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/Smt. NEETA SARA REJI of ***5th*** *Semester class (Roll No.* ***[B22ECB74] 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 : ***[TKM22EC100]***

Staff Member in-charge *External Examiner*

Date:

INDEX

Experiment No:1 Date:29-7-24

SIMULATION OF BASIC TEST SIGNALS

Aim

Simulation of basic test signals using Matlab,

Signals are:

1) Impulse signal
2) Unit step signal
3) Ramp signal
4) Sine signal
5) Cosine signal
6) Square Wave-Bipolar
7) Square Wave-Unipolar
8) Triangular signal
9) Exponential signal

Theory

1. Impulse signal

An **impulse signal** is an idealized, infinitesimally narrow pulse that occurs at a single instant in time, typically at t=0. It is represented mathematically by the Dirac delta function, denotedas

$$\delta(t) = \begin{cases} 0, & \text{if } t \neq 0 \\ \infty, & \text{if } t = 0 \end{cases}$$

2. Unit step signal

is a function that jumps from 0 to 1 at a specified time, typically at t=0. The unit step function is denoted as

$$u(t) = \begin{cases} 0, & if\ t < 0 \\ 1, & if\ t \geq 0 \end{cases}$$

3. Ramp signal

A **ramp signal** is a signal that increases linearly with time, starting from zero. The ramp function is denoted as

$$r(t) = \begin{cases} t, & if\ t \geq 0 \\ 0, & if\ t < 0 \end{cases}$$

4. Sine signal

A **sine signal** is a continuous wave that oscillates smoothly and periodically over time, following the shape of a sine or cosine function. The general form of a sinusoidal signal is given by:

$$x(t) = Asin(\omega t + \phi)$$

Where,

- Ais the amplitude of the signal (the peak value),
- ω is the angular frequency in radians per second, where $\omega = 2\pi f$, f is the frequency in Hertz,
- t is the time variable
- ϕ is the phase shift, which determines the initial angle at t=0.

5. Cosine signal

A **cosine signal** is a type of sinusoidal signal that oscillates in a smooth, periodic manner over time, following the shape of a cosine function. The general form of a cosine signal is given by:

$$x(t) = Acos(\omega t + \phi)$$

Where:

- A is the amplitude of the signal (the peak value),
- ω is the angular frequency in radians per second, where $\omega = 2\pi f$ and f is the frequency in Hertz.
- t is the time variable
- ϕ is the phase shift, which determines the initial angle at $t = 0$.

6. Square Wave-Bipolar

A square wave is a type of periodic waveform that alternates between two distinct levels, typically +A and -A in a bipolar signal. It has a 50% duty cycle, meaning the signal spends equal time at both levels. The equation for a bipolar square wave can be written as:

$V(t)=A\cdot sgn(sin(2\pi ft))$

where A is the amplitude, f is the frequency, and sgn is the sign function.

7. Square Wave-Unipolar

A unipolar square wave is a periodic signal that alternates between 0 and a positive voltage level (e.g., V_max) with abrupt transitions. It has no negative amplitude. The signal is typically represented as:

$f(t)=Vmax$, for $0 \leq t < T/2$ $f(t)=0$, for $T/2 \leq t < T$

Where T is the period of the waveform.

8. Triangular signal

A **triangular signal** is a type of periodic waveform that linearly rises and falls between a maximum and minimum value, forming a triangular shape. The transition between the high and low levels in a triangular wave is gradual, creating a linear slope.

9. Exponential signal

An **exponential signal** is a signal whose amplitude varies exponentially with time. It can either grow or decay depending on the sign of the exponent. The exponential signal is generally expressed as

$$x(t) = Ae^{\alpha t}$$

Where:

- A is the amplitude of the signal,
- α is the exponent that determines the rate of growth or decay,
- t is the time variable.
- If $\alpha > 0$, the signal represents exponential growth.
- If $\alpha < 0$, the signal represents exponential decay.

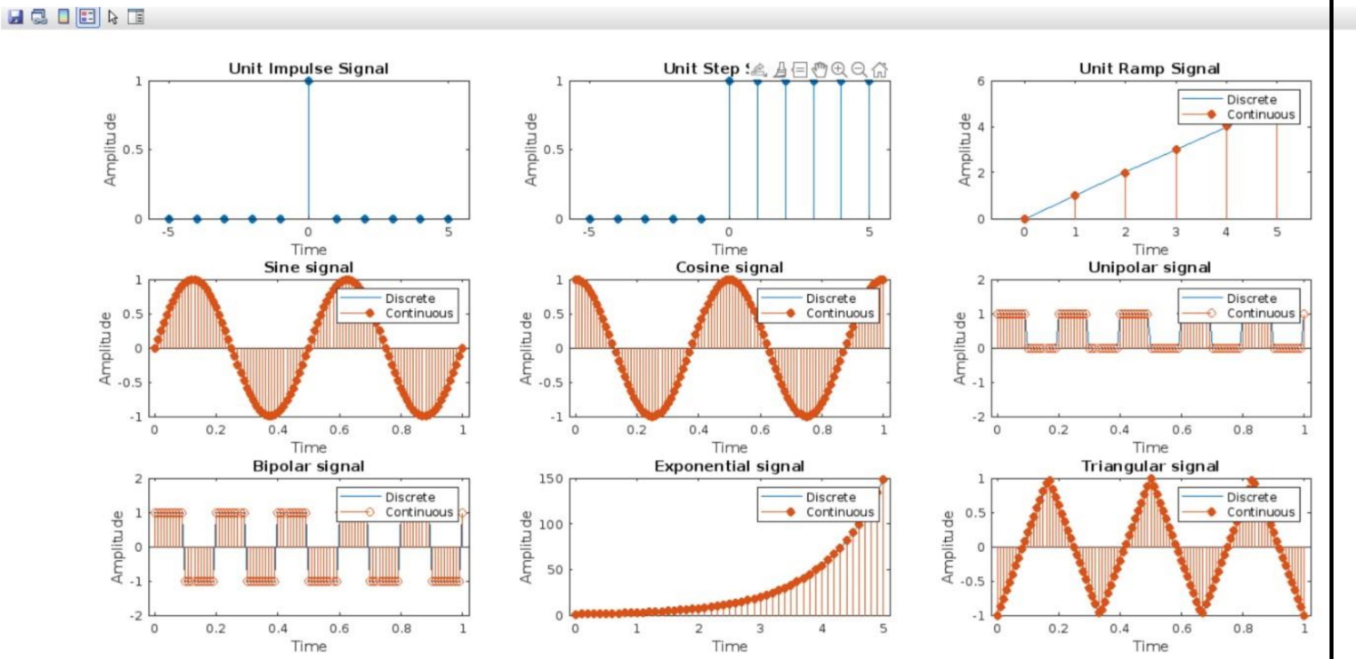
PROGRAM

```
%unit impulse signal
clc;
close all;
t1=-5:1:5;
y1=[zeros(1,5),ones(1,1),zeros(1,5)];
subplot(3,3,1);
stem(t1,y1,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Unit Impulse Signal");
%unit step signal
t2=-5:1:5;
y2=[zeros(1,5),ones(1,6)];
subplot(3,3,2);
stem(t2,y2,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Unit Step Signal");
%unit ramp signal
t3=0:1:5;
y3=t3;
subplot(3,3,3);
plot(t3,y3);
hold on;
```

```
stem(t3,y3,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Unit Ramp Signal");
legend("Discrete","Continuous");
%sine
t4=0:0.01:1;
f4=2;
y4=sin(2*pi*f4*t4);
subplot(3,3,4);
plot(t4,y4);
hold on;
stem(t4,y4,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Sine signal");
legend("Discrete","Continuous");
%cosine signal
t5=0:0.01:1;
f5=2;
y5=cos(2*pi*f5*t5);
subplot(3,3,5);
plot(t5,y5);
hold on;
stem(t5,y5,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Cosine signal");
legend("Discrete","Continuous");
%unipolar signal
```

```
f6=5;
t6=0:0.01:1;
y6=sqrt(square(2*pi*f6*t6));
subplot(3,3,6);
plot(t6,y6);
hold on;
stem(t6,y6);
legend("Discrete","Continuous");
xlabel("Time");
ylabel("Amplitude");
title("Unipolar signal");
ylim([-2,2]);
%bipolar signal
f7=5;
t7=0:0.01:1;
y7=square(2*pi*f7*t7);
subplot(3,3,7);
plot(t7,y7);
hold on;
stem(t7,y7);
legend("Discrete","Continuous");
xlabel("Time");
ylabel("Amplitude");
title("Bipolar signal");
ylim([-2,2]);
%exponential signal
t8=0:0.1:5;
y8=exp(t8);
subplot(3,3,8);
plot(t8,y8);
```

OBSERVATION



```
hold on;
stem(t8,y8,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Exponential signal");
legend("Discrete","Continuous");
%triangular signal
t9=0:0.01:1;
f9=3;
y9=sawtooth(2*pi*f9*t9,0.5);
subplot(3,3,9);
plot(t9,y9);
hold on;
stem(t9,y9,'filled');
xlabel("Time");
ylabel("Amplitude");
title("Triangular signal");
legend("Discrete","Continuous");
```

Result

Simulated and plotted basic test signal in Matlab.

1) Impulse signal
2) Unit step signal
3) Ramp signal
4) Sine signal
5) Cosine signal
6) Square Wave-Bipolar
7) Square Wave-Unipolar
8) Triangular signal
9) Exponential signal

Experiment No:2 Date:6-08-24

VERIFICATION OF SAMPLING THEOREM

Aim

To verify sampling theorem using Matlab.

Theory

The Sampling Theorem, or Nyquist-Shannon theorem, states that a continuous signal can be accurately reconstructed from its samples if it is sampled at a rate at least twice the highest frequency present in the signal. This minimum sampling rate is called the Nyquist rate and is given by:

$$f_s \geq 2f_{max}$$

where fs is the sampling frequency and fmax is the maximum frequency in the signal.

Undersampling occurs when $f_s < 2f_{max}$, leading to aliasing, where high-frequency components appear as lower frequencies.

Nyquist sampling is when $f_s = 2f_{max}$, ensuring perfect reconstruction.

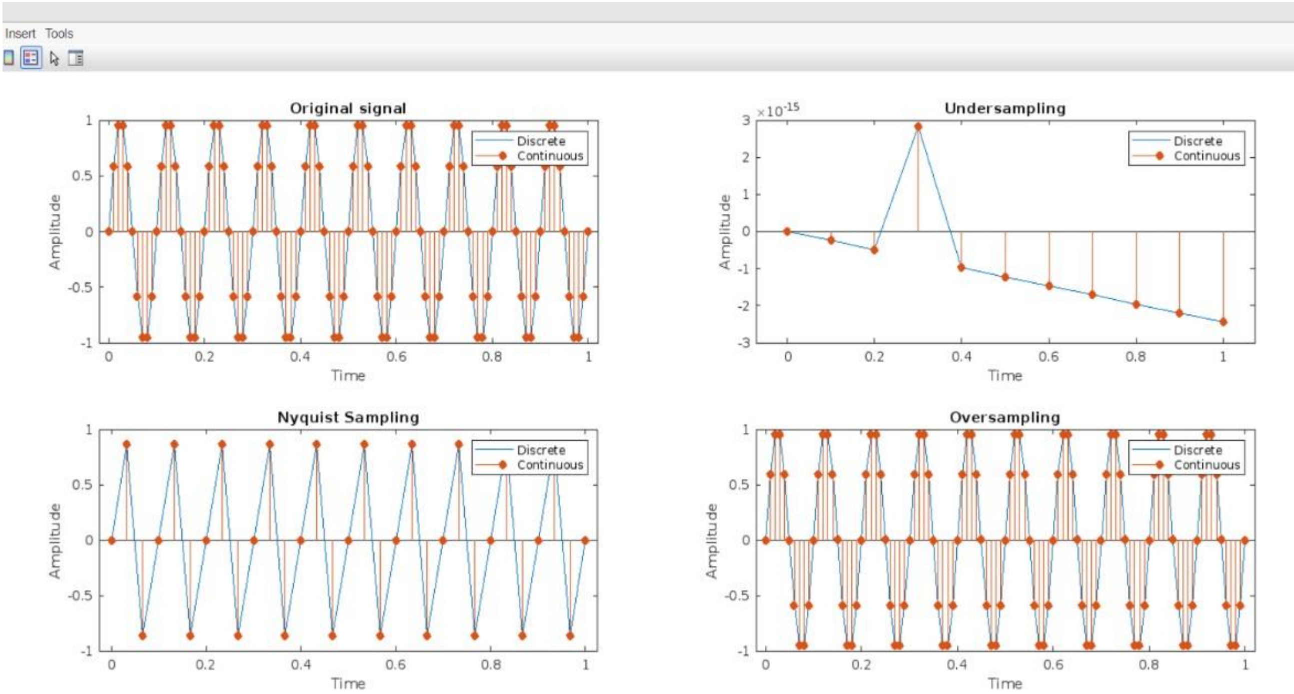
Oversampling is when $f_s > 2f_{max}$, which increases redundancy without aliasing, improving signal quality and noise reduction.

PROGRAM

```
clc;
clear  all;
close  all;
t=0:0.01:1;
fm=10;
y=sin(2*pi*fm*t);
subplot(2,2,1);
plot(t,y);
hold on;
stem(t,y,"filled");
xlabel("Time");
ylabel("Amplitude");
title("Original signal");
legend("Discrete", "Continuous");

%undersampling
fs1=fm;
t1=0:1/fs1:1;
y1=sin(2*pi*fm*t1);
subplot(2,2,2);
plot(t1,y1);
hold on;
stem(t1,y1,"filled");
xlabel("Time");
ylabel("Amplitude");
title("Undersampling");
```

OBSERVATION



```
legend("Discrete","Continuous");
%nyquist sampling
fs2=3*fm;
t2=0:1/fs2:1;
y2=sin(2*pi*fm*t2);
subplot(2,2,3);
plot(t2,y2);
hold on;
stem(t2,y2, "filled");
xlabel("Time");
ylabel("Amplitude");
title("Nyquist Sampling");
legend("Discrete", "Continuous");
%oversampling
fs3=10*fm;
t3=0:1/fs3:1;
y3=sin(2*pi*fm*t3);
subplot(2,2,4);
plot(t3,y3);
hold on;
stem (t3,y3,"filled");
xlabel("Time");
ylabel("Amplitude");
title("Oversampling");
legend("Discrete", "Continuous");
```

RESULT

Verified sampling theorem using Matlab.

Experiment no-3 Date:10-8-24

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 in signal processing to combine two signals, often to understand how one signal modifies another. It operates by sliding one function over another, multiplying corresponding values, and summing the products at each step. For two signals x(n)and h(n), their convolution y(n) is given by:

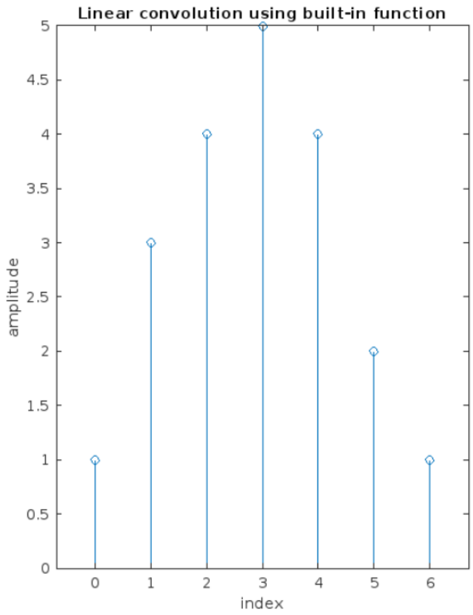
$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

This process evaluates how much the signal h(n), often called the impulse response, overlaps with the input signal x(n). Linear convolution is used for filtering, smoothing, and analyzing system responses in digital and analog signal processing. The result of the convolution is usually a signal whose length is the sum of the lengths of the two input signals minus one.

OBSERVATION

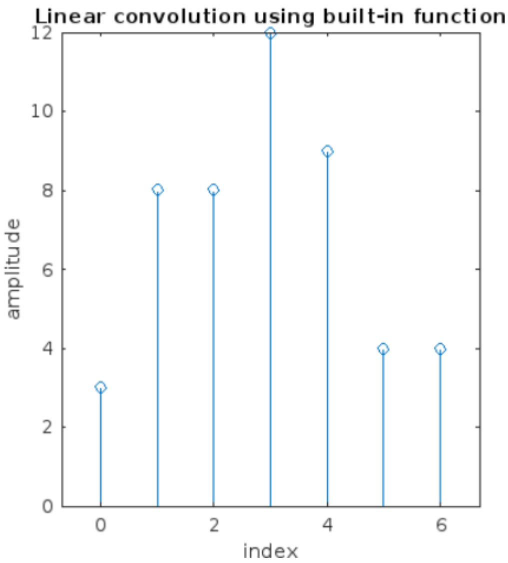
1) x(n) = [1 2 1 1]

h(n) = [1 1 1 1]



2) x(n) = [1 2 1 2]

h(n) = [3 2 1 2]



PROGRAM

a)LINEAR CONVOLUTION USING BUILT IN FUNCTION

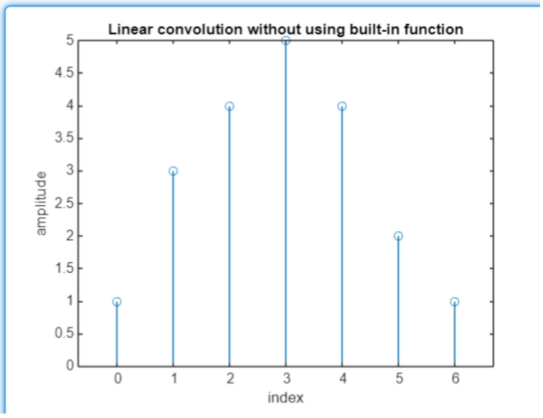
```
clc;
close all;
x=input("enter input");
x_index=input("enter index of x");
h=input("enter impulse response");
h_index=input("enter index of h");
y_index=min(x_index)+min(h_index):max(x_index)+max(h_index);
y=conv(x,h);
disp(y);
subplot(1,2,1);
stem(y_index,y);
xlabel("index");
ylabel("amplitude");
title("Linear convolution using built-in function");
```

OBSERVATION

1) x(n) = [1 2 1 1]

h(n) = [1 1 1 1]

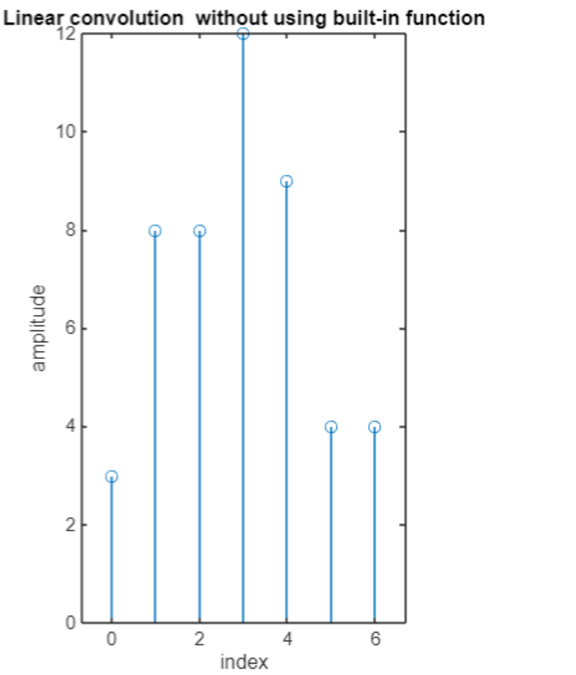
```
Result:
     1     3     4     5     4     2     1
```



2) x(n) = [1 2 1 2]

h(n) = [3 2 1 2]

```
3     8     8    12     9     4     4
```



b)LINEAR CONVOLUTION WITHOUT USING BUILT IN FUNCTION

```
% without using built in
clc;
close all;
x=input("enter input");
x_index=input("enter index of x");
h=input("enter impulse response");
h_index=input("enter index of h");
y_index=min(x_index)+min(h_index):max(x_index)+max(h_index);
n=length(x);
m=length(h);
len_y=length(y_index);
y=zeros(1,len_y);
for i=1:n
    for j=1:m
        y(i+j-1)=y(i+j-1)+x(i)*h(i);
    end

end
disp("Result:")
disp(y)
stem(y_index,y);
xlabel("index");
ylabel("amplitude");
title("Linear convolution without using built-in function");
```

RESULT

Performed linear convolution with and without using built in function in Matlab.

Experiment No:4 Date:3-9-24

CIRCULAR CONVOLUTION

AIM

To find circular convolution using FFT, concentric circle method and matrix method using Matlab.

THEORY

Circular convolution is a mathematical operation used primarily in signal processing. It involves wrapping one signal around a circular buffer and performing the convolution operation on it, often used when signals are periodic or when working with discrete Fourier transforms (DFT). This technique ensures that the result maintains periodicity by aligning the endpoints of signals. It is computationally efficient and widely applied in fast algorithms like the Fast Fourier Transform (FFT). It can be performed by 3 methods:

Using FFT (Fast Fourier Transform):

- Circular convolution is performed by transforming the sequences to the frequency domain using FFT, multiplying them element-wise, and transforming them back using the inverse FFT.

$$y[n] = \mathrm{IFFT}(\mathrm{FFT}(x[n]) \cdot \mathrm{FFT}(h[n]))$$

Concentric Circle Method:

- This is a graphical method where one sequence is placed in a circular pattern, and the other sequence is rotated around it. The inner product of corresponding values after each rotation gives the result.

Matrix Method:

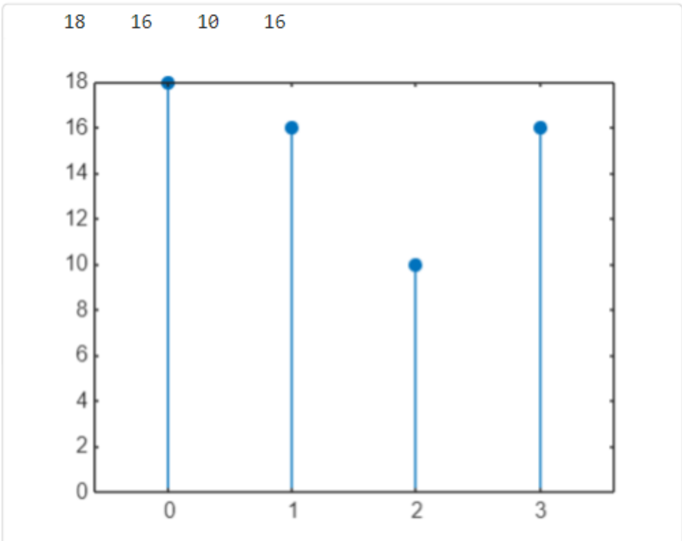
- Circular convolution can be represented as matrix multiplication, where one sequence is arranged in a circulant matrix, and the other is a column vector.

  $y=C \cdot h$

  where C is the circulant matrix formed from x, and h is the vecto

OBSERVATION

CIRCULAR CONVOLUTION USING FFT



PROGRAM

a)Circular convolution using FFT

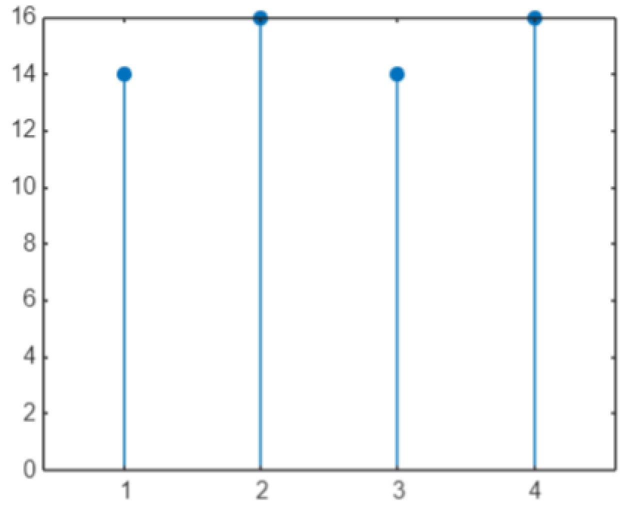
```
clc;
clear;
close all;
x=input("Enter the seq1:");
h=input("Enter the seq2:");
x_len=length(x);
h_len=length(h);
n=max(x_len,h_len);
xnew=[x zeros(1,n-x_len)];
hnew=[h zeros(1,n-h_len)];
x1=fft(xnew);
h1=fft(hnew);
y1=x1.*h1;
y=ifft(y1);
y_ind=0:n-1;
disp(y);
stem(y_ind,y, "filled");
```

OBSERVATION

b) CIRCULAR CONVOLUTION USING CONCENTRIC CIRCLE METHOD

```
Reversed x
     1     2     1     2
```

```
Convolution product
    14    16    14    16
```



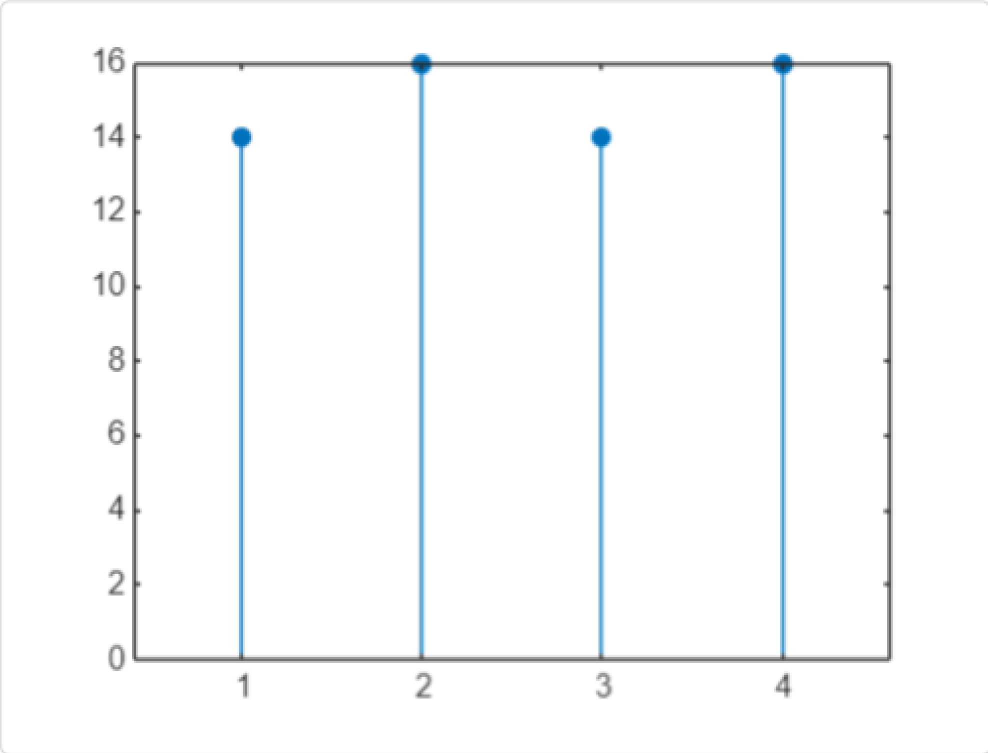
b) Circular convolution using Concentric Circle Method

```
clc;
clear;
close;
x=[2 1 2 1];
X=x(:, end-1:1);
h=[1 2 3 4];
for i=1:length(x)
x=[x(end) x(1:end-1)];
h1=h;
y(i)=sum(x.*h1);
end
disp(y);
```

OBSERVATION

c) CIRCULAR CONVOLUTION USING MATRIX METHOD

14
16
14
16



c)Circular convolution using Matrix Method

```
%Circular convolution using matrix multiplication
clc;
close all;
clear all;
xn=[2 1 2 1]; %defining the matrix
hn=[1 2 3 4];
h=[ ];
hn=hn(:,end:-1:1);%accesing the elements of hn
for i=1:length(hn)
    hn=[hn(end) hn(1:end-1)];
    h=[h;hn];
end
y=h*xn';
disp(y);
stem(y,'filled');
```

RESULT

Obtained circular convolution using FFT, concentric circle method and matrix method in Matlab

Experiment No:5 Date:10-9-24

LINEAR CONVOLUTION USING CIRCULAR CONVOLUTION AND VICE VERSA

AIM

To perform linear convolution using circular convolution and vice versa using Matlab.

THEORY

Performing Linear Convolution Using Circular Convolution

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.

Performing Circular Convolution Using Linear Convolution

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. E

OBSERVATION

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

PROGRAM

a) Linear convolution using circular convolution

```
%linear convolution using circular convolution
clc;
clear;
close all;
x=[1,2,3,4];
y=[1,1,1];
xl=length(x);
yl=length(y);
zl=(xl+yl)-1;
xn=[x zeros(1,zl-xl)];
yn=[y zeros(1,zl-yl)];
xa=fft(xn);
ya=fft(yn);
ans=xa.*ya;
anss=ifft(ans);
disp(anss);
answ=conv(x,y);
disp(answ);
```

OBSERVATION

8 7 6 9

b) Circular convolution using linear convolution

```
%circular convolution using linear convolution
clc;
clear;
close all;
x = [1, 2, 3, 4];
h = [1, 1, 1];
y=conv(x,h);
z=max(length(x), length(h));
r = [y(1:z)];
new = [y(z+1:length(y)) zeros(1, length(y)-z)];
for k = 1:z-1
    r(k)=r(k)+new(k);
end
disp(r);
```

RESULT

Performed linear convolution using circular convolution and vice versa using Matlab.

Experiment No:6 Date: 24/09/24

DFT and IDFT

Aim

To compute DFT and IDFT of a signal using inbuilt functions and manual methods.

Theory:

The Discrete Fourier Transform (DFT) is a fundamental mathematical tool used in signal processing, communication systems, and many areas of engineering and science. It converts a discrete sequence (signal) from the time domain into its representation in the frequency domain. The DFT transforms a finite sequence of equally spaced samples of a function into a sequence of coefficients of complex sinusoids, ordered by their frequencies.

The Inverse Discrete Fourier Transform (IDFT) is a mathematical process that converts a sequence of complex numbers in the frequency domain back into the time domain. It is the inverse operation of the Discrete Fourier Transform (DFT) and is used to recover the original time-domain sequence from its DFT.

Program:

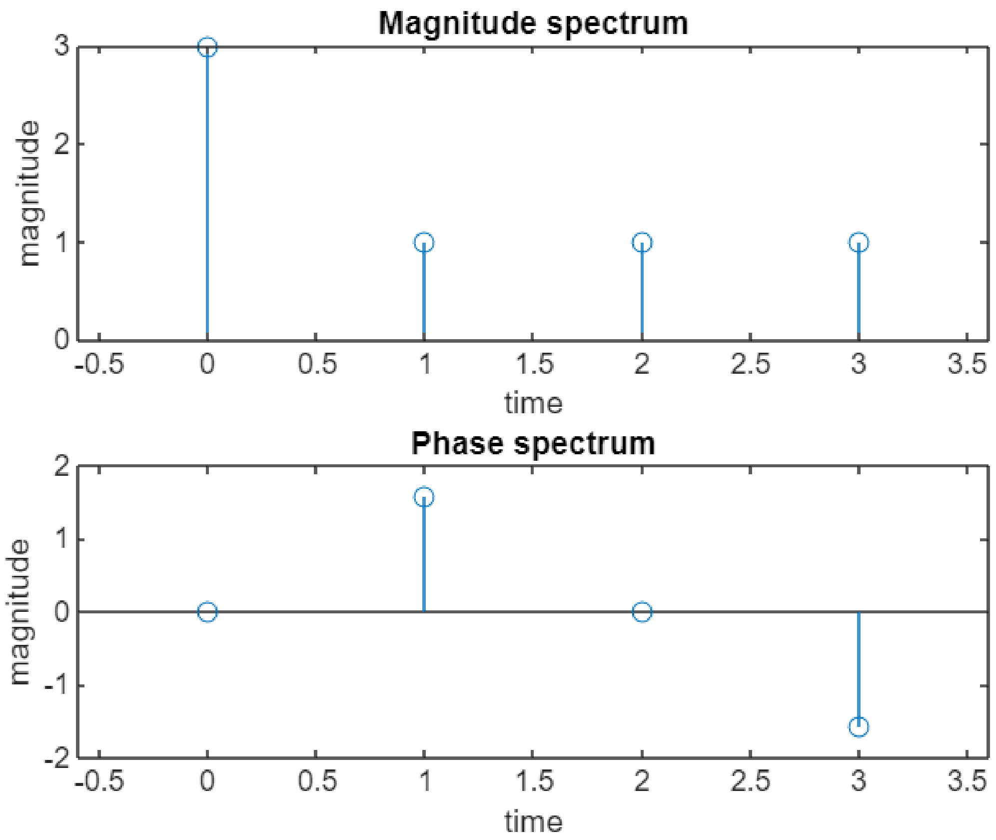
```
%dft using inbuilt and manual methods
clc;
clear all;
close all;
x=input('Enter the sequence:');
N=input('enter the N point DFT: ');
l=length(x);
x=[x zeros(1,N-1)];
X=zeros(N,1);
for k=0:N-1
    for n=0:N-1
        X(k+1)=X(k+1)+x(n+1)*exp(-1j*2*pi*n*(k/N));
    end
end
disp('X');
disp(X);
```

Observation:

Enter the sequence:[1 0 1 1]

enter the N point DFT: 4

X
   3.0000 + 0.0000i
  -0.0000 + 1.0000i
   1.0000 - 0.0000i
   0.0000 - 1.0000i
DFT
3.0000 + 0.0000i 0.0000 + 1.0000i   1.0000 + 0.0000i 0.0000 - 1.0000i



```
disp("DFT");
disp(fft(x,N));

%magnitude spectrum

k=0:N-1;

mag=abs(X);

subplot(2,1,1);

stem(k,mag);

xlabel('time');

ylabel('magnitude');

title('Magnitude spectrum');

%phase spectrum

phase=angle(X);

subplot(2,1,2);

stem(k,phase);

xlabel('time');

ylabel('magnitude');

title('Phase spectrum');
```

Enter the sequence:[1 0 1 1]

Enter the N point DFT: 8

X

3.0000 + 0.0000i

0.2929 - 1.7071i

-0.0000 + 1.0000i

1.7071 + 0.2929i

1.0000 - 0.0000i

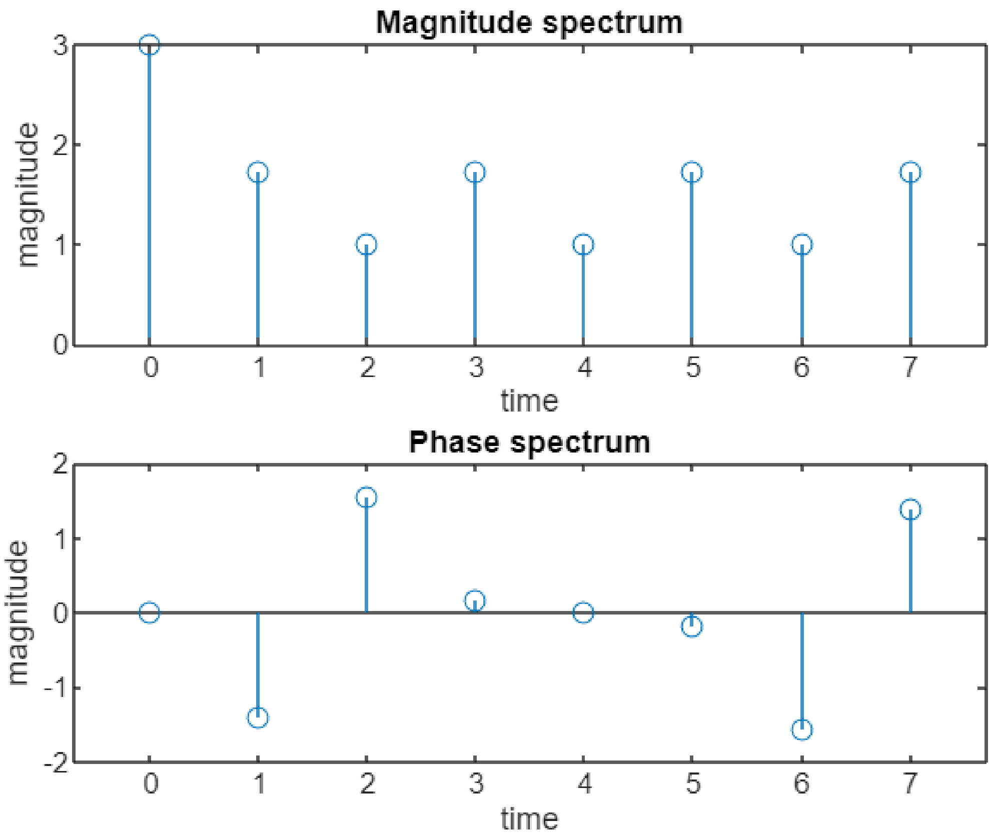
1.7071 - 0.2929i

0.0000 - 1.0000i

0.2929 + 1.7071i

DFT

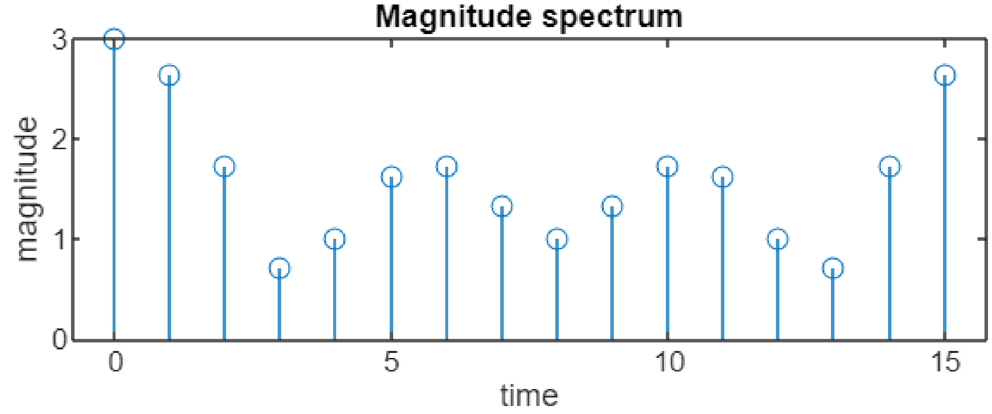
3.0000 + 0.0000i 0.2929 - 1.7071i 0.0000 + 1.0000i 1.7071 + 0.2929i 1.0000 + 0.0000i 1.7071 - 0.2929i 0.0000 - 1.0000i 0.2929 + 1.7071i

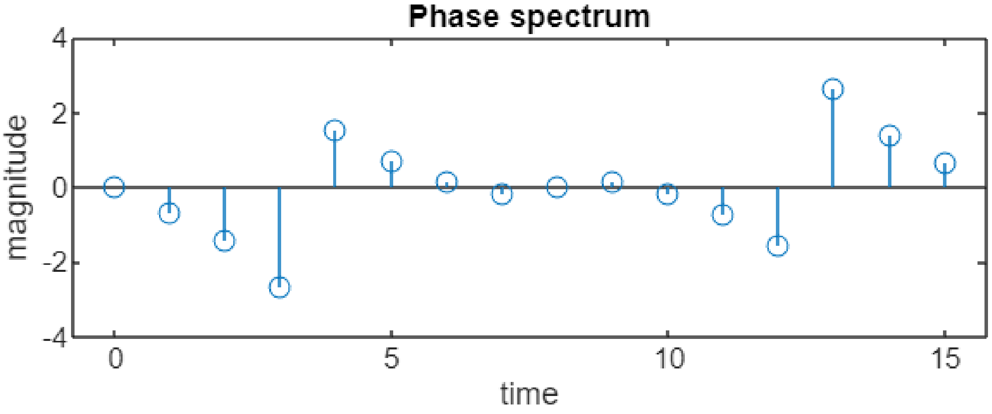


```
Enter the sequence:[1 0 1 1]

enter the N point DFT: 16

X
   3.0000 + 0.0000i
   2.0898 - 1.6310i
   0.2929 - 1.7071i
  -0.6310 - 0.3244i
  -0.0000 + 1.0000i
   1.2168 + 1.0898i
   1.7071 + 0.2929i
   1.3244 - 0.2168i
   1.0000 - 0.0000i
   1.3244 + 0.2168i
   1.7071 - 0.2929i
   1.2168 - 1.0898i
   0.0000 - 1.0000i
  -0.6310 + 0.3244i
   0.2929 + 1.7071i
   2.0898 + 1.6310i
DFT
     3.0000 + 0.0000i   2.0898 - 1.6310i   0.2929 - 1.7071i  -0.6310 - 0.3244i   0.0000 + 1.0000i
   1.2168 + 1.0898i   1.7071 + 0.2929i   1.3244 - 0.2168i   1.0000 + 0.0000i   1.3244 + 0.2168i
1.7071 - 0.2929i   1.2168 - 1.0898i   0.0000 - 1.0000i  -0.6310 + 0.3244i   0.2929 + 1.7071i
2.0898 + 1.6310i
```





Observation:

Enter the sequence:[1 1 1 0]
Enter the N point of DFT:4
x

```
   0.7500 + 0.0000i
   0.0000 + 0.2500i
   0.2500 - 0.0000i
  -0.0000 - 0.2500i
```

IDFT

```
   0.7500 + 0.0000i  0.0000 + 0.2500i   0.2500 + 0.0000i  0.0000 - 0.2500i
```

```
%idft using inbuilt and manual functions
clc;
clear all;
close all;
X=input('Enter the sequence:');
N=input('Enter the N point of DFT:');
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('IDFT');
disp(ifft(X,N));
```

Observation:

Enter the input sequence: [1 0 1 1]

DFT of the input sequence (using Twiddle Factor Matrix):

```
   3.0000 + 0.0000i
  -0.0000 + 1.0000i
   1.0000 - 0.0000i
   0.0000 - 1.0000i
```

```
% DFT using twiddle factor matrix
x = input('Enter the input sequence: ');
N = length(x);
W    = exp(-1i*2*pi*(0:N-1)'*(0:N-1)/N);
X    = W * x(:);
disp('DFT of the input sequence (using Twiddle Factor Matrix):');
disp(X);
```

Observation:

Enter the input sequence: [1 0 1 1]

IDFT of the input sequence (using Twiddle Factor Matrix):

```
   0.7500 + 0.0000i
  -0.0000 - 0.2500i
   0.2500 + 0.0000i
   0.0000 + 0.2500i
```

```
% IDFT using twiddle factor matrix
x = input('Enter the input sequence: ');
N = length(x);
W = exp(1i*2*pi*(0:N-1)'*(0:N-1)/N);
X_idft = (1/N) * (W * x(:));
disp('IDFT of the input sequence (using Twiddle Factor Matrix):');
disp(X_idft);
```

Result:

Computed DFT and IDFT using inbuilt and manual methods and Twiddle factor matrix and verified the output.

Experiment No:7 Date: 01/10/24

PROPERTIES OF DFT

Aim

To prove the following properties of DFT

- Linearity
- Convolution
- Multiplication
- Parseval's Theorem

Theory:

Linearity:

The DFT is a linear transformation, meaning that the DFT of the sum of two signals is equal to the sum of their individual DFTs, and multiplying a signal by a constant in the time domain results in the DFT being multiplied by the same constant.If x1(n) and x2(n) are two sequences and a and b are constants then:

DFT(ax1(n)+bx2(n))= a.DFT(x1(n))+b.DFT(x2(n))

Multiplication:

The DFT of a pointwise multiplication (element-wise product) of two signals in the time domain corresponds to the circular convolution of their DFTs in the frequency domain. If x1(n) and x2(n) are two signals then:

DFT{x1(n).x2(n)}= 1/N DFT{x(n)}⊛DFT{h(n)}

Convolution:

The DFT of the convolution of two sequences in the time domain is the element-wise multiplication of their DFTs in the frequency domain. If x1(n) and x2(n) are two signals, then:

DFT{x1(n)∗x2(n)}=DFT{x1(n)}·DFT{x2(n)}

Parseval's Theorem:

Parseval's theorem states that the total energy of a discrete-time signal (the sum of the squared magnitudes of the signal in the time domain) is equal to the total energy of its DFT (the sum of the squared magnitudes of the DFT coefficients).

Program:

```
%linearity property

clc;

clear all;

close all;

x1=input('Enter the first sequence:');
```

Observation:

Enter the first sequence:[1 2 3 4]

Enter the second sequence:[1 1 1]

LHS:

29.0000 + 0.0000i -4.0000 + 1.0000i -1.0000 + 0.0000i -4.0000 – 1.0000i

RHS:

29.0000 + 0.0000i -4.0000 + 1.0000i -1.0000 + 0.0000i -4.0000 – 1.0000i

LHS=RHS

Linearity Property Verified

```
x2=input('Enter the second sequence:');
a=2;
b=3;
l1=length(x1);
l2=length(x2);
if l1>l2
    x2=[x2 zeros(1,l1-l2)]
else
    x1=[x1 zeros(1,l2-l1)];
end
LHS=fft((a.*x1)+(b.*x2));
RHS=[a.*fft(x1)+b.*fft(x2)];
disp('LHS:');
disp(LHS);
disp('RHS:');
disp(RHS);
disp(['LHS=RHS')_
```

Observation:

LHS

8 7 6 9

RHS

8 7 6 9

Convolution property verified

```
%Convolution property
clc;

clear all;

close all;

x=input('enter sequence 1');

h=input('enter sequence 2');

N=max(length(x),length(h));

X=[x zeros(1,N-length(x))];

H=[h zeros(1,N-length(h))];

X1=fft(X);

H1=fft(H);

LHS= cconv(X,H,N);

RHS=ifft(X1.*H1);

disp(LHS);

disp(RHS);

if LHS==RHS

    disp('Convolution property verified');

else

    disp('Convolution property verified');

end
```

Observation:

enter the first sequence:
[1 2 3 4]

enter the second sequence:

[1 1 1]

```
   6.0000 + 0.0000i  -2.0000 - 2.0000i   2.0000 + 0.0000i  -2.0000 + 2.0000i


   6.0000 + 0.0000i  -2.0000 - 2.0000i   2.0000 + 0.0000i  -2.0000 + 2.0000i
```

```
%multiplication property
clc;
clear all;
close all;
x1=input('enter the first sequence:');
x2=input('enter the second sequence:');
l1=length(x1);
l2=length(x2);
n=max(l1,l2);
x1=[x1 zeros(1,n-l1)];
x2=[x2 zeros(1,n-l2)];
lhs=fft(x1.*x2);
X1=fft(x1);
X2=fft(x2);
rhs=cconv(X1,X2,n)/n;
disp(lhs);
disp(rhs);
```

Observation:

enter the first sequence:
[1 2 3 4]
enter the second sequence:
[1 1 1]
LHS
    6
RHS
    6

```
%parseval's theorem
clc;
clear all;
close all;
x1=input('enter the first sequence:');
x2=input('enter the second sequence:');
l1=length(x1);
l2=length(x2);
n=max(l1,l2);
x1=[x1 zeros(1,n-l1)];
x2=[x2 zeros(1,n-l2)];
lhs=sum(x1.*conj(x2));
rhs=sum(fft(x1).*conj(fft(x2)))/n;
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
```

Result:

Verified linearity, convolution, multiplication,and parseval’s properties of DFT

Experiment No:8 Date: 08/10/24

OVERLAP SAVE AND ADD METHOD

Aim

To perform linear convolution of two sequences using overlap save and add method

Theory:

The Overlap-Add and Overlap-Save methods are efficient techniques used to perform linear convolution of long signals with finite impulse response (FIR) filters using the Fast Fourier Transform (FFT). Both methods help reduce computational complexity by breaking a long signal into smaller chunks, processing them independently in the frequency domain, and then combining the results.

The Overlap Add method splits the input signal into overlapping segments, performs convolution on each segment using the FFT, and then adds the overlapping parts to reconstruct the final output. This method is efficient for filtering long signals by using FFT-based convolution.

The Overlap Save method also divides the input signal into segments, but unlike OLA, it saves the non-overlapping parts and discards the overlapping parts of the convolution output.

Program:

```
%overlap save method
clc;
clear all;
close all;
x=input("Enter sequence 1");
h=input("Enter sequence 2");
N=input('Enter length to divide');
if N<length(h)
   disp('not possible');
else
    xl=length(x);
    hl=length(h);
    L=N-hl+1;
    hnew=[h zeros(1,N-hl)];
    xnew=[zeros(1,hl-1),x,zeros(1,N-1)];
    y=[];
for i=1:L:length(xnew)-N+1
    XB=xnew(i:i+N-1);
    YB=ifft(fft(XB).*fft(hnew));
    y=[y,YB(hl:end)];
end
    disp(y(1:xl+hl-1));
```

```
end
```

Observation:

Enter sequence 1[3 -1 0 1 3 2 0 1 2 1]
Enter sequence 2[1 1 1]
Enter length to divide3
final convoluted sequence

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

Observation:

Enter the input sequence: [0 1 2 3 4 5 6 7 8 9]
Enter the filter sequence: [1 0 1]
Enter the segment length (choose N >= Lh): 3
final convoluted sequence:

```
     0     1     2     4     6     8    10    12    14    16     8     9
```

```
%overlap add method
clc;
clear all;
close all;
x = input('Enter the input sequence: ');
h = input('Enter the filter sequence: ');
Lx = length(x);
Lh = length(h);
N = input('Enter the segment length (choose N >= Lh): ');
if N < Lh
    error('Segment length N must be greater than or equal to filter
length');
end
x = [x, zeros(1, N - mod(Lx, N))];
Lx_padded = length(x);
y = zeros(1, Lx_padded + Lh - 1);
for i = 1:N:Lx_padded
    x_segment = x(i:i+N-1);
    y_segment = conv(x_segment, h);
    y(i:i+length(y_segment)-1)   =   y(i:i+length(y_segment)-1)   +
y_segment;
end
y = y(1:Lx + Lh - 1);
disp('final convoluted sequence:');
disp(y);
```

Result:

Implemented overlap add and overlap save method using MATLAB and verified the output

Experiment No: 9 Date:15/10/24

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.

Filters:

Lowpass: $$h(n) = \begin{cases} \frac{\Omega_c}{\pi} & n = 0 \\ \frac{\sin(\Omega_c n)}{n\pi} \text{ for } n \neq 0 & -M \leq n \leq M \end{cases}$$

Highpass: $$h(n) = \begin{cases} \frac{\pi - \Omega_c}{\pi} & n = 0 \\ -\frac{\sin(\Omega_c n)}{n\pi} \text{ for } n \neq 0 & -M \leq n \leq M \end{cases}$$

Bandpass: $$h(n) = \begin{cases} \frac{\Omega_H - \Omega_L}{\pi} & n = 0 \\ \frac{\sin(\Omega_H n)}{n\pi} - \frac{\sin(\Omega_L n)}{n\pi} \text{ for } n \neq 0 & -M \leq n \leq M \end{cases}$$

Bandstop: $$h(n) = \begin{cases} \frac{\pi - \Omega_H + \Omega_L}{\pi} & n = 0 \\ -\frac{\sin(\Omega_H n)}{n\pi} + \frac{\sin(\Omega_L n)}{n\pi} \text{ for } n \neq 0 & -M \leq n \leq M \end{cases}$$

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 mainlobe 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 cannotbe perfectly achieved using standard windows.

Program:

1. LOW PASS FILTER

```
clc;

clear all;

close all;

wc=0.5*pi;

N = 50;

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 = 50;
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.BANDPASS FILTER

```
clc;
clear all;
close all;
wc1=0.5*pi;
wc2=0.9*pi;
N = 50;
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');
```

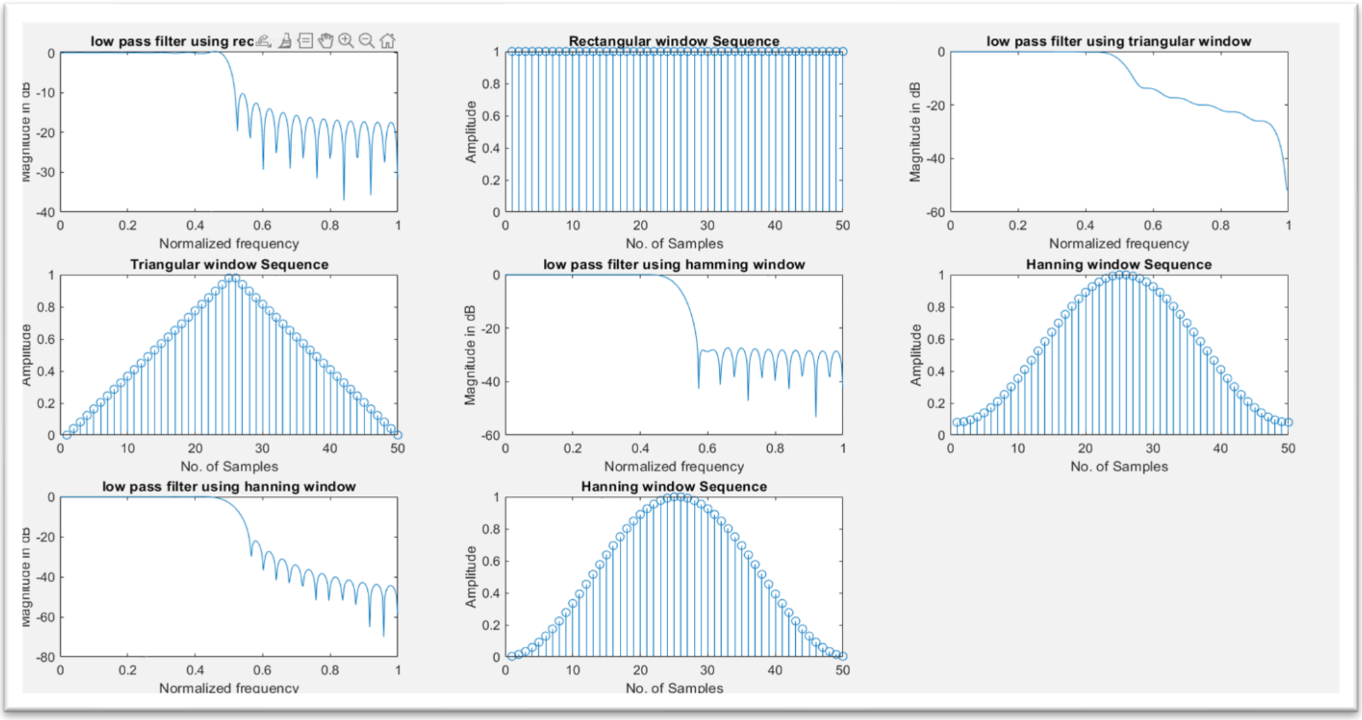
```
ylabel('Magnitude in dB');
subplot(3,3,8);
stem(whn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

4.BANDSTOP FILTER

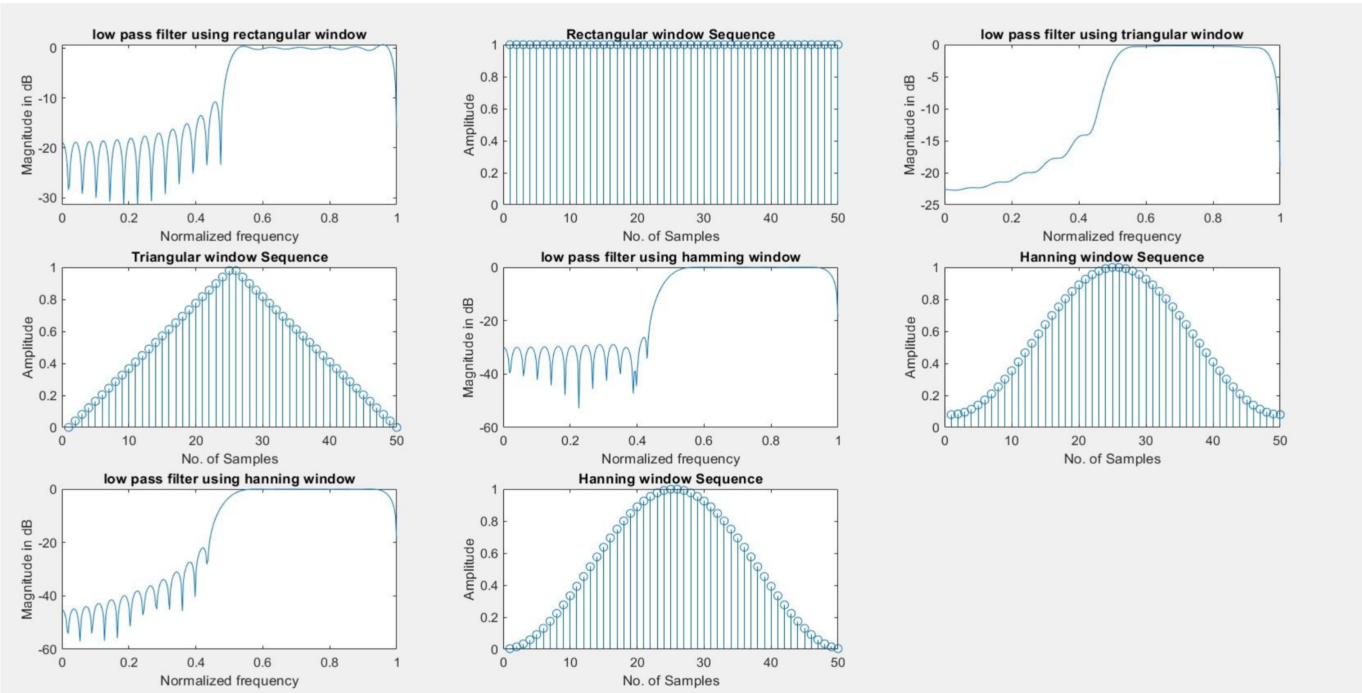
```
clc;
clear all;
close all;
wc1=0.5*pi;
wc2=0.9*pi;
N = 50;
alpha = (N-1)/2;
eps = 0.001;
n = 0:1:N-1;
hd = (sin(wc1*(n-alpha+eps))-sin(wc2*(n-alpha+eps))+sin(pi*(n-alpha)))./(pi*(n-alpha+eps));
wr = boxcar(N);
wt=bartlett(N);
wh=hamming(N);
whn=hanning(N);
hn1 = hd.*wr';
hn2 = hd.*wt';
hn3 = hd.*wh';
hn4 = hd.*whn';
w = 0:0.01:pi;
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
```

Observation:

1.LOW PASS FILTER

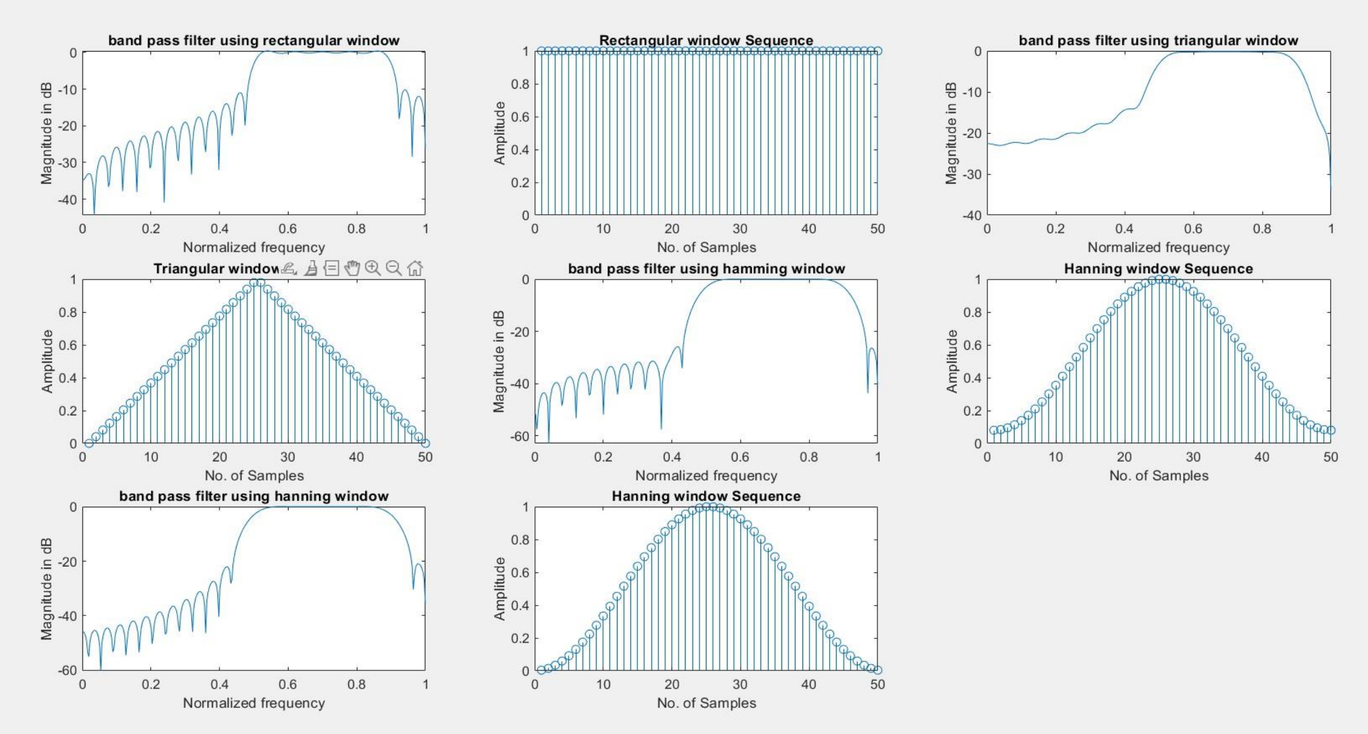


2.HIGH PASS FILTER

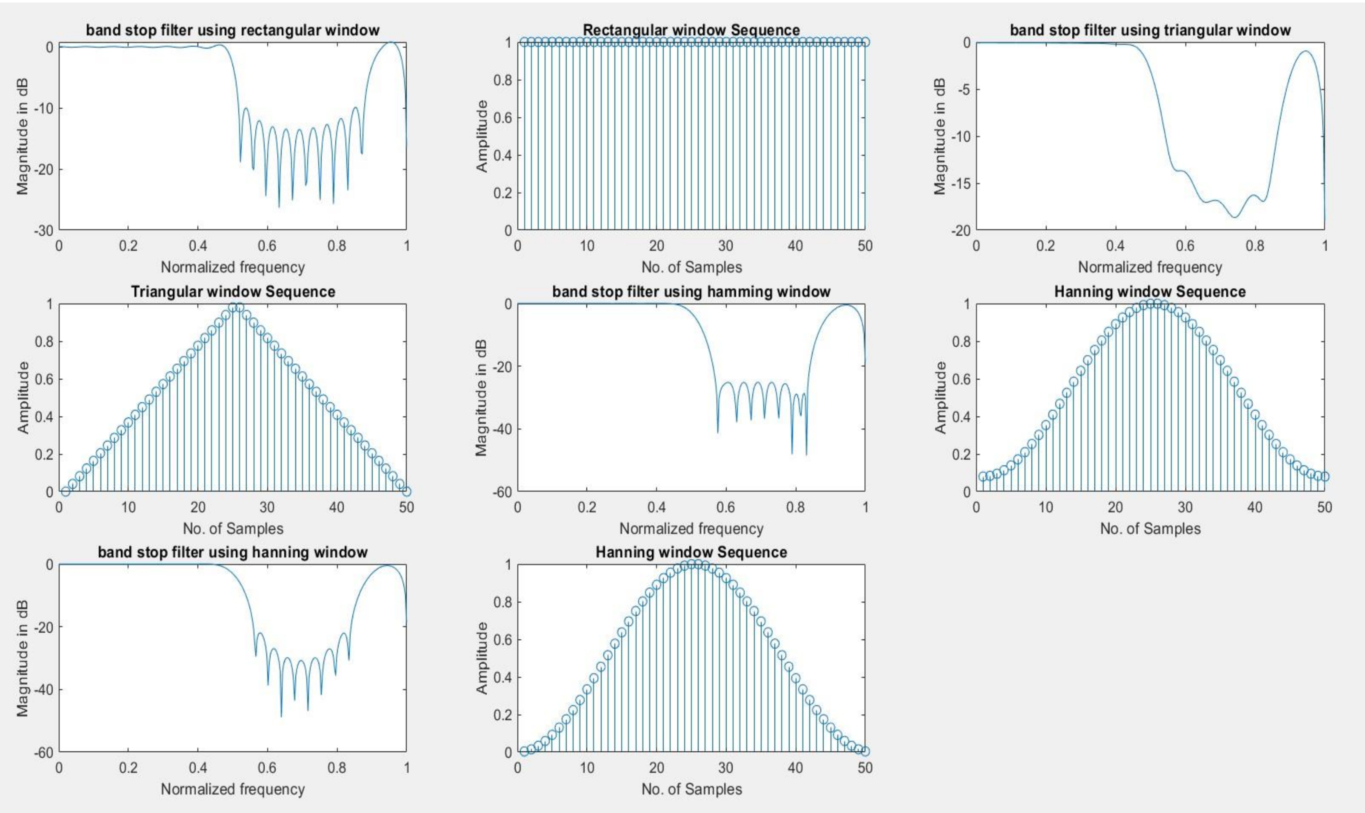


```
h3 = freqz(hn3,1,w);
h4 = freqz(hn4,1,w);
subplot(3,3,1);
plot(w/pi,10*log10(abs(h1)));
title('band stop filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
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');
```

3.BAND PASS FILTER



4.BAND STOP FILTER



```
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

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

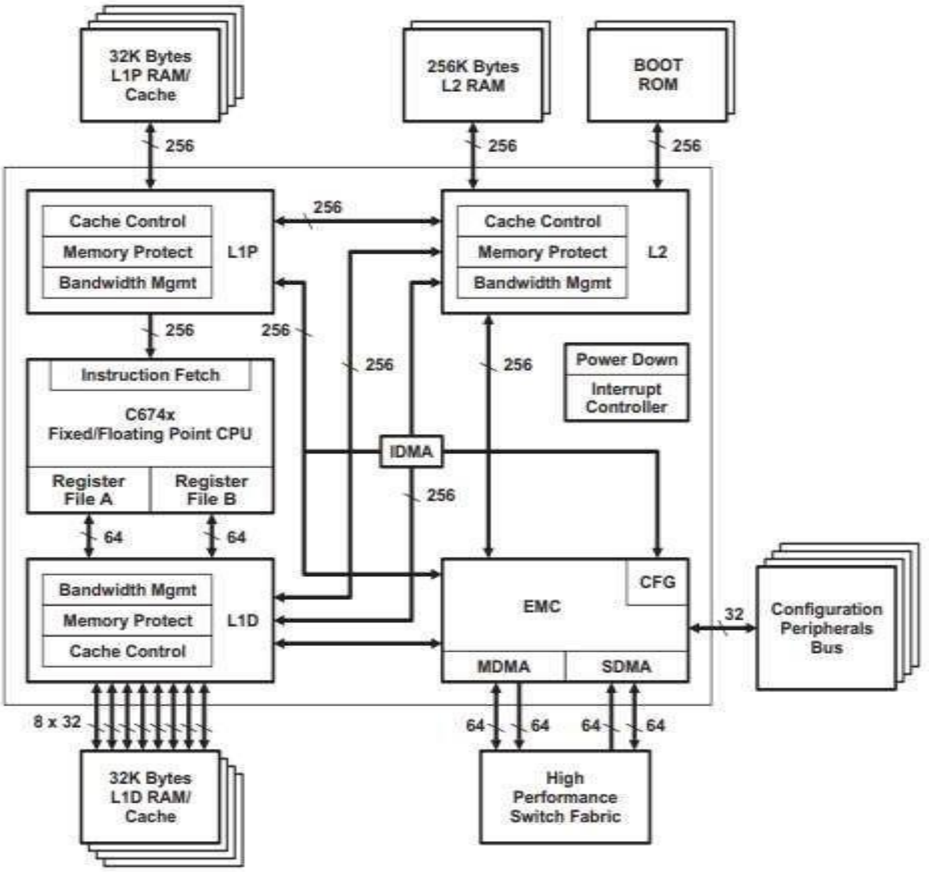


FIGURE: TMS320C 674X DSP CPU BLOCK DIAGRAM

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 used for 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 always an 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 slots per 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

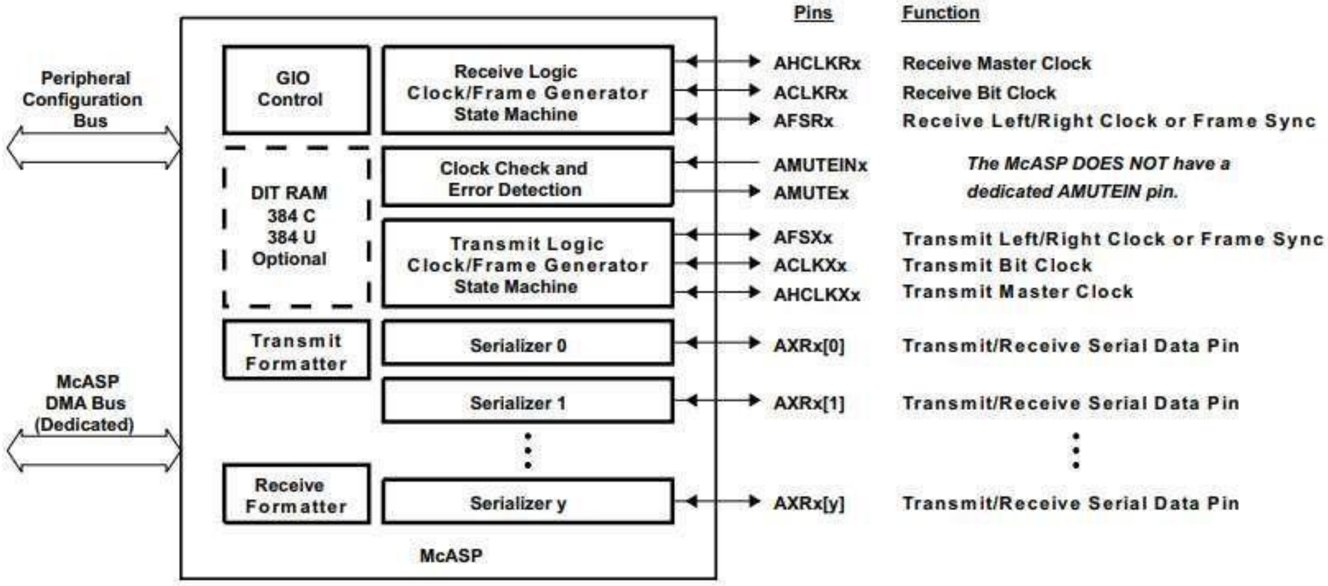
• All unused pins GPIO-capable

• 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 determined by the specific ADC circuitry on the codec, which is 6 V p-p with the onboard codec. After the captured 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 filtering functions are performed by the single-chip codec AIC23 on board the DSK.



Result:

Familiarized the input and output ports of dsp board.

Experiment No: 11 Date: 29/10/24

Generation of Sine Wave using DSP Kit

Aim:

To generate a sine wave using DSP Kit.

Theory:

Sinusoidal are the smoothest 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 increases from a value of 0 at 0° angle to a maximum value of 1 at 90°, it further reaches its minimum value of -1 at 270° and then returns to 0 at 360°. After any angle greater than 360°, the sinusoidal signal repeats the values so we can say that 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 value of 1 and a minimum value of -1. We can also observe that the wave begins to repeat its value after a period or angle value of 2π hence periodicity of sinusoidal signal is 2π.

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

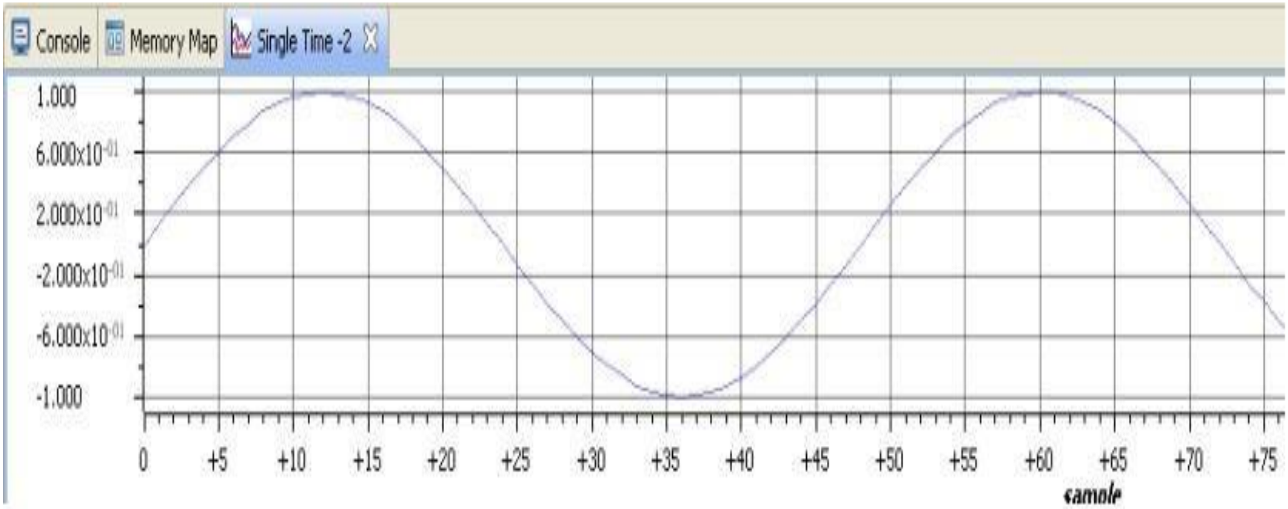
Procedure

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,s).
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 3.14159
float s[100];
void main()
{
int i;
float f=100, Fs=10000;
for(i=0;i<100;i++)
```

Observation:



```
s[i]=sin(2*pi*f*i/Fs);
}
```

Result:

Generated sine wave using DSP Kit.

Experiment No:12 Date:29/10/24

Linear Convolution using DSP Kit

Aim:

To perform linear convolution of two sequences using DSP Kit.

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:

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

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.

Procedure

1. Set Up New CCS Project
Open Code Composer Studio.
Go to File → New → CCS Project.
Target Selection: Choose C674X Floating point DSP, TMS320C6748.
Connection: Select Texas Instruments XDS 100v2 USB Debug Probe.
Name the project and click Finish.

2. Write and Configure the Program
Write the C code for generating and storing a sine wave, configuring it to access data at specified memory locations.
Assign the input Xn and filter Hn values to specified addresses:
Xn: Start at 0x80010000, populate subsequent values at offsets like 0x80010004 for each additional input.
Hn: Start at 0x80011000 with similar offsets for additional values.
Lengths of Xn and Hn should be defined at 0x80012000 and 0x80012004, respectively.

3. Configure Output Location in Code
In the code, configure the output to store convolution results at specific memory addresses starting from 0x80013000, with each result at an offset of 0x04.

4. Save the Program
Go to File → Save As and save the code with a filename like main.c.
Remove any default main.c program that might exist in the project.

5. Build and Debug the Program
Select Debug to build and load the program on the DSP.
Once the build is complete, select Run to execute.

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

6.Execute and Verify Output
In the Debug perspective, click Resume to run the code.
Use the Memory Browser in Code Composer Studio to verify the output at the memory location 0x80013000:
Check 0x80013000 for the first convolution result, 0x80013004 for the second, and so on.
Cross-check the values with the expected convolution results for accuracy.

Program:

```
#include<fastmath67x.h>
#include<math.h>
void main()
{
int  *Xn,*Hn,*Output;
int *XnLength,*HnLength;
int i,k,n,l,m;
Xn=(int *)0x80010000; //input x(n)
Hn=(int *)0x80011000; //input h(n)
XnLength=(int *)0x80012000; //x(n) length
HnLength=(int *)0x80012004; //h(n) length
Output=(int *)0x80013000; // output address
l=*XnLength; // copy x(n) from memory address to variable l
m=*HnLength; // copy h(n) from memory address to variable m
for(i=0;i<(l+m-1);i++) // memory clear
{
Output[i]=0; // o/p array
Xn[l+i]=0; // i/p array
Hn[m+i]=0; // i/p array
}
for(n=0;n<(l+m-1);n++)
{
for(k=0;k<=n;k++)
{
Output[n] =Output[n] + (Xn[k]*Hn[n-k]); // convolution operation.
}
}
}
```

Result:

Performed Linear Convolution using DSP Kit.

.