# Filtering in Frequency Domain using FFT

EE18ACMTECH11005,EE18MTECH11004

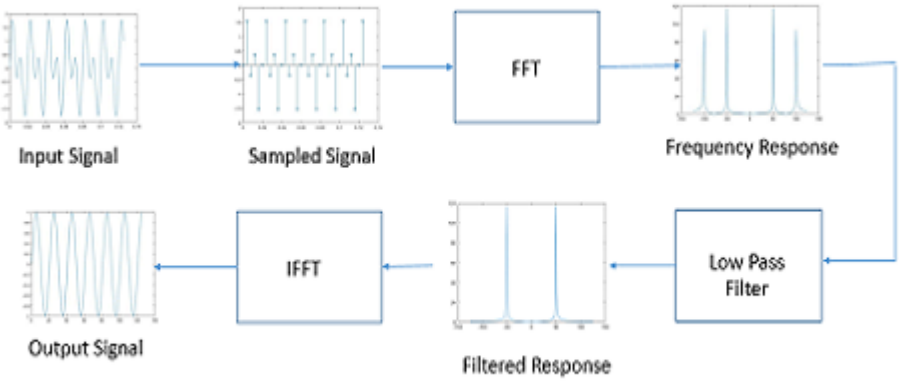
April 26, 2019

# 1 Introduction

The objective of this project is to implement the Fast Fourier Transform (FFT) and filtering the frequencies using low pass filter in frequency domain and taking the Inverse Fast Fourier Transform (IFFT) on a FPGA hardware. This project concentrates on developing FFT and IFFT. The work includes in designing of the module. The design uses 8-point FFT and IFFT for the processing module which indicate that the processing block contain 8 inputs data. The Fast Fourier Transform and Inverse Fast Fourier Transform are derived from the main function which is called Discrete Fourier Transform (DFT). The idea of using FFT/IFFT instead of DFT is that the computation of the function can be made faster where this is the main criteria for implementation in the digital signal processing. In DFT the computation for N-point of the DFT will calculate one by one for each point. While for FFT/IFFT, the computation is done simultaneously and this method saves quite a lot of time. The project uses radix-2 DIT-FFT algorithm which breaks the entire DFT calculation down into a number of 2- point DFTs. Each 2-point DFT consists of a multiply-and-accumulate operation called a butterfly

The modules are designed using Verilog programming language and implemented using FPGA Ico board. The board is connected to computer through serial port and Arduino software is used to provide interface between user and the hardware. All processing is executed in FPGA Ico board and user only requires to send the input data points to the hardware through Arduino software.Output is read back from Ico board to Arduino. Input and output signals are displayed to the user using the serial plotter in the Arduino software and the results is compared using simulation software.

# 2 Block Diagram



We have taken the mixed frequencies(40 and 140Hz) input signal and sampled the input signal to get 8 sample points.Feed these sample points to the FFT module and after getting the FFT coefficients, the low pass filtering the 40 Hz is done by setting middle coefficients to zero which is same as multiplying window with the cutoff frequencies.Filtered coefficients are passed to IFFT module and the output is passed to arduino and displayed using serial plotter in arduino.

# 3 FFT Algorithm

FFT is very popular for transforming a signal from time domain to frequency domain and it has quite an interesting history starting from 1805, when Carl Fredrich Gauss tried to determine the orbits of various asteroids from sample locations. He thereby developed the DFT algorithm even before Fourier published his results for the same in 1822. He developed an algorithm similar to that of Cooley [4] and Tukey [4] but Gauss never published his method or algorithm in his lifetime. It took another 160 years until Cooley and Tukey reinvented the FFT. During this period of 160 years from 1805 to 1965, many other scientists invented various efficient algorithms to compute FFT but none of them was as general as that of Gauss's algorithm.
The results of FFT are same as that of DFT; the only difference is that the algorithm is optimised to remove the redundant calculations, the FFT greatly reduces the amount of calculations required and hence reduces the time required for the computation. Functionally, the FFT decomposes the set of data to be transformed into a series of smaller data sets to be transformed. Then it decomposes those smaller sets into even smaller sets. At each stage of processing, the results of the previous stage are combined in a special way. Finally, it calculates the DFT of each small data set.
DFT can be computed using the below given formula:

$$X[k] = \sum_{k=0}^{N-1} x_k \; e^{-\left(\frac{2\pi j}{N}\right)nk} \quad ; \qquad k = 0,1,\ldots.N-1.$$

A major drawback of this DFT algorithm is the computational complexity. For a sequence of length N, it has a complexity given as: O(N2 ) hence it is not a very efficient method and here the FFT comes into the picture. FFT significantly reduces the number of computations required for a sequence of length N from O(N2 ) to O(N logN) where log is the base-2 logarithm. FFT operates by decomposing an N point time domain signal into N time domain signals each composed of a single point. The second step is to calculate the N frequency spectra corresponding to these N time domain signals. Lastly, the N spectra are synthesized into a single frequency spectrum. So we split the N-point data

sequence into two N/2 point data sequences f1(n) and f2(n), corresponding to the even-numbered and odd-numbered samples of x(n) respectively.

$$f_1(n) = x(2n)$$

$$f_2(n) = x(2n+1)\,; \qquad n = 0,1,2,\dots,\frac{N}{2}-1.$$

Here f1(n) and f2(n) are obtained by decimating x(n) by a factor of 2, and hence the resulting FFT is called a decimation-in-time (DIT) algorithm. Now the N-point DFT can be expressed as:

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn} \;; \qquad k = 0,1,2,\dots,N-1.$$

$$= \sum_{n\ even}^{N-1} x(n)\, W_N^{kn} + \sum_{n\ odd}^{N-1} x(n)\, W_N^{kn}$$

$$= \sum_{m=0}^{\frac{N}{2}-1} x(2m) W_N^{2mk} + \sum_{m=0}^{\frac{N}{2}-1} x(2m+1) W_N^{(2m+1)k}$$

**IMPLEMENTATION OF FFT**

To compute FFT, we need to break a data set into smaller data sets and then compute the DFT of each small set. Figure 1 describes the implementation of 8-point DFT. We observe that computation is performed in three stages, starting with the computation of four 2-point DFTs, then two 4-point DFTs, and finally one 8-point DFT.

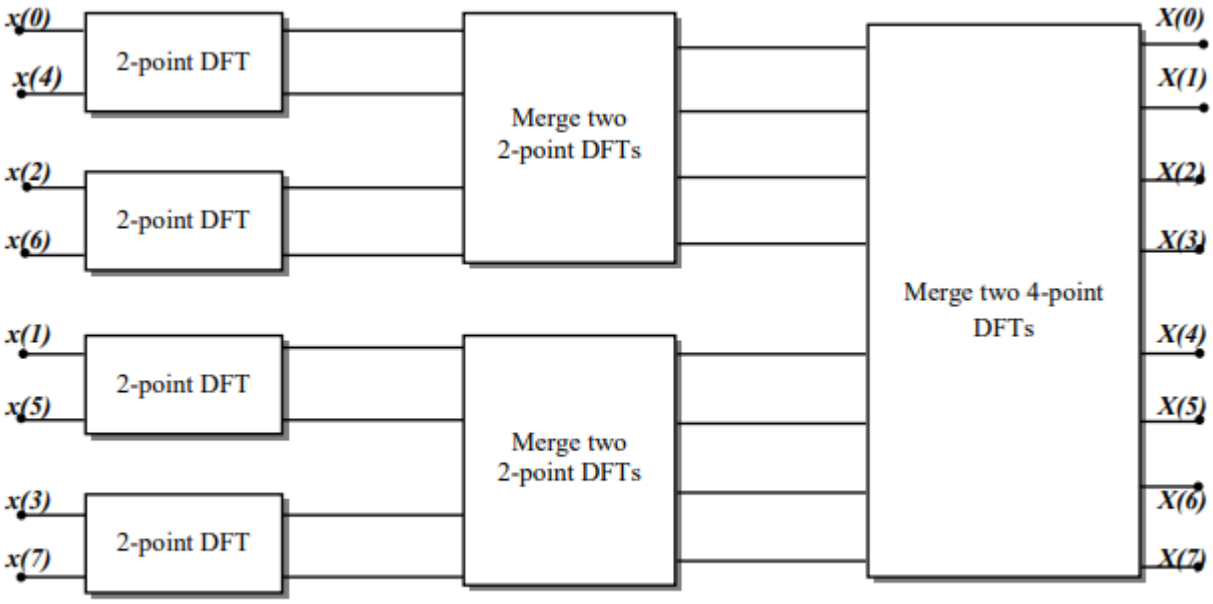



*Fig 1: FFT block diagram*

**BUTTERFLY UNIT**

In radix-2 Cooley-Tukey algorithm, butterfly is simply a 2-point DFT that takes two inputs and gives two outputs. Butterfly unit is the basic building block for FFT computation. The figure 2 shown below describes the basic butterfly unit used in FFT implementation.

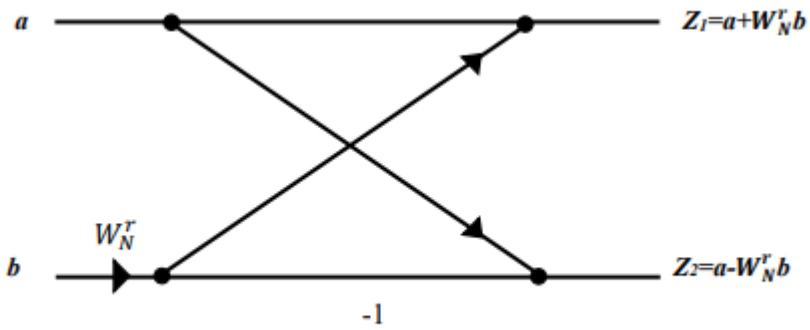



*Fig 2: Basic butterfly unit*

Implementation of FFT requires the computation of butterfly unit at first, which takes two complex inputs 'a' and 'b' and a twiddle factor 'W'. It generates two complex outputs 'Z1' and 'Z2'.

i) Input 'b' is multiplied with 'W' and then added to 'a' for the output 'Z1'.
ii) Input 'b' is multiplied with '-1' and then added to 'a' for the output 'Z2'.
This basic butterfly unit is replicated four times each in the three stages of the FFT and the final output is generated as shown in the figure 3 below:

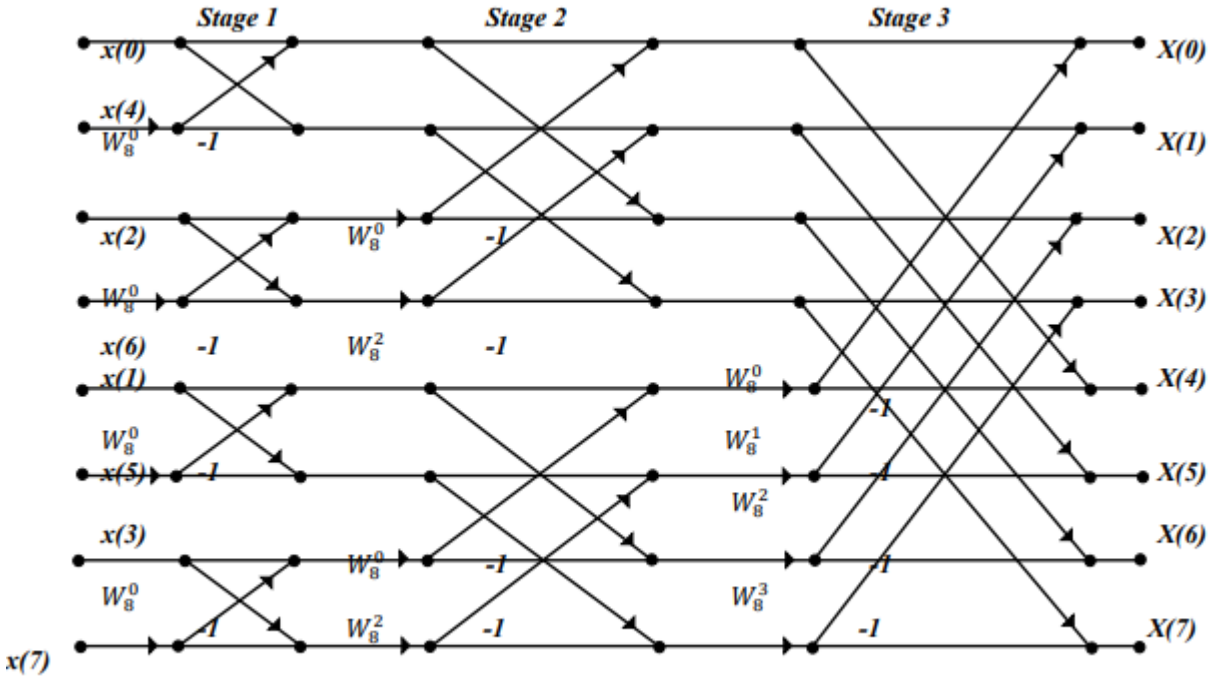



*Fig 3: FFT butterfly diagram*

**BIT REVERSAL**

In FFT computation, the input bits need to be bit reversed so as to obtain the output bits in the normal order. To achieve this, the input sequence is broken into even and odd parts based on their indexes. Then these even parts are again broken down into even and odd parts. Same is done with odd parts respectively. This procedure is continued until only two points are left. In this paper, the sequence is separated in time domain. For example, if we consider the case where N=8, input sequence is x(0), x(1), x(2), x(3), x(4), x(5), x(6), x(7) and after the bit reversal process, the output sequence obtained is x(0), x(4), x(2), x(6), x(1), x(5), x(3), x(7) as shown in table 1 below:

***Table 1: Bit reversal***

| Input | Address (binary) | Output (bit reversed) | Address (binary) |
|---|---|---|---|
| x(0) | 000 | x(0) | 000 |
| x(1) | 001 | x(4) | 100 |
| x(2) | 010 | x(2) | 010 |
| x(3) | 011 | x(6) | 110 |
| x(4) | 100 | x(1) | 001 |
| x(5) | 101 | x(5) | 101 |
| x(6) | 110 | x(3) | 011 |
| x(7) | 111 | x(7) | 111 |