INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal - 500 043, Hyderabad, Telangana

COURSE CONTENT

DIGITAL SIGNAL PROCESSING LABORATORY								
VI Semester: ECE								
Course Code	**Category**	**Hours / Week**			**Credits**	**Maximum Marks**		
AECC42	**Core**	**L**	**T**	**P**	**C**	**CIA**	**SEE**	**Total**
		0	0	4	2	30	70	100
Contact Classes: Nil	**Tutorial Classes: Nil**	**Practical Classes: 45**				**Total Classes: 45**		
Prerequisite: Signals and Systems								

I. COURSE OVERVIEW:
This laboratory course is concerned with the implementation of digital signal processing algorithms using different computational platforms such as MATLAB and DSP tools that give core knowledge to develop the real time applications in DSP. It focuses on the convolution, discrete Fourier trans-form, fast Fourier transform algorithms, digital filter design and multi rate signal processing. Digital signal processing applications are used in speech processing, image processing, audio and video data compression, and communication systems.

II.COURSE OBJECTIVES:
The students will try to learn:

I. The behavior of discrete time signals and systems in time and frequency domain.
II. Analysis of IIR, FIR digital filters and multi rate signal processing systems.
III. The implementation of real time digital signal processing algorithms using MATLAB tool and TI TMSC67XX target board.

III. COURSE OUTCOMES:
At the end of the course students will be able to:

CO1	Develop the various convolution sum methods for filtering long duration sequences efficiently in MATLAB.
CO2	Apply discrete Fourier transform for spectral analysis of discrete signals.
CO3	Compare the magnitude and phase characteristics of IIR digital filter using Butterworth method and Chebyshev methods.
CO4	Build Nth order FIR digital filters using windows and frequency sampling methods.
CO5	Utilize the Goertzel algorithm for the generation and detection of dual-tone multi-frequency (DTMF) signaling.
CO6	Apply multi-rate signal processing methods such as decimation and interpolation for interfacing the digital systems with different sampling rates.

IV.. COURSE CONTENT:

EXERCISES FOR DIGITAL SIGNAL PROCESSINGLABORATORY

1. Getting Started Exercises

To be proficient in MATLAB programming, students need to be able to do:

1. Introduction to MATLAB
2. Generation of various discrete-time signals or sequences
3. Generating Sinusoidal Sequence
4. Generating Unit step Sequence
5. Generating Exponential Sequence
6. Verification of Sampling Theorem

2. Exercises on Time and Frequency response of an LTI System

To be proficient in MATLAB programming, students need to be able to compute response of a system.

1. Time Response of an LTI System
2. Frequency Response of an LTI System

3. Exercises on Discrete-Time Systems

To be proficient in MATLAB programming, students be able to compute:

1. Linear Convolution
2. Circular Convolution
3. Cyclic Convolution
4. Overlap-Add Method to compute linear convolution
5. Overlap-Save Method to compute linear convolution

4. Exercises on DFT and IDFT

To be proficient in MATLAB programming, students be able to compute:

1. Exercises on DFT
2. Exercises on IDFT

5. Exercises on FIR and IIR Filter Design

To be proficient in MATLAB programming, students need to design and implement FIR and IIR Filters.

1. FIR filter design (LPF,HPF,BPF).
2. IIR filter design (Butterworth and Chebyshev).

6. Exercises on Optimum Equiripple FIR Digital Filters

To be proficient in MATLAB programming, students need to design and implement the following Optimum Equiripple FIR digital filters.

1. Equiripple FIR Halfband Filter
2. Equiripple FIR Halfband Interpolator

7. Exercises on DIT-FFT and DIF-FFT algorithms

To be proficient in MATLAB programming, students need to design and implement the following:
1. DIT-FFT algorithm
2. DIF-FFT algorithm

8. Exercises on FIR Digital Filter Using Window Method

To be proficient in MATLAB programming, students need to design and implement the following:

1. FIR Digital Filter Design using Rectangular Window
2. FIR Digital Filter Design using Hanning Window

9. Exercises on IIR Digital Filter using Butterworth and Bilinear Transformation

To be proficient in MATLAB programming, students need to design and implement the following:

1. Designing a Butterworth Low-pass IIR Filter
2. Designing a Chebyshev Type I and bilinear transformation IIR Filter
3. Designing a Chebyshev Type II and bilinear transformation IIR Filter

10. Exercises on DTMF Tone Generation and Detection

To be proficient in MATLAB programming, students need to design and implement the following:

1. DTMF Tone Detection Using Goertzel Algorithm

11. Exercises on Sampling Rate Conversion

To be proficient in MATLAB programming, students need to perform Implementing sampling rate conversion by decimation, interpolation, and a rational factor.

1. Interpolation for Non-Integer Sampling rate

12. Exercises on Sine Wave Generation

To be proficient in MATLAB programming, students need to design and implement the following:
1. Generation of sine wave using a lookup table
2. FIR Filter using Frequency Sampling Method

13. Exercises on TMS 320 C6713 DSK - Code composer studio –Digital Filters

To be proficient in MATLAB programming and DSP processor, students need to design and implement the following:
1. Implementation of LP and HP FIR Filter For Given Sequence

14. Exercises on IIR and FIR Filters Using DSP Kits.

To be proficient in MATLAB programming, students need to design and observe the characteristics of IIR and FIR filters using DSKC6713 processor .

1. DSP applications: Implantation of Decimation process and Interpolation Process.

EXERCISES FOR DIGITAL SIGNAL PROCESSING LABORATORY

Note: Students are encouraged to bring their own laptops for laboratory practice sessions.

1. Getting Started Exercises

Overview of Digital Signal Processing with Introduction to MATLAB

A Digital Signal Processing (DSP) system is designed to manipulate digital signals in order to improve or modify them in some way. The core purpose of DSP systems is to filter, analyze, and transform signals using digital computation. Common applications of DSP systems include audio and speech processing, radar and sonar, telecommunications, and image processing. The flexibility and efficiency of DSP systems come from their ability to apply complex mathematical operations to signals in real-time or near real-time.

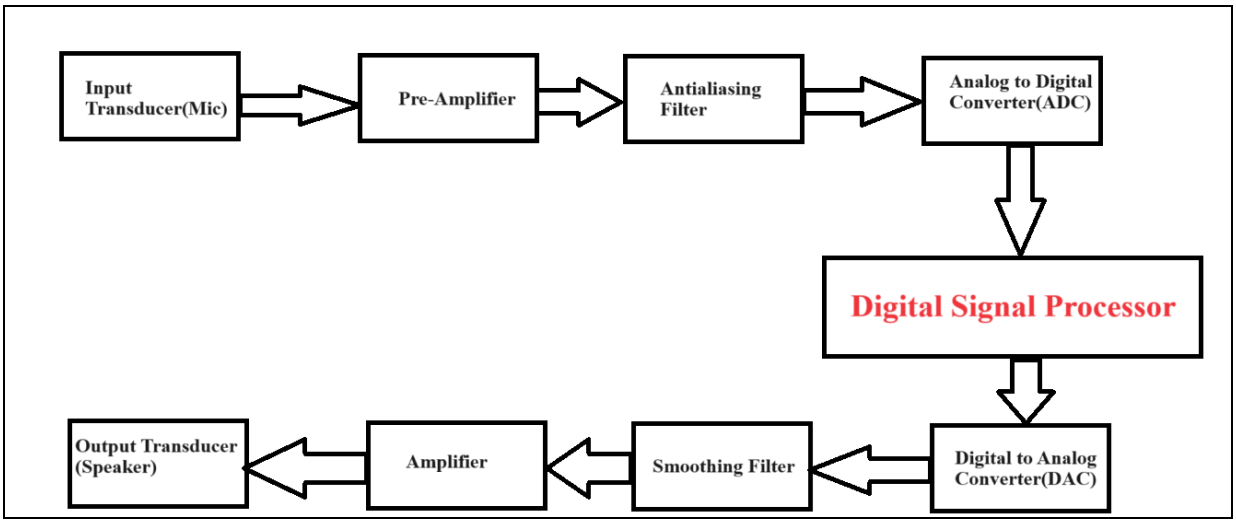

Figure 1.0 Block diagram of a DSP System

1.1 Introduction to MATLAB:

MATLAB is a high-level programming language and environment designed primarily for numerical computing, data analysis, and visualization. It is widely used in academia, research, and industry for tasks ranging from mathematical modeling to algorithm development and simulation.

Basic Syntax:

```
% MATLAB script example
A = [1 2; 3 4];      % Define a matrix.
b = [5; 6];          % Define a vector.
x = A \ b;           % Solve the system of linear equations.
disp(x);             % Display the result.
```

In summary, MATLAB is a versatile tool for scientific computing and engineering applications. Its strengths lie in its ease of use, rich set of functions, and powerful matrix-based operations, making it a preferred choice for professionals and researchers in various fields.

1.2 Generation of various Discrete-time signals or sequences

Generation of various discrete-time sequences using MATLAB.

1.3 Generating Sinusoidal Sequence

Generating a sinusoidal sequence in MATLAB involves defining the amplitude, frequency, phase, sampling frequency and number of samples.

Hint

```
% Parameters
A = 1; % Amplitude
f = 50; % Frequency in Hz
phi = 0; % Phase shift in radians
Fs = 1000; % Sampling frequency in Hz
N = Fs * T
------
```

Try:

1. Generate a sinusoidal sequence with the following parameters A=2, f = 1000 Hz, phi = -90 deg, Fs= 10000 Hz, N = 65536.
2. Generate a sinusoidal sequence with the following parameters A=0.5, f = 10 Hz, phi = 0 deg, Fs= 2000 Hz, N = 2048.

1.4 Generating Unit step Sequence

In MATLAB, to generate a unit step sequence typically needs to define length of the sequence (N) and starting index (n_0).

Hint

```
% Parameters
N = 10; % Length of the sequence
n0 = 0; % Starting index where the step occurs
--
```

Try:

1. Generate a unit step sequence with the N=2000, n_0=10.
2. Generate a unit step sequence with the N=65536, n_0=0.

1.5 Generating Exponential Sequence

To generate an exponential sequence in MATLAB, the required parameters are amplitude, exponential rate, number of samples and sample index array.

Hint

```
% Parameters
```

```
A=2;
alpha = -0.1;
N=50;
---
---
```

Try:

1. Generate an exponential decay sequence x[n] = A(0.6)^n, with A = 10, n = -10:1:10.
2. Generate a damped sinusoidal sequence with the following parameters n = 10, A= 1, r (decay rate)= 0.5, f = 50 Hz, phi(ϕ)=0. (Hint: Use the formula: x[n]=A·(0.5)^n·sin(2πfn+ϕ)).
3. Generate a rectangular sequence: length and amplitude – users' choice.
4. Plot complex exponential sequence of user's choice.
5. Plot the signum sequence: length and amplitude- user's choice.

1.6 Verification of the Sampling Theorem

Verifying the sampling theorem involves demonstrating that a continuous signal can be perfectly reconstructed from its samples if it is sampled at a rate higher than twice its highest frequency component (the Nyquist rate).

Input: Input signal frequency (Fm) = 1000 Hz

Sampling frequency (Fs) = 500 Hz, 1000 Hz, 4000 Hz

Hint

```
% Define parameters
f = 100; % Frequency of the sine wave
T = 1/f; % Period of the sine wave
Fs1 = 500; % Sampling frequency below Nyquist rate (2*f)
Fs2 = 1000; % Sampling frequency above Nyquist rate (2*f)
---
% Generate the continuous signal
---
% Sample the signal
---
% Reconstruct the signal
---
```

Try:

1. Consider a signal $x(t)=\sin(2\pi f_1 t)+0.5\sin(2\pi f_2 t)$, where f_1 = 100 Hz, f_2 = 300 Hz. Plot the sampled version of x(t) if it is sampled at fs = 250 Hz and fs = 600 Hz.
2. Generate a sine wave signal with a frequency of 200 Hz, sample this signal at 800 Hz (oversampling) and 400 Hz (Nyquist rate). Now add random noise to both sampled signals. Apply a low-pass filter to both signals to remove noise. Plot and compare the noise-reduced signals to the original signal.
   a. "Discuss the impact of oversampling on noise reduction and signal quality."

2.Exercises on Time and Frequency Response of an LTI System

The response of a Linear Time-Invariant (LTI) system can be characterized in both the time domain and the frequency domain, offering insights into how the system processes signals. Understanding both responses is crucial for analyzing and designing systems in signal processing.

2.1 Time Domain Response

In the time domain, the response of an LTI system is typically described by its impulse response and step response.

Compute and plot the impulse and step response of a discrete-time LTI system defined by its difference equation: y[n] = 0.5* y[n-1] + x[n] using MATLAB

(i) Impulse Response

Hint

```
% Parameters
% Define coefficients of difference equation
b = [1]; % Coefficients of x[n]
a = [1, -0.5]; % Coefficients of y[n]
---
% Generate an impulse signal
----
% Compute the impulse response using the filter function
---
```

(ii) Step Response

Hint

```
% Parameters
% Define coefficients of difference equation
b = [1]; % Coefficients of x[n]
a = [1, -0.5]; % Coefficients of y[n]
---
% Generate step signal
----
% Compute the impulse response using the filter function
---
```

Try:

1. Compute and plot the impulse response of an LTI system defined by the difference equation $y[n]=-1.2y[n-1]+0.8y[n-2]+x[n]+0.5x[n-1]$. And try to find out the step response.
2. Compute and plot the impulse response and step response of an LTI system defined by the transfer function.

$$H(z) = \frac{z^2-0.5z+0.25}{z^2-1.2z+0.36}$$

2.2 Frequency Response

In the frequency domain, the response of an LTI system is described by its magnitude response and phase response.

Plot the frequency response of an LTI system defined by the difference equation y[n]=0.5y[n−1]+2x[n]−0.4x[n−1] using MATLAB.

Hint

```
% Define the filter coefficients from the difference equation
b = [2, -0.4]; % Feedforward coefficients: b0 = 2, b1 = -0.4
a = [1, -0.5]; % Feedback coefficients: a0 = 1, a1 = -0.5

% Compute the frequency response
---

% Plot the magnitude response
----
grid on;

% Plot the phase response
Figure();
---
grid on
```

Try:

1. Plot the frequency response of an LTI system with the following difference equation:
   y[n] = y[n-1] + x[n]. Comment on the stability of the system.
2. Plot the frequency response of an LTI system with the following difference equation:
   y[n]=2y[n−1]+y[n−2]+x[n]-x[n−1].
3. Plot the frequency response of an LTI system with the following transfer function H(z) in the z- domain, where z is the complex frequency variable:

$$H(z) = \frac{0.5-0.5z^{-1}}{1-0.9z^{-1}}$$

3. Exercises on Discrete-Time Systems

Analyzing the response of discrete-time systems is crucial in understanding how these systems process signals. The response can be determined through various methods, depending on the system's representation (e.g., difference equation, transfer function, state-space model). Here, we'll focus on a common approach using the impulse response and the convolution sum for linear time-invariant (LTI) systems.

1. Linear Convolution
2. Circular Convolution
3. Cyclic Convolution
4. Overlap-Add Method to compute linear convolution
5. Overlap-Save Method to compute linear convolution

3.1 Linear Convolution

Linear convolution is a fundamental operation in signal processing that combines two signals to produce a third signal that represents the amount of overlap between the signals as one signal slides past another.

Consider the two discrete-time signals x[n] = [3,5,2,-1] and h[n] = [2,-1,3].

Perform the following operations using MATLAB.

1. Compute the linear convolution of x[n] and h[n] using MATLAB's built-in conv function.
2. Manually implement a function in MATLAB to perform linear convolution without using the conv function. Your function should take two sequences as inputs and return their linear convolution.
3. Compare the results obtained from MATLAB's built-in conv function and your manual implementation for accuracy.
4. Plot the original signals x[n] and h[n], and the convolution result from the built-in conv function. Label each plot with appropriate titles and axis labels.

Hint

```
x = [3, 5, 2, -1];
h = [2, -1, 3];
y_builtin = conv(x, h);
-----
---
function y = myConv(x, h)
    % Initialize the result vector with zeros of appropriate length
---

    % Perform the convolution operation (fill in the details)
    % Hint: Use nested loops to implement the 'flipping and sliding'
    ---
```

Try:

1. Given two sequences are given as follows: x[n] = [1,-1,2,3] and h[n] = [-1,-2,4] , perform the following tasks.
   a) Compute the linear convolution.
   b) Prove the commutative property of convolution.
   c) If the sequence h[n] is shifted right by 2 units, then find the output sequence.
   d) Zero-pad the shorter sequence to make both sequences of equal length, then compute the convolution again with this new length.

3.2 Circular Convolution

Circular convolution, also known as cyclic or periodic convolution, plays a crucial role in digital signal processing, especially in the context of implementing linear convolution using the Fast Fourier Transform (FFT) for efficient computation.
Two sequences are given as follows: x[n] = [4,3,2,1] and h[n] = [1,-1,2,3].

Perform the following operations using MATLAB.

1. Compute the circular convolution of x[n] and h[n] using MATLAB's built-in cconv function.
2. Compute the linear convolution for comparison.
3. Plot the original signals x[n] and h[n], and the circular convolution results. Label each plot with appropriate titles and axis labels for clarity.

4. Analyze and discuss the difference between the results obtained from the circular convolution and the linear convolution.

Hint

```
% Task 1: Circular Convolution without Zero-Padding
x = [4, 3, 2, 1];
h = [1, 2, 3];
----
% Task 2: Linear Convolution
y_linear = conv(x, h);
---
```

Try:

1. Given two sequences are given as follows: x[n] = [2,1,2,1] and h[n] = [1,2,3] , perform the following tasks.
    a. Compute the circular convolution.
    b. Compute the linear convolution
    c. Analyze and discuss the difference between the results obtained from the circular convolution (with and without zero-padding) and the linear convolution.

3.3 Cyclic Convolution

Cyclic convolution, also known as circular convolution, involves the convolution of two discrete finite-length sequences that wrap around.
Two sequences are given as follows: x[n] = [1,3,2,1,1] and h[n] = [1,0,-1,2,3]

Perform the following operations using MATLAB.

1. Compute the cyclic convolution of x[n] and h[n]
2. Compute the circular and linear convolution for comparison.
3. Plot the original signals x[n] and h[n], and the cyclic, circular, and linear convolution results. Label each plot with appropriate titles and axis labels for clarity.
4. Analyze and discuss the difference between the results obtained from the cyclic, circular, and linear convolution and the linear convolution.

Hint

```
% Task 1: Cyclic Convolution without Zero-Padding
x = [4, 3, 2, 1];
h = [1, 2, 3];
----
% Task 2: Linear Convolution
y_linear = conv(x, h);
---
```

Try:

1. Given two sequences are given as follows: x[n] = [2,1,2,3,4,-1,1,2] and h[n] = [1,-1,-2,-5,0,2] , perform the following tasks.
    a. Compute the cyclic and circular convolution.
    b. Compute the linear convolution
    c. Analyze and discuss the difference between the results obtained from the cyclic, circular, and linear convolution (with and without zero-padding) and the linear convolution.

3.4 Overlap-Add Method to Compute Linear Convolution

The Overlap-Add method is used to perform the linear convolution of a long input signal with a finite impulse response (FIR) filter by dividing the input signal into shorter segments.

Given an input signal x[n] and an FIR filter with impulse response h[n], use the Overlap-Add method to compute the convolution [n]=x[n]∗h[n]. Let the input signal be x[n]=[1,2,3,4,5,6,7,8,9] and the impulse response of the FIR filter be h[n]=[1,−1].

Hint

```
% Parameters
x = [1, 2, 3, 4, 5, 6, 7, 8, 9];
h = [1, -1];
L = 4;
M = length(h);
N = L + M - 1;

% Zero-pad h
h_padded = [h, zeros(1, N - M)];

% Initialize output
y = [];
---

% Process each segment
for start = 1:L:length(x)
    segment = x(start:min(start+L-1, length(x)));

--
```

Try:

1. Let the input signal be x[n]=[2,3,1,−1,2,−2,3,0,1,−1] and the impulse response of the FIR filter be h[n]=[1,0,−1]. Compute the convolution between x[n] and h[n] using Overlap-Add method in MATLAB.
2. Consider two discrete-time signals, x[n] and h[n], where x[n] is an input signal and h[n] is the impulse response of a finite impulse response (FIR) filter. The signals are defined as follows:

   x[n]=[4,3,2,1,2,3,4]

   h[n]=[1,−1]

   Using the Overlap-Add method with a segment length L=3, analyze the process of convolving these two signals in MATLAB.

3.5 Overlap-Save Method for Linear Convolution

The Overlap-Save method is another technique for efficiently computing the convolution of a long input signal with an FIR filter. It is particularly well-suited for real-time processing.

Given an input signal x[n] and an FIR filter with impulse response h[n], use the Overlap-Save method to compute the convolution [n]=x[n]∗h[n] in MATLAB. Let the input signal be x[n]=[3,1,2,2,4,1,2,3,4,1] and the impulse response of the FIR filter be h[n]=[2,1].

Hint

```
% Parameters
x = [3,1,2,2,4,1,2,3,4,1];
h = [2,1];
L = 4;
M = length(h);
N = L + M - 1;

---

% Initialize output
y = [];
---

% Process each segment
for start = 1:L:length(x)
    segment = x(start:min(start+L-1, length(x)));

--
```

Try:

1. Consider two discrete-time signals, x[n] and h[n], where x[n] is an input signal and h[n] is the impulse response of a finite impulse response (FIR) filter. The signals are defined as follows:

   x[n]=[4,5,6,7,8,9,10,11,12,13,14]

   h[n]=[1,-1,2]

   Using the Overlap-Save method with a segment size N=8, analyze the process of convolving these two signals in MATLAB.

4. Exercises on DFT and IDFT

The Discrete Fourier Transform (DFT) is a fundamental tool in signal processing and analysis, enabling the decomposition of discrete signals into their constituent frequencies. It's particularly important for understanding the frequency content of digital signals, which have applications across various fields including audio processing, image analysis, and telecommunications.

4.1 Exercises on DFT

To be proficient in programming, compute the DFT of various signals and interpret the results to understand the frequency characteristics of these signals.

1. Generate a discrete-time signal x[n] = sin(2π*0.1*n) + 0.5 sin(2π*0.2n), and the sampling rate is 1 Hz, use 128 samples for n. a) Compute the DFT of x[n] in MATLAB, b) Plot the magnitude spectrum of the DFT.

Hint

```
% Define the signal
Fs = 1; % Sampling frequency in Hz
T = 1/Fs; % Sampling period
L = 128; % Length of the signal
t = (0:L-1)*T; % Time vector

% Create a sinusoidal signal
---
```

```
x = sin(2*pi*f*t) + sin(2*pi*2*f*t); % Sinusoidal signal with two frequencies
----
% Compute the DFT of the signal using the FFT
X = fft(x);
---
```

Try:

1. Given a signal composed of three sinusoidal waves with frequencies 50 Hz, 120 Hz, and 250 Hz, sampled at 1000 Hz. Perform a DFT to identify these frequency components.

2. Given a noisy signal that combines a useful signal component at 60 Hz with noise components at 200 Hz and 450 Hz, sampled at 1200 Hz, use DFT to isolate and remove the noise components, then perform IDFT to obtain the filtered signal.

4.2 Exercises on IDFT

The objective of this exercise is to demonstrate the concept of Inverse Discrete Fourier Transform (IDFT) and its implementation in MATLAB. i.e., how to reconstruct a time-domain signal from its frequency-domain representation using the IDFT.

Create a discrete-time signal composed of two sinusoids with frequencies of 50 Hz and 120 Hz, respectively, sampled at 1000 Hz for 1 second. Use this signal as the basis for the following tasks.

a. Compute DFT
b. Implement IDFT
c. Reconstruct the signal.
d. Analyze the results.

Hint

```
% Define the signal
function x = myIDFT(X)
    N = length(X); % Number of DFT points
    n = 0:N-1; % Time index
---
    x = sum(X.' .* W, 2) / N; % Compute IDFT using matrix multiplication
end
---
```

Try:

1. Assume we have a frequency domain representation X[k] of a signal, where X[k] = {4,2+j2,0,2-2j} for k = 0,1,2,3. Compute the time domain signal x[n] using IDFT in MATLAB.

2. Assume we have a frequency domain representation X[k] of a signal, where X[k] = {20,-5-j2,0,-j0.5,0,+j0.5,0, -5+j2} for k =0,1,2,3,4,5,6,7. Compute the time domain signal x[n] using IDFT in MATLAB.

5. Exercises on FIR and IIR Filter Design

Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters are two fundamental types of digital filters used in signal processing. Each has distinct characteristics and is designed using different approaches.

5.1 FIR filter design

FIR filters have a finite duration response to an impulse input. This means the impulse response settles to zero in a finite time after the input is applied.

Design a low-pass FIR filter with N= 50 as filter order using the window method with a Hamming window.

Hint

```
% Parameters
N = 50; % Filter order
Fc = 0.3; % Cutoff frequency (normalized)
---
freqz(b,1); % Display frequency response
--
```

Try:

1. Design a multi-band FIR filter and simulate in MATLAB, that passes two distinct frequency bands while attenuating others. This kind of filter is useful in applications like audio processing where you might want to isolate certain frequency ranges.

   Specifications:

   Passbands: 0.2 to 0.3 and 0.5 to 0.6 (normalized frequency, where 1 corresponds to the Nyquist rate).

   Stopbands: 0 to 0.1, 0.4 to 0.5, and 0.7 to 1.

   Passband ripple: ≤ 0.01 dB

   Stopband attenuation: ≥ 60 dB.

2. Design an FIR filter with a very steep transition band, which is a common requirement in applications where It is crucial to minimize interference between adjacent frequency bands, such as in channel selection for telecommunications or isolating specific frequency bands in audio processing. Simulate in MATLAB.

   Specifications:

   Passband: 0 to 0.25 (normalized frequency, where 1 corresponds to the Nyquist rate)

   Stopband: 0.3 to 0.5

   Passband ripple: ≤ 0.5 dB

   Stopband attenuation: ≥ 100 dB

   Transition band: 0.25 to 0.3.

5.2 IIR Filter Design

IIR filters can have an infinite duration response to an impulse input. The impulse response does not become exactly zero after a finite number of steps, theoretically extending indefinitely.

5.3 Butterworth Filter Design

This aims to provide the flattest possible passband response, making it a good choice for applications requiring minimal distortion within the passband. The trade-off is a relatively wide transition band between the passband and stopband.

Design and simulate in MATLAB a low-pass Butterworth IIR filter.
Specifications:

Order of the filter N=5
Cut off frequency Fc=500 Hz
Sampling frequency Fs=1500 Hz

Hint

```
% Parameters
N = 5; % Filter order
Fc = 300; % Cutoff frequency in Hz
Fs = 1000; % Sampling frequency in Hz
----
freqz(b,a); % Display frequency response
```

Try:

1. Design a 3rd-order Butterworth bandpass filter with passband edges at 500 Hz and 1500 Hz, assuming a sampling frequency of 10 kHz. Plot the magnitude response in MTALB.
2. Design a 4th-order high-pass Butterworth filter with a cutoff frequency of 2 kHz, assuming a sampling frequency of 20 kHz. Plot the magnitude response in MTALB.

5.4 Chebyshev Filter Design

These filters allow for a sharper transition between the passband and stopband compared to Butterworth filters by allowing ripple in either the passband (Type I) or stopband (Type II).

Design a Type I Chebyshev low-pass filter with a 1 dB ripple in the passband, a cutoff frequency of 3 kHz, and a minimum attenuation of 40 dB in the stopband starting at 6 kHz. Assume a sampling frequency of 20 kHz.

Hint

```
% Type1 CB filter design
max_ fc = 3000; % Cutoff frequency in Hz
fs = 6000; % Stopband frequency in Hz
Rp = 1; % Passband ripple in dB
Rs = 40; % Stopband attenuation in dB
Fsample = 20000; % Sampling frequency in Hz
--
title('Type I Chebyshev Low-Pass Filter'); fprintf('The minimum sampling rate required is
```

Try:

1. Design a Type II Chebyshev high-pass filter with a 20 dB attenuation in the stopband, a stopband cutoff frequency of 500 Hz, and a passband edge starting at 1 kHz with minimal attenuation. Assume a sampling frequency of 5 kHz.

Hint

```
% Type2 CB filter design
fstop = 500; % Stopband cutoff frequency in Hz
fpass = 1000; % Passband edge frequency in Hz
```

```
Rs = 20; % Stopband attenuation in dB
Fsample = 5000; % Sampling frequency in Hz
--
% Determine the filter order and cutoff frequency
--
title('Type II Chebyshev High-Pass Filter');--
```

Try:

1. Design Type I Chebyshev bandpass filter that meets specific group delay requirements. Specifications: Passband: 2 kHz to 4 kHz, Passband Ripple: 3 dB Stopband Attenuation: ≥ 40 db Sampling Frequency: 16 kHz, Group Delay: Minimize variations within the passband.

6. Exercises on Optimum Equiripple FIR Digital Filter

The design of an optimum equiripple Finite Impulse Response (FIR) digital filter involves creating a filter where the magnitude of the ripple is the same (or equiripple) in both the passband and the stopband. This kind of design aims to minimize the maximum error between the desired and the actual filter response, leading to an optimal filter in the sense of minimizing the maximum deviation within the filter's passband and stopband. The Parks-McClellan algorithm, also known as the Remez Exchange algorithm, is a popular method used to design equiripple FIR filters.

Design an optimum equiripple low-pass FIR digital filter with the following specifications:

- Passband edge frequency: 1500 Hz
- Stopband edge frequency: 2000 Hz
- Passband ripple: 1 dB maximum
- Stopband attenuation: 40 dB minimum
- Sampling frequency: 8000 Hz

Hint

```
% EQR FIR lowpass filter design
% Filter specifications
Fs = 8000; % Sampling frequency in Hz
Fpass = 1500; % Passband edge frequency in Hz
Fstop = 2000; % Stopband edge frequency in Hz
Rp = 1; % Passband ripple in dB
Rs = 40; % Stopband attenuation in dB
% Normalize frequencies
--
---
% Frequency response analysis
freqz(b, 1, 1024, Fs)
title('Optimum Equiripple FIR Low-Pass Filter');grid on;
```

Try:

1. Design and simulate in MATLAB an equiripple FIR halfband filter with the following specifications:
   - Sampling frequency: 2000 Hz
   - Transition band: Centered around 2100 Hz (0.25 times the sampling frequency)
   - Passband ripple and Stopband attenuation: Specify as needed for equiripple condition.

6.1 Equiripple FIR Halfband Filter

An equiripple FIR halfband filter is a specific type of filter with the property that every other coefficient (except for the center one) is zero, and the cutoff frequency is at one-quarter of the sampling rate. Halfband filters are efficient in both their design and implementation, especially useful in multirate signal processing for tasks like interpolation and decimation.

Design and simulate in MATLAB an equiripple FIR halfband filter with the following specifications:

Sampling frequency: 8000 Hz

Transition band: Centered around 2000 Hz (0.25 times the sampling frequency)

Passband ripple and Stopband attenuation: Specify as needed for equiripple condition.

Hint

```
% Parameters
% Sampling frequency
Fs = 8000; % Hz
% Define specifications for the halfband filter
--

% Normalize frequencies
Wp = Fpass/(Fs/2);
Ws = Fstop/(Fs/2);
---
% Estimate the filter order and design the filter using the Parks-McClellan
algorithm
[N, Fo, Ao, W] = firpmord([Wp Ws], [1 0], [10^(-Rp/20) 10^(-Rs/20)]);
---
freqz(b, 1, 1024, Fs);
title('Equiripple FIR Halfband Filter')
```

Try:

1. Incorporate a halfband FIR filter into a multirate digital signal processing (DSP) system designed for audio streaming. The system dynamically adjusts between standard (44.1 kHz) and high-resolution (88.2 kHz) audio streams. The halfband filter is used to efficiently transition between these sampling rates.

   Specification:

   Standard Audio Sampling Frequency: 44.1 kHz
   High-Resolution Audio Sampling Frequency: 88.2 kHz

Transition Requirement: Seamless switching between standard and high-resolution audio streams.

6.2 FIR Halfband Interpolator

An FIR (Finite Impulse Response) halfband interpolator is a specialized type of digital filter used to double the sampling rate of a signal. The halfband filter is characterized by its frequency response, where approximately half of its spectrum is passed (the low-pass region) and the other half is attenuated (the stopband region).
The distinctive feature of halfband filters is that their transition band is centered around the Nyquist frequency (Fs/4, where Fs is the new, doubled sampling rate), and they have a symmetric frequency response with respect to Fs/4.

Design an equiripple FIR halfband low-pass filter to be used in a interpolator process, where the goal is to increase the sampling rate of a signal by a factor of 2.
Specifications:

Desired transition band centered around the Nyquist frequency of the original signal.
Passband ripple: 0.01 dB maximum
Stopband attenuation: 80 dB minimum
Original sampling frequency: 32000 Hz

Hint

```
% Parameters
% Filter specifications for polyphase interpolator component
Fs = 32000; % Original sampling frequency
Rp = 0.01; % Passband ripple in dB
Rs = 80; % Stopband attenuation in dB
% Design the equiripple FIR halfband filter
N = 50; % Preliminary filter order, adjust based on design requirements
f = [0 0.25 0.25 0.5]; % Normalized frequency points for halfband
a = [1 1 0 0]; % Desired amplitude response
b = firpm(N, f, a, [1 1]); % Design filter with equal weight in pass and stopband
---
% Frequency response analysis
freqz(b, 1, 1024, Fs*2);
-
title('Equiripple FIR Halfband Filter')
```

Try:

1. Design and simulate in MATLAB: an optimized FIR halfband interpolator for a signal with a maximum frequency component of 750 Hz. Assume the original sampling frequency of the signal is 2000 Hz, and you want to interpolate it to 4000 Hz.

7.Exercises on DIT-FFT and DIF-FFT Algorithms

Decimation-In-Time (DIT) Fast Fourier Transform (FFT) and Decimation-In-Frequency (DIF) FFT algorithms can be a valuable educational experience for understanding the principles and efficiencies of FFT algorithms.

7.1 DIT-FFT Algorithm

The Fast Fourier Transform (FFT) is a fundamental algorithm in digital signal processing, allowing for the efficient computation of the Discrete Fourier Transform (DFT) of a sequence. The Decimation-In-Time (DIT) approach is a common strategy for implementing the FFT, characterized by its divide-and-conquer methodology, which recursively breaks down a DFT of any composite size N=2^m into smaller DFTs.

Given a sequence x[n]={1,2,3,4,4,3,2,1}, compute its DFT using the DIT FFT algorithm and verify using MATLAB Program.

Hint

```
function X = ditFFT(x)
    % Ensure the input is a row vector
    if size(x,2) == 1
        x = x.';
    end

    N = length(x);
    if N <= 1
        X = x;
        return;
    end
----
```

Try:

1. You have sampled a signal at 1024 Hz, and you obtain the following 8-point sequence from one of its segments: x[n]=[1,−1,1,−1,1,−1,1,−1]. Compute its DIT FFT and discuss the frequency components present in this signal using MATLAB program.
2. Calculate the twiddle factors for an 8-point DIT FFT. Provide the values for $W_8^0, W_8^1, W_8^2, W_8^3$ using MATLAB program.

7.2 DIF-FFT Algorithm

Decimation-In-Frequency (DIF) Fast Fourier Transform (FFT) algorithm provides an excellent opportunity to explore its theoretical understanding, computational efficiencies, and practical applications.

Write a MATLAB function that implements a simple 4-point DIF FFT algorithm. Use this function to compute the FFT of the sequence x[n]=[1,1,1,1] and display the result.

Hint

```
function X = ditFFT(x)
function X = DIF_FFT(x)
    % Ensure x is a row vector
    if iscolumn(x)
        x = x.';
    end

    N = length(x);
  --
  --
```

Try:

1. Implement an 8-point DIF FFT algorithm in MATLAB and use it to analyze a real-valued signal x[n]=[0,1,2,3,4,5,6,7]. Display both the time-domain and frequency-domain representations.

2. Investigate the effect of zero-padding on the frequency resolution. Take a signal x[n] of length 8, zero-pad it to length 16 and perform a DIF FFT on both signals. Compare the frequency spectra using MATLAB.

8. Exercises on FIR Digital Filter Using Window Method

8.1 FIR Digital Filter Design using Rectangular Window

The rectangular window, also known as the boxcar or uniform window, is the simplest type of window function used in signal processing.

Generate a rectangular window.

Hint:

```
N = 64;  % Window length
rectangularWindow = rectwin(N);
% Plot the rectangular window
stem(rectangularWindow);
title('Rectangular Window');
xlabel('Sample Index');
ylabel('Amplitude');
```

visualize the frequency response of a rectangular window using the freqz function.

Hint:

```
N = 64;  % Window length
```

```
rectangularWindow = rectwin(N);
% Frequency response of the rectangular window
Fs = 1000;  % Sampling frequency
freqResponse = freqz(rectangularWindow, 1, 1024, Fs);
% Plot the frequency response
figure;
subplot(2, 1, 1);
plot(rectangularWindow);
title('Rectangular Window in Time Domain');
xlabel('Sample Index');
ylabel('Amplitude');
subplot(2, 1, 2);
plot(linspace(0, Fs/2, length(freqResponse)), 20*log10(abs(freqResponse)));
title('Frequency Response of Rectangular Window');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
FIR Filter Design using Rectangular Window Method
```

Try:

1. Design a digital FIR lowpass filter with the following specifications: wp = 0.2π, ws = 0.3π, Rp = 0.25dB As = 50dB using rectangular window function. Determine the impulse response and provide a plot of the frequency response of the designed filter using MATLAB.

8.2 FIR Digital Filter Design using Hanning Window

The Hanning window, is characterized by a central lobe with smaller sidelobes compared to the rectangular window. visualize the frequency response of the Hanning window using the freqz function

Hint

```
N = 64;  % Window length
hanningWindow = hann(N);
% Frequency response of the Hanning window
Fs = 1000;  % Sampling frequency
freqResponse = freqz(.....);
% Plot the frequency response
figure;
subplot(2, 1, 1);
plot(.....);
```

Try:

1. Design a low-pass FIR filter with a specified cutoff frequency using the Hanning window method, and Analyze its frequency and time-domain characteristics. Sampling frequency = 1000 Hz, Cut off frequency = 150 Hz, Order of the Filter = 50
2. Using MATLAB program, demonstrate the effect of the Hanning window in reducing spectral leakage by comparing the frequency spectra of a sinusoidal signal when windowed with a Hanning window versus a rectangular window.

Generate a sinusoidal signal with a frequency that does not exactly match the FFT bin frequencies (e.g., f=50.5 Hz, sampling frequency fs=1024 Hz, and a duration of 1 second).

Compute and plot the magnitude spectrum of the signal without any windowing. Apply a Hanning window to the signal and then compute and plot the magnitude spectrum again. Compare the results and discuss the effect of the Hanning window on spectral leakage.

9. Exercises on IIR Digital Filter using Butterworth, Chebyshev, and Bilinear Transformation

Designing an IIR (Infinite Impulse Response) digital filter using the Butterworth method and bilinear transformation involves several steps. The Butterworth filter is known for its maximally flat frequency response in the passband, making it a popular choice for many filtering applications. The bilinear transformation method is used to convert an analog filter design into a digital one without introducing aliasing and to preserve the frequency response characteristics as closely as possible.

9.1 Designing a Butterworth Low-pass IIR Filter

Design a digital low-pass filter using the Butterworth design approach and bilinear transformation
Specifications:

Passband frequency, fp=1kHz
Stopband frequency, fs=1.5kHz
Passband ripple, δp=1dB (maximum allowable attenuation in the passband)
Stopband attenuation, δs=40dB (minimum attenuation in the stopband)
Sampling frequency, fsamp=8kHz

Hint

```
% Butterworth Low-Pass IIR Filter Design Using Bilinear Transformation

% Specifications

fp = 1000; % Passband frequency in Hz

fs = 1500; % Stopband frequency in Hz

delta_p = 1; % Passband ripple in dB

delta_s = 40; % Stopband attenuation in dB

f_samp = 8000; % Sampling frequency in Hz

--

--
```

Try:

1. Design a Butterworth low-pass filter to smooth an image by attenuating high-frequency components.

   Specifications:
   - Passband frequency: 20 Hz
   - Stopband frequency: 50 Hz
   - Passband ripple: 1 dB
   - Stopband attenuation: 20 dB
   - Sampling frequency: 100 Hz

9.2 Designing a Chebyshev Type I and bilinear transformation IIR Filter

Designing an IIR (Infinite Impulse Response) filter using a Chebyshev Type I design approach and bilinear transformation involves specific steps. Chebyshev Type I filters allow for a ripple in the passband but have a steeper roll-off compared to Butterworth filters, making them useful in applications where the steepness of the filter's transition band is more critical than the flatness of the passband.

Design a Chebyshev Type I low-pass filter with the following specifications:

- Passband frequency fp: 2 kHz
- Stopband frequency fs: 2.5 kHz
- Passband ripple δp: 1 dB
- Stopband attenuation δs): 40 dB
- Sampling frequency fsamp: 16 kHz

Hint

```
% Filter Specifications

fp = 2000; % Passband frequency in Hz

fs = 2500; % Stopband frequency in Hz

rp = 1; % Passband ripple in dB

rs = 40; % Stopband attenuation in dB

fsamp = 16000; % Sampling frequency in Hz

--
```

Try:

1. Design a Chebyshev Type I low-pass filter to remove high-frequency noise from an Electrocardiogram (ECG) signal.

   Specifications:

   - Passband frequency: 40 Hz
   - Stopband frequency: 100 Hz
   - Passband ripple: 1 dB
   - Stopband attenuation: 20 dB
   - Sampling frequency: 1 kHz

9.3 Designing a Chebyshev Type II and bilinear transformation IIR Filter

Designing an IIR (Infinite Impulse Response) filter using a Chebyshev Type II design approach and bilinear transformation.

Design a Chebyshev Type II low-pass filter with the following specifications:

- Passband frequency fp: 1 kHz
- Stopband frequency fs: 1.5 kHz
- Passband ripple δp: 0.5 dB
- Stopband attenuation δs): 40 dB
- Sampling frequency fsamp: 8 kHz

Hint

```
% Filter Specifications

fp = 1000; % Passband frequency in Hz

fs = 1500; % Stopband frequency in Hz

rp = 0.5; % Passband ripple in dB

rs = 40; % Stopband attenuation in dB

fsamp = 8000; % Sampling frequency in Hz

--
```

Try:

1. Design a Chebyshev Type II low-pass filter to enhance speech signals by attenuating high-frequency noise components.

   Specifications:
   Passband frequency: 3.4 kHz
   Stopband frequency: 4 kHz (
   Passband ripple: 0.5 dB
   Stopband attenuation: 50 dB
   Sampling frequency: 16 kHz

10. Exercises on DTMF Tone Generation and Detection

Dual-tone multi-frequency (DTMF) signaling is used for telecommunication signaling over analog telephone lines in the voice-frequency band between telephone handsets and other communications devices and the switching center. DTMF assigns a specific frequency pair to each key so that it can easily be identified by the electronic circuit.

Generate DTMF tones corresponding to the keys on a standard telephone keypad.

- Look up the frequency pairs for each DTMF key (0-9, A-D, *, #).
- Choose a sampling frequency fs=8000 Hz.
- For a given key, generate two sinusoidal waves corresponding to its frequency pair for a specific duration (e.g., 0.5 seconds).
- Add the two sinusoids to generate the DTMF tone.
- Repeat the process for multiple keys.

Hint

```
lfg = [697 770 852 941]; % Low frequency group
unction generateDTMFTone(key, fs, duration)
    % Define DTMF frequencies for each key
    dtmfFreqs = {
        {'1', '2', '3', 'A'; 697, 697, 697, 697},
        {'4', '5', '6', 'B'; 770, 770, 770, 770},
        {'7', '8', '9', 'C'; 852, 852, 852, 852},
        {'*', '0', '#', 'D'; 941, 941, 941, 941},
--
```

Try:

1. Simulate a simple DTMF-based communication system where a sequence of keys is transmitted, and the receiver decodes the sequence.

10.1 DTMF Tone Detection Using Goertzel Algorithm

The Goertzel algorithm is a digital signal processing technique used to identify frequency components of a signal at specific frequencies. It is particularly efficient when you need to detect a small number of frequencies compared to the total number of samples in the signal, making it highly suitable for applications like DTMF (Dual-Tone Multi-Frequency) tone detection, spectral analysis of sparse signals, and more.

Hint

```
function detectedKey = detectDTMFTone(signal, fs)
    % Define DTMF frequencies
    lowFreqs = [697, 770, 852, 941];
    highFreqs = [1209, 1336, 1477, 1633];
    % Pre-define the keys corresponding to frequency pairs
    keys = ['1', '2', '3', 'A';
            '4', '5', '6', 'B';
            '7', '8', '9', 'C';
            '*', '0', '#', 'D'];
-    % Goertzel algorithm to detect low frequencies
    for f = lowFreqs
     --
    % Goertzel algorithm to detect high frequencies
-   % Determine the detected key
```

--

Try:

1. Generate a signal that simulates a sequence of DTMF tones corresponding to a series of keypad presses (e.g., "123A"). Implement the Goertzel algorithm to detect the two frequency components of each DTMF tone. Determine the keys pressed based on the detected frequency pairs. Display the sequence of detected keys and compare it to the original sequence.

11. Exercises on Sampling Rate Conversion

Implementing sampling rate conversion by decimation, interpolation, and a rational factor in MATLAB involves several steps. The process is aimed at converting the sampling rate of a signal from one rate to another, which is specified by a rational factor. The rational factor is represented by two integers, L and M, where L is the interpolation factor (number of samples to insert between existing samples) and M is the decimation factor (number of samples to remove).

Hint

```
% Generate a sample signal (e.g., a sine wave)
Fs = 1000; % Original sampling rate in Hz
t = 0:1/Fs:1-1/Fs; % Time vector
x = sin(2*pi*100*t); % Sine wave at 100 Hz
L = 2; % Interpolation factor
M = 3; % Decimation factor –
----Interpolation
--- decimation
```

Try:

1. Using MATLAB program, generate a cosine signal with a frequency of 100 Hz, sampled at 1000 Hz. Interpolate the signal by a factor of 2, Decimate the interpolated signal by a factor of 2. Plot the original, interpolated, and decimated signals in different subplots and observe the changes.
2. Perform sampling rate conversion using a non-integer (rational) factor. Generate a signal with multiple frequency components. Choose a target sampling rate that is not an integer multiple of the original rate (e.g., convert from 1000 Hz to 750 Hz). Calculate the appropriate interpolation and decimation factors to achieve the desired conversion. Interpolate, filter, and decimate the signal according to the calculated factors. Plot and compare the original and converted signals in both time and frequency domains.

11.1 Interpolation for Non-Integer Sampling rate

Interpolation for non-integer sampling rate conversion is a critical operation in digital signal processing (DSP) for modifying the sampling rate of a signal. This process is particularly important when you need to convert a signal from one sampling rate to another that is not an integer multiple of the original rate, a common requirement in many audio and telecommunications applications.

Write a MATLAB function to perform interpolation by a non-integer factor on a given signal.

Hint

```
% New (non-integer) sampling rate
function interpolatedSignal = interpolateSignal(inputSignal, originalFs, targetFs)
---
% Example usage
fs = 100; % original sampling rate in Hz
t = 0:1/fs:1-1/fs; % time vector
x = cos(2*pi*10*t); % example signal
--


% Plot original and interpolated signals
```

Try:

1. x(n) = cos(πn). Generate samples of x(n) and interpolate them using I = 2, 4, and 8 . Plot original signal and sampled signal.

12. Exercises on Sine Wave Generation

12.1 Generation of sine wave using lookup table

Generating a sine wave using a lookup table is a common technique in embedded systems and digital signal processing to efficiently produce periodic waveforms. This approach involves pre-calculating the sine values for a range of angles and storing them in a table. When a sine wave output is required, the system retrieves the pre-calculated values from the table instead of performing computationally expensive trigonometric calculations in real-time.

Hint

```
% matlab code to generate the sine values of the look table
% Number of points in the lookup table (e.g., 256 for an 8-bit resolution)
N = 256;
% Generate lookup table values
---
% Optional: Scale and convert to integer if required for your application
% For example, to scale to 12-bit DAC (0 to 4095 range)
scaledLookupTable = round((lookupTable + 1) * (4095/2));
```

Try:

1. Design a MATLAB script to generate a composite signal from three different sine waves (e.g., 440 Hz, 880 Hz, and 1760 Hz) using the lookup table.

12.2 FIR Filter using Frequency Sampling Method

The Frequency Sampling Method for designing FIR (Finite Impulse Response) filters is a technique that allows for the direct specification of the magnitude and phase of the filter's frequency response at a discrete set of frequencies. This method is particularly useful when the desired frequency response is known at specific frequencies, and you want to design a filter that closely matches these specifications.

Designing of basic low-pass FIR filter using the Frequency Sampling Method:

Specifications: Cutoff frequency at 0.25π radians/sample over N=16 points.

Plot the resulting impulse response h[n].

Hint

```
%  Basic FIR Filter Design using Frequency Sampling Method
% Define the number of points
N = 16;
% Frequency array (0 to 2*pi, N points)
---
% Define the desired frequency response H_d[k] for a low-pass filter
% with cutoff frequency at 0.25*pi radians/sample
Hd = zeros(1, N);
cutoff = 0.25*pi; % Cutoff frequency
for k = 1:N
---
```

Try:

1. Investigate how the length of the FIR filter (N) affects its performance, particularly in terms of the sharpness of the transition band and the stopband attenuation. Design three filters using the Frequency Sampling Method with lengths N=16, N=32, and N=64. For each filter, compute the filter coefficients h[n] and plot their impulse responses using MATLAB program.

13. Exercises on TMS 320 C6713 DSK - Code composer studio – Digital Filters

The TI's TMS320C6713 DSK is designed and optimized to perform digital signal processing operations. For short this DSP will be referred to as 'C6713. The family of this DSP is referred to as 'C6x or 'C6000. 'C6713 is a high performance 32-bit floating- point DSP. VLIW architecture. low-cost standalone development platform that enables users to evaluate and develop applications for the TI C67xx DSP family. Start getting familiar with TI's TMS320C6713 by learning the following. An overview of the functional blocks of the board. Code Composer Studio (CCS).

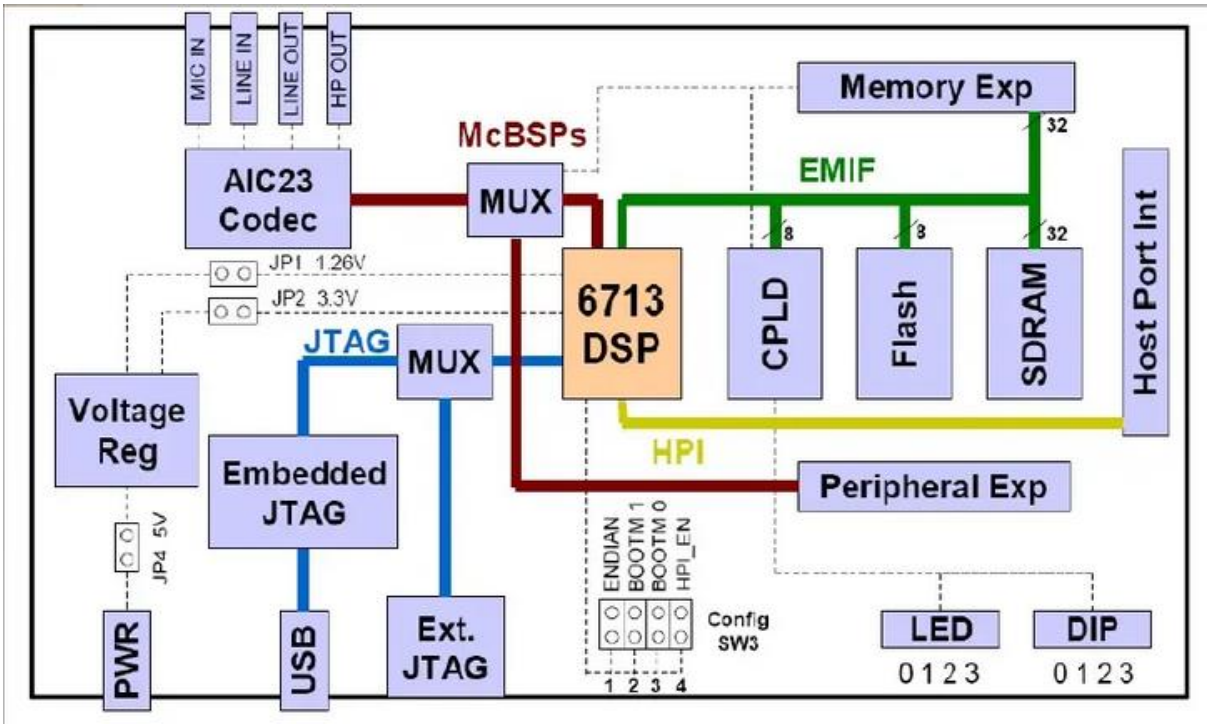

Figure: Block Diagram of TMS320C6713 DSK

Hint

A simple sine wave was created and stored in a workspace variable called simin as in an M-file:

PROGRAM:-

```
npts=2000; % these are the total number of pts per cycle in our simulation
n=0:1:npts-1;% number of pts n are in the increments of n=0,1,2,3...1999
fss=8000; % this is the sampling frequency for the input
delta=1/fss; % this will be used in simulation purposes
t=n*delt; % Total time t=1999/8000=0.249 figure(1),
Vin=sin(2*pi*2000*t); % this is the analog signal
plot(t,Vin),grid on % this plots Vin for the specified t range
xlabel('time(seconds)'); % add axis labels and plot title
ylabel('magnitude of sine wave');
title('A simple sine wave of freq=2000Hz');
whos; %display the contents of all variables used in
matlab workspace
simin=[t' Vin'] % the transpose of t and Vin is saved in
thevariable simin
save myinput.mat % the values of t and Vin saved in the
variable in an input file.
```

Try:

1. Generate cosine wave using TMS320C6713 DSP processor.
2. Generate the signal x(n)=sin(3π)+2cos(3π)using DSP processor.

13.1 Implementation of LP and HP FIR Filter For Given Sequence

To design and implement a Digital FIR Filter & observe its frequency response. In this experiment we design a simple FIR filter to stop or attenuate the required band of frequencies components and pass the frequency components, which are outside the required band.

Hint

Coefficients for FIR Low Pass Kaiser filter:

Cutoff freq: 8khz, sampling freq: 24khz

#define N 82 //length of filter

short h[N]= { 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 1, -1, 0, 2, -2, 0, 2, -2, 0, 2, -2,

0, 3, -3, 0, 4, -4, 0, 5, -6, 0, 10, -14, 0, 70, 70, 0, -14, 10, 0, -6, 5, 0, -4, 4, 0, -3, 3, 0, -2, 2,

0, -2, 2, 0, -2, 2, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0, -1, 1, 0 };

Coefficients for FIR Low Pass rectangular filter:

Cutoff freq: 8khz, sampling freq: 24khz

#define N 82 //length of filter

Try:

1. Band pass filter plot using C6713 DSK processor.
2. Generate cosine wave using TMS320C6713 DSP processor.
3. FIR Band pass filter coefficients: #define N 64 //length of filter.

14. Exercises on IIR and FIR Filters Using DSP Kits

To design and observe the characteristics of IIR and FIR filters using DSKC6713 processor. In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response (or response to any finite length input) is of finite duration, because it settles to zero in finite time. This contrasts with infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely (usually decaying). IIR can be unstable, whereas FIR is always stable.

Hint

/ c program for generation of fir filter using c6713 DSK

#include ”firfiltercfg.h”

#include ”dsk6713.h”

#include ”dsk6713 aic23.h”

#include ”stdio.h”

0.020203};

73

/FIR Low pass Rectangular Filter pass band range 0-1500Hz

Try:

1. High pass filter plot using C6713 DSK processor.
2. Design a multi-band FIR filter and simulate in MATLAB, that passes two distinct frequency bands while attenuating others. This kind of filter is useful in applications like audio processing where you might want to isolate certain frequency ranges.

Specifications:

Passbands: 0.2 to 0.3 and 0.5 to 0.6 (normalized frequency, where 1 corresponds to the Nyquist rate).

Stopbands: 0 to 0.1, 0.4 to 0.5, and 0.7 to 1.

Passband ripple: ≤ 0.01 dB

Stopband attenuation: ≥ 60 dB.

14.1 DSP Applications

Implantation of Decimation process and Interpolation Process: Implementing decimation and interpolation processes on DSP (Digital Signal Processing) kits involves programming the DSP processor to perform sample rate reduction (decimation) and sample rate increase (interpolation).

14.2 DSP Applications: Decimation

Hint

IMPLEMENTATION OF DECIMATION PROCESS:

```
#include <usbstk5515.h>
#include<stdio.h>
#include <math.h>
#define PTS 256 //no of points for FFT
#define PI 3.14159265358979
float y1[PTS],y2[PTS],y3[PTS];
main()
{
int n,m;
printf("Enter Vaue for Decimation Factor\n");
scanf("%d",&m);
printf("Original signal samples:\n");
for(n=0;n<=PTS;n++) //original signal
.....
.....
```

```
{
y2[n]=sin(2*PI*10*n/(m*PTS));
printf("%d, %f\n",n,y2[n]);
}
} //end of main
```

Try:

1. Implement a basic decimation process with a fixed decimation factor and analyze the effects on a test signal. Design a simple low-pass FIR filter to act as an anti-aliasing filter. The cutoff frequency should be appropriate for the decimation factor you choose. Use the DSP kit to implement the filter, inputting the filter coefficients manually or using the kit's software tools. Implement Down-Sampling: Write a routine to down-sample the filtered signal by a factor of M (e.g.,M=2 or M=4).

14.3 DSP Applications: Interpolation

Interpolation involves increasing the sample rate of a signal. It consists of up-sampling followed by filtering to reconstruct the intermediate samples.

Hint

IMPLEMENTATION OF INTERPOLATION PROCESS:

```
#include <usbstk5515.h>
#include<stdio.h>
#include <math.h>
#define PTS 256 //no of points for FFT
#define PI 3.14159265358979
float y1[PTS],y2[PTS],y3[PTS];
main()
::::::
{
y2[n]=sin(2*PI*10*(m*n)/(PTS));
printf("%d, %f\n",n,y2[n]);
}
}
```

Try:

1. Implement a basic interpolation process by a factor of 2 and analyze the effects on a test signal.
2. Explore the effects of different interpolation factors on signal quality and spectral characteristics. Repeat Basic Interpolation with Various Factors: Implement interpolation by factors of 2, 4, and 8. For each interpolated signal, analyze the effects on the signal waveform and spectrum.

V. TEXT BOOKS:

1. Discrete-Time Signal Processing, I second edition, A. V. Oppenheim, R. W. Schafer, J. R. Buck, Prentice Hall, 1999.
2. Real-Time Digital Signal Processing, from MATLAB to C With TMS320C6x DSK,î T. B. Welch, C. H. G. Wright, and M. G. Morrow, Taylor & Francis, 2006.
3. Real-time digital signal processing: Implementations and applications, S. M. Kuo, B. H. Lee, and W. Tian, second edition, John Wiley & Sons, 2006.

VI. REFERENCE BOOKS:

1. Digital Signal Processing Implementation using the TMS320C6000TM Platform, Naim Dahnoun, Prentice Hall, 2000.
2. Digital Signal Processing Using MATLAB , Sanjit Mitra, McGraw-Hill, 1999.
3. CL Wadhwa, *Electrical Circuit Analysis including Passive Network Synthesis*, International, 2nd Edition, 2009.
4. Proakis and Manolakis, "Digital Signal Processing", 3rd edition, Prentice Hall, 1996.
5. Little and Shure, "Signal Processing TOOLBOX User's Guide for use with MATLAB", MathWorks, Inc., 1992.

VII. ELECTRONICS RESOURCES:

1. https://ccrma.stanford.edu/~jos/sasp/Example_Overlap_Add_Convolution.html
2. https://rt-dsp.com/3rd_ed/app_a/App_CCS_5_1_dsk6713.pdf
3. Mathworks. The Mathworks Web Site: http://www.mathworks.com/. Technical report, Mathworks, 2001
4. W. Eaton. Octave web site http://www.che.wisc.edu/octave. Technical report, Univ. Wisconsin, 2001.
5. G. Campbell. DataLab Users' Manual. Technical report, University of Ulster, Interactive Systems Centre, Report isc/94/015/r, available from http://www.cs.qub.ac.uk/~J.Campbell/dl/dlprg.a, 1994.

VIII. MATERIALS ONLINE

1. Course template
2. Lab manual