Vel Tech Rangarajan Dr. Sagunthala R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

Accredited by NAAC with the HIGHEST GRADE A++

Recognised as CATEGORY-1 DEEMED TO BE UNIVERSITY by the UGC University Grants Commission

School of Electrical and Communication

Department of Electronics and Communication Engineering

LABORATORY RECORD

Course Code: 10211EC303

Course Title: Signal Processing Laboratory

Name	**: …………………………………………………………..**
Registration Number:	**………………………………………………………..**
VTU Number	**: …………………………………………………………..**

Summer Semester 2024 – 2025

School of Electrical and Communication

Department of Electronics and Communication Engineering

LABORATORY RECORD

Course Code: 10211EC303

Course Title: Signal Processing Laboratory

Name	: …………………………………………………………..
Registration Number:	…………………………………………………………..
VTU Number	: …………………………………………………………..

Summer Semester 2024 – 2025

Vel Tech
Rangarajan Dr. Sagunthala
R&D Institute of Science and Technology
(Deemed to be University Estd. u/s 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Register No.										

Certified that this is the bonafide record of work done by **…………………………………** **…………………………** in the **SIGNAL PROCESSING** Laboratory during the academic year **2024-2025** VTU. No:.................................. Year 2024-2025 **SUMMER SEMESTER**

Programme: - B.Tech. **ELECTRONICS AND COMMUNICATION ENGINEERING**

Signature of Lab In-Charge with Date

Signature of Head of the Department with Seal

Submitted for the University Practical Examination held on ……………………………………………….

Signature of Internal Examiner

Signature of External Examiner

TABLE OF CONTENTS

S. No	Date	Name of the Experiments	Pg. No.	C.I.A Mark	Sign

EXP.NO. 1 **DATE:**

GENERATION OF CONTINUOUS AND DISCRETE TIME SIGNALS

AIM:

To generate the continuous time and discrete time signals using MATLAB.

SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

THEORY:

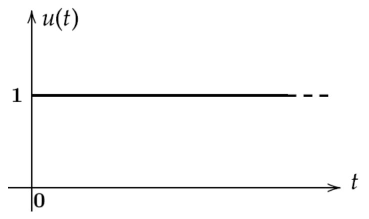
Continuous Time Elementary Signals

Unit Step Signal

The unit signal is mathematically defined as

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

The graphical representation of Unit Step signal is shown below



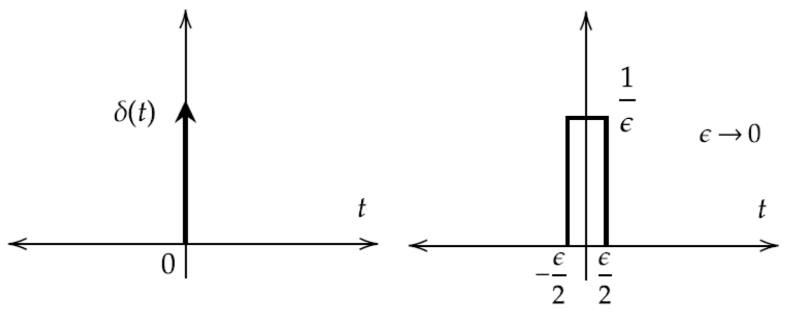
Unit Impulse Signal

The unit impulse function δ(t) is one of the most important functions in the study of signals and systems.

• This function was first defined in two parts by P. A. M. Dirac and is known as Dirac Delta Function.

• The Unit Impulse Signal is defined as

$$\delta(t) = 0\ for\ t \neq 0 \quad and \quad \int_{-\infty}^{\infty} \delta(t)dt = 1$$

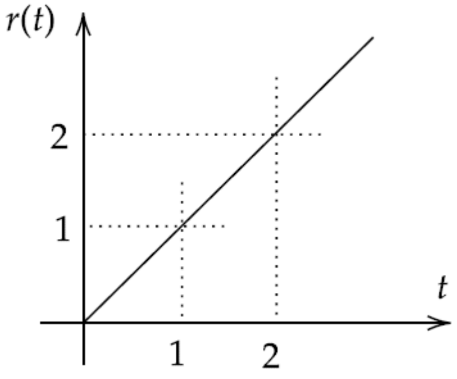


Unit Ramp Signal

The Ramp signal is defined as

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

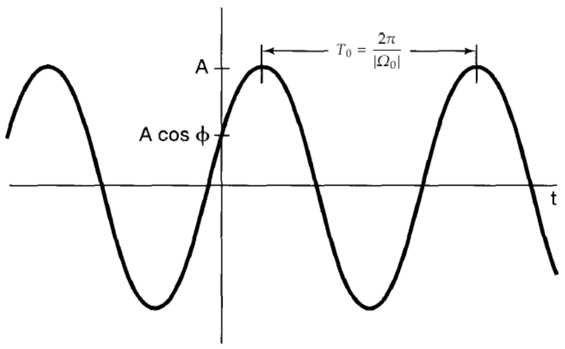
The Ramp signal is illustrated in the below diagram



Sinusoidal Signal

The sinusoidal signal can be represented mathematically as

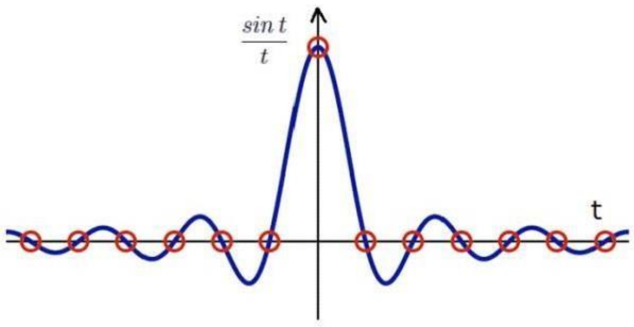
$$x(t) = A\ cos(\Omega_0 t + \phi)$$



Sinc Signal

The Sinc or Sampling Signal is mathematically defined as

$$sinc(t) = \frac{sin\ t}{t}\ for\ -\infty < t < \infty$$



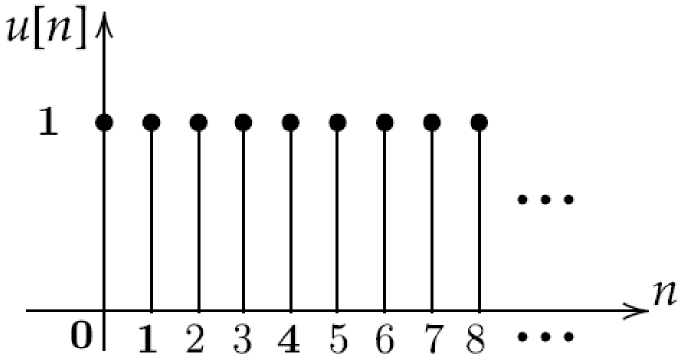
Discrete Time Elementary Signals

Unit Step Sequence

The mathematical model of discrete time Unit Step Sequence is given as

$$u[n] = \begin{cases} 1 & for\ n \geq 0 \\ 0 & otherwise \end{cases}$$

The unit step sequence is illustrated in the diagram given below

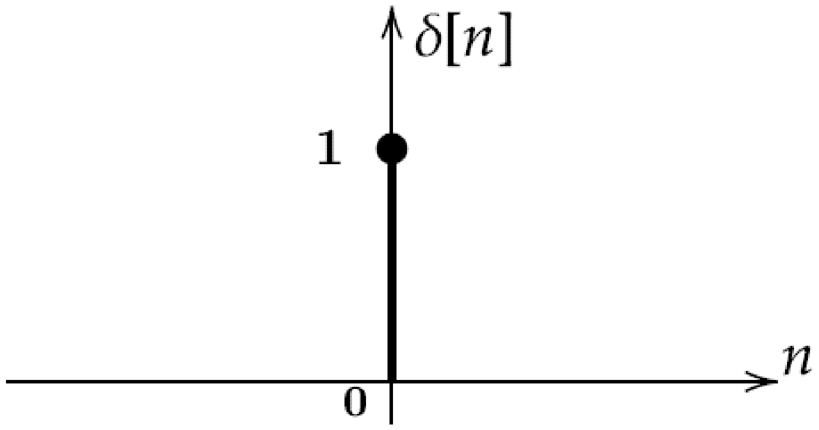


Unit Impulse Sequence

The discrete time unit impulse is given by

$$\delta[n] = \begin{cases} 0 & for\ n \neq 0 \\ 1 & for\ n = 0 \end{cases}$$

The unit impulse sequence is illustrated in the diagram given below

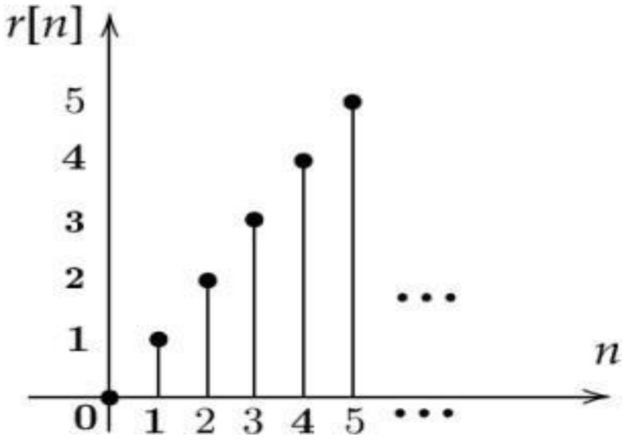


Unit Ramp Sequence

The mathematical model of discrete time Unit Step Sequence is given as

$$r[n] = \begin{cases} n & for\ n \geq 0 \\ 0 & otherwise \end{cases}$$

The unit step sequence is illustrated in the diagram given below

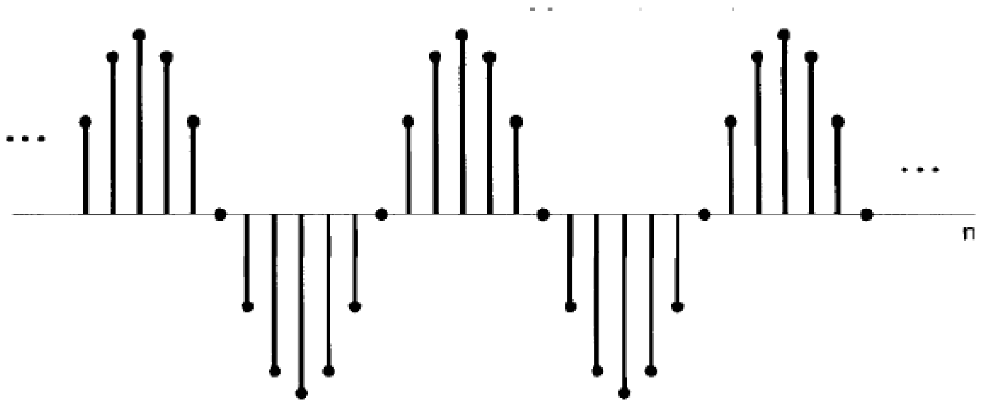


Sinusoidal Sequence

The sinusoidal signal given by

$$x[n] = A\,cos[\omega_0 n + \phi)]$$

The sinusoidal signal is illustrated below



<u>**FLOWCHART /ALGORITHM:**</u>

1. Start the MATLAB program
2. Get the number of samples or the duration of the signal as input
3. Assign the values t for each continuous time signal and n for each discrete time signal.
4. Assign the function for unit step, unit impulse, ramp, sinusoidal, exponential and square signal to generate the corresponding signal.
5. Use the subplot to display all the signals in a single figure window.
6. End the Program

PROGRAM:

Generation of Continuous time signals:

```
clc;
close all;
clear all;
N=input ('Enter the length of the sequence:');

% Unit Step
t=(0:N-1);
y=ones(1, N);
subplot(2,3,1);
plot(t,y);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Step function');

% Unit Impulse
t=[-2, -1, -0.01, 0, 0.01, 1, 2];
y=[0 0 0 1 0 0 0];
subplot(2,3,2);
plot(t,y);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Impulse function');

% Ramp Signal
t=(0:N-1);
y=t;
subplot(2,3,3);
plot(t,y);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Ramp function');

%Sine Function
fs=input('Enter the sampling frequency');
f=input('Enter frequency of signal');
ts=1/fs;
t=ts*(0:N-1);
y=sin(2*pi*f*t);
subplot(2,3,4);
plot(t,y);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Sine function');

% Square Function
t=(0:0.2:2*N);
```

```
y=square(t);
subplot(2,3,5);
plot(t,y);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Square function');

% Exponential Function
t=(0:0.2:N);
a=input('Enter a constant:');
y=exp(a*t);
subplot(2,3,6);
plot(t,y);
xlabel('Time (sec)');
ylabel('Amplitude');
title('Exponential function');
```

Generation of Discrete time signals:

```
clc;
close all;
clear all;
N=input('Enter the length of the sequence:');

% Unit Step Sequence
n=(0:N-1);
y=ones(1, N);
subplot(2,3,1);
stem(n,y);
xlabel('Time (samples)');
ylabel('Amplitude');
title('Step Sequence');

% Unit Impulse Sequence
n=[-2, -1, 0, 1, 2];
y=[0 0  1  0 0];
subplot(2,3,2);
stem(n,y);
xlabel('Time (samples)');
ylabel('Amplitude');
title('Impulse Sequence');

% Ramp Sequence
n=(0:N-1);
y=n;
subplot(2,3,3);
stem(n,y);
xlabel('Time (samples)');
ylabel('Amplitude');
```

```
title('Ramp Sequence');

% Sine Signal
fs=input('Enter the sampling frequency:');
f=input('Enter frequency of signal:');
ts=1/fs;
n=ts*(0:N-1);
y=sin(2*pi*f*n);
subplot(2,3,4);
stem(n,y);
xlabel('Time (samples)');
ylabel('Amplitude');
title('Sine Signal');

% Square Signal
n=(0:0.2:2*N);
y=square(n);
subplot(2,3,5);
stem(n,y);
xlabel('Time (samples)');
ylabel('Amplitude');
title('Square Signal');

% Exponential Sequence
n=(0:0.2:N);
a=input('Enter a constant:');
y=exp(a*n);
subplot(2,3,6);
stem(n,y);
xlabel('Time (samples)');
ylabel('Amplitude');
title('Exponential Sequence');
```

OUTPUT:

Continuous Time Signals:

```
Enter the length of the sequence: 20
Enter the sampling frequency: 500
Enter frequency of signal: 50
Enter a constant: -0.2
a = -0.2000
```

Discrete Time Signals:

```
Enter the length of the sequence: 20
Enter the sampling frequency: 200
Enter frequency of signal: 10
Enter a constant: -0.2
a = -0.2000
```

INFERENCE:

For generating a sinusoidal waveform, sampling frequency should be greater than twice the signal frequency. When the constant ‘a’ is positive in exponential signal it is a growing exponential. When it is a negative value, it is a decaying exponential.

RESULT:

VIVA QUESTIONS:

1. Define the term 'signal'.

- Signal is any physical quantity which varies with time and carries some information.

2. Distinguish continuous time and discrete time signals

- CT signals are those whose amplitude varies continuously with time
- DT signals are those whose amplitude varies discretely with time

3. Name the standard test signals

- Step, ramp, impulse, sinusoidal, exponential signals

4. What are the various operations that can be performed on time variable?

- Time scaling, time shifting, time reversal or folding

5. What are various operations that can be performed on amplitude?

- Amplitude scaling, amplitude reversal, addition, subtraction

6. State sampling theorem.

- Sampling frequency $f_s \geq 2f_m$
  where fm is the maximum signal frequency

EXP.NO. 2 **DATE:**

SAMPLING AND RECONSTRUCTION

AIM:

To perform sampling theorem under different sampling rates and reconstruct the same using MATLAB program.

SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

THEORY:

1. The sampling is the process of conversion of a continuous time signal into a discrete time signal.
2. The sampling is performed by taking samples of continuous time signal at definite intervals of time.
3. Usually, the time interval between two successive samples will be same and such type of sampling is called periodic or uniform sampling.
4. The time interval between successive samples is called sampling time (or sampling period or sampling interval), and it is denoted by T.
5. The unit of sampling period is second (s).
6. The inverse of sampling period is called sampling frequency (or sampling rate), and it is denoted by Fs.
7. The unit of sampling frequency is hertz (Hz).
8. A discrete signal obtained by sampling can be reconstructed to an analog signal, only when it is sampled without aliasing.

FLOWCHART /ALGORITHM:

1. Start the MATLAB program.
2. Get T value.
3. Calculate fm value.
4. Plot continuous time signal.
5. Get assign fs1, fs2, fs3 value.
6. Apply fs1, fs2, fs3 value and check it fs<2fm and fs=2fm and fs>2fm.
7. Plotted the waveform for fs<2fm, fs=2fm, fs>2fm.
8. Terminate the program.

PROGRAM:

```
%Program to verify the Sampling Theorem
clc;
clear all;
close all;

%Generate 15Hz sinusoidal signal of 0.1s duration sampled at 1000Hz
t=0:0.001:0.1;
fm=15;
x=sin(2*pi*fm*t);
figure(1)
plot(t,x);
xlabel('Time');
ylabel('Amplitude');
title('Original Analog signal');

%Analog signal sampled at fs<<2fm
fs=10;          %Sampling Frequency
n=0:1/fs:0.1;
xn=sin(2*pi*fm*n);
figure(2)
subplot(2,1,1);
stem(n,xn);
xlabel('Time');
ylabel('Amplitude');
title('Undersampled fs<<2fm signal');
subplot(2,1,2);
plot(n,xn);
xlabel('Time');
ylabel('Amplitude');
title('Reconstructed undersampled fs<<2fm signal');

%Analog signal sampled at Nyquist rate fs=2fm
fs=30;
n=0:1/fs:0.1;
xn=sin(2*pi*fm*n);
figure(3)
subplot(2,1,1);
stem(n,xn);
xlabel('Time');
ylabel('Amplitude');
title('Sampled at Nyquist rate fs=2fm signal'');
subplot(2,1,2);
plot(n,xn);
xlabel('Time');
ylabel('Amplitude');
title('Reconstructed Nyquist rate fs=2fm signal');
```

```
%Analog signal sampled at oversampling fs>>2fm
fs=500;
n=0:1/fs:0.1;
xn=sin(2*pi*fm*n);
figure(4)
subplot(2,1,1);
stem(n,xn);
xlabel('Time');
ylabel('Amplitude');
title('Oversampled fs>>2fm signal');
subplot(2,1,2);
plot(n,xn);
xlabel('Time');
ylabel('Amplitude');
title('Reconstructed oversampled fs>>2fm signal');
```

RESULT:

VIVA QUESTIONS:

1. **Define is sampling?**
   The sampling is the process of conversion of a continuous time signal into a discrete time signal.

2. **Define is aliasing?**
   Aliasing is the effect of new frequencies appearing in the sampled signal after reconstruction, which were not present in the original signal. It is caused by too low sample rate for sampling a particular signal or too high frequencies present in the signal for a particular sample rate.

3. **What is Nyquist rate?**
   The Nyquist rate, is a value (in units of samples per second or hertz, Hz) equal to twice the highest frequency (bandwidth) of a given function or signal.

4. **State sampling theorem.**
   The sampling theorem specifies the minimum-sampling rate at which a continuous-time signal needs to be uniformly sampled so that the original signal can be completely recovered or reconstructed by these samples alone. This is usually referred to as Shannon's sampling theorem in the literature.

5. **Mention the applications of sampling process?**
   Used in conversion of CT signal into DT signal. In ADC, the first step is sampling.

EXP.NO. 3 **DATE:**

LINEAR AND CIRCULAR CONVOLUTION

AIM:

To perform linear and circular convolution of the given signals using MATLAB program.

$x_1(t)$ =

$x_2(t)$ =

SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

THEORY:

Linear Convolution

- Linear convolution is a mathematical operation done to calculate the output of anyLinear-Time Invariant (LTI) system given its input and impulse response.
- It is applicable for both continuous and discrete-time signals.
- Linear Convolution can be represented as *y(n)=x(n)*h(n)*
  y(n) is the output (also known as convolution sum). x(n) is the input signal, and h(n) isthe impulse response of the LTI system.
- In linear convolution, both the sequences (input and impulse response) may or may not be of equal sizes.

For example, x(n)= [1,2,3] & h(n)= [1,2,3,4,5]

- The number of samples in the output of linear convolution is L = M + N – 1

Where M is the number of samples in x(n). N is the number of samples in h(n)

- It is possible to find the response of a filter using linear convolution.

Circular Convolution

- Circular convolution is essentially the same process as linear convolution. Just like linear convolution, it involves the operation of folding a sequence, shifting it, multiplying it with another sequence, and summing the resulting products
- Circular Convolution is represented as $y(n)=x(n)\oplus h(n)$
- In circular convolution, both the sequences (input and impulse response) must be of equal sizes. Circular convolution is possible only after modifying the signals via a method known as *zero padding.* In zero padding, zeroes are appended to the sequence that has a lesser size to make the sizes of the two sequences equal. If *x(n) = [1,2,3]*, after zero padding *x(n) = [1,2,3,0,0]*

FLOWCHART /ALGORITHM:

1. Start the MATLAB program.
2. Get the two input sequences to be convoluted.
3. Determine the length of the sequences.
4. Perform the necessary convolution operation using inbuilt function and matrix method.
   NOTE: If the lengths of the two sequences are different, then for circular convolution, zero padding must be performed.
5. Plot the input and output sequences.

PROGRAM:

LINEAR CONVOLUTION:

```
clear all;
close all;
x1=input('Enter the first sequence:');
x2=input('Enter the second sequence:');
figure;
subplot(2,2,1);
stem(x1);
xlabel('Time');
ylabel('Amplitude');
title('Input Sequence');
subplot(2,2,2);
stem(x2);
xlabel('Time');
ylabel('Amplitude');
title('Impulse Sequence');
x3=conv(x1, x2);
subplot(2,2,3);
stem(x3);
disp('Linear Convolution Output');
disp(x3);
xlabel('Time');
ylabel('Amplitude');
title('Output Sequence');
```

CIRCULAR CONVOLUTION:

```
clear all;
close all;
x1=input('Enter the first sequence:');
x2=input('Enter the second sequence:');
figure;
subplot(2,2,1);
stem(x1);
xlabel('Time');
ylabel('Amplitude');
title('First Input Sequence');
subplot(2,2,2);
stem(x2);
xlabel('Time');
ylabel('Amplitude');
title('Second Input Sequence');
N1=length(x1);
```

```
N2=length(x2);
N3=max(N1,N2);
x3=fft(x1,N3);
x4=fft(x2,N3);
x5=(x3.*x4);
x6=ifft(x5,N3);
subplot(2,2,3);
stem(x6);
disp('Circular Convolution Output');
disp(x6);
xlabel('Time');
ylabel('Amplitude');
title('Output Sequence');
```

RESULT:

VIVA QUESTIONS:

1. Define convolution.

Convolution is a mathematical operation used to express the relation between input and output of an LTI system. It relates input, output and impulse response of an LTI system as

$$y(t) = x(t) * h(t)$$

Where y (t) - output of LTI, x (t)- input of LTI, h (t) - impulse response of LTI

2. Define zero padding.

Zeroes are appended to the sequence that has a lesser size to make the sizes of the two sequences equal.

3. What are the properties of convolution?

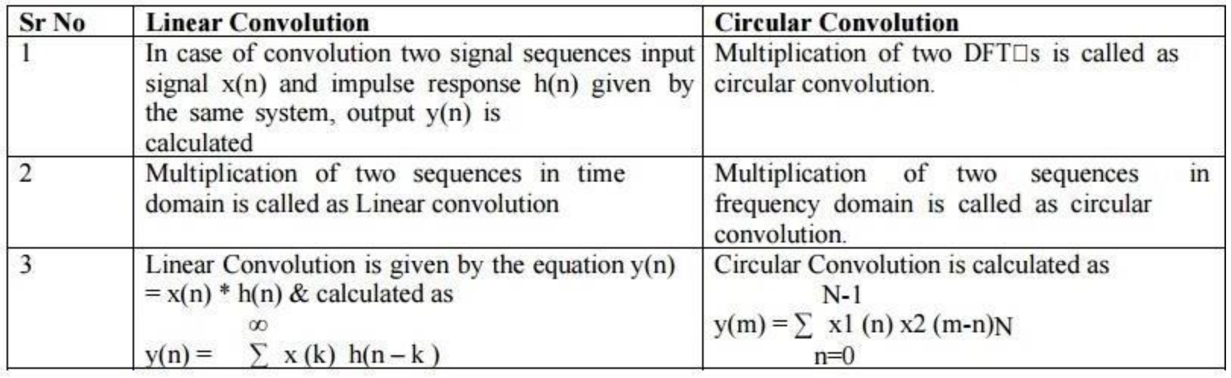
Commutative property $x(n) * h(n) = h(n) * x(n)$
Associative property $[x(n) * h1(n)]*h2(n) = x(n)*[h1(n) * h2(n)]$
Distributive property $x(n) *[h1(n)+h2(n)] = [x(n)*h1(n)]+[x(n) * h2(n)]$

4. What are the types of convolution?

There are two types of convolutions.

- Linear Convolution
- Circular Convolution

5. Compare Linear and Circular convolution



Sr No	Linear Convolution	Circular Convolution
1	In case of convolution two signal sequences input signal x(n) and impulse response h(n) given by the same system, output y(n) is calculated	Multiplication of two DFT□s is called as circular convolution.
2	Multiplication of two sequences in time domain is called as Linear convolution	Multiplication of two sequences in frequency domain is called as circular convolution.
3	Linear Convolution is given by the equation y(n) = x(n) * h(n) & calculated as $y(n) = \sum_{\infty} x(k)\ h(n-k)$	Circular Convolution is calculated as $y(m) = \sum_{n=0}^{N-1} x1(n)\, x2(m-n)_N$

EXP.NO. 4 **DATE:**

DISCRETE FOURIER TRANSFORM

AIM:

To write a MATLAB program to compute the N point Discrete Fourier Transform (DFT) of the given sequence and execute it.

SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

THEORY:

Discrete Fourier Transform

The **Discrete Fourier transform** (**DFT**) converts a finite sequence of equally- spaced samples of a function into a same-length sequence of equally-spaced samples of the discrete-time Fourier transform (DTFT), which is a complex-valued function of frequency. The interval at which the DTFT is sampled is the reciprocal of the duration of the input sequence. An inverse DFT is a Fourier series, using the DTFT samples as coefficients of complex sinusoids at the corresponding DTFT frequencies. It has the same sample-values as the original input sequence. The DFT is therefore said to be a frequency domain representation of the original input sequence. If the original sequence spans all the non-zero values of a function, its DTFT is continuous (and periodic), and the DFT provides discrete samples of one cycle. If the original sequence is one cycle of a periodic function, the DFT provides all the non-zero values of one DTFT cycle.

- The discrete Fourier transform, transforms a sequence of N complex numbers $\{x_n\}= x_0,x_1, x_2,\ldots x_{N-1}$ into another sequence of complex numbers, $\{X_K\}=X_0, X_1,\ldots\ldots X_{N-1}$which is defined by

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N}, k = 0,1,2,\ldots.N-1$$

- where x(n) denotes the input signal at time (sample) n, and X(k) denotes the kth spectral sample.
- The inverse DFT is defined by

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(K)e^{j2\pi nk/N}, n = 0,1,2,\ldots.N-1$$

- The number of complex multiplications involved in N-point DFT is N^2 and the number of complex additions are N(N-1). The N-point DFT can be computed using Fast FourierTransform algorithm namely Decimation in Time and Decimation in Frequency algorithm.
- The number of complex multiplications involved in N-point FFT is $\frac{N}{2}\log_2 N^2$ and the number of complex additions are $N\log_2 N$.

FLOWCHART /ALGORITHM:

1. Obtain the input sequence and number of points of the DFT sequence.
2. Determine the length of the input sequence using the **length ( )** function and check if the length is greater than the number of points. N must always be equal to or greater than the sequence.
3. Perform zero padding if the length of the sequence is less than the number of DFT points.
4. Based on the value of N obtained as input, create the **W_N** matrix using ‘For’ loop.
5. Multiply the twiddle factor matrix with the input to get the DFT of the sequence.
6. Plot the magnitude and the phase of the DFT of the sequence using inbuilt functions **abs ( )** and **angle( )**.

PROGRAM:

DFT:

```
clc;
close all;
clear all;
x=input('Enter the sequence x= ');
N=input('Enter the length of the DFT N= ');
xlen=length(x);
if N>xlen
x=[x zeros(1,N-xlen)];
elseif N<xlen
x=x(1:N);
end
n=0:(N-1);
k=0:(N-1);
nk=n'*k;
W=exp(-i*2*pi/N).^nk;
X=x*W;
disp(X);
subplot(2,1,1);
mag=abs(X);
stem(k,mag);
title('Magnitude Spectrum');
xlabel('Discrete frequency');
ylabel('Amplitude');
grid  on;
subplot(2,1,2);
theta=angle(X);
stem(k,theta);
title('Phase Spectrum');
xlabel('Discrete frequency');
ylabel('Phase Angle');
grid on;
```

IDFT:

```
clc;
close all;
clear all;
X=input('Enter the sequence X(K)= ');
N=input('Enter the length of the IDFT N= ');
xlen=length(X);
if N>xlen
X=[X zeros(1,N-xlen)];
elseif N<xlen
X=X(1:N);
end
n=0:(N-1);
k=0:(N-1);
nk=n'*k;
W=exp(i*2*pi/N).^nk;
x=(1/N).*(X*W);
disp('The IDFT of given X(K):');
disp(x);
subplot(2,1,1);
mag=abs(x);
stem(k,mag);
title('Magnitude Spectrum');
xlabel('No. of Sample');
ylabel('Amplitude');
grid  on;
subplot(2,1,2);
theta=angle(x);
stem(k,theta);
title('Phase Spectrum');
xlabel('No. of Sample');
ylabel('Phase Angle');
grid on;
```

RESULT:

VIVA QUESTIONS:

1. **What is DFT?**

   DFT stands for Discrete Fourier Transform. It is a mathematical transformation that converts a discrete sequence of time-domain samples into a corresponding sequence of frequency-domain samples.

2. **What is the difference between DFT and FFT?**

   DFT (Discrete Fourier Transform) and FFT (Fast Fourier Transform) are closely related. DFT is a mathematical algorithm that calculates the discrete Fourier transform, while FFT is an optimized algorithm to compute DFT more efficiently. In other words, FFT is an efficient implementation of DFT.

3. **What are the applications of DFT?**

   DFT has numerous applications, including:
   Signal processing: DFT is widely used in signal analysis, filtering, and spectral analysis.
   Image processing: DFT is used in image compression, enhancement, and feature extraction.
   Audio processing: DFT is utilized in audio analysis, equalization, and synthesis.
   Communication systems: DFT plays a crucial role in modulation, demodulation, and channel estimation.
   Radar and sonar systems: DFT is employed in target detection, range estimation, and signal processing.
   Medical imaging: DFT finds applications in medical image reconstruction, analysis, and diagnosis.

4. **What is the computational complexity of the DFT algorithm?**

   The direct computation of DFT requires O(N^2) operations, where N is the length of the input sequence. However, using the FFT algorithm, the computational complexity can be reduced to O(N log N), which makes it more efficient for larger values of N.

5. **What is the concept of zero-padding in DFT and why is it used?**

   Zero-padding is the technique of adding zeros to the input sequence before performing DFT. It is used to increase the frequency resolution of the DFT output. By adding more zeros, the resulting DFT spectrum has more points and thus provides a higher-resolution representation of the frequency content of the signal. However, it does not add any additional information about the signal beyond the original data.

6. **What is the windowing technique in DFT and why is it used?**

   Windowing is a technique applied to the input signal before performing DFT. It involves multiplying the signal with a window function such as Hamming, Hanning, or Blackman window. Windowing helps to reduce spectral leakage, which occurs when the frequency components of the signal extend beyond a single DFT bin and spread energy to adjacent bins. By tapering the signal using a window function, spectral leakage is minimized, resulting in a more accurate representation of the frequency content in the DFT spectrum.

EXP.NO. 5 **DATE:**

PROPERTIES OF DISCRETE TIME SYSTEMS

AIM:

To verify the properties of discrete time systems using MATLAB program.

SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

THEORY:

Discrete Time Systems:

- A system is a mathematical or physical device that takes an input signal and produces an output signal.
- A discrete-time system is a type of system that operates on discrete-time signals and produces output as discrete-time signals.

Properties of Discrete Time Systems:

1. Linear and Non-Linear Systems:

- A system that satisfies the superposition principle is said to be a linear system.
- Superposition principle states that the response of the system to a weighted sum of signals should be equal to the corresponding weighted sum of the outputs of the system to each of the individual input signals.
- A system is linear if and only if $T[a_1x_1(n) + a_2x_2(n)] = a_1T[x_1(n)] + a_2T[x_2(n)]$ for any arbitrary constant a_1 and a_2.
- A relaxed system that does not satisfy the superposition principle is called non-linear.

2. Time variant and Time-invariant Systems:

- A system is said to be time-invariant or shift invariant if the characteristics of the system do not change with time.
- For a time-invariant system if $y(n)$ is the response of the system to the input $x(n)$, then the response of a system to the input $x(n-k)$ is $y(n-k)$.
- If the input sequence is shifted by k samples, the generated output sequence is, the original sequence shifted by k samples.

3. Stable and Unstable Systems:

- An LTI system is stable if it produces a bounded output sequence for every bounded input sequence.

- If, for some bounded input sequence $x(n)$, the output is unbounded (infinite), the system is classified as unstable.
- The necessary and sufficient condition for stability is $\sum^{n=\infty} |h(n)| < \infty$

LINEAR AND NON-LINEAR SYSTEMS:

FLOWCHART /ALGORITHM:

1. Generate random input signals **x1** and **x2**.
2. Calculate output signals **y1** and **y2** using the linear operation **y(n) = 2x(n)**.
3. Define scaling constants **a** and **b**.
4. Compute superposition LHS and RHS signals.
5. Compare LHS and RHS using equality operator to check linearity.
6. Display linear or nonlinear result.

PROGRAM:

LINEAR SYSTEMS:

```
% Linear System
% Generate input signals
x1 = rand(1, 100); % First input signal
x2 = rand(1, 100);  % Second input signal
% System operation: y(n) = 2x(n)
y1 = 2 * x1;
y2 = 2 * x2;
% Check superposition
a = 3;
b = 5;
lhs = 2 * (a * x1 + b * x2);
rhs = a * y1 + b * y2;
% Determine linearity
if lhs == rhs
    disp ('The given system is LINEAR.');
else
    disp ('The given system is NON-LINEAR.');
end
subplot (3, 1, 1);
stem (lhs, 'b', 'Line Width', 2);
title ('LHS Signal');
xlabel ('Time');
ylabel ('Amplitude');
grid on;
subplot (3, 1, 2);
stem (rhs, 'r', 'Line Width', 2);
title('RHS Signal');
xlabel ('Time');
ylabel ('Amplitude');
grid on;
subplot(3, 1, 3);
```

```
stem (lhs - rhs, 'g', 'Line Width', 2);
title('Difference between LHS and RHS');
xlabel ('Time');
ylabel ('Difference');
grid on;
```

NON-LINEAR SYSTEMS:

```
% Non Linear System
% Generate input signals
x1 = rand(1, 20); % First input signal
x2 = rand(1, 20);  % Second input signal
% System operation: y(n) = x^2(n)
y1 = x1.^2;
y2 = x2.^2;
% Check superposition
a = 3;
b = 5;
lhs = 2 * (a * x1 + b * x2);
rhs = a * y1 + b * y2;
% Determine linearity
if lhs == rhs
    disp ('The given system is LINEAR.');
else
    disp ('The given system is NONLINEAR.');
end
subplot (3, 1, 1);
stem (lhs, 'b', 'Line Width', 2);
title('LHS Signal');
xlabel('Time');
ylabel('Amplitude');
grid on;
subplot(3, 1, 2);
stem (rhs, 'r', 'Line Width', 2);
title('RHS Signal');
xlabel ('Time');
ylabel ('Amplitude');
grid on;
subplot (3, 1, 3);
stem (lhs - rhs, 'g', 'Line Width', 2);
title ('Difference between LHS and RHS');
xlabel ('Time');
ylabel ('Difference');
grid on;
```

TIME VARIANT AND TIME INVARIANT

FLOWCHART /ALGORITHM:

1. Initialize the input signal and shift value **k**.
2. Calculate the original system response **y_original**.
3. Loop through each input shift value from 1 to **k**:
    - Generate the shifted input signal.
    - Calculate the system response for the shifted input.
4. Loop through each output shift value from 1 to **k**:
    - Generate the shifted output signal.
    - Calculate the system response for the original input.
5. Compare the shifted signals and find the system is time-invariant or time-variant.
6. Plot the outputs due to shifted input and shifted output for visualization.

PROGRAM:

TIME VARIANT AND TIME INVARIANT SYSTEMS:

```
clc;
clear all;
close all;
k = 2;
% Define input signal
n = 0:2 + k;
x = [10 2 5 zeros(1, k)];
% Step 1: Check input shifts
for input_shift = 1:k
    % Shifted input
    x_shifted = [zeros(1, input_shift), x(1:end - input_shift)];

    % Calculate responses
    y_original = x + n .* x;
    y_shifted = x_shifted + n .* x_shifted;
end

% Step 2: Check output shifts
for output_shift = 1:k
    % Shifted output
    y_shifted_output = [zeros(1, output_shift), y_original(1:end - output_shift)];

    % Calculate responses
    y_original = x + n .* x;
end

% Determine and display result
if isequal(y_original, y_shifted)
    disp('The given system is TIME-INVARIANT.');
else
    disp('The given system is TIME-VARIANT.');
end
subplot(2,1,1)
stem(y_shifted)
xlabel("Time");
ylabel("Amplitute")
title("Output due to shiff in input signal")
subplot(2,1,2)
```

```
stem(y_original)
xlabel("Time");
ylabel("Amplitute")
title("Output shifted signal")
```

STABLE AND UNSTABLE SYSTEM

FLOWCHART /ALGORITHM:

1. **Initialization:**
    - Clear existing variables and plots.
    - Define the time vector **n**.
    - Define the impulse response function **h**.
    - Initialize the sum of coefficients **sum** to 0.
2. **Calculate Coefficient Sum:**
    - Iterate through each coefficient in **h**.
    - Check if the absolute value of the coefficient is below a threshold (**1e-6**).
    - Accumulate the absolute coefficient values in the **sum** variable.
3. **Plot Impulse Response:**
    - Plot the impulse response **h** using the **stem** function.
4. **Determine System Stability:**
    - Display the calculated sum of impulse response coefficients.
    - Compare the sum to a stability threshold (**5.0983e+8**).
    - If the sum exceeds the threshold, the system is unstable. Otherwise, it's stable.

PROGRAM:

STABLE AND UNSTABLE SYSTEM

```
clc;
clear all;
close all;

n = 0:0.1:20;
h = 0.5 * exp(-n);
sum = 0;  % Rename Sum to sumCoefficients

for k = 1:length(h)
    if abs(h(k)) < 1e-6
        break;
    end
    sum = sum + abs(h(k));
end


stem(n, h);
title('Impulse Response');
xlabel('Time');
ylabel('Amplitude');
grid on;

disp('The sum of impulse response coefficients is:');
disp(sum);

if sum > 5.0983e+8
    disp('The system is UNSTABLE.');
else
    disp('The system is STABLE.');
end
```

RESULT:

VIVA QUESTIONS:

1. What are discrete time systems?

- A discrete-time system is a type of system that operates on discrete-time signals and produces output as discrete-time signals.

2. Define Linear and non-linear systems.

- A system that satisfies the superposition principle is said to be a linear system, otherwise it is non-linear system.
- Superposition principle states that the response of the system to a weighted sum of signals should be equal to the corresponding weighted sum of the outputs of the system to each of the individual input signals.

3. Give the necessary and sufficient condition for stability.

- The necessary and sufficient condition for stability is $\sum_{n=-\infty}^{n=\infty} |h(n)| < \infty$

4. Define time variant and time invariant systems.

- A system is said to be time-invariant or shift invariant if the characteristics of the system do not change with time.
- For a time-invariant system if $y(n)$ is the response of the system to the input $x(n)$, then the response of a system to the input $x(n-k)$ is $y(n-k)$.
- If the input sequence is shifted by k samples, the generated output sequence is, the original sequence shifted by k samples.

5. Define stable and unstable systems.

- An LTI system is stable if it produces a bounded output sequence for every bounded input sequence.
- If, for some bounded input sequence $x(n)$, the output is unbounded (infinite), the system is classified as unstable.

EXP.NO. 6 **DATE:**

FREQUENCY RESPONSE OF IIR FILTER

AIM:

To design a Butterworth and Chebyshev low pass filter using Bilinear Transformation and Impulse Invariant Transformation method and analyze its frequency response characteristics using MATLAB.

SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

Theory*:*

A digital filter is a linear time invariant discrete time system. The digital filters are classified into two, based on their lengths of impulse response

1. **Finite Impulse response (FIR)**
   They are of non-recursive type and h [n] has finite number of samples
2. **Infinite Impulse response (IIR)**
   h[n] has finite number of samples. They are of recursive type. Hence, their transfer function is of the form

$$H(z) = \sum_{n=0}^{\alpha} h(n) z^{-n}$$

$$H(Z) = \frac{\sum_{K=0}^{M-1} b_k Z^{-k}}{1 + \sum_{j=1}^{N-1} a_j Z^{-j}}$$

The digital filters are designed from analog filters. The two widely used methods for digitizing the analog filters include

1. Bilinear transformation
2. Impulse Invariant transformation

The bilinear transformation maps the s-plane into the z-plane by

$$H(Z) = H(S)|_s = \frac{2}{T} \times \frac{1 - Z^{-1}}{1 + Z^{-1}}$$

This transformation maps the jΩ axis (from Ω = -∞ to +∞) repeatedly around the unit circle (exp(jw), from w=-π to π) by

$$\Omega = \frac{2}{T} \tan\left(\frac{\omega}{2}\right)$$

***Bilinear transformation**:*
Design steps:

1. From the given specifications, Find pre-warped analog frequencies using $\Omega = \frac{2}{T}\tan\left(\frac{\omega}{2}\right)$
2. Using the analog frequencies, find H(s) of the analog filter
3. Substitute $S = \frac{2}{T} \times \frac{1-Z^{-1}}{1+Z^{-1}}$ in the H(s) of Step:2

***Impulse invariant transformation**:*
Design steps:

1. Find the analog frequency using ω / T
2. Find the transfer function of analog filter $H_a(s)$
3. Express the analog filter transfer function as a sum of single pole filters
4. Compute H(Z) of digital filter using the formula

$$H(Z) = \sum_{k=1}^{N} \frac{C_k}{1 - e^{-TP_k} Z^{-1}}$$

Library functions:

- **Impinvar:**

  Impulse Invariant method for analog-to-digital filter conversion [bz,az] = impinvar(b,a,fs) creates a digital filter with numerator and denominator coefficients bz and az, respectively, whose impulse response is equal to the impulse response of the analog filter with coefficients b and a, scaled by 1/fs. If you leave out the argument fs (or) specify fs as an empty vector [ ], it takes the default value of 1 Hz.

- **Bilinear:**
  Bilinear transformation method for analog-to-digital filter conversion. The bilinear transformation is a mathematical mapping of variables. In digital filtering, it is a standard method of mapping the s or analog plane into the z or digital plane. It transforms analog filters, designed using classical filter design

***Procedure*:**

1. Calculate the attenuation in dB for the given design parameters
2. Design the analog counterpart
3. Using Impulse Invariant /Bilinear transformation design the digital filter
4. Display the transfer function. Plot the magnitude response and phase response

BUTTERWORTH FILTER DESIGN

PROGRAM:

```
close all;
clear all;
% Butterworth analog low pass filter
disp('LPF FILTER');
wp=input('Enter the pass band frequency: ');
ws=input('Enter the stop band frequency: ')';
ap=input('Enter the pass band attenuation: ');
as=input('Enter the stop band attenuation: ');
[N,wc]=buttord(wp,ws,ap,as,'s');
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=butter(N,wc,'s');
[H,f]=freqs(Nr,Dr);
subplot(2,2,1);
plot(f,abs(H));
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of LPF');

% Butterworth analog high pass filter
disp('HPF FILTER');
wp=input('Enter the pass band frequency: ');
ws=input('Enter the stop band frequency: ')';
ap=input('Enter the pass band attenuation: ');
as=input('Enter the stop band attenuation: ');
[N,wc]=buttord(wp,ws,ap,as,'s');
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=butter(N,wc,'high','s');
[H,f]=freqs(Nr,Dr);
subplot(2,2,2);
plot(f,abs(H));
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of HPF');

% Butterworth analog band pass filter
disp('BPF FILTER');
wp=input('Enter the pass band frequency: ');
ws=input('Enter the stop band frequency: ')';
ap=input('Enter the pass band attenuation: ');
as=input('Enter the stop band attenuation: ');
% wp=[100 200];
% ws=[50 250];
% ap=2;
% as=30;
[N,wc]=buttord(wp,ws,ap,as,'s');
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=butter(N,wc,'bandpass','s');
H=freqs(Nr,Dr);
subplot(2,2,3);
plot(abs(H));
axis([0 300 0 1.5]);
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of BPF');

% Butterworth analog bandstop filter
disp('BSF / BRF / BEF FILTER');
wp=input('Enter the pass band frequency: ');
ws=input('Enter the stop band frequency: ')';
ap=input('Enter the pass band attenuation: ');
as=input('Enter the stop band attenuation: ');
% wp=[50 250];
% ws=[100 200];
% ap=2;
% as=30;
[N,wc]=buttord(wp,ws,ap,as,'s');
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=butter(N,wc,'stop','s');
H=freqs(Nr,Dr);
subplot(2,2,4);
plot(abs(H));
axis([0 200 0 1.5]);
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of BSF');
```

OUTPUT:

LPF FILTER

```
Enter the pass band frequency: 20
Enter the stop band frequency: 30
Enter the pass band attenuation: 2
Enter the stop band attenuation: 10
Order of the filter: 4
Cut off frequency: 22.7951
```

HPF FILTER

```
Enter the pass band frequency: 20
Enter the stop band frequency: 30
Enter the pass band attenuation: 2
Enter the stop band attenuation: 10
Order of the filter: 4
Cut off frequency: 22.7951
```

BPF FILTER

```
Enter the pass band frequency: [100 200]
Enter the stop band frequency: [50 250]
Enter the pass band attenuation: 2
Enter the stop band attenuation: 30
Order of the filter: 8
Cut off frequency: 96.6119  207.0137
```

BSF / BRF / BEF FILTER

```
Enter the pass band frequency: [50 250]
Enter the stop band frequency: [100 200]
Enter the pass band attenuation: 2
Enter the stop band attenuation: 30
Order of the filter: 8
Cut off frequency: 84.0292  238.0123
```

CHEBYSHEV FILTER DESIGN

PROGRAM:

```
close all;
clear all;
% Chebyshev low pass filter

disp('LPF FILTER');
% fp=input('Enter the pass band frequency: ');
% fs=input('Enter the stop band frequency: ')';
% ap=input('Enter the pass band attenuation: ');
% as=input('Enter the stop band attenuation: ');
% F=input('Enter the sampling frequency: ');
% wp=2*pi*fp/F;
% ws=2*pi*fs/F;
wp=0.2*pi;
ws=0.4*pi;
ap=0.6;
as=30;
[N,wc]=cheb1ord(wp,ws,ap,as,'s');
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=cheby1(N,ap,wc,'low');
w=0:0.01:pi;
[H,f]=freqz(Nr,Dr,w);
subplot(2,2,1);
plot(f,abs(H));
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of LPF');

% Chebyshev high pass filter

disp('HPF FILTER');
% fp=input('Enter the pass band frequency: ');
% fs=input('Enter the stop band frequency: ')';
% ap=input('Enter the pass band attenuation: ');
% as=input('Enter the stop band attenuation: ');
% F=input('Enter the sampling frequency: ');
% wp=2*pi*fp/F;
% ws=2*pi*fs/F;
wp=0.2*pi;
ws=0.4*pi;
ap=0.6;
as=30;
[N,wc]=cheb1ord(wp,ws,ap,as,'s');
```

```
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=cheby1(N,ap,wc,'high');
w=0:0.01:pi;
[H,f]=freqz(Nr,Dr,w);
subplot(2,2,2);
plot(f,abs(H));
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of HPF');

% Chebyshev band pass filter

disp('BPF FILTER');
% fp=input('Enter the pass band frequency: ');
% fs=input('Enter the stop band frequency: ')';
% ap=input('Enter the pass band attenuation: ');
% as=input('Enter the stop band attenuation: ');
% F=input('Enter the sampling frequency: ');
% wp=2*pi*fp/F;
% ws=2*pi*fs/F;
wp=[0.1*pi 0.3*pi];
ws=[0.05*pi 0.35*pi];
ap=0.6;
as=30;
[N,wc]=cheb1ord(wp,ws,ap,as,'s');
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=cheby1(N,ap,wc,'bandpass');
w=0:0.01:pi;
[H,f]=freqz(Nr,Dr,w);
subplot(2,2,3);
plot(f/pi,abs(H));
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of BPF');

% Chebyshev band stop filter

disp('BSF / BRF / BEF FILTER');
% fp=input('Enter the pass band frequency: ');
% fs=input('Enter the stop band frequency: ')';
% ap=input('Enter the pass band attenuation: ');
% as=input('Enter the stop band attenuation: ');
% F=input('Enter the sampling frequency: ');
% wp=2*pi*fp/F;
% ws=2*pi*fs/F;
wp=[0.05*pi  0.25*pi];
ws=[0.1*pi 0.2*pi];
ap=0.6;
as=30;
[N,wc]=cheb1ord(wp,ws,ap,as,'s');
disp('Order of the filter: ');
disp(N);
disp('Cut off frequency: ');
disp(wc);
[Nr,Dr]=cheby1(N,ap,wc,'stop');
w=0:0.01:pi;
[H,f]=freqz(Nr,Dr,w);
subplot(2,2,4);
plot(f/pi,abs(H));
xlabel('frequency');
ylabel('magnitude');
title('Magnitude Response of BSF');
```

<u>OUTPUT:</u>

LPF FILTER

```
Order of the filter: 4
Cut off frequency: 0.6283
```

HPF FILTER

```
Order of the filter: 4
Cut off frequency: 0.6283
```

BPF FILTER

```
Order of the filter: 7
Cut off frequency: 0.3142    0.9425
```

BSF / BRF / BEF FILTER

```
Order of the filter: 5
Cut off frequency: 0.2513    0.7854
```

BILINEAR TRANSFORMATION METHOD

PROGRAM:

```
clc;
clear all;
close all;
t=1;
wp=2*pi/2;
ws=7*pi/2;
wp=wp/t;
e=((1/(0.707)^2)-1)^0.5;
l=((1/(0.2)^2)-1)^0.5;
N=ceil((log10(l/e))/log10(ws/wp)+1);
wc=wp/(e)^(0.5/N);
Fs=1/t;
[Z,P,K]=buttap(N);
P=P*wc;
K=K*wc^N;
B=real(poly(Z));
b=K*B;
a=real(poly(P));
[b,a]=bilinear(b,a,Fs);
[H,w]=freqz(b,a);
subplot(3,1,1);
plot(w,real(H));
title('Frequency Response');
xlabel('Frequency');
ylabel('Magnitude');
mag=abs(H);
subplot(3,1,2);
plot(w,mag);
title('Magnitude Response');
xlabel('Frequency');
ylabel('Magnitude');
phase=angle(H);
subplot(3,1,3);
plot(w,phase);
title('Phase Response');
xlabel('Frequency');
ylabel('Angle');
```

IMPULSE INVARIANT METHOD

PROGRAM:

```
clc;
clear all;
close all;
N=3;
wc=0.95;
T=1;
Fs=1/T;
[Z,P,K]=buttap(N);
P=P*wc;
K=K*wc^N;
B=real(poly(Z));
b=K*B;
a=real(poly(P));
[b,a]=impinvar(b,a,Fs);
[H,w]=freqz(b,a);
subplot(3,1,1);
plot(w,real(H));
title('Frequency Response');
xlabel('Frequency');
ylabel('Magnitude');
mag=abs(H);
subplot(3,1,2);
plot(w,mag);
title('Magnitude Response');
xlabel('Frequency');
ylabel('Magnitude');
phase=angle(H);
subplot(3,1,3);
plot(w,phase);
title('Phase Response');
xlabel('Frequency');
ylabel('Angle');
```

RESULT:

Viva Questions:

1. What are IIR filters?

   IIR filters, short for Infinite Impulse Response filters, are a type of digital filter used in signal processing and digital signal analysis. These filters have feedback in their implementation, which means the output of the filter depends not only on the current input but also on past input samples. The term "Infinite Impulse Response" arises from the fact that the impulse response of an IIR filter does not decay to zero but continues indefinitely, though often with diminishing magnitude over time.

2. Differentiate FIR and IIR filters.

   FIR (Finite Impulse Response) and IIR (Infinite Impulse Response) filters are two distinct classes of digital filters:
   FIR filters: These filters have a finite impulse response, meaning the impulse response becomes zero after a finite number of samples. They do not use feedback in their implementation, and the output is solely determined by the current and past input samples. FIR filters are always stable, have linear phase characteristics, and are generally easier to design compared to IIR filters. However, they may require a larger number of coefficients to achieve the same filter characteristics as IIR filters
   IIR filters: As mentioned earlier, IIR filters have an impulse response that continues indefinitely. They utilize feedback in their implementation, which allows them to achieve similar filtering characteristics with fewer coefficients compared to FIR filters. However, IIR filters can be less stable, may exhibit non-linear phase responses, and are typically more complex to design

3. Why is the impulse response of IIR filter infinite?

   The impulse response of an IIR filter is infinite due to the presence of feedback in its implementation. When an impulse (i.e., a signal with value 1 at a specific sample and 0 elsewhere) is applied to the input of an IIR filter, the filter processes this impulse and generates an output signal. Since IIR filters have feedback, this output signal influences the filter's behavior at subsequent time steps. The process of feedback and recursive calculations results in the impulse response extending infinitely in time, never fully decaying to zero.

4. Define Gibb’s Phenomenon

   Gibbs' Phenomenon is a characteristic phenomenon observed in the frequency domain when approximating a step function or square wave using a finite number of Fourier series harmonics. It occurs when performing Fourier analysis or synthesis. Specifically, when attempting to reconstruct a discontinuous waveform using a limited number of harmonics, oscillations (ripples) occur near the edges of the reconstructed waveform. These oscillations persist even as the number of harmonics used in the approximation increases, and they do not diminish unless an infinite number of harmonics is used. In practical applications, these ripples can lead to undesired artifacts and overshoots in the reconstructed signal.

EXP.NO. 7 **DATE:**

FREQUENCY RESPONSE OF FIR FILTER

AIM:

To design an FIR Filter and obtain the frequency response using windowing technique.

SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

THEORY:

An FIR filter of length 'N' with input $x[n]$ and output $y[n]$ is described by the difference equation

$$\begin{aligned} y[n] &= b_0x[n] + b_1x[n-1] + \ldots\ldots + b_{N-1}x[n-N+1] \\ &= \sum_{k=0}^{N-1} b_k x[n-k] \end{aligned}$$

where b_k is the set of linear coefficients. The design of an FIR filter for the given specifications is determining filter coefficients b_k of the filter.

Designing FIR filters using Windows: The main limitations of finite duration filter design using Fourier series method is oscillations in the pass band and stop band due to the slow convergence. This effect is known as Gibbs phenomenon. To reduce these oscillations, the Fourier series coefficients are modified by multiplying a weighting sequence called window $w[n]$,

where
$$w[-n] = w[n] = \begin{cases} \neq 0, \text{for } |n| \leq \frac{N-1}{2} \\ = 0, \text{for } |n| > \frac{N-1}{2} \end{cases}$$

After multiplication of window sequence $w[n]$ with $h_d[n]$, we get finite duration sequence $h[n]$, that satisfies the desired magnitude response.

$$h[n] = \begin{cases} h_d[n]\, w[n], \text{for } |n| \leq \frac{N-1}{2} \\ 0, \qquad\qquad \text{for } |n| > \frac{N-1}{2} \end{cases}$$

The frequency response $H[e^{j\omega}] = H_d[e^{j\omega}] * W[e^{j\omega}]$.

FLOWCHART /ALGORITHM:

1. Start the MATLAB program.
2. Generate the window functions (Rectangular and Hanning) using Matlab comments and plot the

frequency response of both the windows. (To obtain the N-Point Frequency Response Vector, “h”, and the corresponding Angular Frequency Vector, “omega”, for the Digital Filter with Nr and Dr coefficient stored in B (Rw in the code) and A (1 in the code), respectively.

3. Design an n-th order Low pass/ High pass, depending on the value of ftype and the number of elements of wn, by using the command fir1(n,Wn,ftype)
4. Obtain the frequency response, plot the magnitude and phase spectrum.

PROCEDURE:

1. Start the MATLAB program.
2. Generate the window functions (Rectangular, Hamming and Hanning) using MATLAB comments.
3. Design an N-th order LPF, HPF, BPF and BSF of digital FIR filters using above windowing methods.
4. Plot the magnitude response of the filters.

PROGRAM:

```
% FIR FILTER USING RECTANGULAR WINDOW

close all;
clear all;
% Rectangular Windowing for low pass
disp('LPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=boxcar(N+1);
a=fir1(N,Wn,y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,1);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR LPF using Rectangular window");

% Rectangular Windowing for high pass
disp('HPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=boxcar(N+1);
a=fir1(N,Wn,'high',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,2);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR HPF using Rectangular window");

% Rectangular Windowing for bandpass
disp('BPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
```

```
y=boxcar(N+1);
a=fir1(N,Wn,'bandpass',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,3);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR BPF using Rectangular window");
```

% Rectangular Windowing for bandstop

```
disp('BSF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=boxcar(N+1);
a=fir1(N,Wn,'stop',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,4);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR BSF using Rectangular window");
```

OUTPUT: (Rectangular Window)

LPF

```
Enter the order of the filter: 20
Enter the cut off frequency: 0.2*pi
```

HPF

```
Enter the order of the filter: 20
Enter the cut off frequency: 0.2*pi
```

BPF

```
Enter the order of the filter: 20
Enter the cut off frequency: [0.1*pi 0.2*pi]
```

BSF

```
Enter the order of the filter: 20
Enter the cut off frequency: [0.1*pi 0.2*pi]
```

% FIR FILTER USING HAMMING WINDOW

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

% Hamming Windowing for low pass

```
disp('LPF');
N=input("Enter the order of the filter: ");
```

```
Wn=input("Enter the cut off frequency: ");
y=hamming(N+1);
a=fir1(N,Wn,y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,1);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR LPF using Hamming window");
```

% Hamming Windowing for high pass

```
disp('HPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=hamming(N+1);
a=fir1(N,Wn,'high',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,2);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR HPF using Hamming window");
```

% Hamming Windowing for bandpass

```
disp('BPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=hamming(N+1);
a=fir1(N,Wn,'bandpass',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,3);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR BPF using Hamming window");
```

% Hamming Windowing for bandstop

```
disp('BSF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=hamming(N+1);
a=fir1(N,Wn,'stop',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,4);
plot(o/pi,m);
xlabel("frequency");
```

```
ylabel("magnitude");
title("FIR BSF using Hamming window");
```

OUTPUT: (Hamming Window)

LPF

```
Enter the order of the filter: 20
Enter the cut off frequency: 0.2*pi
```

HPF

```
Enter the order of the filter: 20
Enter the cut off frequency: 0.2*pi
```

BPF

```
Enter the order of the filter: 20
Enter the cut off frequency: [0.1*pi 0.2*pi]
```

BSF

```
Enter the order of the filter: 20
Enter the cut off frequency: [0.1*pi 0.2*pi]
```

%FIR FILTER USING HANNING WINDOW

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

% Hanning Windowing for low pass

```
disp('LPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=hanning(N+1);
a=fir1(N,Wn,y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,1);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR LPF using Hanning window");
```

% Hanning Windowing for high pass

```
disp('HPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=hanning(N+1);
a=fir1(N,Wn,'high',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,2);
plot(o/pi,m);
```

```
xlabel("frequency");
ylabel("magnitude");
title("FIR HPF using Hanning window");
```

% Hanning Windowing for bandpass

```
disp('BPF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=hanning(N+1);
a=fir1(N,Wn,'bandpass',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,3);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR BPF using Hanning window");
```

% Hanning Windowing for bandstop

```
disp('BSF');
N=input("Enter the order of the filter: ");
Wn=input("Enter the cut off frequency: ");
y=hanning(N+1);
a=fir1(N,Wn,'stop',y);
[h,o]=freqz(a,1,256);
m=abs(h);
subplot(2,2,4);
plot(o/pi,m);
xlabel("frequency");
ylabel("magnitude");
title("FIR BSF using Hanning window");
```

OUTPUT: (Hanning Window)

LPF

```
Enter the order of the filter: 20
Enter the cut off frequency: 0.2*pi
```

HPF

```
Enter the order of the filter: 20
Enter the cut off frequency: 0.2*pi
```

BPF

```
Enter the order of the filter: 20
Enter the cut off frequency: [0.1*pi 0.2*pi]
```

BSF

```
Enter the order of the filter: 20
Enter the cut off frequency: [0.1*pi 0.2*pi]
```

RESULT:

VIVA QUESTIONS:

1. List the advantages of FIR filters over IIR filters.

(i) They can have an exact linear phase.
(ii) They are always stable.
(iii) The design methods are generally linear.
(iv) They can be realized efficiently in hardware.
(v) The filter start-up transients have finite duration.

2. Mention the differential equation for an FIR filter.

An FIR filter of length 'N' with input $x[n]$ and output $y[n]$ is described by the difference equation

$$y[n] = b_0 x[n] + b_1 x[n-1] + \ldots\ldots + b_{N-1} x[n-N+1]$$
$$= \sum_{k=0}^{N-1} b_k x[n-k]$$

where b_k is the set of linear coefficients. The design of an FIR filter for the given specifications is determining filter coefficients b_k of the filter.

3. Define Gibb's phenomenon.

The main limitations of finite duration filter design using Fourier series method is oscillations in the pass band and stop band due to the slow convergence. This effect is known as Gibbs phenomenon.

4. State the equation for Rectangular and Hanning Window.

Rectangular Window

$$W_R(n) = \{ \begin{matrix} 1 \; for \; 0 \le n \le N-1 \\ 0 \; otherwise \end{matrix} \}$$

Hanning Window:

$$W_{Hn}(n) = \{ \begin{matrix} 0.5 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) \; for \; 0 \le n \le N-1 \\ 0 \; otherwise \end{matrix} \}$$

5. Mention the MATLAB commands (i) to plot the frequency response of a window function, (ii) to design an Nth order filter

(i) [h,omega1]=freqz(Rw,1,256);

[h,w] = freqz(b,a,n) returns the n-point frequency response vector, h, and the corresponding angular frequency vector, w, for the digital filter with Nr and Dr cfnts stored in b and a, respectively.

(ii) b=fir1(10,0.2,Rw);

fir1(n,Wn,ftype) designs a n-th order LP/ HP, depending on the value of ftype and the number of elements of Wn

EXP.NO. 8 **DATE:**

INTERPOLATION AND DECIMATION

AIM:

To perform Interpolation and decimation operations for the given signal using MATLAB program.

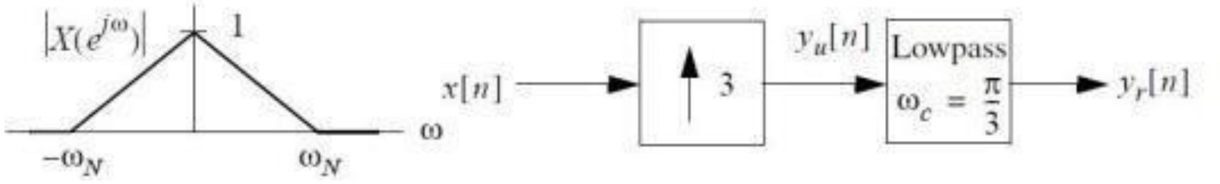
SOFTWARE / HARDWARE REQUIREMENTS:

- MATLAB
- Personal Computer

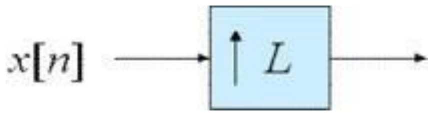
THEORY:

Interpolation:

Interpolation is the process of upsampling followed by filtering. Up-sampler - Used to increase the sampling rate by an integer factor. Low pass filter is called Anti imaging filter. It is used to eliminate anti images in up sampler output



Up-Sampler: An up-sampler with an up-sampling factor L, where L is a positive integer, develops an output sequence $x_u[n]$ with a sampling rate that is L times larger than that of the input sequence x[n]



Input-output relation

$$x_u[n] = \begin{cases} x[n/L], & n = 0, \pm L, \pm 2L, \cdots \\ 0, & \textbf{otherwise} \end{cases}$$

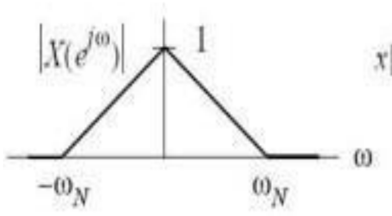
Up-sampling operation is implemented by inserting L-1 equidistant zero-valued samples between two consecutive samples of x[n].

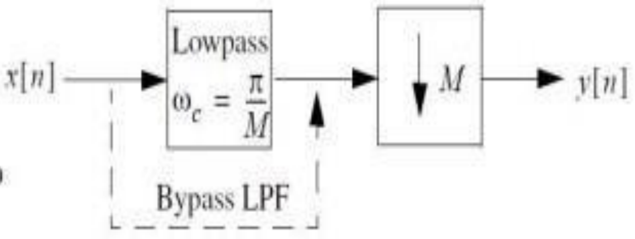
Input-output relation

$$x_u[n] = \begin{cases} x[n/L], & n = 0, \pm L, \pm 2L, \cdots \\ 0, & \textbf{otherwise} \end{cases}$$

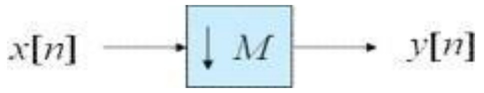
Decimation:

Decimation is the process of filtering followed by down sampling. Down-sampler is used to decrease the sampling rate by an integer factor





Down-Sampler: An down-sampler with a down-sampling factor M, where M is a positive integer, develops an output sequence y[n] with a sampling rate that is (1/M)-th of that of the input sequence x[n]. Down-sampling operation is implemented by keeping every M-th sample of x[n] and removing M-1 in-between samples to generate y[n]



Input-output relation y(n)= x(nM)

Where M is an integer

FLOWCHART /ALGORITHM:

1. Start the MATLAB program.
2. Define the time indices
3. Generate input signal
4. Perform Down sampling/ up sampling operation by the factor
5. Plot the input signal
6. Plot the Down sampling/ up sampling output

PROGRAM:

Interpolation:

```
clc;
clear all;
close all;
t = 0:.0025:1;                  % Time vector
x = sin(2*pi*5*t);
y = interp(x,3);
subplot(1,2,1);
stem(x(1:120));
axis([0 120 -2 2]); % Original signal
title('Original  Signal');
subplot(1,2,2);
stem(y(1:360));
axis([0 360 -2 2]) ;  % Interpolated signal
title('Interpolated Signal')
```

Decimation:

```
clc;
clear all;
close all;
t = 0:.0025:1;                    % Time vector
x = sin(2*pi*5*t);
y = decimate(x,3);
subplot(1,2,1);
stem(x(1:120));
axis([0 120 -2 2])  % Original signal
title('Original  Signal')
subplot(1,2,2);
stem(y(1:40));
axis([0 40 -2 2]);  % Decimated signal
title('Decimated Signal')
```

RESULT:

Viva Questions:

1. **What is meant by decimation?**
   - Decimation is the process of reducing the sampling frequency of a signal to a lower sampling frequency that differs from the original frequency by an integer value

2. **What is meant by interpolation?**
   - In the domain of digital signal processing, the term interpolation refers to the process of converting a sampled digital signal (such as a sampled audio signal) to that of a higher sampling rate (Upsampling) using various digital filtering techniques (for example, convolution with a frequency-limited impulse signal)

3. **What is function of downsampling?**
   - Down sampling a sequence x[n] by retaining only every M^th sample creates a new sequence $x_d[n] = x[nM]$. If the original sequence contains frequency components above $/M$, the down sampler should be preceded by a low-pass filter with cutoff frequency π/M .

4. **What is function of upsampling?**
   - Up sampling a sequence x[n] creates a new sequence $X_e[n]$ where every L^th sample is taken from x[n] with all others zero. The up sampled sequence contains L replicas of the original signal's spectrum

5. **What is need for multirate signal processing?**
   - Multirate digital signal processing the sampling rate of a signal is changed in or- der to increase the efficiency of various signal processing operations. Decimation, or down- sampling, reduces the sampling rate, whereas expansion, or up-sampling, fol- lowed by interpolation increases the sampling rate.

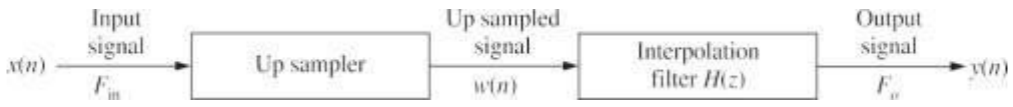
6. **What is the function of anti-imaging filter?**
   - The low-pass filter placed after the up sampler to remove the images created due to up sampling is called the anti-imaging filter.

7. **Mention applications of multi rate digital signal processing**
   - Musical sound processing, speech processing, audio signal processing, video and narrow band filtering

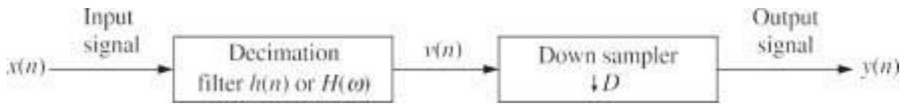
8. **What is subband coding?**
   - Decompose the input signal into different frequency bands. After the input is decomposed to its constituents, we can use the coding technique best suited to each constituent to improve the compression performance

9. **What is sampling rate conversion?**
   - The process of converting a given discrete-time signal at a given rate to a different rate

10. **Draw the block diagram of Interpolator?**



11. **Draw the block diagram of Decimator?**


$x(n)$
Input signal
Decimation filter $h(n)$ or $H(\omega)$
$v(n)$
Down sampler $\downarrow D$
Output signal
$y(n)$

EXP.NO. 9 **DATE:**

LINEAR CONVOLUTION USING DSP DEVELOPMENT KIT

AIM:

To perform linear convolution of the given two signals using TMS320C6748 DSP development kit-LCDK 6748.

x(n) =
y(n) =

SOFTWARE / HARDWARE REQUIREMENTS:

- Personal Computer
- Code composer studio
- TMS320C6748 DSP development kit - LCDK 6748

THEORY:

Linear Convolution

The response y[n] of a LTI system for any arbitrary input x[n] is given by convolution of impulse response h[n] of the system and the arbitrary input x[n].

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

If the input x[n] has N1 samples and impulse response h[n] has N2 samples then the output sequence y[n] will be a finite duration sequence consisting of (N1 + N2 - 1) samples. The convolution results in a non periodic sequence called Aperiodic convolution.

TMS320C6748 DSP development kit - LCDK 6748

The TMS320C6748 DSP development kit (LCDK) is a scalable platform for applications requiring embedded analytics and real-time signal processing, including biometric analytics, communications, and audio. The low-cost LCDK will also speed and ease your hardware development of real-time DSP applications. This new board reduces design work with freely downloadable and duplicable board schematics and design files. A wide variety of standard interfaces for connectivity and storage enable you to easily bring audio, video, and other signals onto the board. The LCDK does not have an onboard emulator. An external emulator from TI or a third-party will be required to start development.

Features

- Integrated floating-/fixed-point DSP with up to 456 MHz performance
- Software, expansion headers, schematics and application demos
- SDKs, DSP/BIOS RTOS, drivers, stacks and protocol, algorithm libraries, flash and boot utilities and StarterWare

FLOWCHART/ALGORITHM:

1. Open code composer studio
2. Open new CCS project
3. Select the device → lcdkc6748, select connection type → Texas Instrument XDS 100 v3 USB debug probe
4. Connect emulator with LCDK6748, also connect it to the PC, and power it on through the adapter provided.
5. Click Verify to check whether the connection is successful or not.
6. Give the project name, add compiler version 7.3.1, select the empty project with main.c, and click finish.
7. Enter the program.
8. Click build and verify whether verified successfully
9. Click debug, which connects the processor and the system. The code will be uploaded in LCDK6748.
10. Click resume, to run the code and execute.
11. The output can be verified with manual calculation.

PROGRAM:

```
#include<stdio.h>
int y[20];
main()
{
        int m=4;        /*Length of i/p samples sequence*/
        int n=4;          /*Length of impulse response Co-efficients */int i=0,j;
        // int x[15]={1,2,3,4,5,6,0,0,0,0,0,0};          /*Input Signal Samples*/
        // int h[15]={1,2,3,4,5,6,0,0,0,0,0,0};          /*Impulse Response Co-efficients*/int
        x[10]={1,2,3,4,0,0,0,0};
         int h[10]={1,2,1,2,0,0,0,0};
                for(i=0;i<m+n-1;i++)
                {
                        y[i]=0;
                                for(j=0;j<=i;j++)
                                [i]+=x[j]*h[i-j];
                }

                for(i=0;i<m+n-1;i++)
                printf("%d\n",y[i]);
 }
```

INFERENCE:

Linear convolution is a fundamental operation in signal processing that enables various important operations on signals, such as filtering, signal analysis, and system characterization. In this experiment, convolution can be learned.

RESULT:

VIVA QUESTIONS:

1. **Can Linear convolution be performed on a continuous-time signal?**
   - Linear convolution is typically performed on discrete-time signals. Continuous-time signals can be discretized to perform linear convolution in a discrete domain.

2. **What are some applications of linear convolution in signal processing?**
   - Linear convolution has various applications in signal processing, including filtering (low- pass, high-pass filters etc), system modeling, image processing, audio processing and linear time-invariant (LTI) system analysis.

3. **How does the length of the input signals affect the output of linear convolution?**
   - The length of the input signals affects the length of the output signal. The output length is the sum of the lengths of the input signals minus one.

4. **What is the LCDK6748 development kit used for?**
   - The LCDK6748 development kit is used for embedded systems development and prototyping. It provides a platform for designing and testing applications based on the Texas Instruments TMS320C6748digital signal processor (DSP)

5. **Are there any limitations or considerations to keep in mind when working with the LCDK6748?**
   - When working with the LCDK6748, it's important to consider factors such as power consumption, memory limitations, real-time constraints, and the need for external peripherals or modules. Careful design and optimization are required to ensure efficient and reliable operation.

EXP.NO. 10 **DATE:**

DISCRETE FOURIER TRANSFORM USING DSP DEVELOPMENT KIT

AIM:

To perform Discrete Fourier Transform of the given signal using TMS320C6748 DSP development kit-LCDK 6748.

x(n) =

SOFTWARE / HARDWARE REQUIREMENTS:

- Personal Computer
- Code composer studio
- TMS320C6748 DSP development kit - LCDK 6748

THEORY:

Discrete Fourier Transform

DFT is a computational tool that stands for Discrete Fourier Transform. To convert a time- domain discrete signal to its equivalent frequency domain response, DFT is used. DFT is the better version of DTFT as problems that occur in DTFT are rectified in DFT. The DFT of a signal x(n) is given as follows,

$$X[k] = \sum_{n=0}^{N-1} x[n]\, e^{\frac{-j2\pi kn}{N}} \qquad where\ k = 0,1,2, \ldots \ldots . N-1$$

The range of a DFT sequence is finite. The range is from 0 to N-1. A DFT sequence contains only positive frequencies. A DFT sequence provides lessnumber of frequency components as compared to DTFT.

TMS320C6748 DSP development kit - LCDK 6748

The TMS320C6748 DSP development kit (LCDK) is a scalable platform for applications requiring embedded analytics and real-time signal processing, including biometric analytics, communications, and audio. The low-cost LCDK will also speed and ease your hardware development of real-time DSP applications. This new board reduces design work with freely downloadable and duplicable board schematics and design files. A wide variety of standard interfaces for connectivity and storage enable you to easily bring audio, video, and other signals onto the board.

The LCDK does not have an onboard emulator. An external emulator from TI or a third-party will be required to start development.

Features

- Integrated floating-/fixed-point DSP with up to 456 MHz performance
- Software, expansion headers, schematics and application demos
- SDKs, DSP/BIOS RTOS, drivers, stacks and protocol, algorithm libraries, flash and boot utilities and StarterWare

FLOWCHART/ALGORITHM:

1. Open code composer studio
2. Open new CCS project
3. Select the device → lcdkc6748, select connection type → Texas Instrument XDS 100 v3 USB debug probe
4. Connect emulator with LCDK6748, also connect it to the PC, and power it on through the adapter provided.
5. Click Verify to check whether the connection is successful or not.
6. Give the project name, add compiler version 7.3.1, select the empty project with main.c, and click finish.
7. Enter the program.
8. Click build and verify whether verified successfully
9. Click debug, which connects the processor and the system. The code will be uploaded in LCDK6748.
10. Click resume, to run the code and execute.
11. The output can be verified with manual calculation.

PROGRAM:

```
#include "stdio.h"
#include "math.h"
float real[4]; float img[4];
float reali[4]; float imgi[4];
#define pi 3.14159
void main()
{
        int x[4]={1,1,1,1};
        int j;
        printf("\nThe DFT is as follows:");
        for (j=0;j<4;j++)
        {
                float realsum=0,imgsum=0;float cospart=0,sinpart=0; int i=0;
                for (i=0;i<4;i++)
                {
                        cospart=cos((2*pi*j*i)/4); sinpart=sin((2*pi*j*i)/4);
                        realsum=realsum+x[i]*cospart; imgsum=imgsum-x[i]*sinpart;
                }
                real[j]=realsum;
                img[j]=imgsum;
                printf("\nX[%d]=%f+(%f)j",j,realsum,imgsum);
        }
}
```

INFERENCE:

The DFT is calculated for a set of N equally spaced discrete frequencies. Hence, the sequence X(k) is called the Discrete Fourier Transform of the sequence x(n). The application of DFT is more significant nowadays as it provides the user with an increase in computational accuracy without any additional increase in computing time.

RESULT:

VIVA QUESTIONS:

1. **In which situations would you prefer a Discrete Time Fourier Transform over a Discrete Fourier Transform?**
   - The Discrete Time Fourier Transform is better suited for signals that are sampled at regular intervals, while the Discrete Fourier Transform is better for signals that are not sampled at regular intervals.

2. **What is the difference between microprocessor and DSP?**
   - Microprocessors are usually generic processor used in desktops, laptops etc. DSP are special processor designed for specific complex applications such as used in mobile, smart phone, tablets etc. Most of the DSPs will have special instruction set which takes care of complex signal processing in very less time and with the use of very small memory. Popular DSP vendors are Texas (TMS320), Qualcomm, LSI, Tensilica etc.

3. **Why DFT is used in signal processing?**
   - The DFT is the sampled Fourier Transform and therefore does not contain all frequencies forming an image, but only a set of samples which is large enough to fully describe the spatial domain image.

4. **What are the applications of LCDK6748 development kit?**
   - The TMS320C6748 DSP development kit (LCDK) is a scalable platform that breaks down development barriers for applications that require embedded analytics and real-time signal processing, including biometric analytics, communications and audio.

5. **What are the limitations of LCDK6748 kit?**
   - The LCDK does not have an onboard emulator. An external emulator from TI or a third- party will be required to start development.

EXP.NO. 11 **DATE:**

FIR FILTER DESIGN USING DSP DEVELOPMENT KIT

AIM:

To design an FIR filter using TMS320C6748 DSP development kit-LCDK 6748.

SOFTWARE / HARDWARE REQUIREMENTS:

- Personal Computer
- Code composer studio
- TMS320C6748 DSP development kit - LCDK 6748

THEORY:

FIR filters

FIR (Finite Impulse Response) filters are fundamental tools in Digital Signal Processing (DSP) used for various signal processing tasks, including filtering, equalization, and convolution. These filters have a finite duration impulse response, making them computationally efficient and easy to implement.

Windowing is a crucial aspect of designing FIR filters. It involves applying a window function to the filter's impulse response to control its characteristics. Common window types include Hamming, Hanning, Blackman, and rectangular windows. Each window type has specific properties, affecting the filter's performance. For example, the Hamming window provides good trade-offs between main lobe width and side lobe suppression, while the rectangular window is simple but has poor side lobe characteristics.

TMS320C6748 DSP development kit - LCDK 6748

The TMS320C6748 DSP development kit (LCDK) is a scalable platform for applications requiring embedded analytics and real-time signal processing, including biometric analytics, communications, and audio. The low-cost LCDK will also speed and ease your hardware development of real-time DSP applications. This new board reduces design work with freely downloadable and duplicable board schematics and design files. A wide variety of standard interfaces for connectivity and storage enable you to easily bring audio, video, and other signals onto the board.

The LCDK does not have an onboard emulator. An external emulator from TI or a third-party will be required to start development.

Features

- Integrated floating-/fixed-point DSP with up to 456 MHz performance
- Software, expansion headers, schematics and application demos
- SDKs, DSP/BIOS RTOS, drivers, stacks and protocol, algorithm libraries, flash and boot utilities and StarterWare.

FLOWCHART /ALGORITHM:

1. Open code composer studio
2. Open new CCS project
3. Select the device lcdkc6748, select connection type Texas Instrument XDS 100 v3 USB debug probe
4. Connect emulator with LCDK6748, also connect it to the PC, and power it on through the adapter provided.
5. Click Verify to check whether the connection is successful or not.
6. Give the project name, add compiler version 7.3.1, select the empty project with main.c, and click finish.
7. Right click the project folder ------> add files -----> select all files to include in the path(.ccxml, linker_dsp, .asm,etc.,) --------------- > ok
8. Right click the project folder ------> properties -----> process -----> select the files to include-----> ok
9. The filter coefficients must be generated using MATLAB FDA Tool.
10. Open MATLAB Command window, enter fda tool to open Filter Design and Analysis Tool window with multiple selection options.
11. The following selections must be done

Response type	----->	LPF
Specify the Order	----->	10
Design Method	----->	FIR Window---> kaiser
Freq.Specifications	----->	Fs= 8000 & Fc = 3000

12. Click Design, to obtain the frequency response of the filter for the given specifications.
13. Select File----> export----> export as coefficients ---->overwrite variables(to be selected) ----- > export.
14. In command window enter dsk_sos_fir67int(SOS,G)

Enter the file name of coefficients: ***FIR_MAT (file name to be given by user)***

15. The filter coefficients will stored in MATLAB with the above file name. Open FIR_MAT and copy the coefficients.
16. Now go to CCS and paste these coefficients and enter the FIR filter design program in main.c
17. Click build and verify whether verified successfully
18. Click debug, which connects the processor and the system. The code will be uploaded in LCDK6748.
19. Click resume, to run the code and execute.
20. To obtain the output ------> right click project --------> compare with --------> general -------- >

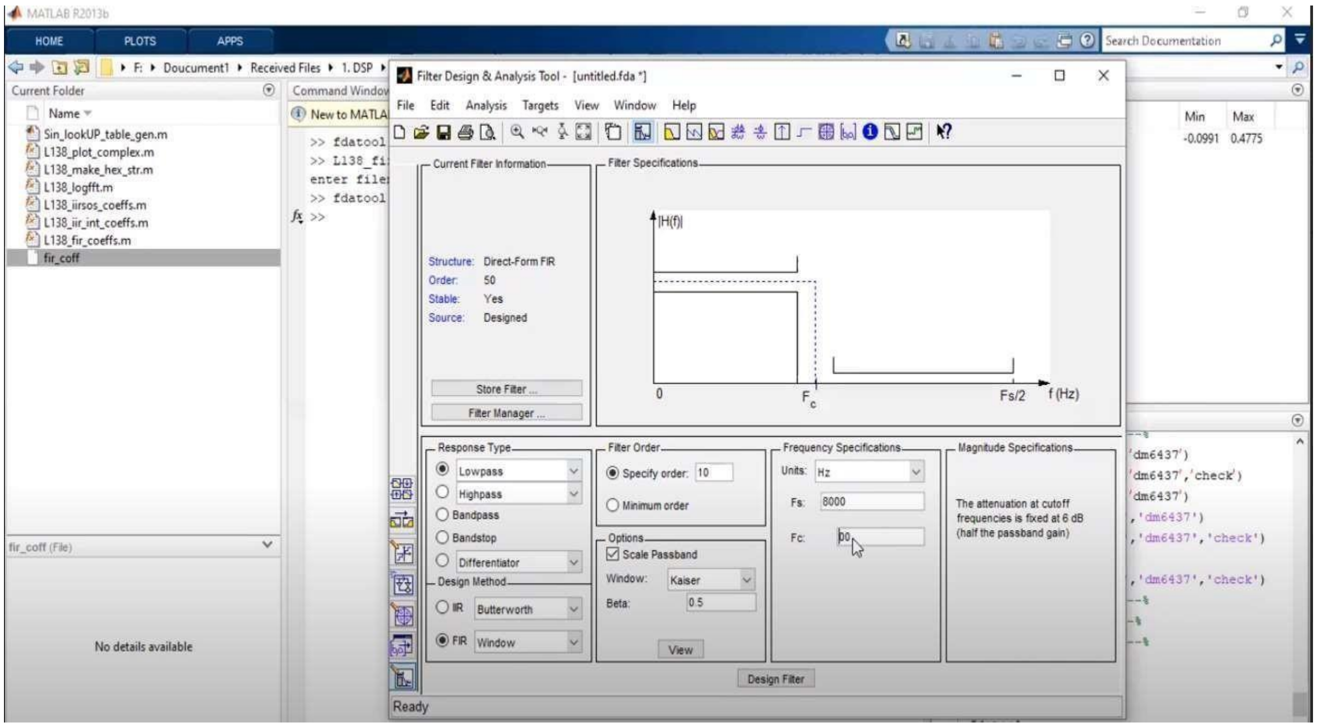
output format -----> legacy off & runwith support library --------> rts6740.lib , to be selected.

21. To view the real time output select the digital oscilloscope in the PC select the frequency less than Fc, the it shows the proper sine wave out, but when the frequency is selected above the Fc the frequency will be suppressed.

PROGRAM:

```
  #include "L138_LCDK_aic3106_init.h"
  #define N 56
  float h[56] = {
     0.005731300022957,     -0.01442286918098,     -0.01505281088785,    0.006516298605502,
    -0.006820391943361,      0.01726259588786,      0.01813058958444,-   0.007903310090601,
     0.00833572238705,      -0.02127766284377,     -0.02255938150832,    0.009938091263959,
    -0.01060670962792,       0.02743935312318,      0.02953864172628,   -0.01324188643609,
     0.0144212321379,       -0.03819937004057,     -0.04228837100782,    0.01960553208679,
    -0.02224819336594,       0.06204498753368,      0.07339656717882,   -0.03718765018171,
     0.04784338581879,      -0.1617836373006,      -0.2697258865447,     0.3352261062289,
     0.3352261062289,       -0.2697258865447,      -0.1617836373006,     0.04784338581879,
    -0.03718765018171,       0.07339656717882,      0.06204498753368,   -0.02224819336594,
     0.01960553208679,      -0.04228837100782,     -0.03819937004057,    0.0144212321379,
    -0.01324188643609,       0.02953864172628,      0.02743935312318,   -0.01060670962792,
     0.009938091263959,     -0.02255938150832,     -0.02127766284377,    0.00833572238705,
    -0.007903310090601,      0.01813058958444,      0.01726259588786,   -0.006820391943361,
     0.006516298605502,     -0.01505281088785,     -0.01442286918098,    0.005731300022957
  };
  float x[N]; // filter delay lineinterrupt void interrupt4(void)
  {
  short i;
  float yn = 0.0;
  x[0] = (float)(input_left_sample()); // input from ADC
for (i=0 ; i<N ; i++)                    // compute filter outputyn += h[i]*x[i];
for (i=(N-1) ; i>0 ; i--)                // shift delay linex[i] = x[i-1];
  output_left_sample((uint16_t)(yn));  // output to DACreturn;
  }
  int main(void)
  {
  L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_IN
  PUT);
  while(1);
  }
```

Sample output: Coefficient generation in MATLAB



```
>> fdatool
>> L138_fi
enter file
>> fdatool
>>
```

Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR Window]

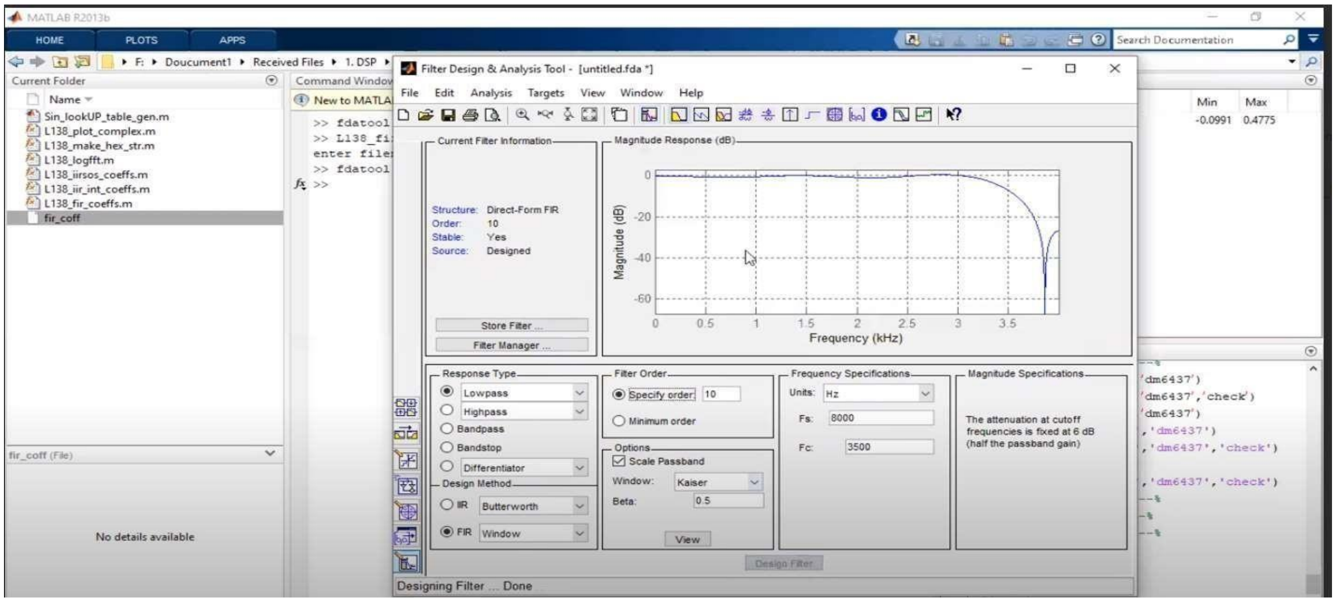
Filter Order: [✓ Specify order: 10] Minimum order

Options: ☑ Scale Passband; Window: Kaiser; Beta: 0.5

Frequency Specifications: Units: Hz; Fs: 8000; Fc: 00

Magnitude Specifications: The attenuation at cutoff frequencies is fixed at 6 dB (half the passband gain)

```
'dm6437')
'dm6437','check')
'dm6437')
,'dm6437')
,'dm6437','check')

,'dm6437','check')
--%
-%
--%
```



```
>> fdatool
>> L138_fi
enter file
>> fdatool
>>
```

Structure: Direct-Form FIR
Order: 10
Stable: Yes
Source: Designed

Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator

Design Method: IIR Butterworth [✓ FIR Window]

Filter Order: [✓ Specify order: 10] Minimum order

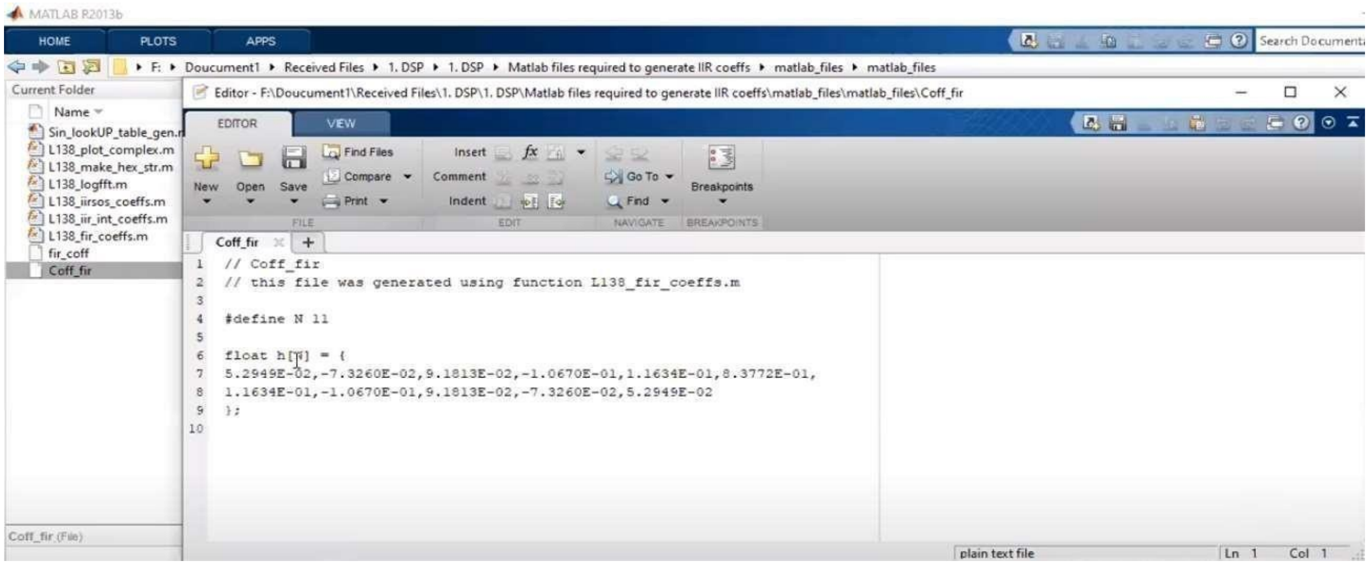
Options: ☑ Scale Passband; Window: Kaiser; Beta: 0.5

Frequency Specifications: Units: Hz; Fs: 8000; Fc: 3500

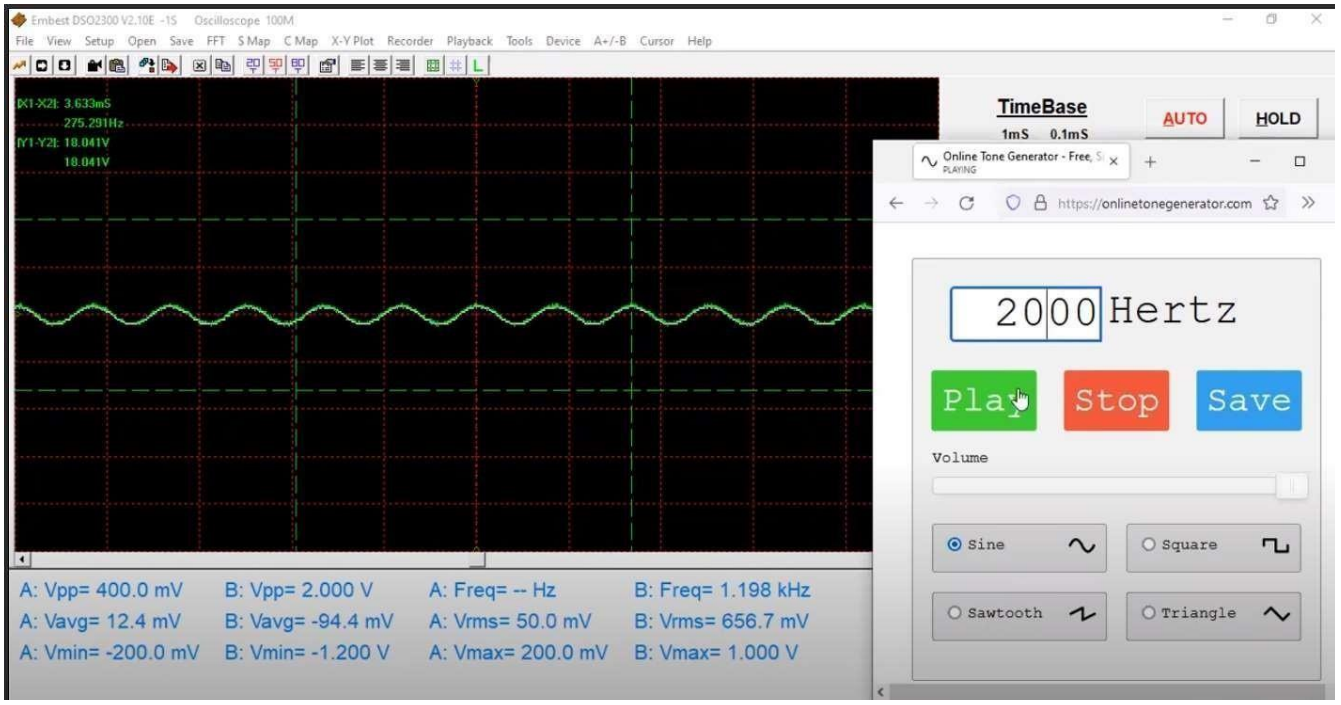
Magnitude Specifications: The attenuation at cutoff frequencies is fixed at 6 dB (half the passband gain)

```
'dm6437')
'dm6437','check')
'dm6437')
,'dm6437')
,'dm6437','check')

,'dm6437','check')
--%
-%
--%
```

Coefficients Generated



```
// Coff_fir
// this file was generated using function L138_fir_coeffs.m

#define N 11

float h[N] = {
5.2949E-02,-7.3260E-02,9.1813E-02,-1.0670E-01,1.1634E-01,8.3772E-01,
1.1634E-01,-1.0670E-01,9.1813E-02,-7.3260E-02,5.2949E-02
};
```



```
[X1-X2]: 3.633mS
         275.291Hz
[Y1-Y2]: 18.041V
         18.041V

A: Vpp= 400.0 mV    B: Vpp= 2.000 V      A: Freq= -- Hz       B: Freq= 1.198 kHz
A: Vavg= 12.4 mV    B: Vavg= -94.4 mV    A: Vrms= 50.0 mV     B: Vrms= 656.7 mV
A: Vmin= -200.0 mV  B: Vmin= -1.200 V    A: Vmax= 200.0 mV    B: Vmax= 1.000 V
```

2000 Hertz

Sine [✓] Square Sawtooth Triangle

RESULT:

EXP.NO. 12 **DATE:**

IIR FILTER DESIGN USING DSP DEVELOPMENT KIT

AIM:

To design an IIR filter using TMS320C6748 DSP development kit-LCDK 6748.

SOFTWARE / HARDWARE REQUIREMENTS:

- Personal Computer
- Code composer studio
- TMS320C6748 DSP development kit - LCDK 6748

THEORY:

IIR filters

IIR (Infinite Impulse Response) filters are a class of digital signal processing (DSP) filters used for various applications, including signal smoothing and frequency selective processing. They employ feedback, allowing output to depend on past inputs. Two main types of IIR filters exist:

Butterworth Filter: Provides a maximally flat frequency response in the passband while rolling off towards the stopband. It's commonly used for applications where a smooth frequency response is crucial.

Chebyshev Filter: Offers improved selectivity but introduces ripple in the passband. There are Type I (ripple in the passband) and Type II (ripple in the stopband) Chebyshev filters. IIR filters are computationally efficient but can be prone to instability in some cases due to feedback.

TMS320C6748 DSP development kit - LCDK 6748

The TMS320C6748 DSP development kit (LCDK) is a scalable platform for applications requiring embedded analytics and real-time signal processing, including biometric analytics, communications, and audio. The low-cost LCDK will also speed and ease your hardware development of real-time DSP applications. This new board reduces design work with freely downloadable and duplicable board schematics and design files. A wide variety of standard interfaces for connectivity and storage enable you to easily bring audio, video, and other signals onto the board.

The LCDK does not have an onboard emulator. An external emulator from TI or a third-party will be required to start development.

Features

- Integrated floating-/fixed-point DSP with up to 456 MHz performance
- Software, expansion headers, schematics and application demos
- SDKs, DSP/BIOS RTOS, drivers, stacks and protocol, algorithm libraries, flash and boot utilities and StarterWare.

FLOWCHART /ALGORITHM:

1. Open code composer studio
2. Open new CCS project
3. Select the device lcdkc6748, select connection type Texas Instrument XDS 100 v3 USB debug probe
4. Connect emulator with LCDK6748, also connect it to the PC, and power it on through the adapter provided.
5. Click Verify to check whether the connection is successful or not.
6. Give the project name, add compiler version 7.3.1, select the empty project with main.c, and click finish.
7. Right click the project folder ------> add files -----> select all files to include in the path(.ccxml, linker_dsp, .asm,etc.,)----------------> ok
8. Right click the project folder ------> properties -----> process -----> select the files to include-----> ok
9. The filter coefficients must be generated using MATLAB FDA Tool.
10. Open MATLAB Command window, enter fda tool to open Filter Design and Analysis Tool window with multiple selection options.
11. The following selections must be done

Response type	----->	LPF
Specify the Order	----->	10
Design Method	----->	FIR Window -- > kaiser
Freq.Specifications	----->	Fs= 8000 & Fc = 3000

12. Click Design, to obtain the frequency response of the filter for the given specifications.
13. Select File----> export----> export as coefficients ---->overwrite variables(to be selected) ------> export.
14. In command window enter dsk_sos_fir67int(SOS,G)

Enter the file name of coefficients: ***FIR_MAT (file name to be given by user)***

15. The filter coefficients will stored in MATLAB with the above file name. Open FIR_MAT and copy the coefficients.
16. Now go to CCS and paste these coefficients and enter the FIR filter design program in main.c
17. Click build and verify whether verified successfully
18. Click debug, which connects the processor and the system. The code will be uploaded in LCDK6748.
19. Click resume, to run the code and execute.
20. To obtain the output ------> right click project --------> compare with --------> general -------->

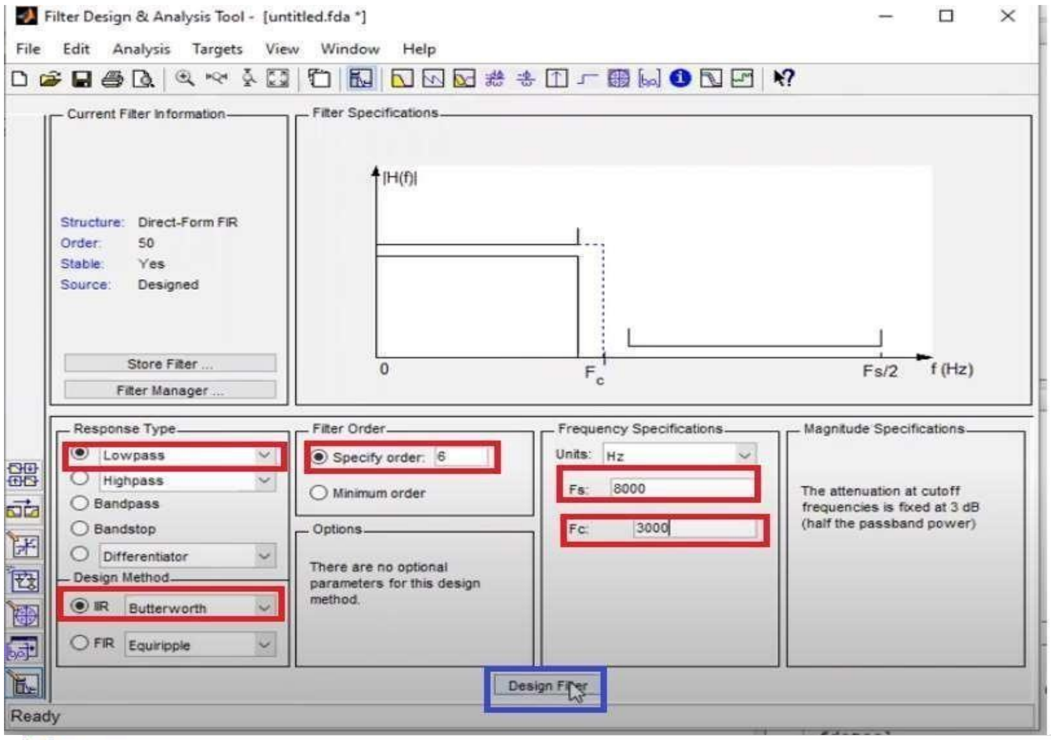
output format -----> legacy off & run with support library > rts6740.lib , to be selected.

21. To view the real time output select the digital oscilloscope in the PC select the frequency less than Fc, the it shows the proper sine wave out, but when the frequency is selected above the Fc the frequency will be suppressed.

PROGRAM:

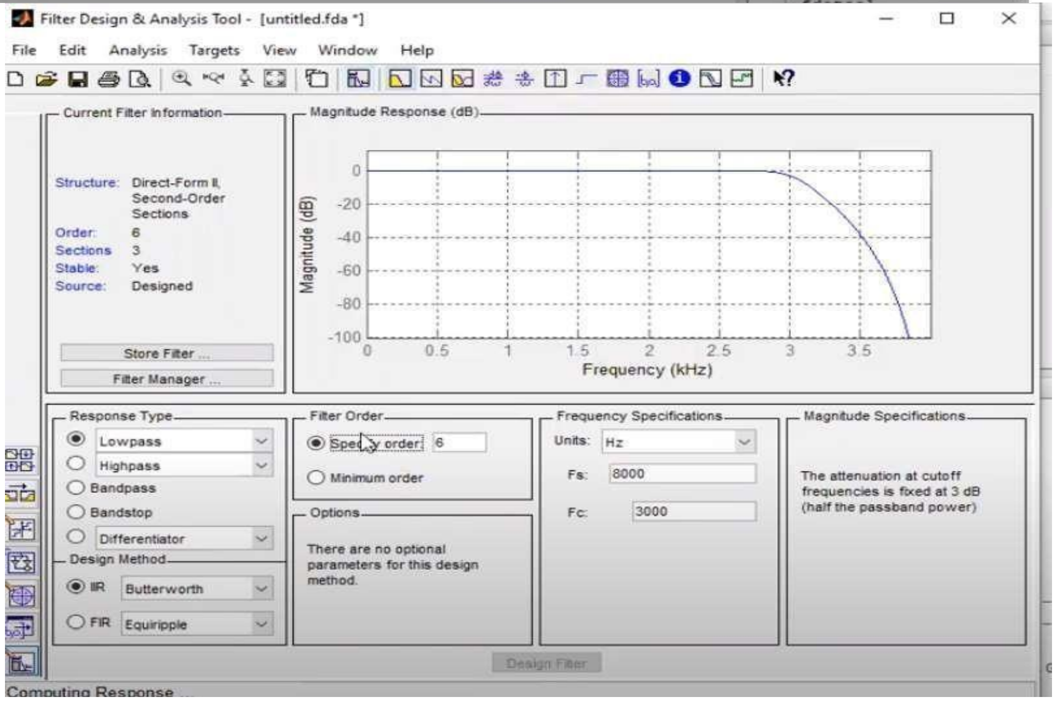
```
// L138_iir_intr.c
// IIR filter implemented using second order sections
// integer coefficients read from file

#include "L138_LCDK_aic3106_init.h"
// fpass = 2000, fstop = 3000, fs = 8000#define NUM_SECTIONS 6

int b[NUM_SECTIONS][3] = {{13216, 26432, 13216}, {11366, 22733, 11366},
{10201, 20401, 10201}, {9511, 19022, 9511}, {9189, 18379, 9189} };

int a[NUM_SECTIONS][2] = {{6576, 25245}, {5382, 14714}, {4629, 8069}, {4154, 3880},
{3873, 1402}, {3742, 247} };

// L138_iirsos_intr.c
// IIR filter implemented using second order sections

int w[NUM_SECTIONS][2] = {0};
interrupt void interrupt4()   //interrupt service routine
{
int section;                  // index for section numberint input; // input to each section
int wn,yn;                    // intermediate and output values in each stage

input = input_left_sample();
for (section=0 ; section< NUM_SECTIONS ; section++)
{
wn = input - ((a[section][1]*w[section][0])>>15) - ((a[section][1]*w[section][1])>>15);
yn = ((b[section][0]*wn)>>15) + ((b[section][1]*w[section][0])>>15) +
((b[section][2]*w[section][1])>>15);
w[section][1] = w[section][0];
w[section][0] = wn;
input = yn;                   // output of current section will be input to next
}
output_left_sample((int16_t)(yn)); // before writing to codec
return;                                    //return from ISR
}
int main(void)
{
L138_initialise_intr(FS_8000_HZ,ADC_GAIN_0DB,DAC_ATTEN_0DB,LCDK_LINE_INPU
T);
while(1);
} // end of main()
// end of main()
```

<u>Sample output: Coefficient generation in MATLAB</u>


Filter Design & Analysis Tool - [untitled.fda *]
File Edit Analysis Targets View Window Help
Store Filter ...
Filter Manager ...
|H(f)|
0
Fc
Fs/2
f (Hz)
Design Filter
Ready

Current Filter Information
Structure: Direct-Form FIR
Order: 50
Stable: Yes
Source: Designed

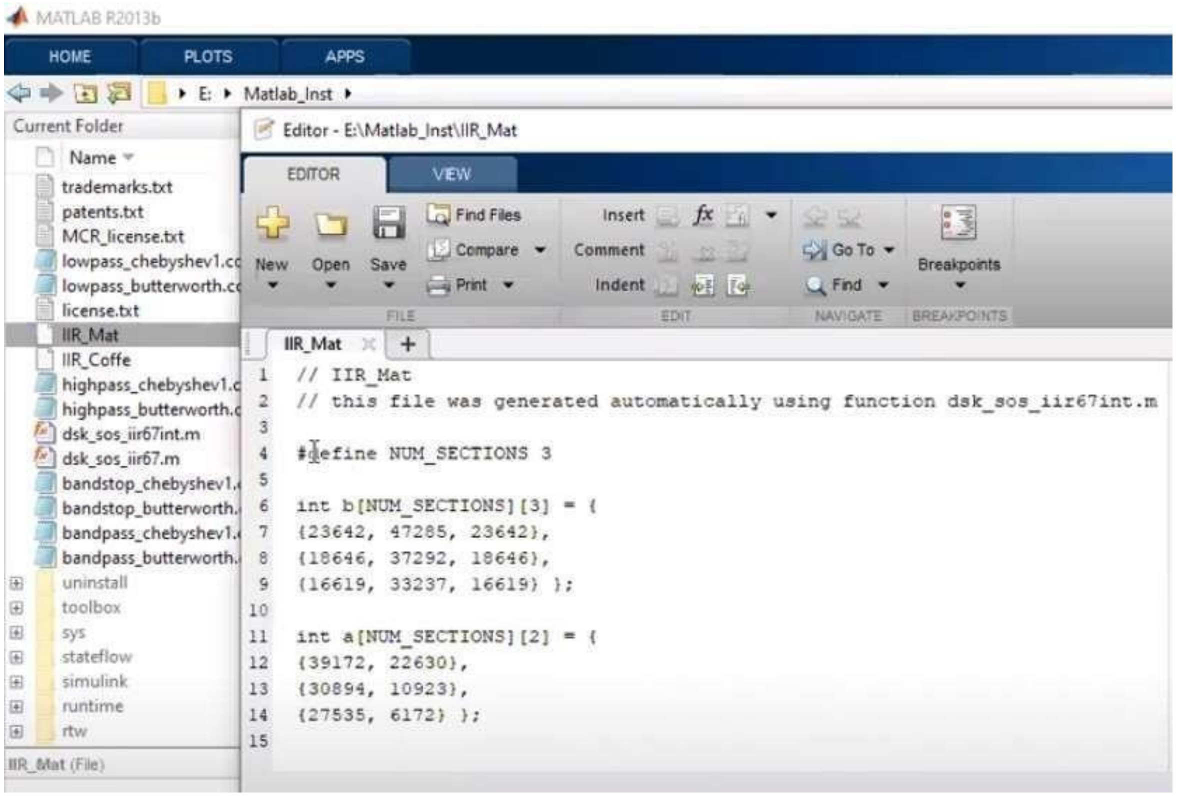
Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: [✓ IIR Butterworth] FIR Equiripple
Filter Order: [✓ Specify order: 6] Minimum order
Options: There are no optional parameters for this design method.
Frequency Specifications: Units: Hz; Fs: 8000; Fc: 3000
Magnitude Specifications: The attenuation at cutoff frequencies is fixed at 3 dB (half the passband power)



Current Filter Information
Structure: Direct-Form II, Second-Order Sections
Order: 6
Sections: 3
Stable: Yes
Source: Designed

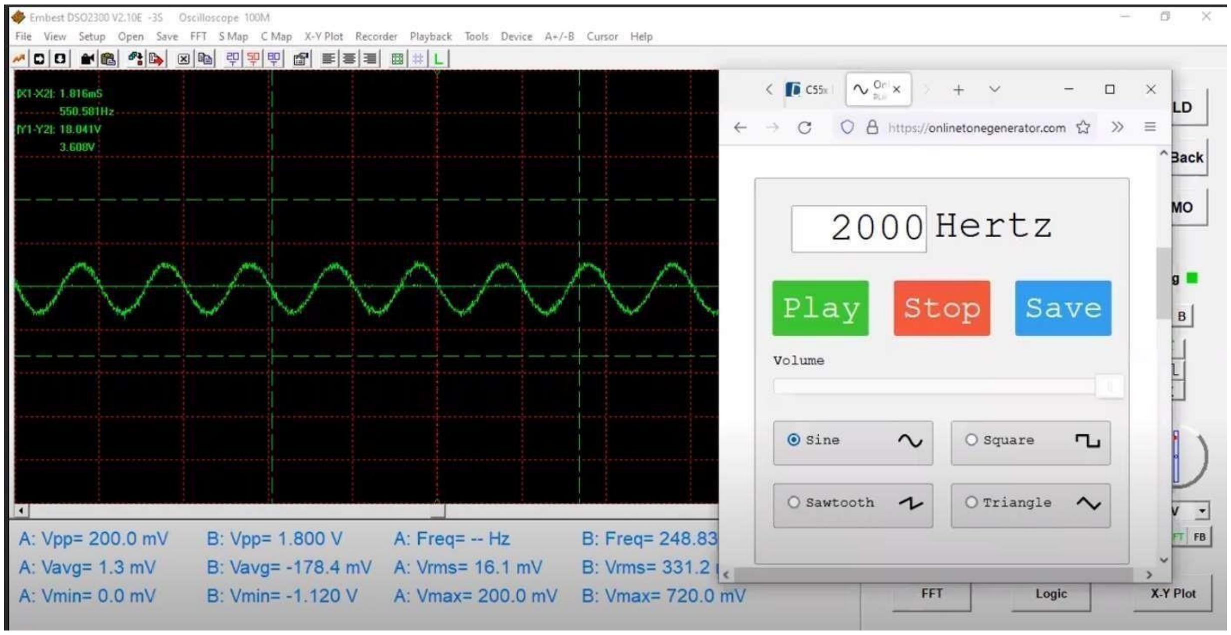
Response Type: [✓ Lowpass] Highpass Bandpass Bandstop Differentiator
Design Method: [✓ IIR Butterworth] FIR Equiripple
Filter Order: [✓ Specify order: 6] Minimum order
Options: There are no optional parameters for this design method.
Frequency Specifications: Units: Hz; Fs: 8000; Fc: 3000
Magnitude Specifications: The attenuation at cutoff frequencies is fixed at 3 dB (half the passband power)

Coefficients Generated



```
// IIR_Mat
// this file was generated automatically using function dsk_sos_iir67int.m

#define NUM_SECTIONS 3

int b[NUM_SECTIONS][3] = {
{23642, 47285, 23642},
{18646, 37292, 18646},
{16619, 33237, 16619} };

int a[NUM_SECTIONS][2] = {
{39172, 22630},
{30894, 10923},
{27535, 6172} };
```

Output:


Embest DSO2300 V2.10E -3S Oscilloscope 100M
File View Setup Open Save FFT S Map C Map X-Y Plot Recorder Playback Tools Device A+/-B Cursor Help
https://onlinetonegenerator.com
Play
Stop
Save
Volume
Sine
Square
Sawtooth
Triangle
LD
Back
MO
FFT
Logic
X-Y Plot

```
|X1-X2|: 1.816mS
         550.581Hz
|Y1-Y2|: 18.041V
         3.608V
```

2000 Hertz

```
A: Vpp= 200.0 mV    B: Vpp= 1.800 V      A: Freq= -- Hz        B: Freq= 248.83
A: Vavg= 1.3 mV     B: Vavg= -178.4 mV   A: Vrms= 16.1 mV      B: Vrms= 331.2
A: Vmin= 0.0 mV     B: Vmin= -1.120 V    A: Vmax= 200.0 mV     B: Vmax= 720.0 mV
```

RESULT: