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. BHADRA J R of **5th** Semester class (Roll No. **[B22ECB23] 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 : **[TKM22EC037]***

Staff Member in-charge *External Examiner*

Date:

Experiment No: 1 Date: 01/08/2024

SIMULATION OF BASIC TEST SIGNALS

Aim:

To generate continues 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 δ(t) =$\{\infty;\ \boldsymbol{t = 0} \& \boldsymbol{0};\ \boldsymbol{t \neq 0}\}$

2. **Unit Step Signal:**

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

3. **Ramp Signal:**

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

4. **Sine Signal:**

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

5. **Cosine Signal:**

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

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 $|t| \leq \tau/2$, 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 $|t| \leq \tau/2$, p(t) = 0 otherwise (assuming A is positive)

8. **Exponential Signal:**

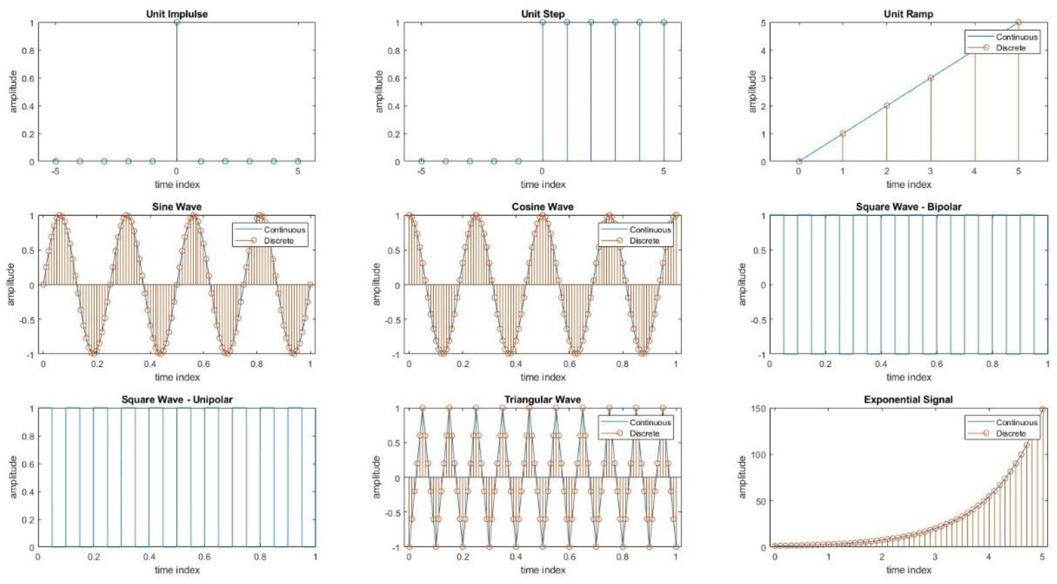
- A signal that increases or decreases exponentially with time. The rate of growth or decay is determined by the constant a.
- Mathematically: e^(at)

<u>Program</u>:

```
clc;

clear all;

close all;

t = -5:1:5;

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

subplot(3,3,1);

stem(t,y1);

xlabel('time index');

ylabel('amplitude');

title('Unit Implulse');

y2 = [zeros(1,5),ones(1,6)];

subplot(3,3,2);

stem(t,y2);

xlabel('time index');

ylabel('amplitude');

title('Unit Step');

t3 = 0:1:5;

y3 = [t3];
```

```
subplot(3,3,3);
plot(t3,y3);
hold on;
stem(t3,y3);
xlabel('time index');
ylabel('amplitude');
title('Unit Ramp');
legend("Continuous","Discrete");
t4 = 0:0.01:1;
f4 = 4;
subplot(3,3,4);
y4 = sin(2*pi*f4*t4);
plot(t4,y4);
hold on;
stem(t4,y4);
xlabel('time index');
ylabel('amplitude');
title('Sine Wave');
legend("Continuous","Discrete");
subplot(3,3,5);
y5 = cos(2*pi*f4*t4);
plot(t4,y5);
hold on;
stem(t4,y5);
xlabel('time index');
ylabel('amplitude');
title('Cosine Wave');
legend("Continuous","Discrete");
t6 = 0:0.0001:1;
f6 = 10;
subplot(3,3,6);
plot(t6,square(2*pi*f6*t6));
```

<u>Observation</u>:



```
xlabel('time index');
ylabel('amplitude');
title('Square Wave - Bipolar');
subplot(3,3,7);
plot(t6,sqrt(square(2*pi*f6*t6)));
xlabel('time index');
ylabel('amplitude');
title('Square Wave - Unipolar');
subplot(3,3,8);
t8 = 0:0.01:1;
f8 = 10;
y8 = sawtooth(2*pi*f8*t8,0.5);
plot(t8,y8);
hold on;
stem(t8,y8);
xlabel('time index');
ylabel('amplitude');
title('Triangular Wave');
legend("Continuous","Discrete");
subplot(3,3,9);
t9 = 0:0.1:5;
y9 = exp(t9);
plot(t9,y9);
hold on;
stem(t9,y9);
xlabel('time index');
ylabel('amplitude');
title('Exponential Signal');
legend("Continuous","Discrete");
```

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:

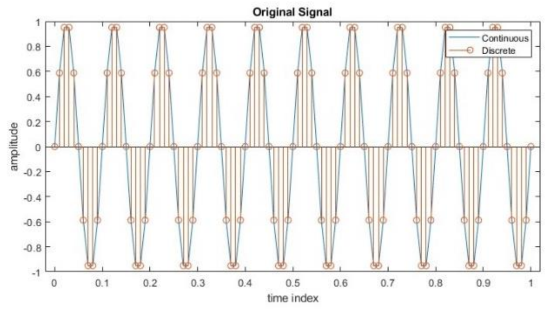
- Digital audio and video processing
- Communication systems
- Image processing
- Medical imaging

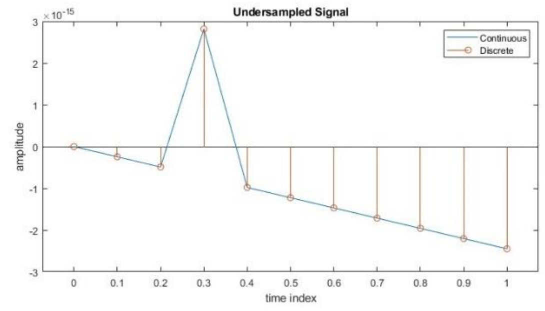
Program:

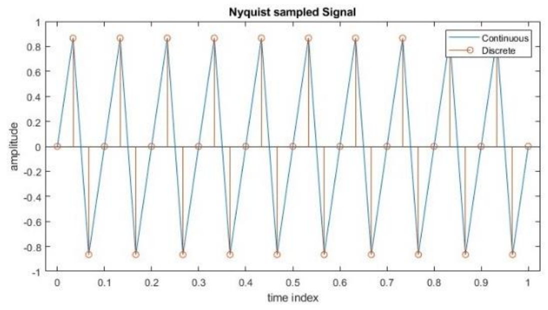
```
clc;

clear all;

close all;


%original signal

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);
```

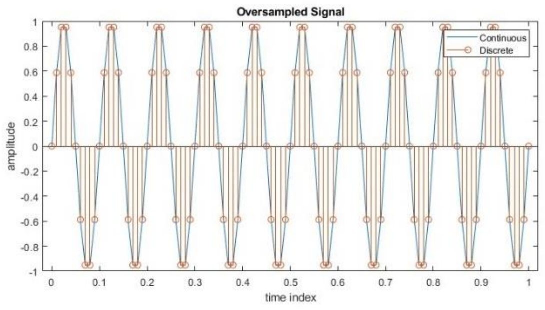
```
xlabel('time index');
ylabel('amplitude');
title('Original Signal');
legend("Continuous","Discrete");

%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 index');
ylabel('amplitude');
title('Undersampled Signal');
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 index');
ylabel('amplitude');
title('Nyquist sampled Signal');
legend("Continuous","Discrete");

%greater than Nyquist rate
```

<u>Observation</u>:









```
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 index');
ylabel('amplitude');
title('Oversampled Signal');
legend("Continuous","Discrete");
```

Result:

Verified sampling theorem using MATLAB.

Experiment No:3 Date: 08/08/2024

LINEAR CONVOLUTION

Aim:

To find the linear convolution with and without using built-in function.

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

b) x(n) = [1 2 1 2]
   h(n) = [3 2 1 2]

Theory:

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

Mathematical Definition:

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

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

Applications:

- Filtering**:** Convolution is used to filter signals, removing unwanted frequencies or noise.
- System Analysis**:** The impulse response of a system completely characterizes its behaviour, and convolution can be used to determine the output of the system given a known input.
- Image Processing: Convolution is used for tasks like edge detection, blurring, and sharpening images.

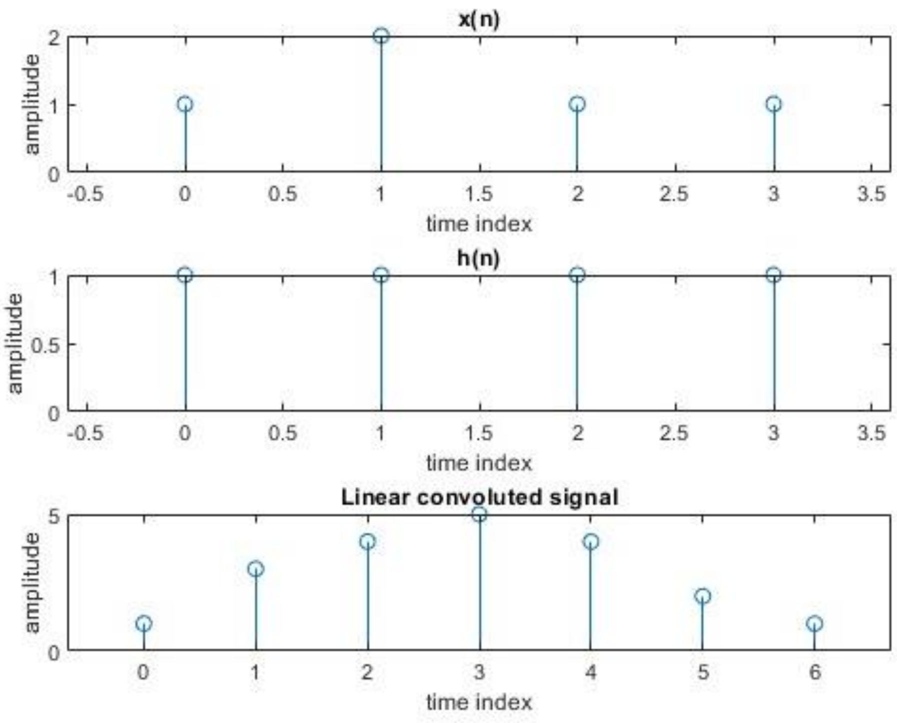
Program:

1.With built- in function:

```
clc;

clear all;

close all;


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);
```

Observation:

```
a) Enter input x(n): [1 2 1 1]
   Enter index of x(n): 0:3
   Enter input h(n): [1 1 1 1]
   Enter index of h(n): 0:3
        1      3     4     5     4     2     1
```

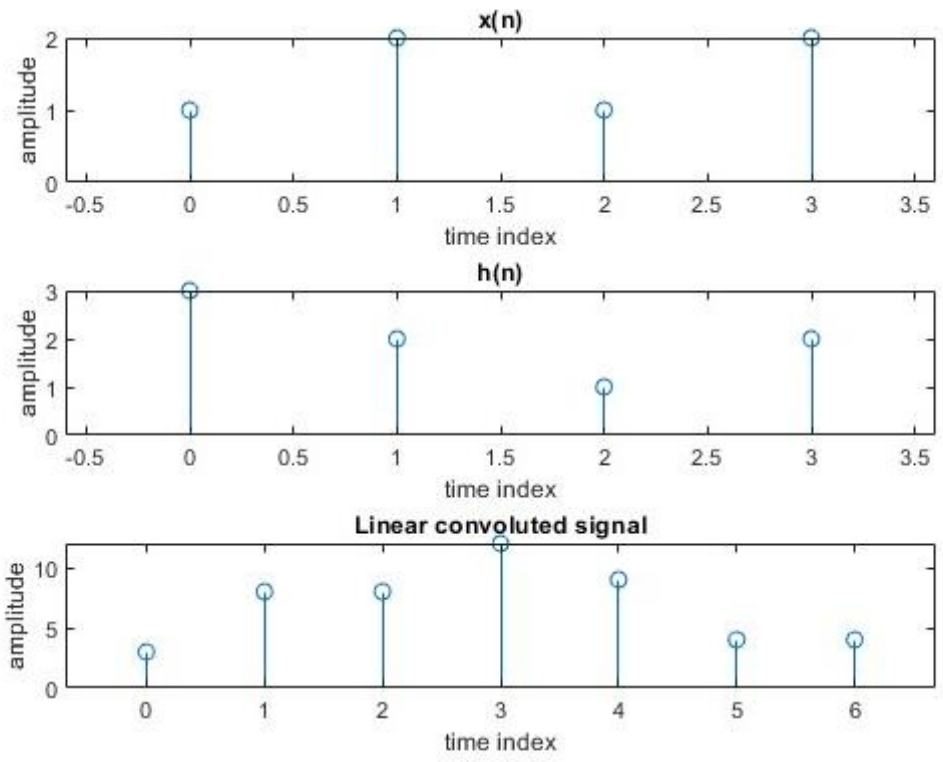


```
disp(y);

subplot(3,1,1);
stem(x_ind,x);
xlabel('time index');
ylabel('amplitude');
title('x(n)');

subplot(3,1,2);
stem(h_ind,h);
xlabel('time index');
ylabel('amplitude');
title('h(n)');

subplot(3,1,3);
stem(y_ind,y);
xlabel('time index');
ylabel('amplitude');
title('Linear convoluted signal');
```

2.Without 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_ind = min(x_ind)+min(h_ind) : max(x_ind)+max(h_ind);
x_len = length(x);
h_len = length(h);
```

```
b) Enter input x(n):[1 2 1 2]
   Enter index of x(n):0:3
   Enter input h(n):[3 2 1 2]
   Enter index of h(n):0:3
        3      8      8     12      9      4      4
```



```
y_len = x_len+h_len-1
y = zeros(1,y_len);

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(['The convolution result is:'])
disp(y);

subplot(3,1,1);
stem(x_ind,x);
xlabel('time index');
ylabel('amplitude');
title('x(n)');

subplot(3,1,2);
stem(h_ind,h);
xlabel('time index');
ylabel('amplitude');
title('h(n)');

subplot(3,1,3);
stem(y_ind,y);
xlabel('time index');
ylabel('amplitude');
title('Linear convoluted signal');
```

Result:

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

Experiment No: 4 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] ⊛ h[n]) = ∑{k=0} ^{N-1} x[k]h[(n-k) mod N]

Applications:

- Discrete-Time Filtering: Circular convolution is used for filtering discrete-time signals.
- Digital Signal Processing: It's a fundamental operation in many digital signal processing algorithms.

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

Program:

a). Using FFT and IFFT:

```
clc;

clear all;

close all;


x = input('Enter 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);
y_len = max(x_len,h_len);
newx = [x zeros(1,y_len-x_len)];
newh = [h zeros(1,y_len-h_len)];

x1 = fft(newx);
h1 = fft(newh);
hx = x1.*h1;
c = ifft(hx);
disp(['Using FFT and IFFT:']);
disp(c);
```

b). Using Concentric Circle Method:

```
clc;
clear all;
close all;
x = input('Enter input x(n): ');
h = input('Enter input h(n): ');
x = x(:,end:-1:1);
for i = 1:length(x)
    x = [x(end) x(1:end-1)];
    y(i) = sum(x.*h);
end
disp('Using Concentric Circle Method: ');
disp(y);
```

c). Using Matrix Method:

```
clc;
clear all;
close all;
xn = input('Enter input x(n): ');
```

<u>Observation</u>:

a) Using FFT and IFFT:

```
Enter input x(n): [1 2 3 4]

Enter index of x(n): 0:3

Enter input h(n):[1 2 1 0]

Enter index of h(n):0:3

Using FFT and IFFT:

    12     8     8    12
```

b) Using Concentric Circle Method:

```
Enter input x(n): [1 2 3 4]

Enter input h(n): [1 2 1 0]

Using Concentric Circle Method:

    12     8     8    12
```

c) Using Matrix Method:

```
Enter input x(n): [1 2 3 4]

Enter input h(n): [1 2 1 0]

Using Matrix method:

    12

     8

     8

    12
```

```
hn = input('Enter input h(n): ');
x = [];
xn = xn(:,end:-1:1);
for i = 1: length(xn)
    xn = [xn(end) xn(1:end-1)];
    x = [x;xn];
end

y = x*hn';
disp(['Using Matrix method:'])
disp(y);
```

<u>Result</u>:

Performed circular convolution using a) FFT and IFFT; b) Concentric Circle method; c) Matrix method and verified the results.

Experiment No: 5 Date: 29/09/2024

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;

clear all;

close all;

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

Observation:

a). Linear Convolution using Circular Convolution:

```
Enter input x(n): [1 2 3 4]

Enter input h(n): [1 1 1]

Linear convolution using Circular convolution:

     1     3     6     9     7     4
```

b). Circular convolution using Linear Convolution:

```
Enter input x(n): [1 2 3 4]

Enter input h(n): [1 1 1]

     8     7     6     9
```

```
hn = input('Enter input h(n): ');
x_len = length(xn);
h_len = length(hn);
y_len = x_len+h_len-1;
newx = [xn zeros(1,y_len-x_len)];
newh = [hn zeros(1,y_len-h_len)];
x = fft(newx);
h = fft(newh);
hx = x.*h;
y = ifft(hx);
disp('Linear convolution using Circular convolution: ');
disp(round(y));
```

b). Circular convolution using Linear Convolution:

```
clc;
clear;
close all;
x = input('Enter input x(n): ');
h = input('Enter input h(n): ');
c = conv(x,h);
x_len = length(x);
h_len = length(h);
y_len = max(x_len,h_len);
r1 = [c(1:y_len)];
r  = [c(y_len+1:end)];
r2 = [r zeros(1,length(r1)-length(r))];
y = r1 + r2;
disp(y);
```

Result:

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

Experiment No: 6 Date: 29/09/2024

DISCRETE FOURIER TRANSFORM AND INVERSE DISCRETE FOURIER TRANSFORM

Aim:

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

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

Theory:

1. Discrete Fourier Transform:

The Discrete Fourier Transform (DFT) is a mathematical technique used to transform a discrete signal from time domain into the frequency domain.

The DFT of a sequence x(n) of length N is given by:

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

where:

- X[k] is the DFT of the sequence x[n].
- $e^{-j\frac{2\pi}{N}kn}$ represents the complex exponential basis function

The DFT is extensively used in digital signal processing applications, such as filtering, spectral analysis, and data compression.

2. Inverse Discrete Fourier Transform:

The Inverse Discrete Fourier Transform (IDFT) is the reverse process of the DFT, transforming the frequency-domain representation of a signal back into the time-domain signal. The IDFT is used to reconstruct the original signal from its frequency components.

The IDFT of a sequence X(k) of length N is given by:

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

where:

- x[n] is the DFT of the sequence X[k].
- $e^{j\frac{2\pi}{N}kn}$ is the inverse of the DFT's complex exponential basis function

The IDFT is extensively used in digital signal processing applications for reconstructing signals, synthesizing waveforms, and recovering time-domain data from frequency-domain representations.

Program:

a). Discrete Fourier Transform:

```
clc;
clear all;
close all;
xn = input('Enter the input x(n): ');
N = input('Enter the no. of points: ');
L = length(xn);
if(N<L)
    error('N should be greater than or equal to L: ');
end
x = [xn zeros(1,N-L)];
X = zeros(N,1);
for k = 0:N-1
    for n = 0:N-1
        X(k+1) = X(k+1) + x(n+1)*exp(-i*2*pi*k*n/N);
    end
end
disp('X');
disp(X);
disp('round(X)');
disp(round(X));
disp('fft(x)');
disp(fft(x));
%magnitude spectrum
subplot(2,1,1);
k = 0:N-1;
magx = abs(X);
stem(k,magx);
```

```
hold on;
plot(k,magx);
xlabel('frequency');
ylabel('magnitude');
title('DFT Magnitude plot')

%phase spectrum
subplot(2,1,2);
phasex = angle(X);
stem(k,phasex);
hold on;
plot(k,phasex);
xlabel('frequency');
ylabel('phase');
title('DFT Phase plot')
```

b) Inverse Discrete Fourier Transform:

```
clc;
clear all;
close all;
Xn = input('Enter the input x(n): ');
N = input('Enter the no. of points: ');
L = length(Xn);
if(N<L)
    error('N should be greater than or equal to L: ');
end
X = [Xn zeros(1,N-L)];
x = zeros(N,1);
for n = 0:N-1
    for k = 0:N-1
        x(n+1) = x(n+1) + X(k+1)*exp(i*2*pi*k*n/N);
    end
```

```
end
x = x/N;
disp('x');
disp(x);
disp('round(x)');
disp(round(x));
disp('ifft(X)');
disp(ifft(X));
```

<u>Result</u>:

Performed

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

2. IDFT using inbuilt function and without using inbuilt function and verified the result.

Experiment No: 7 Date: 29/09/2024

PROPERTIES OF DFT

Aim:

To prove the properties of dft:

a) Linearity Property b) Multiplication Property c) Circular Convolution Property

d) Parseval's Theorem

Theory:

1. Linearity Property:

The linearity property states that the DFT of a linear combination of two signals is the same linear combination of their DFTs.

$$\mathrm{DFT}\{a.x[n] + b.y[n]\} = a.X[k] + b.Y[k]$$

2. Multiplication Property:

The multiplication property states that the pointwise multiplication of two sequences in the time domain corresponds to circular convolution in the frequency domain.

$$\mathrm{DFT}\{x[n].y[n]\} = \frac{1}{N} DFT\{x[n]\} \circledast DFT\{y[n]\}$$

3. Circular Convolution Property:

The convolution theorem states that the circular convolution of two sequences in the time domain is the pointwise multiplication of their DFTs in the frequency domain.

$$\mathrm{DFT}\{x[n] \circledast y[n]\} = X[k].Y[k]$$

4. Parseval's Theorem:

The 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.

$$\sum_{n=0}^{N-1} x[n].y*[n] = \frac{1}{N}\sum_{k=0}^{N-1} X(k).Y*(k)$$

These properties make the DFT a powerful tool for analyzing discrete signals in the frequency domain.

Program:

a). Linearity Property:

```
clc;
clear all;
close all;

x = input('Enter the input x(n): ');
y = input('Enter the input y(n): ');
lx = length(x);
ly = length(y);
l = max(lx,ly);
x = [x zeros(1,l-lx)];
y = [y zeros(1,l-ly)];

X = fft(x);
Y = fft(y);

a = input('Enter the value of a: ');
b = input('Enter the value of b: ');

sum = a.*x + b.*y;
lhs = fft(sum);
rhs = a.*X + b.*Y;

disp("LHS");
disp(lhs);
disp("RHS");
disp(rhs);
```

```
if lhs == rhs
    disp('Linearity property holds');
else
    disp('Linearity property does not hold');
end
```

b). Multiplication Property:

```
clc;
clear all;
close all;

x = input('Enter the input x(n): ');
y = input('Enter the input y(n): ');
lx = length(x);
ly = length(y);
N = max(lx,ly);
x = [x zeros(1,N-lx)];
y = [y zeros(1,N-ly)];

X = fft(x);
Y = fft(y);

p = x.*y;
lhs = fft(p);
c = cconv(X,Y,N);
rhs = c/N;

disp("LHS");
disp(lhs);
disp("RHS");
disp(rhs);
```

```
if lhs == rhs
    disp('Multiplication property holds');
else
    disp('Multiplication property does not hold');
end
```

c). Circular Convolution Property:

```
clc;
clear all;
close all;

x = input('Enter the input x(n): ');
y = input('Enter the input y(n): ');
lx = length(x);
ly = length(y);
N = max(lx,ly);
x = [x zeros(1,N-lx)];
y = [y zeros(1,N-ly)];

X = fft(x);
Y = fft(y);

p = cconv(x,y,N);
lhs = fft(p);
rhs = X.*Y;

disp("LHS");
disp(lhs);
disp("RHS");
disp(rhs);

if lhs == rhs
```

```
    disp('Circular convolution property holds');
else
    disp('Circular convolution property does not hold');
end
```

d). Parseval’s Theorem:

```
clc;
clear all;
close all;

x = input('Enter the input x(n): ');
y = input('Enter the input y(n): ');
lx = length(x);
ly = length(y);
N = max(lx,ly);
x = [x zeros(1,N-lx)];
y = [y zeros(1,N-ly)];

X = fft(x);
Y = fft(y);

lhs = sum(x.*conj(y));
c = sum(X.*conj[(Y));
rhs = c/N;

disp("LHS");
disp(lhs);
disp("RHS");
disp(rhs);

if lhs == rhs
    disp("Parseval's theorem verified");
```

```
else
    disp("Parseval's theorem is not verified");
end
```

<u>Result</u>:

Verified the properties of DFT.

Experiment No: 8 Date: 29/09/2024

OVERLAP SAVE AND OVERLAP ADD METHOD

Aim:

Implement overlap add and overlap save method using Matlab.

Theory:

The Overlap-Save and Overlap-Add methods are efficient techniques in digital signal processing for performing convolution on long signals using the Fast Fourier Transform (FFT). These methods break the input signal into smaller segments and process them separately to handle large data more efficiently. These methods are widely used in practical DSP applications to reduce the computational complexity of filtering operations.

Overlap – Save Method:

The Overlap-Save method is used to handle long convolution operations by breaking the input signal into overlapping segments, applying FFT-based convolution to each segment, and discarding the overlapping parts to get the final result.

1. Divide the input signal into overlapping segments of size N, where $N=L+M-1$ (L is the length of each segment of the input signal, and M is the length of the filter impulse response h(n)). The overlap size is $M-1$, meaning that each segment overlaps with the previous one by $M-1$ samples.

2. Perform FFT on each segment and the filter impulse response (zero-padded).

3. Multiply the FFTs and apply Inverse FFT to get the time-domain result.

4. Discard the first $M-1$ samples (overlap) from each segment.

5. Concatenate the valid portions of all segments to form the output.

Overlap – Add Method:

The Overlap-Add method is another technique to efficiently perform convolution by dividing the input signal into non-overlapping segments, applying FFT-based convolution, and overlapping and adding the results to produce the final output.

1. Divide the input signal into non-overlapping segments of size L.

2. Zero-pad each segment to size $N=L+M-1$.

3. Perform FFT on each segment and the filter impulse response.

4. Multiply the FFTs and apply Inverse FFT (IFFT) to get the time-domain result.

5. The overlapping parts from consecutive segments are added together to form the complete filtered output.

<u>Program</u>:

a) Overlap – Save Method:

```
clc;

clear all;

close all;


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

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

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


y = ovrlsav(x, h, N);

disp("Using Overlap and Save method");

disp(y);

disp("Verification");

disp(cconv(x,h,length(x)+length(h)-1));


function y = ovrlsav(x, h, N)

 if (N < length(h))

 error("N must be greater than the length of h");

 end

 Nx = length(x);

 M = length(h);

 M1 = M - 1;

 L = N - M1;

 x = [zeros(1, M1), x, zeros(1, N-1)];

 h = [h, zeros(1, N - M)];
 K = floor((Nx + M1 - 1) / L);

 Y = zeros(K + 1, N);


 for k = 0:K

 xk = x(k*L + 1 : k*L + N);

 Y(k+1, :) = cconv(xk, h, N);

 end

 Y = Y(:, M:N)';

 y = (Y(:))';

end
```

b) Overlap – Add Method:

```
clc;

clear all;

close all;


x = input('Enter the input sequence x : ');

h = input('Enter the impulse response h : ');

L = length(h);

N = length(x);

M = length(h);

x_padded = [x, zeros(1, L - 1)];

y = zeros(1, N + M +1);

num_sections = (N + L - 1) / L;


for n = 0:num_sections-1
```

```
 start_idx = n * L + 1;

 end_idx = start_idx + L - 1;

 x_section = x_padded(start_idx:min(end_idx, end));

 conv_result = conv(x_section, h);

 y(start_idx:start_idx + length(conv_result) - 1) = y(start_idx:start_idx +
length(conv_result) - 1) + conv_result;

end



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

y_builtin = conv(x, h);

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

disp(y);

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

disp(y_builtin);



figure;

subplot(2, 1, 1);

stem(y, 'filled');

title('Overlap-add Convolution Result');

grid on;

subplot(2, 1, 2);

stem(y_builtin, 'filled');

title('Built-in Convolution Result');

grid on;
```

Result:

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

Experiment No: 9 Date:17/10/24

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)))

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 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 = 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.Band Pass 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');
```

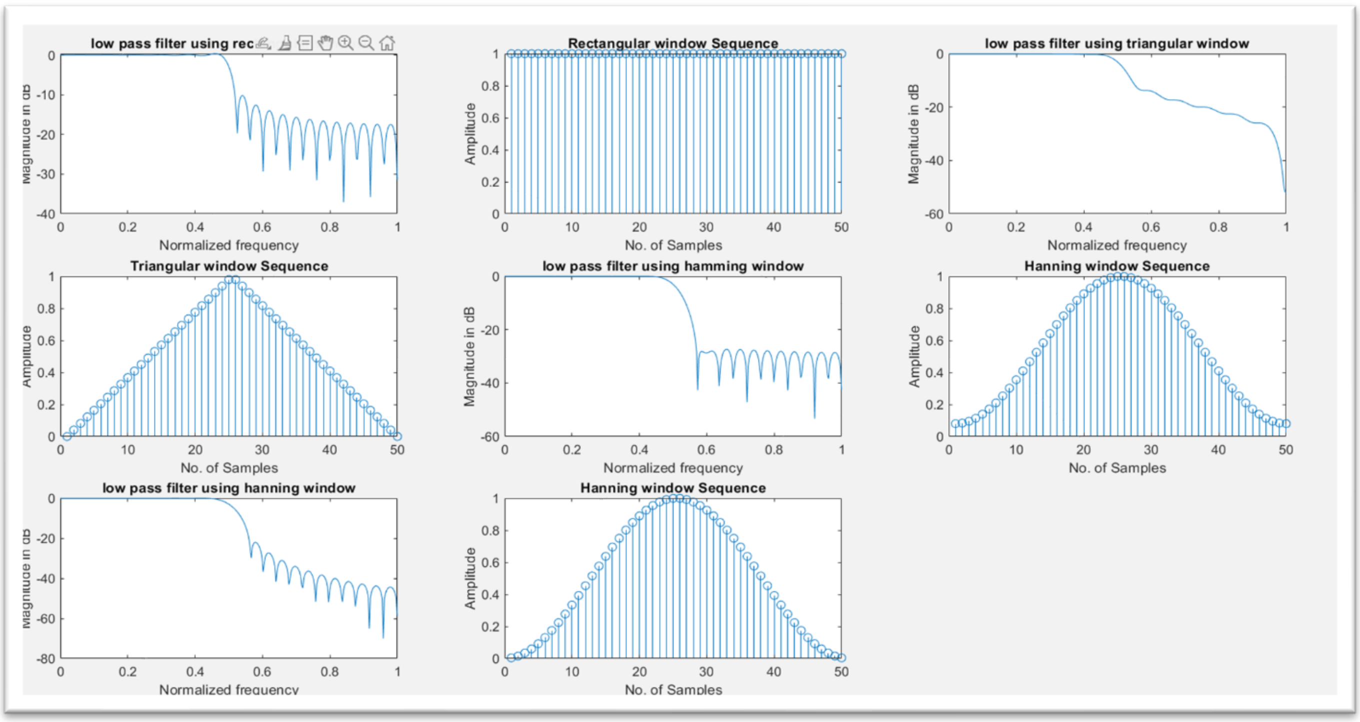
```
subplot(3,3,5);
plot(w/pi,10*log10(abs(h3)));
title('band pass filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,6);
stem(wh);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,7);
plot(w/pi,10*log10(abs(h4)));
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.Band Stop Filter:

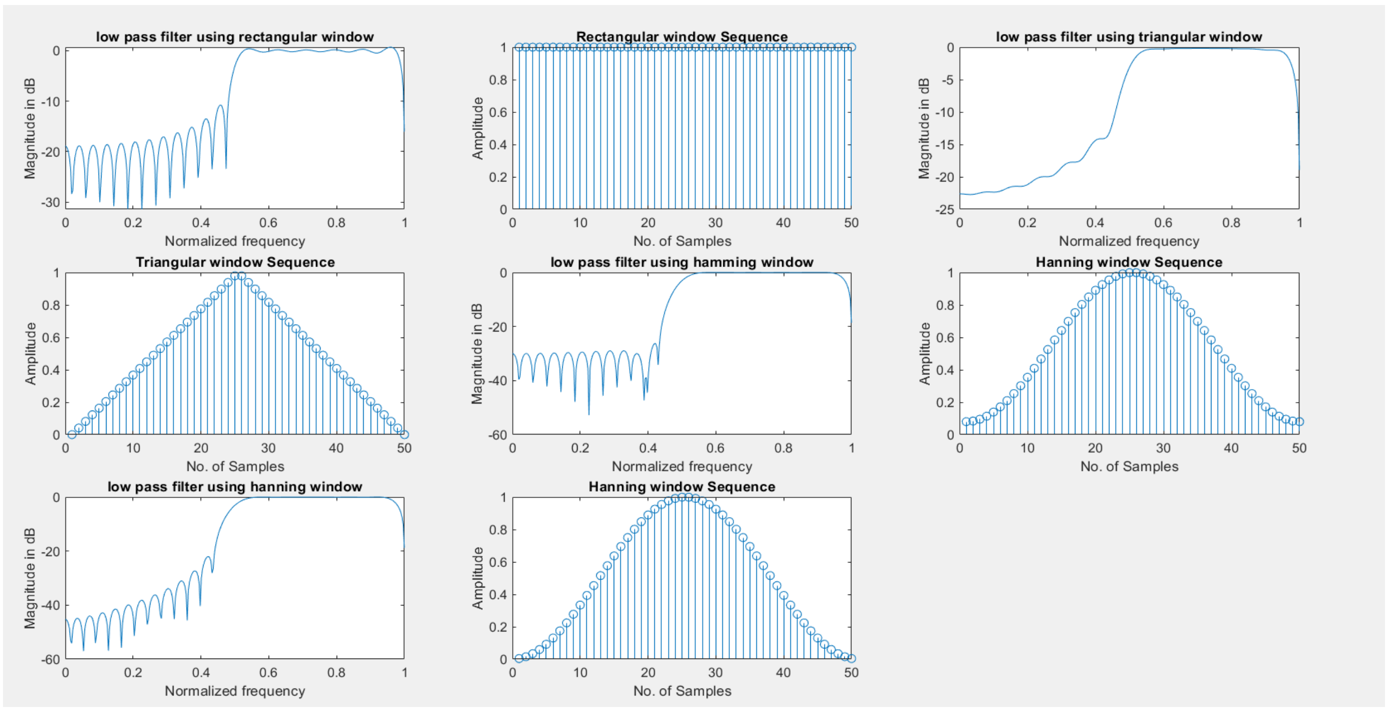
```
 clc;
clear all;
close all;
wc1=0.5*pi;
wc2=0.9*pi;
N = 50;
alpha = (N-1)/2;
eps = 0.001;
n = 0:1:N-1;
```

Observation:

1. Low Pass Filter:

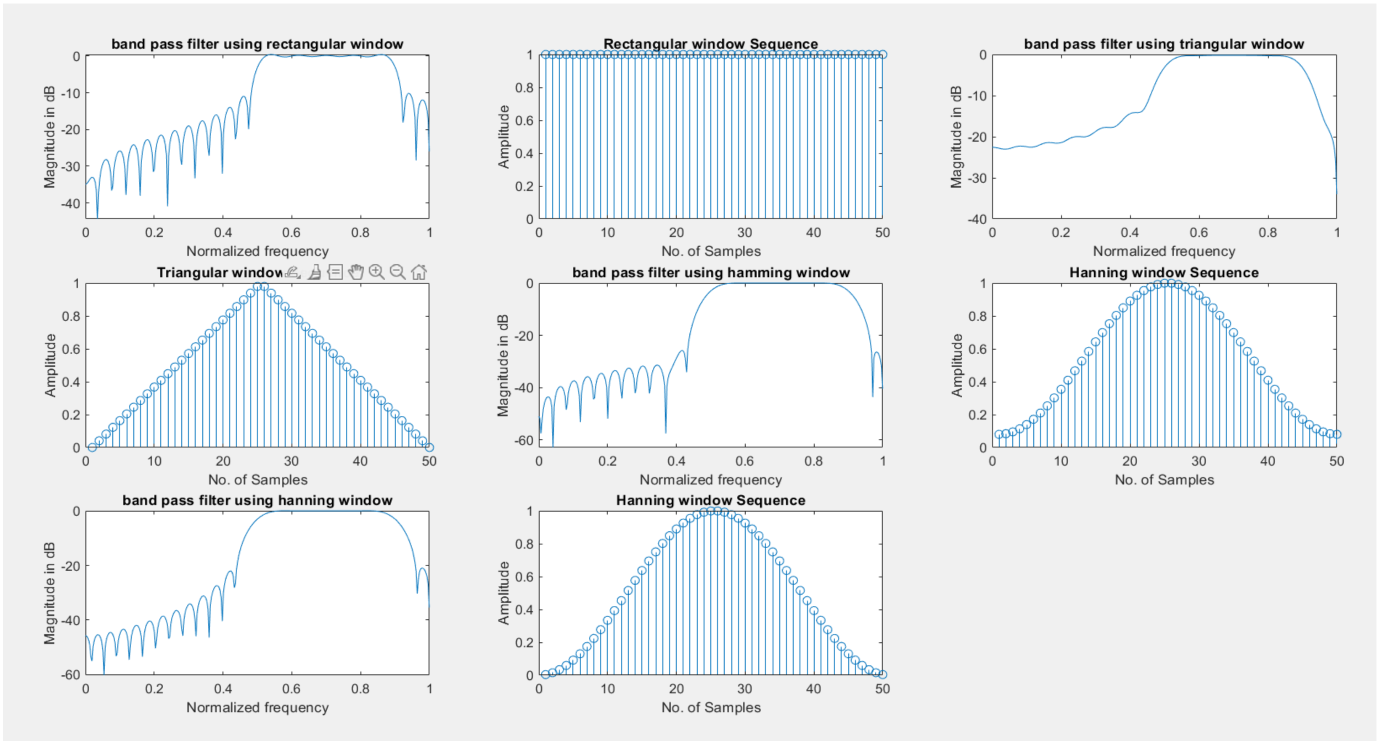


2. High Pass Filter:

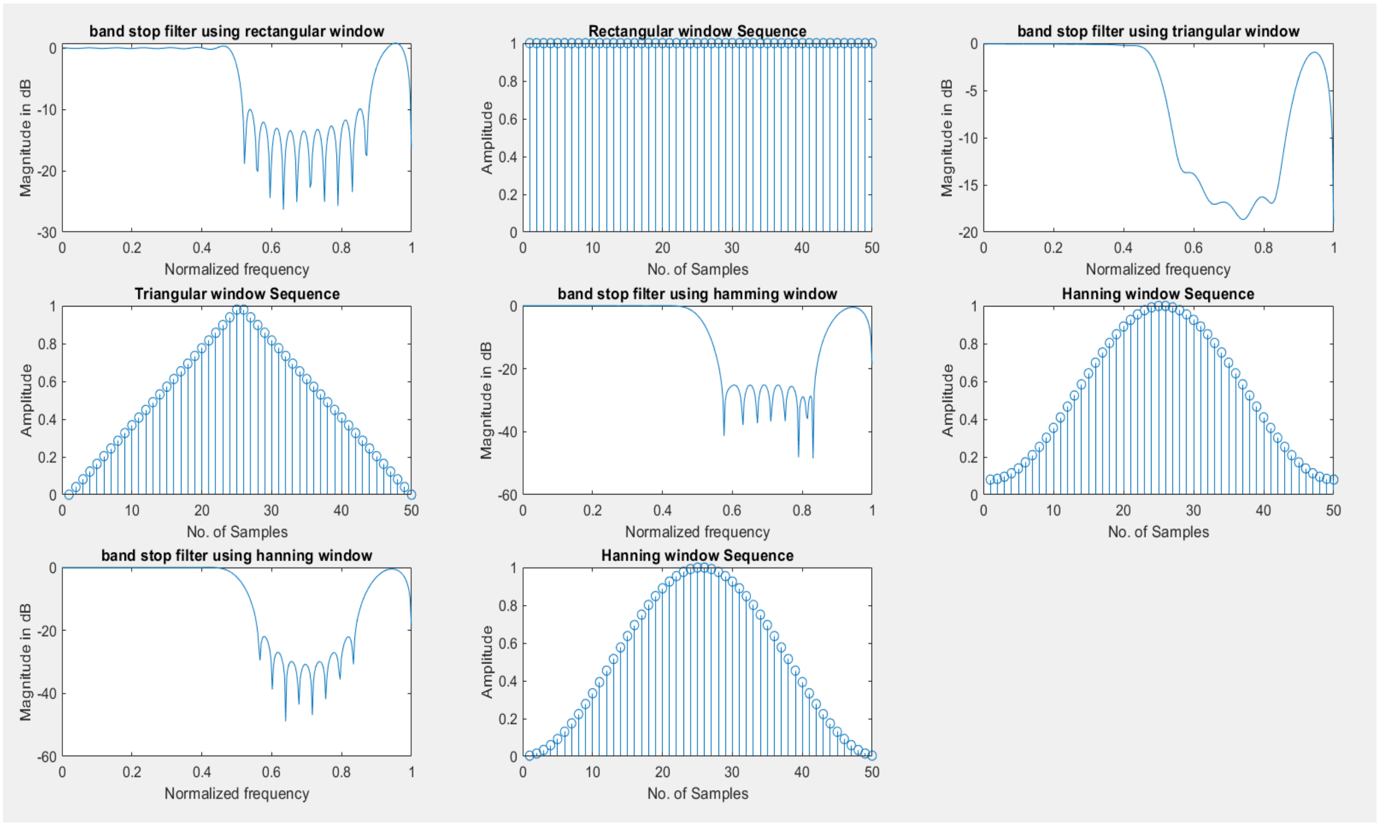


```
hd = (sin(wc1*(n-alpha+eps))-sin(wc2*(n-alpha+eps))+sin(pi*(n-
alpha)))./(pi*(n-alpha+eps));

wr = boxcar(N);

wt=bartlett(N);

wh=hamming(N);

whn=hanning(N);

hn1 = hd.*wr';

hn2 = hd.*wt';

hn3 = hd.*wh';

hn4 = hd.*whn';

w = 0:0.01:pi;

h1 = freqz(hn1,1,w);

h2 = freqz(hn2,1,w);

h3 = freqz(hn3,1,w);

h4 = freqz(hn4,1,w);

subplot(3,3,1);

plot(w/pi,10*log10(abs(h1)));

title('band stop filter using rectangular window');

xlabel('Normalized frequency');

ylabel('Magnitude in dB');

subplot(3,3,2);

stem(wr);

title('Rectangular window Sequence');

xlabel('No. of Samples');

ylabel('Amplitude');

subplot(3,3,3);

plot(w/pi,10*log10(abs(h2)));

title('band stop filter using triangular window');

xlabel('Normalized frequency');

ylabel('Magnitude in dB');

subplot(3,3,4);

stem(wt);
```

3. Band Pass Filter:



4. Band Stop Filter:



```
title('Triangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,5);
plot(w/pi,10*log10(abs(h3)));
title('band stop filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,6);
stem(wh);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(3,3,7);
plot(w/pi,10*log10(abs(h4)));
title('band stop filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(3,3,8);
stem(whn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

Result:

Implemented various FIR filters using different windows

1Low Pass Filter

2.High Pass Filter

3.Band pass Filter

4.Band stop Filter

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

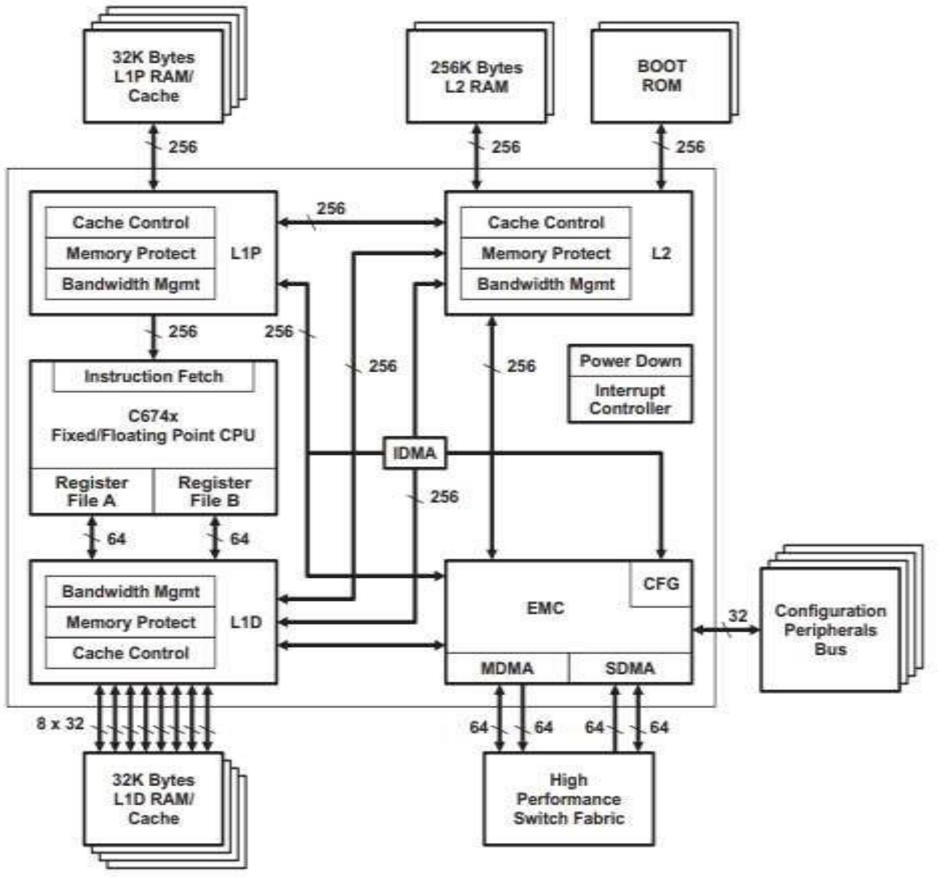


FIGURE: TMS320C 674X DSP CPU BLOCK DIAGRAM

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 usedfor data or can be data address pointers. The data types supported include packed 8-bit data, packed 16-bit data, 32-bit data, 40- bit data, and 64-bit data. Values larger than 32 bits, such as

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

Multichannel Audio Serial Port (McASP):

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

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

• Up to sixteen transmit or receive data pins and serializers

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

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

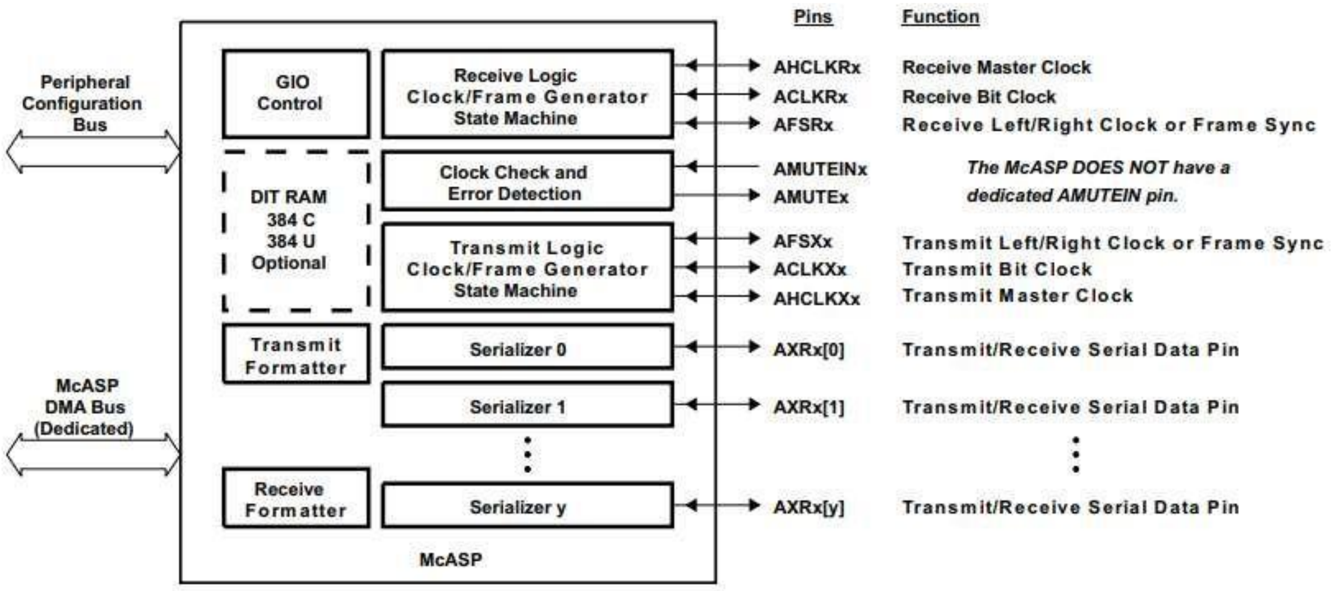
• Extensive error checking and mute generation logic

• All unused pins GPIO-capable

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

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

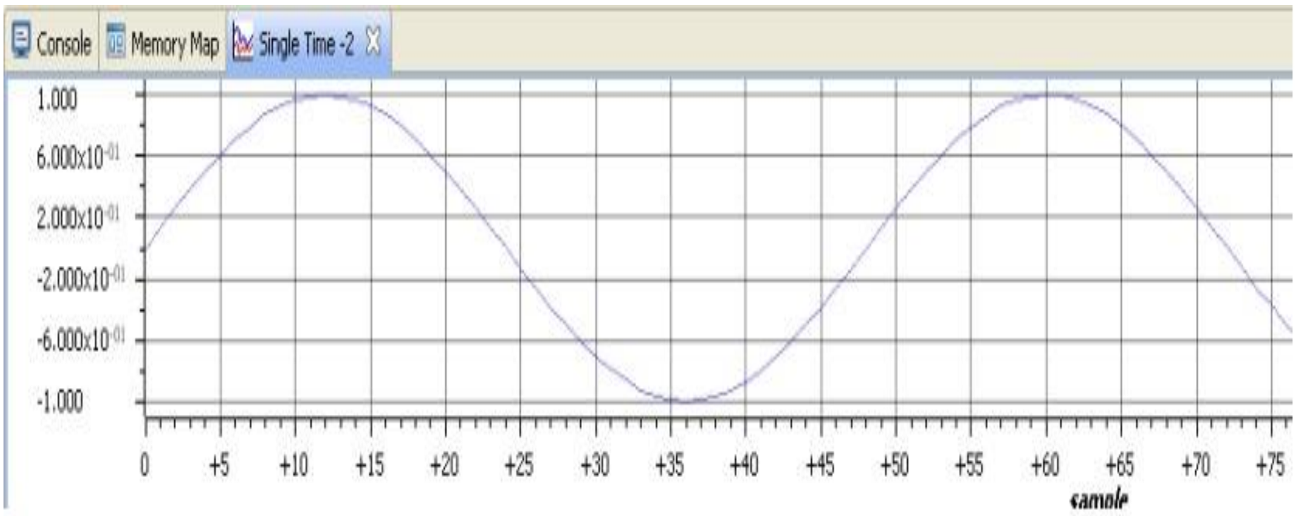
The ADC circuitry on the codec converts the input analog signal to a digital representation to be processed by the DSP. The maximum level of the input signal to be converted is determinedby the specific ADC circuitry on the codec, which is 6 V p-p with the onboard codec. After thecaptured signal is processed, the result needs.
to be sent to the outside world. DAC, which performs the reverse operation of the ADC. An output filter smooths out or reconstructs the output signal. ADC, DAC, and all required filtering functions are performed by the single-chip codec AIC23 on board the DSK.


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

Result:

Familiarized the input and output ports of dsp board.

Observation:



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.

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

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++)
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 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.
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.

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

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.

.