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 Smt. [NANDANA SHIBU] of ***5th*** *Semester class (Roll No.* ***[B22ECB53] 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 : ***[TKM22EC097]***

Staff Member in-charge *External Examiner*

Date:

INDEX

SL No.	DATE	NAME OF THE EXPERIMENTS	PAGE NO.	REMARKS
1	25/08/2024	Simulation of Basic Test Signals		
2	01/08/2024	Verification of Sampling Theorem		
3	08/08/2024	Linear Convolution		
4	15/08/2024	Circular Convolution		
5	22/08/2024	Linear Convolution using circular convolution and vice versa		

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

Experiment Name: Simulation of Basic Test Signals

Aim:

To generate continuous and discrete waveforms for the following:

1. Unit Impulse Signal
2. Bipolar Pulse Signal
3. Unipolar Pulse Signal
4. Ramp Signal
5. Triangular Signal
6. Sine Signal
7. Cosine Signal
8. Exponential Signal
9. Unit Step Signal

Theory:

1. Unit Impulse Signal:

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

2. Bipolar Pulse Signal:

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

3. Unipolar Pulse Signal:

- A pulse signal that alternates between zero and a positive value. It remains at zero for a specified duration and then jumps to a positive constant level (e.g., 0 and 1).
- **Mathematically p(t) = A for $|t| \leq \tau/2$, p(t) = 0 otherwise (assuming A is positive)**

4. Ramp Signal:

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

5. Triangular Signal:

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

6. Sine Signal:

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

7. Cosine Signal:

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

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)**

9. Unit Step Signal:

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

Program:

```
clc;

clear all;

close all;
```

```
subplot(3,3,1);
t = -5:1:5;
y = [zeros(1,5),ones(1,1),zeros(1,5)];
stem(t,y);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Impulse Signal");

subplot(3,3,2);
t2 = 0:0.01:1;
f = 5;
y2 = square(2*pi*f*t2);
stem(t2,y2);
hold on;
plot(t2,y2);
xlabel("Time(s)");
ylabel("Amplitude");
title("Bipolar Pulse Signal");
legend("Discrete","Continuous");

subplot(3,3,3);
t3 = 0:0.1:1;
f = 5;
y3 = abs(square(2*pi*f*t3));
stem(t3,y3);
hold on;
plot(t3,y3);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unipolar Pulse Signal");
legend("Discrete","Continuous");
```

```
subplot(3,3,4);
t4 = -5:1:5;
y4 = t4 .*(t4>=0);
stem(t4,y4);
hold on;
plot(t4,y4);
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Ramp Signal");
legend("Discrete","Continuous");

subplot(3,3,5);
t5 = 0:0.025:1;
f = 10;
y5 = sawtooth(2*pi*f*t5,0.5);
stem(t5,y5);
hold on;
plot(t5,y5);
xlabel("Time(s)");
ylabel("Amplitude");
title("Triangular Signal");
legend("Discrete","Continuous");

subplot(3,3,6);
t6 = 0:0.001:1;
f = 10;
y6 = sin(2*pi*f*t6);
stem(t6,y6);
hold on;
plot(t6,y6);
xlabel("Time(s)");
```

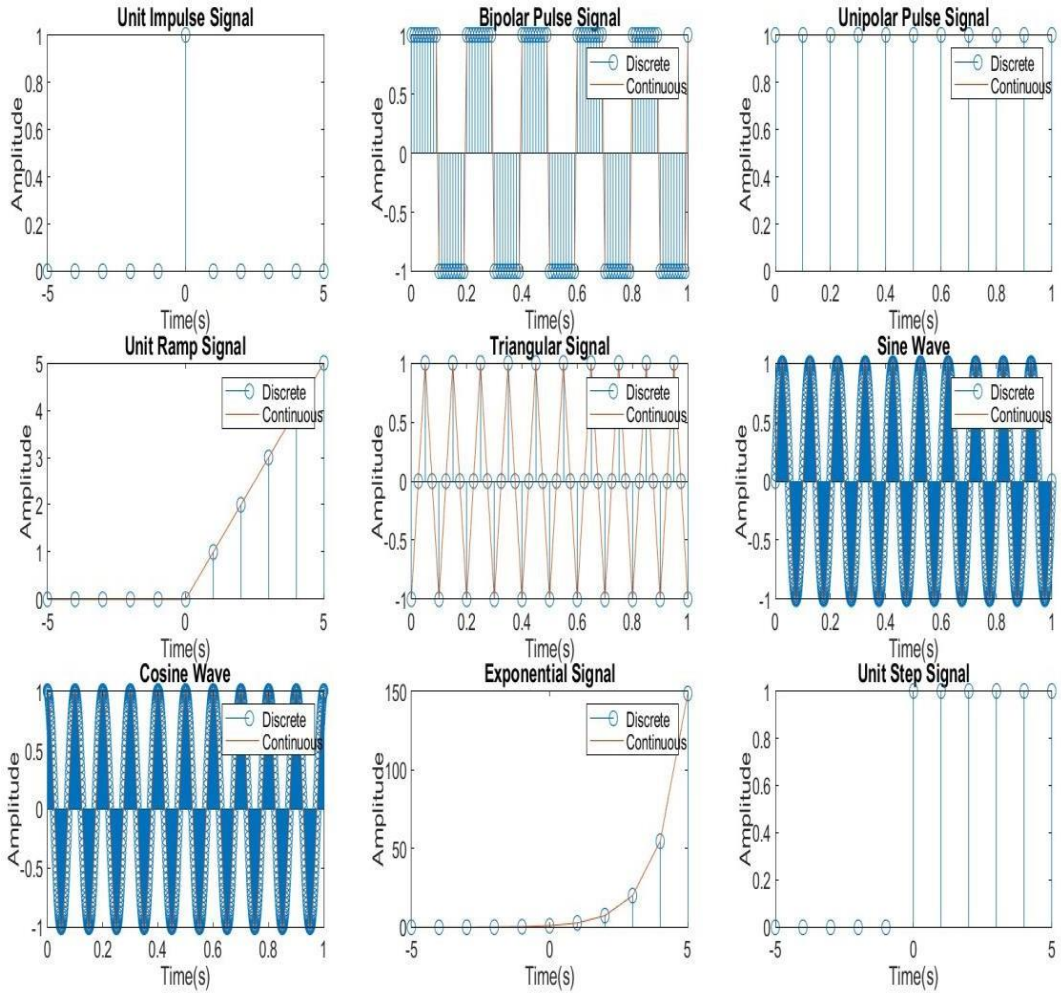
```
ylabel("Amplitude");
title("Sine Wave");
legend("Discrete","Continuous");

subplot(3,3,7);
t7 = 0:0.001:1;
f = 10;
y7 = cos(2*pi*f*t7);
stem(t7,y7);
hold on;
plot(t7,y7);
xlabel("Time(s)");
ylabel("Amplitude");
title("Cosine Wave");
legend("Discrete","Continuous");

subplot(3,3,8);
t8 = -5:1:5;
y8 = exp(t8);
stem(t8,y8);
hold on;
plot(t8,y8);
xlabel("Time(s)");
ylabel("Amplitude");
title("Exponential Signal");
legend("Discrete","Continuous");

subplot(3,3,9);
t9 = -5:1:5;
y9 = [zeros(1,5),ones(1,6)];
 stem(t9,y9);
```

```
xlabel("Time(s)");
ylabel("Amplitude");
title("Unit Step Signal");
```

Result

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

Observation



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

Experiment Name: 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.

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

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;


subplot(2,2,1);

t = 0:0.01:1;

f=10;
```

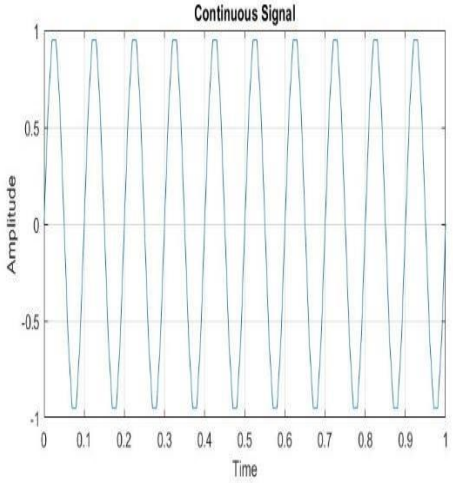
```
y = sin(2*pi*f*t);
plot(t,y);
grid(true);
xlabel("Time");
ylabel("Amplitude");
title("Continuous Signal");

 subplot(2,2,2);
 fs= 0.5*f; Undersampled
 t1 = 0:1/fs:1;
 y1 = sin(2*pi*f*t1);
 stem(t1,y1);
 hold on;
 plot(t1,y1);
 grid(true);
 xlabel("Time");
 ylabel("Amplitude");
 title("Under Sampled Signal");

 subplot(2,2,3);
 fs2= 3*f; Nyquist sampled
 t3 = 0:1/fs2:1;
 y2 = sin(2*pi*f*t3);
 stem(t3,y2);
 hold on;
 plot(t3,y2);
 xgrid(true);
 xlabel("Time");
 ylabel("Amplitude");
 legend("Discrete","Continuous")
```

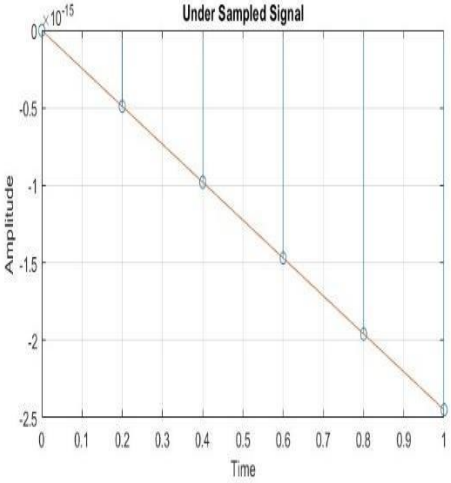
```
title("Nyquist Sampled Signal");
subplot(2,2,4);

fs2= 100*f; Oversampled
t3 = 0:1/fs2:1;
y2 = sin(2*pi*f*t3);
stem(t3,y2);
hold on;
plot(t3,y2);
grid(true);
xlabel("Time");
ylabel("Amplitude");
legend("Discrete","Continuous")
title("Over Sampled Signal");
```

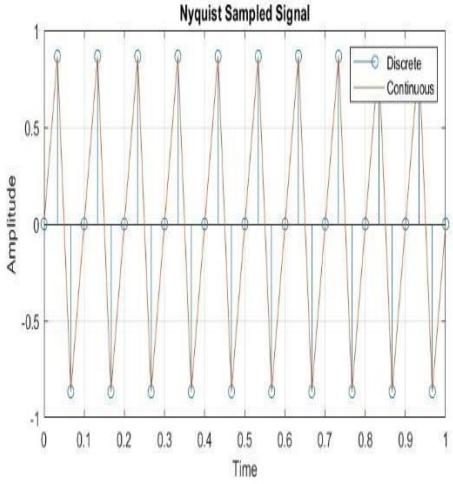
Result

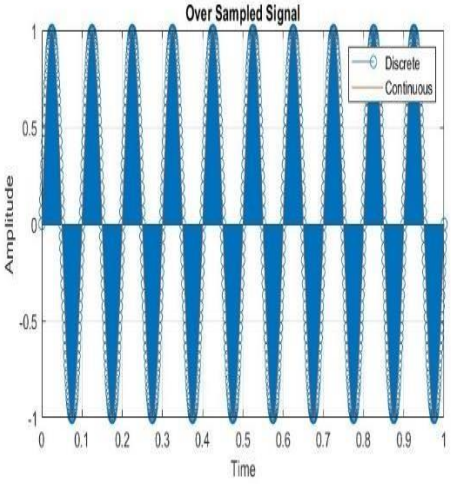
Verified Sampling Theorem using MATLAB.

Observation









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

Experiment Name: Linear Convolution

Aim:

To find linear convolution of following sequences with and without built in function.

- 1. x(n) = [1 2 1 1]
  h(n) = [1 1 1 1]
- 2. x(n) = [1 2 1 2]
  h(n) = [3 2 1 2]

Theory:

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

Mathematical Definition:

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

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

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:

1. with built-in function:

```
clc;

clear all;

close all;

x1 = input("Enter first Sequence");

h1 = input("Enter second Sequence");

y1 = conv(x1,h1);

disp("The convoluted sequence is: ");
```

```
disp(y1);
l = length(x1);
m = length(h1);
k = l+m-1;
n1 = 0:1:l-1;
n2 = 0:1:m-1;
n3 = 0:1:k-1;
 subplot(1,3,1);
 stem(n1,x1,"o");
 xlabel("n");
 ylabel("Amplitude");
 title("x(n)");
 grid on
 xlim([-1 l+1]);
 ylim([0 max(x1)+2]);

 subplot(1,3,2);
 stem(n2,h1,"o");
 xlabel("n");
 ylabel("Amplitude");
 title("h(n)");
 grid on
 xlim([-1 m+1]);
 ylim([0 max(h1)+2]);

 subplot(1,3,3);
 stem(n3,y1,"o");
 xlabel("n");
 ylabel("Amplitude");
 title("y(n)");
 grid on
 xlim([-1 k+1]);
```

```
ylim([0 max(y1)+2]);
```

2. without built-in function:

```
clc;
clear all;
close all;
x1 = input("Enter first Sequence");
h1 = input("Enter second Sequence");
l = length(x1);
m = length(h1);
k = l+m-1;
y1 = zeros(1,k);
for i=1:l
  for j=1:m
    y1(i+j-1) = y1(i+j-1) + x1(i)*h1(j);
  end
end
disp("The convoluted sequence is: ");
disp(y1);

n1 = 0:1:l-1;
n2 = 0:1:m-1;
n3 = 0:1:k-1;
subplot(1,3,1);
stem(n1,x1,"o");
xlabel("n");
ylabel("Amplitude");
title("x(n)");
grid on
xlim([-1 l+1]);
```

```
ylim([0 max(x1)+2]);

subplot(1,3,2);
stem(n2,h1,"o");
xlabel("n");
ylabel("Amplitude");
title("h(n)");



grid on
xlim([-1 m+1]);
ylim([0 max(h1)+2]);

subplot(1,3,3);
stem(n3,y1,"o");
xlabel("n");
ylabel("Amplitude");
title("y(n)");
grid on
xlim([-1 k+1]);
ylim([0 max(y1)+2]);
```

Result

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

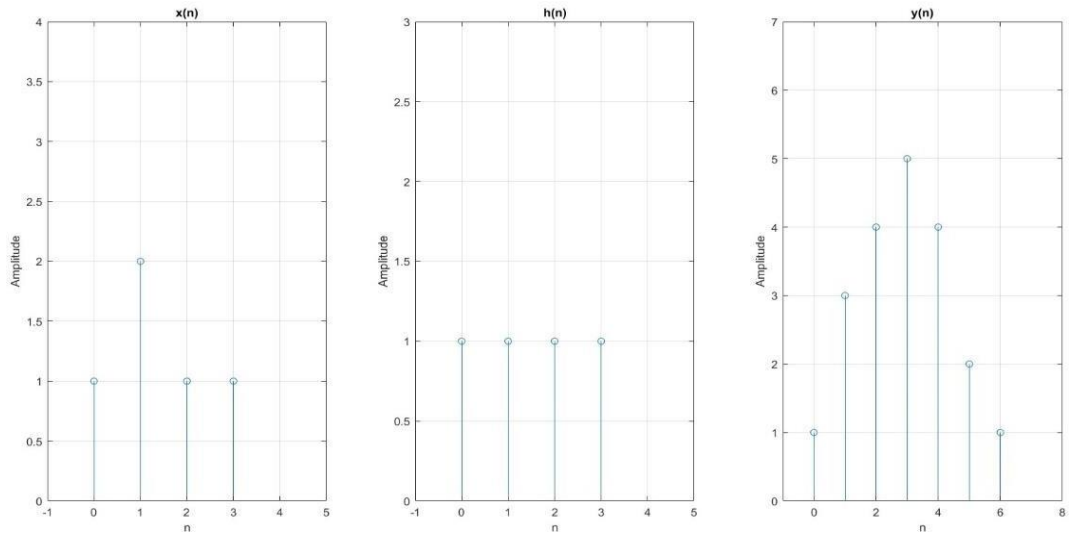
Observation

a) Enter first Sequence [1 2 1 1]

Enter second Sequence [1 1 1 1]

The convoluted sequence is:

1 3 4 5 4 2 1

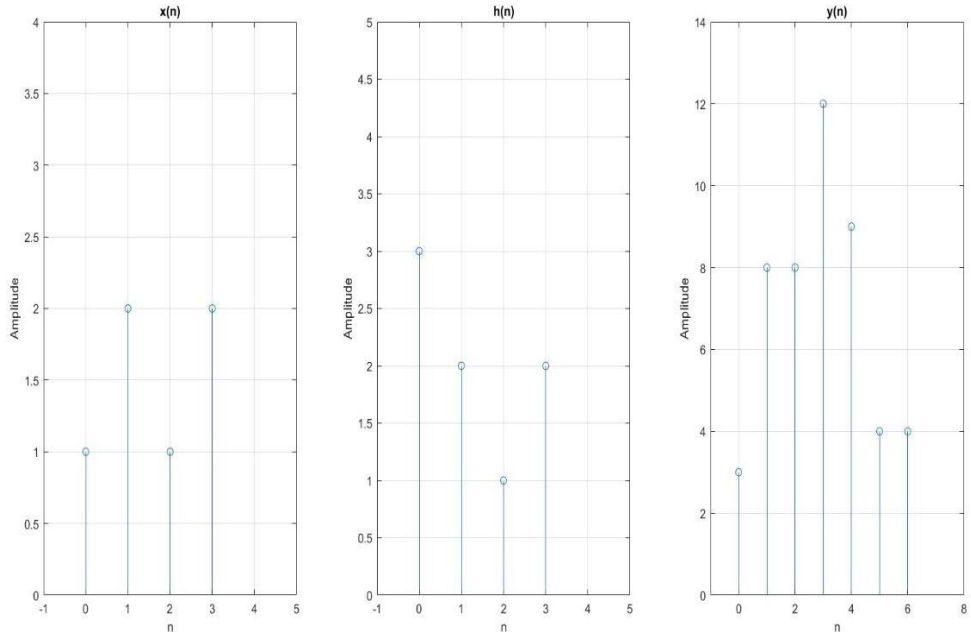


b) Enter first Sequence [1 2 1 2]

Enter second Sequence [3 2 1 2]

The convoluted sequence is:

3 8 8 12 9 4 4



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

Experiment Name: Circular Convolution

Aim:

To find circular convolution

a. Using FFT and IFFT.
b. Using Concentric Circle Method.
c. Using Matrix Method.

Theory:

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

Mathematical Definition:

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

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

Applications:

- Discrete-Time Filtering: Circular convolution is used for filtering discrete-time signals.
- Digital Signal Processing: It's a fundamental operation in many digital signal processing algorithms.
- Cyclic Convolution: In certain applications, such as cyclic prefix OFDM, circular convolution is used to simplify the implementation of linear convolution.

Program:

a. Using FFT and IFFT.

```
clc;

close all; c

lear all;


x1 = [1 2 1 2];

x2 = [1 2 3 4];

X1_k = fft(x1);

X2_k = fft(x2);
```

```
Y1_k = X1_k.*X2_k;
y1 =ifft(Y1_k);
disp("Using FFT and IFFT:")
disp(y1);
```

b. Using Concentric Circle Method.

```
clc;
close all;
clear all;

x = [1 2 1 2];
h = [1 2 3 4];
N = max(length(x),length(h));
y = zeros(1,N);
for n=1:N
  h_s = circshift(h,n-1); %shifting h(n) by 1 unit
  y(n) = sum(x.*h_s);
end

disp("Using Concentric Circle Method:");
disp(y);
```

c. Using Matrix Method.

```
clc;
clear all;
close all;
```

```
 x = [1 2 1 2];
 h = [1 2 3 4];
N = max(length(x),length(h));
h_n = zeros(N,N);
for n=1:N h_s = circshift(h,n-1); %shifting h(n) by 1 unit
h_n(:,n) = h_s;
end
y = h_n *x';
disp("Using Concentric Circle Method:")
disp(y');
```

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

Using FFT and IFFT:

16 14 16 14

b) USING Concentric Circle Method

Using Concentric Circle Method:

16 14 16 14

c) USING Matrix Method

Using Matrix Method.:

16 14 16 14

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

Experiment Name: Linear Convolution using CircularConvolution 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.

Example:

Consider the sequences *x[n] = [1, 2, 3]* and *h[n] = [4, 5]*.

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

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

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.

Example:

Using the same sequences as before, *x[n] = [1, 2, 3]* and *h[n] = [4, 5]*.

1. Zero-padding:
    - Pad *x[n]* to [1, 2, 3, 0, 0] and *h[n]* to [4, 5, 0, 0].
2. Linear Convolution:
    - Perform linear convolution. The result will be [4, 13, 21, 15, 0].
3. Modulus Operation:
    - Apply the modulus operation to the indices: [4, 13, 21, 15, 0] becomes [4, 13, 2, 15, 0].

Program:

1. Linear Convolution using Circular Convolution

```
clc;

clear all;

close all;

x = [1 2 3 4];

h = [1 1 1 ];

l = length(x);

m = length(h);

k = l+m-1;

x = [x zeros(1,k-l)];

h = [h zeros(1,k-m)];
```

```
X_k = fft(x);
H_k = fft(h);
Y_k = X_k.*H_k;
y = ifft(Y_k);
disp("Linear Convolution using Circular Convolution :");
disp(y);
```

2.Circular convolution using Linear Convolution

```
clc;

close all;

clear all;

x = [1 2 3 4];
h = [1 1 1 ];
l = length(x);
m = length(h);
Lc = max(l,m);
Ll= l+m-1;
y = conv(x,h);
for i=1:Ll-Lc
  y(i) = y(i) + y(Lc+i);
end
for i=1:Lc
y1(i) = y(i);
end
disp("Circular convolution using Linear Convolution:")
disp(y1);
```

Result

Performed

a) Linear Convolution using Circular Convolution;

b) Circular Convolution usingLinear Convolution

and verified result.

Observation

1) Linear Convolution using Circular Convolution :

Linear Convolution using Circular Convolution:

1 3 6 9 7 4

2.Circular convolution using Linear Convolution

Circular convolution using Linear Convolution:

8 7 6 9

Experiment No: 6 Date: 01/10/24

DFT AND IDFT

Aim:

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

2. IDFT using inbuilt function and without using inbuilt function.

Theory:

Discrete Fourier Transform (DFT)

The **Discrete Fourier Transform (DFT)** is a mathematical transformation used to analyze the frequency content of discrete signals. For a sequence x[n] of length N, the DFT is defined as:

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

- X[k] is the DFT of the sequence x[n].
- The exponential factor represents $\boldsymbol{e^{-j\frac{2\pi}{N}nk}}$ the complex sinusoidal basis functions.
- The DFT maps the time-domain signal into the frequency domain.

Inverse Discrete Fourier Transform (IDFT)Method:

The **Inverse Discrete Fourier Transform (IDFT)** is used to convert a frequency-domain sequence X[k] back into its time-domain sequence x[n]. The IDFT is defined as:

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

- The IDFT takes the frequency components X[k] and reconstructs the original sequence x[n].
- The exponential factor $\boldsymbol{e^{j\frac{2\pi}{N}nk}}$ is the inverse of the DFT's complex sinusoidal basis functions

.

Application

- Spectrum (Analysis)
- Filtering
- Compression
- Modulation
- Convolution
- Demodulation

- Equalization
- Restoration
- Detection
- Estimation

Program:

1. Discrete Fourier Transform (DFT)

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

k=0:N-1;
magX=abs(X);
```

```
phaseX=angle(X);
subplot(2,1,1);
stem(k,magX);
title("Magnitude Plot");
hold on;
plot(k,magX);
subplot(2,1,2);
stem(k,phaseX);
hold on;
title("Phase Plot");
plot(k,phaseX);
```

2.IDFT

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

```
disp(round(x));
disp('ifft');
disp(ifft(X));
```

Result:

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.

Observation:

1.DFT

enter sequence:[1 1 1 0]

enter the N point:8

X

Columns 1 through 7

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

Column 8

1.7071 + 1.7071i

round(X)

Columns 1 through 7

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

Column 8

2.0000 + 2.0000i

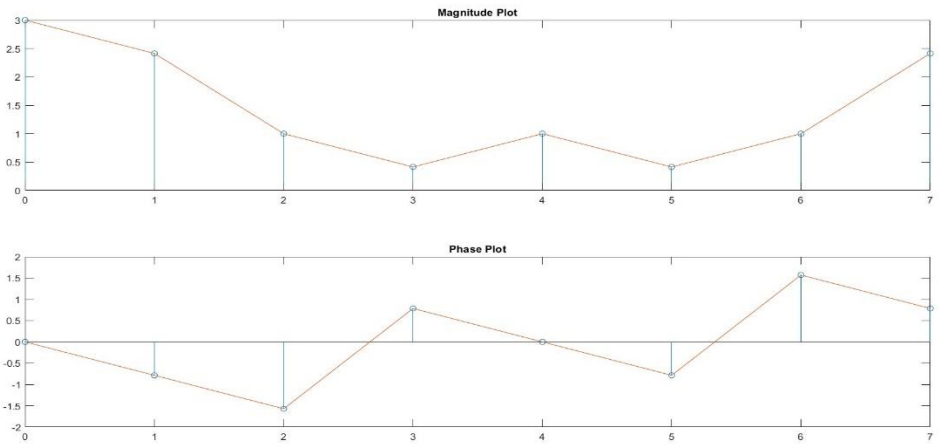
fft

Columns 1 through 7

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

Column 8

1.7071 + 1.7071i



2.IDFT

```
enter sequence: [3.0000 + 0.0000i   1.7071 - 1.7071i   0.0000 - 1.0000i   0.2929 + 0.2929i
1.0000 + 0.0000i   0.2929 - 0.2929i   0.0000 + 1.0000i]
enter the n point:8
x
   0.7866 - 0.2134i
   0.6982 - 0.0000i
   0.7866 + 0.2134i
  -0.0000 + 0.3018i
   0.2134 + 0.2134i
   0.3018 - 0.0000i
   0.2134 - 0.2134i
   0.0000 - 0.3018i

round(x)
     1
     1
     1
     0
     0
     0
     0
     0

ifft
  Columns 1 through 7

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

  Column 8

   0.0000 - 0.3018i
```

Experiment No: 7 Date: 01/10/24

Properties of DFT

Aim:

Verify following properties of DFT using Matlab/Scilab.

1.Linearity Property

2.Parsevals Theorem

3.Convolution Property

4.Multiplication Property

Theory:

1. Linearity Property

The linearity property of the DFT states that if you have two sequences x1[n] and x2[n], and their corresponding DFTs are X1[k] and X2[k], then for any scalar a and b:

$$\mathbf{DFT\{a\cdot x1[n]+b\cdot x2[n]\} = a\cdot DFT\{x1[n]\}+b\cdot DFT\{x2[n]\}}$$

2. Parseval's Theorem

Parseval's theorem states that the total energy of a signal in the time domain is equal to the total energy in the frequency domain. For a sequence x[n] and its DFT X[k]:

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

3.Convolution Property

The convolution property of the DFT states that the circular convolution of two sequences in the time domain is equivalent to the element-wise multiplication of their DFTs in the frequency domain:

$$\mathbf{DFT\{x1[n]\circledast x2[n]\}=DFT\{x1[n]\}\cdot DFT\{x2[n]\}}$$

4. Multiplication Property

The multiplication property of DFT states that pointwise multiplication in the time domain corresponds to circular convolution in the frequency domain:

$$\mathbf{DFT\{x1[n]\cdot x2[n]\}} = \frac{1}{N}\mathbf{DFT\{x1[n]\}\circledast DFT\{x2[n]\}}$$

Program:

1. Linearity Property

```
clc;
```

```
clear all;
close all;
x=input("enter first sequence");
h=input("enter sequence sequence:");
lx=length(x);
lh=length(h);
if lx>lh
    h=[h zeros(1,lx-lh)]
else
    x=[x zeros(1,lh-lx)]
end
a=input("enter value of 'a':");
b=input("enter value of 'b':");
lhs=fft((a.*x)+(b.*h));
rhs=a.*fft(x)+b.*fft(h);
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp('Linearity property verified');
else
    disp('Linearity property not verified');
end
```

2. Parseval's Theorem

```
clc;
clear all;
close all;
x=input("enter first sequence:");
```

```
h=input("enter second sequence:");
N=max(length(x),length(h));
xn=[x zeros(1,N-length(x))];
hn=[h zeros(1,N-length(h))];
lhs=sum(xn.*conj(hn));
rhs=sum(fft(xn).*conj(fft(hn)))/N;
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp("Parseval's Theorem verified");
else
    disp("Parseval's Theorem not verified");
end
```

3.Convolution Property

```
clc;
clear all;
close all;
x=input("enter first sequence");
h=input("enter sequence sequence:");
N=max(length(x), length(h));
xn=[x zeros(N-length(x))];
hn=[h zeros(N-length(h))];
Xn=fft(xn);
Hn=fft(hn);
lhs=cconv(xn,hn,N);
rhs=ifft(Xn.*Hn);
disp('LHS');
```

```
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp('Circular Convolution verified')
else
    disp('Circular Convolution not verified');
end
```

4. Multiplication Property

```
clc;
clear all;
close all;
x=input("enter first sequence");
h=input("enter sequence sequence:");
N=max(length(x), length(h));
xn=[x zeros(N-length(x))];
hn=[h zeros(N-length(h))];
lhs=fft(xn.*hn);
Xn=fft(xn);
Hn=fft(hn);
rhs=(cconv(Xn,Hn,N))/N;
disp('LHS');
disp(lhs);
disp('RHS');
disp(rhs);
if lhs==rhs
    disp('Multiplication property verified');
else
    disp('Multiplication property not verified');
end
```

Result:

Performed and verified the following properties of DFT:

1.Linearity Property

2.Parsevals Theorem

3.Convolution Property

4.Multiplication Property

Observation:

1. Linearity Property

enter first sequence[1 2 3 4]

enter sequence sequence:[1 1 1 1]

x =

1 2 3 4

enter value of 'a':2

enter value of 'b':3

LHS

32.0000 + 0.0000i -4.0000 + 4.0000i -4.0000 + 0.0000i -4.0000 - 4.0000i

RHS

32.0000 + 0.0000i -4.0000 + 4.0000i -4.0000 + 0.0000i -4.0000 - 4.0000i

Linearity property verified

2. Parseval's Theorem

enter first sequence:[1 2 3 4]

enter second sequence:[1 1 1 1]

LHS

10

RHS

10

Parseval's Theorem verified

3.Convolution Property

enter first sequence[1 2 3 4]

enter sequence sequence:[1 1 1 1]

LHS

10 10 10 10

RHS

10 10 10 10

Circular Convolution verified

4.Multiplication Property

enter first sequence[1 2 3 4]

enter sequence sequence:[1 1 1 1]

LHS

Columns 1 through 3

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

Column 4

-2.0000 - 2.0000i

RHS

Columns 1 through 3

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

Column 4

-2.0000 - 2.0000i

Multiplication property verified

Experiment No: 8 Date: 08/10/24

OVERLAP ADD AND OVERLAP SAVE METHOD

Aim:

Implement overlap add and overlap save method using Matlab/Scilab.

Theory:

Both the Overlap-Save and Overlap-Add methods are techniques used to compute the convolution of long signals using the Fast Fourier Transform (FFT). The direct convolution of two signals, especially when they are long, can be computationally expensive. These methods allow us to break the signals into smaller blocks and use the FFT to perform the convolution more efficiently.

Overlap-Save Method

The Overlap-Save method deals with circular convolution by discarding the parts of the signal that are corrupted by wrap-around effects. Here's how it works:

1. Block Decomposition: The input signal is divided into overlapping blocks. If the filter has length and we use blocks of length, the overlap is samples, so each block has new samples and samples from the previous block.

2. FFT and Convolution: Each block is convolved with the filter using FFT. However, because of circular convolution, the result contains artifacts due to the overlap.

3. Discard and Save: We discard the first samples from each block (the part affected by the wrap-around) and save the remaining samples. This gives us the correct linear convolution.

Overlap-Add Method

The Overlap-Add method, on the other hand, handles circular convolution by adding overlapping sections of the convolved blocks. Here’s how it works:

1. Block Decomposition: The input signal is split into non-overlapping blocks of size. Each block is then zero-padded to a size of , where is the length of the filter.

2. FFT and Convolution: Each block is convolved with the filter using FFT. Since the blocks are zero-padded, the convolution produces valid linear results, but the output blocks overlap.

3. Overlap and Add: After convolution, the results of each block overlap by samples. These overlapping regions are added together to form the final output.

Program:

1. Overlap Add

```
clc;
clear all;
close all;

x = input('Enter the input sequence x : ');
h = input('Enter the impulse response h : ');
L = length(h);  % Length of impulse response
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;  % Calculate number of sections
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);
% Display results
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;
```

2.Overlap Save

```
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);  % Length of input sequence x
    M = length(h);   % Length of filter sequence h
    M1 = M - 1;      % Length of overlap
    L = N - M1;      % Length of non-overlapping part
    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
```

Result:

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

Observation:

1. Overlap Add

Enter the input sequence x : [3 -1 0 1 3 2 0 1 2 1]

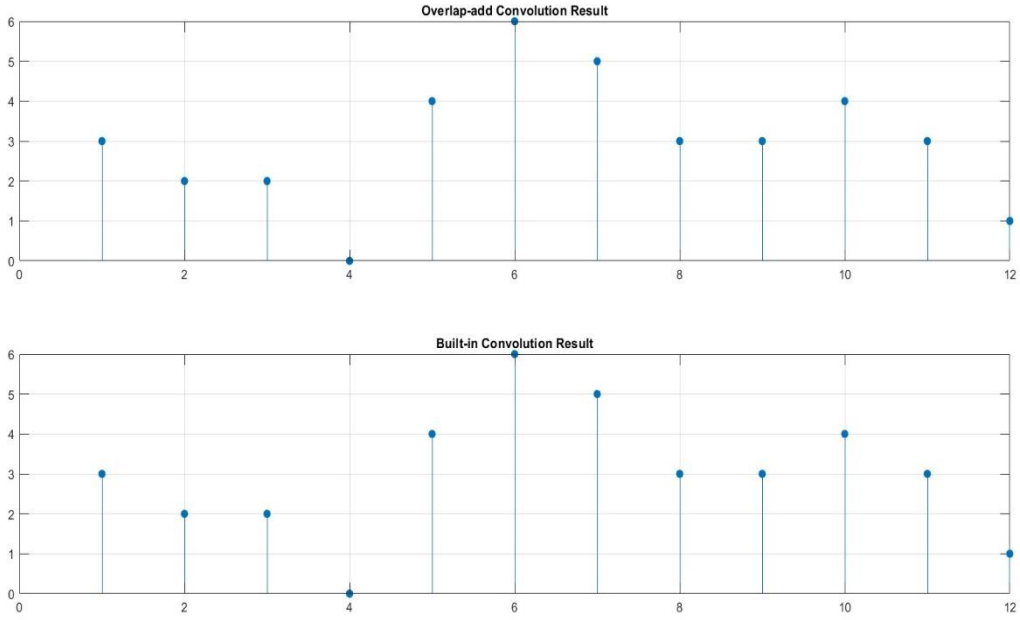
Enter the impulse response h : [1 1 1]

Overlap-add convolution result:

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

Built-in convolution result:

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



2.Overlap Save

Enter 1st sequence: [3 -1 0 1 3 2 0 1 2 1]

Enter 2nd sequence: [1 1 1]

Fragmented block size: 3

Using Overlap and Save method

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

Verification

3.0000 2.0000 2.0000 0 4.0000 6.0000 5.0000 3.0000 3.0000 4.0000 3.0000 1.0000

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

AIM

To familiarize with the input and output ports of dsp board.

THEORY

TMS 320C674x DSP CPU

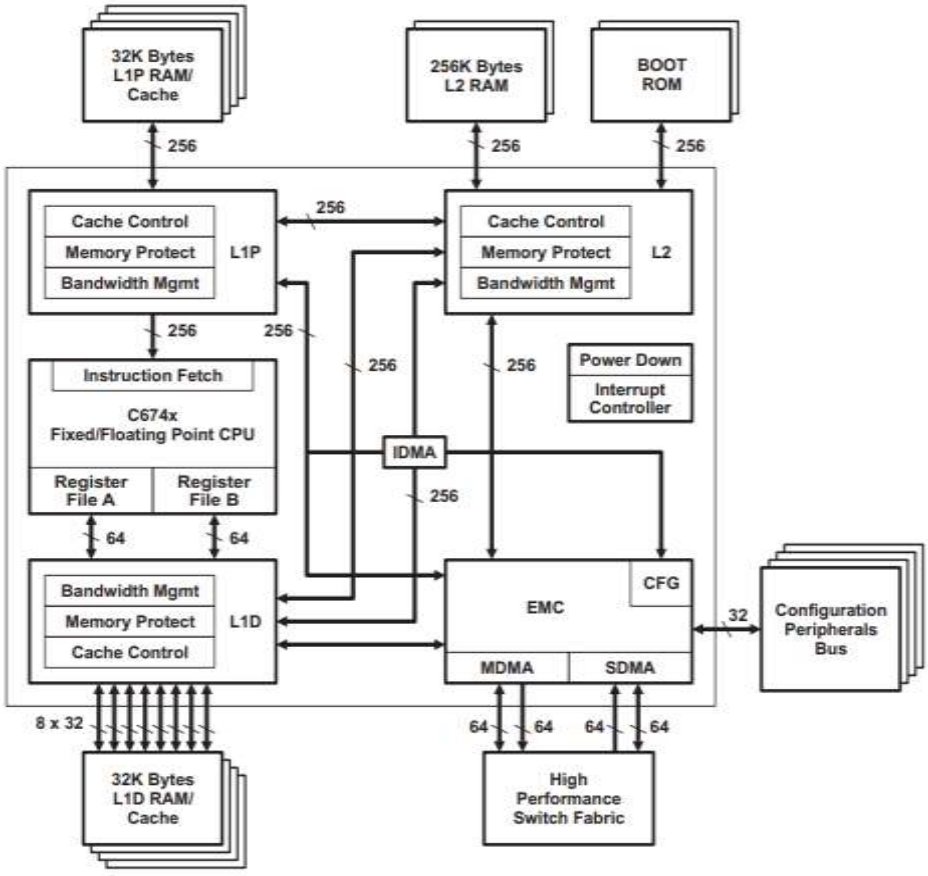


FIGURE: TMS320C 674X DSP CPU BLOCK DIAGRAM

The TMS320C674X DSP CPU consists of eight functional units, two register files, and two data paths as shown in Figure. The two general-purpose register files (A and B) each contain 32 32- bit registers for a total of 64 registers. The general-purpose registers can be used for data or can be data address pointers. The data types supported include packed 8-bit data, packed 16-bit data, 32-bit data, 40- bit data, and 64-bit data. Values larger than 32 bits, such as

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

Multichannel Audio Serial Port (McASP):

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

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

• Up to sixteen transmit or receive data pins and serializers

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

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

• Extensive error checking and mute generation logic

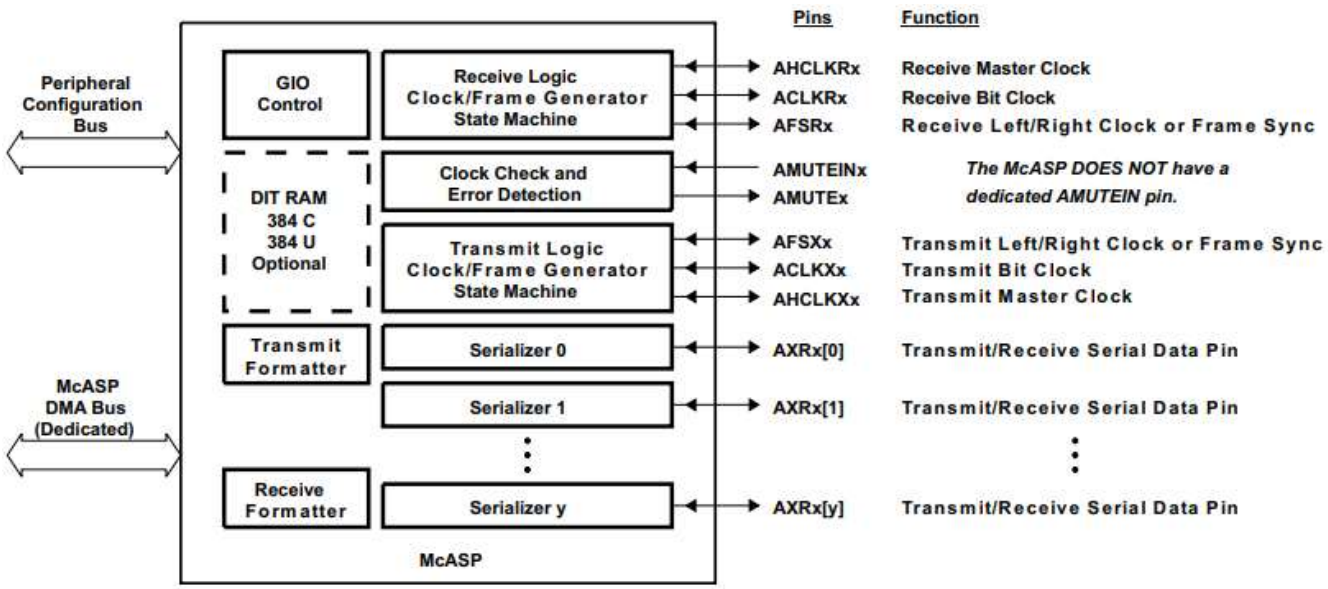
• All unused pins GPIO-capable

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

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

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

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



RESULT

Familiarized the input and output ports of dsp board.

TMS320C6748 DSP Development Kit

Low-cost development kit to jump-start real-time signal processing innovation

Texas Instruments' TMS320C6748 DSP development kit is a new, robust low-cost development board designed to spark innovative designs based on the C6748 processor. Along with TI's new included C6748 SYS/BIOS™ Software Development Kit (SDK), the C6748 development kit is ideal for real-time analytics applications, such as fingerprint recognition and face detection. It includes the C6748 baseboard, SD cards with two demos, BIOS and SDK, and Code Composer Studio™ (CCStudio) Integrated Development Environment, a power supply and cord, VGA cable and USB cable.

Key features and benefits

- TMS320C6748 DSP software and development kit to jump-start real-time signal processing innovation for biometric analytics applications, audio and more
- Reduces design work with downloadable and duplicable board schematics and design files
- Fast and easy development of applications requiring fingerprint recognition and face detection with embedded analytics
- Low-power TMS320C6748 applications processor
- Scalable platform enables a variety of performance, power, peripheral and price options
- 128-MByte DDR2 SDRAM
- 128-MByte NAND Flash memory
- Micro SD/MMC slot
- USB and SD connectors
- Wide variety of peripheral interfaces
- Line in, headphone out, MIC-in ports
- Expansion connectors
- Includes Code Composer Studio IDE 4.0
- Full documentation on CD-ROM

Technical details

The C6748 development kit includes everything needed to start demonstrating applications in less than 10 minutes and to begin writing code in less than an hour.

The development kit is based on the TMS320C6748, a low-power dual-core applications processor based on a fixed-point C64x+™ instruction set and the floating-point C67x+™ instruction set. It provides significantly lower power than other members of the TMS320C6000™ platform of DSPs and provides both floating-point precision and fixed-point performance in the same device.

With a wide variety of standard interfaces for connectivity and storage, the C6748 development kit enables developers to easily bring audio, video and other signals onto the board. Expansion headers allow customers to extend the functionality of the kit to include a camera sensor from Leopard Imaging or an LCD screen. Included interfaces are:

- USB serial port
- Fast Ethernet port (10/100 Mbps)
- USB host port (USB 1.1)
- USB OTG port (USB 2.0)
- SATA port (3 Gbps)
- VGA port (15-pin D-SUB)
- LCD port (Beagleboard-XM connectors)
- 3 audio ports
  - 1 line in
  - 1 line out
  - 1 MIC in
- Composite in (RCA jack)
- Leopard Imaging camera sensor input (32-pin ZIP connector)
- Authentic fingerprint sensor

Easy to write and optimize DSP code

Designers can readily target the C6748 DSP through TI's robust and comprehensive Code Composer Studio (CCStudio) Integrated Development Environment (IDE). CCStudio IDE includes an efficient optimizing C/C++ compiler assembler, linker, debugger; integrated CodeWright editor with CodeSense technology for faster code creation; data visualization; a profiler and a flexible project manager. CCStudio IDE also includes a DSP/BIOS™ real-time kernel and Chip Support Library.

TI's new C6748 SYS/BIOS SDK is included on a SD card with the kit. The SDK includes several demonstrations for biometric analytics applications and also includes the latest SYS/BIOS real-time kernel, C6748 StarterWare software package and code generation tools. Designers can begin writing code in less than one hour with the latest tool chain GCC 4.5 and the latest TI DSP software components (SYS/BIOS and SysLink).

StarterWare provides a C-based OS-independent platform support for the C6748

DSP platform. It provides device abstraction layer libraries, peripheral programming examples such as Ethernet, graphics and USB, and board-level example applications. StarterWare can be used stand-alone or with a real-time operating system (RTOS).

Out-of-box demos in less than 10 minutes

Included in the C6748 development kit is all the hardware and software needed for two demonstrations, a fingerprint-recognition demo and a face-detection demo. The fingerprint recognition demo uses the included fingerprint sensor (swipe-based) from Authentec and allows multiple users to be enrolled. The demo delivers 100% accuracy of results in less than 300 msec to match the fingerprint on the C674x floating-point core.

The included face-detection demo is a frontal face-detection open-source algorithm that allows multiple face detections in a frame. The face detection demo supports D1 (720×480) resolution image processing with 0.5 frames per second face detection. With the two included demos, designers can go from box to demo in less than 10 minutes.

Simple hardware development and software compatibility

TI helps reduce design work with free downloadable and duplicable board schematics and design files following TI's proven design rules. For designs needing only DSP performance, designers can scale between the software and pin-to-pin compatible TMS320C6748/6/2 DSPs as well as other software-compatible TMS320C6000™ DSPs available at a variety of performance, power, peripheral and price options. Designers can also select the ideal combination of ARM and DSP performance needed for any design with the software and pin-compatible OMAP-L138/2 DSP+ARM9™ processors to add high-level operating systems such as Linux™.

The C6748 development kit is supported by TI's online community **e2e.ti.com**. Complete collateral, CCStudio IDE drivers, Chip Support Library (CSL) and all the required production-quality documentation for the C6748 kit is available today. Complete schematics and layout files are available for the tool so customers can use this as a reference for their own system development.

TI's extensive Developer Network, as well as a complete Chip Support Library, comprehensive application notes, reference designs, application guides, videos and online communities help designers develop new products based on the C6748 DSP with confidence and ease.

Get started today

The robust, low-cost C6748 development kit (part number: TMDXLCDK6748) is available now for the low cost of U.S. $195. Pricing includes the TMS320C6748 baseboard as well as the industry-leading CCStudio IDE v.4, StarterWare software package, demo software and BIOS and Linux SDK – everything you need to run demos in less than 10 minutes.
www.ti.com/c6748lcdk

TI Worldwide Technical Support

Internet

TI Semiconductor Product Information Center Home Page
support.ti.com
TI E2E™ Community Home Page
e2e.ti.com

Product Information Centers

Americas	Phone	+1(972) 644-5580
Brazil	Phone	0800-891-2616
Mexico	Phone	0800-670-7544
	Fax	+1(972) 927-6377
	Internet/E-mail	support.ti.com/sc/pic/americas.htm

Europe, Middle East, and Africa
Phone

European Free Call	00800-ASK-TEXAS (00800 275 83927)
International	+49 (0) 8161 80 2121
Russian Support	+7 (4) 95 98 10 701

Note: The European Free Call (Toll Free) number is not active in all countries. If you have technical difficulty calling the free call number, please use the international number above.

Fax	+(49) (0) 8161 80 2045
Internet	www.ti.com/asktexas
Direct E-mail	asktexas@ti.com

Japan

Phone	Domestic	0120-92-3326
Fax	International	+81-3-3344-5317
	Domestic	0120-81-0036
Internet/E-mail	International	support.ti.com/sc/pic/japan.htm
	Domestic	www.tij.co.jp/pic

Asia
Phone

International	+91-80-41381665
Domestic	Toll-Free Number

Note: Toll-free numbers do not support mobile and IP phones.

Australia	1-800-999-084
China	800-820-8682
Hong Kong	800-96-5941
India	1-800-425-7888
Indonesia	001-803-8861-1006
Korea	080-551-2804
Malaysia	1-800-80-3973
New Zealand	0800-446-934
Philippines	1-800-765-7404
Singapore	800-886-1028
Taiwan	0800-006800
Thailand	001-800-886-0010

Fax	+8621-23073686
E-mail	tiasia@ti.com or ti-china@ti.com
Internet	support.ti.com/sc/pic/asia.htm

B011012

Printed in U.S.A. by (Printer, City, State)

TEXAS INSTRUMENTS

SPRT633

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Audio	www.ti.com/audio	Automotive and Transportation	www.ti.com/automotive
Amplifiers	amplifier.ti.com	Communications and Telecom	www.ti.com/communications
Data Converters	dataconverter.ti.com	Computers and Peripherals	www.ti.com/computers
DLP® Products	www.dlp.com	Consumer Electronics	www.ti.com/consumer-apps
DSP	dsp.ti.com	Energy and Lighting	www.ti.com/energy
Clocks and Timers	www.ti.com/clocks	Industrial	www.ti.com/industrial
Interface	interface.ti.com	Medical	www.ti.com/medical
Logic	logic.ti.com	Security	www.ti.com/security
Power Mgmt	power.ti.com	Space, Avionics and Defense	www.ti.com/space-avionics-defense
Microcontrollers	microcontroller.ti.com	Video and Imaging	www.ti.com/video
RFID	www.ti-rfid.com		
OMAP Mobile Processors	www.ti.com/omap		
Wireless Connectivity	www.ti.com/wirelessconnectivity		
	TI E2E Community Home Page		e2e.ti.com

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265

```
clc;
clear all;
close all;
wc=0.5*pi;

N = input('Enter the value of N=');
alpha = (N-1)/2;
eps = 0.001;
n = 0:1:N-1;
hd = sin(wc*(n-alpha+eps))./(pi*(n-alpha+eps));

wr = boxcar(N);
wh=hamming(N);
wn=hanning(N);
wt=bartlett(N);
hn = hd.*wr';
hn1=hd.*wh';
hn2=hd.*wn';
hn3=hd.*wt';
w = 0:0.01:pi;
h = freqz(hn,1,w);
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
h3=freqz(hn3,1,w);
subplot(4,2,1);
plot(w/pi,10*log10(abs(h)));
title('low pass filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,3);
plot(w/pi,10*log10(abs(h1)));
title('low pass filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,4);
stem(wh);
title('Hamming window Sequence');
xlabel('No. of Samples');
```

```
ylabel('Amplitude');
subplot(4,2,5);
plot(w/pi,10*log10(abs(h2)));
title('low pass filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,6);
stem(wn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,7);
plot(w/pi,10*log10(abs(h2)));
title('low pass filter using bartlett window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,8);
stem(wt);
title('bartlett window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

```
clc;
clear all;
close all;
wc=0.9*pi;
eps=0.001;
N = input('Enter the value of N=');
alpha = (N-1)/2;

n = 0:1:N-1;
hd = (sin(pi*(n-alpha+eps))-sin(wc*(n-alpha+eps)))./(pi*(n-alpha+eps));
wr = boxcar(N);
wh=hamming(N);
```

```
wn=hanning(N);
wt=bartlett(N);
hn = hd.*wr';
hn1=hd.*wh';
hn2=hd.*wn';
hn3=hd.*wt';
w = 0:0.01:pi;
h = freqz(hn,1,w);
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
h3=freqz(hn3,1,w);
subplot(4,2,1);
plot(w/pi,10*log10(abs(h)));
title('low pass filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,3);
plot(w/pi,10*log10(abs(h1)));
title('low pass filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,4);
stem(wh);
title('Hamming window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,5);
plot(w/pi,10*log10(abs(h2)));
title('low pass filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,6);
stem(wn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,7);
plot(w/pi,10*log10(abs(h2)));
```

```
title('low pass filter using bartlett window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,8);
stem(wt);
title('bartlett window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

```
clc;
clear all;
close all;
wc1=0.5*pi;
wc2=0.9*pi;
eps=0.001;
N = input('Enter the value of N=');
alpha = (N-1)/2;

n = 0:1:N-1;
hd = (sin(wc1*(n-alpha+eps))-sin(wc2*(n-alpha+eps))+sin(pi*(n-alpha)))./(pi*(n-alpha+eps));
wr = boxcar(N);
wh=hamming(N);
wn=hanning(N);
wt=bartlett(N);
hn = hd.*wr';
hn1=hd.*wh';
hn2=hd.*wn';
hn3=hd.*wt';
w = 0:0.01:pi;
h = freqz(hn,1,w);
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
h3=freqz(hn3,1,w);
subplot(4,2,1);
```

```
plot(w/pi,10*log10(abs(h)));
title('band stop filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,3);
plot(w/pi,10*log10(abs(h1)));
title('band stop filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,4);
stem(wh);
title('Hamming window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,5);
plot(w/pi,10*log10(abs(h2)));
title('band stop filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,6);
stem(wn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,7);
plot(w/pi,10*log10(abs(h2)));
title('bandstop filter using bartlett window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,8);
stem(wt);
title('bartlett window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

```
clc;
clear all;
close all;
wc1=0.5*pi;
wc2=0.9*pi;
eps=0.001;
N = input('Enter the value of N=');
alpha = (N-1)/2;

n = 0:1:N-1;
hd = (sin(wc2*(n-alpha+eps))-sin(wc1*(n-alpha+eps)))./(pi*(n-alpha+eps));
wr = boxcar(N);
wh=hamming(N);
wn=hanning(N);
wt=bartlett(N);
hn = hd.*wr';
hn1=hd.*wh';
hn2=hd.*wn';
hn3=hd.*wt';
w = 0:0.01:pi;
h = freqz(hn,1,w);
h1 = freqz(hn1,1,w);
h2 = freqz(hn2,1,w);
h3=freqz(hn3,1,w);
subplot(4,2,1);
plot(w/pi,10*log10(abs(h)));
title('band stop filter using rectangular window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,2);
stem(wr);
title('Rectangular window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,3);
plot(w/pi,10*log10(abs(h1)));
title('band stop filter using hamming window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
```

```
subplot(4,2,4);
stem(wh);
title('Hamming window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,5);
plot(w/pi,10*log10(abs(h2)));
title('band stop filter using hanning window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,6);
stem(wn);
title('Hanning window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
subplot(4,2,7);
plot(w/pi,10*log10(abs(h2)));
title('bandstop filter using bartlett window');
xlabel('Normalized frequency');
ylabel('Magnitude in dB');
subplot(4,2,8);
stem(wt);
title('bartlett window Sequence');
xlabel('No. of Samples');
ylabel('Amplitude');
```

Sine wave

```
#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);
}
```

```
//#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.
}
}
}
```