Abstract

Wireless and mobile systems are one the major areas of today's research and development. MIMO OFDM systems are one of the widely employed schemes in these systems. In order to facilitate mass production these systems should consume low power, occupy little area and at the same time provide maximum speed. Apart from these requirements, these systems also demand multiple computations without increasing the hardware cost (Re-configurability). MIMO OFDM systems consist of different complex computational blocks; one such block is Fast Fourier Transform (FFT)/Inverse Fast Fourier Transform (IFFT). In this thesis, we have focussed on reducing the power and area requirements for the FFT/IFFT block. Low power architectures mainly target to reduce the computational complexity of the design. In this thesis, we have proposed adaptive re-configurable low power architecture for the computation of FFT/IFFT in a MIMO OFDM system. The design is implemented using pipeline architecture and four parallel datapaths. The design can compute one 128 point FFT or one 64 point FFT or two 64 point FFT at a time by using the same hardware. The twiddle factor multiplication has been dealt strategically by using shift and adds operations to reduce the power consumption. Hard wired connections are used in order to avoid large multiplexing of complex data. The design is a register based architecture and does not use any memory modules. The re-configurability feature saves a lot of hardware, which would otherwise be needed to compute 128/64 point FFT in conventional designs. The design completes one parallel to parallel (i.e. when all available data are stored and first set of computed values are generated in parallel) 128/64 point FFT/IFFT in **20 cycles** and has a average dynamic power consumption of **56mW** at 20MHz and 1.8V voltage supply. The area of the design is **4.5mm^2**. The primary application of this design is in the IEEE 802.11n standard but it can also be used in any application that demands fast operation combined with low power consumption.

.

Acknowledgements

First of all I owe my deepest gratitude to my family members who have supported my decision of attaining higher education in University of Bristol and motivated me throughout the course.

I would also like to thanks my course director Dr. Paul Warr for his support and guidance throughout the coursework.

This work would not have been possible without the guidance of my Supervisor Prof. Dhiraj Pradhan and Co- Supervisor Dr. Jimson Mathew. They have helped me in every stages of the project and made available to me all the resources needed to complete the project.

I am indebted to all my colleagues to support me. I would specially like to thanks my friend Pranav Yeolekar who stood by me in my difficult times and Srikanth Santhanam for his encouragement.

List of Tables

List of Figures

Table of Contents

Chapter 1. Introduction

Last few decades have experienced an exponential growth in the number of transistors per chip. This growth has marked the tremendous progress in the performance and functionality of semiconductor devices. Intel Corporation's cofounder, Gordon Moore, predicted in 1965 that number of transistors per chip will get doubled every 18 or 24 months. Starting from 2200 transistors in Intel's first (1971) microprocessor, the number has reached in billions per processor. However the advancement of technology also introduced new obstacles to maintain this exponential growth. To make the devices smaller and smaller, more and more transistors were packed on a single chip. With reduced area, the devices were supposed to work at the highest possible speed. Because of this, power dissipation of semiconductor chips started reaching colossal proportions and the manufactures were forced to draw their attention to reduce the power dissipation via innovation in process technology as well as in architecture.

One of the major focal point of research and development in today's world is mobile and wireless systems. The widespread deployment of embedded processors in mobile and wireless devices promises to open up new windows in applications. However power crisis and increasing computational demands on mobile computing results in a dramatic power mismatch in today's technology. With the advancement in Internet technologies Broad band wireless has become an essential part of one's life. Broad band wireless systems based on MIMO OFDM (Multiple Input Multiple output Orthogonal Frequency Division Multiplexing) allows packet based high data rate communication suitable for mobile Internet application. The system needs to compute within defined interval to successfully transmit or receive the data. Apart from high speed of operation, the system needs to be energy efficient since it is primarily aimed to be used in portable and mobile devices. With the mobile phones, shrinking to the pocket size, a general purpose DSP cannot be used to speed up the process, as it requires large area and also the average power consumption is high as compared to dedicated hardware.

In a MIMO OFDM system, to transmit the data from different antennas, data needs to be converted from time domain to frequency domain. This step plays a significant role in the correct transmission of the data. FFT/IFFT blocks in the architecture of a MIMO OFDM system perform this task. Due to its non linearity, FFT block consumes a huge amount of power. A number of architectures and algorithms have been proposed by different authors to reduce the power consumption and increase the speed of computation suggesting that this is quite an important topic in the industry.

In this thesis our main focus will be on developing an energy and area efficient algorithm for the hardware implementation of FFT/IFFT module in MIMO OFDM systems. We will exploit different low power techniques to provide an efficient solution. We will also focus on developing a reconfigurable design which can adapt itself to perform multiple FFTs using the same hardware. Re-configurability will help in saving a lot of hardware which is needed otherwise to perform different FFTs. Since timing constraints are associated with MIMO OFDM systems, therefore design will be tested with different parallel datapaths to increase the efficiency and speed. Different datapaths will help the designer to choose the optimum datapath according to performance parameters such as area, power and speed. The results will pave the way for a new approach to compute FFT/IFFT with minimal power consumption and reduced area.

1.1 Motivation

A MIMO OFDM system offers high throughput rate and improved link reliability without increasing the bandwidth. Because of these advantages these systems are an important part of modern wireless technologies such as 3GPP, IEEE 802.11n (Wifi), HSPA+, WiMax and 3GPP LTE. MIMO OFDM is also planned to be used in upcoming wireless and mobile radio telephone standards such as 4G and 3GGP2.

For commercial deployment and mass production, these systems need to be made more power efficient. Apart from being power efficient these systems need to be re-configurable. Power reduction can be achieved by working at the architecture level. One of the major complex computational block in MIMO OFDM systems is FFT/IFFT. This block, due to its high complexity, consumes most of the power.

The primary technological motivation for this proposal is to derive and investigate an unconventional reconfigurable architecture that can adapt itself to compute multiple FFT/IFFTs satisfying timing constraints with moderate silicon area and low power consumption.

1.2 Outline

The report is properly structured in 8 Chapters.

Chapter 1 gives a brief introduction about the need of low power architectures in wireless and mobile applications. The basic motivation for the thesis and a brief outline of the thesis is given in this chapter followed by the Aims and objectives of the project.

Chapter 2 focuses on the relation between MIMO OFDM and FFT. This section explains in brief about MIMO OFDM systems and then explains the block diagram of a typical MIMO OFDM receiver in which FFT/IFFT block plays a key role. Sub-section 2.1 of the report provides system specifications and tries to identify the problems with trivial and current architectures.

Chapter 3 gives a brief introduction of FFT, which gives an insight on the basics of FFT. The sub section 3.1 describes briefly the importance by giving some standard comparisons and explaining properties. In section 3.2 evolution of FFT has been discussed. In this section we have discussed about various proposed algorithms and explained in detail some of the important and popular algorithms by giving some standard equations.

Chapter 4 explains about different hardware architectures and also focuses on previous work done on these architectures.

Chapter 5 formulates the chosen algorithm according to MIMO OFDM systems. This chapter explains the basic algorithm to be implemented in next chapter.

Chapter 6 gives the basic block diagram of the proposed design followed by detailed description of each block in sub sections. In sub section 6.3 a brief description has been given on the implementation of the design. In next subsections (6.4 and 6.5) use of low power techniques and optimisation in the design has been discussed.

Chapter 7 discusses the result of implementation of the design. Sub section 7.2 evaluates the different datapath designs and summarises the optimum datapath. Section 7.3 gives the detailed result of optimum data path. FPGA implementation and layout of the design has also been discussed in this Chapter. Section 7.6 discusses about the main features of the design and also compares the design with other proposed designs.

Chapter 8 concludes the thesis work and also work that can be done in future to enhance the performance of the current design.

1.3 Aims & Objectives

The basic aim of this project is to research and explore better design methodology for the hardware implementation of FFT/IFFT in MIMO OFDM systems.

The main objectives of this thesis are as follows-

(1) Understanding and developing an appropriate algorithm to compute FFT/IFFT involving minimal hardware complexity.
(2) Designing a reconfigurable system that can compute multiple FFTs using the same hardware to reduce the power consumption and silicon area.
(3) Implementation of the design using different data paths to increase the speed and efficiency of the system.
(4) To understand and exploit different low power techniques so as to enhance the performance of the system.
(5) Perform area and power analysis of the design.
(6) Implementation of the design on different FPGAs to check the performance.

Chapter 2. MIMO OFDM and FFT

2.1 Introduction

One of the major challenges in upcoming wireless communications system design is improved link reliability and increased spectral efficiency. The wireless channel constitutes a hostile propagation medium, hence it is very much prone to ***fading*** and ***interference*** *from other users*. Diversity is a powerful means to combat fading and interference.

MIMO channels offers higher capacity and frequency diversity due to delay spread. Use of multiple antennas on the receiver side of a wireless system is a well known subject. But in mobile wireless applications it is difficult to deploy multiple antennas in handset. Orthogonal Frequency Division Multiplexing (OFDM) significantly reduces the complexity of having multiple antennas on the receiver end and therefore the use of MIMO technology in combination with OFDM (MIMO-OFDM) seems to be a lucrative solution for future broadband wireless systems.

2.2 MIMO OFDM system

A variety of schemes are being considered today to improve and enhance the range and performance of communication systems by using multiple antennas both at the transmitter and the receiver. MIMO (multiple input multiple output) systems seems to be most promising multiple antenna technology available today. A MIMO system makes use of multiple antennas at both the transmitter and receiver. Figure 1 below shows a basic MIMO system.

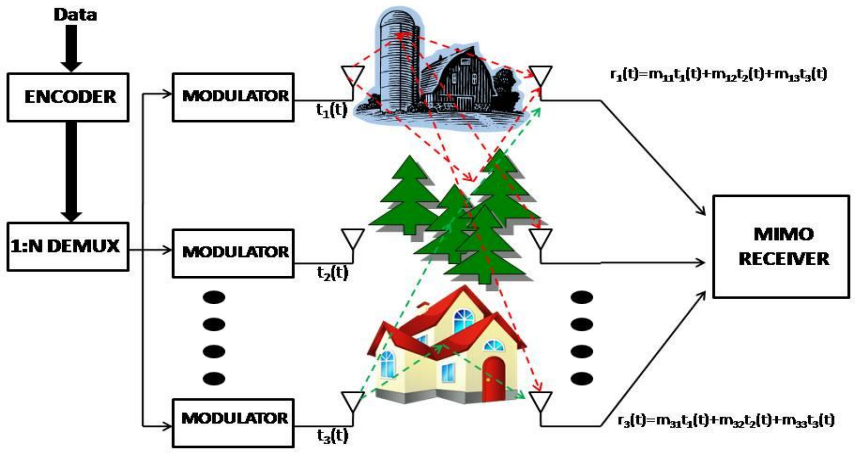


Figure 1: Basic MIMO system

In MIMO systems, independent data (assume t_1, t_2, …,t_N) is transmitted in the same frequency band simultaneously on different antennas. At the receiver end, the system uses multiple antennas to receive the signals. Let us assume that the system uses N antennas to receive the signals and the received signal of each antenna is represented by r_N.

Thus we have,

$$r_1 = m_{11}t_1 + m_{12}t_2 + \ldots.. + m_{1N}t_N$$

$$r_2 = m_{21}t_1 + m_{2}2t_2 + \ldots.. + m_{2N}t_N$$

$$\vdots$$

$$\vdots$$

$$r_N = m_{N1}t_1 + m_{N2}t_2 + \ldots.. + m_{NN}t_N$$

The above equations represent a set of linear equations. m_{ab} is the individual channel weights. In above equations transmitted independent signals are combined. To recover the independent data $\{t_j\}$, the channel can be treated as a matrix. In the process to recover the transmitted signal we have to construct a channel matrix $\boldsymbol{M}$ by estimating individual channel weights m_{ab}

After having the estimated $\boldsymbol{M}$, each received vector r is multiplied with the inverse of $\boldsymbol{M}$ to produce the estimated transmitted vector t.

With the addition of a pair of antennas added to the system, throughput increases linearly because of the fact that multiple data streams are transmitted from different antennas parallely. The advantage with MIMO systems is that they do not increase the bandwidth in order to increase the throughput. They make use of spatial dimension by increasing the number of spatial paths between transmitter and receiver.

Orthogonal Frequency Division Multiplexing (OFDM) is a popular digital modulation scheme used to reliably transmit data through radio channels. In OFDM modulation, the broadband channel is divided into many parallel sub channels as shown in Figure 2. OFDM transmits multiple messages over different frequencies simultaneously. Each of the frequencies is an integer multiple of a fundamental frequency.

The narrowband carriers transmitted are closely spaced in the frequency domain. The received signal sometimes adds up out of phase and sometimes adds up in phase resulting in a fluctuation in the received signal strength. At the receiver end, before extracting the transmitted data, the distortions on each sub channels need to be sensed and corrected. OFDM is effective in correcting such frequency selective distortions.

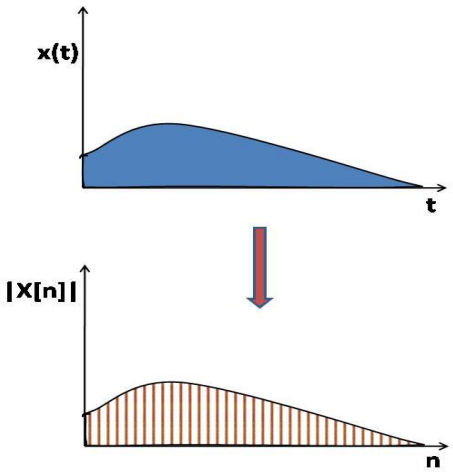


Figure 2: Broadband channel division into parallel sub channels

MIMO OFDM combines MIMO and OFDM techniques thereby achieving spectral efficiency and increased throughput. Multiple antennas' transmits independent OFDM modulated data simultaneously. At the receiver end, OFDM demodulated data on each of the sub channels is extracted from all the antennas by using MIMO decoding.

"In order to avoid the use of large number of modulators and filters at the transmitter and complementary filters and demodulators at the receiver, use of digital signal processing techniques, such as Fast Fourier transform (FFT) is desirable"[1]

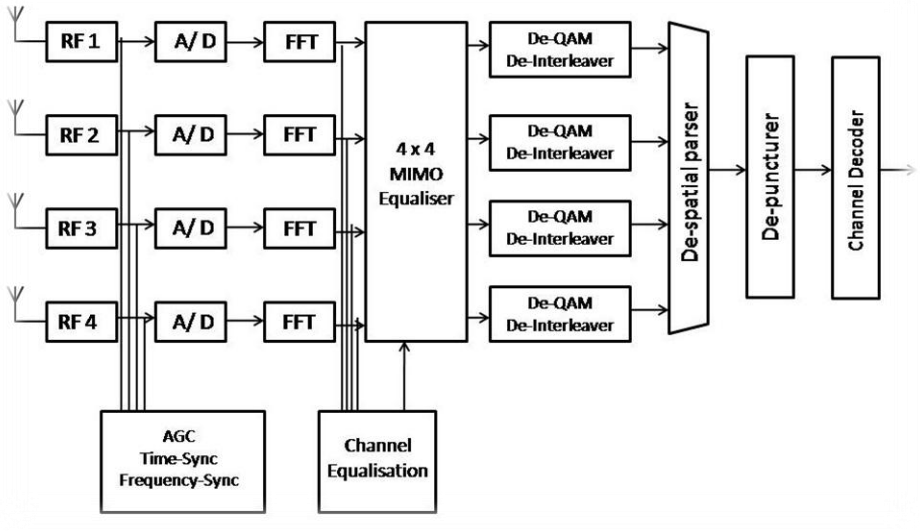


Figure 3: Block Diagram of OFDM receiver of IEEE 802.11n standard

In Figure 3 a block diagram of a receiver of IEEE 802.11n standard is shown. It contains four RFs, four analog-to-digital convertors (ADCs), **four FFTs**, a MIMO equalizer, four de-qam and de-interleaver, a de-spatial parser, a de-puncturer, a channel decoder, and a channel estimation block [2].

Inverse FFT (IFFT) and FFT are the *crucial computational blocks* to perform the baseband multicarrier modulation and demodulation in an OFDM based system. Arithmetic format used to implement FFT determines the computational accuracy, speed and hardware size of FFT. The theoretical computation of FFT is not easy due to the non linearity of FFT and because of these complexities this block consumes large resources in terms of computational power.

2.3 System specifications and Problem Identification

The general specifications of a MIMO OFDM system are as follows:

- The ***bandwidth*** of the transmitted signal is *20* or *40* MHz.
- The ***FFT*** size is *64* points or *128* points.
- The ***guard interval*** is 400 ns or 800 ns.
- Thus, the FFT/ IFFT processor has to calculate 128 points or 64 points within *3.6 or 4 μs* depending on the guard interval and FFT size.

In general, the main design concerns are:

- Silicon area covered by the design.
- How easily the architecture can be implemented in VLSI? *(routability)*
- Number of wire crossings and number of long wires, carrying signals to remote part of the design, needed in actual implementation *(Interconnect delay).*
- Power consumption of the design.

In the physical layer of a MIMO OFDM system, the FFT/IFFT processor is one of the highest computational complexity modules, which accounts for the major power consumption. Thus it's really difficult to realise the physical layer of the MIMO OFDM system with low power consumption and minimal and minimal hardware complexity, in particular computational complexity in VLSI implementation.

The implementation of FFT can be typically classified into two categories:

1. Direct implementations based on Fourier transform.
2. Direct hardware implementation of established FFT signal flow graph. [3]

A major problem with these solutions is that both the approaches are at the algorithmic level, which hardly takes account of its inference on architecture, data flow or chip design levels. Thus designs based on these approaches may have irregular structure; wire dominated and may involve high data storage overheads.

This eventually leads to several disadvantages in complex design systems, such as MIMO OFDM system, where there are tight constraints on timing and low power consumption is an implicit requirement.

The conventional Cooley- Turkey radix 2 FFT algorithm (explained in section 3.2.3.1) requires **(N/2) $\log_2$N** complex butterfly operations. Thus for a *64* point FFT computation *192* complex butterfly operations are required. Since one FFT has to be computed within *3.6µs* or *4µs* in a MIMO OFDM system, one butterfly operation needs to be computed within *20.8ns*. This will result in 48 MHz clock frequency for single butterfly architecture. In [3] it has been shown that shows that for radix-2 butterfly unit power consumption is *17mW* at 50 MHz when synthesised by using IHP 180 nm technology.

Moreover, to store complex twiddle factors one also needs memory for this butterfly unit which will also increase the power consumption. So overall the power consumption of the entire processor will be quite high.

In MIMO OFDM systems, the clock frequency of the input data is 20MHz or 40MHz so it is more suitable to operate the FFT module at this frequency. To satisfy the timing constraints and meet the clock frequency requirement, one has to use parallel architecture of the butterfly unit, which in turn increases the area and power consumption.

To meet the timing constraints, higher radix unit can be used. The *radix-4*(explained in 3.2.3.2) butterfly algorithm is very popular and it can easily satisfy current timing constraints. But a single radix-4 butterfly unit requires eight complex additions and three complex multiplications. Thus, 12 real multiplications and 16 real additions need to be completed at each cycle in order to carry out one radix-4 butterfly. This type of arrangement again results in significant power consumption, as in VLSI design multipliers unit are very power hungry.

Chapter 3. Introduction to FFT

Frequency analysis of discrete – time signals is usually performed on a digital signal processor, which may be a general purpose digital computer or a specially designed digital hardware. To perform frequency analysis on a discrete – time signal *{x(n)}*, the time domain sequence is converted to an equivalent frequency domain representation. The frequency domain represents the samples of its spectrum X (ω). Such a frequency domain representation leads to the Discrete Fourier transform (DFT).

The condition with DFT is that it requires an input function that is discrete and whose non-zero values are of finite duration. DFT only evaluates adequate frequency components to reconstruct/regenerate the *finite segment* of the discrete time signal. In DFT, usually a finite segment is one period of the infinitely extended periodic signal, but if this is not the case then we have to use *window function* to suppress the unwanted structure in the spectrum.

DFT is ideal for processing information stored in computers as the input to the DFT is a finite sequence of real or complex numbers. DFT is widely used in variety of areas including linear filtering, correlation and spectrum analysis. The main factor to enable the above applications is that DFT can be computed efficiently by using a Fast Fourier Transform (FFT).

Figure 4 shows graph of an input signal and its corresponding Discrete Fourier Transform respectively. The FFT is just another method of achieving the same result, but with less overhead involved in the calculations.

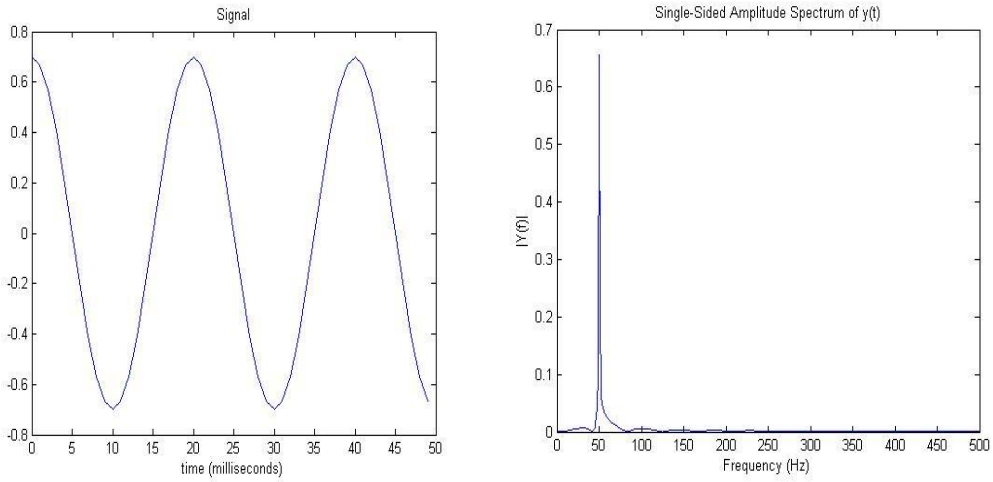


Figure 4: Discrete input signal and corresponding output

Improved efficiency and fast processing are the main features of FFT. These features have made many DFT based algorithms practical. In the next section, a mathematical proof of how FFT reduces complexity and employs less overhead has been described in brief.

3.1 Why FFT?

DFT is directly computed by using –

$$\forall k, 0 \le k \le N-1 : X(k) = \sum_{n=0}^{N-1} \left(x[n] e^{-(j\frac{2\pi kn}{N})} \right)$$

… **Eq.(3.1)**

Also,

$$\forall k, 0 \le k \le N-1 : X(k) = \sum_{n=0}^{N-1} x[n]\, W_n^{kn}$$

…**Eq.(3.2)**

where

$$W_n^{kn} = e^{-(j\frac{2\pi kn}{N})}$$

W_n^{kn} is called "***Twiddle factor***". As can be seen in Eq. 3.1, direct computation of the discrete Fourier transform is complicated to work out as it involves many additions and multiplications involving complex numbers. A simple eight sample signal would require 64 complex multiplications and 56 complex additions to directly work out the DFT. But in real world applications, realistic signals can have more samples (e.g. 512, 1024, 2048) and in that case the calculations required will mount up to unmanageable proportions (Refer to Table 1). The FFT is a simple and efficient method of computation and is much faster for large values of N, where N is the number of samples in the input sequence. The concept behind the FFT is *divide and conquers approach* (developed by Cooley and Turkey). In this approach the original N point sample is broken into two (N/2) sequence. This is done because it is easier to solve a series of smaller problems than to solve a large one. The direct computation of an N point DFT requires –

- N^2 – Complex Multiplications and
- N (N-1) – Complex additions.

whereas by using divide and conquer approach N point samples will be broken down into a series of 2 point samples which require only 1 multiplication and 2 addition and the recombination of the points which is minimal. Thus, a FFT algorithm requires –

- $(N/2)\log_r N$ – Complex multiplications and
- $N\log_r N$ – Complex additions , where r is the radix.

FFT algorithm reduces the $O(N^2)$ operations to O(NlogN) by simply ignoring trivial operations such as multiplication by unity and other such similar operations. Table 1 shows comparison of computational complexity for direct computation of DFT versus FFT

.

Number of points	Direct Computation of DFT		DFT computation using FFT (Radix 2 FFT)	
	Complex Multiplication	**Complex additions**	**Complex Multiplication**	**Complex additions**
N	N^2	N^2-N	$(N/2)log_2N$	$Nlog_2N$
4	16	12	4	8
16	256	240	32	64
64	4096	4032	192	384
128	16384	16256	448	896
256	65536	65280	1024	2048
1024	1048576	1047552	5120	10240

Table 1: Comparison of Computational complexity of DFTand FFT

As can be seen in the above table, with increase in the value of N the number of additions and multiplications increases enormously for direct computation method. The FFT algorithm offers comparatively less computational complexity and does not allow the operations to reach unmanageable proportions with an increase in the number of samples (N).

Direct computation of the DFT is basically inefficient primarily because it does not exploit the properties of the phase factor W_N. In particular, these two properties are:

- **Symmetry property :** $W_N^{k+N/2} = -W_N^{kn}$
- **Periodicity property:** $W_N^{kN} = W_N^{k(N+n)} = W_N^{(k+N)n}$

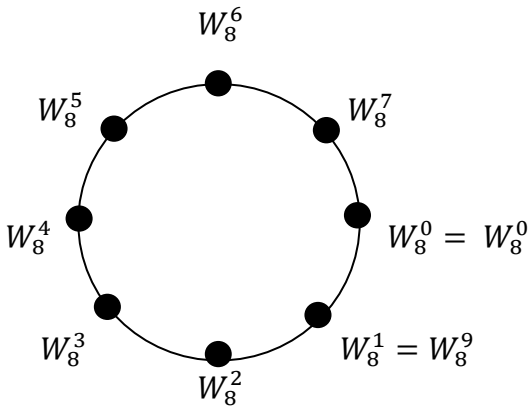


Figure 5: Properties of Twiddle factor

According to periodicity property the value of twiddle factors will repeat itself after one period and according to symmetry property half of the produced twiddle factors will symmetrical to the other half. Thus, utilising these two properties will help in generating less number of twiddle factors and reducing the computational complexity. For 128 and 64 point FFT only 32 and 16 twiddle factors need to be generated respectively.

Next section gives a brief introduction to the evolution of various FFT algorithm.

3.2 Evolution of FFT algorithm

This section reviews the evolution of FFT designs both from architecture and algorithm perspective. Thus, some of the most popular algorithms based on the designs are described in brief.

3.2.1 Overview of Fast Fourier Transform (FFT) Evolution

As explained in the section 3.1, direct computation of DFT requires an order of N^2 operations, where N is the FFT size. The Cooley-Turkey algorithm (divide and conquer approach, developed in 1965) got a breakthrough because it reduced the number of operations from $\mathbf{N^2}$ **to** $\mathbf{N\log_2 N}$. It is worth noting that hundreds of different version of the basic algorithm has developed over the years but still the basic one is the most popular FFT algorithm. In nearly all wireless communication systems, classical FFT algorithm is used in different forms.

Cooley- Turkey's algorithm is suited for any *composite* length i.e. $N = 2^n$. An approach was developed by Goods and Thomas to compute the DFT when the factors of the transform lengths are co-prime. This had the advantage of not producing twiddle factor (unlike in Cooley – Turkey algorithm) and thus providing a mono to multi dimensional mapping. This algorithm formed the base of all the research based on efficient FFT computation without using twiddle factors. But this algorithm has a drawback, that is, it requires efficient computable DFTs of co-prime lengths. At first, this seemed difficult to compute small lengths DFTs in less than N^2 operations. However, in *1968*, Rader came up with a way to map a DFT of length N (N is prime), into a circular convolution of length (N - 1) [4].

In 1976, Winograd came up with a new theoretical reduction in the complexity of multiplication. A point worth noting is that Winograd used just the converse of conventional method. Winograd algorithm used convolution to compute DFTs whereas conventional algorithm was computing convolutions by means of DFTs. Winograd mixed the results of Goods and Rader with his fast convolution algorithm to develop an efficient algorithm. The algorithm has low multiplicative complexity but it requires a large amount of adders.

But results of implementation of the Winograd algorithm were disappointing. Meanwhile, Cooley Turkey algorithm was developed further leading to the so called split radix algorithm (explained in detail in section 3.2.2). This algorithm had some attractive features such as low arithmetic complexity and relatively simple structure compared to Winograd. Thus, Winograd's algorithm failed to replace popular Cooley-Turkey type algorithm.

During 1980s, many FFT algorithms were proposed on the concept of real transforms. The main idea was to reduce computational complexity by avoiding complex operations such as multiplications. But as a result, they suffered from increase output level noise, i.e. high NSR (noise to signal ratio).

Meanwhile, Bruun proposed another important algorithm which had the advantage of reduced arithmetic complexity as well as lower NSR, compared to other real transform algorithms.

Bruun's algorithms still standouts from other real transform algorithms. But it was found that in practical [4] this algorithm also suffers from higher sensitivity to noise and therefore was not used for many years despite having advantages of low complexity and regular structure.

Split radix algorithm was proposed in 1984. The basic idea of this algorithm was to use different radix for even and odd part of the transform. This algorithm is well adapted to real and symmetric data and also has the lowest number of operations for composite FFTs. However, this structure results in complex designs when implemented on reconfigurable and programmable systems.

The main problem is that in the early stages of FFT development computational complexity was defined in terms of multiplication counts only. When theoretically good designs began to fail practically only then it was realised that number of additions, memory accesses and the cost of hardware are also important while considering a design as effective solution for FFT computation.

To summarize, almost all of the algorithms discussed above mainly use the main concept of *divide and conquer approach* i.e. to divide a large computation into small ones to reduce the computational complexity. Hence in our proposed algorithm we have also used the basic approach and modified to meet the system specifications described in Chapter 4.

3.2.2 Divide and Conquer Approach to computation of DFT

As explained in section 3.1, divide and conquer algorithm is based on the decomposition of an N length DFT into small length DFTs. Let us consider the computation of N point DFT, where N can be represented as a product of two integers, i.e.

$$N = LM \qquad \textbf{..Eq. 3.3}$$

Padding any sequence with zeros avoids the restriction of N being not a prime number. This will ensure the factorization of the above form.

Now let us consider the input sequence to be x(n), where $0 \leq n \leq N-1$. This sequence can be stored in 1-D array indexed by n or as 2-D array indexed by l (row index), where $0 \leq l \leq L-1$, and m (column index), where $0 \leq m \leq M-1$. Both forms of storage are shown in Figure 6 and 7. Thus the storage of the sequence x (n) will depend on the mapping of index n to the indexes (l, m).

The 2-D array storage of data sequences can be done in two ways:

- **Row wise storage:** In this arrangement, the elements of the input sequence will be stored row wise. The general mapping equation is given as:
  $$\mathbf{n = M\,l + m}$$
- **Column wise storage:** In this arrangement, the elements of the input sequence will be stored column wise. The mapping equation is given as:
  $$\mathbf{n = l + m\,L}$$

The above arrangements can be used similarly to store the computed DFT values. For the computed DFT values storage the mapping is from the index k to a pair of indices p and q, where $0 \leq p \leq L-1$ and $0 \leq q \leq M-1$. Thus, mapping $\mathbf{k = M\,p + q}$ will store output values row wise and using mapping $\mathbf{k = q\,L + p}$ will result in column wise storage.

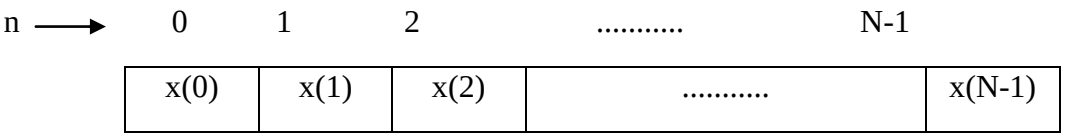


Figure 6: One dimensional array storage of input sequence

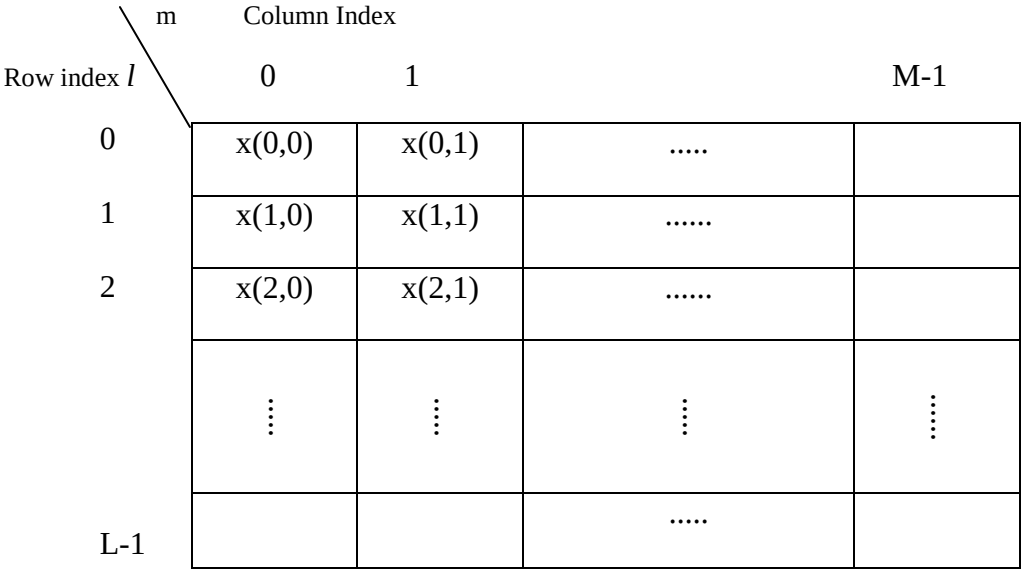


Figure 7: Two dimensional array storage of input sequence

Now assuming a column wise mapping for the input sequence x (n) and row wise mapping for the output sequence X (k), the DFT can be expressed as a double sum over the elements of the rectangular array multiplied by the corresponding phase factors [1]. Thus, rewriting Eq. (3.2) by substituting n = *l* + m L and k = M p + q, we have

$$X(k) = X(p,q) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[l,m]\, W_N^{(Mp+q)(l+mL)}$$

... Eq. 3.4

But,

$$W_N^{(Mp+q)(l+mL)} = W_N^{MLmp}\, W_N^{mLq}\, W_N^{Mpl}\, W_N^{lq}$$

From properties of twiddle factors, we can conclude that -

$$W_N^{Nmp} = 1,\ W_N^{mqL} = W_{N/L}^{mq} = W_M^{mq}, \text{and } W_N^{Mpl} = W_{N/M}^{pl} = W_L^{pl}$$

With these substitutions, Eq.3.4 can be represented as

$$X(p,q) = \sum_{l=0}^{L-1} \left\{ W_N^{lq} \left[\sum_{m=0}^{M-1} x(l,m)\, W_M^{mq} \right] \right\} W_L^{lp}$$

..... **Eq. 3.5**

The above expression involves the computation of DFT of length M and L. To make it clear, let us elaborate the above equation in three simple steps:

- In first step, we compute the M point DFTs for each rows *l.*

$$A(l,q) = \sum_{m=0}^{M-1} x(l,m)\, W_M^{mq}$$

- Secondly, we compute a new rectangular array B(*l*, q), which is basically multiplication of A(p, q) by the phase factors. It is defined as

$$B(l,q) = W_N^{lq}\, F(l,q)$$

- Finally, compute L point DFTs for each column q , of array B(*l*, q)

$$X(p,q) = \sum_{l=0}^{L-1} B(l,q)\, W_L^{lp}$$

To summarize, FFT computation involves following steps:

- Signal is stored column wise by using 2-D storage.
- M point DFT of each row is computed.
- The resulting array is multiplied by the corresponding phase factors W_N^{lq}
- L point DFT of each column is computed
- Output of the resulting array is read row wise.

To form the basis of further observations and proposals, a brief description of efficient FFT algorithms is presented in the next section. It should be noted that all the algorithms, described in the next section, are based on basic Cooley-Turkey algorithm as it is the most efficient and popular algorithm among all algorithms discussed above.

3.2.3 Efficient FFT algorithms based on Divide and Conquer Approach

This section will be describing three most efficient and widely used FFT algorithms. These are:

- Radix 2 FFT Algorithm
- Radix 4 FFT Algorithm
- Split Radix FFT Algorithm

3.2.3.1 Radix 2 FFT algorithm:

Divide and Conquer approach [5] is in particular very efficient when N is highly composite, that is when N can be factored as $N=r_1r_2r_3r_4\ldots\ldots r_a$, where $[r_b]$ are prime. In general, realistic signals are of the form $N = r^v$, which means $r_1= r_2 = r_3 = r_4 =\ldots\ldots r_a= r_v = r$. Here number ***r*** is called as the **Radix** of the FFT. Since the DFTs are of size r, therefore computation of the DFT has a regular pattern. Radix 2 algorithms are the simplest FFT algorithms. They partition a DFT into two – half length DFTs. The outputs of these reduced or rather shorter FFTs are reused to compute the DFT. Now based on the decimation [Appendix A], radix algorithm is divided into two approaches –

- Radix 2 –Decimation in Time (DIT)
- Radix 2 – Decimation in Frequency (DIF)

Radix 2 –Decimation in Time (DIT)

This approach splits the N point data sequence x(n) into two N/2 point data sequences x1(n) and x2(n), where x1(n) represents the even numbered samples and x2(n) represents the odd numbered samples of x(n).Since the decimation is done on x(n) , i.e. time domain , therefore it is called Decimation in Time Algorithm.

$$\left.\begin{aligned} x_1[n] &= x[2n] \text{ and} \\ x_2[n] &= x[2n+1] \qquad , 0 < n < (N/2)-1 \end{aligned}\right\} \ldots \textbf{Eq. (3.6)}$$

Thus Eq. (3.2) can be rewritten as –

$$X(k) = \sum_{n\ even} x(n)W_N^{kn} + \sum_{n\ odd} x(n)\,W_N^{kn}$$

$$X(k) = \sum_{i=0}^{\left(\frac{N}{2}\right)-1} x(2i)\,W_N^{2ik} + \sum_{i=0}^{\left(\frac{N}{2}\right)-1} x(2i+1)\,W_N^{(2i+1)k}$$

…Eq. (3.7)

Substituting $x_1[n]$ and $x_2[n]$ and applying $W_N^{2kn} = W_{N/2}^{kn}$ in Eq. (3.7)…. we get,

$$X(k) = \sum_{i=0}^{\left(\frac{N}{2}\right)-1} x1[n]\, W_{N/2}^{ik} + W_N^{kn} \sum_{i=0}^{\left(\frac{N}{2}\right)-1} x2[n]\; W_{N/2}^{ik}$$

$$= X_1(k) + W_N^{kn}\, X_2(k) \quad k = 0, 1, \ldots..N\text{-}1 \qquad \textbf{…Eq. (3.8)}$$

where $X_1(k)$ and $X_2(k)$ are the N/2 point DFTs of the sequences $x_1[n]$ and $x_2[n]$ respectively. Further by using periodic property of Twiddle factor we can say that –

$$X(k) = X_1(k) + W_N^{kn}\, X_2(k) \qquad k = 0,1,\ldots..(N/2)\text{-}1 \quad \textbf{…Eq. (3.9)}$$

$$X(k + N/2) = X_1(k) - W_N^{kn}\, X_2(k) \qquad \textbf{....Eq. (3.10)}$$

Now as we know that to compute N point DFT we require N^2 complex multiplications. Therefore to compute N/2 point DFT we require $(N/2)^2$ complex multiplications. This applies to both $X_1(k)$ and $X_2(k)$ and an additional N/2 complex multiplications to compute $\boldsymbol{W_N^{kn}\, X_2(k)}$.

Hence in total X (k) requires – $2 * (N/2)^2 + N/2 =$ **$N^2/2 + N/2$ complex multiplications**. Thus there is a reduction of about a factor of 2 from **N^2 to $N^2/2 + N/2$**. The decimation of the data sequence is repeated until the resulting sequences are reduced to 1 point sequence. Decimation is done for **$\log_2 N$ times** to compute the final DFT. Thus Radix 2 DIT algorithm reduces the total number of complex multiplications to **$(N/2)\log_2 N$** and reduces the complex additions to **$N \log_2 N$.**

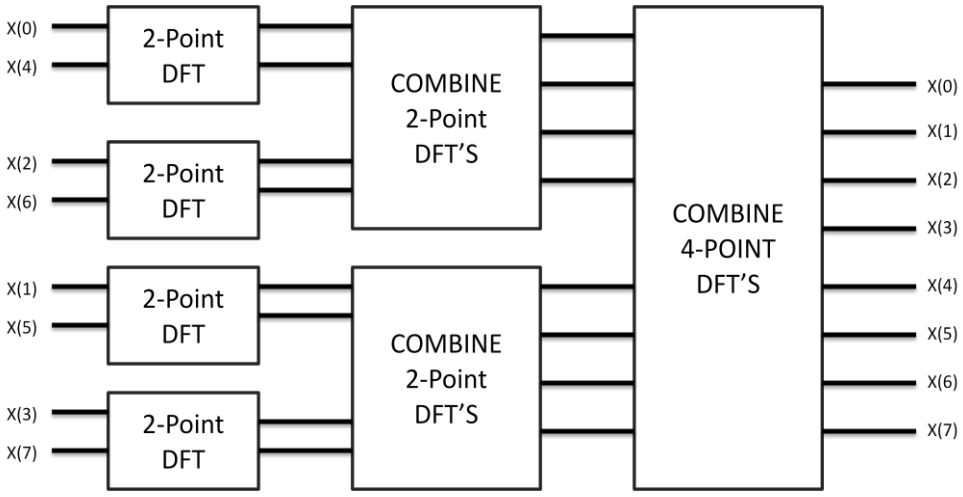


Figure 8: Block diagram of N = 8 Point DFT

Figure 8 shows an illustrative block diagram depicting the computation of an N=8 point DFT. As said earlier that decimation will be performed **$\log_2 N$ (i.e. $\log_2 8 = 3$)** times to compute the DFT, hence the block diagram has three stages, beginning with the computation of four 2 point DFTs, then two 4 point DFTs, and in last stage one 8 point DFT.

The Figure 9 below shows a basic butterfly diagram that is used to compute the N point DFT. Butterfly diagram is graphical representation of Eq. 3.9 and 3.10.

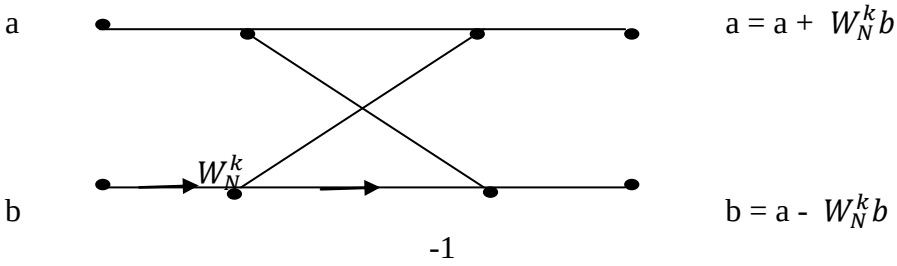


Figure 9: DIT Butterfly diagram of radix-2

Figure 10 represents the block diagram of Figure 8 showing basic butterfly computation**.** Since the output of one stage is the input of the next stage , therefore input are stored only once and that memory location is used throughout to store different pairs of input. Consequently we require **2N storage registers** in order to store the results.

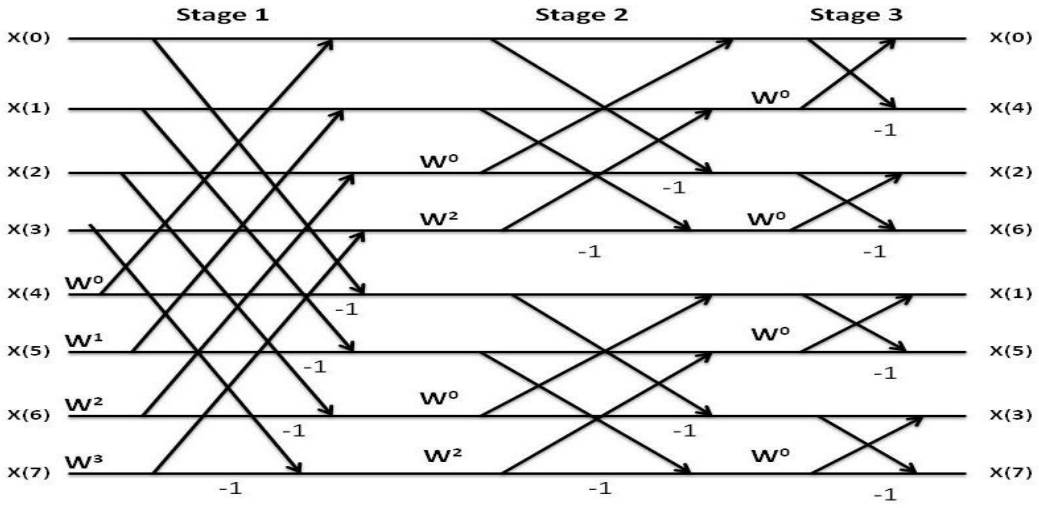


Figure 10: Eight Point DIT –FFT Algorithm

Few observations can be made from Figure 10 which will help us in efficiently implementing FFT algorithm –

- Each stage has the same number of butterflies (number of butterflies = **N/2**, N is number of points).
- The number of DFT groups per stage is equal to $\mathbf{N/2^{stage}}$.
- The difference between the upper and lower leg is equal to $\mathbf{2^{stage-1}}$.
- The number of butterflies in the group is equal to $\mathbf{2^{stage-1}}$.

[6]

Radix 2 –Decimation in Frequency (DIF)

This approach divides the N point data sequence into two N/2 point DFT. One DFT involves the first N/2 data input sequence and second DFT involves the rest half of the data sequence. Thus we have

$$X(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(n)\, W_N^{nk} + \sum_{n=\left(\frac{N}{2}\right)}^{N-1} x(n)\, W_N^{nk}$$

By using periodic property of Twiddle factor above equation can be rewritten as

$$X(k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(n)\, W_N^{nk} + (-1)^k \sum_{n=0}^{(N/2)-1} x(n+N/2)\, W_N^{nk}$$

$$(Since\ W_N^{Nk/2} = (-1)^k)$$

Now if instead of splitting x(n), we split (decimate) X(k) into even and odd number of samples, then the resulting equations will be

$$X(2k) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} [x(n) + x\left(n+\frac{N}{2}\right)] W_{N/2}^{nk}$$

… **Eq. (3.11)**

$$X(2k+1) = \sum_{n=0}^{\left(\frac{N}{2}\right)-1} \{[x(n) - x\left(n+\frac{N}{2}\right)] W_N^n\} W_{N/2}^{nk}$$

… **Eq. (3.12)**

The two algorithms differ only in decimation. Otherwise number of multiplications, additions and number of stages all remain the same for both of them.

- Complex Multiplications – $\mathbf{N^2/2 + N/2}$.
- Complex Additions - $\mathbf{N \log_2 N}$
- Number of Stages - $\mathbf{\log_2 N}$

Figure11 the basic butterfly diagram to compute DFT using Radix 2 DIF.

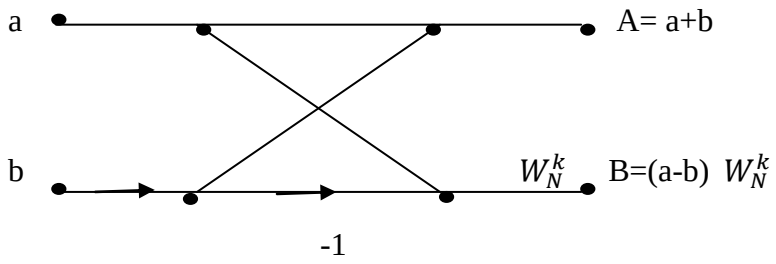


Figure 11: DIF Butterfly diagram of radix 2

Figure 12 illustrates the computation 8 point DFT using DIF FFT Algorithm. We can observe that the input sequence is in the normal order whereas the output is bit reversed.

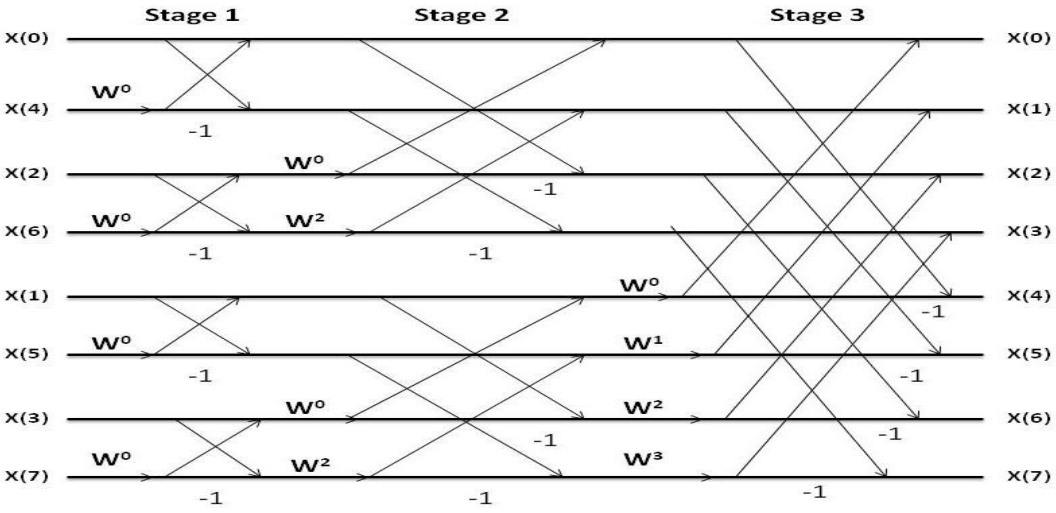


Figure 12: Eight point DIF FFT algorithm

3.2.3.2 Radix 4 FFT algorithm

Radix 4 algorithm is used to efficiently compute the DFT when the number of data points N is a power of 4 (i.e. $N = 4^v$). Although Radix 2 FFT algorithm can also be used to compute the same but Radix 4 FFT algorithm is much more efficient and involves less number of calculation overhead. Radix 4 also uses *divide and conquer approach* to divide N data sequences into N/4 data sequences. Thus the resulting four sub sequences are given as x (4n), x(4n+1), x(4n+2), x(4n+3) where n is equal to 0,1....,N/4 –1.

Thus Eq. (3.2) can be rewritten as

$$X(k) = \sum_{i=0}^{\left(\frac{N}{4}\right)-1} x(4i)\, W_N^{4ik} + \sum_{i=0}^{\left(\frac{N}{4}\right)-1} x(4i+1)\, W_N^{(4i+1)k} + \sum_{i=0}^{\left(\frac{N}{4}\right)-1} x(4i+2)\, W_N^{(4i+2)k} + \sum_{i=0}^{\left(\frac{N}{4}\right)-1} x(4i+3)\, W_N^{(4i+3)k}$$

… Eq. 3.13

Also,

$X(k) = DFT_{N/4}[x(4n)] + DFT_{N/4}[x(4n+1)] + DFT_{N/4}[x(4n+2)] + DFT_{N/4}[x(4n+3)]$

Eq.3.13 represents decimation in time (DIT) as the time samples are decimated. Figure 13 shows the basic butterfly diagram for the computation of Radix 4 FFT algorithm.

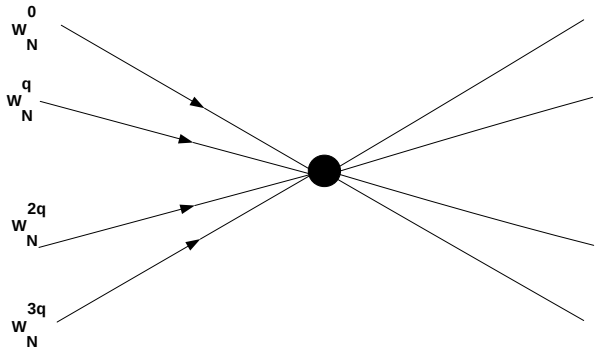


Figure 13: Butterfly diagram of radix-4

Radix 4 FFT operation counts:

- **Complex Multiplication : (3/8) $N\log_2 N$**
- **Complex Addition : $N\log_2 N$**
- **Number of Stages : $\log_4 n$**

As compared to Radix 2, Radix 4 requires 25% less multiplication to compute a DFT which makes it widely used FFT algorithm. A Radix -4 DIF FFT can be derived similarly to the radix 2 DIF FFT, by calculating all four groups of every fourth output frequency sample group of every fourth data sequence. The outputs ends in digit reversed order for DIF algorithm. Operation counts and number of stages remains same for both algorithms.

3.2.3.3 Split Radix FFT algorithm

Split radix was proposed by P. Duhamel and H. Hollmann (1986). Split radix is a variant of the basic FFT algorithm (Cooley-Turkey FFT). An inspection of the Figure 8 (Radix- 2 DIF FFT) indicates that even and odd numbered points of the DFT can be computed independently. Thus in order to reduce the number of computations, different and efficient algorithms can be applied to individual parts (even and odd) independently. This idea was exploited by Split Radix FFT (SRFFT) by using radix -2 and radix – 4decomposition in the same FFT algorithm.

The DFT is given by Eq.(3.2).The split radix algorithm expresses this equation in terms of three smaller summations. The resulting equations is given below –

$$X(k) = \underbrace{\sum_{i=0}^{\left(\frac{N}{2}\right)-1} x(2i)\, W_{N/2}^{4ik}}_{\text{Radix -2 (I)}} + \underbrace{\sum_{i=0}^{\left(\frac{N}{4}\right)-1} x(4i+1)\, W_{N/4}^{(4i+1)k} + \sum_{i=0}^{\left(\frac{N}{4}\right)-1} x(4i+3)\, W_{N/4}^{(4i+3)k}}_{\text{Radix -4 (II)}}$$

…Eq. (3.14)

Now these smaller summations are exactly DFTs of length N/2 and N/4. To compute the result these summations can be performed recursively and then recombined. For the odd part of Eq.(3.2), Radix–4 is selected since the odd numbered samples of the DFT require the pre – multiplication of the input sequence with the twiddle factors W_N^{nk} and we know that for these types of samples, radix - 4 algorithm has the largest multiplication free butterfly.
The basic butterfly for SRFFT algorithm is given below –

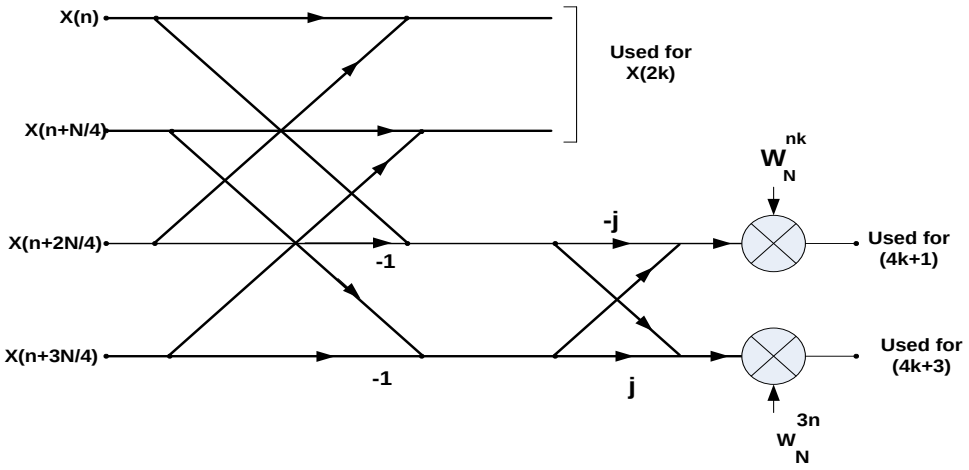


Figure 14: Butterfly for SRFFT Algorithm

Decomposing the half and quarter length DFTs will result in full split radix algorithm. The combination of different lengths FFTs in different part of the flow graph results in an irregular structure. However in [7] it has been shown how to adjust the computation so that the data retains the radix-2 bit reverse order. Analogously a DIF (decimation in frequency) split radix FFT can also be derived.

In split- radix, multiplicative complexity is about 75% less than radix-2 FFT. It is also better than radix-4 FFT or any other higher power-of-two radix as well. The addition used within the twiddle factor is also reduced to a great extent. Addition operations in split radix remains same as compared to radix-2 and radix- 4 FFT because of the fact that underlying butterfly tree remains the same in all composite input algorithms.

Thus in general,

- **Complex Multiplication:** **$(N/3)\log_2 N$**
- **Complex Additions:** **$N\log_2 N$**

Chapter 4. Different Hardware Architectures and related work

4.1 Introduction

In the last three decades different possible architectures have been proposed based on the algorithms discussed. A brief overview of some important hardware implementations and related work is presented in this section.

4.2 Direct FFT Array or Parallel FFT

Direct path array FFT signifies parallel FFT. In this architecture FFT is implemented such that the input signals follows a direct path from input to output. The advantage is that the delay from first input to first output (latency) is low. But it suffers from huge area requirement. Therefore, direct path array FFT is implemented for small, dedicated accelerator blocks in critical applications to meet critical timing requirements.
This architecture usually requires one *radix* - 4 (or higher radix) or 2 or more radix 2 butterflies. In addition it also requires three multi port buffers of size *N*. The three ports are required to parallelize the input data, process the data and output the data.
For $N > 256$ more than one *radix* - 4 butterfly units is needed to achieve real time processing. With every additional butterfly number of overhead also increases. One of the main disadvantage of radix - 4 or higher radix algorithm is that they restrict the transform size *N* to powers of four or higher, thus forcing serious limitation on the flexibility or re-usability of the core. Figure 15 below shows the block diagram of the Array or parallel architecture.

Previous Work done:

In [8] the input data is split into four parallel paths. Each path has its own RAM to read real and imaginary part of complex data parallely. Each path also contains an arithmetic unit consisting of multiplier, three adders/subtractors and various holding registers. Each path acts as a separate radix 4 processor. This work suffered from huge area requirement and also use of Radix – 4 restricted the transform size to powers of 4.

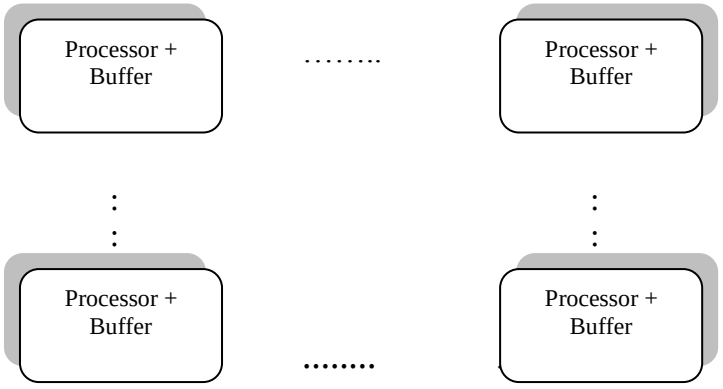


Figure 15: Block diagram of Parallel Array architecture

4.3 Pipelined FFT

The pipelined FFT architecture typically falls into one of the two following categories.

- Single-Path Delay Feedback (SDF) Architecture
- Multi-Path Delay Commutator (MDC)Architecture

The pipelined FFT architecture uses $\log_r N$ butterflies, connected in a serial order, where *r* is the radix used for the FFT computation. The data arrives at the input of leftmost butterfly or buffer and final results will come out of the rightmost butterfly.

The memory requirement for this architecture is up to *N* - 1spaces, and the latency is of the order of *N, where N is number of inputs*. The reason for such a large memory requirement is that there is no parallelisation of the data sequence at the input and all the processing is done serially.

Adding one more stage increases the size of the transform by a power equal to the radix, r. In the context of a reconfigurable real-time processing core, the pipelined architecture is more suitable [4]. Because of its serial processing and regular structure it is a preferred choice among the designers.

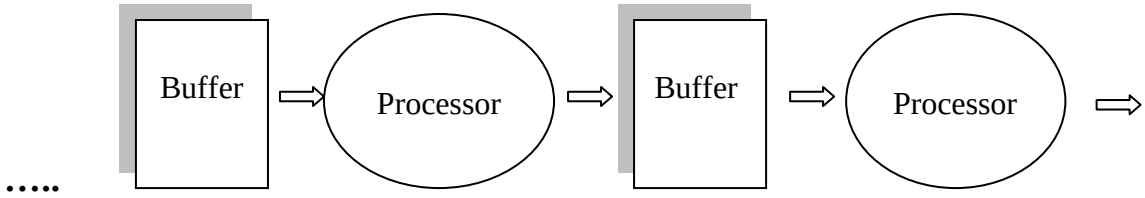


Figure 16: Pipeline Architecture Block Diagram

Single Path Delay Feedback(SDF) Architecture:

A general pipelined radix-2 based SDF architecture is given in Figure17. BF stands for butterfly and rectangular boxes containing the numbers represents memory required to store that many samples. It uses feedback techniques to reduce memory requirements.

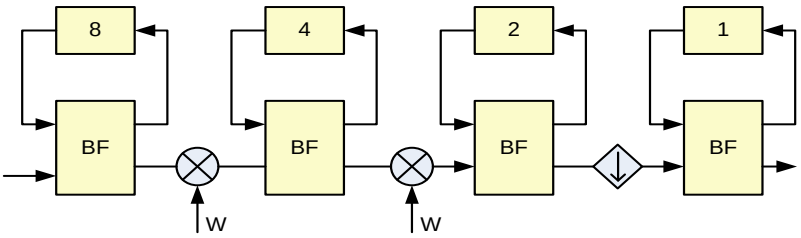


Figure 17: Single Path Delay Feedback (SDF) Architecture

Multi- Path Delay Commutator (MDC) Architecture:

Figure 18 shows the general pipelined radix-2 based MDC architecture. The naming conventions are same as for SDF architecture. One can easily see in the figure that MDC has more memory requirement than SDF (Note: N=16 in both cases). MDC architecture has the advantage of being fast as compared to SDF architecture.

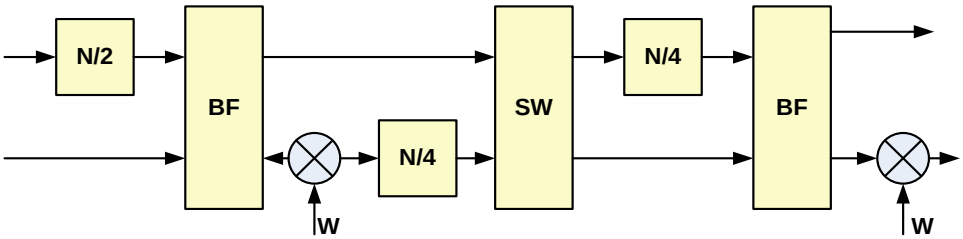


Figure 18: Multi-Path Delay Commutator(MDC) Architecture

Various Pipeline FFT Architecture and previous work done:

R2MDC [9]: *Radix-2 Multi-path Delay Commutator* was the most basic approach for pipeline implementation of radix-2 FFT algorithm. In [9] this architecture has been used. The input sequence was divided into two parallel data streams. The data elements entering the butterfly were at correct distance from each other and were scheduled properly by delays. However, the hardware resources were not efficiently utilised as they only operated for 1/r of the time, where r is the radix of the FFT. This architecture requires-

- $\log_2 N$ - 2 multipliers,
- $\log_2 N$ radix-2 butterflies and
- 3/2N - 2 registers (delay elements).

R2SDF [10]: *Radix-2 Single-path Delay Feedback* uses feedback shift registers to store the result of one butterfly and thus reduces the memory requirement. In every stage the input to the multiplier is a single data stream. The proposed architecture was slow and resulted in increased overheads.
This architecture requires-

- $\log_2 N$ – 2 multipliers
- $\log_2 N$ radix-2 butterflies and
- N-1 registers (delay elements).

R4SDF: *Radix-4 Single-path Delay Feedback* was proposed as a radix-4 version of R2SDF, employing CORDIC iterations [4][11]. This architecture effectively utilises the multipliers by storing 3 out of 4 butterfly outputs (Radix – 4).Thus the utilisation of multipliers is increased to 75% as compared to 50% of R2SDF. This architecture though, achieves reduction in multipliers but it complicates the control of the architecture. This results in a dip of 25% in the utilisation of radix-4 butterfly.

The requirement of this architecture is

- log $_2N$ - 1 multipliers,
- log $_2N$ full radix-4 butterflies and
- Storage of size $N - 1$.

R4MDC : *Radix-4 Multi-path Delay Commutator* is a radix-4 version of R2MDC.This architecture was implemented in the initial VLSI implementation of pipeline FFT processor [12] and massive wafer scale integration [5]. However, it suffers from low(25%) utilisation of all components. This low utilisation can only be compensated in some special applications where four FFTs are being processed simultaneously. Hence it is a suitable design for MIMO applications. But because of its high memory requirement, it is generally not preferred by the designers. Using this architecture, one needs-

- 3 log $_4N$ multipliers,
- log $_4N$ full radix-4 butterflies and
- *5/2N* - 4 registers are required.

R4SDC [13]: Radix-4 Single-path Delay commutator uses a modified radix-4 algorithm. To achieve higher utilisation of multipliers, this architecture uses programmable ¼ radix-4 butterflies. A multiplexed version of this architecture is used in [13] which further reduces memory requirement to ***2N*** – 2. But the butterfly and delay-commutator become relatively complicated due to the programmability requirement. The hardware requirement of this architecture is-

- (3/4) log $_4N$ multipliers,
- log $_4N$ full radix-4 butterflies and
- 2N-2 registers.

R2^2 SDF and R2^2 MDC - Radix 2^2 uses a 3D linear index mapping. It decomposes the DFT and cascades the "twiddle factor" into the next step of decomposition. This in turn helps this architecture to exploit the exceptional values in the multiplication of twiddle factor, just before the butterflies are constructed. The Radix 2^2 algorithm is characterized with the trait that it has same multiplicative complexity as radix-4, but still retains the radix -2 butterfly structures.

MRMDF [14]: *Mixed-Radix Multipath Delay Feedback* architecture can provide higher throughput rate with minimal hardware cost by combining the features of MDC and SDF [14]. Three-step radix-8 FFT algorithm [2] is used to increase the throughput as much as 4 times as compared to R2SDF.
In [15], pipelining algorithm is implemented with a novel cascade decomposition of the twiddle factors. In this paper it was shown that by cascading the decomposition of twiddle factors, new form of FFT with high spatial regularity can be obtained.
In [2], MRDS architecture is chosen in order to deal with 1-4 simultaneous data sequences. In this architecture delay feedback scheme was used to minimize memory requirement by reordering the input data and the intermediate results of each module.

4.4 Memory Based FFT

Another popular FFT architecture is memory based architecture. These architectures are more area efficient but require higher clock frequency. There are around two types of Memory based architectures: *Single Memory* and *Dual Memory*. These architectures basically require a centralized memory block to store input or intermediate data and a control unit to handle memory accesses and data flow directions. Single butterfly based FFT architectures are a perfect example of memory based FFT computation.

To increase the efficiency of memory based architecture, following techniques can be used:

- **Data partitioning** – rearrangement of data flow to get in place output (bit reversed output).
- **Memory partitioning** – signifies dividing a single memory block in to many smaller blocks. These partitioning techniques are based on changes in address generator outputs which are used at central controller to maintain proper signal flow. Memory partitions also helps in increasing the memory access and width as it can be done easily on independent sub-blocks compared to single, larger block of memory.

Figure 19: Single and Dual memory Architecture

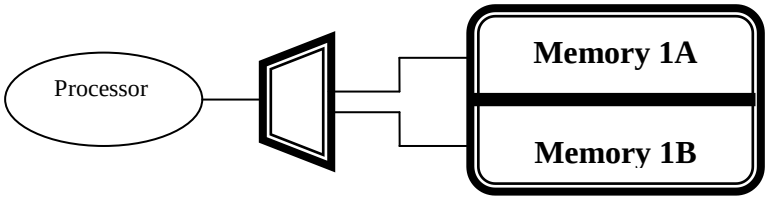


Figure 20: Memory partitioning technique

- **Using cache memory** (to reduce power and latency) - Due to their largely memory dependent structure, memory based architectures require higher clock rates to achieve higher throughput. As a solution, cache memory can be used to reduce the both power consumption and latency or computational delay involved in computation.

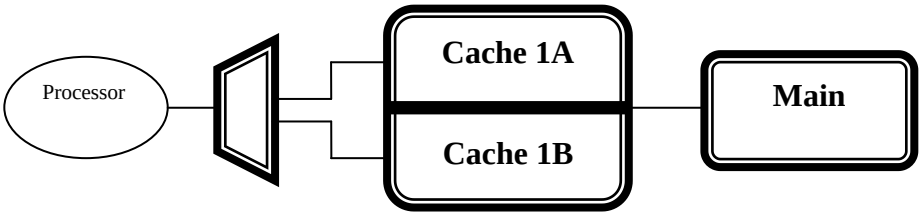


Figure 21: Cache memory technique

Related work:

In [16], cached memory architecture was implemented. The whole idea was to isolate the main memory from the high speed portion of the processor and it was implemented by defining a portion of the cached FFT algorithm, called *epoch*, where all N data words are loaded into a cache, processed, and written back to the main memory once. . It allows the data to be accessed from cache memory (e.g. register file) which are much faster than a slower data cache or DRAM. The architecture resulted in low power consumption and minimal silicon area but the use of cache memory makes it an expensive solution.

Summary

In this section we have explained various hardware structures for FFT implementation and also work related to them. It has been concluded that pipeline architectures are preferred because of their advantages. Hence, we also intend to use parallel pipelining scheme for the implementation of our proposed approach.

In the next section we will formulate the algorithm to be used for FFT/IFFT computation in MIMO OFDM systems.

Chapter 5. Algorithmic Formulation for MIMO OFDM systems

In this chapter we will discuss the formulation of divide and conquer algorithm for MIMO OFDM systems. It has been shown in previous chapters that divide and conquer approach is the most suitable algorithm present today and almost all algorithms are based on this approach only. Hence we are formulating this algorithm keeping in mind the specifications of MIMO OFDM system.

MIMO OFDM system requires 128 or 64 point FFT to be computed within the limited time frame. Now from Eq.3.3, we have N = L M. Let us consider L=16 and M=8 to formulate 128 point FFT. Then n = l + m16.

From Eq.3.5, we have

$$X(p,q) = \sum_{l=0}^{L-1} \left\{ W_N^{lq} \left[\sum_{m=0}^{M-1} x(l,m)\, W_M^{mq} \right] \right\} W_L^{lp}$$

Substituting the value of M and L, we have

$$X(p,q) = \sum_{l=0}^{15} \left\{ W_{128}^{lq} \left[\sum_{m=0}^{7} x(l + m16)\, W_8^{mq} \right] \right\} W_{16}^{lp} \qquad \textbf{....Eq. 5.1}$$

Equation (5.1) suggests that the 128 point DFT can be computed by first computing 8 point FFT, then multiplying them by 128 complex inter dimensional constant multiplications. However, since $l \in \{0, 1, \ldots 15\}$ and $m \in \{0, 1, \ldots 7\}$ and $W_N^0 = 1$, the number of non trivial complex multiplications is only 105.This is followed by computing 16 point FFT on the intermediate data to get the final result.

Every 16th sample of the incoming data sequence forms the input of an 8 point FFT. The resultant data of 8 point FFT is then multiplied by 8 twiddle factors (W_{128}^{lq}). Effectively, resultant data undergoes only 7 multiplications as the zeroth term of the 8 point FFT gets multiplied by 1. Sixteen such computations are needed to generate a complete set of 128 intermediate data. This intermediate data will undergo 16 point FFT with the appropriate data reordering (every eighth intermediate data). Eight such 16 point FFT are required to compute 128 point DFT. Proper reshuffling of the resultant data from the 16 point FFT will constitute the final output of 128 point FFT.

The first 8 point FFT has been realised by using the conventional DIT radix-2 algorithm. The main advantage with this algorithm is that one does not need to use any explicit multiplication operation. As shown in Figure 10, stage 1 is multiplied by 1 (since $W_N^0 = 1$) and stage 2 is multiplied by j (since $W_N^2 = -j$). Thus multiplication by these values is merely addition/subtraction operations with the appropriate reordering of the data. In third stage also, the multiplications of the constants can be easily realised by shift and add operation as in this stage the constants are addition/subtraction operation followed by the multiplication of $1/\sqrt{2}$ (since $W_N^3 = -(1+j)/\sqrt{2}$). Hence, an 8 point FFT can be computed without using any true digital multiplier [3]. A cost effective and low power 128 point FFT can be realised by using this architecture.

16 point FFT can be easily realised by again using divide and conquer approach and breaking N = 16 into DFTs of length 2 and 8. Thus M = 2 and L = 8. Following the same procedure as described above, we can compute 16 point FFT with reduced hardware complexity. It should be noted that to compute 16 point FFT, 8 point FFT is used in the last stage which is again realised in the similar manner as described above. The detailed description of each block is given in the next Chapter.

To compute 64 point FFT, we have considered L = M = 8. Thus Eq. 3.3 will be rewritten as

$$X(p,q) = \sum_{l=0}^{7} \left\{ W_{64}^{lq} \left[\sum_{m=0}^{7} x(l + m8)\, W_8^{mq} \right] \right\} W_8^{lp}$$

…. Eq. (5.2)

Equation (5.2) suggests that 64 point FFT can be computed by first computing 8 point FFT (with every 8^{th} input sample being the input data for 8 point FFT) , followed by multiplication with the twiddle factors W_{64}^{lq}. Similar to 128 point FFT, number of non trivial multiplication for 64 point FFT is only 49. After the multiplication with the twiddle factors, the intermediate data again undergoes 8 point FFT. The proper reshuffling of the output data of last 8 point FFT generates the final output of the 64 point FFT. The multiplications of the constants in 8 point FFT are realised in the same manner as described above i.e. by using shift and add operations.

During the transmission phase, inverse fast Fourier transform (IFFT) of the input sequence need to be performed. The IFFT of the input sequence can be performed by just swapping the real and imaginary part of the input data and the output data. The first swapping takes place in input sequence. The swapped data is processed and output is collected. The real and imaginary parts of output data is once again swapped to get final results of IFFT.

The main advantage of this method is that both FFT and IFFT can be performed by using single hardware without introducing any change in any of the internal coefficients.

In the next chapter, we will propose a new architecture, for the hardware implementation of FFT processor for MIMO OFDM systems, based on the above algorithm. The new architecture will also try to justify the reason for choosing the specific values of L and M while decomposing 128 point and 64 point FFT.

Chapter 6. Architecture of the proposed design and its implementation

6.1 Introduction

In this chapter we will propose a new adaptive reconfigurable architecture for the hardware implementation of FFT/IFFT processor for MIMO OFDM systems which will be able to compute a 128 point FFT or two 64 point FFT or one 64 point FFT using the same hardware. For 128 point FFT we are using (8 x 16) architecture and for 64 point FFT we are using (8 x 8) architecture. The proposed design is implemented using various parallel data paths (4, 8, and 16). The parallel data path increases the computational efficiency of the system and also makes the design much faster. The proposed architecture reduces hardware complexity by using an efficient multiplier unit, which will be discussed in detail in the following sections.

Later in this chapter, we will discuss the implementation of the design and define the low power techniques (such as clock gating) used and optimisations done to increase the efficiency of the design.

6.2 Block Diagram

The block diagram of the proposed design is depicted in Figure 22. The block diagram shows the generalised architecture which can be further extended to different data paths (4, 8 and 16 data paths). We will briefly describe the various modules of the architecture based on 4 data paths and at appropriate points difference between the modules for different data paths will be highlighted.

The processor has five functional units: Input buffer, Module 1, Twiddle unit, Reconfigurable block and an Output buffer unit. Two control signals, namely *count* and *fft_128*, act as master control for the whole design. The large number of global wires resulting from the multiplexing of the complex data has been dealt strategically in the current architecture. In the next subsections, each block is explained in detail.

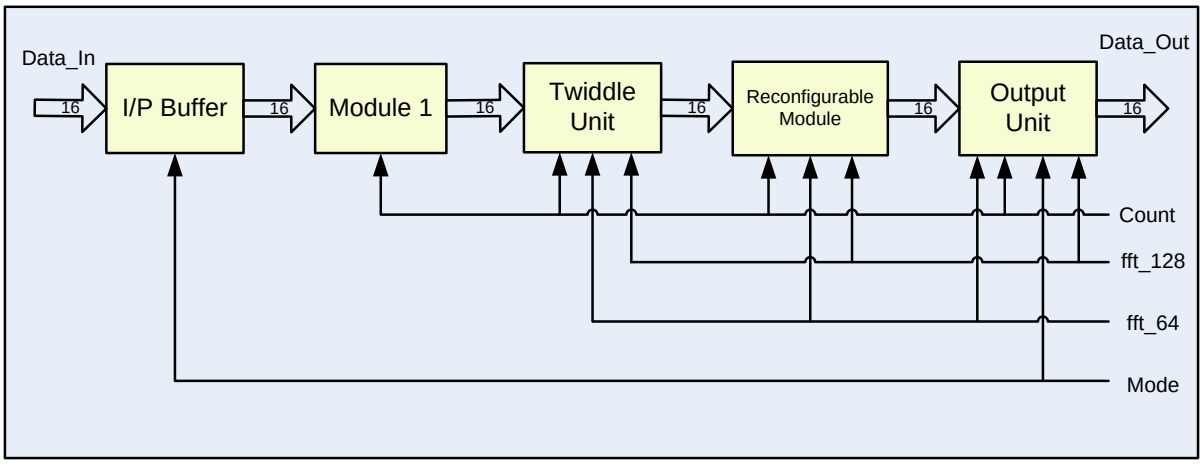


Figure 22: Block diagram of proposed design

6.2.1 Input Buffer

The input buffer consists of an input register bank that can store complex data. The table given below describes the input buffer size requirement for different data paths.

Number of Data paths	Input buffer			
	No. of register files	Size	Width	
			Real	Imaginary
4	4	28	16 bit	16 bit
8	8	16	16 bit	16 bit
16	16	8	16 bit	16 bit

Table 2: Number of Input buffer and its size for various data paths

For 4 data path architecture, the input buffer consists of 4 register files and each register file can store 28 complex data.

The input buffer has two single bit signals, *mode* and *data_in*. Input unit starts its operation based on the logic state of *data_in* signal. Logic state 1 of *data_in* signal indicates the presence of valid input stream. The input unit will only start to store the input data if this signal is asserted.

The logic state of *mode* signal indicates the mode of FFT/IFFT processor. Logic state 0 indicates FFT mode and logic 1 states IFFT mode. The logic state of *mode* signal decides the swapping of the input data. If the processor is in IFFT mode i.e. *mode* = 1, the real and imaginary parts of input samples are swapped and the swapped data is entered in the register bank whereas if *mode* is in logic 0 state i.e. FFT mode, the incoming input samples are directly fed to the input register bank and no swapping takes place.

When the *data_in* signal arrives, first four samples of input data are inserted parallely at every clock cycle. The input data enters only at the 28^{th} location of each register file. In successive clock cycles the input data is shifted downwards by 1 location. Thus if the input data stored in input register files is having index *k*, then with the arrival of the next set of input data, on successive clock cycles, the data stored will be shifted to the $k\text{-}1^{th}$ position.

For 4 data path architecture, 28 cycles are required to completely fill the Input buffer. When the Input buffer i.e. each register file, is completely full, it outputs the appropriate data to the next module i.e. 8 point FFT. By appropriate data we mean a data octet consisting of every 16^{th} sample of the input sequence for 128 point FFT and every 8^{th} sample for 64 point FFT.

To output the data, each register file is equipped with 8 hard wired (16 bit) outputs and these hard-wired outputs are connected to the corresponding register file position index 4*a*, where $a \in \{0, 1, \ldots, 7\}$. This is the reason why we choose the size of Input buffer as 28. In the next cycle the same procedure will keep on repeating until all the 128 samples are processed through by the Input buffer.

For 64 point FFT, the first two registers files will hold the input samples of first 64 point FFT and the other two registers files will hold the input samples of the second 64 point FFT. Thus for 64 point FFT, the design *adapts* itself and uses a 2 data path structure.

For 64 point FFT, the output of the input buffer has to be a data octet constituting every 8 sample of the input sequence (see eq. 5.2). The design adapts in such a way that for 64 point FFT we need not change the hard wired connection. The hard-wired outputs will get self aligned to output the appropriate data.

By using data shifting technique, we have avoided the use of a large multiplexing scheme which would otherwise be needed to multiplex the input data to the 8 point FFT. The input buffer asserts a CE signal once the input buffer is full. This signal is used to enable the counter *count (5 bit)* which controls all the other modules. The Figure 23 given below shows the block diagram of Input buffer.

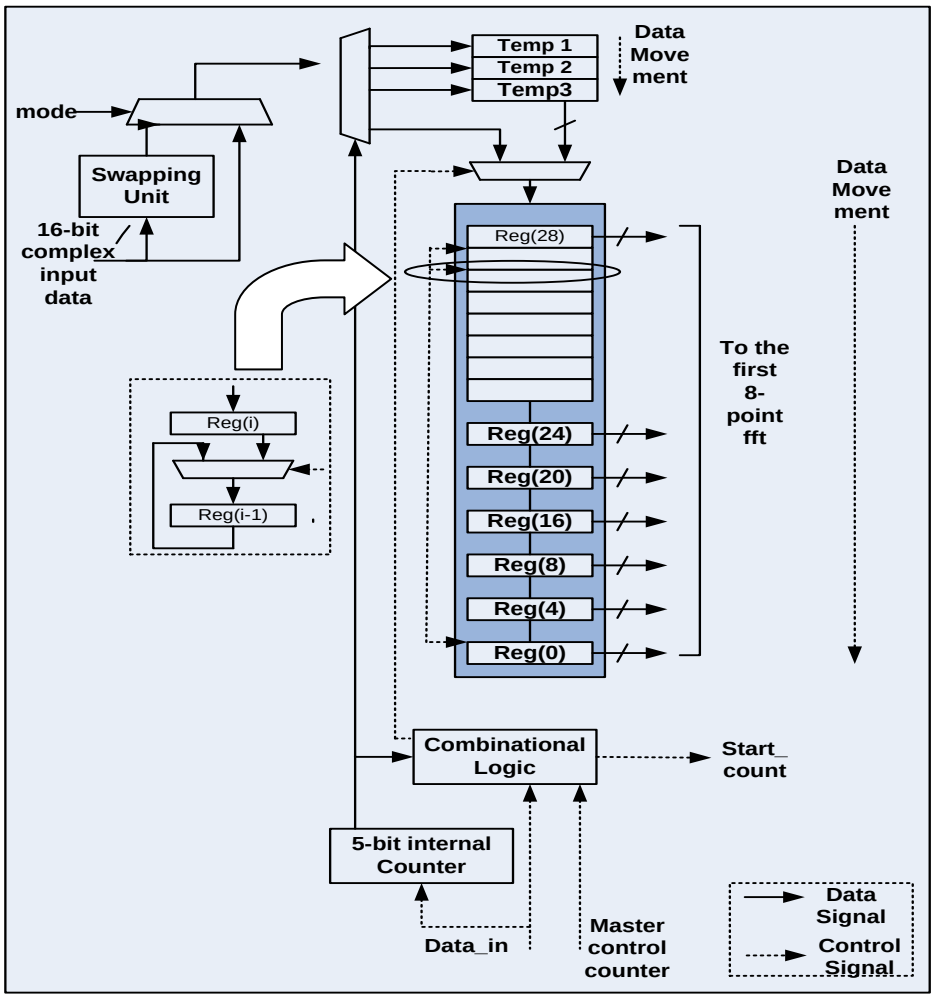


Figure 23: Block diagram of Input unit

6.2.2 Module 1

Module 1 is used to implement the 8 point FFT by using conventional radix-2 DIT eight point FFT algorithm (explained in 3.2.3.1). As discussed in Chapter 5, this scheme results in computing 8 point FFT without using any digital multiplier. Since for 4 data path structure, each of the four register files will output a data octet therefore we need four of such modules

to process complete set of input samples. Table 3 below shows the requirement of Module 1 for different data paths.

Number of Data paths	Number of 8 point FFTs required	Word length
4	4	16
8	8	16
16	16	16

Table 3: Number of Module 1 required for different data paths

As shown in Figure 10, there are three stages of computation. Thus an 8 point FFT needs three cycles to compute the result. Module 1 is controlled by control signal *count.*

When *count* is equal to 1 then Module 1 takes the input from the input buffer and starts to process it. After 3 cycles i.e. at *count* = 4, each 8 point FFT gives its output. Since 32 samples are processed at a time by four 8 points FFT therefore after four cycles of *count* assertion, Module 1 stops taking the data. Since 8 point FFT takes three cycles to compute, so the output of last 32 samples will be available only after three clock cycles i.e. when *count* = 7. The output of all the 8 point FFTs is fed to the twiddle unit for multiplication with twiddle factors.

6.2.3 Twiddle unit

For 128 point FFT only 105 nontrivial inter-dimensional constants or twiddle factors ($W_{128}^{lq}, l \in \{0,1,\dots 15\}$and $q \in \{0,1,\dots .7\}$) and similarly for 64 point FFT 49 such inter-dimensional constants are required ($W_{64}^{lq}, l, q \in \{0,1,\dots .7\}$). These constants need to be multiplied with the output of the 8 point FFTs.

The corresponding block diagram of the twiddle unit is given in Figure 24. Table 4 below shows the inter-dimensional constants and Figure 25 shows how the multiplication takes place. In figure 25, X1(0,1,...7), X2(0,1,...7), X3(0,1,...7) and X4(0,1,...7) denotes the output of each 8 point FFT.

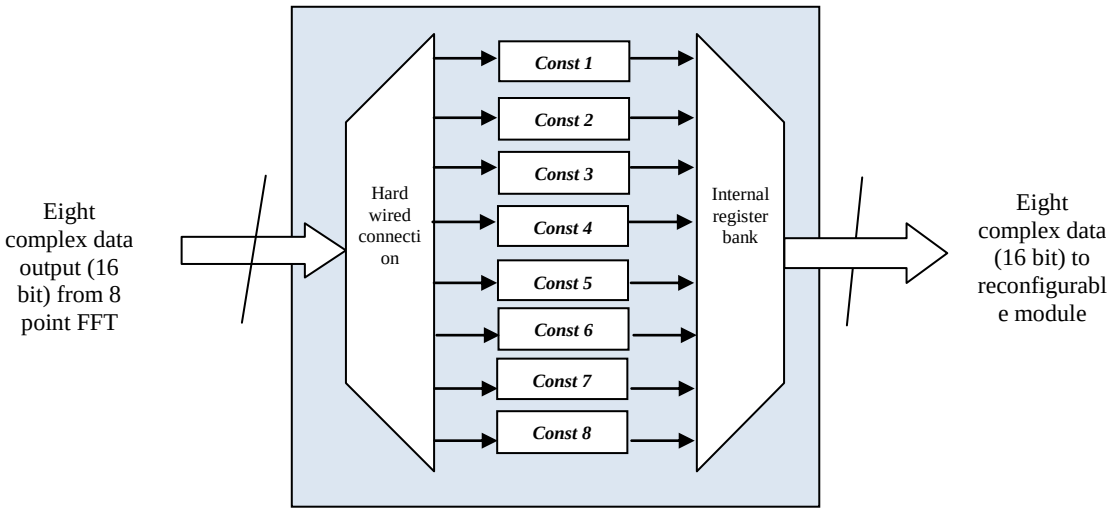


Figure 24: Block diagram of Twiddle unit

l \ q	0	1	2	3	4	5	6	7
0	W^0	W^0	W^0	W^0	W^0	W^0	W^0	W^0
1	W^0	W^1	W^2	W^3	W^4	W^5	W^6	W^7
2	W^0	W^2	W^4	W^6	W^8	W^{10}	W^{12}	W^{14}
3	W^0	W^3	W^6	W^9	W^{12}	W^{15}	W^{18}	W^{21}
4	W^0	W^4	W^8	W^{12}	W^{16}	W^{20}	W^{24}	W^{28}
5	W^0	W^5	W^{10}	W^{15}	W^{20}	W^{25}	W^{30}	W^{35}
6	W^0	W^6	W^{12}	W^{18}	W^{24}	W^{30}	W^{36}	W^{42}
7	W^0	W^7	W^{14}	W^{21}	W^{28}	W^{35}	W^{42}	W^{49}
8	W^0	W^8	W^{16}	W^{24}	W^{32}	W^{40}	W^{48}	W^{56}
9	W^0	W^9	W^{18}	W^{27}	W^{36}	W^{45}	W^{54}	W^{63}
10	W^0	W^{10}	W^{20}	W^{30}	W^{40}	W^{50}	W^{60}	W^{70}
11	W^0	W^{11}	W^{22}	W^{33}	W^{44}	W^{55}	W^{66}	W^{77}
12	W^0	W^{12}	W^{24}	W^{36}	W^{48}	W^{60}	W^{72}	W^{84}
13	W^0	W^{13}	W^{26}	W^{39}	W^{52}	W^{65}	W^{78}	W^{91}
14	W^0	W^{14}	W^{28}	W^{42}	W^{56}	W^{70}	W^{84}	W^{98}
15	W^0	W^{15}	W^{30}	W^{45}	W^{60}	W^{75}	W^{90}	W^{105}

Table 4: Inter-Dimensional constants or twiddle factors

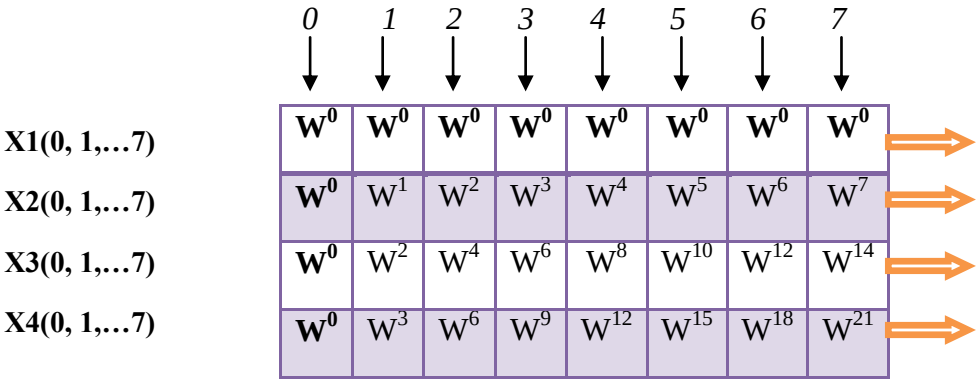


Figure 25: Multiplication of output of 8 point FFT with twiddle factors

Table 4 shows the non trivial inter dimensional constants. From Eq.3.2, we know that

$$W_N^{kn} = e^{-(j\frac{2\pi kn}{N})}$$

Expanding above equation in terms of sine and cosine function, we get

$$W_N^{lq} = e^{-\left(j\frac{2\pi lq}{N}\right)} = \cos\left(\frac{2\pi lq}{\mathrm{N}}\right) - \mathrm{j}\sin\left(\frac{2\pi lq}{\mathrm{N}}\right)$$

From the above equation, we can get the values of each twiddle factor to be multiplied with the complex data. Applying properties of twiddle factors, following observations were made:

- Only 31 values are unique (given in Table 7)
- Other constants can be derived by appropriate swapping of the real and imaginary part of these values and using the appropriate sign.
- These 31 values also cover the twiddle factors required to perform 64 point FFT as $W_N^{k/2} = W_{N/2}^{k}$.
- The sign convention for 128 point FFT is:

Twiddle factor	Real	Imaginary
$\boldsymbol{W_N^0}$ **to** $\boldsymbol{W_N^{31}}$	+	+
$\boldsymbol{W_N^{32}}$	Swap real and imaginary parts	
$\boldsymbol{W_N^{33}}$ **to** $\boldsymbol{W_N^{63}}$	+	-
$\boldsymbol{W_N^{64}}$	Swap real and imaginary parts	
$\boldsymbol{W_N^{65}}$ **to** $\boldsymbol{W_N^{95}}$	-	+
$\boldsymbol{W_N^{96}}$	Swap real and imaginary parts	
$\boldsymbol{W_N^{97}}$ **to** $\boldsymbol{W_N^{105}}$	-	-

Table 5: Sign convention for 128 point FFT

- The sign convention for 64 point FFT is :

Twiddle factor	Real	Imaginary
$\boldsymbol{W_N^0}$ **to** $\boldsymbol{W_N^{31}}$	+	+
$\boldsymbol{W_N^{32}}$	Swap real and imaginary parts	
$\boldsymbol{W_N^{33}}$ **to** $\boldsymbol{W_N^{63}}$	+	-
$\boldsymbol{W_N^{64}}$	Swap real and imaginary parts	

Table 6: Sign convention for 64 point FFT

Thus one needs to store only 31 values instead of 105 values to carry out the multiplication operation. This scheme offers a significantly less storage space as compared to classical DIT FFT approach.

Now we will discuss the multiplier scheme used to generate 31 twiddle factors. Generation of twiddle factors from conventional method will result in requirement of huge silicon area and high power consumption. A better approach can be proposed by decomposing these constants as addition/subtraction based on powers of 2. Thus, these constants can be generated by using only shift and add operations.

For example, consider the multiplication of input *x* with the constant 5 i.e. 0.970031. The constant 0.970031 can be expressed in powers of 2 as $1-2^{-5}+2^{-9}-2^{-11}-2^{-13}-2^{-14}$ (Appendix B.1). Thus effectively multiplication of input *x* with this expression will result in series of addition/subtraction of right shifted values of input *x*. The Figure 26 explains this feature. Since the numbers of shifts are fixed, the shifters shown can be easily implemented by using hard wired connection.

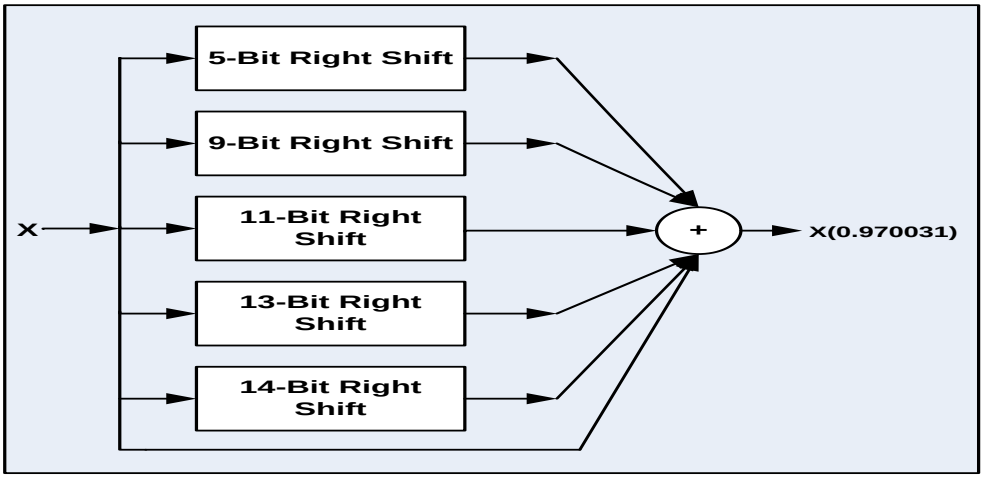


Figure 26: Circuit diagram of Multiplier unit

Name of constant	Value of constant	Values in terms of powers of 2
Const 1	0.998534	$1-2^{-10}-2^{-12}$
Const 2	0.995732	$1-2^{-8}-2^{-10}+2^{-14}$
Const 3	0.989678	$1-2^{-7}-2^{-9}-2^{-10}-2^{-14}$
Const 4	0.980335	$1-2^{-6}-2^{-8}+2^{-12}+2^{-14}$
Const 5	0.970031	$1-2-^{5}+2^{-9}-2^{-11}-2^{-13}-2^{-14}$
Const 6	0.965789	$1-2^{-5}-2^{-7}-2^{-8}-2^{-14}$
Const 7	0.941423	$1-2^{-5}-2^{-6}-2^{-7}-2^{-8}+2^{-13}$
Const 8	0.923689	$1-2^{-4}-2^{-7}-2^{-8}-2^{-9}$
Const 9	0.903956	$1-2^{-4}-2^{-5}-2^{-9}-2^{-11}+2^{-13}$
Const 10	0.881832	$1-2^{-3}+2^{-7}-2^{-10}$
Const 11	0.857756	$1-2^{-3}-2^{-6}-2^{-8}+2^{-12}+2^{-14}$
Const 12	0.831590	$1-2^{-2}-2^{-6}-2^{-8}+2^{-11}$
Const 13	0.803235	$1-2^{-3}-2^{-4}-2^{-7}-2^{-10}-2^{-11}$
Const 14	0.773056	$1-2^{-2}+2^{-6}+2^{-7}-2^{-11}+2^{-14}$
Const 15	0.740987	$2^{-1}+2^{-2}-2^{-7}-2^{-10}-2^{-12}$
Const 16	0.707434	$2^{-1}+2^{-3}+2^{-4}+2^{-6}+2^{-8}+2^{-14}$
Const 17	0.671578	$2^{-1}+2^{-3}+2^{-5}+2^{-6}-2^{-12}-2^{-14}$

Const 18	0.634545	$2^{-1}+2^{-3}+2^{-7}+2^{-10}+2^{-11}+2^{-14}$
Const 19	0.595676	$2^{-1}+2^{-1}+2^{-5}+2^{-9}-2^{-14}$
Const 20	0.555221	$2^{-1}+2^{-4}-2^{-7}+2^{-10}-2^{-13}$
Const 21	0.514189	$2^{-1}+2^{-6}-2^{-9}+2^{-11}-2^{-14}$
Const 22	0.471332	$2^{-1}-2^{-5}+2^{-9}+2^{-11}+2^{-12}-2^{-14}$
Const 23	0.427790	$2^{-1}-2^{-4}-2^{-7}-2^{-9}-2^{-13}-2^{-14}$
Const 24	0.382607	$2^{-2}+2^{-3}+2^{-7}-2^{-13}+2^{-14}$
Const 25	0.336976	$2^{-2}+2^{-4}+2^{-6}+2^{-7}+2^{-10}$
Const 26	0.290854	$2^{-2}+2^{-5}+2^{-7}+2^{-10}+2^{-12}$
Const 27	0.242356	$2^{-2}-2^{-7}+2^{-11}+2^{-12}+2^{-14}$
Const 28	0.195535	$2^{-3}+2^{-4}+2^{-7}-2^{-12}$
Const 29	0.146999	$2^{-3}+2^{-6}+2^{-8}-2^{-9}+2^{-12}$
Const 30	0.098065	$2^{-4}+2^{-5}+2^{-8}+2^{-12}-2^{-13}$
Const 31	0.049047	$2^{-5}+2^{-6}+2^{-9}+2^{-12}$

Table 7: Representation of twiddle factors in powers of 2

Working: Twiddle unit is also controlled by control signal *count*. At *count* = 4, first set of data arrives from each 8 point FFTs present in Module 1. This set is multiplied by the corresponding twiddle factors and after appropriate addition/subtraction they are stored in an internal register bank which is capable of holding 128 complex values.

Since we are using different data paths at the input, different numbers of registers are required to store the result of multiplication. In case of 4 data paths, only 32 registers are required at a time to store the result of multiplication. For other data paths the numbers of registers simply depend on the number of outputs of 8 point FFT. These registers can be reused in the next clock cycle when the new data arrives. With every clock cycle (after *count* = 4) a new set of data arrives at the input of twiddle unit and is processed as discussed above.

This unit is also controlled by two control signal *fft_128 and fft_64*. The table given below describes how these control signals control the functionality of this module.

Control signal		**Description**
fft_128	***fft_64***	
0	0	Performs two 64 point FFT
0	1	Performs one 64 point FFT
1	0	Performs 128 point FFT
1	1	Not valid

Table 8: Control signal description

The output of this module has to be reordered before it goes to next module i.e. Re-configurable module. A trivial way will be to use a multiplexing technique to reorder the data. But multiplexing 128/64 signals will result in increase in silicon area and high power consumption. Moreover the complexity of the design will also increase. To avoid these overheads, we have adopted a hard wired direct storage technique to store the result of addition/subtraction of multiplied data at the appropriate location.

6.2.4 Reconfigurable block FFT:

The block diagram of reconfigurable module is given below:

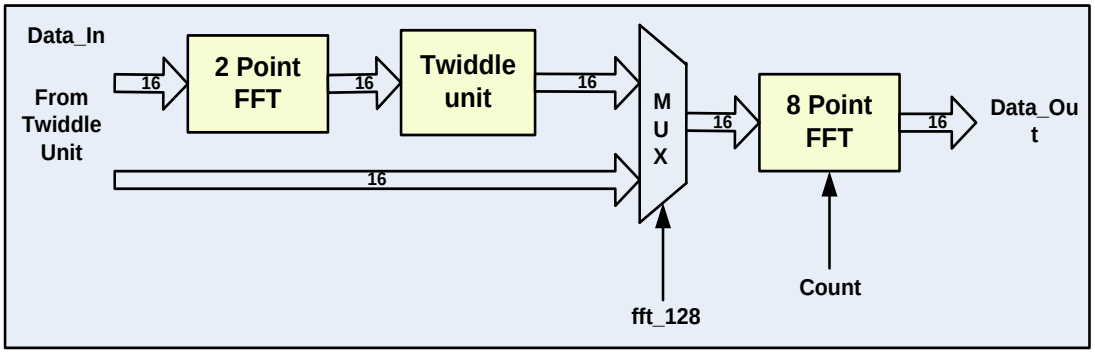


Figure 27: Block diagram of Re-configurable module

Based on the logic states of *fft_128* and *fft_64* control signals this module decides whether to compute 16 point FFT or 8 point FFT. The reason for choosing the particular architecture for 128 point FFT and 64 point FFT (8 x 2 x 8 and 8 x 8 respectively) will become clear in this section.

After twiddle unit processes all 128/64 samples, the output of twiddle unit is fed as input to this module. This module is implemented by using 2 x 8 architecture i.e. first 2 point radix can be computed and then after multiplication with twiddle factors, 8 point FFT will be computed.

To compute 16 point FFT, radix – 4 algorithm can be used to improve efficiency of the design. But the main objective of our design is to have *re-configurability* (computing multiple 64 and single 128 point FFT using the same hardware) and to compute 64 point FFT we have to compute 8 point FFT as the last step. As we know that 8 is not a power of 4, thus 8 point FFT cannot be done using radix- 4 algorithm. Also in [17] it has been shown that, using 4 x 4 architecture to implement 16 point FFT eventually results in more power consumption than using 2 x 8 architecture because of the requirement of an extra resort unit to reorder the data. Thus, we have used 2 x 8 architecture to compute 16 point FFT.

Now to compute 64 point FFT, we have to compute 8 point FFT in the final stage. To implement 8 point FFT we are using a novel multiplexing approach which will result in low power consumption and also less complexity. The multiplexing scheme is based on the fact that in current architecture last stage is 8 point FFT. To perform 16 point FFT, multiplexing scheme directs the data to the 2 point Radix to completely utilise the 2 x 8 architecture whereas if 8 point FFT has to be performed, the multiplexing scheme will direct the input data to 8 point FFT directly. This can be done by efficiently using a multiplexer whose select line will be the control signal *fft_128*. It should be noted that this multiplexer has to used strategically otherwise it will increase the power and area overheads.

Thus re-configurable module computes 16 point FFT for 128 point FFT and computes 8 point FFT for 64 point FFT using the same hardware.

Table 9 describes the number of reconfigurable module needed for different data paths.

Data paths	Number of reconfigurable blocks required
4	2
8	4
16	8

Table 9: Number of reconfigurable block for different data paths

The output of this module has to be reordered before being read as result. To reorder the data and to provide final result, output unit is used and is described next.

6.2.5 Output unit

The output unit is just the reversal of input unit. For a 4 data path structure, the input enters the 28 location of every output register file and with every successive cycle the data shifts downwards and new data enters the output buffer at the top most position. The data from the reconfigurable block is hard wired to the output unit in such a way that every 16^{th} data of the output is fed to the output buffer. Once the buffer is full, the final data is taken from the output unit.

This unit is also equipped with the control signal *mode*. If this signal is in high logic state then real and imaginary parts of the output data are swapped and scaled down to 128/64 by using right shift operation. If the logic state is 0, no swapping takes place.

The size of output registers files for different data paths will be same as of the size of input register files. Figure 28 explains the block diagram of output unit.

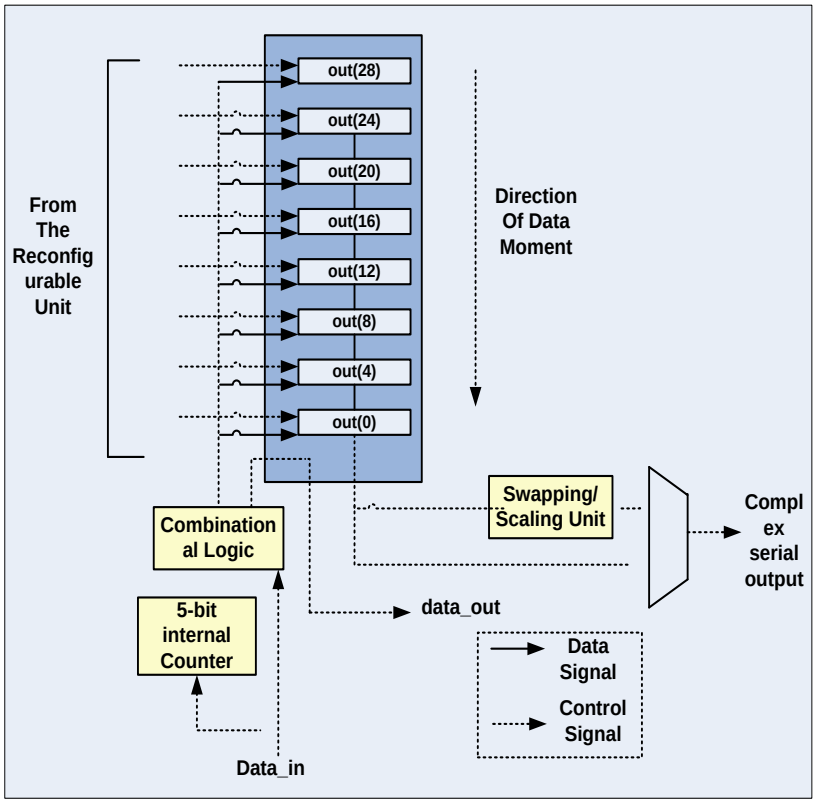


Figure 28: Block diagram of Output unit

6.3 Implementation:

The architecture is modeled in **VHDL** and the functionality of the design is verified by using **Mentor Graphics' Modelsim™ simulator**. The outputs of the design are validated against a standard **MATLAB FFT** function. The synthesis of the design was done by using **Design Vision tool from Synopsys®**. We have used the **umcl18g212t3_tc_180V_25C** library for the synthesis results. The area, power and timing reports were generated by using this tool itself. The design was successfully *back annotated* by using **Cadence™ IUS simulator**. The layout of the design was done by using **SOC Encounter tool from Cadence™**. The technology used is **180nm.**

To implement proposed multiplier unit we have generated the values by writing a **C code** which gives the result in powers of 2.

FPGA implementation of the coded design was also done on **IDE Design Suite tool from Xilinx™**. FPGA's used were **Virtex 7®** and **Virtex 6®.** The results of the implementation are discussed in detail in results and analysis section.

6.4 Use of Low power technique and Optimisation

6.4.1 Low power techniques:

While designing the RTL, our main objective was to reduce computational complexity. To reduce the power consumption we have used ***clock gating*** as a low power technique.

In synchronous designs normally system clock is connected to every flip flop in the design. This results in following components of power consumption:

- ***Combinational logic power consumption:*** In combinational logic, value changes with each clock edge, this results in power consumption.
- ***Flip-flops power consumption:*** Flips flops consume power because they hold a non-zero value even if the input is not changing.
- ***Clock buffer tree power consumption:*** System clock needs to be distributed throughout the design so that the cells can use the system clock. This distribution requires the formation of clock buffer tree, which consumes a lot of power.

To reduce the combinational logic power consumption, gate resizing techniques can be used. But this will result in a minimal power reduction. To achieve a significant reduction in power consumption we have used clock gating at RTL level. RTL clock gating is capable enough to reduce clock network power as well as the combinational logic power [18].

To implement RTL clock gating one needs to identify a common enable signal which is shared by all the flip flops (or flip flops that need to be gated). Usually this enable signal is used as select lines in combination with clock enable to enable the multiplexer or flip flops. RTL clock gating uses this enable signal to gate the clock so that system clock reaches the clock port of the module only when desired.

In our design control signal *count* can be used to generate an enable signal to implement clock gating. As shown in block diagram, *count* is connected to all the modules (except Input Buffer). Depending on specific *count* values modules start their processing. The table given below describes how count controls different modules.

Module	**Count**	
	Start	Stop
Module 1	1	7
Twiddle unit	4	13
Reconfigurable unit	8	17/15(64 point FFT)
Output unit	14	5(after one complete cycle of *count*)

Table 10: Value of count for different modules in 128 point FFT

Using *count* signal values we have asserted enable signal for each module (See Appendix B.2). This enable signal will be held high for the above described count values. Hence if we AND this enable signal with the system clock then system clock will only reach the particular module only when needed. This reduces the power to a great deal as unnecessary switching activity is avoided. Figure 29 below shows simulation waveform without using clock gating.

Figure 29: Simulation waveform without clock gating

As can be seen in Figure 29, every module is using system clock for the whole period. After implementation of RTL level clock gating, modules will get the system clock only when they require it. This is shown in the Figure 30 below.

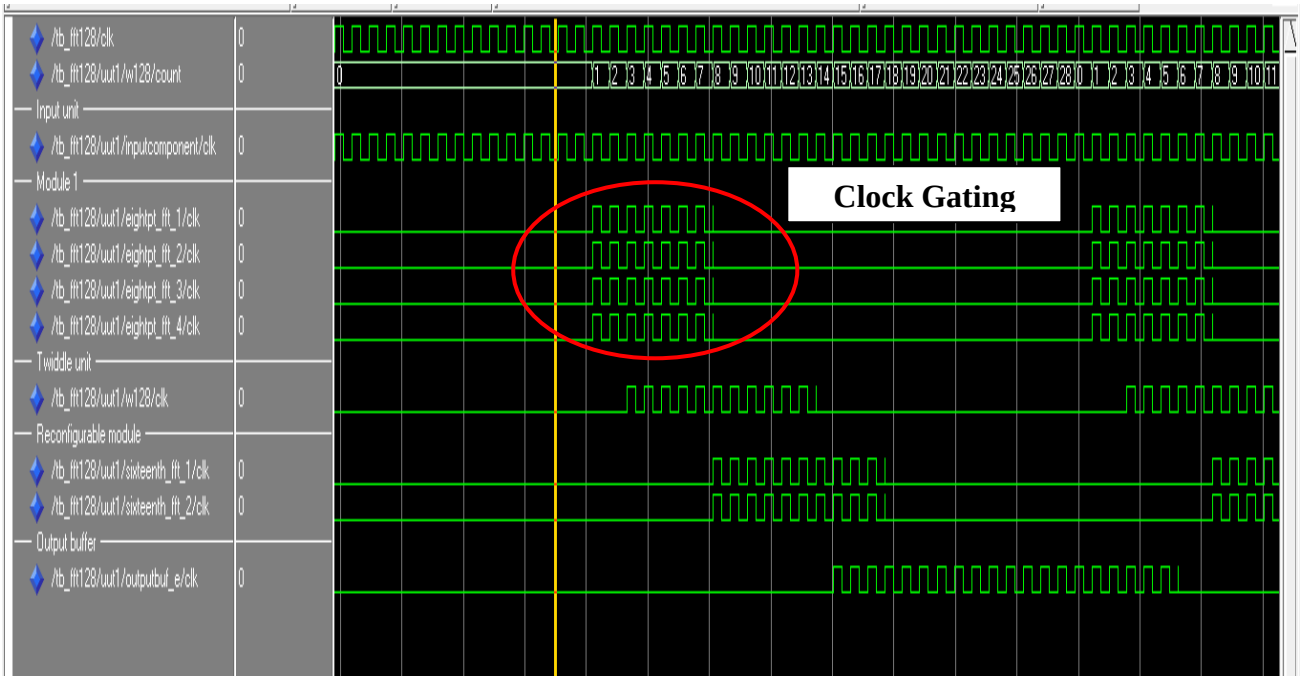


Figure 30: Simulation waveforms with clock gating

The results of reduction in power consumption after implementing RTL clock gating will be seen in results and analysis section.

6.4.2 Optimisation

From Table 10 and Figure 5 it is clear that, Module 1 is switched ON at *count* = 1 and is switched OFF when *count* reaches 7 whereas Reconfigurable block is switched ON at *count* = 8 and is switched OFF at *count* = 17 or 15 (for 64 point FFT). That means Module 1 and Reconfigurable block never function at the same time. We have exploited this condition for the reduction in power and silicon area.

We have reused the 8 point FFT in Module 1 to substitute for the 8 point FFT used in reconfigurable block. The detailed block diagram of the optimised design is given in Figure 31. In case of 128 point FFT, the data after twiddle multiplication in reconfigurable block is redirected to the Module 1. The 8 point FFT is carried out there and using hard wired connection the data is reordered and is provided back to the output unit.

In case of 64 point FFT, the input data of Reconfigurable block is directly directed to Module 1 to perform 8 point FFT.

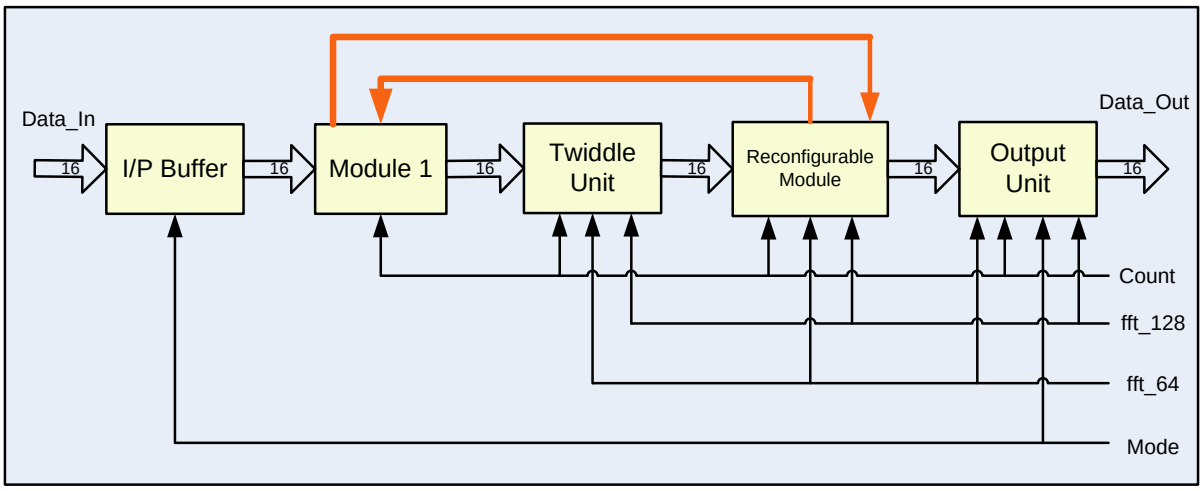


Figure 31: Block diagram of optimised design

The results of this optimisation are really impressive as it greatly reduces the power consumption and silicon area. Although this optimisation increases the hardware complexity as it employs multiplexing scheme. But this disadvantage is compensated by significant reduction in power and silicon area as detailed in results and analysis section.

In Appendix B.3, a section of VHDL code is presented to describe the implementation of this optimisation.

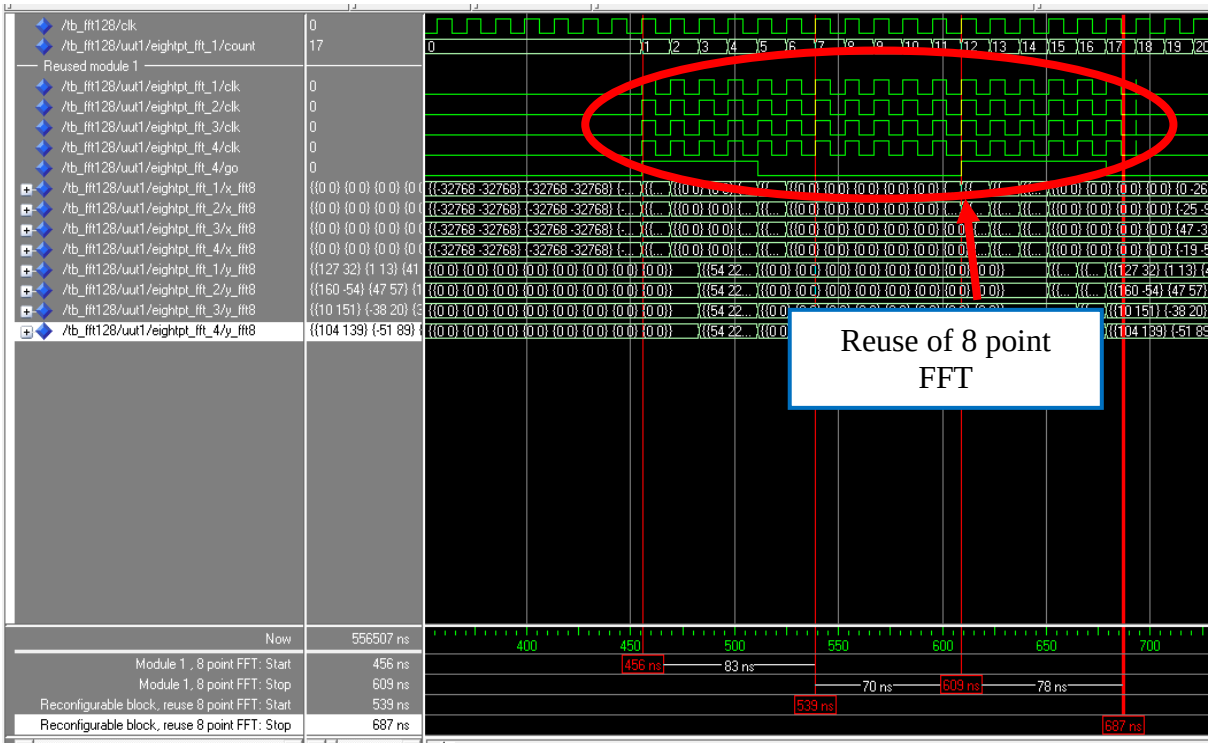


Figure 32: Simulation waveform showing reuse of Module 1 for 8 point FFT

Chapter 7. Results and discussions

7.1 Introduction

An adaptive reconfigurable design was modeled by using different data paths. The aim of using different data paths was to suggest the optimum data paths for this design. The performance parameters for this design are power consumption, silicon area and speed.

Following this introduction, we will discuss the performance evaluation of different data paths. Based on the results of this section, optimum data path is selected for further exploration. Later sections will discuss the FPGA implementation of the design and also summarizes the Layout of the design. The main features of the design and comparison of the design with other existing designs will also be done.

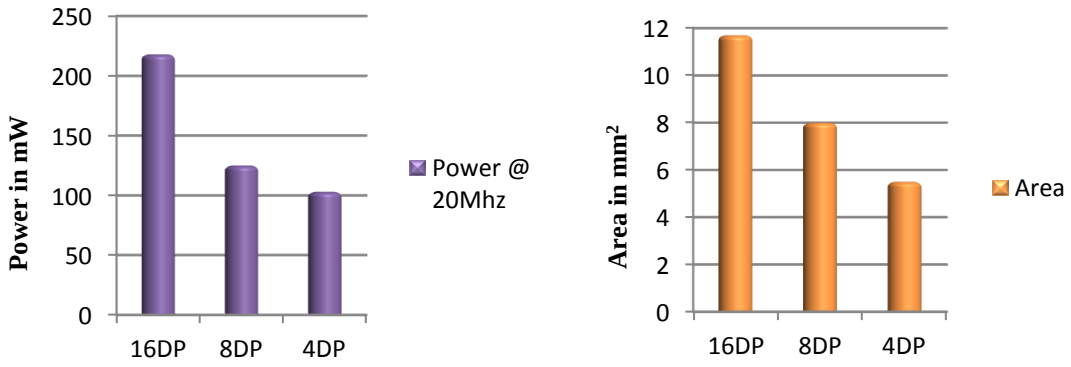
7.2 Performance evaluation of different data paths

The design was modeled using three different data paths i.e. 4, 8 and 16 data paths. Table 11 given below will give the area and power required for each design.

Table 11: Area and power of different data paths

Data path	Area(in mm^2)	Power(in mW)
16	11.7	218.53
8	8.0	125.45
4	5.5	103.17

Figure 33: Power and area of different data paths



The power and area requirement increases with increase in parallel inputs. During RTL designing it was found that, with increase in data paths the number of parallel implementations of Module 1 and Reconfigurable module will also increase. One point to be noted is that there is only difference of 17% in power between 4 and 8 data path but there is a difference of 38% in power between 16 and 8 data path. The reason for the significant

difference is that in case of 8 data path reordering of the data in the input module is decreased.

With increase in data path more number of samples gets processed in a single clock cycle and hence the computation time decreases. It should be noted that by computation time we mean the time taken in computing the FFT and outputting the first set of computed data. It does not include the time taken by the input buffer to load the data. Table 12 given below summarizes the number of clock cycles taken by each data path.

Data paths	Clock cycles required for 128 point FFT	Clock cycles required for 64 point FFT
4	20	18
8	16	14
16	14	12

Table 12: Clock cycles required by different data paths

After evaluating the performance parameters for all the data paths we can now conclude the optimum design.

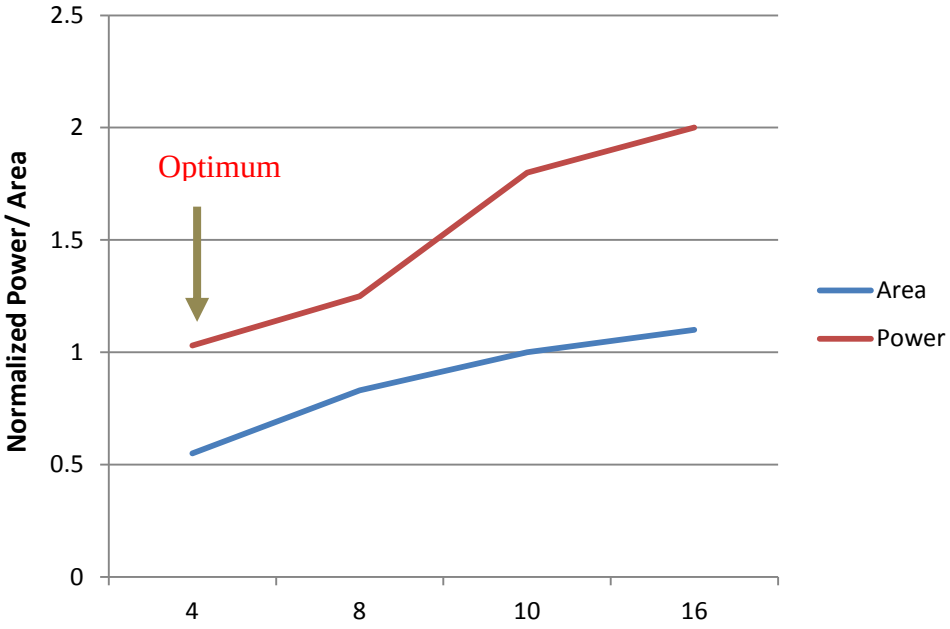


Figure 34: Optimum data path

Thus, 4 data path is the optimum data path and now in following sections we will only discuss the performance of 4 data path.

7.3 Detailed results of Optimum data path

Now we will discuss the detailed module level area and power of 4 data path design.

Table 13 gives the detailed analysis of overall area.

Area analysis	Area in mm^2
Combinational Area	4.2403
Non combinational area	1.3274
Design Area	5.5677
Total Cell Area	5.5681

Table 13: Detailed area analysis of 4 data path

Table 14 describes detailed power analysis of 4 data path.

Power analysis	Power (mW)
Cell internal power	75.3803
Net switching power	28.4280
Total dynamic power	103.8083
Cell leakage power	00.2494

Table 14: Detailed power analysis of 4 data path

Module level Area

Modules	Area(in mm^2)
Input module	65.78
Module 1	30.65
Twiddle unit	150.48
Re- configurable unit	75.62
Output unit	55.91

Table 15: Module wise area analysis

The detailed analysis of power and area suggests that total area of the design is **5.5mm^2** and the total power consumption is **103mW**. The analysis also reveals that Cell internal power is high which can be reduced by using low power techniques.

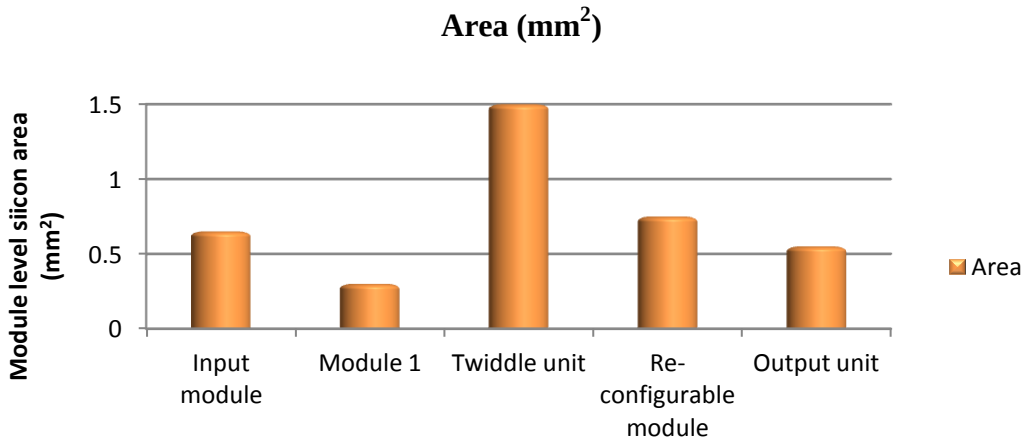


Figure 35: Module level area of 4 data path

Module level area also suggests that twiddle unit occupies the maximum silicon area. This is because of the necessity to store complex values before giving it to the Reconfigurable module.

Module level Power

Modules	Power(in mW)
Input module	20.54
Module 1	13.67
Twiddle unit	39.82
Re- configurable unit	24.96
Output unit	18.93

Table 16: Module wise power analysis

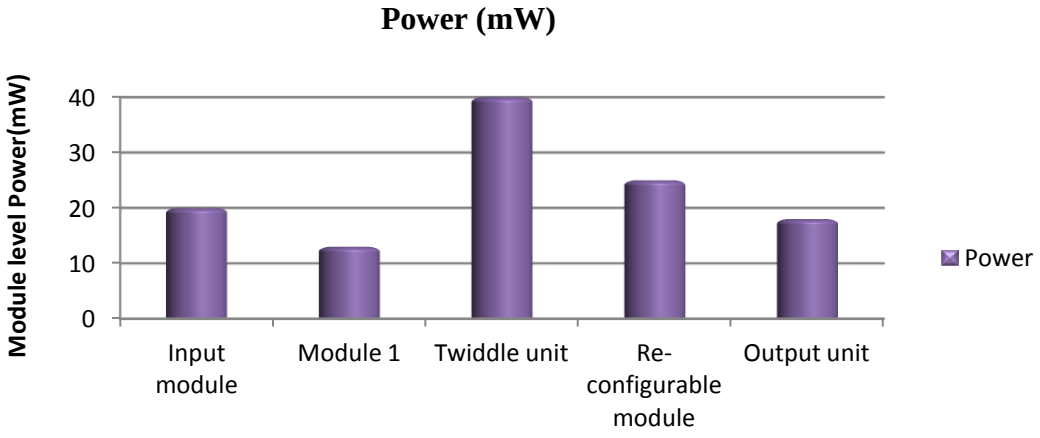


Figure 36: Module level power for 4 data path

The module level area and power analysis leads us to conclusion that which module should be focussed to reduce area and power.

We have implemented clock gating as a low power technique. The performance evaluation of the design after implementing clock gating has been done next.

7.3.1 Performance evaluation after implementing clock gating

The results of RTL clock gating results in significant performance enhancement of the design. However RTL clock gating has no effect on the area of the design. Figure 37 shown below summarises the reduction in power after implementing clock gating.

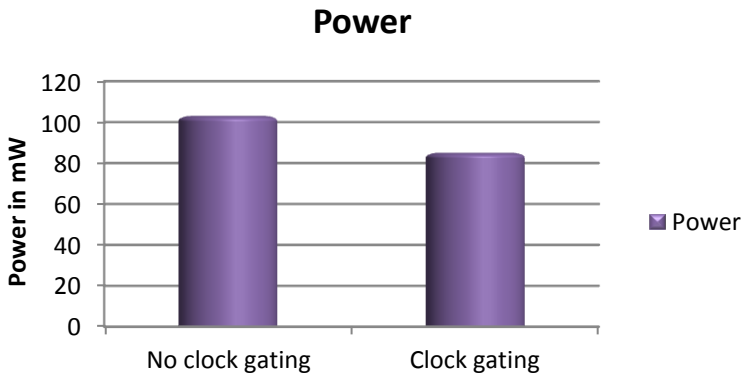


Figure 37: Comparison of power with and without clock gating

There is a 17.4% reduction in power with the use of clock gating. Clock gating reduces net switching power by switching off different modules. Since there are lot of hard-wired connections in the design, switching off the modules stops the switching of these hard wired connections which eventually results in reducing Cell internal power.

7.3.2 Performance evaluation after optimisation

The optimisation technique, explained in Chapter 6.4.2, results in more than 45% reduction in power and 18% reduction in area.

Area analysis	**Area in mm^2**
Combinational Area	3.5303
Non combinational area	1.0274
Design Area	4.5581
Total Cell Area	4.5590

Table 17: Detailed Area analysis

Power analysis	**Power (mW)**
Cell internal power	44.5803
Net switching power	11.4280
Total dynamic power	56.0083
Cell leakage power	00.2494

Table 18: Detailed power analysis

Comparison chart

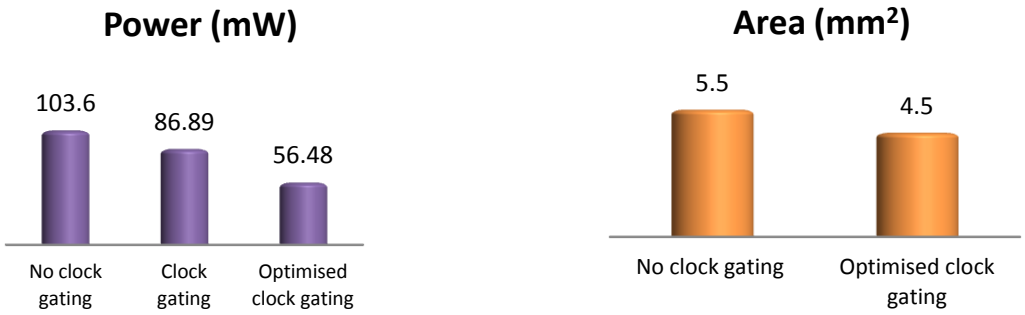


Figure 38: Power and area consumption comparison after optimisation

Module level Power

Modules	Power(mW)
Input module	12.78
Module 1	9.65
Twiddle unit	19.94
Re- configurable unit	6.76
Output unit	5.12

Table 19: Module wise power analysis of optimised design

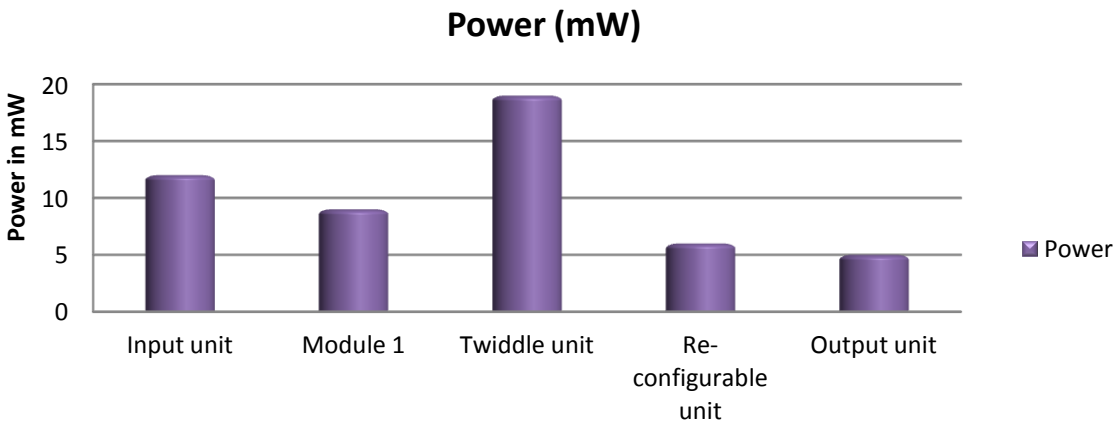


Figure 39: Module level power consumption with optimisation

One can notice the significant difference in the power consumption of Reconfigurable module after optimisation. The reduction in power is because of reusing Module 1 to replace the 8 point FFT in the last stage of Reconfigurable module. Figure 40 shows the module level area requirement.

Module level Area

Modules	Area(in mm^2)
Input module	65.78
Module 1	38.89
Twiddle unit	150.48
Re- configurable unit	51.54
Output unit	55.91

Table 20: Module wise area of optimised data path

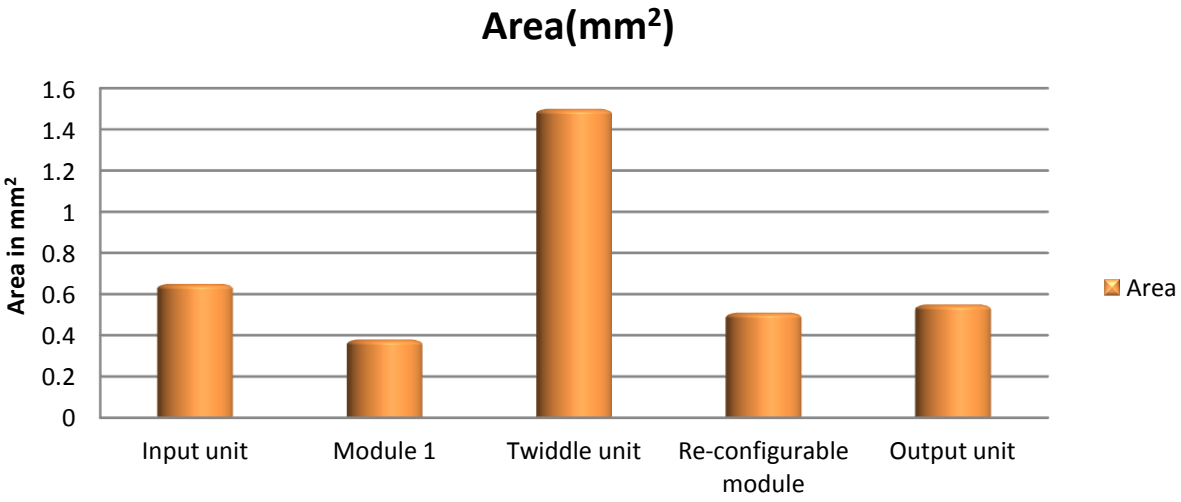


Figure 40: Module level area requirement after optimisation

As we can see in figure 40, there is a significant decrease in the area requirement of reconfigurable unit as we have completely removed the 8 point FFT and reused Module 1 to compute 8 point FFT.

Thus, with the help of optimisation the proposed design consumes **56 mW** of power and requires **4.5 mm^2** of silicon area. This results in a low power reconfigurable design which can compute either 128/64 point FFT or two 64 point FFT in the same hardware.

The main features and comparison of the design with other proposed design will be discussed in section 7.6. In next section, we will discuss FPGA implementation of the proposed design.

7.4 FPGA implementation

We have implemented the proposed design on different FPGAs to test the performance of the design. FPGAs can achieve higher computing speed than digital signal processors and are also cost effective. They achieve ASIC like performance with lower development time and risks [19].
Modern FPGAs offer more functionality and consume less overall power. Thus we have selected two latest FPGAs Virtex® -7 and Virtex® -6 to implement the design. This section will provide device utilisation summary, timing summary and power consumed by the design for both the FPGAs.

7.4.1 Virtex® -7 implementation

Virtex-7 FPGAs are used in advanced systems where high performance and high bandwidth connectivity is required. The target device is **xc7v2000t-2-flg1925.** It is built on 28nm technology designed for maximum power efficiency. It delivers two times better performance and lower power consumption as compared to previous generation FPGAs [20].

7.4.1.1 *Device utilisation summary*

Table 21: Device utilisation summary for Virtex 7

Slice Logic Utilisation	Used	Available	Utilisation
Number of Slice registers	32,598	2,443,200	1%
Number used as Flip Flops	23,249		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	9,349		
Number of Slice LUTs	69,665	1,221,600	5%
Number used as logic	68,371	150,720	5%
Number using O6 output only	48,364		
Number using O5 output only	4,745		
Number using O5 & O6	15,262		
Number used as Memory	128	344,800	1%
Number used as shift registers	128		
Number used exclusively as route-thrus	1,166		
Number of Occupied Slice	20,155	305,400	6%
Number of LUT FF pair used	73,373		
Number of Bonded IOBs	389	1200	32%
Number of BUFG/BUFGCTRLs	2	128	1%

7.4.1.2 Timing summary

Speed Grade: -2

- *Minimum period*: **7.560ns** (**Maximum Frequency: 132.271MHz**)
- *Minimum input arrival time before clock*: **2.761ns**
- *Maximum output required time after clock*: **0.593ns**
- *Maximum combinational path delay*: No path found

7.4.1.3 On-chip power summary

On chip	Power(mW)	Used	Available	Utilisation %
Clocks	18.45	1	---	---
Logic	2.09	77383	1221600	6
Signals	3.67	98920	---	---
IOs	0.44	389	1200	32
Quiescent	1037.49			
Total	1062.15			

Table 22: On-chip power summary for the proposed design

7.4.1.4 Power supply summary

	Total	Dynamic	Quiescent
Supply power (mW)	1064.98	**24.66**	1040.32

Table 23: Power consumption of proposed design on Virtex 7 FPGA

The synthesis tool has allocates the following resources (Table 21): 32,598 Slice registers (1%), 69,665 lookup table (5%) and 128 (1%) number of shift registers. The minimum period is 7.560ns which indicates the maximum operating frequency of 132.271MHz. The dynamic power of the design is only 24.66 mW.

7.4.2 Virtex® -6 implementation

Virtex-6 FPGAs provides high performance logic fabric. They contain many built in system level blocks to allow the designers to achieve highest levels of performance and functionality. It is built on 40 nm process technology. The target device is **xc6vlx760-2ff1760**. This family of FPGA is a programming alternative to custom ASIC technology.

7.4.2.1 Device utilisation summary

Slice Logic Utilisation	Used	Available	Utilisation
Number of Slice registers	32,622	948,480	3%
Number used as Flip Flops	23,273		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	9,349		
Number of Slice LUTs	69,384	474,240	14%
Number used as logic	68,400	474,240	14%
Number using O6 output only	48,378		
Number using O5 output only	4,742		
Number using O5 & O6	15,280		
Number used as Memory	128	132,480	1%
Number used as shift registers	128		
Number used exclusively as route-thrus	1,306		
Number of Occupied Slice	20,500	118,560	17%
Number of LUT FF pair used	73,096		
Number of Bonded IOBs	389	1200	32%
Number of BUFG/BUFGCTRLs	2	32	6%

Table 24: Device utilisation summary for Virtex 6

7.4.2.2 Timing summary

Speed Grade: -2

- *Minimum period*: **7.650ns** (**Maximum Frequency: 130.715MHz**)
- *Minimum input arrival time before clock*: **2.797ns**
- *Maximum output required time after clock*: **0.659ns**
- *Maximum combinational path delay*: No path found

7.4.2.3 On-chip power summary

On chip	Power(mW)	Used	Available	Utilisation %
Clocks	26.02	1	---	---
Logic	2.57	69834	474240	14
Signals	3.53	98977	---	---
IOs	0.67	389	1200	32
Quiescent	4448.46			
Total	4481.25			

Table 25: On-chip power summary for the proposed design

7.4.2.4 Power supply summary

	Total	Dynamic	Quiescent
Supply power (mW)	4481.25	**32.80**	4448.46

Table 26: Power consumption of proposed design on Virtex 7 FPGA

The synthesis tool has allocates the following resources (Table 24): 32,622 Slice registers (3%), 69,834 lookup table (14%) and 128 (1%) number of shift registers. The minimum period is 7.650ns which indicates the maximum operating frequency of 130.715 MHz. The dynamic power of the design is only 32.80 mW.

7.5 Layout

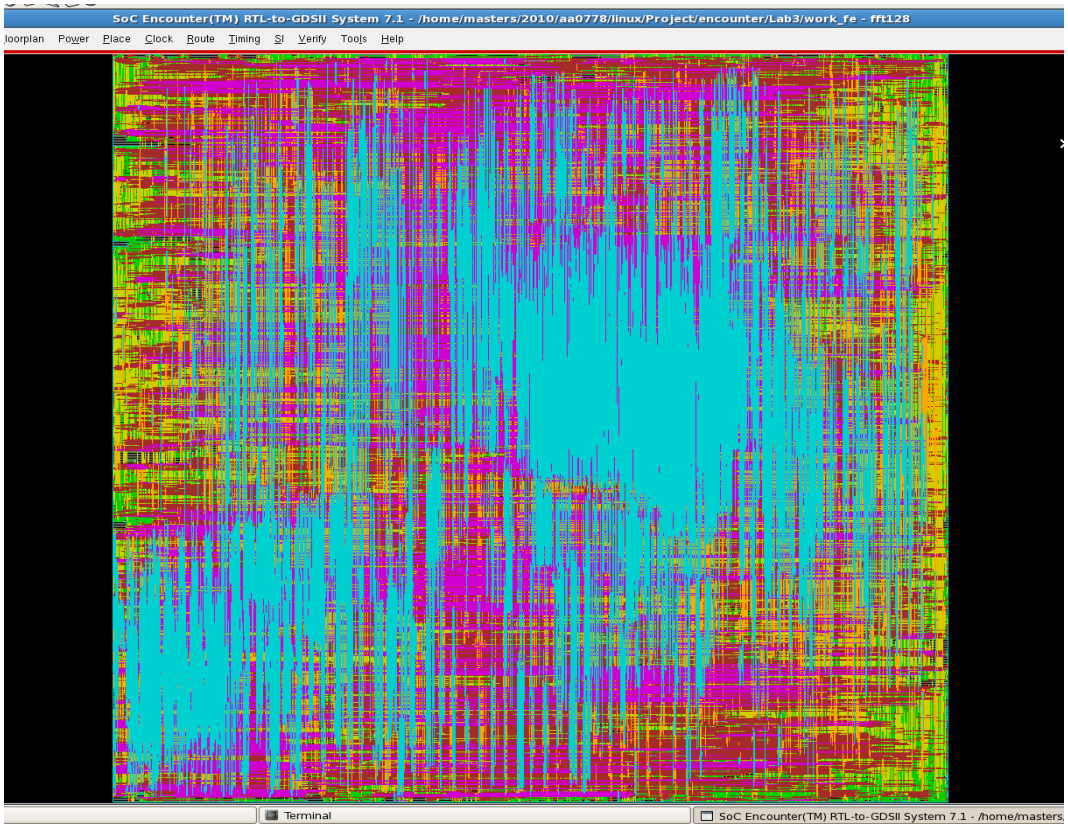


Figure 41: Layout of the proposed design

Summary report for Module: Top Cell

Library used: Artisan 180nm

Design Statistics:

- Number of Pins: 789632
- Number of IO Pins: 389
- Number of Nets: 260092
- Average Pins Per Net (Signal): 3.0360e+00

Chip Utilisation:

- Core Size: 0.6173e+07 um^2
- Chip Size: 0.6177e+07 um^2

Number of Cell Rows: 629

7.6 Main features and comparison

The main features of our design are as follows:

- ***Re-configurability***: The main objective of this project was to design a re-configurable system for FFT implementation. Our design can compute **one 128/64 point or two 64 point FFT at a time**. This feature gives our design an edge over other design.
  In MIMO OFDM systems, when 64 point FFT needs to be computed, input data from two antennas can be given directly to our design to compute the results. This will save the use of an extra hardware, which is needed otherwise in conventional designs.

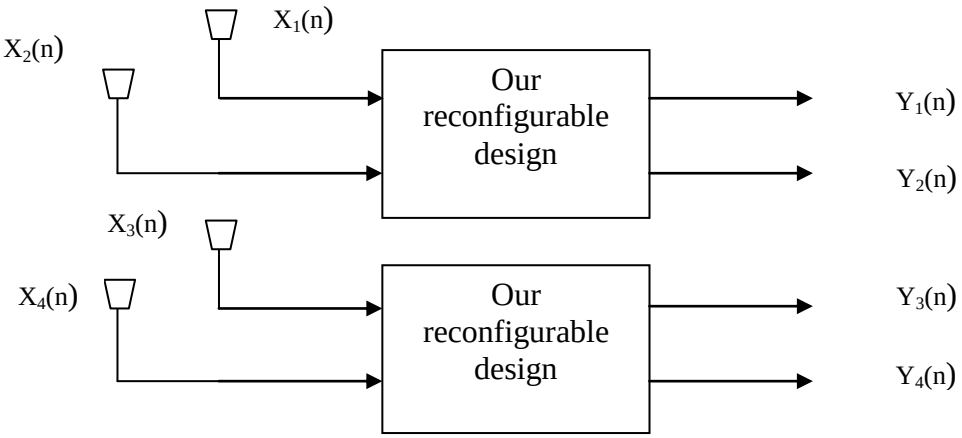


Figure 42: Re-configurability in our design

- ***Low computational complexity:*** Mapping of algorithm with architecture was done to reduce global wirings and multiplexing. Shifting of data downwards in input and output unit reduces global wiring and saves lot of multiplexing.

  1. Using the conventional scheme we would have used four input register files each of size 32 instead of 28(for 4 data path) and more hard wired connection to output the data from input and output unit.

  2. The use of decomposing the twiddle factors in right shifted powers of 2 results in reduced overheads, as multiplication operation requires a lot of global wires and consumes more power than shift and add operations.

  It's the result of these strategies that the design is having a simplified structure with minimal silicon area and power consumption.

- ***Speed:*** Our design can compute 128 and 64 point FFT in just 20 and 18 clock cycles respectively. Thus it easily satisfies the timing constraints of MIMO OFDM systems.

  For f = 20MHz, T = 50ns. If n = number of clock cycles required to compute FFT and **completely output the data** and t is the total time taken, then

$t = n * T = 50 * (20 + 16)$ ns = **1800ns or 1.8µs**. [128 point FFT]

$t = n * T = 50 * (18 + 16)$ ns = **1700ns or 1.7µs**. [64 point FFT]

According to specifications of MIMO OFDM systems, 128 or 64 point FFT needs to be computed within 3.7 µs. Our proposed design easily meets the timing requirements.

- ***Reduced Silicon area and power consumption:*** As compared to conventional non reconfigurable design, our design is having a reduced or comparable area with the additional feature of re-configurability. Power consumption is greatly reduced with respect to conventional designs.

7.6.1 Comparison with other proposed designs

Parameter	**Designs**				
	[22]	**[24]**	**[14]**	**[23]**	**Proposed**
FFT/IFFT	128	128/256/1024/2048	128	64	64/128
Technology	180nm	180nm	180nm	180nm	180nm
Reconfigurable	No	Yes	No	No	Yes
Word Length	16	12	10	16	16
Power	900mW @132 MHz	132.37mW for 1024 samples	77.6mW @110MHz	68.95mW @50MHz	**56mW @20MHz**
Power / bit	56.25	11.03	7.76	4.30	**3.5**

Table 27: Performance comparison of the proposed design with other proposed designs

Table 27 highlights the fact that the proposed algorithm consumes the lowest power among the methods that are discussed. It can be seen that the proposed algorithm provides the best trade off for the features discussed in the above table. The main feature of our proposed design is that it can perform two 64 point FFTs, using the same hardware, which gives this design an edge over the others.

Chapter 8. Conclusion and future work

8.1 Critical Analysis

- An appropriate and efficient algorithm was developed using classical divide and conquer approach.
- Re-configurability was achieved as proposed design can compute one 64 point or two 64 point or one 128 point FFT using the same hardware.
- Design was implemented using three different datapaths (4, 8 and 16). 4 datapath was found to be the optimum datapath.
- Clock gating was used as low power technique to reduce power consumption.
- The design was optimised by reusing 8 point FFT and further reduction in silicon area and power consumption was achieved.
- All the specifications of a MIMO OFDM system were met and it was found that proposed design computes FFT/IFFT in almost half the time set by timing constraints.
- The design was successfully backannotated to cross check the simulation and synthesis results.
- The design was verified for its performance on FPGAs. The design was found to have low power dissipation and utilises minimum resources.

8.2 Conclusion

We have observed that FFT/IFFT is one of the basic building blocks of a MIMO OFDM system. FFT is used as it provides an efficient way to convert the time domain sequence into frequency domain. Various efficient algorithms of FFT were discussed in this report that focused on reducing the number of computations involved eventually reducing the power consumed and increasing the speed of the system.

Based on the basic algorithms, various architectures have been discussed for the hardware implementation. As discussed, pipeline architecture's serial processing, lower memory and logic requirements make it a preferred choice for hardware implementation. However, for applications requiring higher clock frequency, memory based architectures are preferred. The study of different architectures and algorithms provided a motivation for the new approaches that can be developed.

The main design issues with the hardware implementation of such huge designs include area coverage, computational complexity; interconnect delay, routability, and power consumption. One needs to deal strategically with these design issues in order to implement an efficient architecture.

In this thesis we have proposed an efficient and appropriate algorithm for the hardware implementation of FFT/IFFT module for a MIMO OFDM system. The architecture is based on the decomposition of 128 point into 16(further decomposed into 2 and 8 point FFT) and 8 point FFT and 64 point FFT into two 8 point FFTs. The algorithm to architecture mapping has been done in such a way such that it is appropriate for VLSI implementation. From a VLSI point of view, the design is regular, modular, re-configurable and is simple wired. The re-configurability feature of the design makes it attractive as it results in computing multiple

FFTs using the same hardware. Our proposed architecture is also low power and requires minimal area.

8.3 Future work

In the given time, we think that we have achieved all the objectives which we thought of initially. But in every design there is always a scope of improvement.

The design presented by us is a CMOS technology based approach and hence we have given main emphasis on power reduction at the architectural level. The technology library does not contain appropriate memory modules because of which we had to use register-based memory structure. This eventually resulted in a design that will occupy more silicon area and consumes more power as compared to memory-based design. So in future the same design can be implemented using memory based structure which can improve the performance of the current design.

As can be seen from analysis of results, twiddle unit consumes most of the power. An efficient technique can be adopted to design multiplier unit to further reduce the power and area. There are lots of approaches (based on memory structure) [2] [21] to efficiently compute twiddle factor multiplication. One can refer to these approaches to modify the current design according to their specific needs.

The same design can be extended to compute two 128 point FFT using the same hardware. This work can greatly reduce the power consumption of the overall MIMO OFDM system.

Thus extending the current design according to the above suggestions will result in a highly efficient design.

Chapter 9. References

[1] **John G.Prokias, Dimitris G. Manolakis,** "*Digital Signal Processing Principles Signal Processing"*, 2005.

[2] **Yu-Wei Lin and Chen-Yi Lee**, "*Design of an FFT/IFFT Processor for MIMO OFDM Systems*," IEEE Transactions On Circuits And Systems –I: Regular Papers, Vol.54, No.4, April 2007.

[3] **Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold**, "*A 64 point Fourier Transform Chip for High Speed Wireless LAN Application Using OFDM*", IEEE Journal of Solid State Circuits, Vol.39,No.3,March 2004.

[4] **Shashank Mittal, Md. Zafar Ali Khan, M.B. Srinivas**, "*A Comparative Study of Different FFT Architectures for Software Defined Radio*", Proccedings of VII SAMOS Workshop, LNCS, Vol. 4599,Greece, July 2007.

[5] **E. E.Swartzlander, V. K. Jain, and H. Hikawa**," *A radix 8 wafer scale FFT processor*" Journal VLSI Signal Processing, May 1992.

[6] **Dr. Naim Dahnoun**, "*Digital Signal Processing system*", University of Bristol, 2011.

[7] **H.V. Sorensen, M.T. Heideman, and C.S. Burrus, "***On computing the split-radix FFT",* IEEE Transactions on Signal Processing, 34(1), 1986.

[8] **J. O'Brien, J. Mather, and B. Holland**, *"A 200 MIPS single-chip 1 k FFT processor,"* in Proc. IEEE Int. Solid-State Circuits Conf., 1989.

[9] **L.R. Rabiner and B. Gold**, Theory and Application of Digital Signal Processing. Prentice-Hall, Inc., 1975.

[10] **E.H. Wold and A.M. Despain**, "*Pipeline and parallel-pipeline FFT processors for VLSI implementation*", IEEE Trans. Computation, May 1984.

[11] **A.M. Despain**, " *Fourier transform computer using CORDIC iterations",* IEEE Trans. Comput., C-23(10):993-1001, Oct.1974.

[12] **E. E. Swartzlander, W. K. W. Young, and S. J. Joseph**, "*A radix 4 delay commutator for fast Fourier transform processor implementation*" IEEE J. Solid-state Circuits, SC- 19(5):702-709, Oct. 1984.

[13] **G. Bi and E. V. Jones**, "*A pipelined FFT processor for word sequential data",* IEEE Trans. Acoust., Speech, Signal Processing, 37(12):1982-1985, Dec. 1989.

[14] **Y.-W. Lin, H.-Y. Liu, and C.-Y. Lee**, "*A 1 GS/s FFT/IFFT processor for UWB applications*" IEEE J. Solid-State Circuits, vol. 40, no. 8, pp. 1726–1735, Aug. 2005.

[15] **H. Shousheng and M. Torkelson**, *"Designing pipeline FFT processor for OFDM (de)modulation,"* in Proc. URSI Int. Symp. on Signals, Syst, Electron., Oct. 1998.

[16] **B. M. Bass,** "*A low-power,high-performance, 1024-point FFT processor*," IEEE J. Solid-State Circuits, vol. 34, no. 3, pp. 380–387, Mar.1999.

[17] **J.Mathew, K.Maharatna, D.K.Pradhan and A.P.Vinod** ,"*Exploration of power optimal implementation technique of 128 point FFT/IFFT for WPAN using Pseudo-Parallel Datapath Structure",* 10th IEEE Singapore International conference, October, 2006.

[18] **Frank Emnett and Mark Biegel**, "*Power Reduction Through RTL clock gating*", SNUG, San Jose, 2000.

[19] **Ahmed Saeed, M. Elbably, G. Abdelfadeel and M.I. Eladawy**, "*Efficient FPGA implementation of FFT/IFFT processor,*" International Journal of Circuits, Systems and signal processing, Issue 3, Volume 3, 2009.

[20] <http://www.xilinx.com/products/silicon-devices/fpga/virtex-7/index.htm>, *Xilinx Virtex FPGA family*, 2011 [Accessed August 2011].

[21] **Jen-Chuan Chi and Sau-Gee Chen**, "*An Efficient FFT Twiddle factor generator*" National Chiao Tung University Department of Electronics Engineering and Institute of Electronics.

[22] **G.Zhang, F.Chen**, "*Parallel FFT with cordic for ultra wide band*", 15th IEEE International Symposium on Personal, Indoor and Mobile Radio communications, PIMRC, 2004.

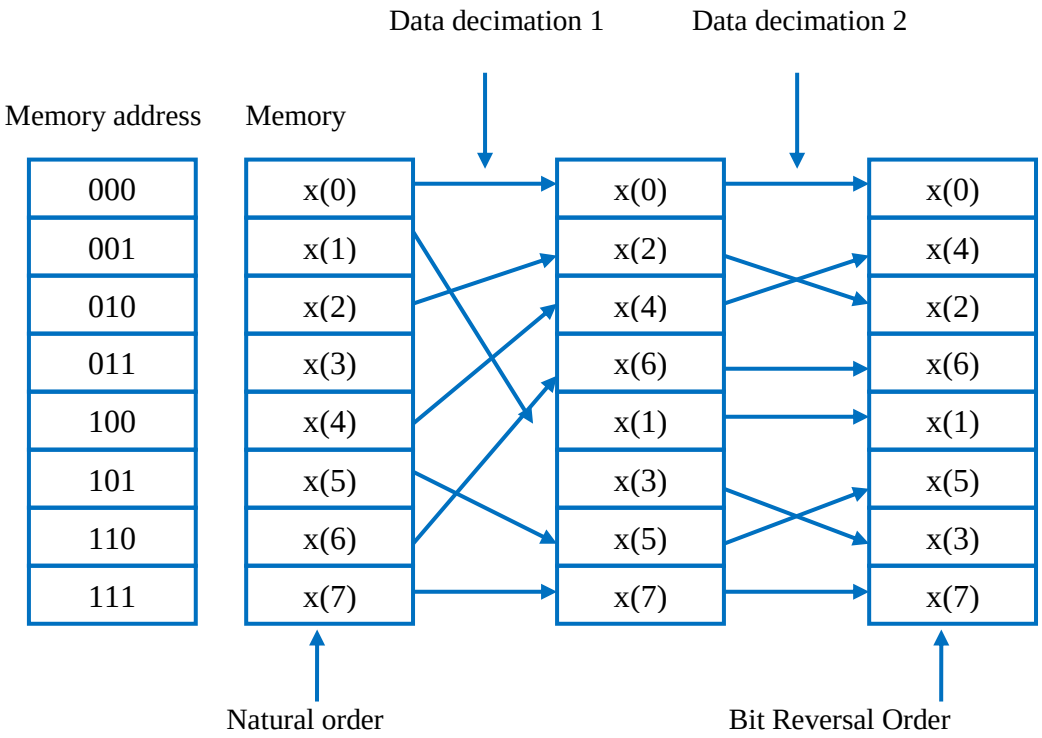
[23] **Wei Han, T. Arslan, A.T. Erdogan and M. Hasan**, "*A novel power pipelined FFT based on sub expression sharing for wireless LAN applications,*" in Proc. IEEE workshop on Signal Processing Systems (SIPS), October 2004, pp. 83-88.

[24] **Ching-Hua Wen, Jen-Chih Kuo**, **Chih-Hsiu Lin, An-Yeu (Andy) Wu**, "*Improved Architecture of variable length FFT processor*", EURASIP Journal on Applied Signal Processing .2003.

Chapter 10. Appendix A – Decimation

Shuffling of data and bit reversal

When the input data sequence is stored in bit-reversed order and butterfly computations are performed in place, the DFT sequence at the output is obtained in normal order. The bit reversal method is described below.



$N_2N_1N_0$	$N_0N_2N_1$	$N_0N_1N_2$
000	000	000
001	100	100
010	001	010
011	101	110
100	010	001
101	110	101
110	011	011
111	111	111

Figure 43: A: Shuffling of the data and bit reversal

Chapter 11. Appendix B – Code Snippets

B.1 Multiplier unit

```
#include"stdio.h"
#include"stdlib.h"
#include"math.h"

/*main function begins*/
int main()
{
	int i,j,array[6],k;
	float num,copy_num;
	printf("enter the number that needs to be expressed interms of 2's powers\n");
	scanf("%f",&num);
	copy_num = num;

	if((num > 0.0) && (num <= 0.25))
	{
		j=0;
		i = 2;
		array[j++] = 0;
		while ((num != 0) && (j <= 5) )
	    {
			if(pow(2,-i) <= num)
			{
				array[j++] = i;
				num -= pow(2,-i);
			}
			i++;
		}
		printf("the number in 2's powers is:\n");
		for(j=0;j<6;j++)
		{
			if(j == 0)
				printf("%f=%d+",copy_num,array[j]);
			else if(j == 5)
				printf("1/2^%d\n",array[j]);
			else
				printf("1/2^%d+",array[j]);
		}
	}
```

```
else if((num > 0.25) && (num <= 0.5))
{
        j=0;
        i = 2;
        array[j++] = 2;
        num = num-0.25;
        while ((num != 0) && (j <= 5) )
      {
                if(pow(2,-i) <= num)
                {
                        array[j++] = i;
                        num -= pow(2,-i);
                }
                i++;
        }
        printf("the number in 2's powers is:\n");
        for(j=0;j<6;j++)
        {
                if(j == 5)
                        printf("1/2^%d\n",array[j]);
                else
                        printf("1/2^%d+",array[j]);
        }

}
else if((num > 0.5) && (num <= 0.75))
{
        j=0;
        i = 2;
        array[j++] = 1;
        num = num-0.50;
        while ((num != 0) && (j <= 5) )
      {
                if(pow(2,-i) <= num)
                {
                        array[j++] = i;
                        num -= pow(2,-i);
                }
                i++;
        }
        printf("the number in 2's powers is:\n");
        for(j=0;j<6;j++)
        {
                if(j == 5)
                        printf("1/2^%d\n",array[j]);
                else
                        printf("1/2^%d+",array[j]);
        }
}
```

```
else if((num > 0.75) && (num < 1))
        {
                j=0;
                i = 2;
                array[j++] = 1;
                array[j++] = 2;
                num = num - 0.75;
                while ((num != 0) && (j <= 5) )
               {
                        if(pow(2,-i) <= num)
                        {
                                array[j++] = i;
                                num -= pow(2,-i);
                        }
                        i++;
                }
                printf("the number in 2's powers is:\n");
                for(j=0;j<6;j++)
                {
                        if(j == 5)
                                printf("1/2^%d\n",array[j]);
                        else
                                printf("1/2^%d+",array[j]);
                }
        }
        else
                printf("enter the number in the range 0 to 1\n");
}
```

B-2 Enable signal assertion using control signal '*count*'

```
if (clk = '1' and clk'event) then
                        -- turns on 16 point FFT
                        if (count_toplevel > 8 and count_toplevel < 15) then
                                go_fft16 <= '1';
                                else null;
                        end if;

                        -- turns on 8 point FFT
                        if (Enable= '1' and count_toplevel < 4) then
                                        go_fft8 <='1';
                                else
                                        go_fft8 <='0';
                        end if;
                else null;
        end if;  -- clk
```

B-3 Optimised design implementation

```
if (clk = '1' and clk'event) then
                                -- turns on 16 point FFT

                                if (count_toplevel > 8 and count_toplevel < 15) then
                                        go_fft16 <= '1';
                                        else null;
                                end if;

                                -- turns on 8 point FFT (Re-configurability)

                                 if ((Enable= '1' and count_toplevel < 4) or (fft_128 ='1'
and count_toplevel > 10 and count_toplevel < 16) or (fft_128 ='0' and (count_toplevel > 8
and count_toplevel < 14))) then
                                                go_fft8 <='1';
                                        else
                                                go_fft8 <='0';
                                end if;
                        else null;
                end if;  -- clk
```

```
process(count_toplevel)

        begin
                        -- Reusing Module 1, 8 point FFT

                                if(count_toplevel > 7 and count_toplevel < 18 ) then
                                        reconfig_1 <= back_fft8_1;
                                        reconfig_2 <= back_fft8_2;
                                        reconfig_3 <= back_fft8_3;
                                        reconfig_4 <= back_fft8_4;
                        -- I^st stage 8 point FFT

                                elsif (count_toplevel > 0 and count_toplevel < 8 ) then
                                        reconfig_1 <= x_fft8_1;
                                        reconfig_2 <= x_fft8_2;
                                        reconfig_3 <= x_fft8_3;
                                        reconfig_4 <= x_fft8_4;
                                end if;
end process;
```