Signal Processing Application of FFT with FIR filter



Capital University of Science and Technology

Department of Computer Science

Project by:

Ruquiya Nasir Bcs213065

Laiba Shahzadi Bcs213053

Submitted to:

Sir Masroor Ahmed

A project report presented to

Capital University of Science and Technology

# 

# ABSTRACT

In [mathematics](https://en.wikipedia.org/wiki/Mathematics), a **Fourier transform** (**FT**) is a [mathematical transform](https://en.wikipedia.org/wiki/Integral_transform) that decomposes [functions](https://en.wikipedia.org/wiki/Function_(mathematics)) depending on space or time into functions depending on spatial or temporal frequency, such as the expression of a musical [chord](https://en.wikipedia.org/wiki/Chord_(music)) in terms of the volumes and frequencies of its constituent notes. The term Fourier transform refers to both the [frequency domain](https://en.wikipedia.org/wiki/Frequency_domain) representation and the [mathematical operation](https://en.wikipedia.org/wiki/Operation_(mathematics)) that associates the frequency domain representation to a function of space or time. FFT-based FIR Filter is a unit to perform the finite impulse response filter based on the Fast Fourier Transform (FFT). This function combines two important techniques to speed up filtering of long signals, the overlap-add method, and FFT convolution. The overlap-add method is used to break long signals into smaller segments for easier processing or preventing memory problems. FFT convolution uses the overlap-add method together with the Fast Fourier Transform, allowing signals to be convolved by multiplying their frequency spectra. For filter kernels longer than about 64 points, FFT convolution is faster than standard convolution, while producing exactly the same result.

Thus for the application Four sinusoidal signals with different frequencies were used. A noisy signal was generated.  The four signals along with noise were added to generate a single signal with different frequency components. The FFT of this signal was computed using the fft() function and the frequency spectrum was plotted to visualize the frequency components.

The only motto of this project is to make one understand how Fast Fourier Transform can be used in the simplest of ways to remove unwanted frequencies in a given signal

Table of Contents

[ABSTRACT 2](#_Toc135923130)

[1.1 Description 5](#_Toc135923131)

[1.1.1 Fast Fourier Transformation FFT 5](#_Toc135923132)

[1.1.2 FFT-based FIR Filter 6](#_Toc135923133)

[1.2 Objective: 6](#_Toc135923134)

[Chapter 2: Literature Review 8](#_Toc135923135)

[2.1 FAST FIR FILTERING USING THE FFT: 8](#_Toc135923136)

[2.2 Existing Method 9](#_Toc135923137)

[IIR filters 9](#_Toc135923138)

[FIR filters 9](#_Toc135923139)

[2.3 Fast Fourier Transformation FFT 11](#_Toc135923140)

[2.3.1 FFT Mathematically: 14](#_Toc135923141)

[2.4 FFT-based FIR Filter 18](#_Toc135923142)

[2.4.1 Main Features 19](#_Toc135923143)

[2.5 Comparison table 20](#_Toc135923144)

[2.6 Decision Formats: 20](#_Toc135923145)

[Chapter 3: Methodology and Implementation 23](#_Toc135923146)

[3.1 Introduction 23](#_Toc135923147)

[Application of FFT with FIR filter 23](#_Toc135923148)

[Fast Fourier Transform (FFT): 23](#_Toc135923149)

[3.2 Flowchart: 25](#_Toc135923150)

[3.3 Matlab Code for Implementing application: 25](#_Toc135923151)

[3.3.1 Main file: 25](#_Toc135923152)

[3.3.2 BPF File: 29](#_Toc135923153)

[3.3.3 LPF File: 29](#_Toc135923154)

[3.4 Results and Steps Followed: 30](#_Toc135923155)

[3.5 Error: 36](#_Toc135923156)

[CONCLUSION: 37](#_Toc135923157)

TABLE OF FIGURES

[Figure 1 View of signal in time and frequency domain 9](file:///C:\Users\ruqui\Documents\numerical%20project.docx#_Toc135862718)

[Figure 2 Representation of the FFT of a signal with small and large block length 11](file:///C:\Users\ruqui\Documents\numerical%20project.docx#_Toc135862719)

[Figure 3 Un-Windowed time signal with smeared spectrum 12](#_Toc135862720)

[Figure 4 Windowed time signal with spectrum 13](#_Toc135862721)

[Figure 5 FIR and IIR filters comparison 19](#_Toc135862722)

[Figure 6 FT and DFT comparison 19](#_Toc135862723)

[Figure 7 DFT VS FFT 22](#_Toc135862724)

[Figure 8 All Signals 29](#_Toc135862725)

[Figure 9 Overall Signals 30](#_Toc135862726)

[Figure 10 Frequency Response original signal 30](#_Toc135862727)

[Figure 11 BPF\_Toolbox 31](#_Toc135862728)

[Figure 12 LPF\_toolbox 31](#_Toc135862729)

[Figure 13 MagResponse of BPF 32](#_Toc135862730)

[Figure 14 Mag response of LPG 32](#_Toc135862731)

[Figure 15 Freq Response After Passing BPF 32](#_Toc135862732)

[Figure 16 Freq Response After Passing LPF 33](#_Toc135862733)

[Figure 17 Comparison of all Signals 34](#_Toc135862734)

[Figure 18 Magnitude response 34](#_Toc135862735)

[Figure 19 35](#_Toc135862736)

CHAPTER 1: INTRODUCTION

# Description

# Fast Fourier Transformation FFT

A fast Fourier transform (FFT) is a highly optimized implementation of the discrete Fourier transform (DFT), which convert discrete signals from the time domain to the frequency domain. FFT computations provide information about the frequency content, phase, and other properties of the signal.

The "Fast Fourier Transform" (FFT) is an important measurement method in the science of audio and acoustics measurement. It converts a signal into individual spectral components and thereby provides frequency information about the signal.

[Y](https://www.mathworks.com/help/matlab/ref/fft.html#f83-998360-Y) = fft([X](https://www.mathworks.com/help/matlab/ref/fft.html#f83-998360-X)) computes the [discrete Fourier transform](https://www.mathworks.com/help/matlab/ref/fft.html#buuutyt-6) (DFT) of X using a fast Fourier transform (FFT) algorithm.

* If X is a vector, then fft(X) returns the Fourier transform of the vector.
* If X is a matrix, then fft(X) treats the columns of X as vectors and returns the Fourier transform of each column.
* If X is a multidimensional array, then fft(X) treats the values along the first array dimension whose size does not equal 1 as vectors and returns the Fourier transform of each vector.

[Y](https://www.mathworks.com/help/matlab/ref/fft.html#f83-998360-Y) = fft([X](https://www.mathworks.com/help/matlab/ref/fft.html" \l "f83-998360-X),[n](https://www.mathworks.com/help/matlab/ref/fft.html" \l "f83-998360-n)) returns the n-point DFT. If no value is specified, Y is the same size as X.

* If X is a vector and the length of X is less than n, then X is padded with trailing zeros to length n.
* If X is a vector and the length of X is greater than n, then X is truncated to length n.
* If X is a matrix, then each column is treated as in the vector case.
* If X is a multidimensional array, then the first array dimension whose size does not equal 1 is treated as in the vector case.

[Y](https://www.mathworks.com/help/matlab/ref/fft.html#f83-998360-Y) = fft([X](https://www.mathworks.com/help/matlab/ref/fft.html" \l "f83-998360-X),[n](https://www.mathworks.com/help/matlab/ref/fft.html" \l "f83-998360-n),[dim](https://www.mathworks.com/help/matlab/ref/fft.html#f83-998360-dim)) returns the Fourier transform along the dimension dim. For example, if X is a matrix, then fft(X,n,2) returns the n-point Fourier transform of each row.

## FFT-based FIR Filter

The Frequency-Domain FIR Filter block implements frequency-domain, fast Fourier transform (FFT)-based filtering to filter a streaming input signal. In the time domain, the filtering operation involves a convolution between the input and the impulse response of the finite impulse response (FIR) filter. In the frequency domain, the filtering operation involves the multiplication of the Fourier transform of the input and the Fourier transform of the impulse response. The frequency-domain filtering becomes more efficient than time-domain filtering as the impulse response grows longer. You can specify the filter coefficients directly in the frequency domain by setting **Numerator domain** to Frequency.

This block uses the overlap-save and overlap-add methods to perform the frequency-domain filtering. For filters with a long impulse response length, the latency inherent to these two methods can be significant. To mitigate this latency, the Frequency-Domain FIR Filter block partitions the impulse response into shorter blocks and implements the overlap-save and overlap-add methods on these shorter blocks. To partition the impulse response, select the **Partition numerator to reduce latency** check box.

# Objective:

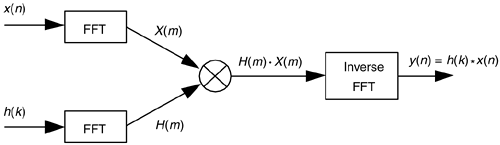
The only motto of this Research is to create the concept of Fast Fourier Transform and also to make one understand how Fast Fourier Transform can be used in the simplest of ways to remove unwanted frequencies in a given signal

* To clearly understand the concept and difference between of DFT and FFT.
* To clearly identify between two filters to determine which to use when and why?
* FIR and IIR uses and difference.
* To implement an application after identifying the correct requirements to easily and simply remove unwanted frequencies from a given signal.

# Chapter 2: Literature Review

# FAST FIR FILTERING USING THE FFT:

Digital signal processing practitioners realized that convolution could sometimes be performed more efficiently using FFT algorithms than it could be using the direct convolution method. This FFT-based convolution scheme, called fast convolution..Processing diagram of fast convolution.



The standard convolution equation for an M-tap nonrecursive FIR filter, given in Eq. (5-6) and repeated here, is

https://flylib.com/books/2/729/1/html/2/images/0131089897/graphics/13equ77.gif

where h(k) is the impulse response sequence (coefficients) of the FIR filter and the \* symbol indicates convolution. It has been shown that when the final y(n) output sequence has a length greater than 30, the process requires fewer multiplications than implementing the convolution expression directly. Consequently, this fast convolution technique is a very powerful signal processing tool, particularly when used for digital filtering. Very efficient FIR filters can be designed using this technique because, if their impulse response h(k) is constant, then we don't have to bother recalculating H(m) each time a new x(n) sequence is filtered. In this case the H(m) sequence can be pre-calculated and stored in memory.

# Existing Method

Digital filters are divided into the following two categories:

* Infinite impulse response (IIR)
* Finite impulse response (FIR)

## ****IIR filters****

IIR (infinite impulse response) filters are generally chosen for applications where linear phase is not too important and memory is limited. They have been widely deployed in audio equalisation, biomedical sensor signal processing, IoT/IIoT smart sensors and high-speed telecommunication/RF applications.

## ****FIR filters****

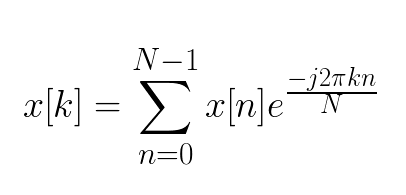
FIR (finite impulse response) filters are generally chosen for applications where linear phase is important and a decent amount of memory and computational performance are available. They have a widely deployed in audio and biomedical signal enhancement applications. Their all-zero structure (discussed below) ensures that they never become unstable for any type of input signal, which gives them a distinct advantage over the IIR.

Here we could have chosen any of the transformation way:

1. Discrete Fourier Transform (DTF)
2. Fast Fourier Transform (FFT)

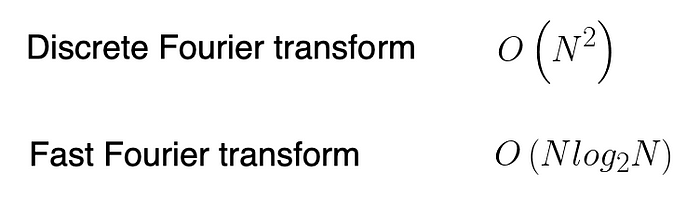
**DTF and FFT**

The Discrete Fourier Transform (DTF) can be written as follows.

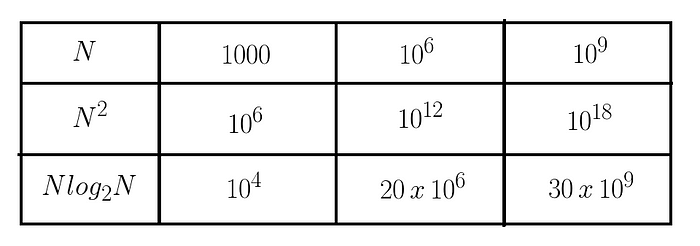


To determine the DTF of a discrete signal x[n] (where N is the size of its domain), we multiply each of its value by e raised to some function of n. We then sum the results obtained for a given n. If we used a computer to calculate the Discrete Fourier Transform of a signal, it would need to perform N (multiplications) x N (additions) = O(N²) operations.

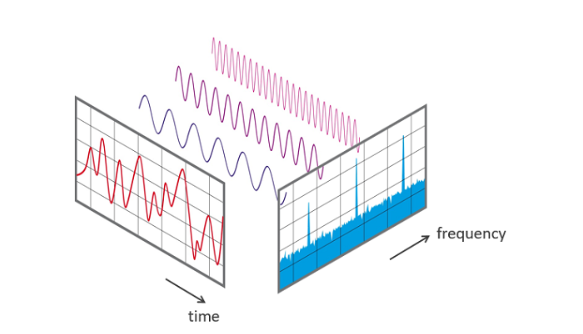
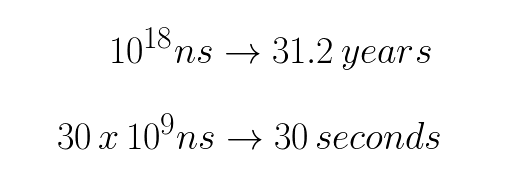
As the name implies, the Fast Fourier Transform (FFT) is an algorithm that determines Discrete Fourier Transform of an input significantly faster than computing it directly. In computer science lingo, the FFT reduces the number of computations needed for a problem of size N from O(N^2) to O(NlogN).



On the surface, this might not seem like a big deal. However, when N is large enough it can make a world of difference. Have a look at the following table.



Say it took 1 nanosecond to perform one operation. It would take the Fast Fourier Transform algorithm approximately 30 seconds to compute the Discrete Fourier Transform for a problem of size N = 10⁹. In contrast, the regular algorithm would need several decades.

****

# Fast Fourier Transformation FFT

It converts a signal into individual spectral components and thereby provides frequency information about the signal. FFTs are used for fault analysis, quality control, and condition monitoring of machines or systems. This article explains how an FFT works, the relevant parameters and their effects

Figure 1 View of signal in time and frequency domain

On the measurement result.  
  
Strictly speaking, the FFT is an optimized algorithm for the implementation of the "Discrete Fourier Transformation" (DFT). A signal is sampled over a period of time and divided into its frequency components. These components are single sinusoidal oscillations at distinct frequencies each with their own amplitude and phase.

This transformation is illustrated in the following diagram. Over the time period measured, the signal contains 3 distinct dominant frequencies.

**Step by step:**

In the first step, a section of the signal is scanned and stored in the memory for further processing. Two parameters are relevant:

* The sampling rate or sampling frequency fs of the measuring system (e.g. 48 kHz). This is the average number of samples obtained in one second (samples per second).
* The selected number of samples; the blocklength BL. This is always an integer power to the base 2 in the FFT (e.g., 2^10 = 1024 samples)

From the two basic parameters fs and BL, further parameters of the measurement can be determined.

**Bandwidth fn** (= Nyquist frequency). This value indicates the theoretical maximum frequency that can be determined by the FFT.

*fn = fs / 2*

For example at a sampling rate of 48 kHz, frequency components up to 24 kHz can be theoretically determined. In the case of an analog system, the practically achievable value is usually somewhat below this, due to analog filters - e.g. at 20 kHz.

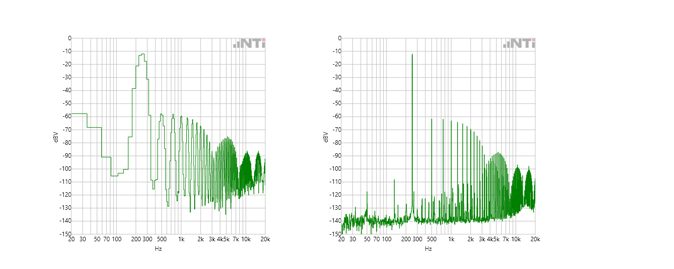
**Measurement duration D**. The measurement duration is given by the sampling rate fs and the blocklength BL.  
  
*D = BL / fs.*  
  
At fs = 48 kHz and BL = 1024, this yields 1024/48000 Hz = 21.33 ms

**Frequency resolution df**. The frequency resolution indicates the frequency spacing between two measurement-results.  
  
*df = fs / BL*

At fs = 48 kHz and BL = 1024, this gives a df of 48000 Hz / 1024 = 46.88 Hz.  
  
In practice, the sampling frequency fs is usually a variable given by the system. However, by selecting the blocklength BL, the measurement duration and frequency resolution can be defined. The following applies:

* A small blocklength results in fast measurement repetitions with a coarse frequency resolution.
* A large blocklength results in slower measuring repetitions with fine frequency resolution.

Figure 2 Representation of the FFT of a signal with small and large block length



**To Infinity…**:

In the Fourier transformation, the assumption is that the sampled signal segment is repeated periodically for an infinite period of time. This brings two conclusions:

1. The FFT is only suitable for periodic signals.
2. The sampled signal segment must contain a whole number of periods.

It can be seen that condition 2. would apply only to very few signals. The sampling of a signal whose frequencies are not an integer multiple of df would begin and end within a block of 2^n samples with different values. This results in a jump in the time signal, and a "smeared" FFT spectrum. (aka Leakage)

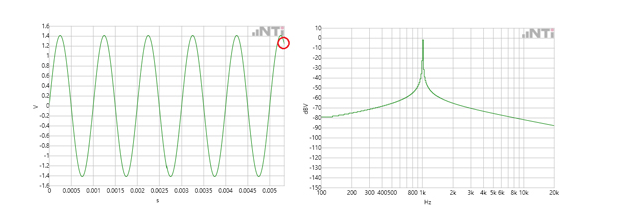
****

Figure 3 Un-Windowed time signal with smeared spectrum

**Windowing:**

In order to prevent this smearing, in practice "windowing" is applied to the signal sample. Using a weighting function, the signal sample is more or less gently turned on and off. The result is that the sampled and subsequent "windowed" signal begins and ends at amplitude zero. The sample can now be repeated periodically without a hard transition.

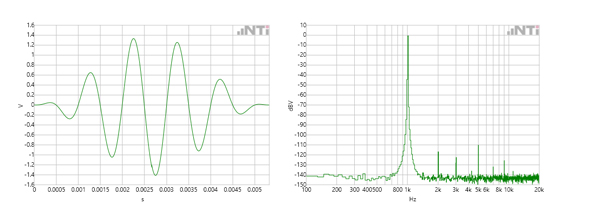
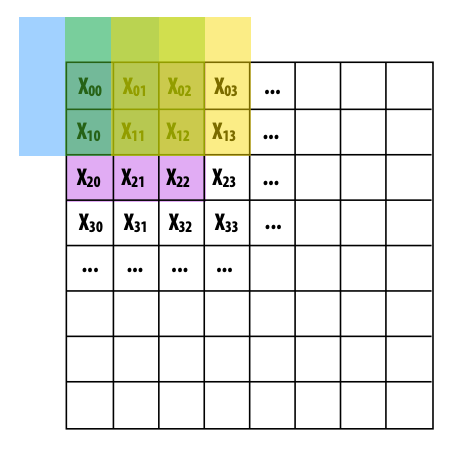
-

Figure 4 Windowed time signal with spectrum

## FFT Mathematically:

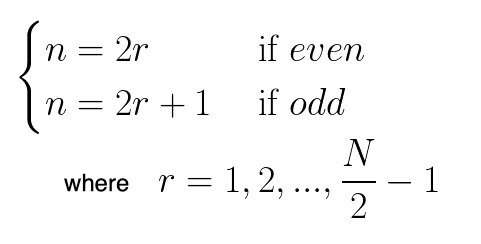
Convolutional layer overlays a kernel on a section of an image and performs bit-wise multiplication with all of the values at that location. The kernel is then shifted to another section of the image and the process is repeated until it has traversed the entire image.

The Fourier Transform can speed up convolutions by taking advantage of the following property.

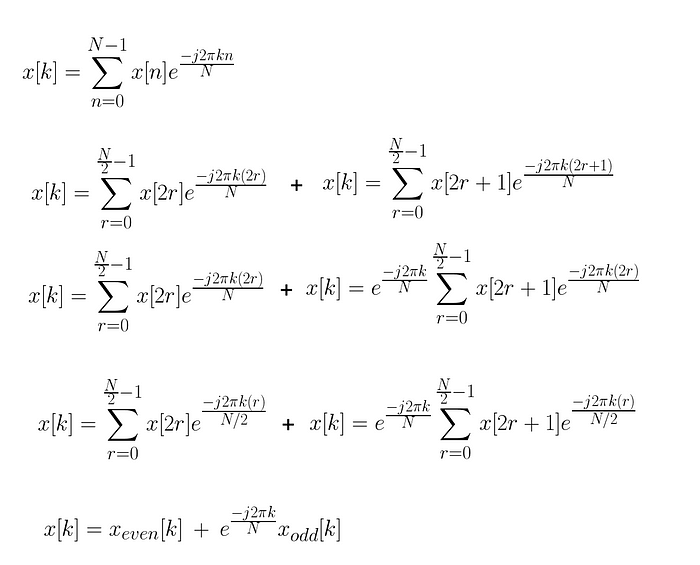


The above equation states that the convolution of two signals is equivalent to the multiplication of their Fourier transforms. Therefore, by transforming the input into frequency space, a convolution becomes a single element-wise multiplication. In other words, the input to a convolutional layer and kernel can be converted into frequencies using the Fourier Transform, **multiplied once** and then converted back using the inverse Fourier Transform. There is an overhead associated with transforming the inputs into the Fourier domain and the inverse Fourier Transform to get responses back to the spatial domain. However, this is offset by the speed up obtained from performing a single multiplication instead of having to multiply the kernel with different sections of the image.

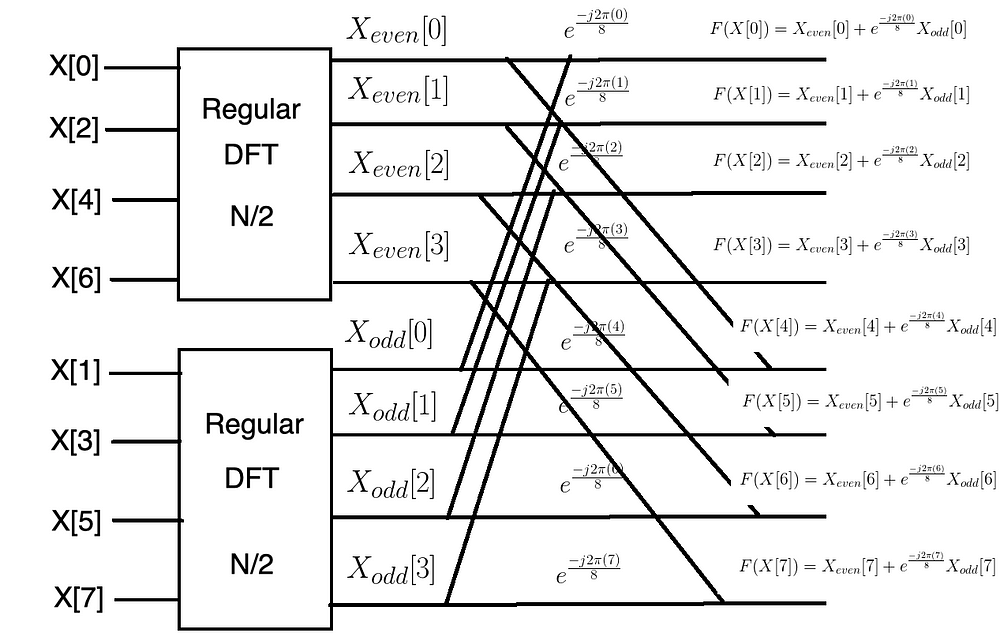
Suppose, we separated the Fourier Transform into even and odd indexed sub-sequences.



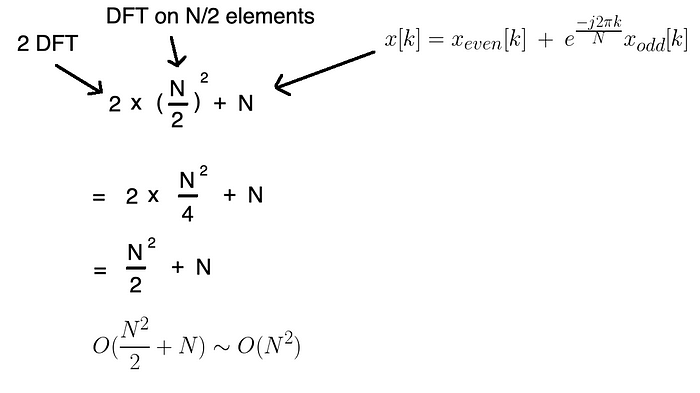
After performing a bit of algebra, we end up with the summation of two terms. The advantage of this approach lies in the fact that the even and odd indexed sub-sequences can be computed concurrently.



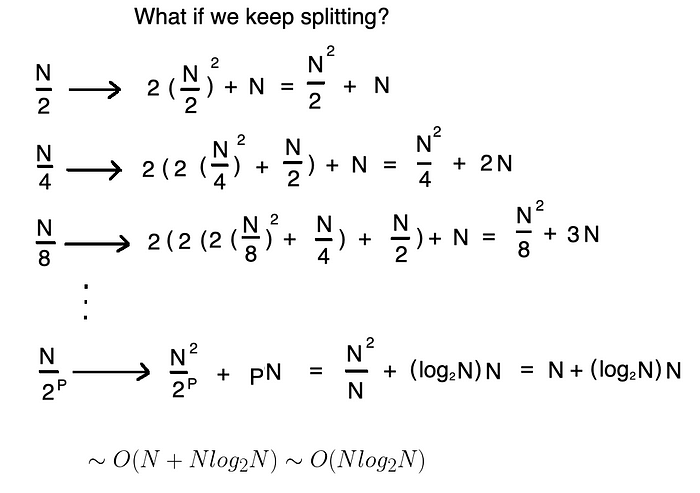
Suppose, N = 8 , to visualize the flow of data with time, we can make use of a butterfly diagram. We compute the Discrete Fourier Transform for the even and odd terms simultaneously. Then, we calculate x[k] using the formula from above.



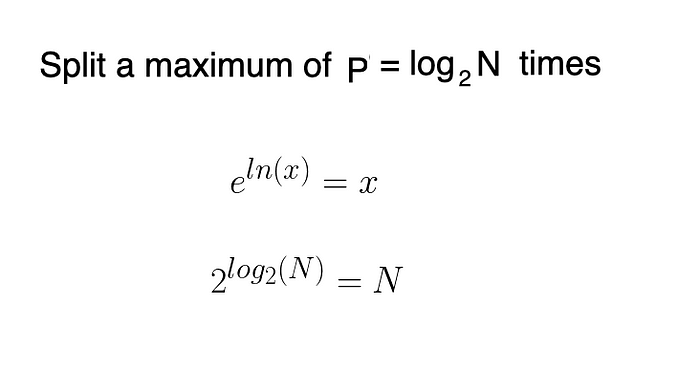
We can express the gains in terms of Big O Notation as follows. The first term comes from the fact that we compute the Discrete Fourier Transform twice. We multiply the latter by the time taken to compute the Discrete Fourier Transform on half the original input. In the final step, it takes N steps to add up the Fourier Transform for a particular k. We account for this by adding N to the final product.



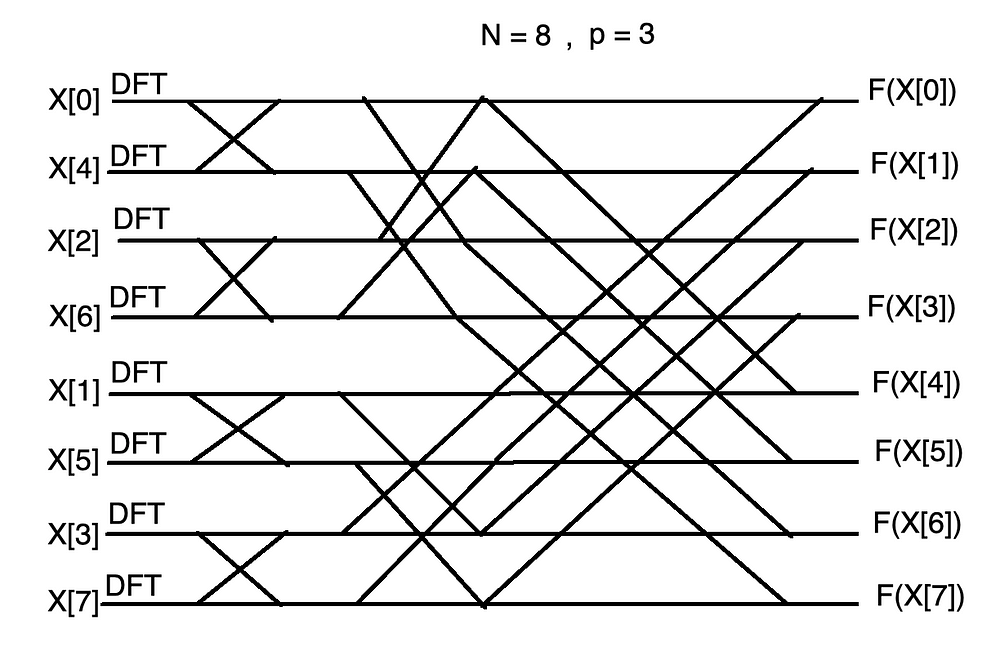
Notice how we were able to cut the time taken to compute the Fourier Transform by a factor of 2. We can further improve the algorithm by applying the divide-and-conquer approach, halving the computational cost each time. In other words, we can continue to split the problem size until we’re left with groups of two and then directly compute the Discrete Fourier Transforms for each of those pairs.



So long as N is a power of 2, the maximum number of times you can split into two equal halves is given by p = log(N).



Here’s what it would look like if we were to use the Fast Fourier Transform algorithm with a problem size of N = 8. Notice how we have p = log(8) = 3 stages.



# FFT-based FIR Filter

FFT-based FIR Filter is a unit to perform the finite impulse response filter based on the Fast Fourier Transform (FFT). It performs the convolution of the unlimited signal sequence with the synthesized impulse response of the length of Ni=N/2 samples, where N = 64, 128, 256, 512, 1024. The data and coefficient widths are tunable in the range 8 to 18.

This function combines two important techniques to speed up filtering of long signals, the overlap-add method, and FFT convolution. The overlap-add method is used to break long signals into smaller segments for easier processing or preventing memory problems. FFT convolution uses the overlap-add method together with the Fast Fourier Transform, allowing signals to be convolved by multiplying their frequency spectra. For filter kernels longer than about 64 points, FFT convolution is faster than standard convolution, while producing exactly the same result.

The overlap-add technique works as follows. When an N length signal is convolved with a filter kernel of length M, the output signal is N + M - 1 samples long, i.e., the signal is expanded 'to the right'. The signal is then broken into k smaller segments, and the convolution of each segment with the f kernel will have a result of length N / k + M -1. The individual segments are then added together. The rightmost M - 1 samples overlap with the leftmost M - 1 samples of the next segment. The overlap-add method produces exactly the same output signal as direct convolution.

## Main Features

* The filtering algorithm is the sectioned convolution with accumulating based on N-point radix-2 FFT, where N = 64, 128, 256, 512, 1024
* One complex signal channel or two parallel real signal channels.
* Filter types are LPF; LPF and HPF; LPF and HPF, and differentiator; LPF and HPF,and double differentiator.
* Input data, output data, and coefficient widths are generics
* Bandpass frequencies of the LPF and HPF filters, filter type are dynamically tunable parameters. The frequencies for both real channels are tuned independently
* Stop band ripple for 16-bit dates is higher than 60 db. The transitional frequency band is less than 6 bins (1 bin = Fs/N, where Fs is the sampling frequency
* Dynamic range for 16-bit dates is higher than 70 db
* Structure optimized for Xilinx Virtex2, Virtex4, Spartan3 FPGA devices, and can be implemented in Altera, Actel, Lattice devices as well.
* The maximum clock frequency for Virtex4 devices is equal to Fclk = 190 MHz, and for Spartan3E devices is equal to Fclk = 80 MHz.
* The maximum sampling frequency Fs by N=1024 is less than Fclk/29.
* The latent delay of the filter by N=1024 is equal to 1790 cycles of Fs.

# Comparison table

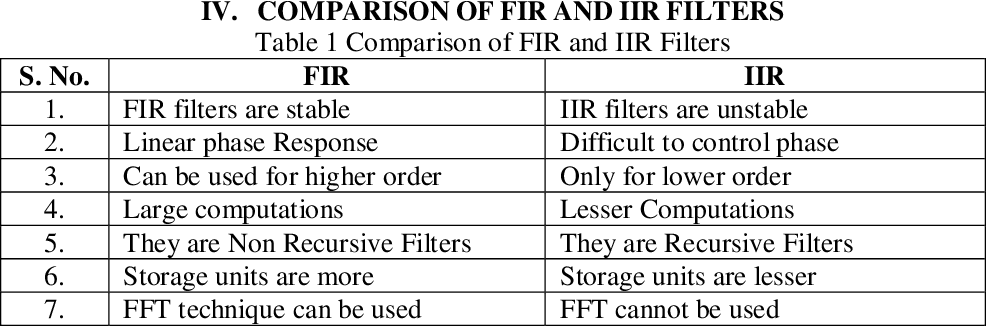
****

Figure 5 FIR and IIR filters comparison

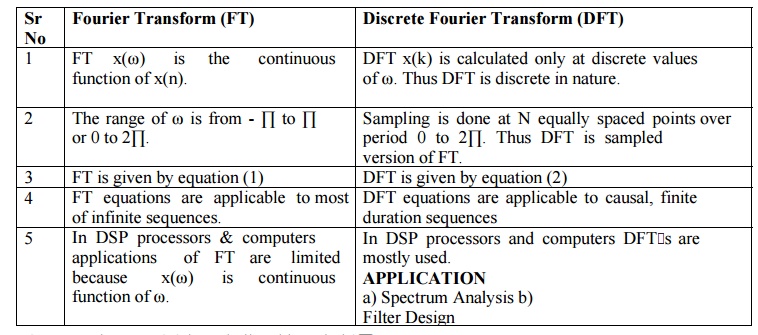


Figure 6 FT and DFT comparison

# Decision Formats:

We used Fast Fourier Transform (FFT) instead of discrete Fourier transform (DFT). as The FFT is much faster than the discrete Fourier transform (DFT). The FFT can be used to compute the DFT of a sequence that is not a power of two, while the DFT can only be used to compute the DFT of a sequence that is a power of two.

FFT is based on divide and conquer algorithm where you divide the signal into two smaller signals, compute the DFT of the two smaller signals and join them to get the DFT of the larger signal. The order of complexity of DFT is O(n^2) while that of FFT is O(n. logn) hence, FFT is faster than DFT.

Also we used FIR instead of IIR as Fir is more stable and can be used with FFT while IIR is unstable and cannot be used with FFT.As FIR requires more memory there storage space is also great.

| **Basis for Comparison** | **FIR Filter** | **IIR Filter** |
| --- | --- | --- |
| Stands for | Finite Impluse Response | Infinite Impulse Response |
| Nature | Non-recursive | Recursive |
| Computational Efficiency | Less | Comparatively more |
| Usage | Difficult | Quite easy |
| Feedback | Absent | Present |
| Stability | More | Less |
| Requirement to generate current output | Present and past samples of input. | Present and past samples of input along with past output. |
| Delay offered | High | Comparatively lower |
| Transfer function | Only zeros are present. | Both poles and zeros are present. |
| Memory requirement | More | Less |
| Sensitivity | Less | Comparatively more |
| Resolution offered at low frequencies | Less | More |
| Controllability | Easy | Quite Difficult |

.

An FIR filter is a filter with no feedback in its equation. This can be an advantage because it makes an FIR filter inherently stable. Another advantage of FIR filters is the fact that they can produce linear phases.

The crucial difference between FIR and IIR filters is that the FIR filter provides an impulse response of a finite period. As against IIR is a type of filter that generates impulse response of infinite duration for a dynamic system.

# Chapter 3: Methodology and Implementation

# Introduction

# Application of FFT with FIR filter

The only motto of this project is to make one understand how Fast Fourier Transform can be used in the simplest of ways to remove unwanted frequencies in a given signal.

## Fast Fourier Transform (FFT):

* A fast Fourier transform (FFT) is an algorithm that calculates the discrete Fourier transform (DFT) of some sequence and transforms the structure of the cycle of a waveform into sine components.
* A fast Fourier transform can be used in various types of signal processing. It may be useful in reading things like sound waves, or for any image-processing technologies.
* A fast Fourier transform can be used to solve various types of equations, or show various types of frequency activity in useful ways.
* For example, fast Fourier transform might be helpful in sound engineering, seismology or in voltage measurements.

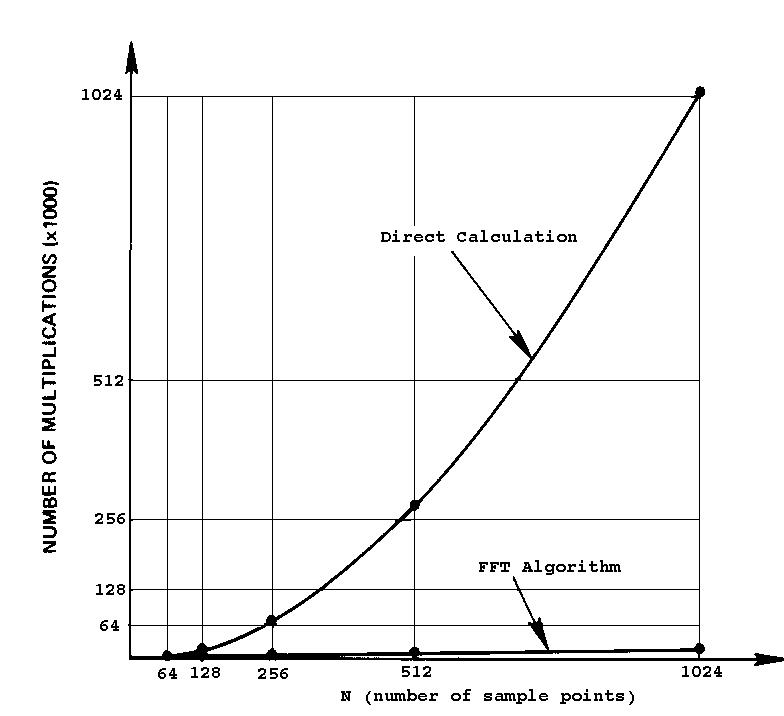
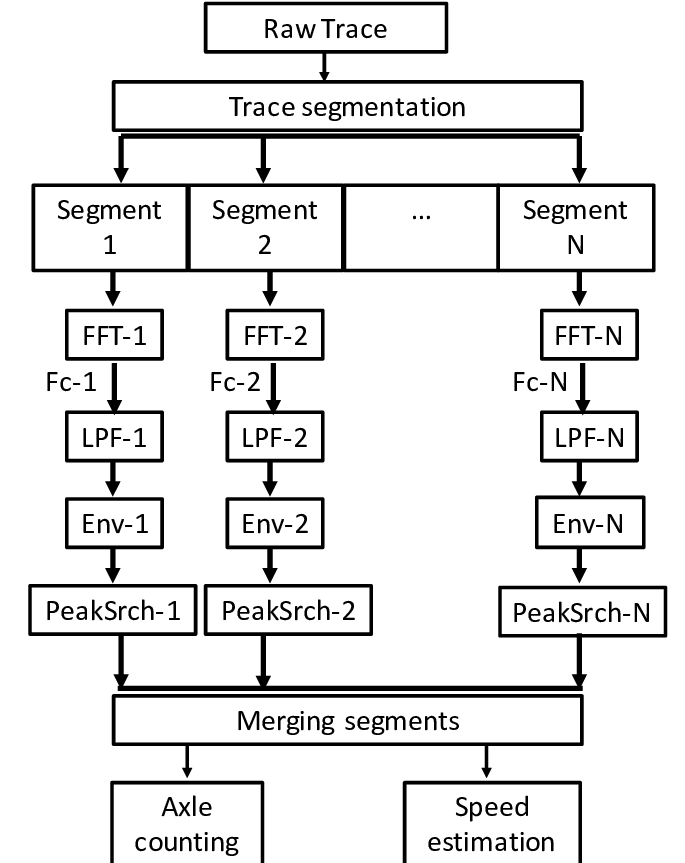
[](https://github.com/Saadia-Hassan/Application_of_FFT_with_FIR_filter/blob/master/images/DFT%20vs.%20FFT.jpg)

Figure 7 DFT VS FFT

For a sample set of 1024 values, the FFT is 102.4 times faster than the discrete Fourier transform (DFT). The basis for this remarkable speed advantage is the bit-reversal scheme of the CooleyTukey algorithm.

Thus for the application Four sinusoidal signals with different frequencies were used. A noisy signal was generated.  The four signals along with noise were added to generate a single signal with different frequency components. The FFT of this signal was computed using the fft() function and the frequency spectrum was plotted to visualize the frequency components.

# Flowchart:

****

# Matlab Code for Implementing application:

## Main file:

|  |
| --- |
| clc; |
|  | clear variables; |
|  | close all; |
|  |  |
|  | %% Setting a sampling frequency and generating a noise signal |
|  | Fs = 1000; |
|  | t = 0:1/Fs:2; |
|  |  |
|  | noiseAmplitude=randi(5); %Amplitudes random to 5) |
|  | noise=noiseAmplitude\* rand(1, length(t)); %Random 5 mul with same length random number |
|  |  |
|  | %% Generating sine waves with noise |
|  | y1 = 5\*sin(2\*pi\*60\*t) + noise; |
|  | y2 = 2.5\*sin(2\*pi\*100\*t) + noise; |
|  | y3 = 1.2\*sin(2\*pi\*150\*t)+ noise; |
|  | y4 = 7\*sin(2\*pi\*250\*t) + noise; |
|  |  |
|  | %% Plotting all the signals seperately |
|  | subplot(2,2,1); |
|  | plot(t, y1, 'r'); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Signal 1'); |
|  |  |
|  | subplot(2,2,2); |
|  | plot(t, y2, 'y'); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Signal 2'); |
|  |  |
|  | subplot(2,2,3); |
|  | plot(t, y1, 'g'); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Signal 3'); |
|  |  |
|  | subplot(2,2,4); |
|  | plot(t, y1, 'c'); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Signal 4'); |
|  |  |
|  | %% Summing up all signals and playing sound using sound(y, Fs) function |
|  | y = y1 + y2 + y3 + y4; |
|  | sound(y, Fs); |
|  |  |
|  | figure(1) |
|  | plot( t, y1, 'r'); % Red |
|  | hold on; |
|  |  |
|  | plot(t, y2, 'y'); % Yellow |
|  | hold on; |
|  |  |
|  | plot(t, y3, 'g'); % Green |
|  | hold on; |
|  |  |
|  | plot(t, y4, 'c'); % Cyan |
|  | hold off; |
|  |  |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Overall Signal'); |
|  |  |
|  | %% Finding FFT of signal |
|  | L = length(y); |
|  | n = 2^nextpow2(L); |
|  |  |
|  | Y = fft(y,n); |
|  |  |
|  | %% Plot the signal in Frequency Domain |
|  | f = Fs\*(0:(n/2))/n; |
|  | P = abs(Y/n); |
|  | Y\_val = P(1:n/2+1); |
|  | plot(f,Y\_val) |
|  | xlabel('Frequency'); |
|  | ylabel('Amplitude'); |
|  | title('Frequency response of actual signal'); |
|  |  |
|  | %% Applying LPF to actual signal |
|  | LPFSignal = filter(lpf, y); |
|  | plot(t, LPFSignal); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('LPF Signal'); |
|  |  |
|  | sound(LPFSignal, Fs) |
|  |  |
|  | %% Apply FFT to LPF signal and plot to see the retained frequencies |
|  | LPFSignalFFT = fft(LPFSignal, n); |
|  |  |
|  | f = Fs\*(0:(n/2))/n; |
|  | LPFValue = abs(LPFSignalFFT/n); |
|  | AmpOfLPFSignal = LPFValue(1:n/2+1); |
|  | plot(f,AmpOfLPFSignal); |
|  | xlabel('Frequency'); |
|  | ylabel('Amplitude'); |
|  | title('Low Pass Filtered Signal'); |
|  |  |
|  |  |
|  | %% Applying BPF to actual signal |
|  | BPFSignal = filter(bpf, y); |
|  | plot(t, BPFSignal); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('BPF Signal'); |
|  |  |
|  | sound(BPFSignal, Fs) |
|  |  |
|  | %% Apply FFT to BPF signal and plot to see the retained frequencies |
|  | BPFSignalFFT = fft(BPFSignal, n); |
|  |  |
|  | f = Fs\*(0:(n/2))/n; |
|  | BPFValue = abs(BPFSignalFFT/n); |
|  | AmpOfBPFSignal = BPFValue(1:n/2+1); |
|  | plot(f,AmpOfBPFSignal); |
|  | xlabel('Frequency'); |
|  | ylabel('Amplitude'); |
|  | title('Band Pass Filtered Signal'); |
|  |  |
|  |  |
|  | %% Comparing before and after filtering |
|  | subplot(3,2,1); |
|  | plot(t, y); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Actual Signal in Time Domain'); |
|  |  |
|  | subplot(3,2,2); |
|  | plot(f, Y\_val); |
|  | xlabel('Frequency'); |
|  | ylabel('Amplitude'); |
|  | title('Actual Signal in Frequency Domain'); |
|  |  |
|  | subplot(3,2,3); |
|  | plot(t, LPFSignal); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Low Pass Filtered Signal in Time Domain'); |
|  |  |
|  | subplot(3,2,4); |
|  | plot(f,AmpOfLPFSignal); |
|  | xlabel('Frequency'); |
|  | ylabel('Amplitude'); |
|  | title('Low Pass Filtered Signal in Frequency Domain'); |
|  |  |
|  | subplot(3,2,5); |
|  | plot(t, BPFSignal); |
|  | xlabel('Time'); |
|  | ylabel('Amplitude'); |
|  | title('Band Pass Filtered Signal in Time Domain'); |
|  |  |
|  | subplot(3,2,6); |
|  | plot(f,AmpOfBPFSignal); |
|  | xlabel('Frequency'); |
|  | ylabel('Amplitude'); |
|  | title('Band Pass Filtered Signal in Frequency Domain'); |

## BPF File:

|  |
| --- |
| function Hd = BPF |
|  | %BPF Returns a discrete-time filter object. |
|  |  |
|  | % MATLAB Code |
|  | % Generated by MATLAB(R) 9.5 and DSP System Toolbox 9.7. |
|  | % Generated on: 15-May-2020 23:50:03 |
|  |  |
|  | % Equiripple Bandpass filter designed using the FIRPM function. |
|  |  |
|  | % All frequency values are in Hz. |
|  | Fs = 1000; % Sampling Frequency |
|  |  |
|  | N = 60; % Order |
|  | Fstop1 = 25; % First Stopband Frequency |
|  | Fpass1 = 50; % First Passband Frequency |
|  | Fpass2 = 260; % Second Passband Frequency |
|  | Fstop2 = 285; % Second Stopband Frequency |
|  | Wstop1 = 1; % First Stopband Weight |
|  | Wpass = 1; % Passband Weight |
|  | Wstop2 = 1; % Second Stopband Weight |
|  | dens = 20; % Density Factor |
|  |  |
|  | % Calculate the coefficients using the FIRPM function. |
|  | b = firpm(N, [0 Fstop1 Fpass1 Fpass2 Fstop2 Fs/2]/(Fs/2), [0 0 1 1 0 ... |
|  | 0], [Wstop1 Wpass Wstop2], {dens}); |
|  | Hd = dsp.FIRFilter( ... |
|  | 'Numerator', b); |
|  |  |
|  | % [EOF] |

## LPF File:

|  |
| --- |
| function Hd = LPF |
|  | %LPF Returns a discrete-time filter object. |
|  |  |
|  | % MATLAB Code |
|  | % Generated by MATLAB(R) 9.5 and DSP System Toolbox 9.7. |
|  | % Generated on: 15-May-2020 23:00:49 |
|  |  |
|  | % Equiripple Lowpass filter designed using the FIRPM function. |
|  |  |
|  | % All frequency values are in Hz. |
|  | Fs = 1000; % Sampling Frequency |
|  |  |
|  | N = 30; % Order |
|  | Fpass = 100; % Passband Frequency |
|  | Fstop = 120; % Stopband Frequency |
|  | Wpass = 1; % Passband Weight |
|  | Wstop = 1; % Stopband Weight |
|  | dens = 20; % Density Factor |
|  |  |
|  | % Calculate the coefficients using the FIRPM function. |
|  | b = firpm(N, [0 Fpass Fstop Fs/2]/(Fs/2), [1 1 0 0], [Wpass Wstop], ... |
|  | {dens}); |
|  | Hd = dsp.FIRFilter( ... |
|  | 'Numerator', b); |
|  |  |
|  | % [EOF] |

# Results and Steps Followed:

Four sinusoidal signals with different frequencies were used to do this project as shown in figure (though the frequencies aren't clear). You'll find the hidden frequencies shortly.

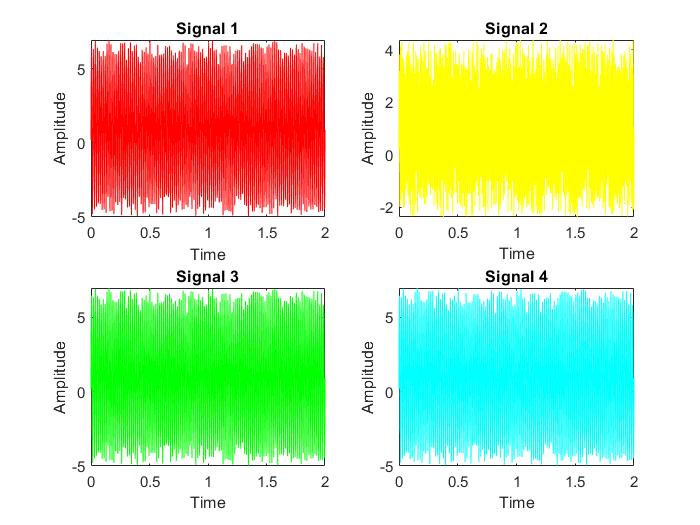
[](https://github.com/Saadia-Hassan/Application_of_FFT_with_FIR_filter/blob/master/images/AllSignals.jpg)

Figure 8 All Signals

A noisy signal was generated using MATLAB rand command which was added to all the signals. The four signals along with noise were added to generate a single signal with different frequency components, have a look (not clear either!)

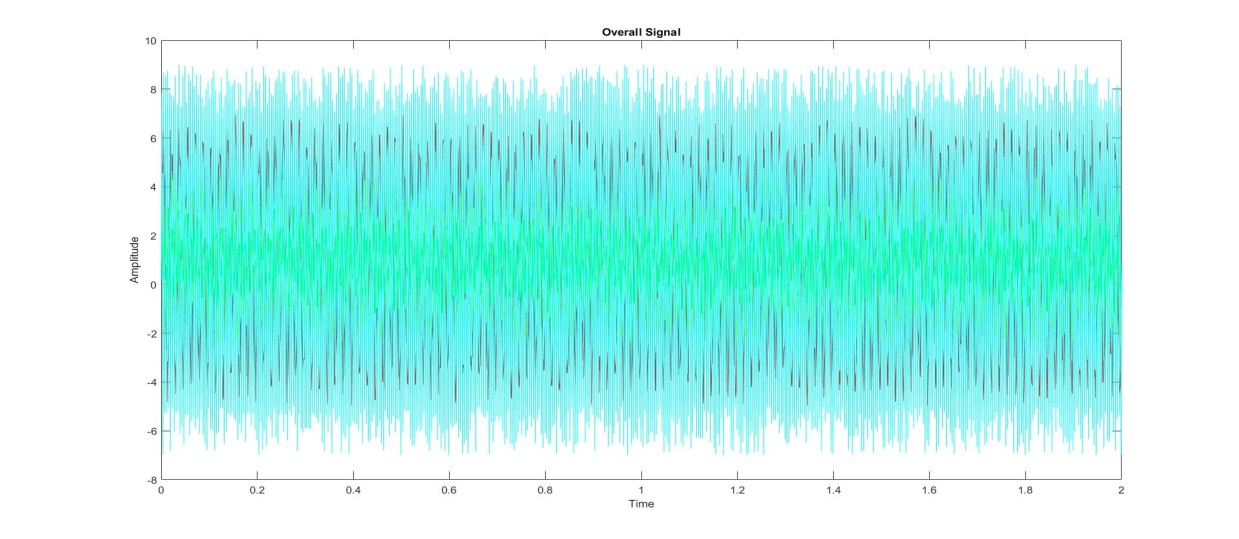
[](https://github.com/Saadia-Hassan/Application_of_FFT_with_FIR_filter/blob/master/images/OverallSignal.jpg)

Figure 9 Overall Signals

The FFT of this signal was computed using the fft() function and the frequency spectrum was plotted to visualize the frequency components. Now you clearly see the frequency components along with the noise.

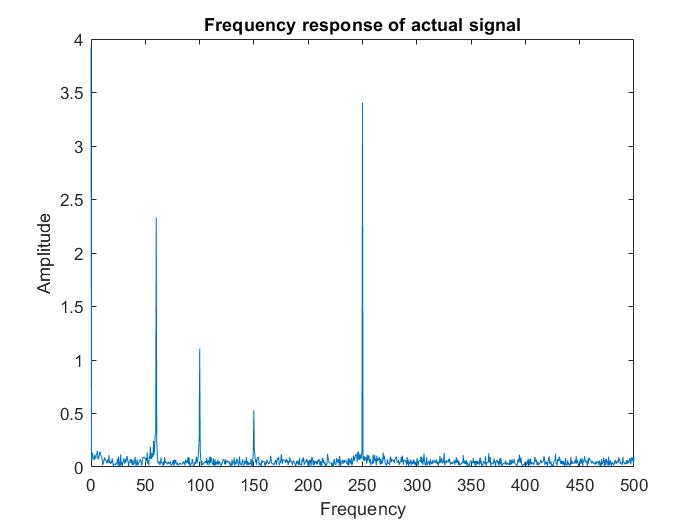
[](https://github.com/Saadia-Hassan/Application_of_FFT_with_FIR_filter/blob/master/images/FreqResponseOriginalSignal.jpg)

Figure 10 Frequency Response original signal

Next, a BPF was generated using the Filter Designer Toolbox and desired parameters were set which looked something like this, see the Magnitude Spectrum? Interesting, isn't it? You can change the parameters as you like to get the desired outcome. I wanted to remove the noise and retain the frequencies which I initially created my signal with. Thus the parameters.

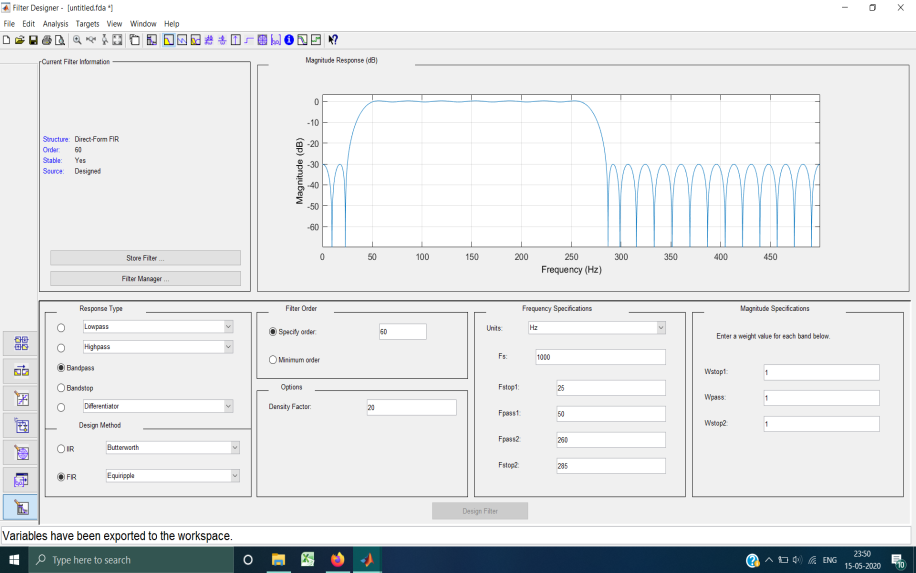
[](https://github.com/Saadia-Hassan/Application_of_FFT_with_FIR_filter/blob/master/images/BPF_Toolbox.png)

Figure 11 BPF\_Toolbox

Thus designed filter was exported to the workspace as an object, this filter was applied to the actual signal and corresponding FFT was computed to check for the frequency components retained.

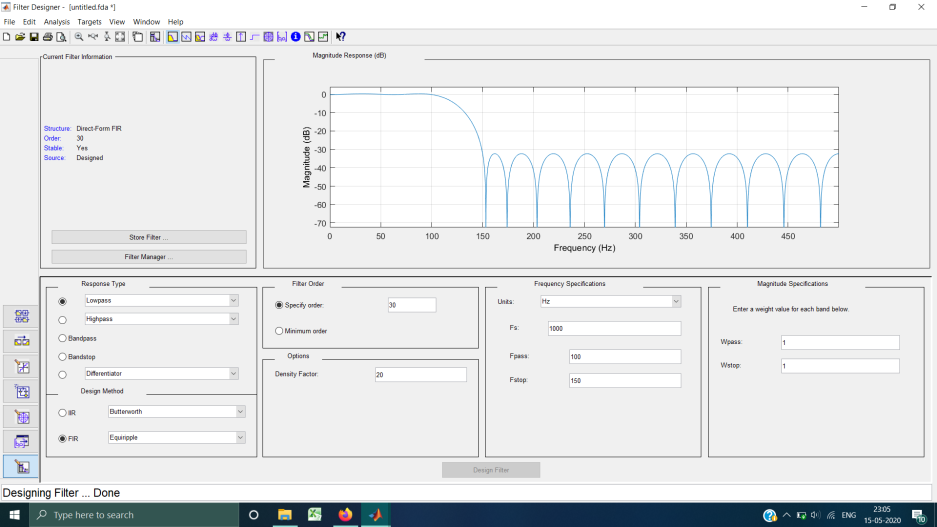


Figure 12 LPF\_toolbox

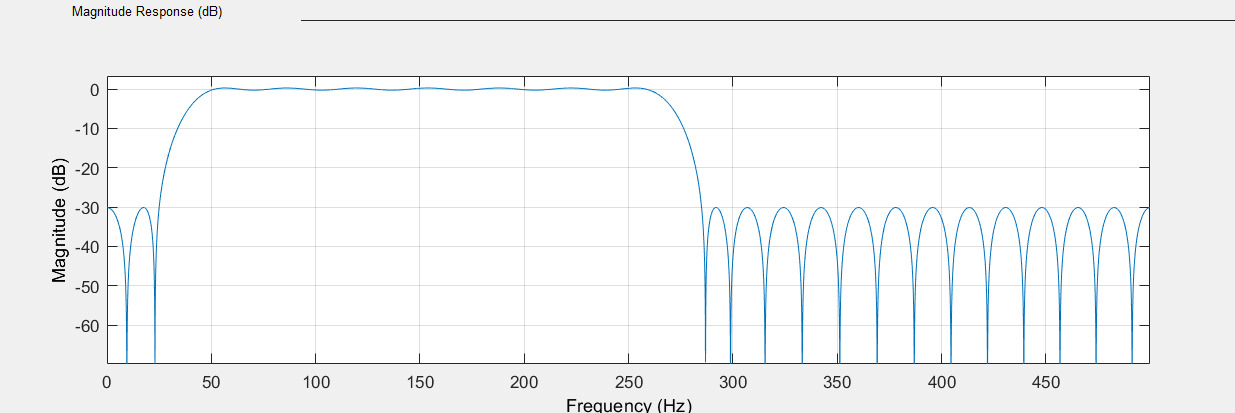


Figure 13 MagResponse of BPF

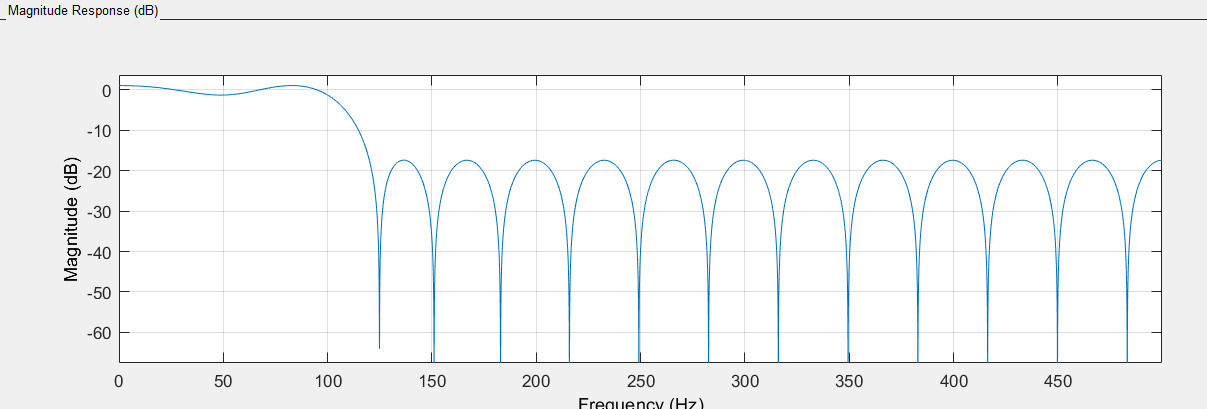


Figure 14 Mag response of LPG

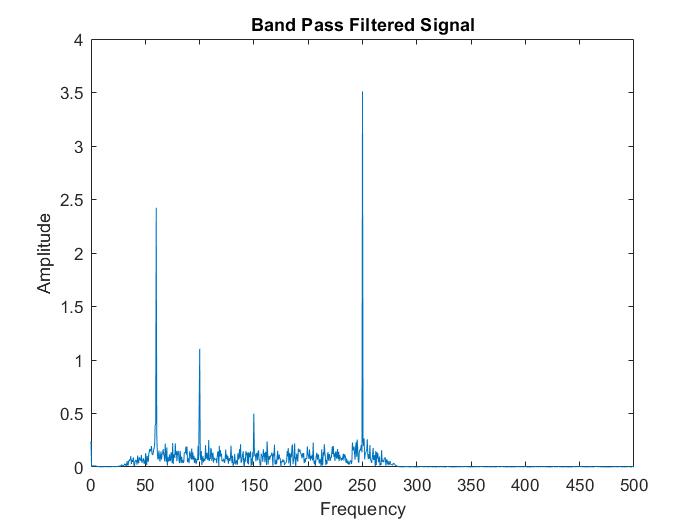
[](https://github.com/Saadia-Hassan/Application_of_FFT_with_FIR_filter/blob/master/images/FreqResponseAfterPassingBPF.jpg)

Figure 15 Freq Response After Passing BPF

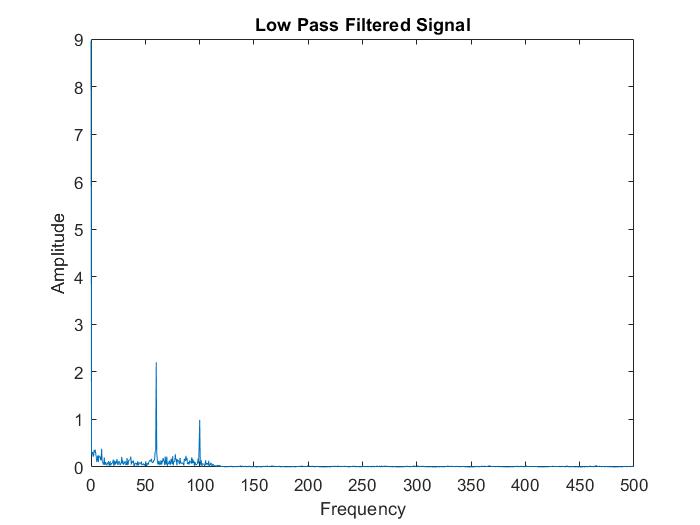


Figure 16 Freq Response After Passing LPF

I did this with both LPF and BPF just to get the hang of it, you can do as many modifications and as much processing as you like and have fun with the toolbox. The possibilities are wide.

Here's an image comparing all the signals generated in the process.

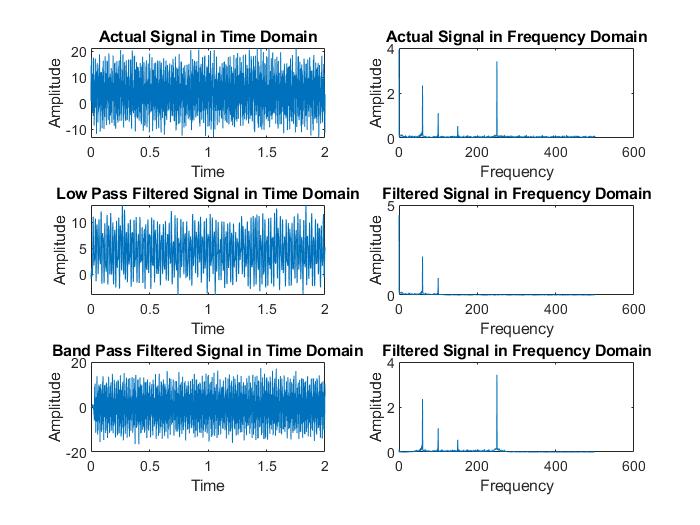
[](https://github.com/Saadia-Hassan/Application_of_FFT_with_FIR_filter/blob/master/images/ComparisonOfAllSignals.jpg)

Figure 17 Comparison of all Signals

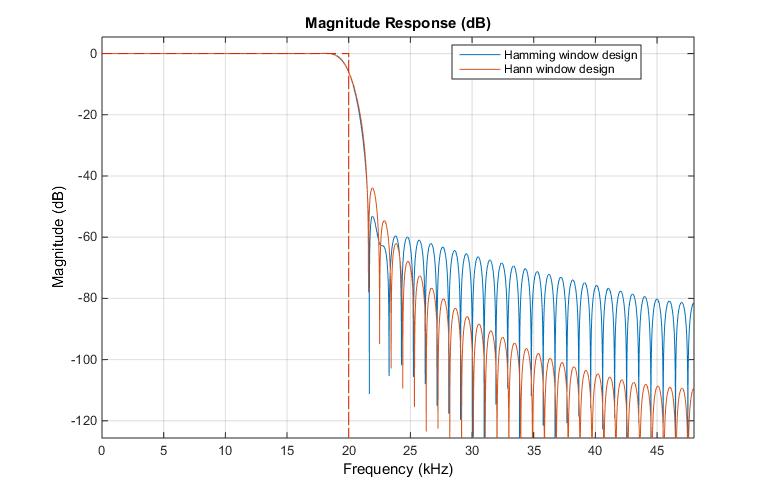


Figure 18 Magnitude response

Figure 19

# Error:

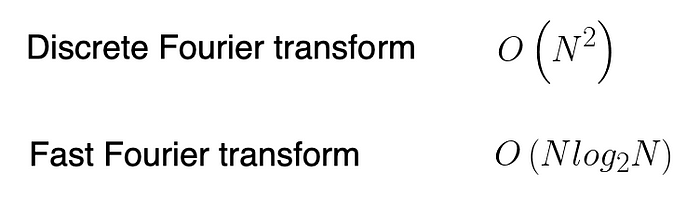
We had used Fir filter due to its stability. But The advantage of IIR filters over FIR filters is that IIR filters usually require fewer coefficients to execute similar filtering operations, that IIR filters work faster, and require less memory space.Also only FIR filters can be used with FFT.

The FFT is much faster than the DFT but requires more memory. The FFT can be used to compute the DFT of a sequence that is not a power of two, while the DFT can only be used to compute the DFT of a sequence that is a power of two.

To summarize both FIR and FFT requires more memory.

# CONCLUSION:

The Fast Fourier Transform (FFT) is an algorithm that determines Discrete Fourier Transform of an input significantly faster than computing it directly. In computer science lingo, the FFT reduces the nu mber of computations needed for a problem of size N from O(N^2) to O(NlogN).



Therefore, Fast Fourier Transform with FIR can be used in the simplest of ways to remove unwanted frequencies in a given signal. For the implementation of such application we took Four sinusoidal signals with different frequencies. Then A noisy signal was generated using MATLAB rand command which was added to all the signals. The four signals along with noise were added to generate a single signal with different frequency components. After that The FFT of this signal was computed using the fft() function and the frequency spectrum was plotted to visualize the frequency components. Next, a BPF was generated using the Filter Designer Toolbox and desired parameters were set. Thus designed filter was exported to the workspace as an object, this filter was applied to the actual signal and corresponding FFT was computed to check for the frequency components retained. It was done with both LPF and BPF.

**How was Objective Achieved?**

A detail research was done on FFT and DFT, along with FIR and IIR .After Identifying the requirements an application of filtering unwanted frequencies form a signal using FFT with FIR was implemented on MATLAB. The application included a noisy signal generated and added into four sinusoidal signals. After that the FFT of this signal was computed and then BPF AND LPF was generated using toolbox.

**Weakness of the proposed system:**

To summarize both FIR and FFT requires more memory.

**Future Directions:**

In future an algorithm should be generated to use IIR filter with FFT. To Increase efficiency of the system. As for a really big project FFT with FIR would require a large amount of memory, where as using IIR filter with FFT will help to rude this limitations.

**Bibliography**

[1]<https://www.analog.com/media/en/technical-documentation/dsp-book/dsp_book_Ch12.pdf>

[2]<https://openaccess.thecvf.com/content_CVPR_2020/html/vahid_Butterfly_Transform_An_Efficient_FFT_Based_Neural_Architecture_Design_CVPR_2020_paper.html>

[3] <https://pubs.aip.org/aip/acp/article-abstract/2523/1/020054/2874909/FFT-implementation-of-OFDM-for-future-SG?redirectedFrom=fulltext>

[4] <https://www.fftw.org/fftw-paper-icassp.pdf>

[5]https://www.researchgate.net/publication/319362822\_Sparse\_Interpolation\_the\_FFT\_Algorithm\_and\_FIR\_Filters

[6]<https://flylib.com/books/en/2.729.1/fast_fir_filtering_using_the_fft.html>

[7]<https://link.springer.com/chapter/10.1007/978-3-540-30494-4_4>

[8]<https://mathworld.wolfram.com/FastFourierTransform.html>

[9]<https://www.eetasia.com/tips-for-using-an-fft-with-an-oscilloscope/#:~:text=The%20fast%20Fourier%20Transform%20(FFT,versus%20frequency%20(Figure%201)>.

[10]<https://www.tek.com/en/support/faqs/what-fft-fast-fourier-transform-math-function-oscilloscope-useful>

[11]<https://opencores.org/projects/fft_fir_filter>

[12]<https://towardsdatascience.com/fast-fourier-transform-937926e591cb>