+sx+sh-1;
X_K=fft(x);
H_K=fft(h);
Y_K=X_K.*H_K;
y=ifft(Y_K);
disp("output:");
disp(y);
%ploting
subplot(1,3,1);
stem(rx,x);
xlabel('n');
ylabel('amplitude');
grid on;
title('x(n)');
subplot(1,3,2);
stem(rh,h);
xlabel('n');
ylabel('amplitude');
grid on;
title('h(n)');
subplot(1,3,3);
stem(rk,y);
xlabel('n');
ylabel('amplitude');
grid on;
title('circular convoluted output');
```

b)<u>Using Concentric Circle Method</u>

```
clc;
clear all;
```

```
close all;
x=input('enter first sequence:');
h=input("enter second sequence:");
lx=length(x);
lh=length(h);
k=max(lx,lh);
sx=input("enter starting index of first sequence:");
sh=input("enter starting index of second sequence:");
rx=sx:1:lx+sx-1;
rh=sh:1:lh+sh-1;
rk=sx+sh:1:k+sx+sh-1;
x=x(:,end:-1:1);
for i=1:lx
    x=[x(end) x(1:end-1)];
    h1=h;
    y(i)=sum(x.*h1);
end
disp("output:");
disp(y);
%ploting
subplot(1,3,1);
stem(rx,x);
xlabel('n');
ylabel('amplitude');
grid on;
title('x(n)');
subplot(1,3,2);
stem(rh,h);
xlabel('n');
ylabel('amplitude');
grid on;
```

```
title('h(n)');
subplot(1,3,3);
stem(rk,y);
xlabel('n');
ylabel('amplitude');
grid on;
title('circular convoluted output');
```

c)<u>Using Matrix Method</u>

```
clc;
clear all;
close all;
x=input('enter first sequence:');
h=input("enter second sequence:");
lx=length(x);
lh=length(h);
k=max(lx,lh);
sx=input("enter starting index of first sequence:");
sh=input("enter starting index of second sequence:");
rx=sx:1:lx+sx-1;
rh=sh:1:lh+sh-1;
rk=sx+sh:1:k+sx+sh-1;
x=x(:,end:-1:1);
m=[];
for i=1:lx
    x=[x(end) x(1:end-1)];
    m=[m;x];
end
y=m*h';
disp("output:");
disp(y);
```

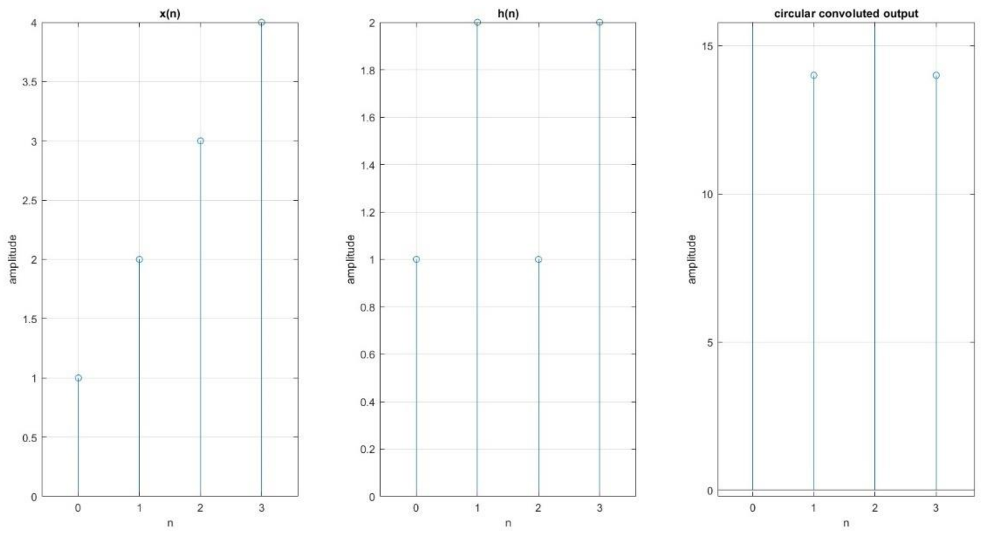
```
%ploting
subplot(1,3,1);
stem(rx,x);
xlabel('n');
ylabel('amplitude');
grid on;
title('x(n)');
subplot(1,3,2);
stem(rh,h);
xlabel('n');
ylabel('amplitude');
grid on;
title('h(n)');
subplot(1,3,3);
stem(rk,y);
xlabel('n');
ylabel('amplitude');
grid on;
title('circular convoluted output');
```

Result:

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

Observation:

```
enter first sequence:[1 2 3 4]

enter second sequence:[1 2 1 2]

enter starting index of first sequence:0

enter starting index of second sequence:0

output:

16  14  16  14
```



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

Linear Convolution using Circular Convolution 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:

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

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

Program:

1. Linear Convolution using Circular Convolution.

```
clc;
clear all;
close all;
x=[1 2 3 4];
h=[1 1 1];
lx=length(x);
lh=length(h);
lt=lx+lh-1;
x=[x zeros(1,lh-1)];
h=[h zeros(1,lx-1)];
x_k=fft(x);
h_k=fft(h);
```

```
y_k=x_k.*h_k;
y=ifft(y_k);
disp('Linear Convolution using Circular Convolution output');
disp(y);

%verification
x=[1 2 3 4];
h=[1 1 1];
y=conv(x,h);
disp('Verification');
disp(y);
```

2. Circular Convolution using Linear Convolution.

```
clc;
clear all;
close all;

x=[1 2 3 4];
h=[1 1 1];
lx=length(x);
lh=length(h);
lt=max(lx,lh);
k=lx+lh-1;
y=conv(x,h);
disp(y);
for i=1:k-lt
    y(i)=y(i)+y(lt+i);
end
for i=1:lt
    res=y(i);
end
disp(y);


% Verification using FFT-based circular convolution
x = [1 2 3 4];
h = [1 1 1];
l = length(x);
m = length(h);

% Pad h with zeros to make it the same length as x
h = [h, zeros(1, l - m)];
X_K = fft(x);
H_K = fft(h);
Y_K = X_K .* H_K;
y_fft = ifft(Y_K);
```

```
disp('Verification (Circular Convolution using FFT)');
disp(y_fft);
```

Result:

Performed a) Linear Convolution using Circular Convolution; b) Circular Convolution using Linear Convolution and verified result.

Observation:

1. Linear Convolution using Circular Convolution output

   ```
    1    3    6    9    7    4
   ```

   Verification

   ```
   1    3    6    9    7    4
   ```

2. Circular convolution using Linear Convolution output

   ```
   8    7    6    9
   ```

   Verification (Circular Convolution using FFT)

   ```
     8    7    6    9
   ```

Experiment No: 6   Date: 01/10/24

DFT AND IDFT

Aim:

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

2.IDFT using inbuilt function, without using inbuilt function, and twiddle factor.

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.

The twiddle factor is a complex number that is used in the Cooley-Tukey algorithm, a fast Fourier transform (FFT) algorithm. It is defined as:

W_N^k = exp(-j * 2 * pi * k / N)

- The twiddle factor represents a rotation in the complex plane by an angle of $2pi$k/N radians. It is used to combine the results of the smaller FFTs that are computed in the Cooley-Tukey algorithm to obtain the final FFT result.

Application

- Spectrum (Analysis)
- Filtering
- Compression
- Modulation
- Convolution
- Demodulation
- 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)];
X1=zeros(1,N);
for k=0:N-1
    for n=0:N-1
        X1(k+1)=X1(k+1)+x(n+1)*exp(-1j*2*pi*n*k/N);
    end
end
X2 = zeros(N,1);
T = zeros(N, N);
for k = 0:N-1
    for n = 0:N-1
        T(k+1, n+1) = exp(-1i * 2 * pi * k * n / N);
    end
end
X2=T*x';
disp('Using built-in function');
disp(fft(x));
disp('Without using built-in function');
disp(X1);
disp('Using twiddle factor');
disp(X2);
%plotting
k=0:N-1;
```

```
magX=abs(X1);
phaseX=angle(X1);
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)];
x1=zeros(N,1);
for k=0:N-1
    for n=0:N-1
        x1(n+1)=x1(n+1)+X(k+1)*exp(1j*2*pi*n*k/N);
    end
end
x1=1/N.*x1;
x2 = zeros(N,1);
T = zeros(N, N);
for k = 0:N-1
```

```
    for n = 0:N-1
        T(k+1, n+1) = exp(1i * 2 * pi * k * n / N);
    end
end
x2=T*X';
x2=(1/N).*x2;
disp('Without Using built in function');
disp(x1);
%verification
disp('Using built in function');
disp(ifft(X));
disp('Using Twiddle factor');
disp(x2);
```

Result:

Performed

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

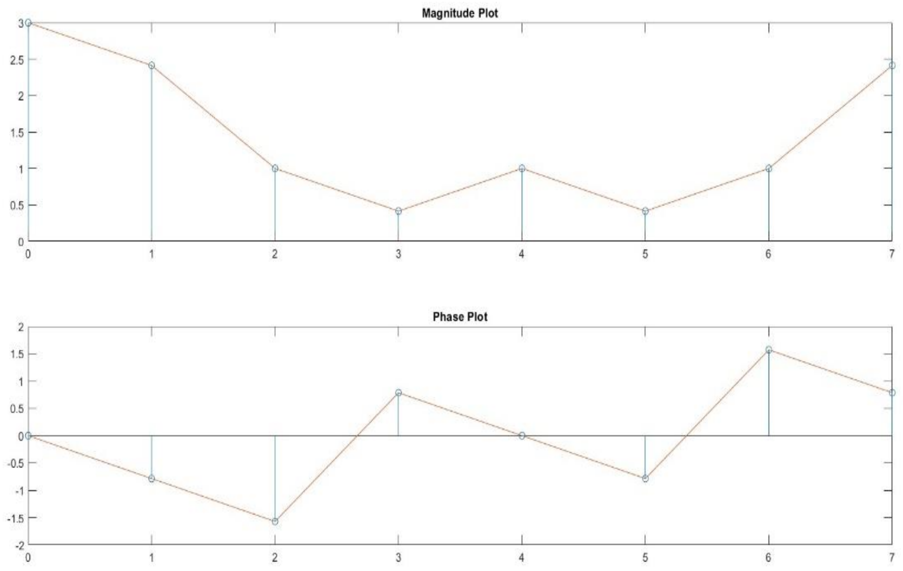
2)IDFT using inbuilt function, without using inbuilt function and twiddle factor.

and verified the result.

Observations:

1.DFT

```
enter sequence:[1 1 1 0]

enter sequence:[1 1 1 0]

enter the N point:8

Using built-in function

  Columns 1 through 3

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

  Columns 4 through 6

   0.2929 + 0.2929i   1.0000 + 0.0000i   0.2929 - 0.2929i

  Columns 7 through 8

   0.0000 + 1.0000i   1.7071 + 1.7071i

Without using built-in function

  Columns 1 through 3

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

  Columns 4 through 6

   0.2929 + 0.2929i   1.0000 + 0.0000i   0.2929 - 0.2929i

  Columns 7 through 8

  -0.0000 + 1.0000i   1.7071 + 1.7071i

Using twiddle factor

   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

   1.7071 + 1.7071i
```



2.IDFT

```
enter sequence:[ 3 -i 1 i]

enter the n point:4

Without Using built in function

   1.0000 + 0.0000i

   1.0000 - 0.0000i

   1.0000 - 0.0000i

   0.0000 + 0.0000i


Using built in function

     1     1     1     0


Using Twiddle factor

   1.0000 + 0.0000i

   1.0000 - 0.0000i

   1.0000 - 0.0000i

   0.0000 + 0.0000i
```

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:

DFT{a·x1[n]+b·x2[n]} = a·DFT{x1[n]}+b·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:

DFT{x1[n]⊛x2[n]}=DFT{x1[n]}·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:

DFT{x1[n]·x2[n]} = $\frac{1}{N}$DFT{x1[n]}⊛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;
% User input for the input sequence
x = input('Enter the input sequence x : ');
% User input for the impulse response
h = input('Enter the impulse response h : ');
% Section length for overlap-save
L = length(h);  % Length of impulse response
% Initialization
N = length(x);
M = length(h);
% Pad input x with zeros
x_padded = [x, zeros(1, L - 1)];
% Prepare the output array
y = zeros(1, N + M +1);
% Calculate the number of sections
num_sections = (N + L - 1) / L;  % Calculate number of sections
% Process sections
for n = 0:num_sections-1
    % Determine the current section
    start_idx = n * L + 1;
    end_idx = start_idx + L - 1;
    % Ensure the section does not exceed the bounds
    x_section = x_padded(start_idx:min(end_idx, end));
    % Convolution
    conv_result = conv(x_section, h);
    % Save the results to the output
    y(start_idx:start_idx + length(conv_result) - 1) =y(start_idx:start_idx +
length(conv_result) - 1) + conv_result;

end

% Trim the output to the valid part

y = y(1:N + M - 1);

% Compare with built-in convolution

y_builtin = conv(x, h);

% Display results

disp('Overlap-add convolution result:');

disp(y);

disp('Built-in convolution result:');

disp(y_builtin);

% Plotting results

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;

% Input the sequences and block size

x = input("Enter 1st sequence: ");

h = input("Enter 2nd sequence: ");

N = input("Fragmented block size: ");
```

```
% Call the overlap-save function
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
    % Zero-padding for input and filter sequences
    x = [zeros(1, M1), x, zeros(1, N-1)];
    h = [h, zeros(1, N - M)];
    % Number of blocks
    K = floor((Nx + M1 - 1) / L);
    % Initialize the output matrix Y
    Y = zeros(K + 1, N);
    for k = 0:K
        xk = x(k*L + 1 : k*L + N);  % Extract block of input sequence
        Y(k+1, :) = cconv(xk, h, N);  % Circular convolution
    end
    Y = Y(:, M:N)';
    y = (Y(:))';
end
```

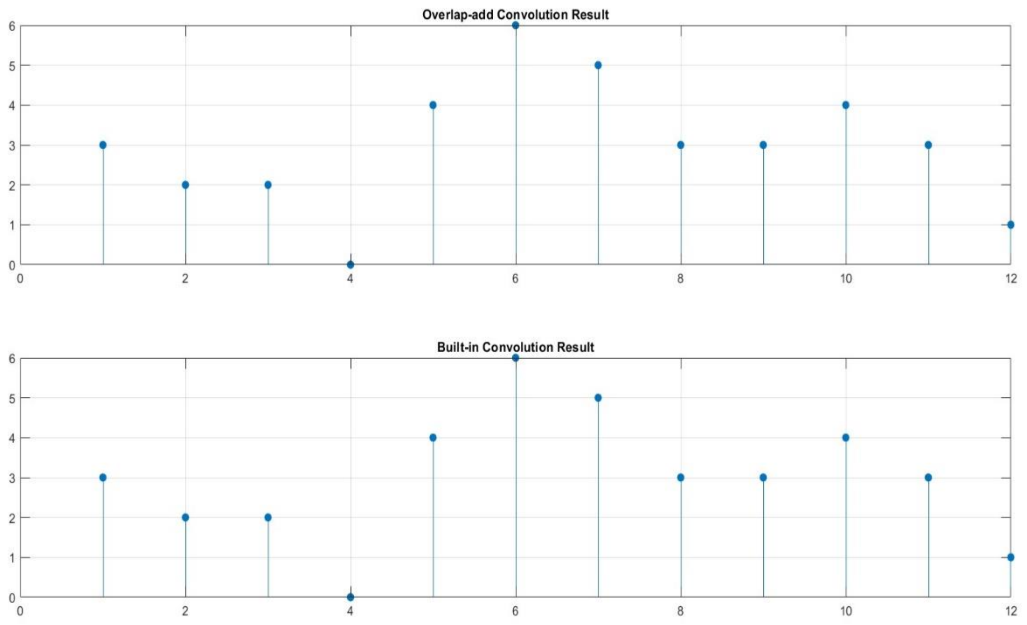
Result:

Performed Overlap Add and Overlap Save methods 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
```

IMPLEMENTATION OF FIR FILTERS

Aim:

Implement various FIR filters using different windows

1. Low Pass Filter
2. High Pass Filter
3. Band pass Filter
4. Band stop Filter

Theory:

Design of FIR Filters Using Window Methods

In FIR (Finite Impulse Response) filter design, the goal is to create a filter with specific frequency response characteristics, such as low-pass, high-pass, band-pass, or band-stop. Using window methods, we can shape the filter response by applying a window function to an ideal filter impulse response.

Step 1: Define the Ideal Impulse Response

The ideal impulse response, h_ideal(n), of a low-pass filter with a cutoff frequency f_c is given by:

h_ideal(n) = sin(2 * pi * f_c * (n - (N - 1) / 2)) / (pi * (n - (N - 1) / 2))

Where:

•f_c: Normalized cut off frequency
• N: Filter length
• n: Sample index

Step 2: Select an Appropriate Window Function

The choice of window affects the trade-off between the main lobe width and the sidelobe levels. Common windows include the Rectangular, Hamming, Hanning, Blackman, and Kaiser windows.

Window Type	Formula
Rectangular	w(n) = 1
Triangular	w(n) = 1 - 2*abs(n) / (N - 1)
Hamming	w(n) = 0.54 - 0.46 * cos(2 * pi * n / (N - 1))
Hanning	w(n) = 0.5 * (1 - cos(2 * pi * n / (N - 1)))
Blackman	w(n) = 0.42 - 0.5 * cos(2 * pi * n / (N - 1)) + 0.08 * cos(4 * pi * n / (N - 1))
Kaiser	w(n) = I0(beta * sqrt(1 - (2 * n / (N - 1) - 1)^2)) / I0(beta)

Step 3: Apply the Window to the Ideal Impulse Response

The windowed impulse response is computed as:

h(n) = h_ideal(n) * w(n)

Step 4: Construct the FIR Filter

The final impulse response h(n) defines the FIR filter coefficients that can be used in filtering algorithms.

Advantages and Disadvantages of Window-Based FIR Design

Advantages:
• Simplicity: Windowing is straightforward and does not require iterative optimization.
• Control over Leakage: Different windows provide different control over sidelobes and main lobe width.

Disadvantages:
• Fixed Frequency Response: Once the window is chosen, the frequency response characteristics are determined.
• Trade-Off Limitations: Some applications require specific frequency responses that cannot be perfectly achieved using standard windows.

Program:

1. Low Pass Filter

```
clc;

clear all;

close 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);

wt=bartlett(N);

wh=hamming(N);

whn=hanning(N);

hn1 = hd.*wr';

hn2 = hd.*wt';

hn3 = hd.*wh';

hn4 = hd.*whn';

w = 0:0.01:pi;

h1 = freqz(hn1,1,w);

h2 = freqz(hn2,1,w);

h3 = freqz(hn3,1,w);

h4 = freqz(hn4,1,w);

subplot(3,3,1);
```

```
plot(w/pi,10*log10(abs(h1)));
title('low pass filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,3);
plot(w/pi,10*log10(abs(h2)));
title('low pass filter using triangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,4);
stem(wt);
title('Triangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,5);
plot(w/pi,10*log10(abs(h3)));
title('low pass filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,6);
stem(wh);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,7);
plot(w/pi,10*log10(abs(h4)));
```

```
title('low pass filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,8);
stem(whn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

2.High Pass Filter

```
clc;
clear all;
close 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(pi*(n-alpha+eps))-sin(wc*(n-alpha+eps)))./(pi*(n-alpha+eps));
wr = boxcar(N);
wt=bartlett(N);
wh=hamming(N);
whn=hanning(N);
hn1 = hd.*wr';
hn2 = hd.*wt';
hn3 = hd.*wh';
hn4 = hd.*whn';
w = 0:0.01:pi;
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
h3 = freqz(hn3,1,w);
```

```
h4 = freqz(hn4,1,w);
subplot(3,3,1);
plot(w/pi,10*log10(abs(h1)));
title('high pass filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,3);
plot(w/pi,10*log10(abs(h2)));
title('high pass filter using triangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,4);
stem(wt);
title('Triangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,5);
plot(w/pi,10*log10(abs(h3)));
title('high pass filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,6);
stem(wh);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

```
subplot(3,3,7);
plot(w/pi,10*log10(abs(h4)));
title('high pass filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,8);
stem(whn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

3.Band Pass Filter

```
clc;
clear all;
close all;
wc1=0.5*pi;
wc2=0.9*pi;
N = input('Enter the value of N=');
alpha = (N-1)/2;
eps = 0.001;
n = 0:1:N-1;
hd = (sin(wc2*(n-alpha+eps))-sin(wc1*(n-alpha+eps)))./(pi*(n-alpha+eps));
wr = boxcar(N);
wt=bartlett(N);
wh=hamming(N);
whn=hanning(N);
hn1 = hd.*wr';
hn2 = hd.*wt';
hn3 = hd.*wh';
hn4 = hd.*whn';
w = 0:0.01:pi;
```

```
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
h3 = freqz(hn3,1,w);
h4 = freqz(hn4,1,w);
subplot(3,3,1);
plot(w/pi,10*log10(abs(h1)));
title('band pass filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,3);
plot(w/pi,10*log10(abs(h2)));
title('band pass filter using triangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,4);
stem(wt);
title('Triangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,5);
plot(w/pi,10*log10(abs(h3)));
title('band pass filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,6);
stem(wh);
```

```
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,7);
plot(w/pi,10*log10(abs(h4)));
title('band pass filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,8);
stem(whn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

4.Band Stop Filter

```
 clc;
clear all;
close all;
wc1=0.5*pi;
wc2=0.9*pi;
N = input('Enter the value of N=');
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)))./(pi*(n-alpha+eps));
wr = boxcar(N);
wt=bartlett(N);
wh=hamming(N);
whn=hanning(N);
hn1 = hd.*wr';
```

```
hn2 = hd.*wt';
hn3 = hd.*wh';
hn4 = hd.*whn';
w = 0:0.01:pi;
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
h3 = freqz(hn3,1,w);
h4 = freqz(hn4,1,w);
subplot(3,3,1);
plot(w/pi,10*log10(abs(h1)));
title('band stop filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,3);
plot(w/pi,10*log10(abs(h2)));
title('band stop filter using triangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,4);
stem(wt);
title('Triangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,5);
plot(w/pi,10*log10(abs(h3)));
title('band stop filter using hamming window');
```

```
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,6);
stem(wh);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,7);
plot(w/pi,10*log10(abs(h4)));
title('band stop filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,8);
stem(whn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

Result:

Implemented various FIR filters using different windows

1Low Pass Filter

2.High Pass Filter

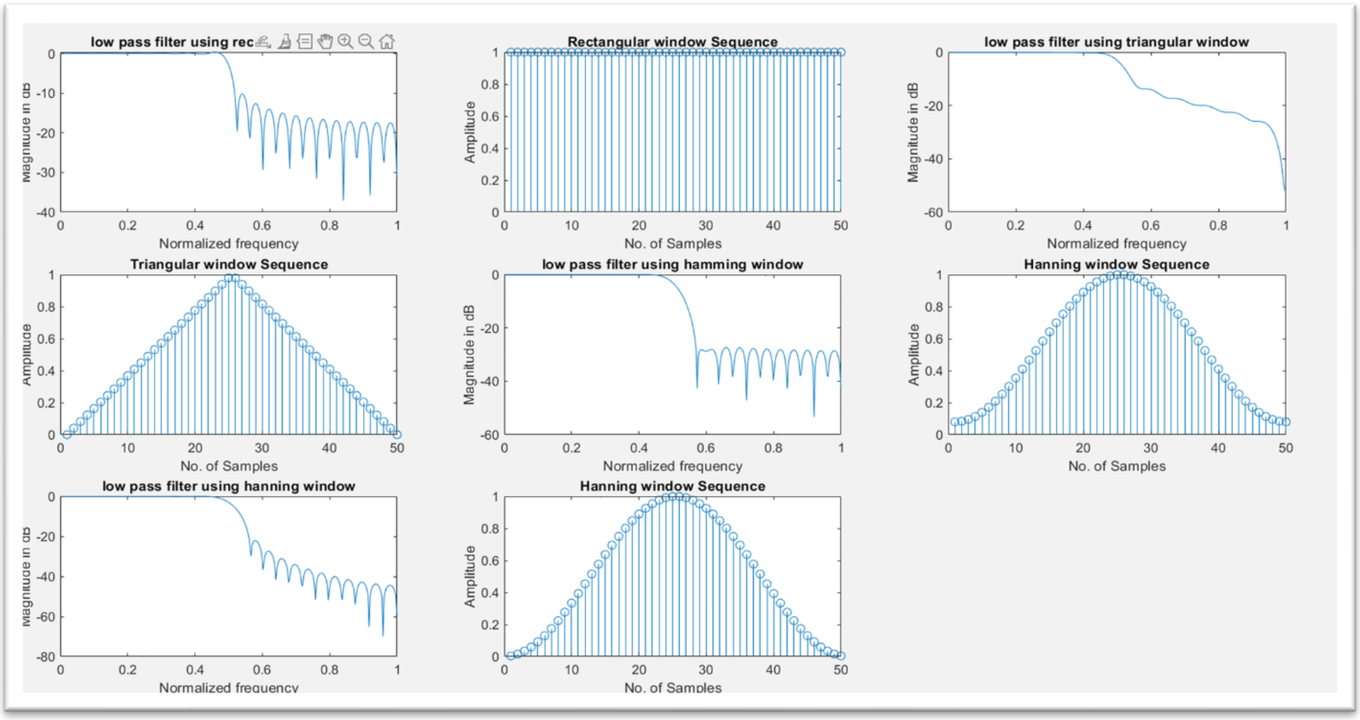
3.Band pass Filter

4.Band stop Filter

Observation:

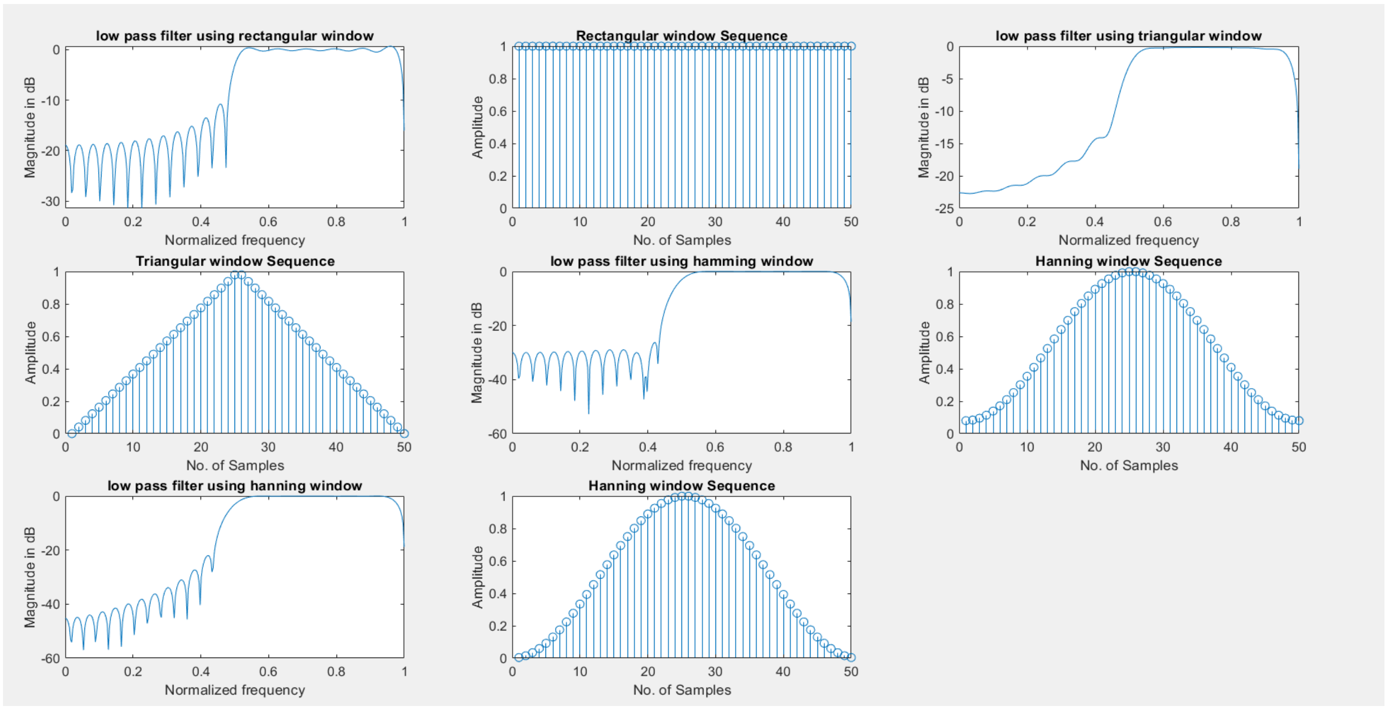
1.Low Pass Filter

Enter the value of N=50



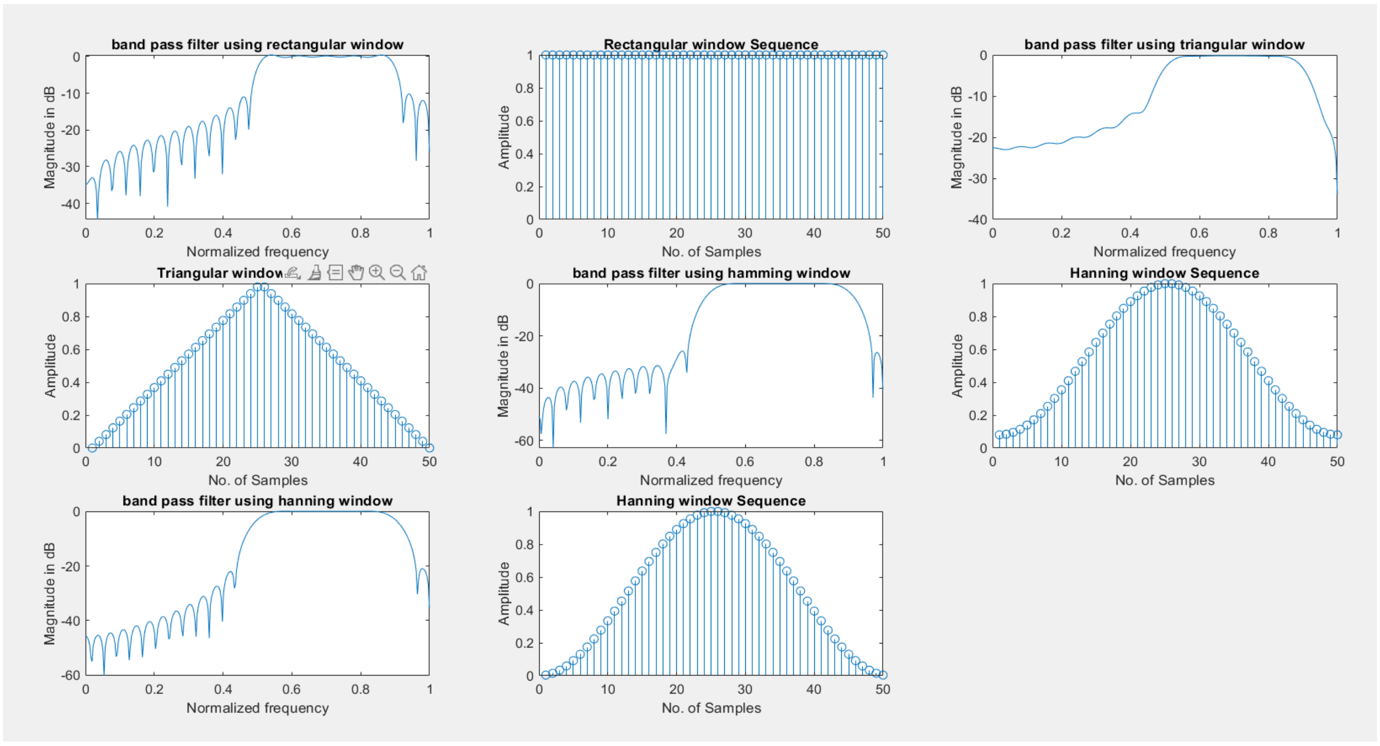
2.High Pass Filter

Enter the value of N=50



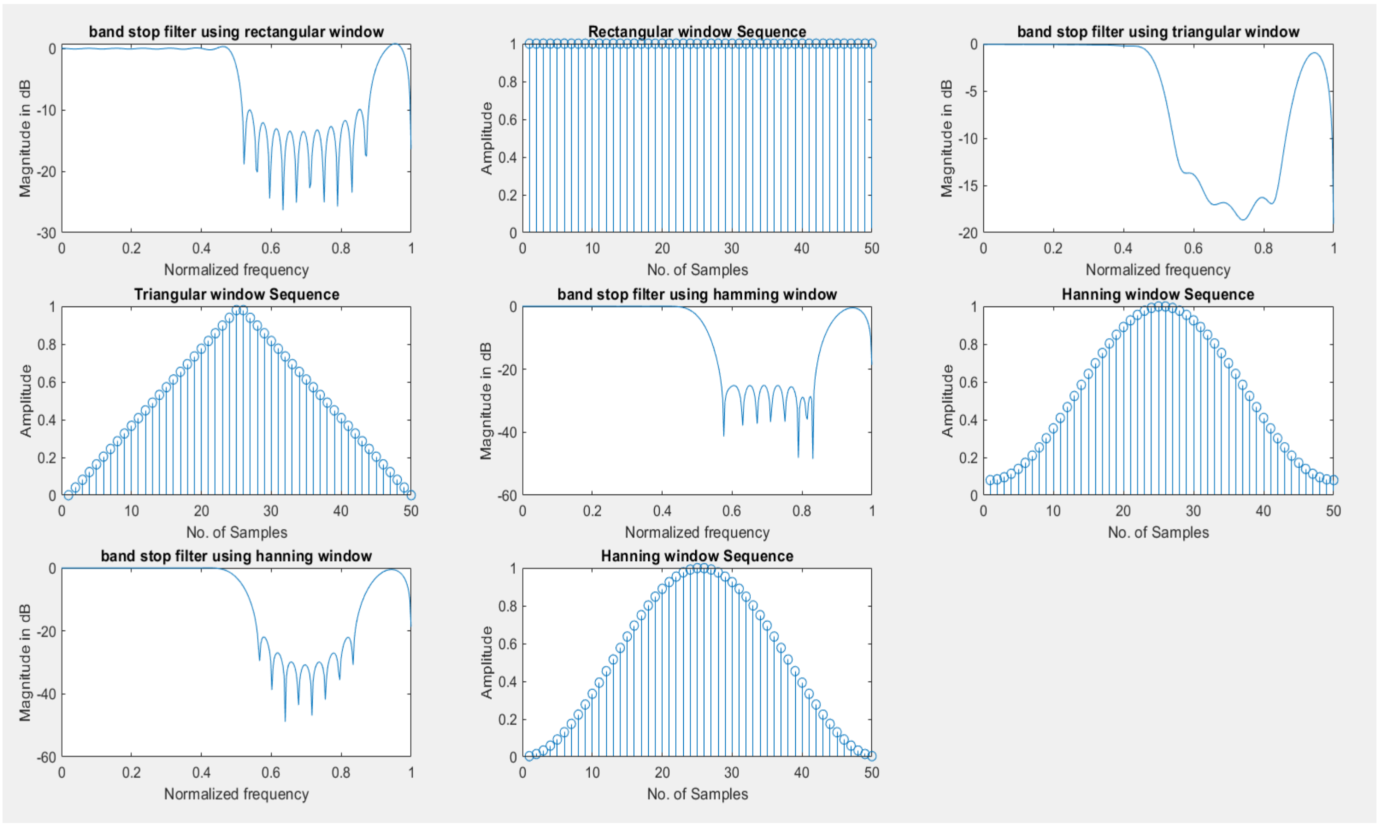
3.Band Pass Filter

Enter the value of N=50



4.Band Stop Filter

Enter the value of N=50



Experiment No: 10 Date: 22/10/24

FAMILIARIZATION OF THE ANALOG AND DIGITAL INPUT AND OUTPUT PORTS OF DSP BOARD

Aim:

Familiarization of the analog and digital input and output ports of DSP Boards.

Theory:

TMS 320C674x DSP CPU

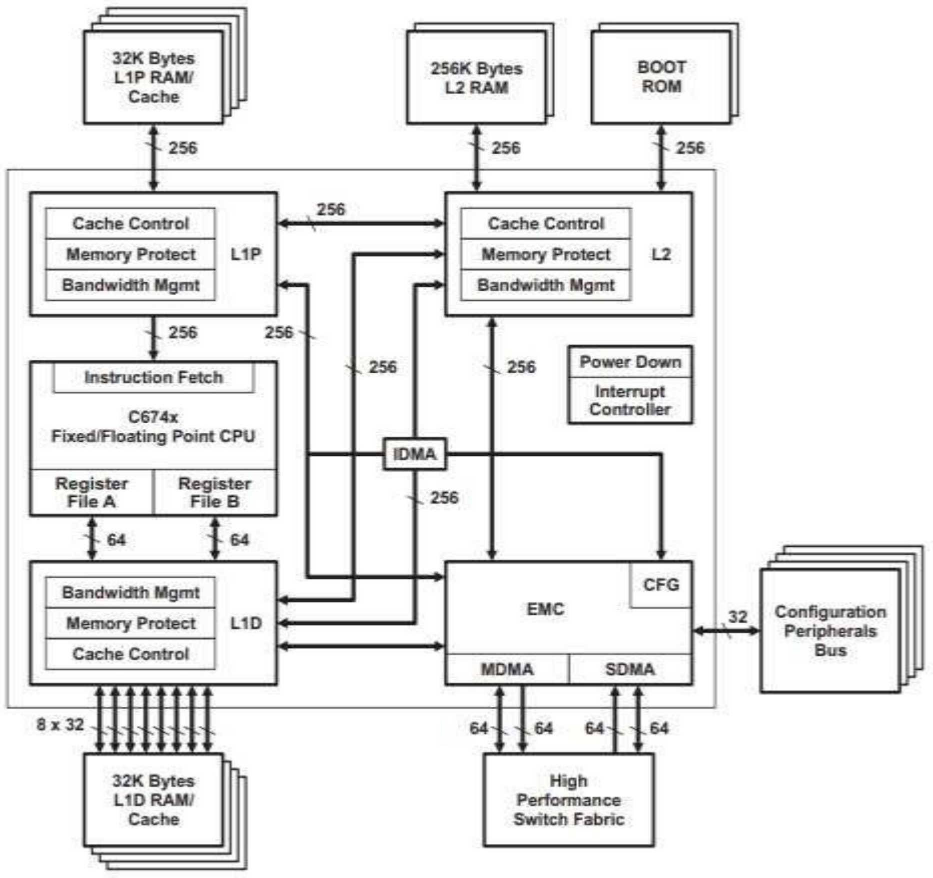


FIGURE: TMS320C 674X DSP CPU BLOCK DIAGRAM

The TMS320C674X DSP CPU consists of eight functional units, two register files, and two data paths as shown in Figure. The two general-purpose register files (A and B) each contain 32 32- bit registers for a total of 64 registers. The general-purpose registers can be usedfor data or can be data address pointers.

The data types supported include packed 8-bit data, packed 16-bit data, 32-bit data, 40- bit data, and 64-bit data. Values larger than 32 bits, such as 40-bit-long or 64-bit-long values are stored in register pairs, with the 32 LSBs of data placed in an even register and the remaining 8 or 32 MSBs in the next upper register (which is alwaysan odd-numbered register). The eight functional units (.M1, .L1, .D1, .S1, .M2, .L2, .D2, and.S2) are each capable of executing one instruction every clock cycle. The .M functional units perform all multiply operations. The .S and .L units perform a general set of arithmetic, logical,and branch functions. The .D units primarily load data from memory to the register file and store results from the register file into memory.

Multichannel Audio Serial Port (McASP):

The McASP serial port is specifically designed for multichannel audio applications.Its key features are:

• Flexible clock and frame sync generation logic and on-chip dividers

• Up to sixteen transmit or receive data pins and serializers

• Large number of serial data format options, including: – TDM Frames with 2 to 32 time slotsper frame (periodic) or 1 slot per frame (burst) – Time slots of 8,12,16, 20, 24, 28, and 32 bits
– First bit delay 0, 1, or 2 clocks – MSB or LSB first bit order – Left- or right-aligned datawords within time slots

• DIT Mode with 384-bit Channel Status and 384-bit User Data registers

• Extensive error checking and mute generation logic

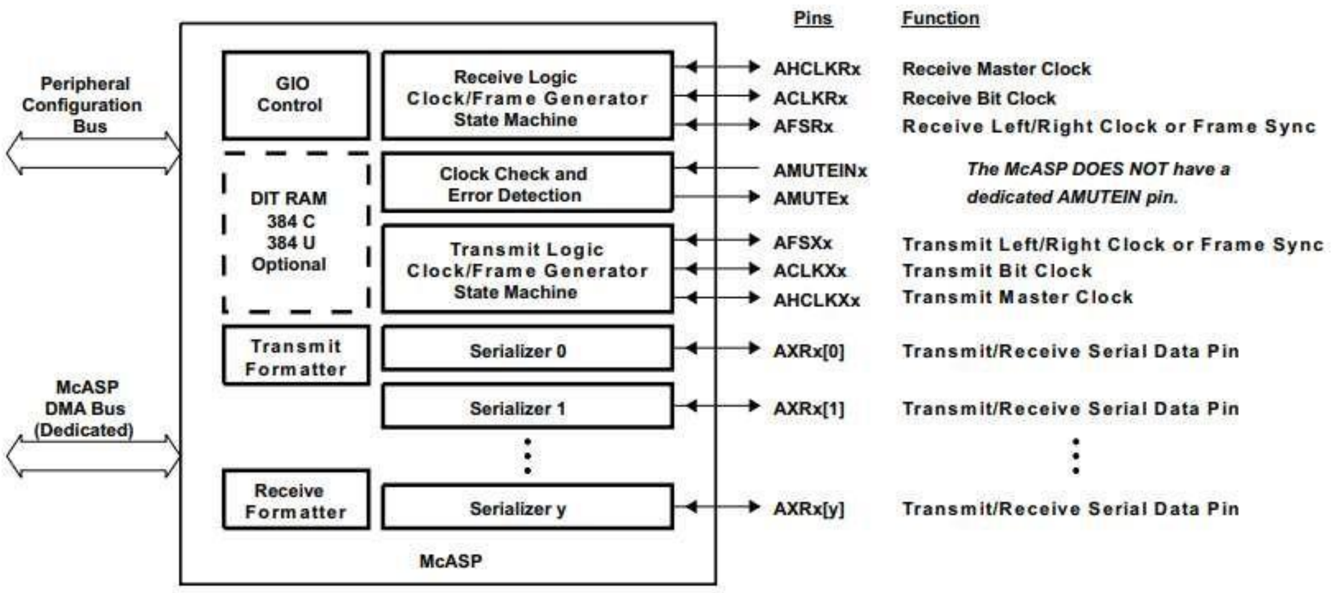
• All unused pins GPIO-capable

• Transmit & Receive FIFO Buffers allow the McASP to operate at a higher sample rate by making it more tolerant to DMA latency.

• Dynamic Adjustment of Clock Dividers – Clock Divider Value may be changed without resetting the McASP. The DSK board includes the TLV320AIC23 (AIC23) codec for input and output.

The ADC circuitry on the codec converts the input analog signal to a digital representation to be processed by the DSP. The maximum level of the input signal to be converted is determinedby the specific ADC circuitry on the codec, which is 6 V p-p with the onboard codec. After thecaptured signal is processed, the result needs.

to be sent to the outside world. DAC, which performs the reverse operation of the ADC. An output filter smooths out or reconstructs the output signal. ADC, DAC, and all required filteringfunctions are performed by the single-chip codec AIC23 on board the DSK.


Pins
Function
Peripheral Configuration Bus
GIO Control
Receive Logic Clock/Frame Generator State Machine
AHCLKRx
Receive Master Clock
ACLKRx
Receive Bit Clock
AFSRx
Receive Left/Right Clock or Frame Sync
DIT RAM 384 C 384 U Optional
Clock Check and Error Detection
AMUTEINx
AMUTEx
The McASP DOES NOT have a dedicated AMUTEIN pin.
Transmit Logic Clock/Frame Generator State Machine
AFSXx
Transmit Left/Right Clock or Frame Sync
ACLKXx
Transmit Bit Clock
AHCLKXx
Transmit Master Clock
Transmit Formatter
Serializer 0
AXRx[0]
Transmit/Receive Serial Data Pin
McASP DMA Bus (Dedicated)
Serializer 1
AXRx[1]
Transmit/Receive Serial Data Pin
Receive Formatter
Serializer y
AXRx[y]
Transmit/Receive Serial Data Pin
McASP

Result:

Familiarized the input and output ports of dsp board

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 periodicsignal 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:

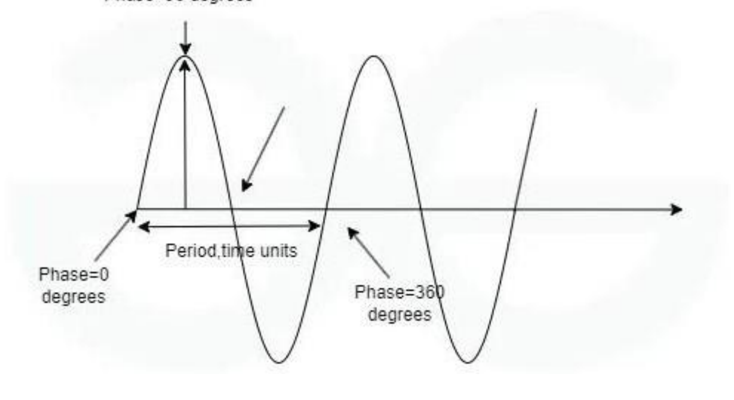
- **Graph:** It is a plot used to depict the relation between quantities. Depending upon the number of variables, we can decide to number of axes each perpendicular to the other.
- **Time period: The** period for a signal can be defined as the time taken by a periodic signal to complete one cycle.
- **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 phaseof 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

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



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 thealready 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.

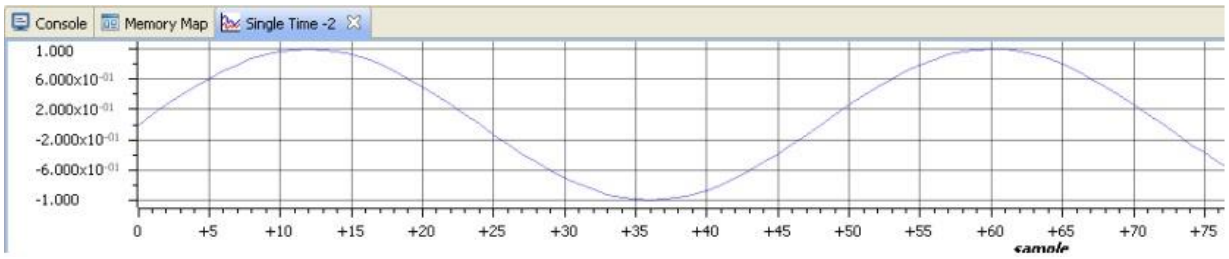
Program:

```
#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

Observation:



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, signalfiltering, 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) andg(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. DSPsystems 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.

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.

Observation:

Xn

0x80010000 - 1

0x80010004 - 2

0x80010008 - 3

Hn

0x80011000 - 1

0x80011004 - 2

XnLength

0x80012000 - 3

HnLength

0x80012004 - 2

Output

0x80013000 - 1

0x80013004 - 4

0x80013008 - 7

0x8001300C - 6