

# Digital Signal Processing Lab (EE 521)

## Final Assignment Report

### Problem Statements:

1. Realize a band-pass filter that satisfies the following requirements:

- Passband: 3750Hz – 5250Hz
- Stopband 1: All frequencies below 2750Hz
- Stopband 2: All frequencies above 6250Hz
- Passband maximum ripple of 0.5dB
- Stopband minimum rejection 40dB (both stop bands)

Pick any sampling rate and any design method you want, e.g. equiripple, least squares, etc. Use the Matlab filter design tools, (fdatool) to design filters that satisfy all of the requirements. Use fdatool to plot the impulse response, magnitude response, and phase response of your filter. Export your filter coefficients to C header files from fdatool. All filter coefficients in this assignment should be generated as single precision floats. Then implement your filter in real-time on the TMS320C6416 DSK kit and compute the filter output. Compare the magnitude response of your real-time filter with the fdatool design.

2. The time-domain sequence  $x(n)$  consists of exactly 10 cycles of a real-valued cosine wave (assuming a sampling frequency of 8 kHz, the frequency of the cosine wave is 800 Hz). The DFT of this sequence,  $X(k)$ , is equal to zero for all  $k$ , except at  $k = 10$  and at  $k = 90$ . These two real values correspond to frequency components at  $\pm 800$  Hz. Implement the DFT of this sequence on the TMS320C6416 DSK kit and plot the frequency response in Graphical Display of Code Composer Studio (CCS).

### Brief Description about TMS320C6416 Kit:

The DSK package is a complete DSP system. The DSK board, with an approximate size of 5x8 in., includes the C6416 floating-point digital signal processor and a 32-bit stereo codec TLV320AIC23 (AIC23) for input and output. The onboard codec AIC23 uses a sigma-delta technology that provides ADC and DAC. It connects to a 12-MHz system clock. Variable sampling rates from 8 to 96 kHz can be set readily. The DSK board includes 16 MB (megabytes) of synchronous dynamic random access memory (SDRAM) and 256 kB (kilobytes) of flash memory. Four connectors on the board provide input and output: MIC IN for microphone input, LINE IN for line input, LINE OUT for line output, and HEADPHONE for a headphone output (multiplexed with line output). The status of the four user dip switches on the DSK board can be read from a program and provides the user with a feedback control interface. The DSK operates at 225 MHz.

### ***DSK Support tools:***

To perform the experiment, the following tools are used:

#### 1. TI's DSP starter kit (DSK). The DSK package includes:

- (a) Code Composer Studio (CCS), which provides the necessary software support tools. CCS provides an integrated development environment (IDE), bringing together the C compiler, assembler, linker, debugger, and so on.
- (b) A board that contains the TMS320C6416 (C6416) floating-point digital signal processor as well as a 32-bit stereo codec for input and output (I/O) support.
- (c) A universal synchronous bus (USB) cable that connects the DSK board to a PC.
- (d) A 5 V power supply for the DSK board.

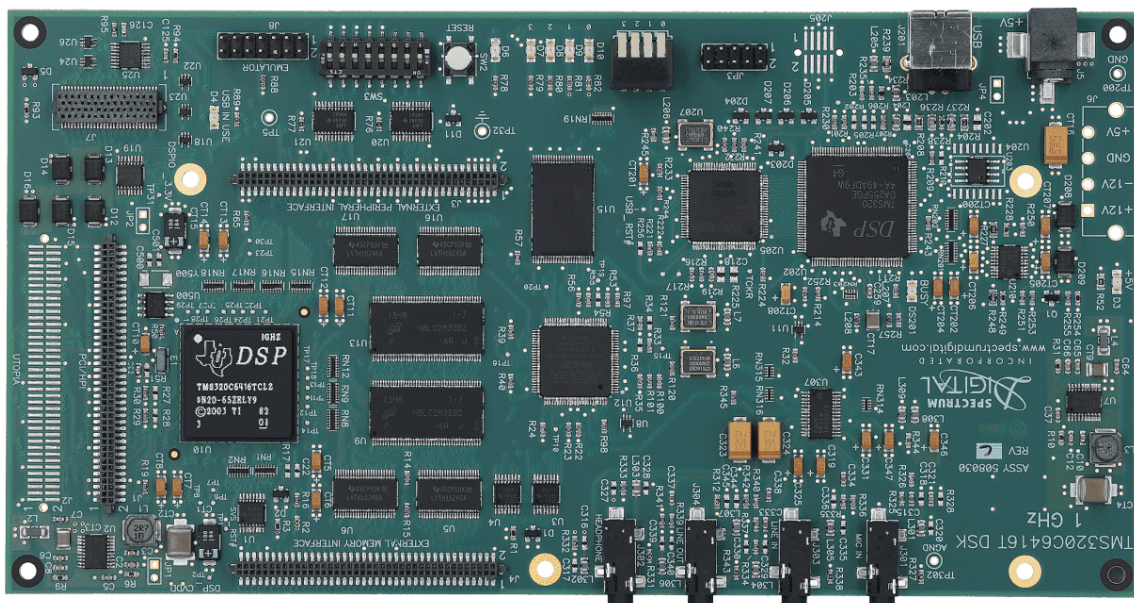
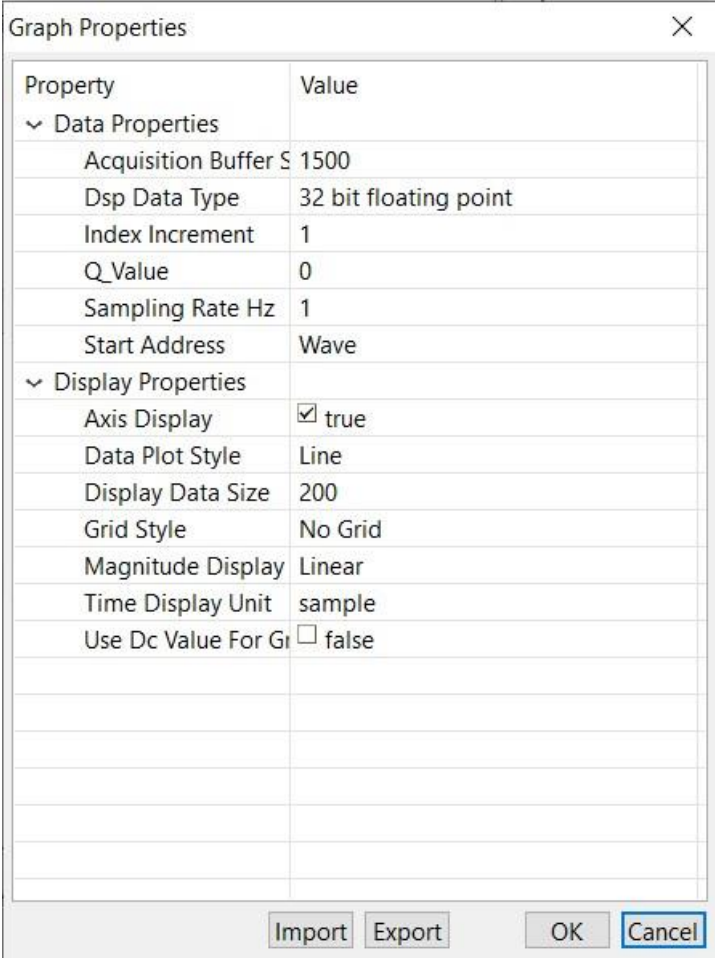


Fig: TMS320C6416 Board

## Procedure:

- Launch CCS and then connect to the DSK.
- Create a new project file :  
Open CC Studio IDE and choose: File → New → CCS Project
- Choose a project name, family device as C6000, Variant as C641X Performance Value DSP for DSK6416. Choose Connection type as Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator.
- Choose Build option → C6000 Compiler → Include Options → Add Button
- Choose C6000 Linker → File Search Path → Add option → Select the Libraries csl6416.lib from C6XCSL Folder and dsk6416bsl.lib from DSK6416 Folder.
- Once the code has been created and project settings are configured, right click on the project folder and click on Build Project. Initially it might take some time to build for the first time.
- After Build, Click on Run option → Debug to debug the code that is built.
- **Graphical Displays in CCS:** Select View → Graph → Time/Frequency and set the Graph Property Dialog properties as shown in Figure below.



The image shows the 'Graph Properties' dialog box in the CCS IDE. It has a title bar with a close button (X). The dialog is divided into two main sections: 'Data Properties' and 'Display Properties'. The 'Data Properties' section includes fields for 'Acquisition Buffer Size' (1500), 'Dsp Data Type' (32 bit floating point), 'Index Increment' (1), 'Q\_Value' (0), 'Sampling Rate Hz' (1), and 'Start Address' (Wave). The 'Display Properties' section includes checkboxes for 'Axis Display' (checked), 'Data Plot Style' (Line), 'Display Data Size' (200), 'Grid Style' (No Grid), 'Magnitude Display' (Linear), 'Time Display Unit' (sample), and 'Use Dc Value For Gr' (unchecked). At the bottom of the dialog are buttons for 'Import', 'Export', 'OK', and 'Cancel'.

Property	Value
▼ Data Properties	
Acquisition Buffer Size	1500
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate Hz	1
Start Address	Wave
▼ Display Properties	
Axis Display	<input checked="" type="checkbox"/> true
Data Plot Style	Line
Display Data Size	200
Grid Style	No Grid
Magnitude Display	Linear
Time Display Unit	sample
Use Dc Value For Gr	<input type="checkbox"/> false

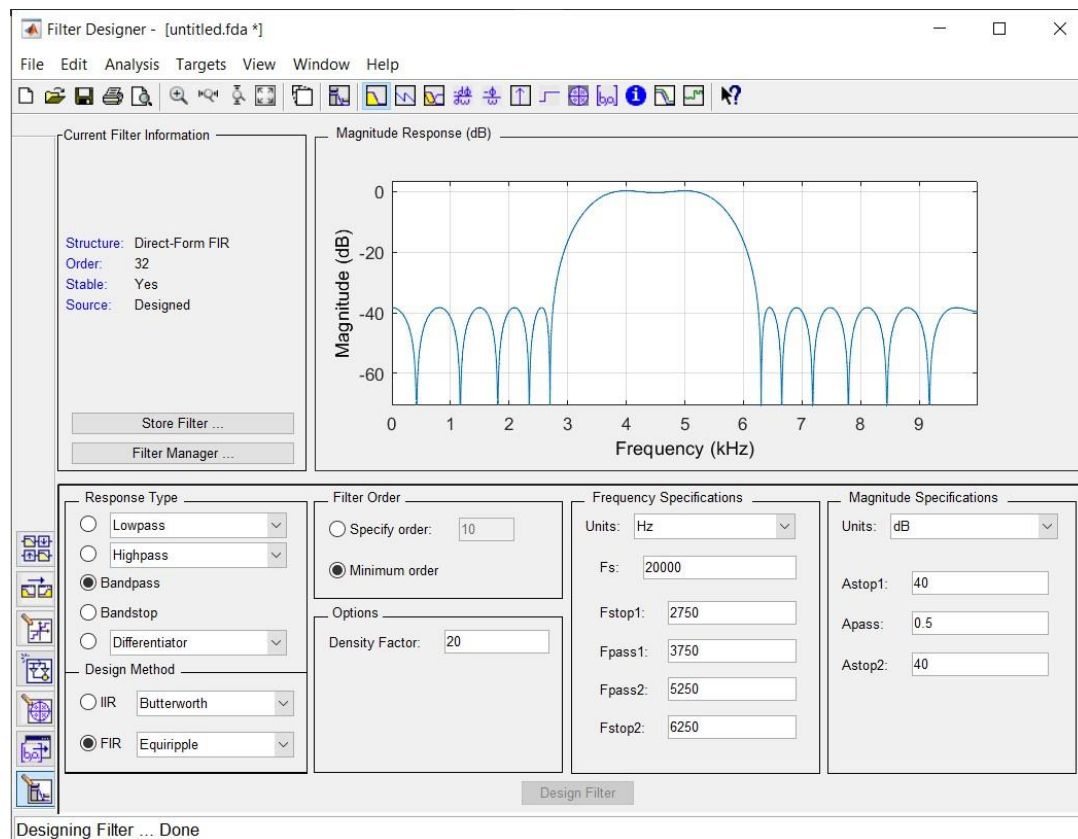
## Code and Results:

### Problem – 1:

Here, the pass band is 3750Hz – 5250Hz.

We have taken the sampling frequency: 20000Hz

First we have designed the FIR band-pass filter using equi-ripple method in MATLAB Filter Designer Tool. The settings are as following:



We have taken mixture of two sinusoids of frequencies **4000Hz** (within pass band) and **2000Hz** (outside pass band) as input signal.

**CODE :**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

#define N 1500 // length of x[n]
#define M 33 // length of h[n]
#define L (N + M - 1) // length of y[n]
#define M_PI 3.14159265358979323846
#define f 4000
#define fs 20000

float Wave[N];

//filter coefficients from MATLAB FDA tool
float FILT[M] = {0.0031, 0.0091, -0.0134, -0.0261, 0.0099, 0.0300, 0.0005, -
0.0136,
                0.0032, -0.0341, -0.0422, 0.0781, 0.1223, -0.0851, -0.2039,
0.0360, 0.2403,
                0.0360, -0.2039, -0.0851, 0.1223, 0.0781, -0.0422, -0.0341,
0.0032, -0.0136,
                0.0005, 0.0300, 0.0099, -0.0261, -0.0134, 0.0091, 0.0031};

float conv_sig[L];
float X[N], Y[L];

float sine(float);

void convolution(float x[], float h[], float y[]);
void dft(float x[], float Xre[], float Xim[], float X[], int P);

int main(void)
{
    float Xre[N], Xim[N], Yre[N], Yim[N];

    int i = 0;
    int j = 0;
    printf("Our signals with %d Hz and %d Hz freq. : \n", f/2, f);
    for (i = 0; i < N; i++) {
        Wave[j] = 80*sine(2 * M_PI * f * ((float)i/(float)fs));
        printf("%f, ", Wave[j]);
        j++;
    }
    printf("\nOur convoluted signal is : \n");
    convolution(Wave, FILT, conv_sig);

    printf("\nDFT of input signal is : \n");
    dft(Wave, Xre, Xim, X, N);
    printf("\nDFT of convoluted signal is : \n");
    dft(conv_sig, Yre, Yim, Y, L);

    return 0;
}
```

```
void convolution(float x[], float h[], float y[])
{
    int i, j, k;

    // initialize the output signal to zero
    for (i = 0; i < L; i++) {
        y[i] = 0;
    }

    // perform the convolution
    for (i = 0; i < L; i++) {
        for (j = 0; j < M; j++) {
            k = i - j;
            if (k >= 0 && k < N) {
                y[i] += x[k] * h[j];
            }
        }
        printf("%f, ", y[i]);
    }
}

float sine(float x) {
    return sin(x) + sin (x/2.0);
}

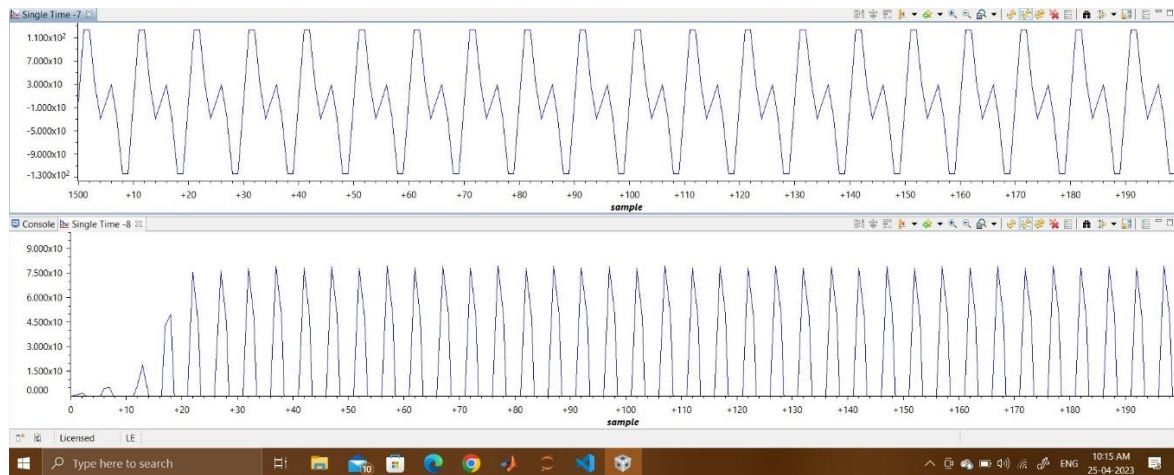
void dft(float x[], float Xre[], float Xim[], float X[], int P)
{
    int k, n;
    float w;

    // compute the DFT of x[n]
    for (k = 0; k < P; k++) {
        Xre[k] = 0;
        Xim[k] = 0;
        for (n = 0; n < P; n++) {
            w = 2 * M_PI * k * n / P;
            Xre[k] += (x[n] * cos(w));
            Xim[k] -= (x[n] * sin(w));
        }
        X[k] = sqrt(Xre[k]*Xre[k] + Xim[k]*Xim[k]);
        printf("%f, ", X[k]);
    }
}
```



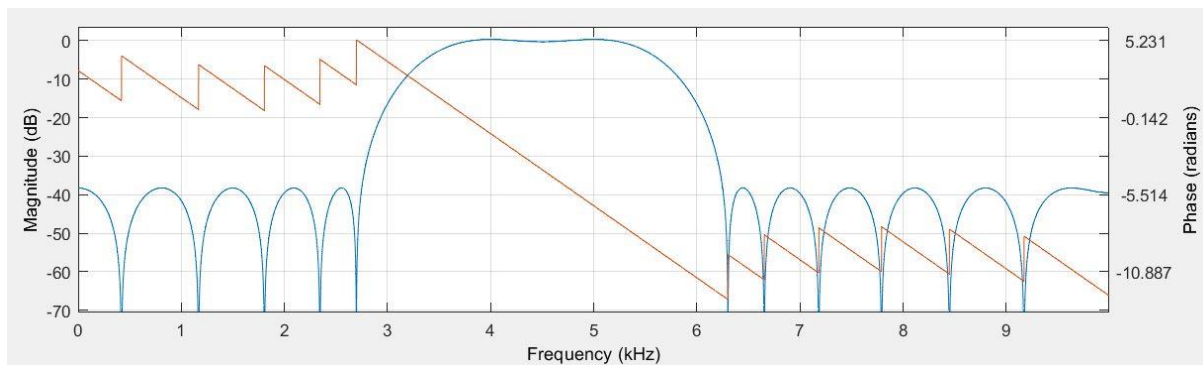
## RESULTS :

The input (top) and output (bottom) waveforms in Graphical Display of Code Composer Studio (CCS) are given below:

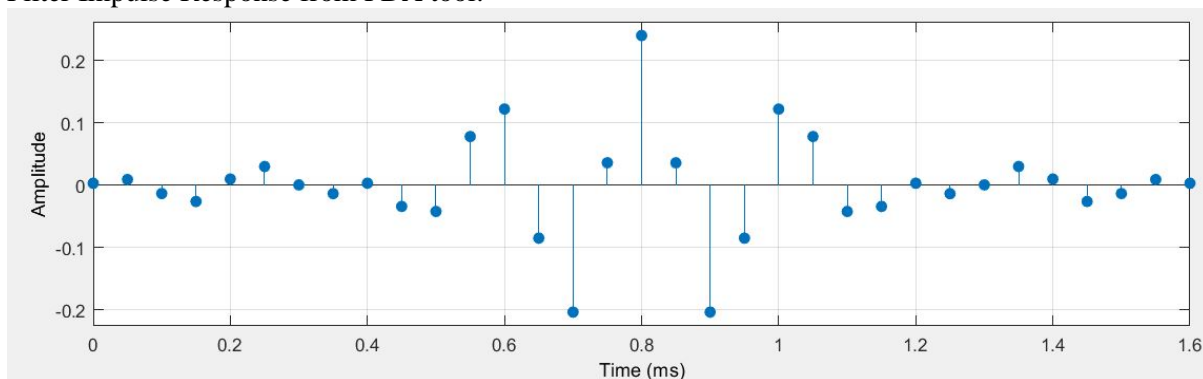


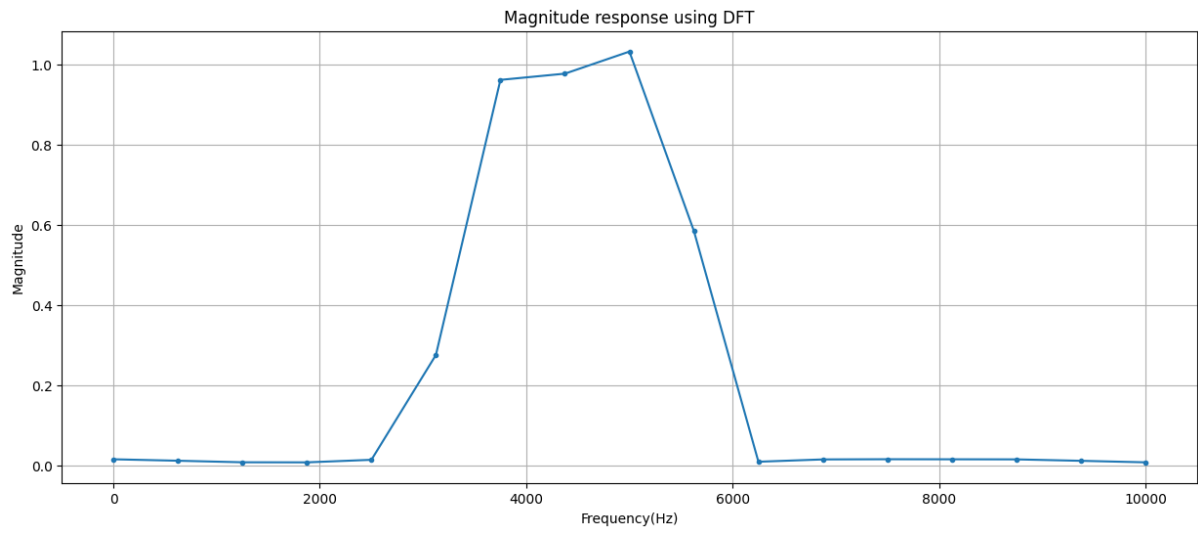
We can see the low frequency (2000Hz) component has got removed in the output waveform. Thus, our band-pass filter is working properly.

Magnitude and Phase response from FDA tool:



Filter Impulse Response from FDA tool:







## Problem 2:

We have a cosine signal of 800 Hz frequency with 8000Hz sampling frequency. We are taking the signal length  $N = 100$ , to fit 10 cycles of the signal.

Here we have to implement DFT of the given signal. We are interested only in the magnitude plot.

The C programming language code we have used is:

### CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>

#define N 100 // length of x[n]
#define M_PI 3.14159265358979323846
#define f 800
#define fs 8000

float Wave[N];

float X[N];

float cosine(float);

void dft(float x[], float Xre[], float Xim[], float X[], int P);

int main(void)
{
    float Xre[N], Xim[N];

    int i = 0;
    int j = 0;
    printf("Our signals with %d Hz freq. : \n", f);
    for (i = 0; i < N; i++) {
        printf("a ");
        Wave[j] = 80*cosine(2 * M_PI * f * ((float)i/(float)fs));
        printf("%f, ", Wave[j]);
        j++;
    }

    printf("\nDFT of input signal is : \n");
    dft(Wave, Xre, Xim, X, N);

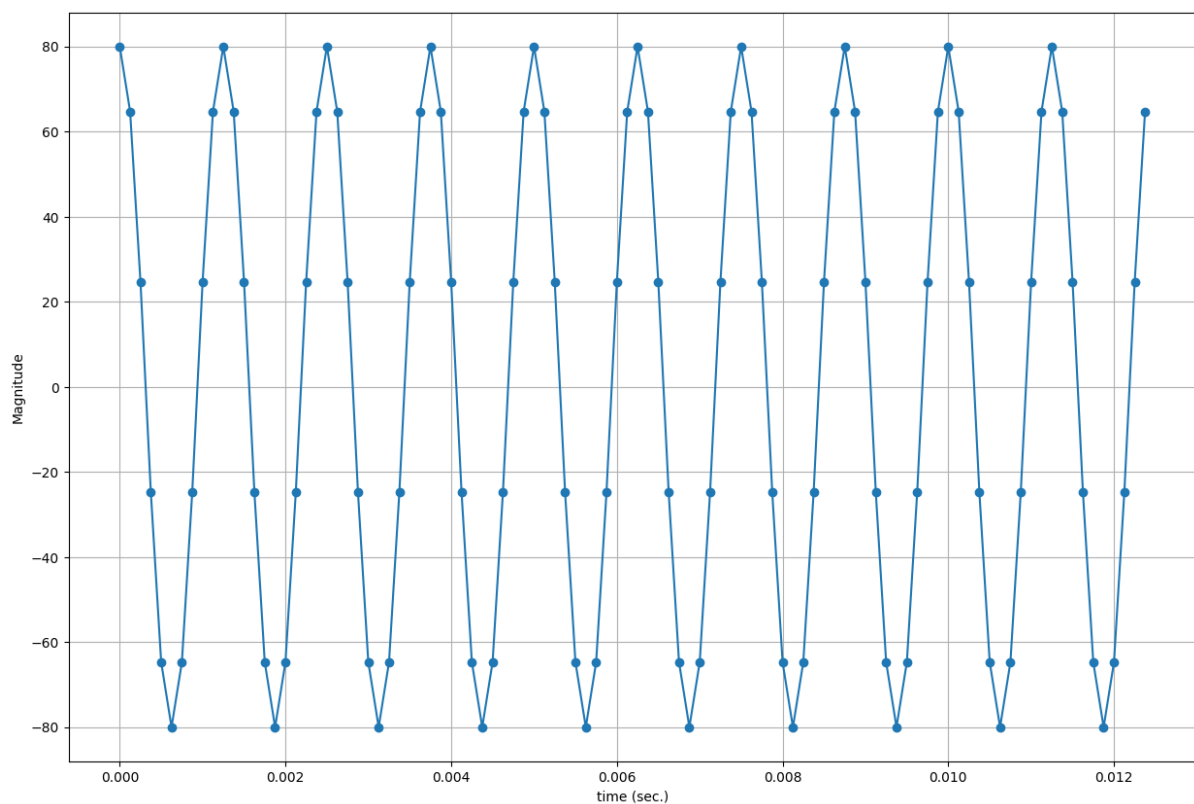
    return 0;
}

float cosine(float x){
    return cos(x);
}
```

```
void dft(float x[], float Xre[], float Xim[], float X[], int P)
{
    int k, n;
    float w;

    // compute the DFT of x[n]
    for (k = 0; k < P; k++) {
        Xre[k] = 0;
        Xim[k] = 0;
        for (n = 0; n < P; n++) {
            w = 2 * M_PI * k * n / P;
            Xre[k] += (x[n] * cos(w));
            Xim[k] -= (x[n] * sin(w));
        }
        X[k] = sqrt(Xre[k]*Xre[k] + Xim[k]*Xim[k]);
        printf("%f, ", X[k]);
    }
}
```

Input Signal:



## RESULTS :

Output DFT Sequence:

The 100 points DFT magnitude sequence generated by the aforementioned code is:

```
output = [0.000107, 0.000398, 0.000184, 0.000539, 0.000151, 0.000717,
0.000778, 0.000737, 0.000187, 0.000317, 4000.000000, 0.000470,
0.000130, 0.000405, 0.001537, 0.001291, 0.000962, 0.001232, 0.002021,
0.000563, 0.000489, 0.001299, 0.000952, 0.000759, 0.000436, 0.000659,
0.000962, 0.002761, 0.002580, 0.001940, 0.002960, 0.004054, 0.001593,
0.001891, 0.003502, 0.000783, 0.001591, 0.002314, 0.000620, 0.000345,
0.001211, 0.003364, 0.001045, 0.002173, 0.001164, 0.001237, 0.002954,
0.000329, 0.000436, 0.000439, 0.000489, 0.003497, 0.003203, 0.002447,
0.002181, 0.002542, 0.000870, 0.005741, 0.002957, 0.001702, 0.002702,
0.003140, 0.000882, 0.001728, 0.002126, 0.003507, 0.003503, 0.008529,
0.004182, 0.002755, 0.010787, 0.002331, 0.002969, 0.000825, 0.004818,
0.002003, 0.004437, 0.003256, 0.002073, 0.006535, 0.000402, 0.001692,
0.001857, 0.001619, 0.003105, 0.003287, 0.001266, 0.002534, 0.001232,
0.008665, 4000.000977, 0.007397, 0.002180, 0.001552, 0.004005,
0.004705, 0.008446, 0.005655, 0.004499, 0.002113]
```

Here, all the entries are almost zero except at indices 10 and 90 which represent the signal frequency 800 Hz and its reflection at  $(8000 - 800)$  Hz. The magnitude spectrum of the sequence is as the following:

