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. BIBIN BABY of **5th** Semester class (Roll No. **B22ECB24 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 : **TKM22EC038***

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 for the following :

1. unit impulse signal
2. unit step signal
3. ramp signal
4. sine signal
5. cosine wave
6. bipolar pulse signal
7. unipolar pulse signal
8. triangular signal
9. exponential signal

Theory

A digital signal can be either a deterministic signal that can be predicted with certainity, or a random signal that is unpredictable. Due to ease in signal generation and need for predictability, deterministic signal can be used for system simulation studies. A continuous time signal is defined for all values of time t.

1. Unit impulse signal:
   The simplest signal is the unit impulse signal which is defined as,
   $\delta(t) = \infty; \boldsymbol{t=0}$
   $\quad \boldsymbol{=0}; \boldsymbol{t \neq 0}$

2. Unit step signal:
   A signal that is zero for all negative time values and one for positive time values.It is defined as,
   $u(t) = 1$ for $t \geq 0$
   $\quad = 0$ for $t < 0$

3. Ramp signal:
   A signal that increases linearly with time. This signal is given by,
   $r(n) = n$ for $n \geq 0$
   $\quad = 0$ for $n < 0$

4. Sine signal :
   A continuous periodic signal. It oscillates smoothly between -1 and 1.It is defined as,
   $y(t)=A\sin(2\pi ft)$

5. Cosine wave :
   A continuous periodic signal like the sine wave but phase-shifted by π\2.It is defined as,
   $y(t)=A\cos(2\pi ft)$

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

7. Unipolar pulse signal:
   A pulse signal that alternates between zero and a positive value. It remains at zero for a specified duration and then jumps to a positive constant level (e.g., 0 and 1). It is given by,
   $p(t) = A$ for $|t| \leq \tau/2$,
   $= 0$ 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. It is given by,
   $\Lambda(t) = 1 - |t|$ for $|t| \leq 1$,
   $= 0$ 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 . It's general form is,
   $x(n) = a^n$ for all n.

Program

```
clc;

clear all;

close all;

%unit impulse

t=-5:1:5;

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

subplot(3,3,1);

stem(t,y1);
```

```
title('unit impulse');
xlabel('time index');
ylabel('amplitude');
%unit step
y2=[zeros(1,5),ones(1,6)];
subplot(3,3,2);
stem(t,y2);
title('unit step');
xlabel('time index');
ylabel('amplitude');
%ramp
t3=0:1:10;
y3=[t3];
subplot(3,3,3);
stem(t3,y3);
hold on;
plot(t3,y3);
title('ramp');
xlabel('time index');
ylabel('amplitude');
legend("discrete","continuous");
%sine wave
t4=0:0.01:1;
f4=4;
subplot(3,3,4);
stem(t4,sin(2*pi*f4*t4));
hold on;
plot(t4,sin(2*pi*f4*t4));
title('sine wave');
xlabel('time index');
```

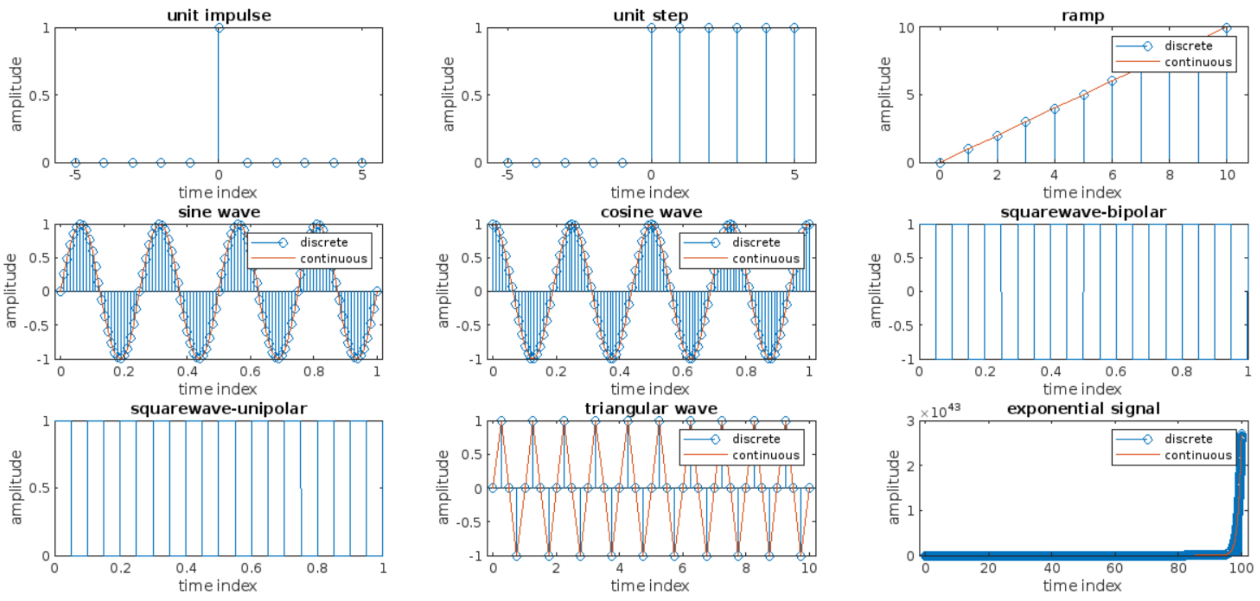
```
ylabel('amplitude');
legend("discrete","continuous");
%cosine wave
subplot(3,3,5);
stem(t4,cos(2*pi*f4*t4));
hold on;
plot(t4,cos(2*pi*f4*t4));
title('cosine wave');
xlabel('time index');
ylabel('amplitude');
legend("discrete","continuous");
%squarewave-bipolar
t6=0:0.0001:1;
f6=10;
subplot(3,3,6);
plot(t6,square(2*pi*f6*t6));
title('squarewave-bipolar');
xlabel('time index');
ylabel('amplitude');
%squarewave-unipolar
subplot(3,3,7);
plot(t6,sqrt(square(2*pi*f6*t6)));
title('squarewave-unipolar');
xlabel('time index');
ylabel('amplitude');
%triangular wave
t8=0:0.25:10;
f8=5;
subplot(3,3,8);
stem(t8,sin(2*pi*f8*t8));
```

```
hold on;
plot(t8,sin(2*pi*f8*t8));
title('triangular wave');
xlabel('time index');
ylabel('amplitude');
legend("discrete","continuous");
%exponential signal
t9=0:0.01:100;
y9=exp(t9);
subplot(3,3,9);
stem(t9,y9);
hold on;
plot(t9,y9);
title('exponential signal');
xlabel('time index');
ylabel('amplitude');
legend("discrete","continuous");
```

Result

Generated and verified various waveforms of basic test signal.

Observation



Experiment No: 02 Date:08-08-2024

Verification of Sampling Theorem

Aim

To verify Sampling Theorem.

Theory

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

fs ≥ 2·fmax

Where:

• fs is the sampling frequency (rate at which the signal is sampled),

• fmax is the highest frequency present in the signal.

Applications:

• Digital audio and video processing

• Communication systems

• Image processing

• Medical imaging

Program

```
clc;
clear all;
close all;
%original signal
t=0:0.01:1;
fm=10;
```

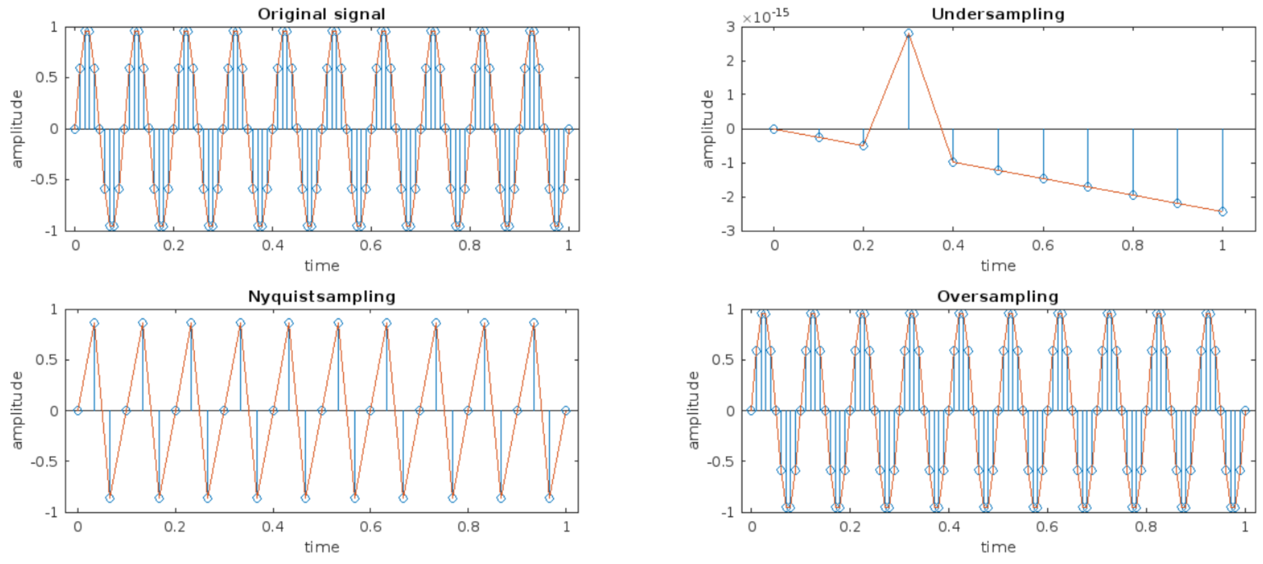
```
y=sin(2*pi*fm*t);
subplot(2,2,1);
stem(t,y);
hold on;
plot(t,y);
title("Original signal");
xlabel("time");
ylabel("amplitude");
%less than nyquist range
fs1=fm;
t1=0:1/fs1:1;
y1=sin(2*pi*fm*t1);
subplot(2,2,2);
stem(t1,y1);
hold on;
plot(t1,y1);
title("Undersampling");
xlabel("time");
ylabel("amplitude");
%equal to nyquist rate
fs2=3*fm;
t2=0:1/fs2:1;
y2=sin(2*pi*fm*t2);
subplot(2,2,3);
stem(t2,y2);
hold on;
plot(t2,y2);
title("Nyquistsampling");
xlabel("time");
ylabel("amplitude");
```

```
%greater than nyquist rate
fs3=10*fm;
t3=0:1/fs3:1;
y3=sin(2*pi*fm*t3);
subplot(2,2,4);
stem(t3,y3);
hold on;
plot(t3,y3);
title("Oversampling");
xlabel("time");
ylabel("amplitude");
```

Result

Verified Sampling Theorem using MATLAB.

Observation



Experiment No: 03 Date: 08-08-2024

Linear Convolution

Aim

To find linear convolution of two input sequences ,

a) with built in function.
b) Without 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:

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

Applications:

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

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

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

Program

a)with built in function

```
clc;

clear all;

close all;

x=input("Enter the elements in x[n]:");

x_ind=input("Enter the index of x[n]:");

h=input("Enter the elements in h[n]:");

h_ind=input("Enter the index of h[n]:");

y=conv(x,h);
```

```
y_ind=min(x_ind)+min(h_ind):max(x_ind)+max(h_ind);
disp('Linear convolution result:');
disp(y);
stem(y_ind,y);
title("Linear convolution");
xlabel("time index");
ylabel("amplitude");
```

b)without built in function

```
clc;
clear all;
close all;
x=input("Enter the elements in x[n]:");
x_ind=input("Enter the index of x[n]:");
h=input("Enter the elements in h[n]:");
h_ind=input("Enter the index of h[n]:");
n1 = length(x);
n2 = length(h);
n = n1 + n2 - 1;
y = zeros(1, n);
for i = 0:n-1
    for j = 0:n1-1
        if (i - j >= 0 && i - j < n2)
            y(i+1) = y(i+1) + x(j+1) * h(i - j + 1);
        end
    end
end
disp('Linear convolution result:');
disp(y);
y_ind=min(x_ind)+min(h_ind):max(x_ind)+max(h_ind);
```

```
stem(y_ind,y);
title("Linear convolution");
xlabel("time index");
ylabel("amplitude");
```

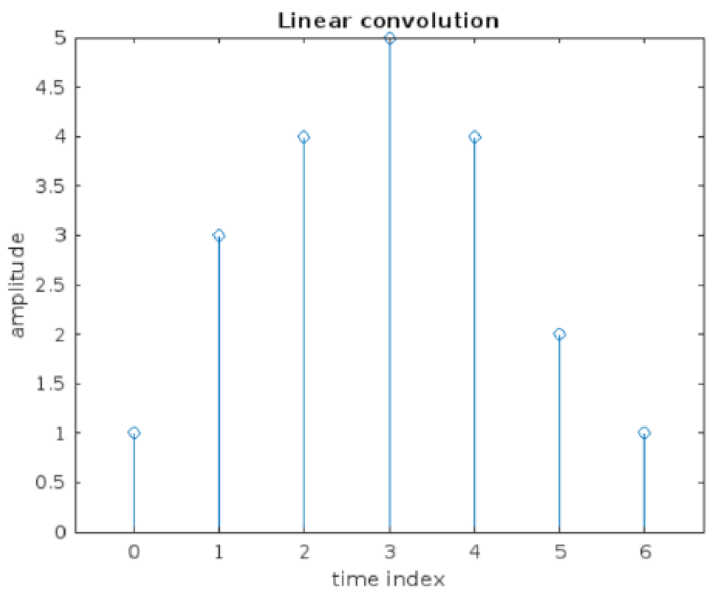
Result

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

Observation

a)with built in function

```
INPUT:
Enter the elements in x[n]:
[1 2 1 1]
Enter the index of x[n]:
0:3
Enter the elements in h[n]:
[1 1 1 1]
Enter the index of h[n]:
0:3
OUTPUT:
Linear convolution result:
   1    3    4    5    4    2    1
```



b)without built in function

INPUT:

Enter the elements in x[n]:

[1 2 1 1]

Enter the index of x[n]:

0:3

Enter the elements in h[n]:

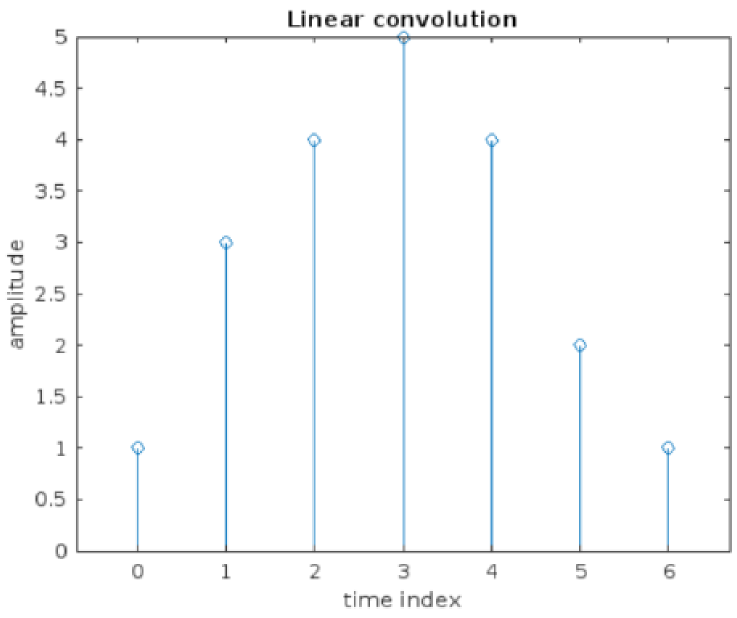
[1 1 1 1]

Enter the index of h[n]:

0:3

OUTPUT:

```
Linear convolution result:

    1    3    4    5    4    2    1
```



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] ⊛ 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 the elements in x[n]:");
x_ind=input("Enter the index of x[n]:");
h=input("Enter the elements in h[n]:");
```

```
h_ind=input("Enter the index of h[n]:");
figure;
subplot(3,1,1);
stem(x_ind,x);
title("x[n]");
xlabel("time ");
ylabel("amplitude");
grid;
subplot(3,1,2);
stem(h_ind,h);
title("h[n]");
xlabel("time");
ylabel("amplitude");
grid;
len_x=length(x);
len_h=length(h);
N=max(len_x,len_h);
new_x=[x zeros(1,N-len_x)];
new_h=[h zeros(1,N-len_h)];
x1=fft(new_x);
h1=fft(new_h);
y1=x1.*h1;
y=ifft(y1);
ny=0:N-1;
disp(y);
subplot(3,1,3);
stem(ny,y);
title("Circular convolution output y[n]");
xlabel("time ");
ylabel("amplitude");
```

```
grid;
```

b)Using concentric circle method

```
clc;
clear all;
close all;
x=input("Enter the elements in x[n]:");
x_ind=input("Enter the index of x[n]:");
h=input("Enter the elements in h[n]:");
h_ind=input("Enter the index of h[n]:");
x1=x;
x=x(:,end:-1:1);
for i=1:length(x)
    x=[x(end) x(1:end-1)];
    y(i)=sum(x.*h);
end
disp(y);
figure;
subplot(3,1,1);
stem(x_ind,x1);
title("x[n]");
xlabel("time ");
ylabel("amplitude");
grid;
subplot(3,1,2);
stem(h_ind,h);
title("h[n]");
xlabel("time ");
ylabel("amplitude");
grid;
```

```
subplot(3,1,3);
Ny=0:3;
stem(Ny,y);
title("circular convolution output y[n]");
xlabel("time ");
ylabel("amplitude");
grid;
```

c)Using matrix method

```
clc;
clear all;
close all;
x=input("Enter the elements in x[n]:");
x_ind=input("Enter the index of x[n]:");
h=input("Enter the elements in h[n]:");
h_ind=input("Enter the index of h[n]:");
hr=[];
h1=h;
h=h(:,end:-1:1);
for i=1:length(h);
    h=[h(end) h(1:end-1)];
    hr=[hr;h];
end
y=hr*x';
disp(y);
figure;
subplot(3,1,1);
stem(x_ind,x);
title("x[n]");
```

```
xlabel("time ");
ylabel("amplitude");
grid;
subplot(3,1,2);
stem(h_ind,h);
title("h[n]");
xlabel("time ");
ylabel("amplitude");
grid;
subplot(3,1,3);
Ny=0:3;
stem(Ny,y);
title("circular convolution output y[n]");
xlabel("time ");
ylabel("amplitude");
grid;
```

Result

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

Observation

a)Using FFT and IFFT

INPUT:

Enter the elements in x[n]:

[2 1 2 1]

Enter the index of x[n]:

0:3

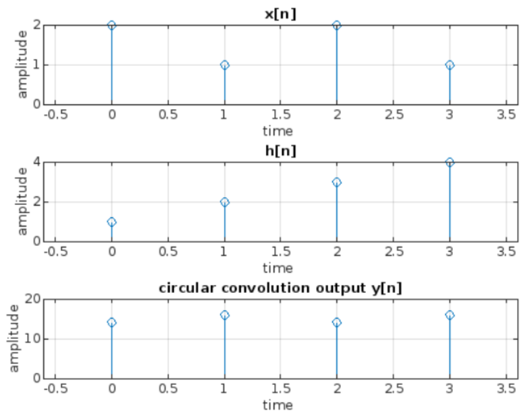
Enter the elements in h[n]:

[1 2 3 4]

Enter the index of h[n]:

0:3

OUTPUT:



b)Using concentric circle method

INPUT:

Enter the elements in x[n]:

[2 1 2 1]

Enter the index of x[n]:

0:3

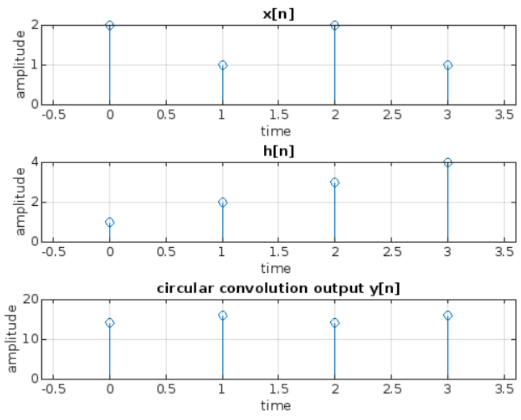
Enter the elements in h[n]:

[1 2 3 4]

Enter the index of h[n]:

0:3

OUTPUT:



c)Using matrix method

INPUT:

Enter the elements in x[n]:

[2 1 2 1]

Enter the index of x[n]:

0:3

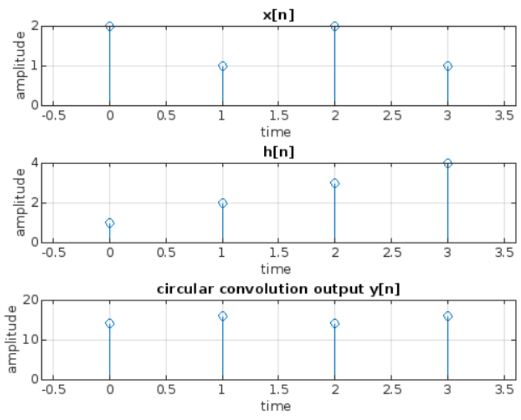
Enter the elements in h[n]:

[1 2 3 4]

Enter the index of h[n]:

0:3

OUTPUT:



Experiment No: 05 Date: 12-09-2024

Linear Convolution using Circular Convolution and Vice versa

Aim

1. To perform Linear Convolution using Circular Convolution.

2. To perform Circular Convolution using Linear Convolution.

Theory

Performing Linear Convolution Using Circular Convolution

Method:

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

Method:

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

Program

a)Linear convolution using circular convolution

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

b)Circular convolution using linear convolution

```
clc;
clear all;
close all;
x=[1 2 3 4];
h=[1 1 1];
y=conv(x,h);
conv_len=max(length(x),length(h));
result=[y(1:conv_len)];
```

```
new_arr=[y(conv_len+1:length(y)) zeros(1,length(y)-conv_len)];
result=result+new_arr;

disp("Circular convolution using Linear Convolution:")
disp(result);
```

Result

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

Observation

a)Linear convolution using circular convolution

```
Linear Convolution using Circular Convolution:
1     3     6     9     7     4
```

b)Circular convolution using linear convolution

```
Circular convolution using Linear Convolution:
8     7     6     9
```

Experiment No:06 Date: 3-09-2024

DFT AND IDFT

Aim

To perform DFT and IDFT with and without twiddle factor and to plot the magnitude and phase plot of DFT sequence.

Theory

Discrete Fourier Transform is the transformation used to represent the finite duration frequencies. DFT of a discrete sequence x(n) is obtained by performing sampling operations in both time domain and frequency domain. It is the frequency domain representation of a discrete digital signal.

The DFT of a sequence x (n) of length N is given by the following equation,

$$X(k) = \{\sum_{n=0}^{N-1} x(n)\, e^{\frac{-j2\Pi kn}{N}};0\le k\le N-1 \}$$

IDFT performs the reverse operation of DFT, to obtain the time domain sequence x(n) from frequency domain sequence X(k). IDFT of the sequence is given as,

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(K).\, e^{\frac{j2\Pi kn}{N}};n = 0,1,2,..,N-1$$

The IDFT takes the frequency components X[k] and reconstructs the original sequence x[n].

The exponential factor $e^{\frac{j2\Pi kn}{N}}$ is the inverse of the DFT's complex sinusoidal basis functions.

Program

a)

i)DFT

```
clc;

clear all;

close all;

x=[1 1 0 0];

N=length(x);
```

```
X=zeros(4,1);
for k=0:N-1
    for n =0:N-1
        X(k+1)=X(k+1)+x(n+1)*exp(-i*2*pi*n*k/N);
    end
end
disp(round(X));
%using built in function
disp("Using built in function:");
disp(fft(x));
```

ii)IDFT

```
clc;
clear all;
close all;
X=[2 1-i 0 1+i];
N=length(X);
x=zeros(4,1);
for n=0:N-1
    for k =0:N-1
        x(n+1)=(x(n+1)+X(k+1)*exp(i*2*pi*n*k/N));
    end
end
x=x/N;
disp(round(x));
%using built in function
disp("Using built in function:");
disp(ifft(X));
```

b)

i) DFT using twiddle factor

```
clc;
clear all;
close all;
x = [1 2 3 4];
N = length(x);
X = zeros(N, 1);
twiddle_factors = zeros(N, N);
for k = 0:N-1
    for n = 0:N-1
        twiddle = exp(-2*pi*1i*k*n/N);
        twiddle_factors(k+1, n+1) = twiddle;
        X(k+1) = X(k+1) + x(n+1) * twiddle;
    end
end
disp("Twiddle factors :");
disp(twiddle_factors);
%display dft result
disp("Dft of x: ");
disp(X);
```

ii)IDFT using twiddle factor

```
clc;
clear all;
close all;
X=[2,1-i,0,1+i];
N=length(X);
x=zeros(N,1);
twiddle_factors=zeros(N,N);
```

```
for n=0:N-1
    for k=0:N-1
        twiddle =exp(2*i*pi*k*n/N);
        twiddle_factors(n+1,k+1)=twiddle;
        x(n+1)=x(n+1)+X(k+1)*twiddle
    end
end
x=x/N;
disp(twiddle_factors);
disp("IDFT:");
disp(x);
```

c) Magnitude and phase plot of dft

```
clc;
clear all;
close all;
xn=[1 1 1];
N=input("enter the no: ");
 L=length(xn);
 if(N<L)
    error('N must be greater than or equal to L')
 end
 x=[xn,zeros(1,N-L)];
 N=length(x);
 Xk=zeros(N,1);
 for k=0:N-1
    for n =0:N-1
        Xk(k+1)=Xk(k+1)+x(n+1)*exp(-i*2*pi*n*k/N);
    end
 end
 mgXk=abs(Xk);
```

```
phaseXk=angle(Xk);
k=0:N-1;
subplot(2,1,1);
stem(k,mgXk);
hold on
plot(k,mgXk);
title('DFT sequence');
xlabel('Frequency');
ylabel('Magnitude');
subplot(2,1,2);
stem(k,phaseXk);
hold on
plot(k,phaseXk);
title('Phase of the DFT sequence');
xlabel('Frequency');
ylabel('Phase');
```

Result

Performed the DFT and IDFT operations using MATLAB and also plotting the phase and magnitude spectrum of DFT for better understanding. In addition performed the dft and and idft using twiddle factor .

Observation

a)

i)DFT

```
  2.0000 + 0.0000i
   1.0000 - 1.0000i
   0.0000 + 0.0000i
   1.0000 + 1.0000i
```

Using built in function:

```
   2.0000 + 0.0000i
   1.0000 - 1.0000i
   0.0000 + 0.0000i
   1.0000 + 1.0000i
```

ii)IDFT

```
     1
     1
     0
     0
```

Using built in function:

```
     1
     1
     0
     0
```

b)

i)DFT using twiddle factor

```
Twiddle factors for DFT:

 1.0000 + 0.0000i    1.0000 + 0.0000i   1.0000 + 0.0000i    1.0000 +
 0.0000i

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

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

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

DFT of x:

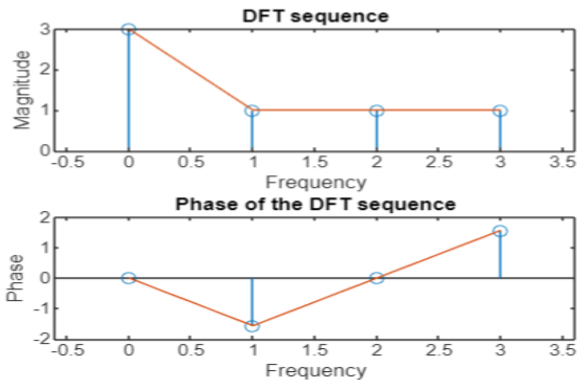
```
  10.0000 + 0.0000i

  -2.0000 + 2.0000i

  -2.0000 - 0.0000i

  -2.0000 - 2.0000i
```

ii)IDFT using twiddle factor

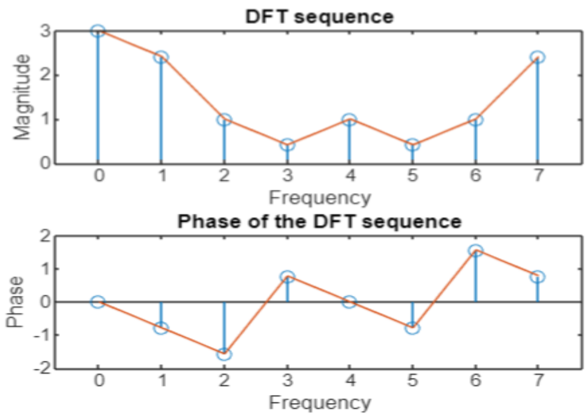
```
Twiddle factors for IDFT:
   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 + 0.0000i   1.0000 +
   0.0000i

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

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

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

IDFT result:
   1.0000 + 0.0000i

   1.0000 - 0.0000i

  -0.0000 + 0.0000i

   0.0000 + 0.0000i
```

c) Magnitude and phase plot of dft

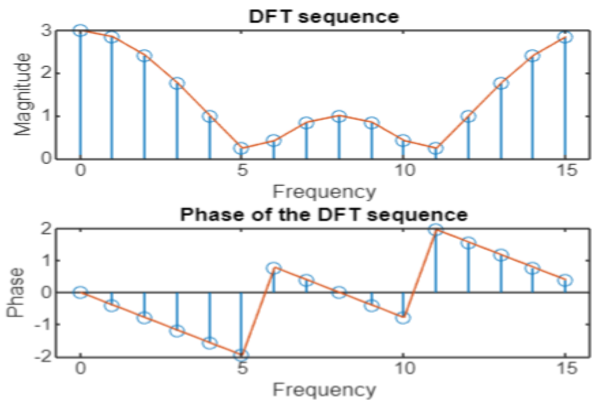
N=4



N=8



N=16



Experiment No:07 Date: 3-09-2024

PROPERTIES OF DFT

Aim

To perform the following properties of DFT:

1. Linearity
2. Convolution
3. Multiplication
4. Parsevals theorem

Theory

1. Linearity

The linearity property of DFT states that the DFT of a linear weighted combination of two or more signals is equal to similar linear weighted combination of the DFT of individual signals.

Let,

$$DFT\{x1(n)\}=X_1(K) \ \& \ DFT\{x2(n)\}=X2(K)$$

then,

$$DFT[a_1x_1(n)+a_2x_2(n)\}=a_1X_1(K)+a_2X_2(K) \text{ where a, \& } a_2 \text{ are constants}$$

2. Convolution

The Circular Convolution of two N-Point Sequences $x_1(n)$ & $x2(n)$ is defined as,

$$x_1(1) \ \Theta \ x2(n) = \Sigma{=}x_1(n) \ x_2((n-m))_N$$

The Convolution Property of DFT says that, the DFT of circular convolution of two sequences is equivalent to product of their individual DFTS. Let,

$$DFT[x_1(n)\}=X_1(K) \text{ and } DFT(x2(n)\}=X2(K)$$

then, By Convolution property,

$$DFT(x_1(n) \ \Theta \ x2(n)\}=X_1(K) \ X_2(K)$$

3. Multiplication

The Multiplication Property of DFT says that the DFT of product of two discrete time sequences is equivalent to circular convolution of DFT's of the individual sequences scaled by the factor of 1/N.If,

$$DFT(x(n))=X(K)$$

then,

$$DFT(x,(n)\ x2(n)\}=1/N[X_1(K)\Theta X_2(K))$$

4. Parseval theorem

Let DFT(x_1(n)]=X_1(K) & DFT[x2(n))=X2(K) then by Parseval' theorem

$$\Sigma_{n=0}^{N-1} x_1(n)x_2^*(n)=1/N\ \Sigma_{k=0}^{N-1} X_1(K)\ X_2^*(K)$$

Program

1)Linearity property

```
clc;
clear all;
close all;
x1=[1 2 3 4];
x2=[2 1 2 1];
a1=2;
a2=3;
x1k=fft(x1);
x2k=fft(x2);
lhs=(a1*x1)+(a2*x2);
lhsk=fft(lhs);
disp('LHS=');
disp(lhsk);
rhsk=(a1*x1k)+(a2*x2k);
disp("RHS=");
disp(rhsk);
```

2)Convolution property

```
clc;
```

```
close all;
clear all;
x1=[1 2 3 4];
x2= [2 1 2 1];
N=max(length(x1), length(x2));
x1new=[x1 zeros(1, N-length(x1))];
x2new=[x2 zeros(1, N-length(x2))];
X1= fft(x1new);
X2 =fft(x2new);
circular_conv_time =cconv(x1new, x2new, N);
product_freq =ifft(X1 .*X2);
disp("x1(n) cconv x2(n):");
disp(circular_conv_time);
disp("IDFT(X1(k)*X2(k)):");
disp(product_freq);
```

3)Multiplication property

```
clc;
close all;
clear all;
x1 = [1 2 1 2];
x2 =[1 2 3 4];
N=max(length(x1), length(x2));
x1new=[x1 zeros(1, N-length(x1))];
x2new=[x2 zeros(1, N-length(x2))];
product_time=x1new.*x2new;
dft_product_time=fft(product_time);
X1=fft(x1new);
X2=fft(x2new);
```

```
Y=cconv (X1,X2, N);
disp("DFT(x1(n)*x2(n)}:");
disp(dft_product_time);
disp("X1(k)circconvX2(k)/N:");
disp(Y./N);
```

4)Parsevals theorem

```
clc;
clear all;
close all;
x1=[1 2 1 1];
x2=[1 2 3 4];
N =max(length(x1), length(x2));
x1new=[x1 zeros(1, N-length(x1))];
x2new=[x2 zeros(1, N-length(x2))];
time_domain_value=sum(x1new.*conj (x2new));
freq_domain_value = sum(fft(x1new).*conj (fft(x2new)))./ N;
disp("Sum{n:0->N-1; x1(n) *conj(x2(n))}:");
disp(time_domain_value);
disp("Sum{k:0->N-1;X1(k)*conj(X2(k))}/N:");
disp(freq_domain_value);
```

Result

Performed various properties of dft and verified the outputs.

Observation

1)Linearity property

```
LHS=
38.0000 + 0.0000i -4.0000 + 4.0000i 2.0000 + 0.0000i -4.0000 - 4.0000i
RHS=
38.0000 + 0.0000i -4.0000 + 4.0000i 2.0000 + 0.0000i -4.0000 - 4.0000i
```

2)Convolution property

```
x1(n) cconv x2(n):
   14    16    14    16
IDFT(X1(k)*X2(k)):
   14    16    14    16
```

3)Multiplication property

```
DFT(x1(n)*x2(n)}:
  16.0000 + 0.0000i  -2.0000 + 4.0000i  -8.0000 + 0.0000i  -2.0000 - 4.0000i
X1(k)circonvX2(k)/N:
  16.0000 + 0.0000i  -2.0000 + 4.0000i  -8.0000 + 0.0000i  -2.0000 - 4.0000i
```

4)Parsevals theorem

```
Sum{n:0->N-1; x1(n) *conj(x2(n))}:
12
Sum{k:0->N-1;X1(k)*conj(X2(k))}/N:
12
```

Experiment No:08 Date: 03-10-2024

OVERLAP ADD AND OVERLAP SAVE METHOD

Aim

To simulate linear convolution of two signals using overlap add and overlap save methods

Theory

1. Overlap-Add Method

This procedure cuts the signal up into equal length segments with no overlap. Then it zero-pads the segments and takes the DFT of the segments. Part of the convolution result corresponds to the circular convolution. The tails that do not correspond to the circular convolution are added to the adjoining tail of the previous and subsequent sequence. This addition results in the aliasing that occurs in circular convolution.

2. Overlap-Save Method

This procedure cuts the signal up into equal length segments with some overlap. Then it takes the DFT of the segments and saves the parts of the convolution that correspond to the circular convolution. Because there are overlapping sections, it is like the input is copied therefore there is not lost information in throwing away parts of the linear convolution.

Program

1) Overlap add method

```
clc;

clear all;

close all;

% Step 1: Define the Input Signal and Filter

Xn =input('enter the elements of xn: ');

Hn = input('enter the elements of hn: ');

L = 5;

NXn = length(Xn);

M = length(Hn);
```

```
M1 = M - 1;

R = rem(NXn, L);

N = L + M1;

% zero padding

Xn = [Xn, zeros(1, L - R)];

Hn = [Hn, zeros(1, N - M)];

%overlap add method

K = floor(length(Xn) / L);

y = zeros(1, length(Xn) + M - 1);

z = zeros(1, M1);


for k = 0:K-1

    startIndex = L * k + 1;

    endIndex = min(startIndex + L - 1, length(Xn));

    Xnp = Xn(startIndex:endIndex);


    Xnk = [Xnp, z];

    convSegment = mycirconv(Xnk, Hn);


    % Add the current segment to the output

    outputStart = startIndex;

    outputEnd = outputStart + N - 1;

    y(outputStart:outputEnd) = y(outputStart:outputEnd) +
convSegment(1:N);

end

disp('Input Signal:');

disp(Xn);

disp('Filter:');

disp(Hn);

disp('Output Signal (Convolved using Overlap-Add Method):');
disp(y);

% Function for Circular Convolution

function y = mycirconv(x, h)

    % Compute the circular convolution using FFT

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

    y = ifft(fft(x, N) .* fft(h, N)); % FFT-based circular
convolution

end
```

2) Overlap save method

```
clc;

clear all;

close all;

x = input('enter the elements of x: ');

h = input('enter the elements of h: ');

N = 5;

lx = length(x);

lh = length(h);

m = lh - 1;

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

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

L = N - lh + 1;

k = floor((length(x) - m) / L);

p = [];

for i = 0:k-1

    y = x(i * L + 1 : i * L + N);

    q = mycirconv1(y, h);

    p = [p, q(lh:N)];

end

disp("Output Signal (Convolved using Overlap-Save Method):");
disp(p);
function y = mycirconv1(x, h)
    N = length(x);
    y = ifft(fft(x, N) .* fft(h, N));
end
```

Result

Performed overlap add and overlap save method using MATLAB and verified the output.

Observation

1)Overlap add method

```
Input signal

Xn =[2,-2,8,-2,-2,-3,-2,1,-1,9,1,3];

Hn = [1,2,3];

output Signal (Convolved using Overlap-Add Method)


2.0000    2.0000   10.0000    8.0000   18.0000  -13.0000  -14.0000
-12.0000   -5.0000   10.0000   16.0000   32.0000    9.0000    9.0000
```

2)Overlap save method

```
Input signal

x = [1 2 3 4 5 6 7 8 9 10];

h = [1 1 1];

Output Signal (Convolved using Overlap-Save Method):


1.0000    3.0000    6.0000    9.0000   12.0000   15.0000   18.0000
21.0000   24.0000   27.0000   19.0000   10.0000
```

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

Window Type	Formula
Rectangular	w(n) = 1
Triangular	w(n) = 1 - 2*abs(n) / (N - 1)
Hamming	w(n) = 0.54 - 0.46 * cos(2 * pi * n / (N - 1))
Hanning	w(n) = 0.5 * (1 - cos(2 * pi * n / (N - 1)))

Blackman	w(n) = 0.42 - 0.5 * cos(2 * pi * n / (N - 1)) + 0.08 * cos(4 * pi * n / (N - 1))
Kaiser	$w(n) = I_0(\beta\sqrt{1 - (2n/L-1)^2}) / I_0(\beta)$

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.

Step 3: Apply the Window 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('Please enter the order of the filter: ');

alpha=(N-1)/2;

n=0:1:N-1;

eps=0.001;

w=0:0.01:pi;%note the normalization later

hd_lowpass=sin(wc*(n-alpha+eps))./(pi*(n-alpha+eps)); %Coefficients
of IIR filter

freq_hd_lowpass=freqz(hd_lowpass,1,w);

w_rect=boxcar(N);

w_hann=hamming(N);

w_hamm=hamming(N);

w_bartlett=bartlett(N);


subplot(4,2,1);

stem(w_rect);

title('Rectangular Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');


hn1=hd_lowpass.*w_rect;%both beingrow vectors

h1=freqz(hn1,1,w);

subplot(4,2,2);

plot(w/pi,10*log10(abs(h1))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');


subplot(4,2,3);

stem(w_hamm);

title('Hamming Window sequence');
```

```
xlabel('No. of samples');

ylabel('Amplitude');


hn2=hd_lowpass.*w_hamm; %both being row vectors

h2=freqz(hn2,1,w);

subplot(4,2,4);

plot(w/pi,10*log10(abs(h2))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');


subplot(4,2,5);

stem(w_hann);

title('Hanning Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');


hn3=hd_lowpass.*w_hann; %both being row vectors

h3=freqz(hn3,1,w);

subplot(4,2,6);

plot(w/pi,10*log10(abs(h3))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');

subplot(4,2,7);

stem(w_bartlett);

title('Triangular Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');
```

```
hn4=hd_lowpass.*w_bartlett'; %both being row vectors

h4=freqz(hn4,1,w);

subplot(4,2,8);

plot(w/pi,10*log10(abs(h4))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');
```

2)HIGH PASS FILTER

```
clc;

clear all;

close all;

wc=0.5*pi;

N=input('Please enter the order of the filter: ');

alpha=(N-1)/2;

n=0:1:N-1;

eps=0.001;

w=0:0.01:pi;%note the normalization later

hd=((sin(pi*(n-alpha+eps))-sin(wc*(n-alpha+eps)))./(pi*(n-
alpha+eps))); %Coefficients of IIR filter

freq_hd=freqz(hd,1,w);

w_rect=boxcar(N);

w_hann=hamming(N);

w_hamm=hamming(N);

w_bartlett=bartlett(N);


subplot(4,2,1);

stem(w_rect);

title('Rectangular Window sequence');
```

```
xlabel('No. of samples');

ylabel('Amplitude');


hn1=hd.*w_rect;%both beingrow vectors

h1=freqz(hn1,1,w);

subplot(4,2,2);

plot(w/pi,10*log10(abs(h1))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');


subplot(4,2,3);

stem(w_hamm);

title('Hamming Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');


hn2=hd.*w_hamm; %both being row vectors

h2=freqz(hn2,1,w);

subplot(4,2,4);

plot(w/pi,10*log10(abs(h2))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');


subplot(4,2,5);

stem(w_hann);

title('Hanning Window sequence');

xlabel('No. of samples');
```

```
ylabel('Amplitude');


hn3=hd.*w_hann; %both being row vectors

h3=freqz(hn3,1,w);

subplot(4,2,6);

plot(w/pi,10*log10(abs(h3))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');

subplot(4,2,7);

stem(w_bartlett);

title('Triangular Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');


hn4=hd.*w_bartlett'; %both being row vectors

h4=freqz(hn4,1,w);

subplot(4,2,8);

plot(w/pi, 10*log10(abs(h4))); %To find magnitude in db,w/pi taken
to normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');
```

3)BANDPASS FILTER

```
clc;

clear all;

close all;

wc2=0.5*pi;

wc1=0.2*pi;
```

```
N=input('Please enter the order of the filter: ');

alpha=(N-1)/2;

n=0:1:N-1;

eps=0.001;

w=0:0.01:pi;%note the normalization later

hd=((sin(wc2*(n-alpha+eps))-sin(wc1*(n-alpha+eps)))./(pi*(n-
alpha+eps))); %Coefficients of IIR filter

freq_hd=freqz(hd,1,w);

w_rect=boxcar(N);

w_hann=hamming(N);

w_hamm=hamming(N);

w_bartlett=bartlett(N);


subplot(4,2,1);

stem(w_rect);

title('Rectangular Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');


hn1=hd.*w_rect;%both beingrow vectors

h1=freqz(hn1,1,w);

subplot(4,2,2);

plot(w/pi,10*log10(abs(h1))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');


subplot(4,2,3);

stem(w_hamm);

title('Hamming Window sequence');
```

```
xlabel('No. of samples');
ylabel('Amplitude');

hn2=hd.*w_hamm; %both being row vectors
h2=freqz(hn2,1,w);
subplot(4,2,4);
plot(w/pi,10*log10(abs(h2))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1
title('Frequency response of windowed FIR filter');
xlabel('normalized frequency');
ylabel('Magnitude in dB');

subplot(4,2,5);
stem(w_hann);
title('Hanning Window sequence');
xlabel('No. of samples');
ylabel('Amplitude');

hn3=hd.*w_hann; %both being row vectors
h3=freqz(hn3,1,w);
subplot(4,2,6);
plot(w/pi,10*log10(abs(h3))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1
title('Frequency response of windowed FIR filter');
xlabel('normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,7);
stem(w_bartlett);
title('Triangular Window sequence');
xlabel('No. of samples');
ylabel('Amplitude');
```

```
hn4=hd.*w_bartlett'; %both being row vectors

h4=freqz(hn4,1,w);

subplot(4,2,8);

plot(w/pi,10*log10(abs(h4))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');
```

4)BANDSTOP FILTER

```
clc;

clear all;

close all;

wc2=0.5*pi;

wc1=0.2*pi;

N=input('Please enter the order of the filter: ');

alpha=(N-1)/2;

n=0:1:N-1;

eps=0.001;

w=0:0.01:pi;%note the normalization later

hd=((sin(wc1*(n-alpha+eps))-sin(wc2*(n-alpha+eps))+sin(pi*(n-
alpha+eps)))./(pi*(n-alpha+eps))); %Coefficients of IIR filter

freq_hd=freqz(hd,1,w);

w_rect=boxcar(N);

w_hann=hamming(N);

w_hamm=hamming(N);

w_bartlett=bartlett(N);


subplot(4,2,1);

stem(w_rect);
```

```
title('Rectangular Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');


hn1=hd.*w_rect;%both beingrow vectors

h1=freqz(hn1,1,w);

subplot(4,2,2);

plot(w/pi,10*log10(abs(h1))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');


subplot(4,2,3);

stem(w_hamm);

title('Hamming Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');


hn2=hd.*w_hamm; %both being row vectors

h2=freqz(hn2,1,w);

subplot(4,2,4);

plot(w/pi,10*log10(abs(h2))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');

subplot(4,2,5);

stem(w_hann);

title('Hanning Window sequence');

xlabel('No. of samples');
```

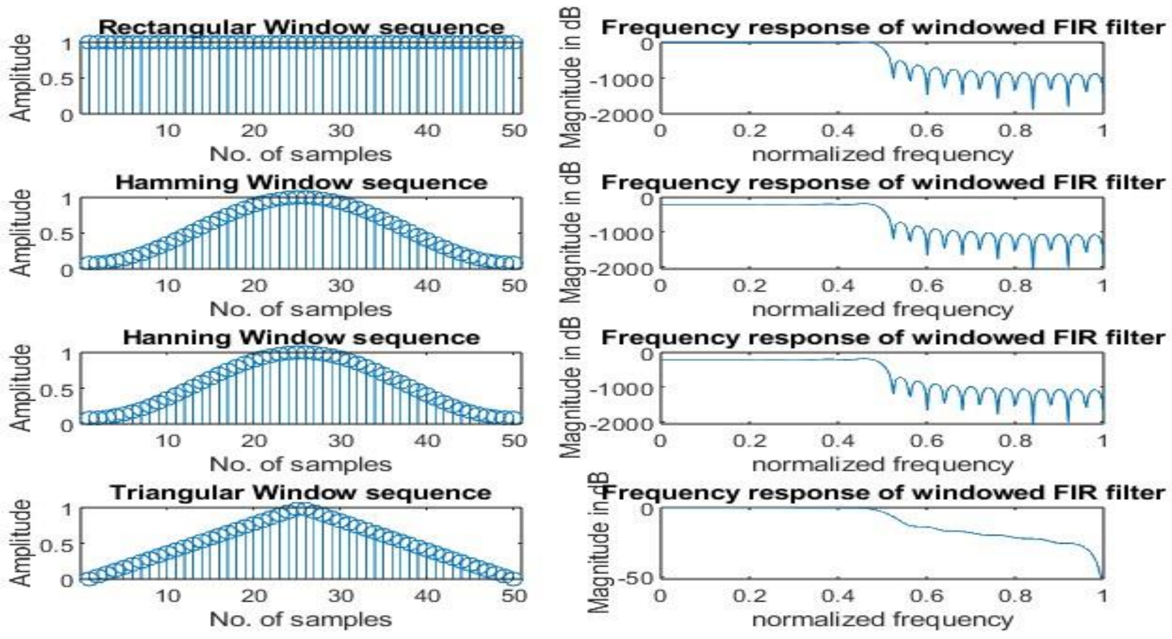
```
ylabel('Amplitude');

hn3=hd.*w_hann; %both being row vectors

h3=freqz(hn3,1,w);

subplot(4,2,6);

plot(w/pi,10*log10(abs(h3))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');

subplot(4,2,7);

stem(w_bartlett);

title('Triangular Window sequence');

xlabel('No. of samples');

ylabel('Amplitude');

hn4=hd.*w_bartlett'; %both being row vectors

h4=freqz(hn4,1,w);

subplot(4,2,8);

plot(w/pi,10*log10(abs(h4))); %To find magnitude in db,w/pi taken to
normalize frequency axis from 0 to 1

title('Frequency response of windowed FIR filter');

xlabel('normalized frequency');

ylabel('Magnitude in dB');
```

Result:

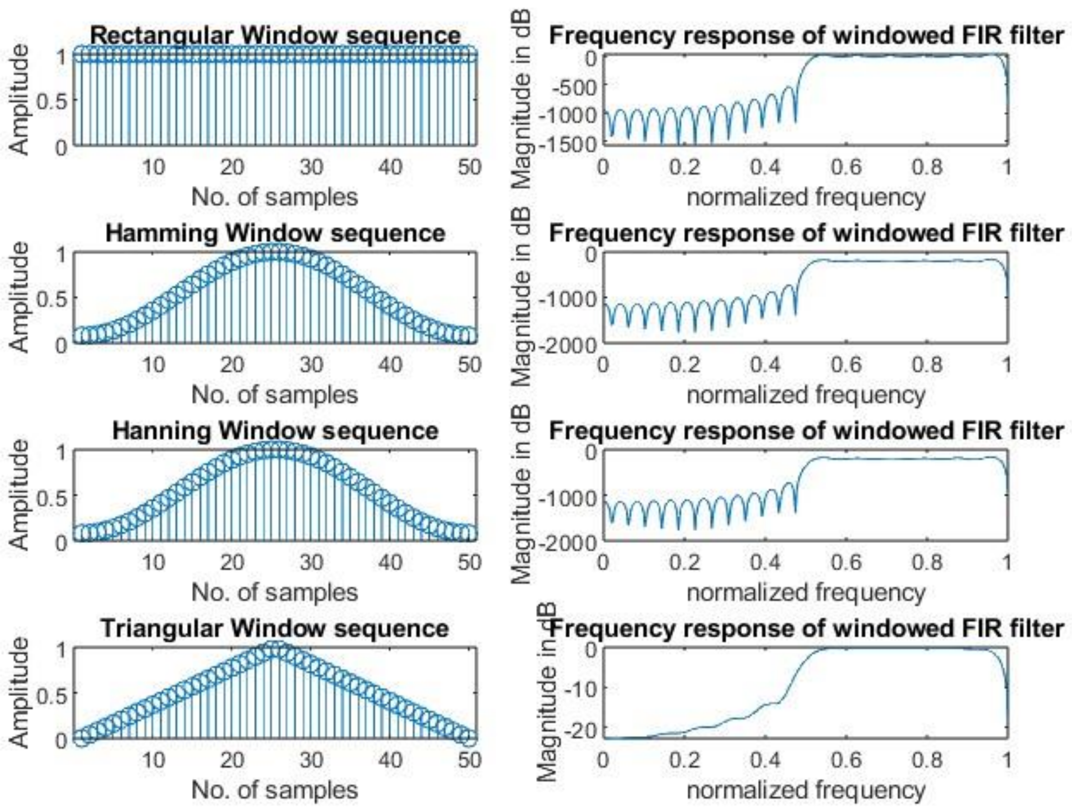
Implemented various FIR filters using different windows
1.Low Pass Filter
2.High Pass Filter
3.Band pass Filter
4.Band stop Filter

Observation:

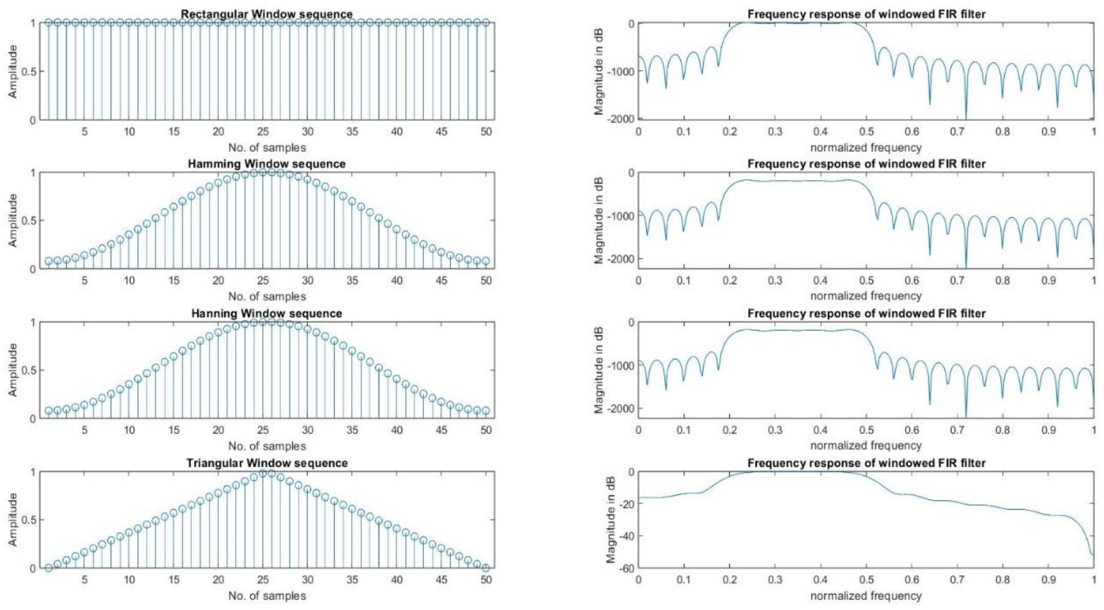
1.LOW PASS FILTER



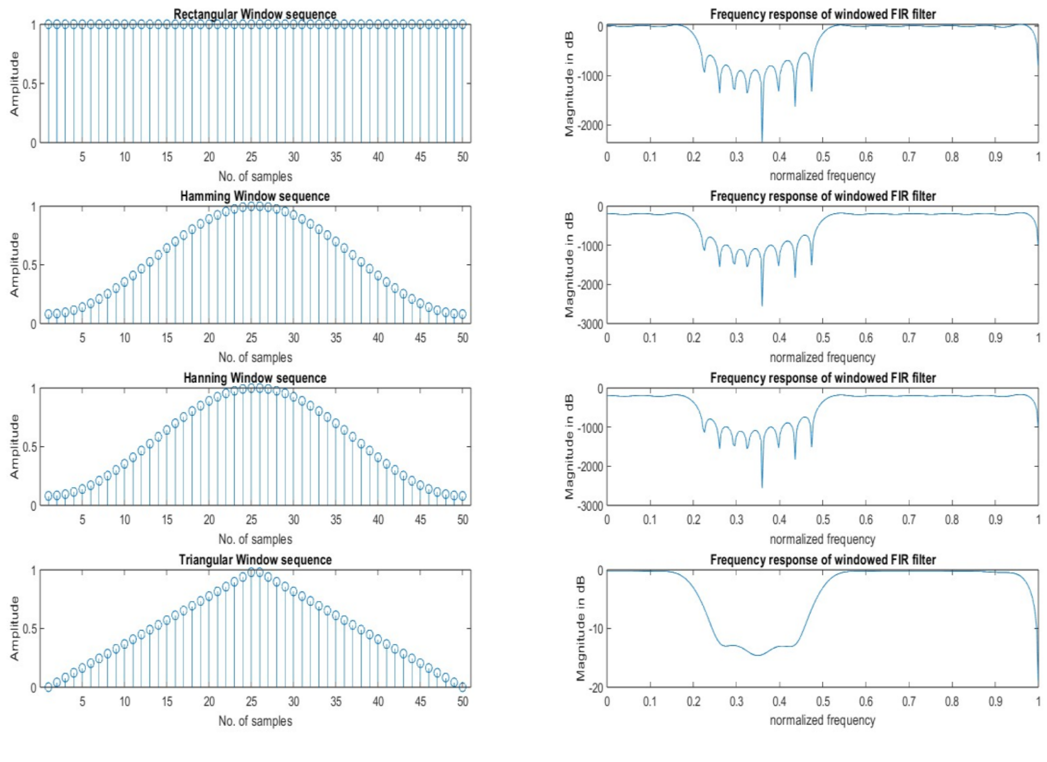
2.HIGH PASS FILTER



3.BAND PASS FILTER



4.BAND STOP FILTER



Experiment No: 10 Date: 24/10/24

FAMILIARIZATION OF DSP HARDWARE

Aim:

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

Theory:

TMS 320C674x DSP CPU

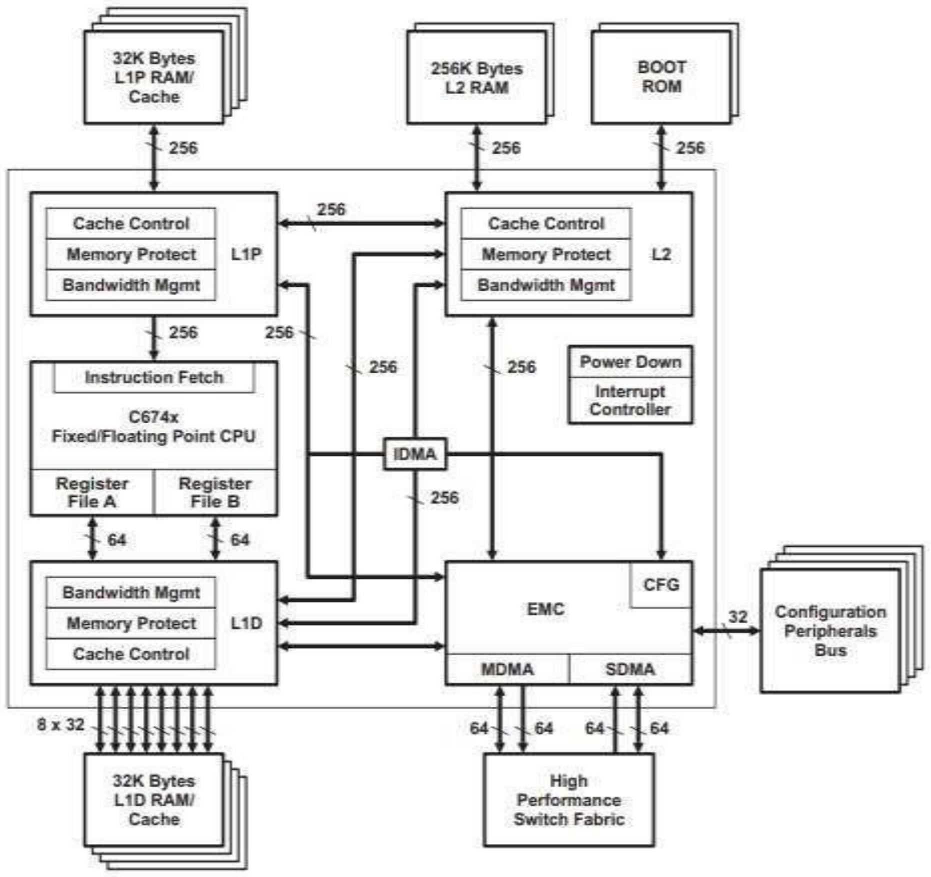


FIGURE: TMS320C 674X DSP CPU BLOCK DIAGRAM

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.

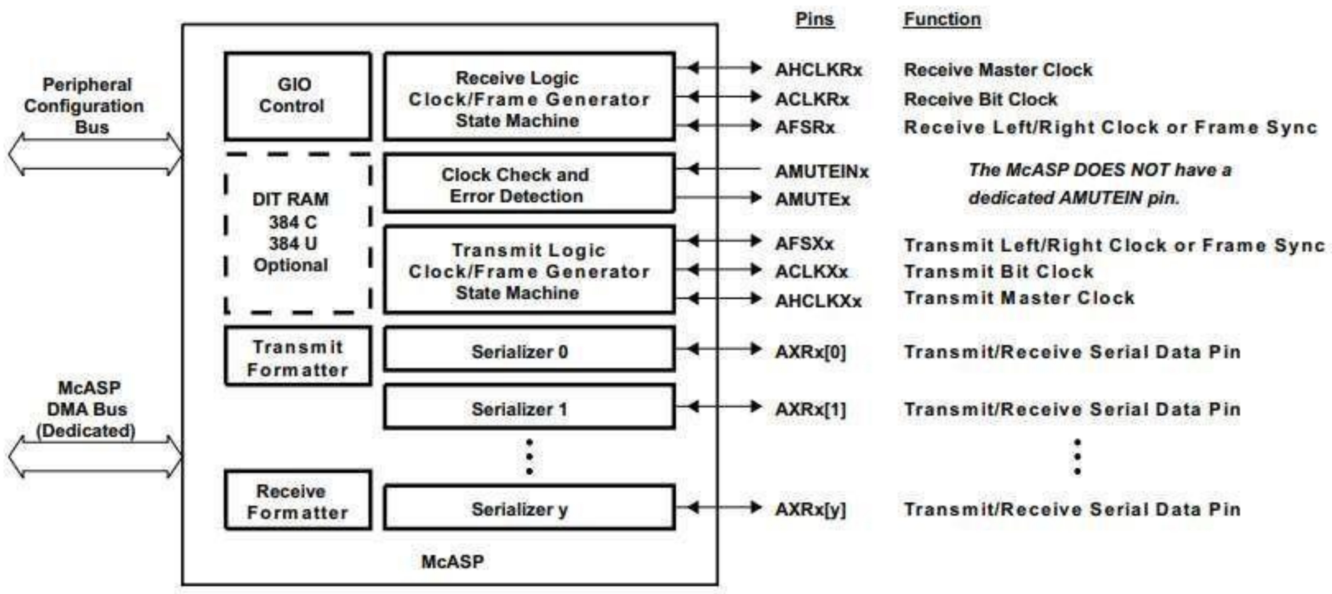
Multichannel Audio Serial Port (McASP):
The McASP serial port is specifically designed for multichannel audio applications.
Its key features are:

- 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.


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

Result:

Familiarized the input and output ports of dsp board.

Experiment No: 11 Date: 24 /10/24

GENERATION OF SINE WAVE USING DSP PROCESSOR

Aim:

To generate a sine wave using DSP processor

Theory:

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

These are sinusoidal signal parameters:

• **Graph**: It is a plot used to depict the relation between quantities. Depending upon the number of variables, we can decide to number of axes each perpendicular to the other.
• **Time period**: The period for a signal can be defined as the time taken by a periodic signal to complete one cycle.
• **Amplitude**: Amplitude can be defined as the maximum distance between the horizontal axis and the vertical position of any signal.
• **Frequency:** This can be defined as the number of times a signal oscillates in one second. It can be mathematically defined as the reciprocal of a period.
• **Phase**: It can be defined as the horizontal position of a waveform in one oscillation. The symbol θ is used to indicate the phase.
If we consider a sinusoidal signal y(t) having an amplitude A, frequency f, and phase of quantity then we can represent the signal as

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

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

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

Procedure

1. Open Code Composer Studio,
   Click on File - New – CCS Project

Select the Target – C674X Floating point DSP , TMS320C6748 , and
Connection – Texas Instruments XDS 100v2 USB Debug Probe and Verify.
Give the project name and select Finish.

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

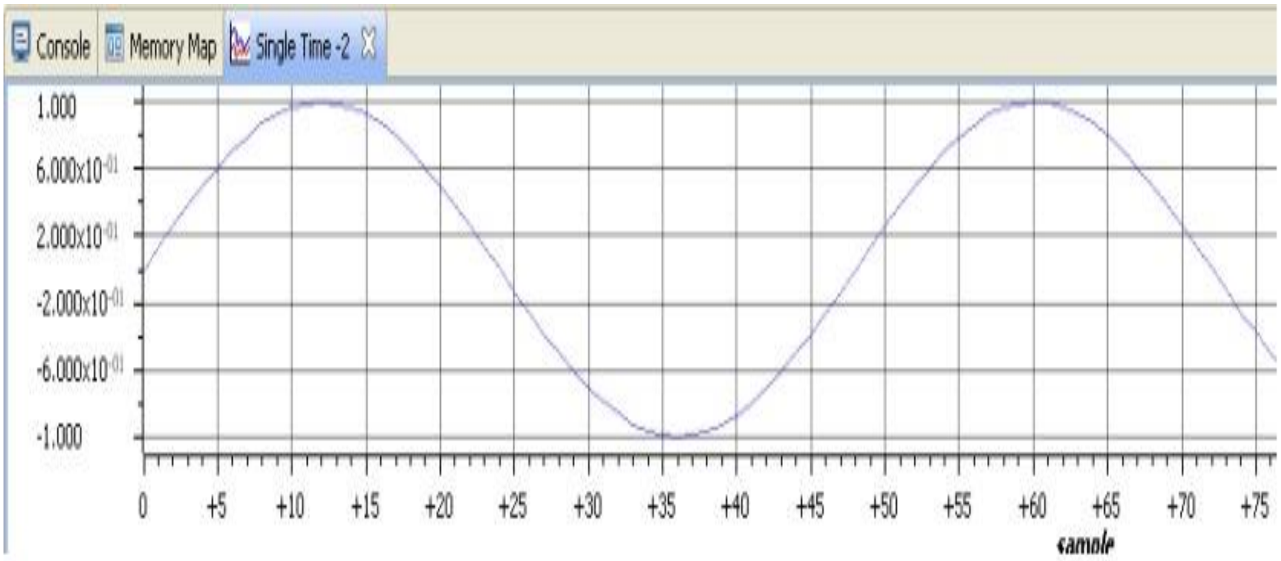
Program

```
//Sine wave using dsp kit
#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 a sine wave using DSP processor.

Observation



Experiment No:12 Date: 24/10/24

Linear Convolution using DSP Processor

Aim

To perform linear convolution of two sequences using DSP processor

Theory

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

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

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

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

where:

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

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

Applications of Linear Convolution :

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

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

```
//Linear convolution using dsp kit

#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 of two sequences using DSP processor

Observation

```
Xn
0x80010000 - 1
0x80010004 - 2
0x80010008 - 3
Hn
0x80011000 - 1
0x80011004 - 2
XnLength
0x80012000 - 3
S
HnLength
0x80012004 - 2

Output
0x80013000 - 1
0x80013004 - 4
0x80013008 - 7
0x8001300C - 6
```