THANGAL KUNJU MUSALIAR COLLEGE OF ENGINEERING
KOLLAM – 691 005

ELECTRONICS AND COMMUNICATION ENGINEERING

22ECL509

DIGITAL SIGNAL PROCESSING LAB

LABORATORY RECORD

YEAR 2024-25

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. **ANUSREE PRADEEP** of **5th** Semester class (Roll No. **[B22ECB15]** 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 : ***TKM22EC024***

Staff Member in-charge   *External Examiner*

Date

INDEX

Experiment No: 01   Date: 01/08/2024

SIMULATION OF BASIC TEST SIGNALS

AIM:

To generate continuous and discrete waveforms of

1. Unit Impulse signal

2. Unit Step signal

3. Ramp signal

4. Sine wave

5. Cosine wave

6. Square wave-bipolar

7. Square wave-unipolar

8. Triangular wave

9. Exponential signal

THEORY:

1. Unit Impulse Signal:

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

2. Unit Step Signal:

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

3. Ramp Signal:

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

4. Sine Signal:

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

5. Cosine Signal:

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

6. Square wave-bipolar:

- 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| ≤ τ/2, p(t) = 0 otherwise**

7. Square wave-unipolar:

- 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| ≤ τ/2, p(t) = 0 otherwise (assuming A is positive)**

8. Triangular Signal:

- 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**

9. Exponential Signal:

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

PROGRAM:

```
clc;
clear all;
close all;
subplot(3,3,1);
t = -5:1:5;
y = [zeros(1,5),ones(1,1),zeros(1,5)];
stem(t,y);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Impulse Signal");
subplot(3,3,2);
t2 = 0:0.01:1;
f = 5;
y2 = square(2*pi*f*t2);
stem(t2,y2);
```

```matlab
hold on;
plot(t2,y2);
xlabel("Time(s)");
ylabel("Amplitude");
title("Bipolar Pulse Signal");
legend("Discrete","Continuous");
subplot(3,3,3);
t3 = 0:0.1:1;
f = 5;
y3 = abs(square(2*pi*f*t3));
stem(t3,y3);
hold on;
plot(t3,y3);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unipolar Pulse Signal");
legend("Discrete","Continuous");
subplot(3,3,4);
t4 = -5:1:5;
y4 = t4 .*(t4>=0);
stem(t4,y4);
hold on;
plot(t4,y4);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Ramp Signal");
legend("Discrete","Continuous");
subplot(3,3,5);
t5 = 0:0.025:1;
f = 10;
```

```
y5 = sawtooth(2*pi*f*t5,0.5);

stem(t5,y5);

hold on;

plot(t5,y5);

xlabel("Time(s)");

ylabel("Amplitude");

title("Triangular Signal");

legend("Discrete","Continuous");

subplot(3,3,6);

t6 = 0:0.001:1;

f = 10;

y6 = sin(2*pi*f*t6);

stem(t6,y6);

hold on;

plot(t6,y6);

xlabel("Time(s)");

ylabel("Amplitude");

title("Sine Wave");

legend("Discrete","Continuous");

subplot(3,3,7);

t7 = 0:0.001:1;

f = 10;

y7 = cos(2*pi*f*t7);

stem(t7,y7);

hold on;

plot(t7,y7);

xlabel("Time(s)");

ylabel("Amplitude");

title("Cosine Wave");

legend("Discrete","Continuous");
```

OBSERVATION
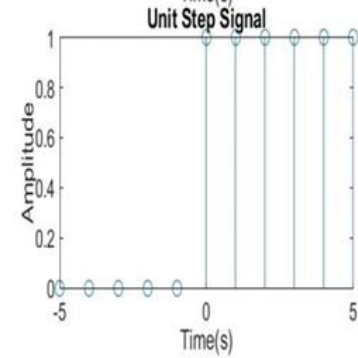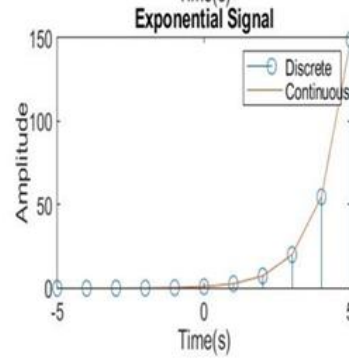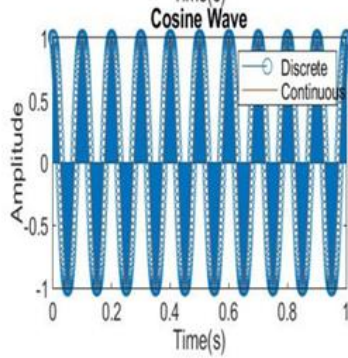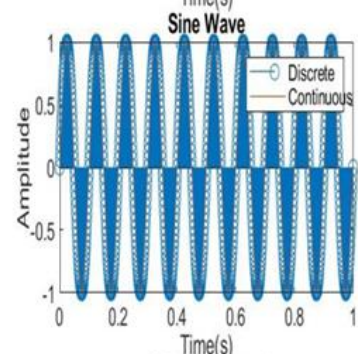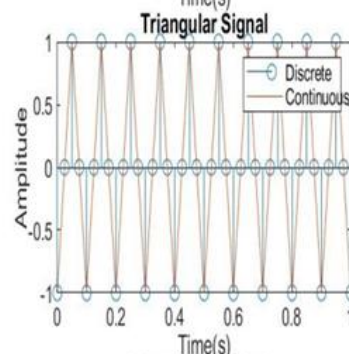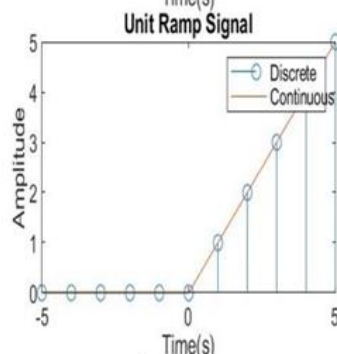
```
subplot(3,3,8);

t8 = -5:1:5;

y8 = exp(t8);

stem(t8,y8);

hold on;

plot(t8,y8);

xlabel("Time(s)");

ylabel("Amplitude");

title("Exponential Signal");

legend("Discrete","Continuous");


subplot(3,3,9);

t9 = -5:1:5;

y9 = [zeros(1,5),ones(1,6)];

stem(t9,y9);

xlabel("Time(s)");

ylabel("Amplitude");

title("Unit Step Signal");
```
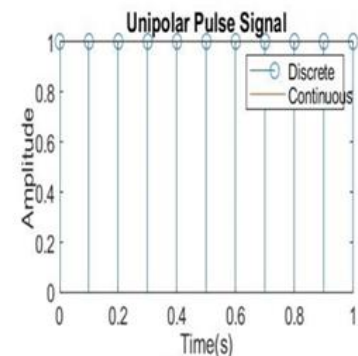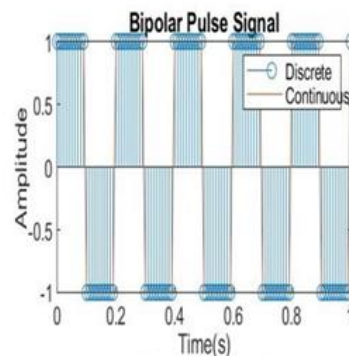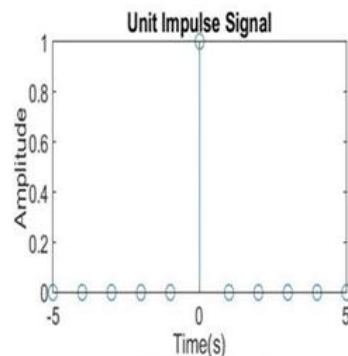
RESULT:

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

Experiment No: 2   Date: 08/08/2024

VERIFICATION OF SAMPLING THEOREM

AIM:

To verify sampling theorem using sinusoidal signal in matlab.

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.

$$fs \geq 2 \cdot fmax$$

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:

```
%verification of sampling theorem
clc;
clear all;
close all;
%original signal
t=0:0.01:1;
fm=10;
y=sin(2*pi*fm*t);
figure;
subplot(2,2,1);
plot(t,y);
```

```matlab
hold on;
stem(t,y,".");
xlabel("Time");
ylabel("Amplitude");
title("Original signal");
legend("Continuous","Discrete");
%less than Nyquist rate
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,'.');
xlabel("Time");
ylabel("Amplitude");
title("Undersampling");
legend("Continuous","Discrete");
%equal to nyquist rate
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,'.');
xlabel("Time");
ylabel("Amplitude");
title("Nyquist sampling");
legend("Continuous","Discrete");
```

OBSERVATION:
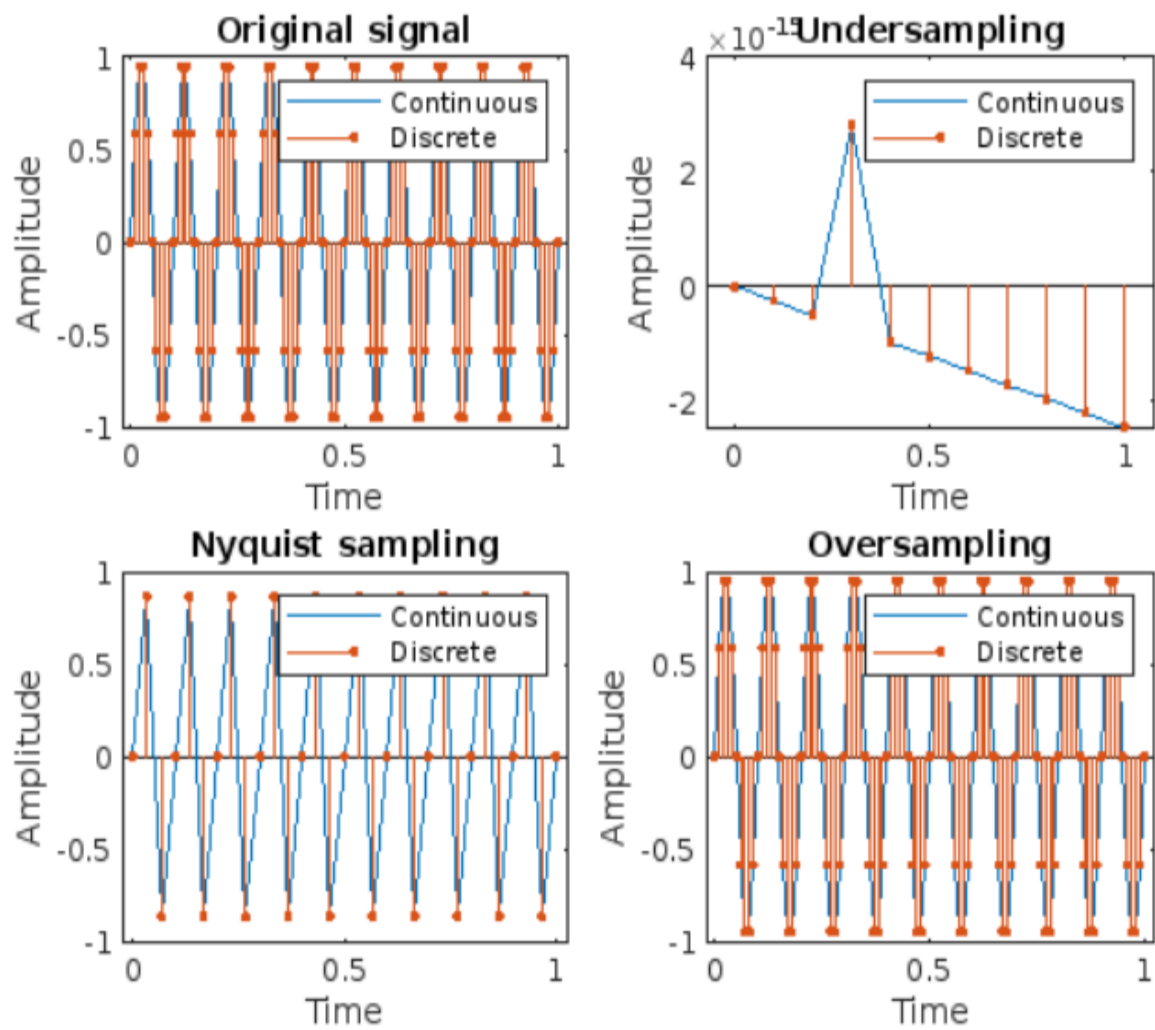
```
%greater than nyquist rate

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,'.');

xlabel("Time");

ylabel("Amplitude");

title("Oversampling")

legend("Continuous","Discrete");
```

RESULT:

Verified Sampling Theorem using MATLAB.

Experiment No:03   Date:08/08/2024

LINEAR CONVOLUTION

AIM:

To find the linear convolution with and without using function.

(a) Using Built-in function

(b) Without Using Built-in function

THEORY:

Linear convolution is a mathematical operation used to combine two signals to produce a third signal.

It's a fundamental operation in signal processing and systems theory.

Mathematical Definition:

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

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

Applications:

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

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

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

PROGRAM:

(a) using built-in function:

```
clc;

clear all;

close all;

x=input('Enter input x(n):');

x_ind=input('Enter index of x(n):');

h=input('Enter input h(n):');

h_ind=input('Enter index of h(n):');
```

```
y=conv(x,h);
y_ind=min(x_ind)+min(h_ind):max(x_ind)+max(h_ind);
disp(y);
figure();
subplot(3,1,1);
stem(x_ind,x);
xlabel('Time index');
ylabel('Amplitude');
title('Input x(n)');
subplot(3,1,2);
stem(h_ind,h);
xlabel('Time index');
ylabel('Amplitude');
title('Input h(n)');
subplot(3,1,3);
stem(y_ind,y);
xlabel('Time index');
ylabel('Amplitude');
title('Linear convolution');
```

(b) without using built-in function:

```
clc;
clear all;
close all;
x=input('Enter input x(n):');
x_ind=input('Enter index of x(n):');
h=input('Enter input h(n):');
h_ind=input('Enter index of h(n):');
x_len=length(x);
h_len=length(h);
```

SERI

OBSERVATION:

(a) using built-in function:
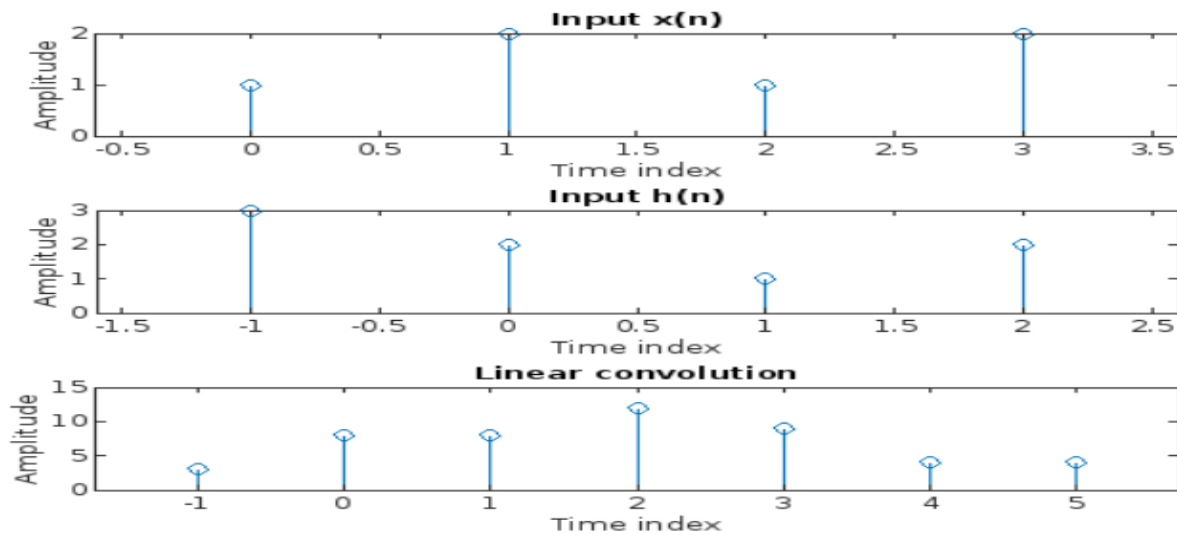
3   8   8   12   9   4   4

(b) without using built-in function:

3   8   8   12   9   4   4

```matlab
y_ind=min(x_ind)+min(h_ind):max(x_ind)+max(h_ind);
y=zeros(y_ind);
for i=1:x_len
    for j=1:h_len
        y(i+j-1)=y(i+j-1)+x(i)*h(j);
    end
end
disp(y);
figure();
subplot(3,1,1);
stem(x_ind,x);
xlabel('Time index');
ylabel('Amplitude');
title('Input x(n)');
subplot(3,1,2);
stem(h_ind,h);
xlabel('Time index');
ylabel('Amplitude');
title('Input h(n)');
subplot(3,1,3);
stem(y_ind,y);
xlabel('Time index');
ylabel('Amplitude');
title('Linear convolution');
```

RESULT:

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

Experiment No:04   Date: 22/08/2024
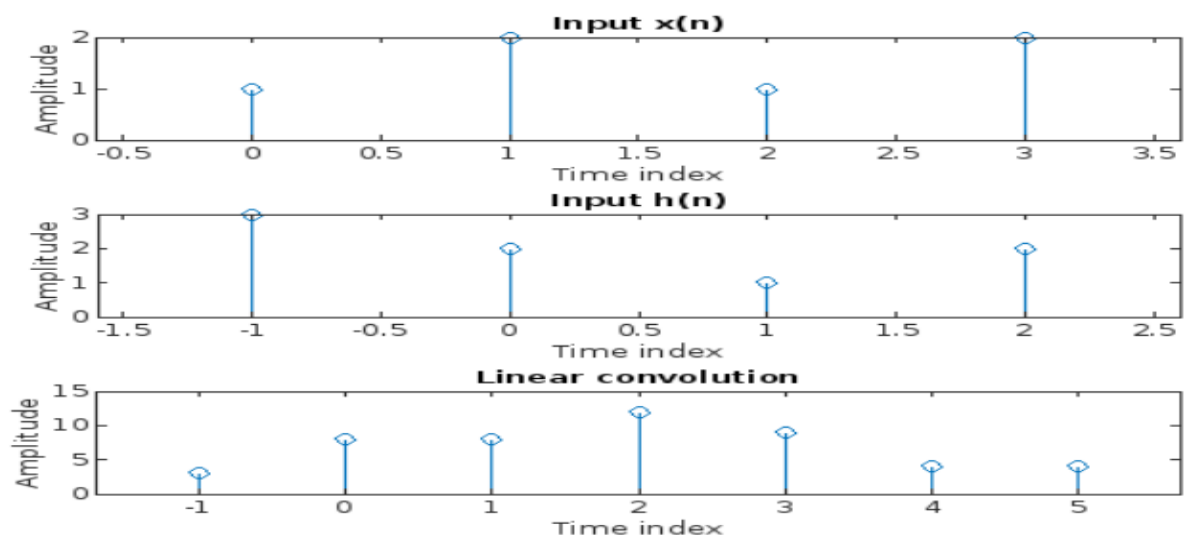
CIRCULAR CONVOLUTION

AIM:

To find circular convolution

       (a) Using FFT and IFFT.

       (b) Using Concentric Circle Method.

       (c) Using Matrix Method.

THEORY:

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

Mathematical Definition:

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

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

Applications:

• Discrete-Time Filtering: Circular convolution is used for filtering discrete-time signals.

• Digital Signal Processing: It's a fundamental operation in many digital signal processing algorithms.

• Cyclic Convolution: In certain applications, such as cyclic prefix OFDM, circular convolution is used to simplify the implementation of linear convolution.

PROGRAM:

(a) Using FFT and IFFT.

```
clc;
close all;
clear all;
x1=[1 2 3 4];
h1=[1 2 1 0];
nx=length(x1);
nh=length(h1);
n=max(nh,nx);
newx=[x1 zeros(n-nx)];
newh=[h1 zeros(n-nh)];
```

```
x=fft(newx);

h=fft(newh);

hx=x.*h;

y=ifft(hx);

disp("Using FFT and IFFT:")

disp(y);
```

(b) Using Concentric Circle Method.

```
clc;

clear all;

close all;

x=[1 2 3 4];

h=[1 2 1 0];

x=x(:,end:-1:1);

for i=1:length(x)

    x=[x(end) x(1:end-1)];

    h1=h;

    y(i)=sum(x.*h);

end

disp("Using Concentric Circle Method:")

disp(y);
```

(c) Using Matrix Method.

```
clc;

close all;

clear all;

xn=[1 2 3 4];

hn=[1 2 1 0];

x=[];

xn=xn(:,end:-1:1);

for i=1:length(xn)

    xn=[xn(end) xn(1:end-1)];
```

OBSERVATIONS:

(a) Using FFT and IFFT:

     12   8   8   12

(b) Using Concentric Circle Method:

     12   8   8   12

(c) Using matrix method:

     12   8   8   12

```
    x=[x;xn];
end
y=x*hn';
disp("Using matrix method:");
disp(y);
```

RESULT:

Performed Circular Convolution using (a) FFT and IFFT; (b) Concentric Circle method;

(c) Matrix method and verified result.

Experiment No: 05   Date:12/09/2024

LINEAR CONVOLUTION USING CIRCULAR CONVOLUTION AND VICE VERSA

AIM:

 (a) To perform Linear Convolution using Circular Convolution.

 (b) To perform Circular Convolution using Linear Convolution.

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.

PROGRAM:

(a) Linear Convolution using Circular Convolution

```
clc;
close all;
clear all;
x = [1 2 3 4];
h = [1 1 1 ];
l=length(x);
m=length(h);
n=l+m-1;
x=[x zeros(1,n-l)];
h=[h zeros(1,n-m)];
x1=fft(x);
h1=fft(h);
y1=x1.*h1;
y=ifft(y1);
disp(y);
```

(b) Circular convolution using Linear Convolution

```
clc;
close all;
clear all;
x = [1 2 3 4];
h = [1 1 1 ];
y=conv(x,h);
l=length(x);
m=length(h);
n=max(l,m);
r1=y(1:n);
r2=[y(n+1:end) zeros(1:length(y)-n)];
```

OBSERVATION:

(a) Linear Convolution using Circular Convolution

     1   3   6   9   7   4

(b) Circular convolution using Linear Convolution

     8   7   6   9

```
r=r1+r2;
```

```
disp(r);
```

RESULT:

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

Experiment No:06   Date:12/09/2024

DISCRETE FOURIER TRANSFORM AND

INVERSE DISCRETE FOURIER TRANSFORM

AIM:

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

plot of DFT

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

THEORY:

Discrete Fourier Transform (DFT):

The DFT converts a discrete-time signal from the time domain to the frequency domain, representing it as a sum of complex exponentials. For a sequence of length N, the DFT is:

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

where k=0, 1…, N-1. It is widely used for spectral analysis and filtering.

Inverse Discrete Fourier Transform (IDFT):

The IDFT converts the frequency-domain representation back to the time domain ensuring complete reconstruction of the original signal.

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{\frac{j2\pi kn}{N}}$$

Applications:

- Spectrum (Analysis)
- Filtering
- Compression
- Modulation
- Convolution
- Demodulation
- Equalization
- Restoration
- Detection
- Estimation

PROGRAM:

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

%plotting
k=0:N-1;
magX=abs(X);
phaseX=angle(X);
subplot(2,1,1);
stem(k,magX);
title("Magnitude Plot");
hold on;
```

OBSERVATION

enter sequence:[1 1 1 0]

enter the N point:8

X

 Columns 1 through 7

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

0.2929 - 0.2929i -0.0000 + 1.0000i

 Column 8

  1.7071 + 1.7071i

round(X)

 Columns 1 through 7

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

0.0000 + 0.0000i   0.0000 + 1.0000i

 Column 8

  2.0000 + 2.0000i

fft

 Columns 1 through 7

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

0.2929 - 0.2929i   0.0000 + 1.0000i

 Column 8

  1.7071 + 1.7071i

```matlab
plot(k,magX);
subplot(2,1,2);
stem(k,phaseX);
hold on;
title("Phase Plot");
plot(k,phaseX);


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

IDFT

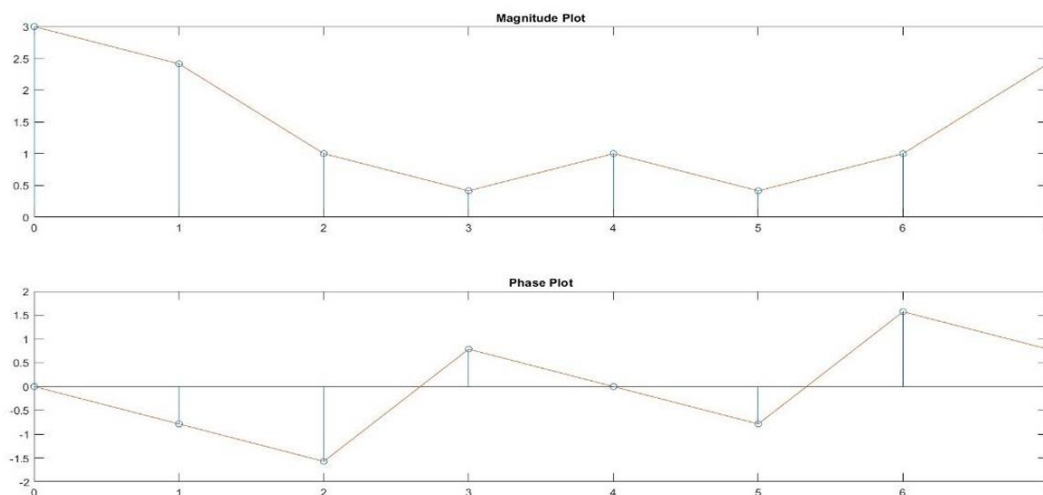enter sequence: [3.0000 + 0.0000i   1.7071 - 1.7071i   0.0000 - 1.0000i   0.2929 + 0.2929i

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

enter the n point:8

x

  0.7866 - 0.2134i

  0.6982 - 0.0000i

  0.7866 + 0.2134i

 -0.0000 + 0.3018i

  0.2134 + 0.2134i

  0.3018 - 0.0000i

  0.2134 - 0.2134i

  0.0000 - 0.3018i

round(x)

   1

   1

   1

   0

   0

   0

   0

   0

 ifft

Columns 1 through 7

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

0.2134i   0.3018 + 0.0000i   0.2134 - 0.2134i

Column 8

0.0000 - 0.3018i

```
%DFT using twiddle factor matrix
clc;
clear;
close all;
x=[1 2 3 4];
N=length(x);
n=0:N-1;
k=n';
W=exp(-1i*2*pi*(k*n)/N);
X=x*W;
disp('Using twiddle factor');
disp(X);
disp('Using built-in function');
disp(fft(x));
%IDFT using twiddle factor matrix
clc;
clear;
close all;
X=[10  -2 + 2i  -2  -2 - 2i];
N=length(X);
n=0:N-1;
k=n';
W=exp(1i*2*pi*(k*n)/N);
x=X*W/N;
disp(x);
disp(ifft(X));
```

DFT using twiddle factor

Using twiddle factor

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

Using built-in function

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

IDFT using twiddle factor

Using twiddle factor

  1.0000 + 0.0000i   2.0000 + 0.0000i   3.0000 - 0.0000i   4.0000 - 0.0000i

Using built-in function

  1   2   3   4

RESULT:

Performed

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

    (2) IDFT using inbuilt function, without using inbuilt function and using twiddle factor matrix and verified the result.

Experiment No: 07   Date:03/10/2024

PROPERTIES OF DFT

AIM:

Verify following properties of DFT using Matlab.

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:

$$\textbf{DFT\{a·x1[n]+b·x2[n]\} = a·DFT\{x1[n]\}+b·DFT\{x2[n]\}}$$

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:

$$\textbf{DFT\{x1[n]}\circledast\textbf{x2[n]\}=DFT\{x1[n]\}·DFT\{x2[n]\}}$$

4. Multiplication Property:

The multiplication property of DFT states that pointwise multiplication in the time domain

corresponds to circular convolution in the frequency domain:

$$\textbf{DFT\{x1[n]·x2[n]\} = } \frac{1}{N}[\text{DFT\{x1[n]\}}\circledast\text{DFT\{x2[n]\}}]$$

PROGRAM:

```matlab
%Linearity Property
clc;
clear all;
close all;
x=input("enter first sequence");
h=input("enter second 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
```

```
%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


%Convolution Property
clc;
clear all;
close all;
x=input("enter first sequence");
h=input("enter second sequence:");
N=max(length(x), length(h));
xn=[x zeros(N-length(x))];
hn=[h zeros(N-length(h))];
```

OBSERVATION

1. Linearity Property

enter first sequence[1 2 3 4]

enter second 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 second sequence:[1 1 1 1]

LHS

   10   10   10   10

RHS

   10   10   10   10

Circular Convolution verified

```matlab
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
%Multiplication Property
clc;
clear all;
close all;
x=input("enter first sequence");
h=input("enter second 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);
```

4.Multiplication Property

enter first sequence[1 2 3 4]

enter second 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

```
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

Experiment No: 08   Date:10/10/2024

OVERLAP ADD AND OVERLAP SAVE METHOD

AIM:

Implement overlap add and overlap save method using Matlab.

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:

```
%Overlap Add
x=input("Enter the sequence")
p=length(x);
h=input("Enter the sequence");
q=length(h);
cc=cconv(x,h);
r=0;
v=ceil(p/q);
while r ~= 0
p = p + 1;
r = mod(p,q);
if r==0
break
end
end
x=[x zeros(1,p)];
Lx=length(x);
Lh=length(h);
Lb = Lh; %Lb=blockLength
a=1;
b=Lb;
x=[x zeros(1,Lb)];
h=[h zeros(1,Lb-1)];
y1=[zeros(1,2*Lb-1)];
y=[];
for i =1:v+1
x1=x(a:b);
a=a+Lb;
```

```
b=b+Lb;
x1=[x1 zeros(1,Lb-1)];
y2=cconv(x1,h,2*Lb-1);
y3=y1+y2;
y=[y y3(1:Lb)];
y1=[y2(Lb+1:end) zeros(1,Lb)];
end
disp(cc);
disp(y);

% Overlap Save
x=input("Enter the sequence:);
p=length(x);
h=input("Enter the sequence:);
q=length(h);
cc=cconv(x,h);
r=0;
v=ceil(p/q);
while r ~= 0
p = p + 1;
r = mod(p,q);
if r==0
break
end
end
x=[x zeros(1,p)];
Lx=length(x);
Lh=length(h);
Lb = Lh; %Lb=blockLength
a=1;
```

```
b=Lb;
x=[x zeros(1,Lb)];
h=[h zeros(1,Lb-1)];
y1=[zeros(1,2*Lb-1)];
y=[];
for i =1:v+1
x1=x(a:b);
a=a+Lb;
b=b+Lb;
y2=[y1(Lb+1:end) x1];
y3=cconv(y2,h,2*Lb-1);
y=[y y3(Lb:end)];
y1=y2;
end
disp(cc);
disp(y);
```

RESULT:

Performed Overlap add and Overlap Save method.

Experiment No 09   Date:17/10/2024

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 Window	w(n) = 1
Hamming Window	w(n) = 0.54 - 0.46 * cos(2 * pi * n / (N - 1))
Hanning Window	w(n) = 0.5 * (1 - cos(2 * pi * n / (N - 1)))
Triangular Window	w(n) = 1 - 2*abs(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) = 0.42 - 0.5 * cos(2 * pi * n / (N - 1)) + 0.08 * cos(4 * pi * n / (N - 1))

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;
```

```matlab
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*(nalpha+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 = 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*(nalpha+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');
```

```matlab
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 = 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*(nalpha))). /(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);
```
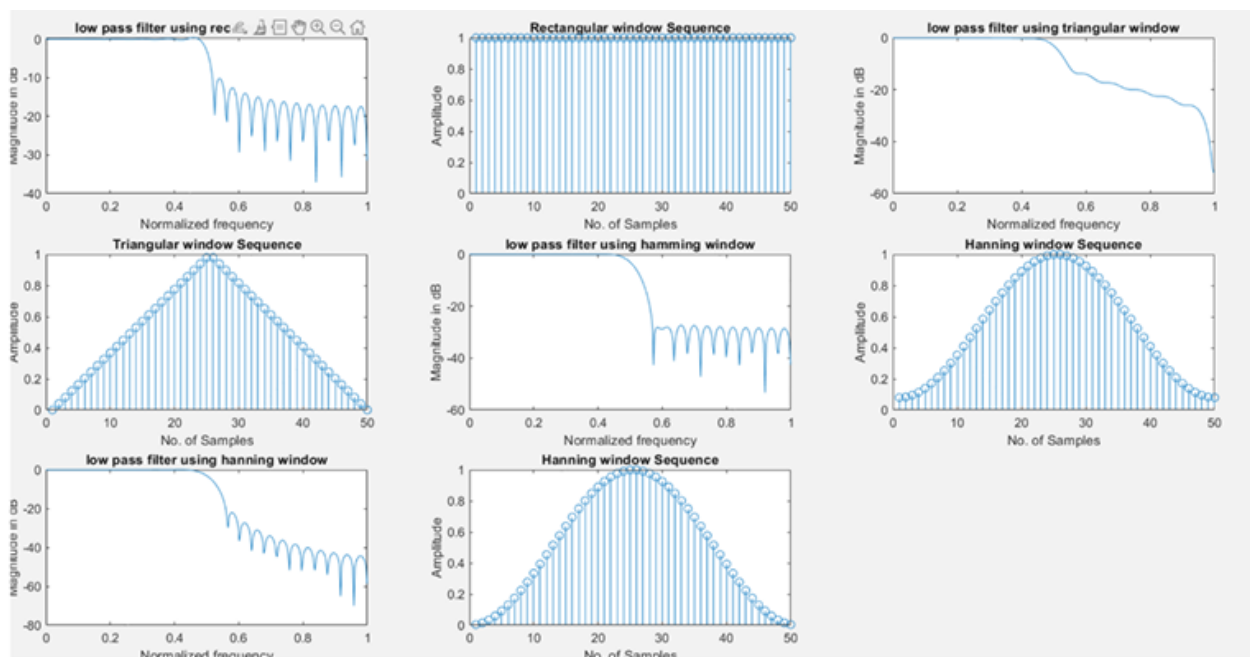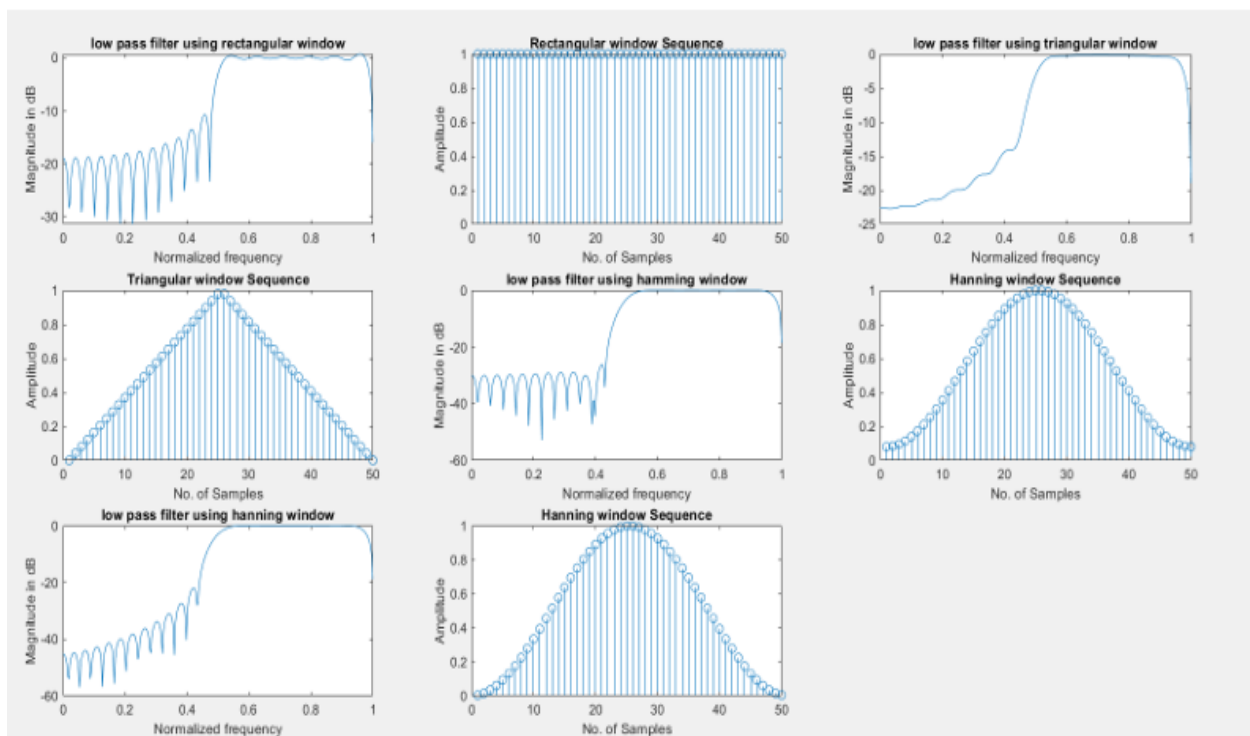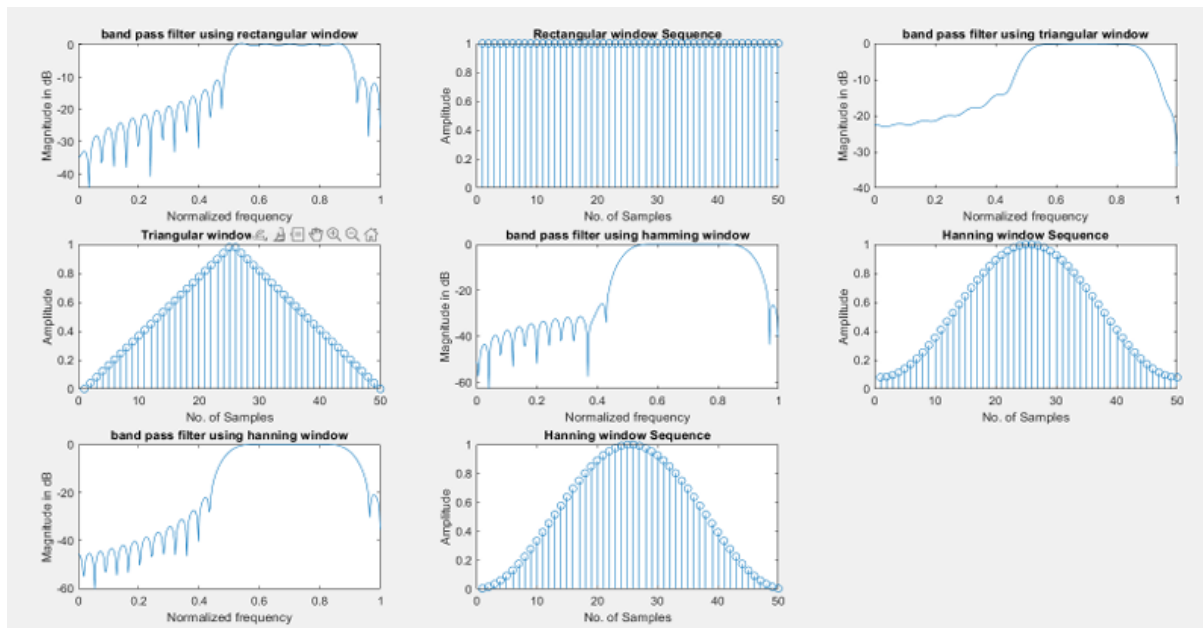
OBSERVATION

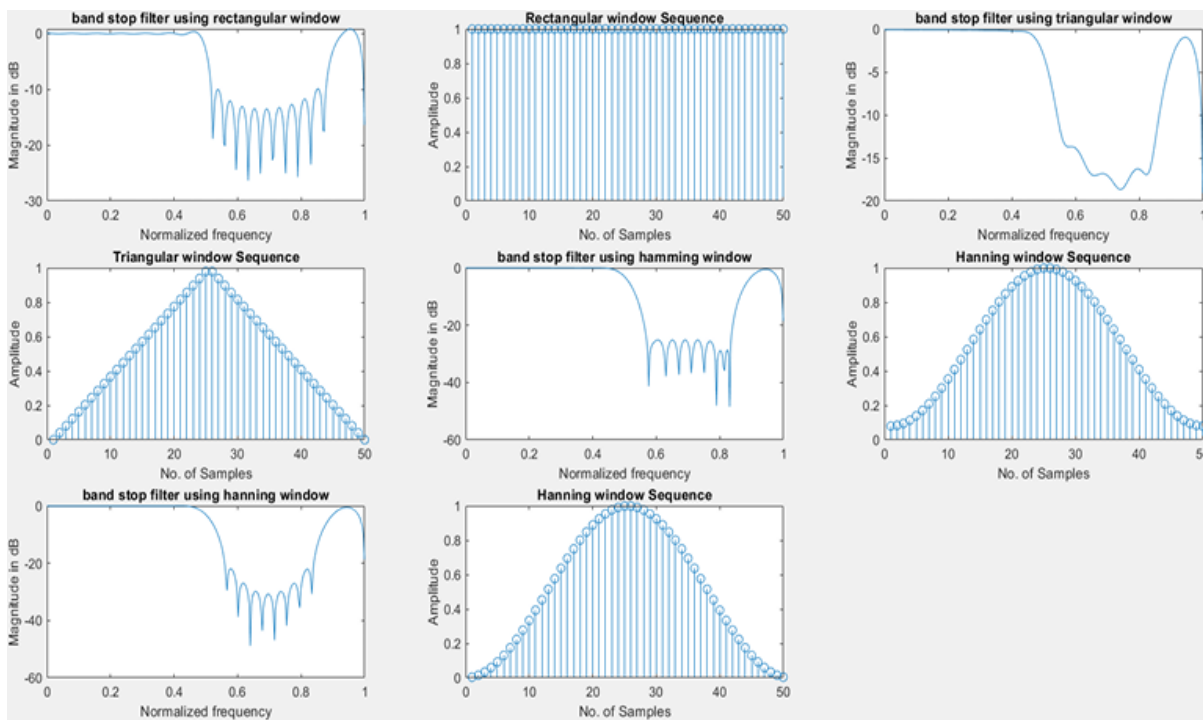1.LOW PASS FILTER

2.HIGH PASS FILTER

```
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');
```

3.BAND PASS FILTER

4.BAND STOP FILTER

```
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:24 /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:
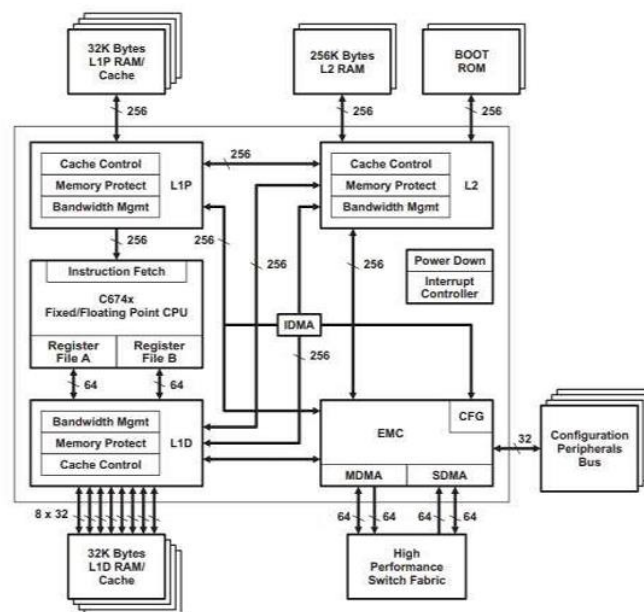
TMS 320C674x DSP CPU

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 data words 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.
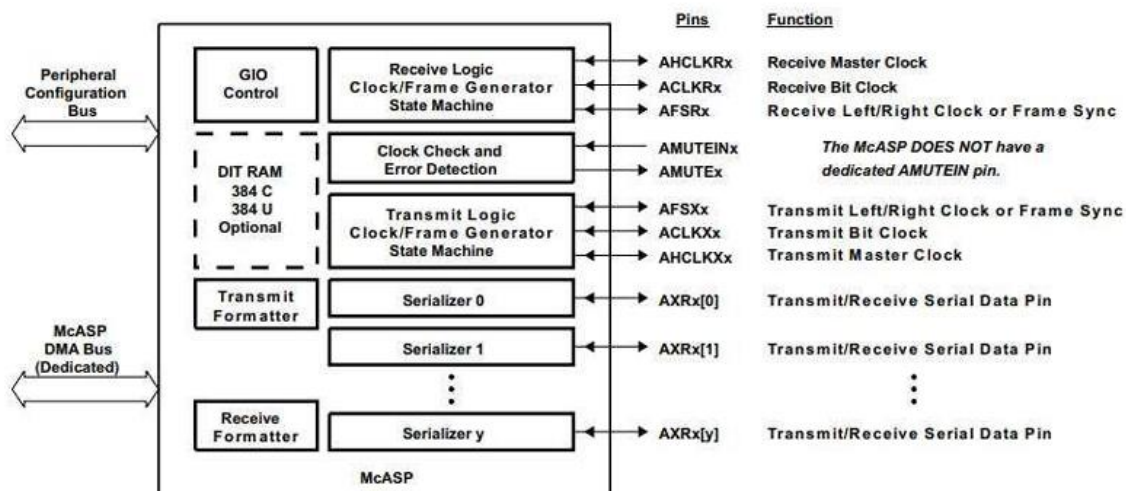
RESULT:

Familiarized the input and output ports of dsp board.

Experiment No: 11   Date: 24/10/24

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).
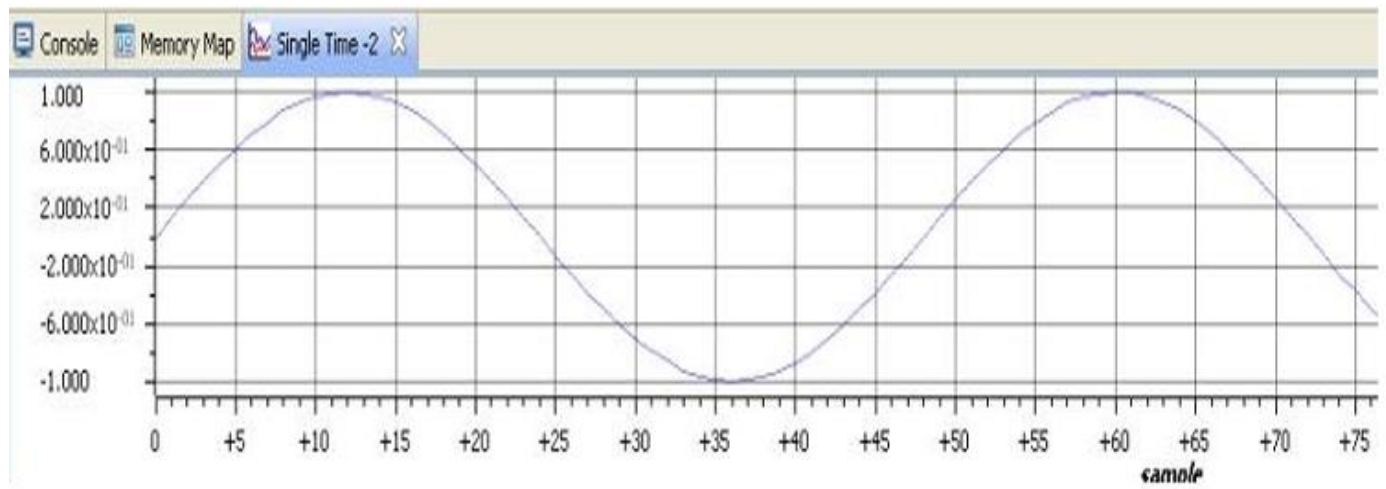
Change the Start address with the array name used in the program(here,s).

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

PROGRAM:

```
#include<stdio.h>

#include<math.h>
```

OBSERVATION:

```
#define pi 3.14159
float s[100];
void main()
{
int i;
float f=100, Fs=10000;
for(i=0;i<100;i++)
  s[i]=sin(2*pi*f*i/Fs);
}
```

RESULT:

Generated sine wave using DSP Kit.

Experiment No:12   Date: 24 /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

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:

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.

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
```

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

```
{
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.

.