LABORATORY WORK BOOK

For Academic Session _____

Semester _____

DIGITAL SIGNAL PROCESSING

(TC-212)

For

SE (TC) & TE(EL)

Name:_____

Roll Number:_____

Batch:_____

Department:_____

Year/Semester:_____

Department of Electronic Engineering
NED University of Engineering & Technology, Karachi

LABORATORY WORK BOOK

For The Course

TC-212 DIGITAL SIGNAL PROCESSING

Prepared By:

Ms. Tahniyat Aslam

Reviewed By

Dr. Muhammad Imran Aslam (Associate Professor)

Approved By:

Board of Studies of Department of Electronic Engineering

CONTENTS

LAB SESSION 1

Effects of Sampling and Aliasing in Discrete Time Sinusoids

OBJECTIVE:

1. Simulate and plot two continuous time (CT) sinusoids of 10 Hz and 110 Hz for $0 < t < 0.2$ sec.
2. Sample both sinusoids at Fs = 100 samples/sec and plot them in discrete form.
3. Observe and note the aliasing effects.
4. Explore and learn.

INTRODUCTION:

Signals are physical quantities that carry information in their patterns of variation. Continuous-time signals are continuous functions of time, while discrete-time signals are sequences of numbers. If the values of a sequence are chosen from a finite set of numbers, the sequence is known as a digital signal. Continuous-time, continuous-amplitude signals are also known as analog signals.

Signal processing is concerned with the acquisition, representation, manipulation, transformation, and extraction of information from signals. In analog signal processing these operations are implemented using analog electronic circuits. Digital signal processing involves the conversion of analog signals into digital, processing the obtained sequence of finite precision numbers using a digital signal processor or general purpose computer, and, if necessary, converting the resulting sequence back into analog form. When stored in a digital computer, the numbers are held in memory locations, so they would be indexed by memory address. In order to bridge the gap between analog and digital domain, we have to perform two basic operations Sampling and Quantization.

Sampling:

It is the process of acquiring data at discrete (finite) instants of time

Sampling Theorem:

A continuous time signal x(t) can be reconstructed exactly from its samples x(n)=x(nTs), if the samples are taken at a rate Fs=1/Ts that is greater than 2*Fmax.

Aliasing: A common problem that arises when sampling a continuous signal is aliasing, where a sampled signal has replications of its sinusoidal components which can interfere with other components. Itis an effect that causes two discrete time signals to become indistinct due to improper sampling (fd>1/2). Aliasing also occurs on television whenever we see a car whose tires appear to be spinning in the wrong direction. A television broadcast can be thought of as a series of images, sampled at a regular rate, appearing on screen. If the wheels happen to rotate less than a full circle between frames (images), then they appear to be turning slowly in the opposite direction.

PROCEDURE:

```
 % Plotting the two CTCV sinusoids
clear all;
close all;
clc;
F1 = 10;
F2 = 110;
Fs = 100;
Ts = 1/Fs;
t = [0 : 0.0005 : 0.2];
x1t = cos(2*pi*F1*t);
x2t = cos(2*pi*F2*t);
figure,
plot(t,x1t,t,x2t, 'LineWidth',2);
xlabel('cont time (sec)');
ylabel('Amp');
xlim([0 0.1]);
grid on;
legend('10Hz','110Hz');
title('Two CTCV sinusoids plotted');
% Sampling the two CTCV sinusoids
nTs = [0 : Ts : 0.2];
n = [0 : length(nTs)-1 ];
x1n = cos(2*pi*F1*nTs);
x2n = cos(2*pi*F2*nTs);
figure,
subplot(2,1,1),
stem(nTs,x1n,’filled’,'LineWidth',2);
grid on;
title('10Hz sampled');
xlabel('discrete time (sec)');
ylabel('Amp');
xlim([0 0.1]);
subplot(2,1,2)
stem(nTs,x2n,’filled’,'LineWidth',2);
grid on;
title('110Hz sampled')
xlabel('discrete time (sec)');
ylabel('Amp');
xlim([0 0.1]);
% Plotting the sinusoids along with sampled versions
figure,
plot(t,x1t,t,x2t,'LineWidth',2);
hold
stem(nTs,x1n,'r','LineWidth',2);
xlabel('time (sec)');
ylabel('Amp');
xlim([0 0.05]);
legend('10Hz','110Hz');
```

EXERCISE/TASK:

These tasks needs to be completed before your next practical and saved.

Task-1: Consider the following CT signal: x(t) = sin (2 pi F0 t). The sampled version will be: x(n) =sin (2 pi F0/Fs n), where n is a set of integers and sampling interval Ts=1/Fs.
Plot the signal x(n) for n = 0 to 99 for Fs = 5 kHz and F1 = 0.5, 2, 3 and 4.5 kHz. Explain the similarities and differences among various plots. Also mention that whether aliasing occurs or not.

Task-2: Consider the given CT signal: x(t) = sin (2 pi F0 t). Suppose that F0 = 2 kHz and Fs = 50 kHz.
i) Plot the signal x(n). What will be the discrete frequency fdof the signal x(n) ?
ii) Plot the signal y(n) created by taking the even numbered samples of x(n). Is this a sinusoidal signal? Why? If so, what is the frequency?

RESULT:

LAB SESSION 2

Discrete-Time Convolution

OBJECTIVE:

1. Consider the impulse response of a system as $h(n) = \{\mathbf{3}, 2, 1, -2, 1, 0, -4, 0, 3\}$
2. For $x(n) = \{\mathbf{1}, -2, 3, -4, 3, 2, 1\}$, find the output *y(n).*
3. Modify the code such that $h(n) = \{3, 2, \mathbf{1}, -2, 1, 0, -4, 0, 3\}$ (origin is shifted), and check the causality property .

THEORY:

A ***discrete-time system*** is a computational process or algorithm that transforms or maps a sequence x(n), called the input signal, into another sequence y(n), called the output signal. In practice, a discrete-time system is a numerical algorithm that processes an input sequence x(n), to produce an output sequence y(n).

A system is called ***causal*** if the present value of the output does not depend on future values of the input, that is, $y(n0)$ is determined by the values of $x(n)$ for $n \leq n0$, only. If the output of a system depends on future values of its input, the system is ***noncausal***.

Causality implies that if x(n) = 0 for $n < n0$, then y(n) = 0 for $n < n0$; that is, a causal system cannot produce an output before the input is applied. Clearly, this problem does not exist if the entire input sequence is already stored in memory. Although causality is necessary for the real-time implementation of discrete-time systems, it is not really a problem in off-line applications where the input signal has been already recorded.

The main premise of this lab is that the response of a linear time-invariant (LTI) system to any input can be determined from its response $h(n)$ to the unit sample sequence $\delta(n)$,using a formula known as ***convolution summation***. The sequence $h(n)$, which is known as ***impulse response***, can also be used to infer all properties of a linear time-invariant system.

If we know the impulse response of an LTI system, we can compute its response to any input without using the actual system $\boldsymbol{y(n)} = \boldsymbol{x(n)} * \boldsymbol{h(n)}$. Furthermore, if we have no access to the internal implementation of the system (that is, we treat the system as a black box), we can try to "reverse-engineer" the system from its impulse response. Convolution describes how a linear time-invariant system modifies the input sequence to produce its output.
Convolving two waveforms in time domain is equivalent to multiply their spectra (frequency content) in frequency domain.

PROCEDURE:

```
clear all;
close all;
clc;
h = [3 2 1 -2 1 0 -4 0 3]; %impulse response
org_h = 1; %sample number at which the origin of h(n) exists
```

```
nh = [1:length(h)] - org_h; %index vector for h(n)
x = [1 -2 3 -4 3 2 1]; %input sequence
org_x = 1; %sample number at which the origin of x(n) exists
nx = [1:length(x)] - org_x; %index vector for x(n)
y = conv(h,x);
ny = [nh(1)+nx(1) : nh(end)+nx(end)];
figure,
subplot(3,1,1),
stem(nh,h,'filled');
xlabel('time index n');
ylabel('amplitude');
xlim([nh(1)-1 nh(end)+1]);
title('impulse response h(n)');
grid;
subplot(3,1,2),
stem(nx,x,'filled');
xlabel('time index n');
ylabel('amplitude');
xlim([nx(1)-1 nx(end)+1]);
title('input signal x(n)');
grid;
subplot(3,1,3),
stem(ny,y,'r','filled');
xlabel('time index n');
ylabel('amplitude');
xlim([ny(1)-1 ny(end)+1]);
title('output obtained by convolution y(n)');
grid;
```

EXERCISE:

Task-1 : What will happen if we provide input x(n) = {0,0,1,0,0} into the given system.

Task-2

a) Can you prove commutative property of the convolution?

b) Modify the code to prove Associative and Distributive properties of the convolution.

Task-3: What are the practical methods for measuring impulse response of an acoustic space?

Task-4: What is Convolution reverb? Also explain how special effects can be created in sound using convolution?

Task-5: Discuss the difference between echo and reverberation?

RESULT:

LAB SESSION 3

Effects of Quantization in Discrete Time Continuous Valued Signals

OBJECTIVE:

1. Simulate a DTCV sinusoid of 1/50 cycles/sample with length of the signal be 500 samples.
2. Choose the method of Quantization (round-off, floor & ceil) and apply to the signal generated above.
3. Compute the error signals and SQNR.
4. Explore and observe.

INTRODUCTION:

The digitization of any real world signal usually involves two stages: sampling, i.e. the measurement at discretely spaced time intervals, and quantization, i.e. the transformation of the measurements (amplitudes) into finite-precision numbers (allowed discrete levels), such that they can be represented in computer memory. Quantization leads to a loss in the signal quality, because it introduces a “quantization error” or “quantization noise” or “distortion”. The number of bits used for each sample, called the bit depth (bits/sample), determines the precision with which the sample amplitudes can be represented. Each bit in a binary number holds either a 1 or a 0. The largest decimal number that can be represented with an b-bit binary number is 2b - 1, and the number of different values that can be represented is 2b. For example, the decimal values that can be represented with an 8-bit binary number range from 0 to 255, so there are 256 different values. A bit depth of 8 allows 256 different discrete levels at which samples can be approximated or recorded. A bit depth of 16 allows 216 = 65,536 discrete levels, which in turn provides much higher precision than a bit depth of 8. With a bit depth of 3, 23 = 8 quantization levels ranging from -4 to 3 are possible. By convention, half of the quantization levels are below the horizontal axis (that is 2b-1, of the quantization levels). One level is the horizontal axis itself (level 0), and 2b-1 -1 levels are above the horizontal axis. Quantization error represents the quality of quantization process but the total error may also turn out to be zero, so another quality deciding parameter is used which is called signal-to-quantization-noise-ratio (SQNR) and it can be defined as SQNR = 10 log10 (Px/Pe) where Px and Pe are average powers of the DTCV and quantization error signal.

$$P_x = \frac{1}{N}\sum_{n=0}^{N-1}\left|x(n)\right|^2 \text{ and. } P_e = \frac{1}{N}\sum_{n=0}^{N-1}\left|x_q(n) - x\ (n)\right|^2$$

PROCEDURE:

a) Effects of Quantization to desired level of precision

```
 clear all; close all; clc;
fd = 1/50; n = [0: 499];
q = input('No. of digits after decimal points to be retained (0-9): ');
x = 2*cos(2*pi*fd*n);
Px = sum(abs(x).^2)/length(x);
xq = round(x*10^q)/10^q;
xe = xq- x;
Pe = sum(abs(xe).^2)/length(xe);
SQNR = 10*log10(Px/Pe); disp(['The signal to Quantization Noise Ratio is: '
num2str(SQNR) ' dB.' ]);
figure,
subplot(2,1,1);
stem(n,x);
hold;
stem(n,xq, 'r');
grid;
xlabel('indices');
ylabel('Amp');
xlim([0 49]);
ylim([-2.1 2.1]);
legend('DTCV','DTDV');
subplot(2,1,2);
plot(n,xe, 'k');
grid;
xlabel('indices');
ylabel('Error');
xlim([0 49]);
```

b) Relationship between SQNR and significant digits

```
clear all;
close all;
clc;
fd = 1/50;
n = [0 : 499 ];
q = [0 : 10 ]; %No. of Digits after decimal points to be retained
x = 2*cos(2*pi*fd*n);
Px = sum(abs(x).^2)/length(x);
for num = 1:length(q)
x1q = round(x*10^q(num))/10^q(num);
x1e = x-x1q;
Pe1 = sum(abs(x1e).^2)/length(x1e);
SQNR1(num) = 10*log10(Px/Pe1);
x2q = floor(x*10^q(num))/10^q(num);
x2e = x-x2q;
Pe2 = sum(abs(x2e).^2)/length(x2e);
SQNR2(num) = 10*log10(Px/Pe2);
x3q = ceil(x*10^q(num))/10^q(num);
x3e = x-x3q;
Pe3 = sum(abs(x3e).^2)/length(x3e);
SQNR3(num) = 10*log10(Px/Pe3);
end
figure,
plot(q,SQNR1,q,SQNR2,q,SQNR3,'--','Linewidth', 2);
legend('Round-off','Floor','Ceil')
```

```
grid;
xlabel('Significant Digits');
ylabel('SQNR (dB)');
xlim([q(1) q(end)]);
```

EXERCISE/TASK

Task-1 Modify the code P031.m (discussed in lab) such that it takes method of quantization from user as an input.

Task-2 Consider the sampled signal: x(n)= sin(2pi fd n)
a) For fd = 1/50 cycles/sample and N = 200 samples, write a program to quantize the signal x(n), using round-off to 64, 128 and 256 quantization levels. In each case plot the signals x(n), xq(n)and xe(n)and compute the corresponding SQNR.
b) Repeat (a) by using floor instead of rounding.
c) Comment on the results obtained in (a) and (b).
d) Compare the experimentally measured SQNR with the theoretical SQNR predicted by the following formula and comment on the differences and similarities.
SQNR = 1.76 + 6.02*b [Note: b = No. of bits of Quantizer or bit depth]

Task-3
a) Develop a MATLAB code to quantize both speech and audio signals (other than speech) to the desired number of levels.
b) Play both the original and the quantized sounds in each case when rounding and floor are used. Which quantization method leads to the best quality sound after quantization?
c) For both recorded speech signal and any ringtone, is there a point at which the signal quality deteriorates drastically? At what point (if any) does it become incomprehensible?
d) Which signal's quality deteriorate faster as the number of levels decreases?
e) What improvement in terms of SQNR can we expect from adding one bit in the quantizer?
f) Do you think 4 bits/sample is acceptable for telephone systems? What about 2 bits/sample?

Task-4 Develop a MATLAB code to import any image file and quantize it to the fewer bits/pixel than original and observe the images. For example an 8-bit image can be quantized to 7, 6, 5, 4 bits/pixel. Print hard copies of the 7, 6, 5, and 4 bits/pixel images, as well as the original. Describe the artifacts (errors) that appear in the image as the number of bits is lowered? Also note the number of bits/pixel at which the image quality noticeably deteriorates.

RESULT:

LAB SESSION 4

Application of Correlation:

OBJECTIVE:

1. Detection of periodic signal buried in noise!
   a) Given a sine wave signal (amplitude and frequency of your choice and 100 samples on x-axis).
   b) Apply auto-correlation of the sine wave signal and plot the output
   c) Generate AWGN signal of equal number of samples (100)
   d) Plot the noise signal
   e) Add the sine wave signal (generated previously) to the noisy signal and plot the graph showing the corrupt signal
   f) Now, apply auto-correlation of the sine + noise signal and plot the output. What do you observe? Write your conclusion.

THEORY:

Correlation is given as:

$R_{XY}(l) = \Sigma\, x(n)\, y(n-l) = \Sigma\, x(n+l)\, y(n)$; where 'l ' is the lag.

This is called cross-correlation and it gives the magnitude and location of similarity between two signals. The correlation between y(n) and x(n) is not necessarily the same as the correlation between x(n) and y(n).It is given as:

$$r_{yx}(l) = \Sigma\, y(n)x(n-l) = \Sigma\, y(n+l)x(n).$$

Generally, $r_{xy}(l) = r_{yx}(-l)$.

These two are the same when x(n) and y(n) are the same signals or when x(n) and y(n) are even symmetric signals. The length of the resulting correlation sequence is N+M-1, where, N and M are the lengths of the two signals. Correlation may also be computed using convolution algorithm with a modification that we need to fold one of the signals before applying the convolution.

Mathematically, $r_{xy}(n) = x(n) * y(-n)$.

.

PROCEDURE:

```
clc; clear all; close all;
F=1000;
Fs=6000;
n=[0:100];
Fd=F/Fs;
x=sin(2*pi*Fd*n);
figure(1);
plot(x);
xlabel('samples');
```

```
ylabel('amplitude');
title('Discrete signal');

figure(2);
Rxx=xcorr(x);
plot(Rxx);
xlabel('samples');
ylabel('amplitude');
title('Autocorrelate signal');
figure(3);
SNR=-1;
w=awgn(x,SNR);
plot(w);
xlabel('samples');
ylabel('amplitude');
title('Noise signal');
figure(4);
y=x+w;
plot(y);
xlabel('samples');
ylabel('amplitude');
title('Sinusoid + noise signal');
figure(5);
Ryy=xcorr(y);
plot(Ryy);
xlabel('samples');
ylabel('amplitude');
title('Auto correlate signal');
```

RESULT:

LAB SESSION 5

Discrete Fourier Transform

OBJECTIVE:

1. Load an audio file 'noisy.wav' into MATLAB.
2. There is a tone added to the speech in this file. The objective is to find the frequency of this tone.
3. Computing the DFT of this signal
4. Generating frequency vector in Hz.
5. Displaying the spectrum (DFT) of audio file and observing the frequency of the added tone.

THEORY:

With the increased use of digital computers and specialized hardware in digital signal processing (DSP), we are normally interested in those transforms that are suitable for digital computations. Because of the continuous nature of spectrum provided by DTFT, direct implementation of the DTFT is not suitable on such digital devices. In this lab we apply the Discrete Fourier Transform (DFT), which can be computed efficiently on digital computers and other DSP boards. The most distinguishing feature of DFT is that it can be applied to any type of signal whether periodic or non periodic. It simply treats all signals as the time limited or finite length sequence which corresponds to signal of interest/region of interest/sample interval denoted by N.

The DFT is an extension of the DTFT for time-limited sequences with an additional restriction that the spectrum is discretized to a finite set of values. The number of the frequency samples can have any value, but is typically set equal to the length of the time-limited sequence x(n).

DFT plays a central role in spectral analysis, the implementation of digital filters, and a variety of other signal processing applications. DFT is a powerful reversible operation for finite segments of discrete time sequences. Many "fast" algorithms have been developed for computing the DFT, and collectively these are known as Fast Fourier Transform (FFT) algorithms. The discovery of FFT algorithms established the DFT as one of the fundamental tools in digital signal processing.

DFT analysis equation: $X(k)=\sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}$ k=0,1,…,N-1

DFT synthesis equation: $x(n)=1/N \sum_{k=0}^{N-1} X(k)e^{j2\pi kn/N}$ n=0,1,….,N-1

Frequency resolution is given by ΔF=Fs/N (Hz)

Analysis frequencies or bin frequencies are given by $F_k=k\times\Delta F$ $k=0,1,..,N-1$

Note: Always keep in mind that an FFT algorithm is not a different mathematical transform: it is simply an efficient means to compute the DFT.

PROCEDURE:

```
clear all; close all; clc;

[y,Fs,bits]=wavread('noisy.wav');
n = [0:length(y)-1];
Ts = 1/Fs;
t = n.*Ts;
k = n;
Df = Fs./length(y); % Frequency resolution
F = k.*Df; % Analysis frequencies
Y = fft(y);
magY = abs(Y);
sound(y,Fs);
plot(t,y);
grid;
xlabel('time(sec)');
ylabel('Amplitude');
figure,
subplot(2,1,1);
plot(F,magY);
grid on;
xlim([0 Fs/2]);
xlabel('Frequency (Hz)');
ylabel('DFT Magnitude');
title('Discrete Fourier Transform');
subplot(2,1,2);
plot(F,magY);
grid on;
xlim([0 2000]);
xlabel('Frequency (Hz)');
ylabel('DFT Magnitude');
title('Discrete Fourier Transform');
```

EXERCISE/TASK:

- Try to remove the tone spikes by observing the spectra of the noisy signal.
- Take Inverse DFT (ifft) of the modified spectra.
- Listen to this new time-domain signal and see if the tone noise is removed.

RESULT:

LAB SESSION 6

Discrete Fourier Transform & circular convolution

OBJECTIVE:

Consider the following given sequences, $x1(n) = \{1,2,1,2\}$ and $x2(n) = \{1,2,3,4\}$ and apply circular convolution and verify the result using DFT method.

THEORY:

Circular convolution is another way of finding the convolution sum of two input signals. It resembles the linear convolution, except that the sample values of one of the input signals is folded and right shifted before the convolution sum is found. Also note that circular convolution could also be found by taking DFT of the two input signals and finding the product of two frequency domain signals. The inverse DFT of the product would give the output of the signal in the time domain which is the circular convolution output.

PROCEDURE:

```
clc;  clear all;close all;
x1=input('enter sequence 1:')     %[2,1,2,1];
x2=input('enter sequence 2:')      %[1,2,3,4];
n1=length(x1);
n2=length(x2);
if n1>n2
    nmax=n1
else
    nmax=n2
end
disp('Circular Convolution')
x4=cconv(x1,x2,nmax)
 y1=fft(x1);
y2=fft(x2);
disp('Using DFT')
y3=ifft(y1.*y2)
```

RESULT:

LAB SESSION 7

Spectral Analysis: Windowing, Zero-Padding, and FFT

OBJECTIVE:

To study the effect of windowing and zero padding w.r.t FFT on a signal.

THEORY:

Spectral Analysis of Signals:

It is very common for information to be encoded in the sinusoids that form a signal. This is true of naturally occurring signals, as well as those that have been created by humans. Many things oscillate in our universe. For example, speech is a result of vibration of the human vocal cords; stars and planets change their brightness as they rotate on their axes and revolve around each other; ship's propellers generate periodic displacement of the water, and so on. The *shape* of the time domain waveform is not important in these signals; the key information is in the *frequency*, *phase* and *amplitude* of the component sinusoids. The DFT is used to extract this information.

Windowing function in Signal Processing:

In signal processing, a window function (also known as an apodization function or tapering function) is a mathematical function that is zero-valued outside of some chosen interval. For instance, a function that is constant inside the interval and zero elsewhere is called a rectangular window, which describes the shape of its graphical representation. When another function or waveform/data-sequence is multiplied by a window function, the product is also zero-valued outside the interval: all that is left is the part where they overlap, the "view through the window".

In typical applications, the window functions used are non-negative, smooth, "bell-shaped" curves. Rectangle, triangle, and other functions can also be used. A more general definition of window functions does not require them to be identically zero outside an interval, as long as the product of the window multiplied by its argument is square integrable, and, more specifically, that the function goes sufficiently rapidly toward zero.

Zero Padding:

Zero padding is a simple concept; it simply refers to adding zeros to end of a time- domain signal to increase its length.

MATLAB CODE

- **w = blackman(M)** returns the M-point Blackman window function in array w.

PROCEDURE:

```
M = 64;w = blackman(M);
figure(1);
subplot(3,1,1); plot(w,'*'); title('Blackman Window');
xlabel('Time (samples)'); ylabel('Amplitude');
 % Also show the window transform:
zpf = 8;                        % zero-padding factor
xw = [w',zeros(1,(zpf-1)*M)]; % zero-padded window
Xw = fft(xw);                   % Blackman window transform
spec = 20*log10(abs(Xw));      % Spectral magnitude in dB
spec = spec - max(spec);       % Normalize to 0 db max
nfft = zpf*M;
spec = max(spec,-100*ones(1,nfft)); % clip to -100 dB
fni = [0:1.0/nfft:1-1.0/nfft];   % Normalized frequency axis
subplot(3,1,2); plot(fni,spec,'-'); axis([0,1,-100,10]);
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)'); grid;
 % Replot interpreting upper bin numbers as frequencies<0:
nh = nfft/2;
specnf = [spec(nh+1:nfft),spec(1:nh)];  % see fftshift()
fninf = fni - 0.5;
subplot(3,1,3);
plot(fninf,specnf,'-');
axis([-0.5,0.5,-100,10]); grid;
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)');
figure(2);
% Windowed, zero-padded data:
n = [0:M-1];           % discrete time axis
f = 0.25 + 0.5/M;      % frequency
xw = [w'.*cos(2*pi*n*f),zeros(1,(zpf-1)*M)];
% Smoothed, interpolated spectrum:
X = fft(xw);
 % Plot time data:
subplot(2,1,1);
plot(xw);
title('Windowed, Zero-Padded, Sampled Sinusoid');
xlabel('Time (samples)');
ylabel('Amplitude');
 spec = 10*log10(conj(X).*X);  % Spectral magnitude in dB
spec = max(spec,-60*ones(1,nfft)); % clip to -60 dB
subplot(2,1,2);
plot(fninf,fftshift(spec),'-');
axis([-0.5,0.5,-60,40]);
title('Smoothed, Interpolated, Spectral Magnitude (dB)');
xlabel('Normalized Frequency (cycles per sample))');
ylabel('Magnitude (dB)'); grid
```

RESULT:

LAB SESSION 8(a)

Relationship between Laplace and CTFT

OBJECTIVE*:*

1. Generate pole zero constellation in s plane.
2. Plot corresponding Frequency (Bode magnitude) response.
3. Plot impulse response and determine that the system is FIR or IIR.
4. Modify location of poles in s plane to observe the corresponding change in frequency and impulse response**.**

THEORY:

The Laplace Transform of a general continuous time signal x (t) is defined as;

$$X(S) = \int_{-\infty}^{\infty} x\,(t)\, e^{-st}\, dt$$

Where the complex variable s=δ + j w, with δ and w the real and imaginary parts. CTFT is a subset of Laplace when δ =0. Since 'δ' information is not present in CTFT, therefore information about stability can only be obtained from Laplace. If pole lies on L.H.S of splane, system is stable. If pole lies on R.H.S of s-plane, system is unstable. If pole lies on y(jw)-axis, system is marginally stable or oscillatory. If system has FIR, it is stable. If system is IIR, it can be stable or unstable.

PROCEDURE*:*

1. Make a folder at desktop and name it as your current directory within MATLAB.
2. Open M-file editor and write the following code:

```
clear all;
close all;
 clc;
Num = poly([(0-(i*(pi/2))),(0+(i*(pi/2)))]);
Zeros=roots(Num)
Den = poly([-1,-1]);
poles=roots(Den)
sys=tf(Num,Den)
figure;
subplot(3,1,1);
pzmap(sys);
xlim([-2 2]);
ylim([-4 4]);
subplot(3,1,2);
[mag phase w]=bode(sys);
```

```
 mag=squeeze(mag);
 plot(w,mag);
subplot(3,1,3);
 impulse(sys);
H=dfilt.df1(Num,Den);
A=isfir(H)
```

RESULT:

LAB SESSION 8(b)

Relationship between z transform and DTFT

OBJECTIVE:

1. Generate pole zero constellation in z plane.
2. Plot corresponding Frequency (Bode magnitude) response.
3. Plot impulse response and determine that the system is FIR or IIR.
4. Modify location of poles in z plane to observe the corresponding change in frequency and impulse response.

THEORY:

The z - Transform of a general discrete time signal x(n) is defined as;

$$X(z)=\sum_{n=-\infty}^{n=\infty} x(n)\, z^{-n}$$

Where the complex variable z=r ∠w, with r the radius and w the angle. DTFT is a subset of z transform when r =1. Since 'r' information is not present in DTFT, therefore information about stability in discrete time can only be obtained from z transform. If pole lies inside the unit circle, system is stable. If pole lies outside the unit circle, system is unstable. If pole lies at the unit circle, system is marginally stable or oscillatory. If system has FIR, it is stable. If system is IIR, it can be stable or unstable.

PROCEDURE:

1. Make a folder at desktop and name it as your current directory within MATLAB.
2. Open M-file editor and write the following code:

```
clear all;
close all;
 clc;
Num = poly([(0-(i*(pi/2))),(0+(i*(pi/2)))]);
Den = poly([-1,-1]);
Num1 = poly([j,-j]);
Den1 = poly([exp(-1),exp(-1)]);
sys1=tf(Num1,Den1,1)
figure;
subplot(3,1,1);
pzmap(sys1);
xlim([-2 2]);
ylim([-4 4]);
subplot(3,1,2);
[mag phase w]=bode(sys1);
mag=squeeze(mag);
plot(w,mag);
xlim([0 100])
```

```
subplot(3,1,3);
impulse(sys1);
H=dfilt.df1(Num,Den);
 A=isfir(H)
figure;
pzmap(sys1)
grid on;
```

EXERCISE:

Change the location of poles from inside the unit circle to outside and at the unit circle and observe the effects.

RESULT:

LAB SESSION 9

FIR filters

OBJECTIVE:

To design of FIR filters using MATLAB

THEORY:

Digital filters refers to the hard ware and software implementation of the mathematical algorithm which accepts a digital signal as input and produces another digital signal as output whose wave shape, amplitude and phase response has been modified in a specified manner.

Digital filter play very important role in DSP. Compare with analog filters they are preferred in number of application due to following advantages.

- Truly linear phase response
- Better frequency response
- Filtered and unfiltered data remains saved for further use.

There are two type of digital filters.

- FIR (finite impulse response) filter
- IIR (infinite impulse response) filter

1. FIR filters are Finite Impulse Response filters with no feedback, whereas IIR contains feedback.
2. Transfer function of FIR filter does not contain any non-trivial poles. Their frequency response is solely dependent on zero locations. IIR filters contain poles as well as zeros.
3. As there are no poles, FIR filters cannot become unstable; however, IIR filters can become unstable if any pole lies outside the unit circle in z-plane.
4. More number of coefficients is needed to design the desired filter in FIR than IIR.

Commands used in FIR Filter Design

FIR1 designs FIR filter using the window method. B = FIR1(N,Wn) designs an N'th order low pass FIR digital filter and returns the filter coefficients in length N+1 vector B. **The cutoff frequency Wn must be between 0 < Wn < 1.0, with 1.0 corresponding to half the sample rate**. The filter B is real and has linear phase. The normalized gain of the filter at Wn is -6 dB.

B = FIR1(N,Wn,'high') designs an N'th order highpass filter.

You can also use B = FIR1(N,Wn,'low') to design a lowpass filter.

If Wn is a two-element vector, Wn = [W1 W2], FIR1 returns an order N bandpass filter with passband W1 < W < W2.

B = FIR1(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2]. You can also specify If Wn is a multi-element vector, Wn = [W1 W2 W3 W4 W5 ... WN], FIR1 returns an order N multiband filter with bands 0 < W < W1, W1 < W < W2, ..., WN < W < 1.

B = FIR1(N,Wn,'DC-1') makes the first band a passband.
B = FIR1(N,Wn,'DC-0') makes the first band a stopband.

By default FIR1 uses a Hamming window. Other available windows, including Boxcar, Hann, Bartlett, Blackman, Kaiser and Chebwin can be specified with an optional trailing argument. For example, B = FIR1(N,Wn,kaiser(N+1,4)) uses a Kaiser window with beta=4.
B = FIR1(N,Wn,'high',chebwin(N+1,R)) uses a Chebyshev window.

freqz returns the frequency response based on the current filter coefficients.

PROCEDURE:

Designing a Low Pass Filter that passes all frequencies below 1200 Hz

```
% Sampling Frequency in samples/sec
Fs = 8000;
t = [0 : 1/Fs : 1]; % Time Vector
F=1200/(Fs/2);
N=64;
b=fir1(N,F,'low'); % Zeros of the filter

% Magnitude Plot of the filter
 [H,w]=freqz(b,1,64,8000);
subplot(2,1,1)
plot(w,abs(H)); % Normalized Magnitude Plot
xlim([0 Fs/2]);
title('Frequency response of Low Pass Filter');
xlabel('Frequency-Hz')
ylabel('Normalised Magnitude');

% Pole Zero Constellations
[b,a] = eqtflength(b,1); % Equalize lengths of transfer function's numerator
and denominator
[z,p,k] = tf2zp(b,a);
subplot(2,1,2),
zplane(b,a);
title('Low Pass Filter');
```

Repeat similar procedure to implement:

- High Pass Filter that pass all frequencies above 1200 Hz
- Band Pass Filter that passes frequencies between 1200 and 1800 Hz
- Band stop Filter that rejects frequencies between 1200 and 1800 Hz
- Notch filter that reject frequency between 1500 and 1550 Hz

TASK:

1. Create a signal vector containing two frequencies as: i) 100 Hz. and ii) 150 Hz, with Fs = 1000 Hz.
2. Design two bandpass FIR filters with 64 coefficients and with passbands as:
   i) 75 to 125 Hz. and ii) 125 to 175 Hz.
3. Use both filters on the created signal and observe their outputs.
4. Plot frequency responses and pole-zero constellations of both filters and note observations.

```
close all;
clear all;
clc;
% Frequencies in Hz
f1 = 100;
f2 = 150;

% Sampling Frequency in samples/sec
Fs = 1000;
t = [0 : 1/Fs : 1]; % Time Vector
F = Fs*[0:length(t)-1]/length(t); % Frequency Vector
x = exp(j*2*pi*f1*t)+2*exp(j*2*pi*f2*t);% Signal Vector
b1 = fir1( 64 , [75 125]/500); % filter coeffs. for bandpassing F1
b2 = fir1( 64 , [125 175]/500); % filter coeffs. for bandpassing F2
[h1,w1]=freqz(b1,1,length(t),'whole'); % Frequency response for filter 1
[h2,w2]=freqz(b2,1,length(t),'whole'); % Frequency response for filter 2

% Filter operation - see filtfilt in help to learn what it does
y1 = filtfilt(b1,1,x);
y2 = filtfilt(b2,1,x);

% Plotting
figure,
subplot(5,1,1)
plot(F,abs(fft(x)));
xlim([0 Fs/2]);
title('FFT of orgianl signal');
subplot(5,1,2),
plot(F,abs(h1));
xlim([0 Fs/2]);
title('Frequency response of Filter One');
subplot(5,1,3),
plot(F,abs(fft(y1)));
xlim([0 Fs/2]);
title('FFT of filtered signal from filter one');
subplot(5,1,4),
plot(F,abs(h2));
xlim([0 Fs/2]);
title('Frequency response of Filter Two');
subplot(5,1,5),
plot(F,abs(fft(y2)));
```

```
xlim([0 Fs/2]);
title('FFT of filtered signal from filter two');
xlabel('Hz.')

% Pole Zero Constellations
[b1,a1] = eqtflength(b1,1);
[z1,p1,k1] = tf2zp(b1,a1);
[b2,a2] = eqtflength(b2,1);
[z2,p2,k2] = tf2zp(b2,a2);

figure,
subplot(1,2,1),
zplane(b1,a1);
xlim([-1.5 1.5]);
ylim([-1.5 1.5]);
title('Filter One');
subplot(1,2,2),
zplane(b2,a2);
xlim([-1.5 1.5]);
ylim([-1.5 1.5]);
title('Filter Two');
```

RESULT:

LAB SESSION 10

IIR filters

OBJECTIVE:

To design of IIR filters using MATLAB.

THEORY:

Matlab contains various routines for design and analyzing digital filter IIR. Most of these are part of the signal processing tool box. A selection of these filters is listed below.

- Buttord (for calculating the order of filter)
- Butter (creates an IIR filter)
- Ellipord (for calculating the order of filter)
- Ellip (creates an IIR filter)
- Cheb1ord (for calculating the order of filter)
- Cheyb1 (creates an IIR filter)

Buttord- Butterworth filter order selection [N, Wn] = BUTTORD(Wp, Ws, Rp, Rs)
Ellipord- Elliptic filter order selection [N, Wn] = ELLIPORD(Wp, Ws, Rp, Rs)
Cheb1ord- Chebyshev Type I filter order selection [N, Wn] = CHEB1ORD(Wp, Ws, Rp, Rs)

returns the order N of the lowest order digital Butterworth, elliptic and Chebyshev Type I filter respectively that loses no more than Rp dB in the pass band and has at least Rs dB of attenuation in the stop band. Wp and Ws are the pass band and stop band edge frequencies, normalized from 0 to 1 (where 1 corresponds to pi radians/sample).

BUTTORD also returns Wn, the Butterworth natural frequency (or, the "3 dB frequency") to use with BUTTER to achieve the specifications.

ELLIPORD also returns Wn, the elliptic natural frequency to use with ELLIP to achieve the specifications.

CHEB1ORD also returns Wn, the Chebyshev natural frequency to use with CHEBY1 to achieve the specifications.

Butter- Butterworth digital and analog filter design

[B,A] = BUTTER(N,Wn) designs an Nth order lowpass digital Butterworth filter and returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cutoff frequency Wn must be $0.0 < Wn < 1.0$, with 1.0 corresponding to half the sample rate.

If Wn is a two-element vector, Wn = [W1 W2], BUTTER returns an order 2N bandpass filter with passband W1 < W < W2. [B,A] = BUTTER(N,Wn,'stop') is a bandstop filter.

[B,A] = BUTTER(N,Wn,'high') designs a highpass filter.

When used with three left-hand arguments, as in [Z,P,K] = BUTTER(...), the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K.

Ellip- Elliptic or Cauer digital and analog filter design

[B,A] = ELLIP(N,Rp,Rs,Wn) designs an Nth order low pass digital elliptic filter with Rp decibels of peak-to-peak ripple and a minimum stop band attenuation of Rs decibels. ELLIP returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator).The cutoff frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. Use Rp = 0.5 and Rs = 20 as starting points, if you are unsure about choosing them.

If Wn is a two-element vector, Wn = [W1 W2], ELLIP returns an order 2N band pass filter with pass band W1 < W < W2. [B,A] = ELLIP(N,Rp,Rs,Wn,'stop') is a band stop filter if Wn = [W1 W2].

[B,A] = ELLIP(N,Rp,Rs,Wn,'high') designs a high pass filter.

When used with three left-hand arguments, as in [Z,P,K] = ELLIP(...), the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K.

Cheby1- Chebyshev Type I digital and analog filter design

[B,A] = CHEBY1(N,R,Wn) designs an Nth order lowpass digital Chebyshev filter with R decibels of peak-to-peak ripple in the passband. CHEBY1 returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The cutoff frequency Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to half the sample rate. Use R=0.5 as a starting point, if you are unsure about choosing R.

If Wn is a two-element vector, Wn = [W1 W2], CHEBY1 returns an order 2N bandpass filter with passband W1 < W < W2.

[B,A] = CHEBY1(N,R,Wn,'high') designs a highpass filter.

[B,A] = CHEBY1(N,R,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

When used with three left-hand arguments, as in [Z,P,K] = CHEBY1(...), the zeros and poles are returned in length N column vectors Z and P, and the gain in scalar K.

PROCEDURE:

Buttord and Butter Filter:

Designing IIR Low Pass Filter:

Suppose our target is to design a filter to pass all frequencies below 1200 Hz with pass band ripples = 1 dB and minimum stop band attenuation of 50 dB at 1500 Hz. The sampling frequency for the filter is 8000 Hz;

```
fs=8000;
[n,w]=buttord(1200/4000,1500/4000,1,50); % finding the order of the filter
[b,a]=butter(n,w); % finding zeros and poles for filter
figure(1)
freqz(b,a,512,8000);
figure(2)
[h,q] = freqz(b,a,512,8000);
plot(q,abs(h)); % Normalized Magnitude plot
xlim([0 fs/2])
xlabel('Frequncy- Hz')
ylabel('Normailized Magnitude')
grid
figure(3)
f=1200:2:1500;
freqz(b,a,f,8000) % plotting the Transition band
figure(4)
zplane(b,a) % pole zero constellation diagram
```

Designing IIR High Pass Filter:

We will consider same filter but our target now is to pass all frequencies above 1200 Hz

```
fs=8000;
[n,w]=buttord(1200/4000,1500/4000,1,50); % finding the order of the filter
[b,a]=butter(n,w,'high'); % finding zeros and poles for filter
figure(1)
freqz(b,a,512,10000);
figure(2)
[h,q] = freqz(b,a,512,10000);
plot(q,abs(h)); % Normalized Magnitude plot
xlim([0 fs/2])
xlabel('Frequncy- Hz')
ylabel('Normailized Magnitude')
grid
figure(3)
f=1200:2:1500;
freqz(b,a,f,10000) % plotting the Transition band
figure(4)
zplane(b,a) % pole zero constellation diagram
```

Designing IIR Band Pass Filter:

Now we wish to design a filter to pass all frequencies between 1200 Hz and 2800 Hz with pass band ripples = 1 dB and minimum stop band attenuation of 50 dB. The sampling frequency for the filter is 8000 Hz;

```
[n,w]=buttord([1200/4000,2800/4000],[400/4000, 3200/4000],1,50);
[b,a]=butter(n,w,'bandpass');
figure(1)
freqz(b,a,128,8000)
figure(2)
[h,w]=freqz(b,a,128,8000);
plot(w,abs(h))
```

```
grid
figure(3)
f=600:2:1200;
freqz(b,a,f,8000); % Transition Band
figure(4)
f=2800:2:3200;
freqz(b,a,f,8000); % Transition Band
figure(5)
zplane(b,a)
```

Designing IIR Band Stop Filter:

```
[n,w]=buttord([1200/4000,2800/4000],[400/4000, 3200/4000],1,50);
[b,a]=butter(n,w,'stop');
figure(1)
freqz(b,a,128,8000)
[h,w]=freqz(b,a,128,8000);
figure(2)
plot(w,abs(h));
grid
figure(3)
f=600:2:1200;
freqz(b,a,f,8000); % Transition Band
figure(4)
f=2800:2:3200;
freqz(b,a,f,8000); % Transition Band
figure(5)
zplane(b,a);
```

TASK:

Design all above filter using following commands
Ellipord( )
Ellip( )
Cheb1ord( )
Cheby1( )
Compare the results of the butter worth LPF with ellip LPF and cheby1 LPF on following basis

Filter	Order	Minimum stop band attenuation	Linearity in the phase plots within the pass band and outside	Pole –zeros plot (pole most close to the unit circle)
Butterworth				
Elliptic				
Chebyshev				

RESULT:

LAB SESSION 11

Designing filters using Signal Processing Toolkit

OBJECTIVE:

Filter designing by MATLAB toolkit.

THEORY:

Description:

The goal of filtering is to perform frequency-dependent alteration of a signal. A simple design specification for a filter might be to remove noise above a certain cutoff frequency. A more complete specification might call for a specific amount of passband ripple (Rp, in decibels). Stopband attenuation (Rs, in decibels), or transition width (Wp-Ws, in hertz).

Filter Configurations:

All the filter design functions in the Signal Processing Toolbox operate with normalized frequencies, so that they do not require the system sampling rate as an extra input argument. The normalized frequency is always in the interval $0 \leq f \leq 1$. For example, with a 1000 Hz sampling frequency, 300 Hz is 300=500 = 0.6. To convert normalized frequency to angular frequency around the unit circle, multiply by π. To convert normalized frequency back to Hertz, multiply by half the sample frequency.

Digital filter specifications are often given is terms of the loss function (in dB),

$$A(\omega) = -20\log 10|G(e^{j\omega}|$$

Filter Design with SPTool:

The Filter Designer in the SPTool allows you to design and edit IIR and FIR filters of various lengths and types, with lowpass, highpass, bandpass, and bandstop, and multiband configurations. To activate the Filter Designer, click either the New button or the Edit button under the Filters list box in SPTool.

Try:

```
sptool
```

Filter Design with FDATool:

The FDATool, supports many advanced techniques not available in SPTool. It also allow realization of Simulink models of quantized direct form FIR filters. Perform digital frequency transformations of filters.

Try:

`fdatool`

Tasks

1. Design a Lowpass, FIR Eqiripple filter, Minimum order , Fs=1000 Hz, Fpass=60 Hz, Fstop=200 Hz , Apass =1dB, Astop =8dB. Note the following observations:
    - Magnitude and phase response
    - Phase and group delay
    - Filter impulse response
    - Filter information, filter coefficient , filter parameters
    - Pole-zero plot

Export the filter parameters (Num) to your workspace. Do size (Num) to obtain the size of your FIR filter. Do freqz(Num,1). In the command window, do sptool. In sptool, Import Num to sptool. Change sampling frequency from the default 1 to 1000. Now under SPTool: startup.spt, view the signal (Recall that Num is just the values of the 18 point FIR filter; i.e. the FIR impuls response). Under Spectra, use\ create and do a 1024 point fft. Under options > Magnitude > Linear. Observe the difference between observing on a linear and log scales. Do you see the equiripples in the stopband.

2. Now in fdatool, for the same specs as the FIR filter, design an IIR Butterworth filter. (Match exactly the passband and stopbands). Look at the magnitude response. Linear phase? Look at the impulse response. (Note that it is NOT symmetric). Look at the filter information. Did it meet specs? How many poles and zeros? Does it have any properties dissimilar to that of FIR filter ? Explain.

RESULT:

LAB SESSION 12

Multi rate sampling

OBJECTIVE:

a. To generate triangular wave and its spectrum.
b. Down sample the signal by a factor and plot its spectrum. Pass signal by low pass filter and observe its spectrum by down sampling it.
c. Same work required for up sampling as in part b.

THEORY:

Multirate simply means "multiple sampling rates". A multirate DSP system uses multiple sampling rates within the system. Whenever a signal at one rate has to be used by a system that expects a different rate, the rate has to be increased or decreased, and some processing is required to do so. Therefore "Multirate DSP" really refers to the art or science of *changing* sampling rates.

PROCEDURE:

a) Down sampling:

```
clc; close all; clear all;
fc=1/4;
f=-1:1/512:1;
n=0:1024;
% Generate a sinc
x=fc.*(sinc(fc.*(n-512))).^2;
[h w]=freqz(x,1,2*pi*f);
%downsample it w/o filter
ynf=downsample(x,3);
h2=freqz(ynf,1,2*pi*f);
%Define filter
xf=filter(h,1,x);
h3=freqz(xf,1,2*pi*f);
%downsample x with filter
yf=downsample(xf,3);
h4=freqz(yf,1,2*pi*f);
%plotting
subplot(4,1,1);
plot(f,abs(h));
title('Magnitude Response of Sinc');
subplot(4,1,2);
plot(f,abs(h2));
title('Magnitude Response of Sinc w/o Filter');
subplot(4,1,3);
plot(f,abs(h3));
title('Magnitude Response of Filter');
subplot(4,1,4);
plot(f,abs(h4));
```

```
title('Magnitude Response of Sinc with Filter');
```

b) Up sampling:

```
clc; close all; clear all;
fc=1/4;
f=-1:1/512:1;
n=0:1024;
% Generate a sinc
x=fc.*(sinc(fc.*(n-512))).^2;
[h w]=freqz(x,1,2*pi*f);
%Upsample it w/o filter
ynf=upsample(x,3);
h2=freqz(ynf,1,2*pi*f);
%Define filter
xf=filter(h,1,x);
h3=freqz(xf,1,2*pi*f);
%upsample x with filter
yf=upsample(xf,3);
h4=freqz(yf,1,2*pi*f);
%plotting
subplot(4,1,1);
plot(f,abs(h));
title('Magnitude Response of Sinc');
subplot(4,1,2);
plot(f,abs(h2));
title('Magnitude Response of Sinc w/o Filter');
subplot(4,1,3);
plot(f,abs(h3));
title('Magnitude Response of Filter');
subplot(4,1,4);
plot(f,abs(h4));
title('Magnitude Response of Sinc with Filter');
```

RESULT:

LAB SESSION 13

Generation of sine waves using lookup table using C-6713 DSK

OBJECTIVE:

To generate a sinusoid and plotting it with Code Composer Studio (sine8_buf).

REQUIREMENTS:

¾ C6713 DSK(DSP Starter Kit) and its support tools ¾ Oscilloscope ¾ Head phones/ Speaker

THEORY:

This exercise generates a sinusoid with eight points. More important, it illustrates CCS capabilities for plotting in both time and frequency domains. The program used in this exercise creates a buffer to store the output data in memory.

Buffer:

Buffer is an array in which data to be written or to be read from the disk is placed. Putting the contents of a file in a buffer has certain advantages; we can perform various operations on the contents of buffer without having to access the file again. The size of the buffer is important for efficient operation. Depending on the operating system, buffers of certain sizes are handled more efficiently than others. In this program, an output buffer is created to capture a total of 256 sine data values.

Interrupt:

An interrupt can be issued internally or externally. An interrupt stops the current CPU process so that it can perform a required task initiated by the interrupt. The source of the interrupt can be an ADC, timer etc. On an interrupt; the conditions of the current process must be saved so that they can be restored after the interrupt task is performed.

PROCEDURE:

1. Connect DSK C6713 board to the USB port of PC and give +5V supply to the board.
2. Perform the diagnostic and quick tests of DSK (refer to Appendix A).
3. Connect the DSK by selecting Debug→ Connect, if it has not already been connected.
4. Create New Project (refer to Appendix B).
5. Add files to the project (refer to Appendix C).
6. Bulid the program (refer to Appendix D).
7. Select File→ Load Program in order to load sine_buf.out on the DSK.
8. Connect a speaker to the LINE OUT connector on the DSK 9. Select Debug→ Run.

OBSERVATION:

A tone of 1 kHz can be heard on speaker or can be viewed on an oscilloscope.

EXERCISE:

Plotting with Code Composer Studio (CCS):

To plot the current output data stored in the buffer using CCS, perform following steps:

1. Select View→ Graph→ Time/Frequency. Change the Graph Property Dialog so that the options in Figure 2.1 are selected for a time-domain plot. The other options can be left as default. Press OK and observe the wave pattern in time-domain within CCS.
2.
3. Now change CCS Graph Property Dialog for a frequency-domain plot according to Figure 2.2. Press OK and observe the FFT magnitude plot. The spike at 1000Hz represents the frequency of the sinusoid generated.

Display Type	Single Time
Graph Title	Graphical Display
Start Address	out_buffer
Acquisition Buffer Size	256
Index Increment	1
Display Data Size	64
DSP Data Type	16-bit signed integer
Q-value	0
Sampling Rate (Hz)	8000
Plot Data From	Left to Right
Left-shifted Data Display	Yes
Autoscale	On
DC Value	0
Axes Display	On
Time Display Unit	S
Status Bar Display	On
Display Type	FFT Magnitude
Graph Title	Graphical Display
Signal Type	Real
Start Address	out_buffer
Acquisition Buffer Size	256
Index Increment	1
FFT Frame Size	256
FFT Order	8
FFT Windowing Function	Rectangle
Display Peak and Hold	Off

RESULT:

LAB SESSION 14

Generation of sine waves using interrupts

OBJECTIVE:

To illustrate a Loop Program using interrupt (loop_intr).

REQUIREMENTS:

¾ C6713 DSK(DSP Starter Kit) and its support tools ¾ Oscilloscope ¾ Function Generator ¾ Head phones/ Speaker

THEORY:

This lab session illustrates input and output with the codec. This program example is very important since it can be used as a base program to build on. For example, to implement a digital filter, one would need to insert the appropriate algorithm between the input and output functions. An interrupt occurs every sample period Ts=1/Fs at which time an input sample value is read from the codec's ADC and then sent as output to the codec's DAC.

Interrupt:

An interrupt can be issued internally or externally. An interrupt stops the current CPU process so that it can perform a required task initiated by the interrupt. The source of the interrupt can be an ADC, timer etc. On an interrupt; the conditions of the current process must be saved so that they can be restored after the interrupt task is performed.

Loop:

We frequently need to perform an action over and over, often with variations in the details each time. The mechanism that meets this need is the "loop". In programming, it is often the case that you want to do something a fixed number of times. The loop is ideally suited for such cases. There are three major loop structures in 'C': the for loop, the while loop and the do while loop. The program in this exercise uses while loop

PROCEDURE:

1. Connect DSK C6713 board to the USB port of PC and give +5V supply to the board.
2. Perform the diagnostic and quick tests of DSK (refer to Appendix A).
3. Connect the DSK by selecting Debug→ Connect, if it has not already been connected.
4. Create New Project (refer to Appendix B).
5. Add files to the project (refer to Appendix C).
6. Build the program (refer to Appendix D).
7. Select File→ Load Program in order to load loop_intr.out on the DSK.

8. Input a sinusoidal waveform to the LINE IN connector on the DSK with an amplitude of approximately 2V p-p and a frequency between approximately 1and 3 kHz.
9. Select Debug→ Run.

OBSERVATION:

A tone of same input frequency, but attenuated to approximately 0.8V p-p, can be heard on speaker or can be viewed on an oscilloscope connected to the LINE OUT connector of the DSK.

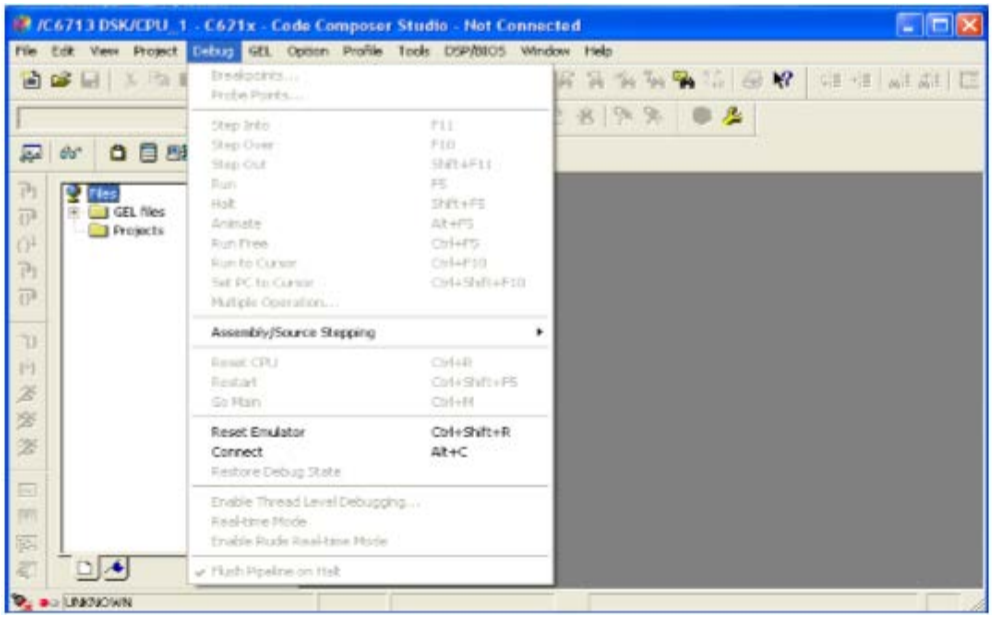
EXERCISE:

Increase the amplitude of the input sinusoidal waveform beyond 6V p-p and observe that the output signal becomes distorted. This is because; the maximum level of the input signal to the codec is 6 Vp-p.

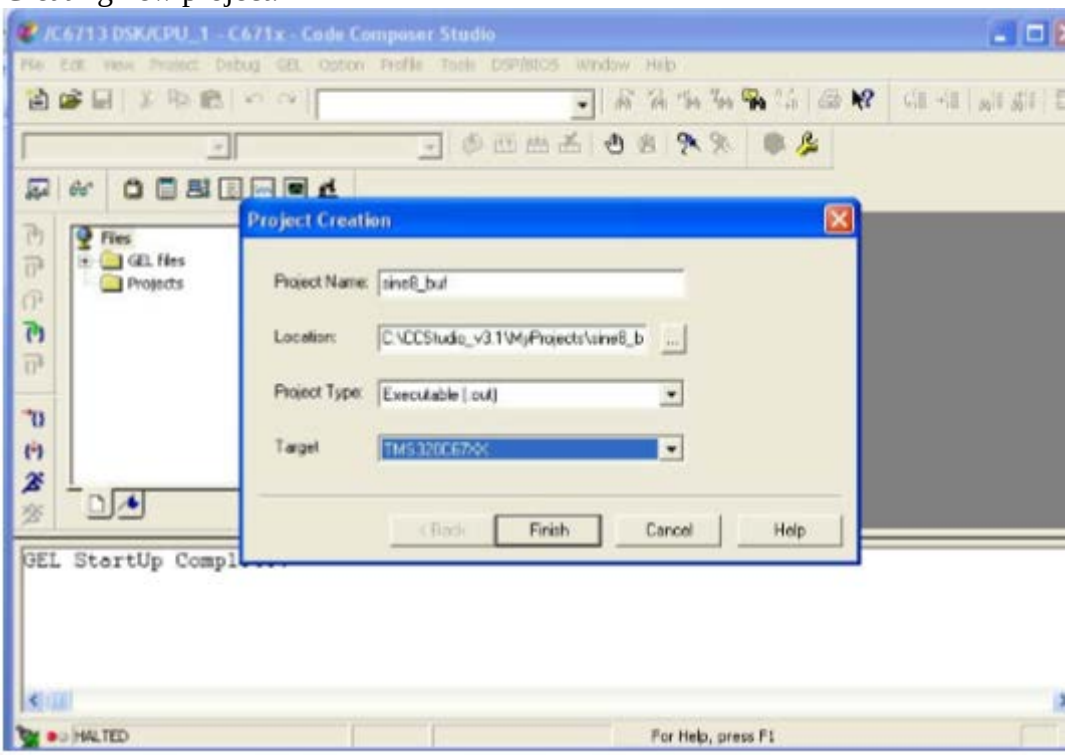
RESULT:

APPENDIX
(Compiler Details for DSP kit)

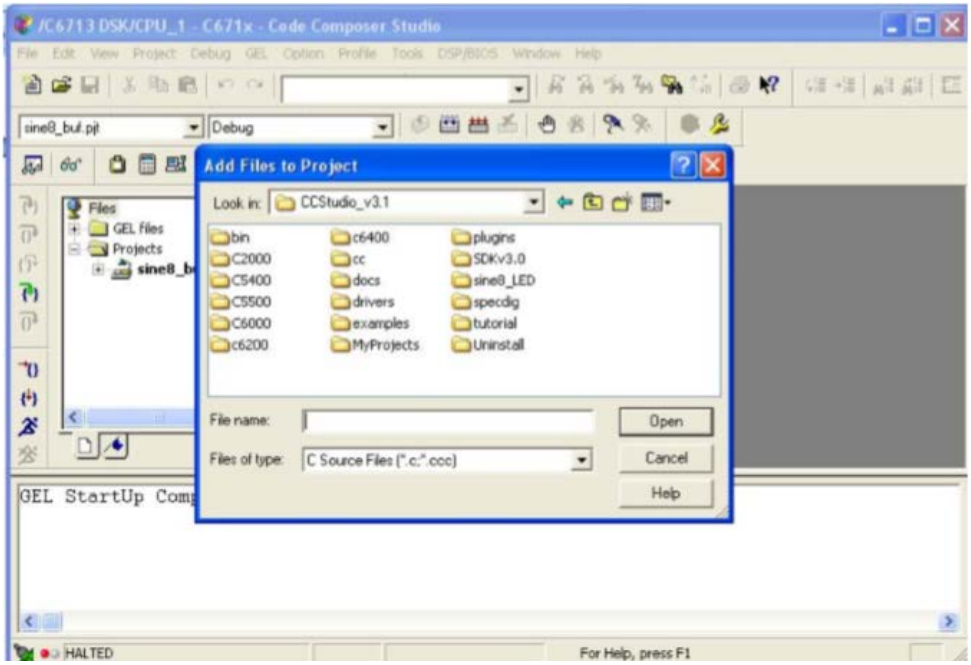
1. In debug options select connect. A message will appear that the target is now connected.



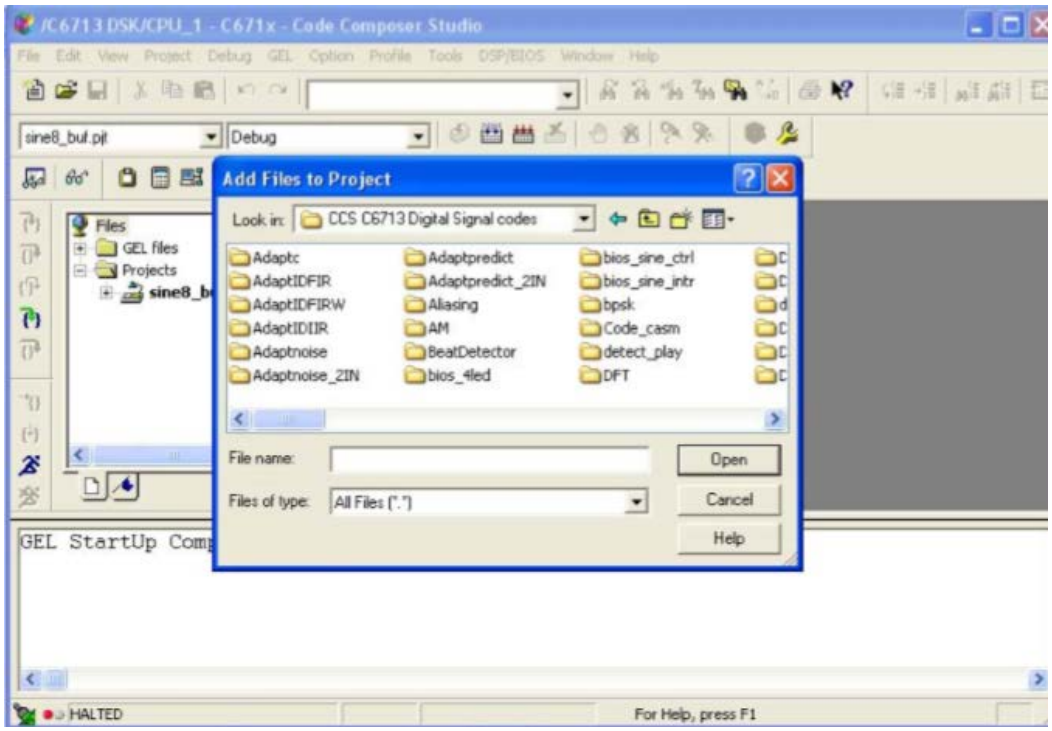
2. Creating new project.



3. Adding files to the project. Go to Project, add files to project and do the following:



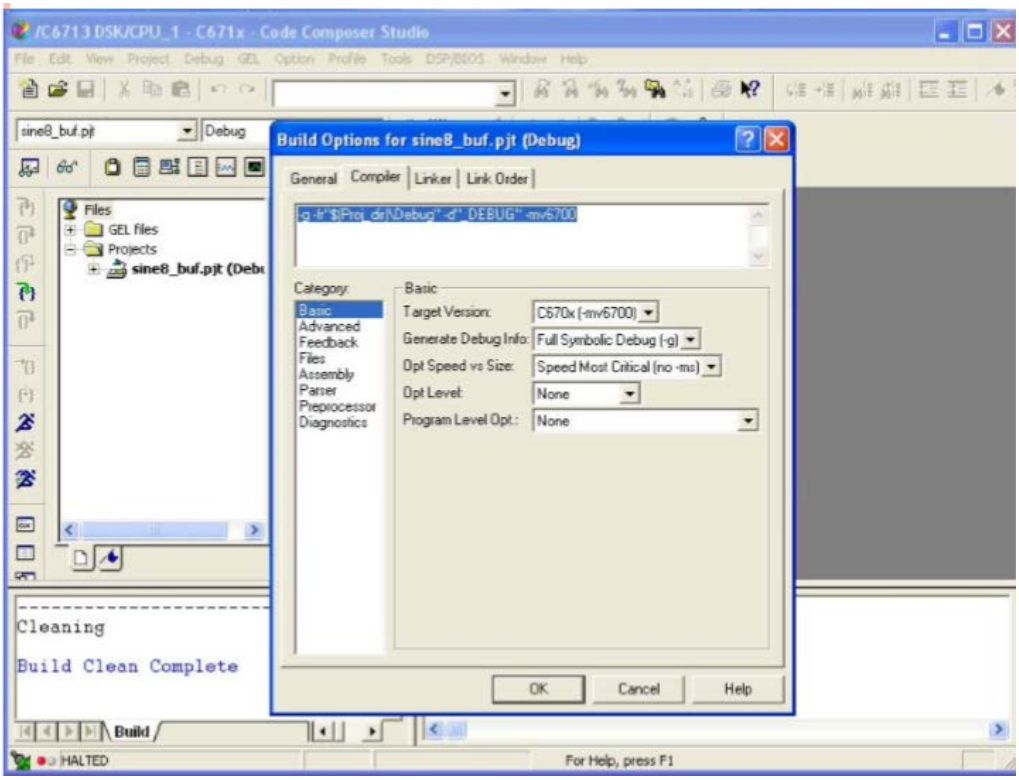
(a). Look in CCStudio_v3.1. Select folder C6000, change the file of type to All Files and add the following files:(i) Select cgtools, lib, rts6700 and click open.(ii)Select csl, lib, csl6713 and click open.(iii)Select dsk6713, lib, dsk6713bsl and click open.



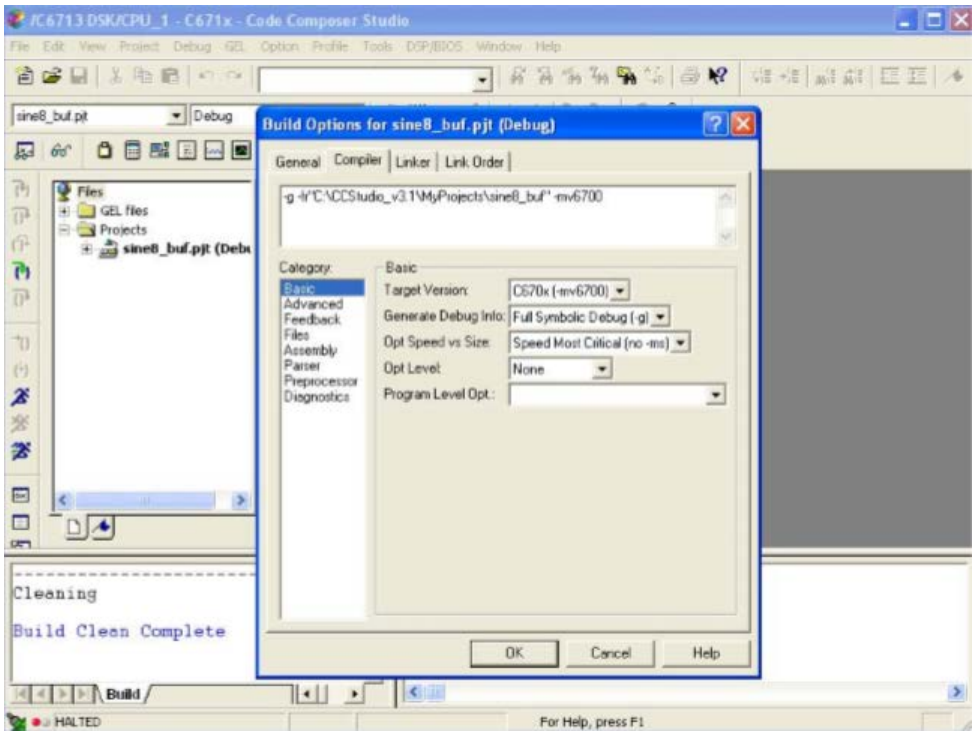
(b)Look in Desktop, select folder “CCS C6713 Digital Signal codes”.

Select folder ‘Support’ and add the following files (i)C6713dsk (ii)C6713dskinit[Type C file not ASM or H file].(iii)Select Vectors_intr or Vectors_poll according to the requirement of the code. In our program its Vectors_intr. (ii) Select sine8_buf folder and then select sine8_buf type C file.

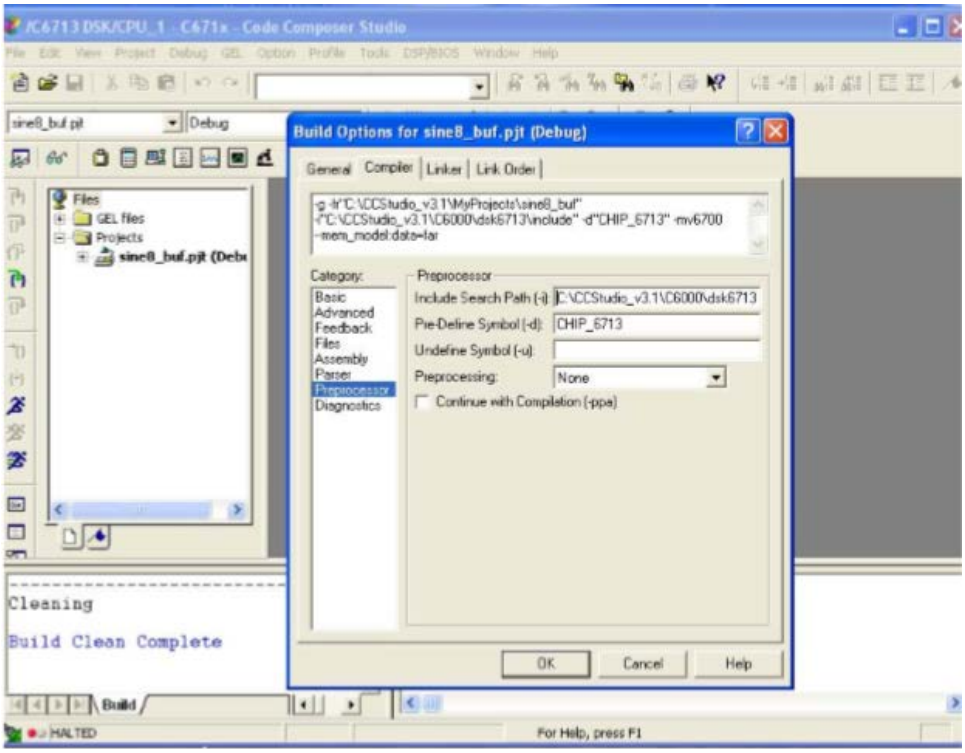
4. Building Program.



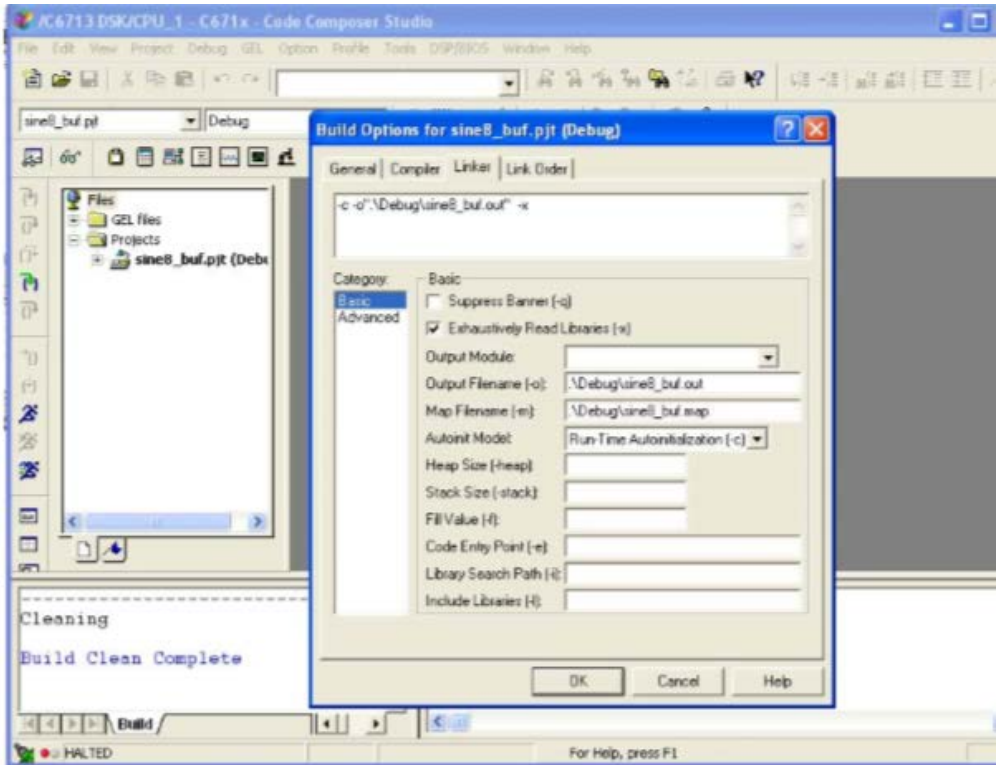
Building the program:
(a) Go to project and then build options. (i) In compiler options Basic settings, remove the location from dollar sign to DEBUG and then in quotation marks write the location of sine8_buf i.e C:\CCStudio_v3.1\MyProjects\sine8_buf.



(ii) In Compiler option Advanced settings change the memory mmodel field to Far{-mem_model:data=far}. (iii) In Compiler option Preprocessor settings: (a) In Pre-Define Symbol {-d}, write CHIP_6713. (b) In Include Search Path{-i}, write C:\CCStudio_v3.1\C6000\dsk6713\include.



(iv) In linker Option delete top field from –m to before -0. Delete –w as well.



(b)Press OK at the end.
(c)Press Build Clean in project.
(d)Finally press Build to build the program.
To open the code go to File, Open, Look in sine8_buf and open sine8_buf C type file.