University of Strathclyde

Department of Electronic & Electrical Engineering

On The Realisation of Sequential Fast Fourier Transform for Orthogonal Frequency Division Multiplexing Based on Field Programmable Gate Array

By

Yousif Awad

A thesis presented in partial fulfilment of the requirements for the degree of Doctor of Philosophy

2012

Abstract

In the last 25 years Digital Signal Processing (DSP) has become one of the key components in the development of mobile and wireless technologies. One of the core DSP algorithms widely used in many applications is the Fourier Transform, in both the standard Discrete Fourier Transform (DFT) and the Fast Fourier Transform (FFT) implementation. The DFT and FFT are widely used for many applications such as spectral analysis, but in modern mobile and wireless communications standards, their particular significance is in the context of the signalling strategy of Orthogonal Frequency Division Multiplexing (OFDM), where the FFT and Inverse FFT (IFFT) are deployed at the transmit and receive sides of both uplink and downlink.

Over the last few years the physical layer of many 4^{th} generation wireless standards, such as IEEE 802.20, IEEE 802.16, and 3GPP2, Ultra Mobile Broadband (UMB), as well as 3G Long Term Evaluation (LTE), all rely heavily on using OFDM multicarrier modulation. OFDM is now favoured in many physical layers for mobile and wireless radio systems, given that it has good properties for mitigating multipath propagation and operating in fading channels.

In the implementation of OFDM, FFTs and IFFTs are required at various lengths, word lengths, speeds of operation, and on a variety of platforms such as DSP processors, Application Specific Integrated Circuits (ASICs) and – in basestations, mostly likely – on Field Programmable Gate Arrays (FPGAs). Therefore in this thesis we have focused on methods of efficient implementation of the FFT and IFFT on FPGAs, taking very clear note of issues such as resource costs, speeds, word length, and programmability.

In this thesis, two FFT architectures have been introduced, the first based on the classic butterfly computation and the use of look-up tables to store FFT trigonometric constants, and the second based on using a COordinate Rotation Digital Computer (CORDIC) method to generate the trigonometric constant values.

In this work, the FFT Radix-2 Decimation in Frequency (DIF) algorithm has been chosen and efficiently implemented on a Xilinx FPGA in a programmable form to suit a variety of PHY layer standards requirements. The design implements a

pipelined sequential architecture to reduce the resource area and maintain high throughput. A key contribution is the introduction of an optimized butterfly processor that uses only two multipliers for the twiddle factor multiply, rather than the more conventional four as found in the designs available from FPGA vendors and IP repositories. The direct implementation of the butterfly requires four multipliers (to perform a complex data multiplication) and four corresponding block RAMs to store the input and output data. The proposed architecture utilises the 2 multipliers technique to reduce the resource cost, albeit it a cost of maximum throughput. For the CORDIC based design, no hardware multipliers are required since the CORDIC can directly implement the twiddle factor multiply. For both architectures to increase the speed of the FFT sequential architecture, a dual clock method has been used. To verify and generate demonstrable designs, the architectures have been simulated, synthesized and implemented on a Virtex 5/Xilinx FPGA.

Acknowledgment

This thesis would not have been possible without the support of many people. I wish to express my gratitude to my supervisor, Prof. Robert W. Stewart who was abundantly helpful and offered invaluable assistance, support and guidance. My thanks also go to Dr. Stephan Weiss for many helpful suggestions on the thesis. My deepest gratitude to Ousman Sadiq and Dr. Louise Crockett whose knowledge and assistance made this study successful and a special thanks also to all of the DSP Enabled Communication Group and University of Strathclyde.

I would like to thank my parents for supporting and encouraging me to pursue this degree.

I would also like to convey thanks to the Ministry of Higher Education and Scientific Research/Iraq and Iraqi Embassy Cultural Attaché/London for providing financial support.

Table of Contents

List of Figures

List of Tables

1 Introduction and Overview

1.1 Introduction

Over the last two decades significant progress has been made in the extensive deployment and update of cellular mobile and wireless networks. In 2011 there are now estimated to be over four billion subscribers worldwide receiving voice and data services through the wireless and cellular networks. For many people getting access to the internet in a "anywhere" and "anyway" manner is the key goal in both business and customer markets and to provide these services the continuing development and data rate increase of the mobile internet technology is essential. In 2011 the new 4G wireless networks such as LTE offer high-speed network access with potential data rates of greater than 10 Mbits/sec. One of the core computational signal processing components of the physical layers (PHY) in many 4G wireless standards is Orthogonal Frequency Division Multiplexing (OFDM), used as a modulation and multiplexing technique in both up and down links for many wireless standards [3-4], including the new emerging LTE global standard.

In OFDM systems, one stage of the modulation and demodulation are performed by using the Fast Fourier Transform (FFT) and its inverse (the IFFT) to orthogonalise the transmit signals [5-6].

For implementation of 4G standards on base stations, FPGAs have become one of the key components for implementation. Therefore in considering only the FFT and IFFT components in this thesis, the need for high speed FFTs for multiple standards and data streams is clear. The FPGAs have a key advantage over single core DSP processors due to the extensive parallelism available on the device; albeit exploiting this parallelism efficiently is something that system designer must do carefully.

To produce initial designs, and in common with other engineers designing for OFDM on FPGAs, in the research work of this thesis we have used the Xilinx FPGA, and made use where appropriate of the FFT support tools (System Generator [7] and Core Generator), to produce "vendor" designs. (Other tools used for design and investigation include, Synplicity [8] and Modelsim [9]). In using these tools it is often the case that many of the DSP blocks are pre-configured and not necessarily optimised for particular algorithms and architectures being implemented for a particular radio standard. Hence for the research work in this thesis we have focused on the FFT algorithm implementation as used for OFDM, and considered its use in various 4G standards with respect to FFT size, word length, throughput etc, and hence aiming to recommend the optimal method for implementation of the FFT. This choice includes decisions on resources (amount of hardware), flexibility and programmability and generally on providing architectures that will achieve the speed of implementation required by a given 4G standard.

The core output of this research works has been to evaluate available FFT IP blocks and standard FPGA implementations, to then investigate and design an efficient, high-speed optimized FFT/IFFT dedicated to the OFDM system and applicable and programmable for the current 4G standards. The properties of the presented FFT designs are novel, programmable, extendable and integratable into the physical layer of a number of 4G radio designs. In particular in this thesis we will benchmark the designs against achieving timing and performance for LTE, 802.16 and 802.20 standards.

Therefore we will show the design of an optimized dynamic OFDM transmitter to accommodate different physical layers of the 4G wireless standards.

1.2 Mobile and Wireless Networks Overview

To set the scene of mobile and wireless evolution in this section, the historical background of cellular networks is briefly reviewed. Figure 1.1 summarises the mobile and wireless evaluation from 1st to 4th generation. The 1st generation of mobile/wireless was of course analogue and essentially a Frequency Division Multiple Access (FDMA) system. The emergence of 2G network was the first digital implementation to feature Time Division Multiple Access (TDMA), frequency division multiple access (FDMA), and classic modulation methods. The Global System for Mobile communications (GSM) in particular was the main standard during this era in the 1990's. The single carrier Gaussian Minimum Shift Keying (GMSK) modulation technique was used in GSM due its resistance to Inter symbol interference (ISI). The Gaussian shape impulse response filter spreads the input bits width, achieving a narrower transmission spectrum and an improved resistance to ISI [10]. Developments in GSM then targeted an increase in data rate and established 2.5G with its General Packet Radio System (GPRS).

The 3G cellular release was largely based on spread spectrum and brought Code Division Multiple Access (CDMA) principles. One of the key reasons of moving to CDMA was the demonstration that spread spectrum helped minimise the multipath problem that limited data rates with 2G. Despite its success it is interesting to then note that most of the 4G access technologies moved to OFDM based techniques, with one of the key motivations being, again, the mitigating of multipath problems. OFDM splits the available spectrum into a number of narrowband transmission channels known as subcarriers which, independent of the transmission channel being frequency-selective or frequency-flat, are not subject to ISI and only change amplitude and phase of OFDM-multiplexed symbols At the receiver the data streams received on all subcarriers can be de-multiplexed to form the original data stream. In OFDM systems this was achieved by IFFT/FFT essentially channelizing

the transmit bandwidth and allowing simple FFT-bin-based complex equalisers to be used.

Figure 1.1 also summarises some of the different issues related to multiplexing, technologies, systems and features. The upper layer of the diagram gives the multiplexing and bit rate while the middle layer shows the technologies used. The lower layer summarises some of the systems that have then been established and deployed.

The first generation (1G) analogue cellular system supported voice communication but with limited roaming. It used FDMA as multiplexing technology and analogue frequency modulation from baseband to carrier. The first available 1G cellular telephone system, the Advanced Mobile Phone System (AMPS), was introduced in 1979 in the United States.

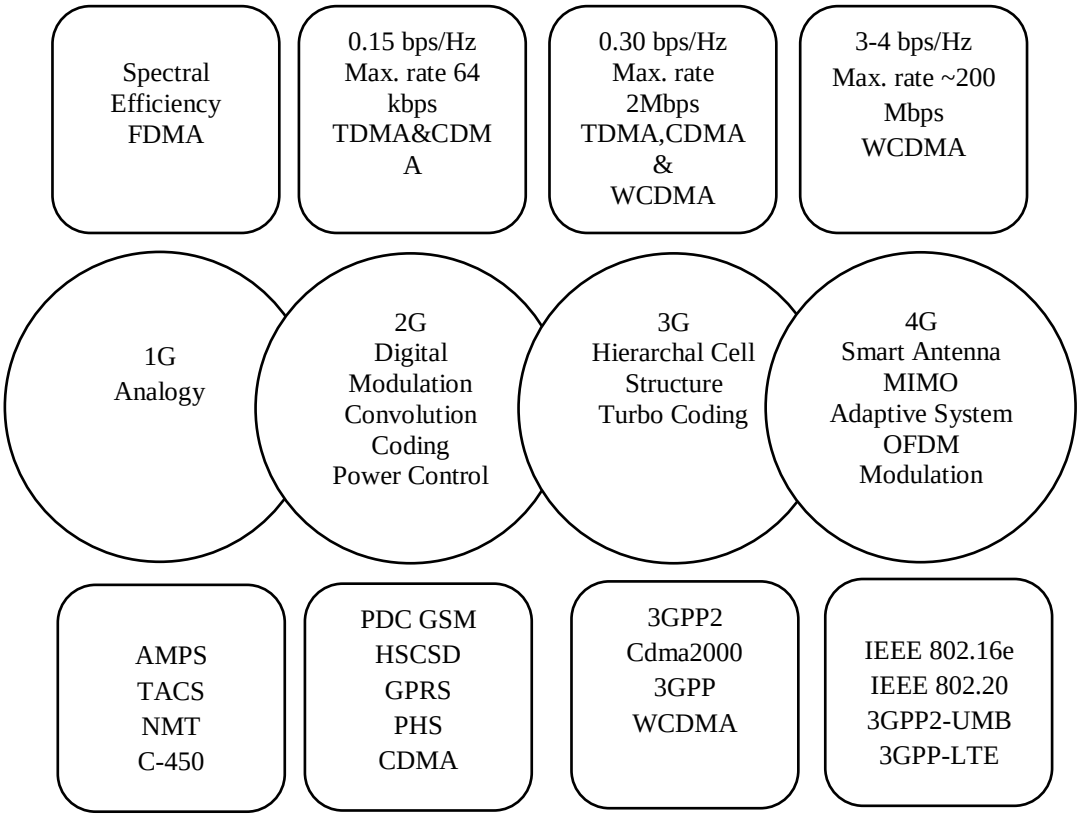

Figure 1.1: Cellular Networks Background

Several other analogue cellular systems including TACS, NMT, C450, were introduced in Western Europe, albeit all were independent and not inter-operable. The second generation (2G) systems were based on Time Division Multiple Access (TDMA) and also featured the introduction of Code Division Multiple Access (CDMA) technologies. The 2G was primarily designed to improve voice quality and provide a set of rich voice features, but also brought limited data in the form of text options. The two most widely deployed 2G systems are the *Global System for Mobile communications* (GSM) and *Code Division Multiple Access* (CDMA). Both the GSM and CDMA standards groups followed their 2G success by forming their own separate 3G partnership projects (3GPP and 3GPP2, respectively) to then research on the next (i.e. third) generation of mobile systems based on spread spectrum techniques and using much wider bandwidths than 2G. The 3G standard in 3GPP was referred to as Wideband Code Division Multiple Access (WCDMA) because it uses a larger 5MHz bandwidth compared to the 1.25MHz bandwidth used in 3GPP2's cdma2000 system [11-12].

In today's society, access to networked data and information services has become critically important to users of business, entertainment, and social networking applications. Users look for high-speed, high-reliability, and high-quality access to this information while they are fully mobile [4]. As such Fourth generation (4G) wireless standards are developed to meet these requirements of high speed data (> 10 Mbits/sec) anywhere and anytime. Some of the more popular 4G standards are Mobile WiMAX (IEEE 802.16e), MobileFi (IEEE 802.20), 3Generation Partnership Project 2-Ultra Mobile Broadband (3GPP2-UMB) and 3Generation Partnership Project- Long Term Evaluation (3GPP-LTE). The Standards use Orthogonal Frequency Division Multiplexing (OFDM) techniques for modulation and multiplexing. It is interesting to note however that the adoption of a standard has as much to do with politics as it do with technology and efficiency.

1.2.1 Open System Interconnection (OSI)

Virtually all modern wireless networks consist of Medium Access Control (MAC) and physical (PHY) layers, with the MAC responsible for setting the rules which determine how to access the medium and send data, while the Physical (PHY) component deals with details of transmission and reception. MAC and PHY are part of a primary architecture 7 layers model for solving problems in communication networks, as developed by the International Organization of Standardization (ISO).

The layer model is called Open System Interconnection (OSI) model, and is shown in Figure 1.2. Beyond the above PHY and MAC layers, the Data Link layer breaks up the input data into data frames – typically a few hundred or a few thousand bytes – and transmits the fames sequentially. The data link layer in the receiver returns acknowledgement frames to confirm correct receipt of each frames. The network layer determines and controls how packets are routed from source to destination. The Transport layer accepts data from the Session Layer and split it up into smaller units that are then passed to the Network Layer. The Session Layer allows users on different machines to establish sessions between them. The Presentation Layer is concerned with the syntax and semantics of the information transmitted. Finally, the Application Layer has a variety of protocols that are commonly needed by users such as HyperText Transfer Protocol (HTTP)[13].

Application Layer	Layer 7
Presentation Layer	Layer 6
Session Layer	Layer 5
Transport Layer	Layer 4
Network Layer	Layer 3
Datalink Layer	Layer 2
Physical Layer	Layer 1

Figure 1.2 : Open System Interconnection (OSI) reference Model

1.2.2 IEEE 802.20 and 3GPP2_UMB Standards Overview

IEEE 802.20 was ratified in 2008 and is a standard offering high-speed, highly reliable and cost-effective broadband communication; the IEEE 802.20 Mobile Broadband Wireless Access (MBWA) Working Group was first established in December 2002. 802.20 was claimed to be a superior and more flexible standard to what was currently offered elsewhere [4], and a clear candidate for providing 4G wireless levels of service. The standard is Internet Protocol (IP) based and provides a broadband packet-based air interface for mobile users with speeds up to 250 km/h [3]. The use of Internet Protocol (IP)-based technologies is a strategic element in the design of many 4G standards including 802.20. 802.20 aim for the production of low-cost, always-on, and mobile broadband wireless networks. Interestingly this standard changed the direction of wireless networking, by coming up with a strategy to sit on existing cellular towers, and provide coverage area the same as that of a mobile phone system but providing data rates and connection techniques that are more equivalent to a Wi-Fi connection [14].

The next system driving OFDM usage for wireless connectivity was the 3GPP2 Ultra Mobile Broadband (UMB), (which in fact was integrated as a core part of 802.20). The standard is designed to provide high speed Frequency Division Duplex (FDD) and Time Division Duplex (TDD) mobile broadband access and optimized for high spectral efficiency and short latencies, using OFDM modulation, link adaptation and multi-antenna communication techniques. Other features in UMB included provision for fast handoff and fast power control. The inter-sector interference management is embedded in the design, which helps to facilitate communication in highly mobile environments [15].

IEEE 802.20 specifies two modes of operation, either wideband mode or a 625k-MC mode while the UMB uses wideband mode only.

The wideband mode is based on Orthogonal Frequency Division Multiple Access (OFDMA) techniques. This standard can use one of two techniques: either the FDD or the TDD. The bandwidths that can be occupied are from 5 MHz to 20 MHz in the case of IEEE 802.20 standard, and from 1.25 MHz to 20 MHz in UMB. The 625k-multicarrier (625k-MC) mode is a TDD air interface mode only. It was developed to obtain maximum benefit from adaptive, multiple-antenna signal processing [4, 16-17].

The OFDM in IEEE 802.20 and UMB is based on a scalable bandwidth. The modulation can change the FFT size based on the bandwidth of transmission and the length of FFT range from 128 points up to 2048 points. The subcarrier spacing is kept constant within different bandwidths by changing the chip rate of each FFT size. There are different lengths of cyclic prefix; Table 1.1 shows the OFDM symbol parameters. The chip rate is the sampling rate that the FFT is required to work with. The maximum sampling rate actually required for both standards is around 20 MHz in 2048 FFT point.

Table 1.1: OFDM Symbol Parameters for IEEE802.20 and UMB Standards

Parameter	NFFT= 128	NFFT= 256	NFFT = 512	NFFT = 1024	NFFT = 2048
Chip Rate Mcps	1.2288	2.4576	4.9152	9.8304	19.6608
Subcarrier Spacing khz	9.6	9.6	9.6	9.6	9.6
Bandwidth of OperationMHz	1.25	1.25-2.5	2.5-5	5-10	10-20
Cyclic Prefix Duration µs	6.51, 13.02, 19.53, or 26.04	6.51, 13.02, 19.53, or 26.04	6.51, 13.02, 19.53, or 26.04	6.51, 13.02, 19.53, or 26.04	6.51, 13.02, 19.53, or 26.04
Guard Interval µs	3.26	3.26	3.26	3.26	3.26
OFDM Symbol Duration µs	113.93, 120.44, 126.95, or 133.46	113.93, 120.44, 126.95, or 133.46	113.93, 120.44, 126.95, or 133.46	113.93, 120.44, 126.95, or 133.46	113.93, 120.44, 126.95, or 133.46

1.2.3 Mobile Broadband with IEEE802.16e (Mobile WiMax)

IEEE 802.16e has the ability to support movement up to 150 km/h and operate in both Line-Of-Sight (LOS) and Non-Line-Of-Sight (NLOS) environments. The IEEE 802.16e air interface is based on Orthogonal Frequency Division Multiple Access (OFDMA), the main aim of which is to achieve better performance in non-line-of-sight environments. In an NLOS link, the transmitted signals arrive at the receiver over multiple reflected paths. OFDM provides efficient means to overcome such challenges of NLOS propagation thought the use of a cyclic prefix in the OFDM symbols, which eliminates inter-symbol interference (ISI) and the resulting added complexities through the need for adaptive equalisation. Because the OFDM waveform is consisted of multiple narrowband orthogonal subcarriers, frequency

selective fading only results on amplitude changes and phase rotations of symbols, which can be corrected by a single-coefficient equaliser per subcarrier [18].

The IEEE 802.16e introduced scalable channel bandwidth up to 20 MHz and using Multiple Input Multiple Output (MIMO) strategies. In mobile WiMAX, the FFT size can vary between 128 and 2048, keeping the subcarrier spacing at 11.16 KHz. Table 1.2 summarises the OFDM symbol parameters [19-20].

Table 1.2: OFDM Symbol Parameter of IEEE 802.16e

FFT length	128	256	1024	2048
Transmission Bandwidth(MHz)	1.25	5	10	20
Subcarrier Spacing(kHz)	11.16	11.16	11.16	11.16
Symbol Duration us	100.8	100.8	100.8	100.8

1.2.4 Long Term Evolution (LTE)

Universal Mobile Telecommunications System (UMTS) Long Term Evolution (LTE) Release 8 (2009) provides improved system capacity and coverage, high peak data rates, low latency, reduced operating costs, multi-antenna support, flexible bandwidth operation and seamless integration with existing systems. The air interface of LTE is based on OFDMA (Orthogonal Frequency Division Multiple Access) and MIMO (Multiple-Input Multiple Output) in downlink (DL) and uses SC-FDMA (Single Carrier Frequency Division Multiple Access) in the uplink (UL) direction. LTE Release 8 also supports scalable bandwidth up to 20 MHz. In the UL SC-FDMA is implemented via Discrete Fourier Transform Spread OFDM (DFT-SOFDM). DFT-SOFDM has similar algorithm structure and implementation to that of the OFDM transmission scheme used on the DL, the main difference being that the constellation symbols are DFT precoded before mapping to the different

subcarriers. A block diagram for SC-FDMA implemented via DFT-SOFDM is shown in Figure 1.3 [21-22].

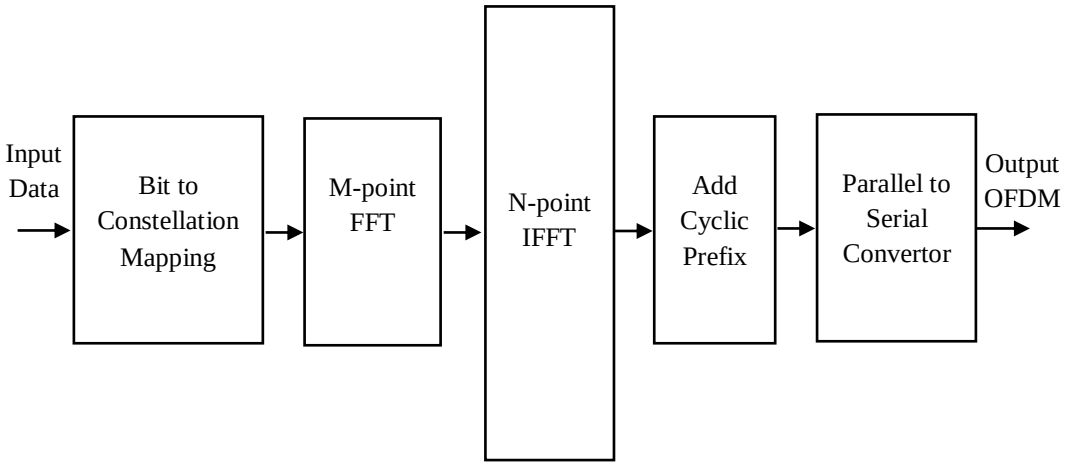

Figure 1.3: Single Carrier Frequency Division Multiple Access Transmitter Block Diagram of LTE Standard

Table 1.3: OFDM Modulation Parameters for LTE Release 8

Transmission Bandwidth MHz	1.25	2.5	5	10	15	20
Sub-carrier spacing KHz	15					
Sampling frequency MHz	1.92	3.84	7.68	15.36	23.04	30.72
FFT size	128	256	512	1024	1536	2048

1.3 Principles of Multicarrier Techniques

In the continually growing world of wireless telecommunications, a number of technology and algorithm trends are gaining widespread popularity in the development of radio PHY layers. In the last few years there has been a clear increase of interest in multi-carrier communications and their application to efficient wireless multiple-access systems development. In particular, there has been a great deal of research and subsequent deployment of OFDM, and MC-CDMA (Multi-Carrier Code Division Multiple Access). Both of these methods are based on orthogonalisation methods and as such require the implementation of Fourier transforms and inverse Fourier transforms [23]. A well known benefit of multicarrier systems is that they can be used provide excellent adaptability to the time and frequency selectivity of radio propagation channels, they allow for simpler narrowband equalization for multipath mitigation, and generally are less susceptible to impulsive noise. Generally these methods can also be structured to provide full and efficient use of available bandwidth.

Although first reviewed and developed in the 1960s OFDM took a number of decades before processing technology was sufficiently fast to allow real time implementation. In the last 10 years with advent of 802.11, 802.16, and LTE to name a few, OFDM is now widely deployed and accepted as the best multicarrier technology for robust and reliable high-rate and high-speed data transmission. Interestingly it is not just found in wireless, but is also used for wired communication systems because of its spectral efficiency, and ability to mitigate the effects of delay spread and inter symbol interference (ISI) [24-25].

1.3.1 Basic OFDM System

To review the concept of OFDM, an incoming data stream, most likely with a high-data rate, enters (D) at the transmitter side as in Figure 1.4. From a hardware perspective this incoming data enters a serial to parallel converter which is used for

mapping the high rate input data stream into *N* lower rate parallel data streams. Sets of pulse shaping filters are used to band limited the spectra of the impulses of the input data (D). The pulse shaping filter form Nyquist systems, which, if correctly synchronised in time and by selection of appropriate subcarrier frequencies, can be overlapped in both time and frequency without causing interference. Each data stream is then placed on its own quadrature carrier $(f_1, f_2, ..., f_N)$. The carrier spacing is carefully selected to ensure orthogonality. The multicarrier modulator generates subcarriers spaced by (1/*T*) Hz. The values of $(f_1, f_2, ..., f_N)$ are as shown in Equations (1.1), (1.2) and (1.3):

$$f_1 = \frac{1}{T} \tag{1.1}$$

$$f_2 = \frac{2}{T} \tag{1.2}$$

$$f_N = \frac{N}{T} \tag{1.3}$$

where *T* is the fundamental period. The orthogonality between carriers is necessary to ensure that carriers can be perfectly separated from each other at the receiver side. The *N* modulated streams are next added together, and the final stage would be a modulation to carrier (or Intermediate Frequency IF) frequency modulated up to the transmit radio carrier frequency (f_c) to output the D_out) signal which would then pass through a (very) high speed DAC (digital to analogue converter) to produce the radio frequency signal. (Note that the DAC is more likely in current systems to include an IF stage (10's of MHz) before the carrier frequency (f_c) modulator (100's of MHz); however in future software defined radio systems we can expect the DAC to be at the RF output stage).

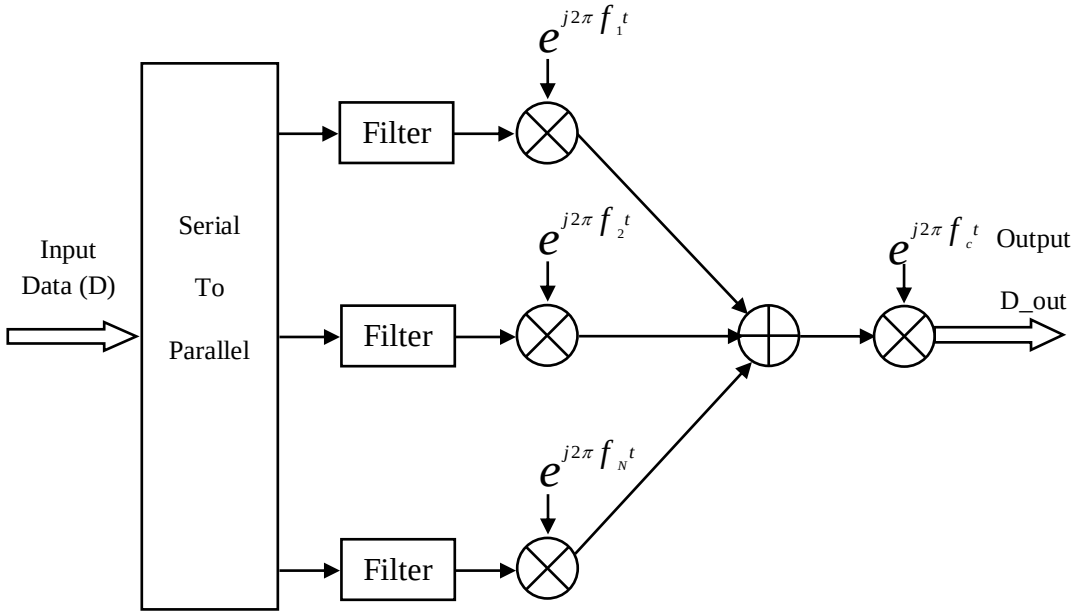

Figure 1.4: Basic OFDM Transmitter

At the receiver side, the incoming OFDM symbol is first returned to baseband by use of a radio frequency mixer at the carrier frequency f_c as shown in the basic OFDM receiver in Figure 1.5. For the next receiver stage, the incoming OFDM signal is separated into its N streams and the N frequency bands and demodulated back to baseband. (Note in these figures of the transceiver, we assume that other necessary receiver stages such as synchronisation and phase locking are appropriately performed.)

After each mixer a low pass filter would be required to remove the higher order demodulation frequencies and leave the original baseband. Thereafter once the data streams have been separated from each other a simple decision device is applied consists of resample, threshold detector and parallel to serial convertor.

In multi-carrier systems, the arrays of sinusoidal generators and coherent demodulators for a large number of channels would clearly be required to be implemented in powerful parallel processing systems and the implementation of such arrays by the traditional techniques using oscillators and modulators/demodulators is computationally very expensive. However a key observation is that the multi-carrier data signal is effectively the inverse Fourier transform of the original serial

data string when the evenly spaced quadrature amplitude modulators (QAM) are used as above. Similarly the bank of coherent demodulators is effectively a Fourier transform computation. This view of the multicarrier QAM system suggests a completely digital modem built by using FFT and its inverse [25] to achieve the modulation (therefore orthogonalisation) of the signal in the baseband. [26].

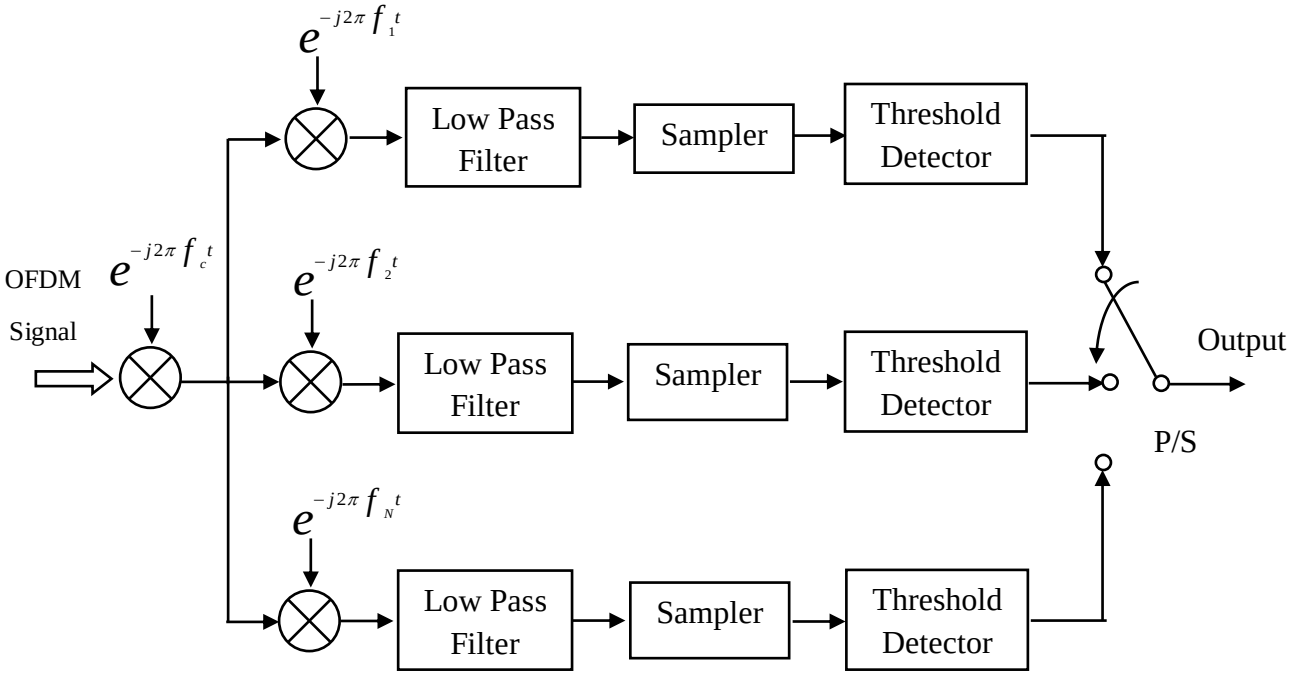

Figure 1.5: Basic OFDM Receiver

Therefore as is well known, the Figure 1.4 and Figure 1.5 implementations effectively show the OFDM transceiver where the baseband modulation or orthogonalisation stages can be greatly simplified by the use of the FFT and the IFFT. Given the availability of high speed processors and fast algorithm implementation the efficiency of implementation through using FFT and its inverse combined with its properties makes the OFDM an attractive modulation and multiplexing technique for fourth generation wireless networks [23].

The idea of using parallel-data communication and Frequency Division Multiplexing (FDM) was developed in the mid-1960s (i.e. OFDM is an idea and technology from

50 years ago!) [27]. In OFDM techniques, parallel data and frequency division multiplexing (FDM) with overlapping sub channels are used and one key benefit is the avoidance of the use of full band (high speed) data equalization; this can now be performed in the smaller frequency bands (or in each FFT bin). The immunity to combat impulsive noise has increased, making it possible to fully utilize the available bandwidth [28]. OFDM is therefore a multicarrier modulation technique which enables robust, high data rate communication over time varying and noise wireless communication channels and in the decade of 2000-2010 its time for deployment has arrived [29].

1.3.2 OFDM Advantages and Disadvantages

OFDM's many advantages over single carrier modulations can be summarized as follows:

1. In OFDM, the band is divided into a number of overlapping frequency channels. This technique results in better use of the available spectrum [30].

2. OFDM is based on orthogonality between the subcarriers. This means that each subcarrier does not interfere with others; thus no guard bands are required.

3. Parallel data systems required complex circuits. The use of FFT and its inverse algorithms eliminates arrays of sinusoidal generators and coherent demodulation. This makes the implementation of the technology cost-effective [28].

Of course OFDM still has many problems that could affect its performance and it will be useful to explain some of these problems before going into more detail and describing the steps of the OFDM generation and demodulation process. (Also to specifically mention one *disadvantage* of OFDM is the cost of implementing the FFT and IFFT in the transceiver – hence the investigation of this thesis.)

One way to address the implementation complexity issues of the Fourier transform and inverse Fourier transform in the OFDM transceiver has been simply achieved by simply using FFT and its inverse rather than the standard DFT – this does of course limit the flexibility of different data lengths that can be used (and largely limits to power of 2 for radix-2 efficient implementation). However other more difficult to resolve issues are identifiable as:

1) *Large peak-to-average ratio (PAR) of the transmitted signal*. Different techniques are used to decrease PAR. These techniques are incorporated to control the resulting nonlinear distortion at the power-amplification stage.

2) *The data symbols are transmitted on subcarriers*. The OFDM transceiver is sensitive to mismatch and Doppler effects of transmit–receive oscillators. This leads to subcarrier frequency offset (CFO).

3) *Uncoded OFDM does not enable the available multipath (or frequency) diversity*. In fact, only diversity order one is possible through multipath Rayleigh fading channels [31].

1.3.3 OFDM Transceiver

The general block diagram for the OFDM transceiver is shown in Figure 1.6. In this block diagram, the Analogy to Digital Convertor (ADC) and the up/down converter have been omitted largely because this work focuses on the baseband implementation of the OFDM transceiver on FPGA, rather than optimising these other very important components.

As discussed in OFDM systems, the FFT and IFFT pair is used to modulate and demodulate the data constellation onto the subcarriers [32] and hence the heart of an OFDM modulator and demodulator consists of the inverse FFT (IFFT) and FFT respectively [33]. The orthogonality between carriers is necessary to ensure that carriers can be perfectly separable one from another at the receiver side and a simple OFDM style spectrum is shown in Figure 1.7 [2].

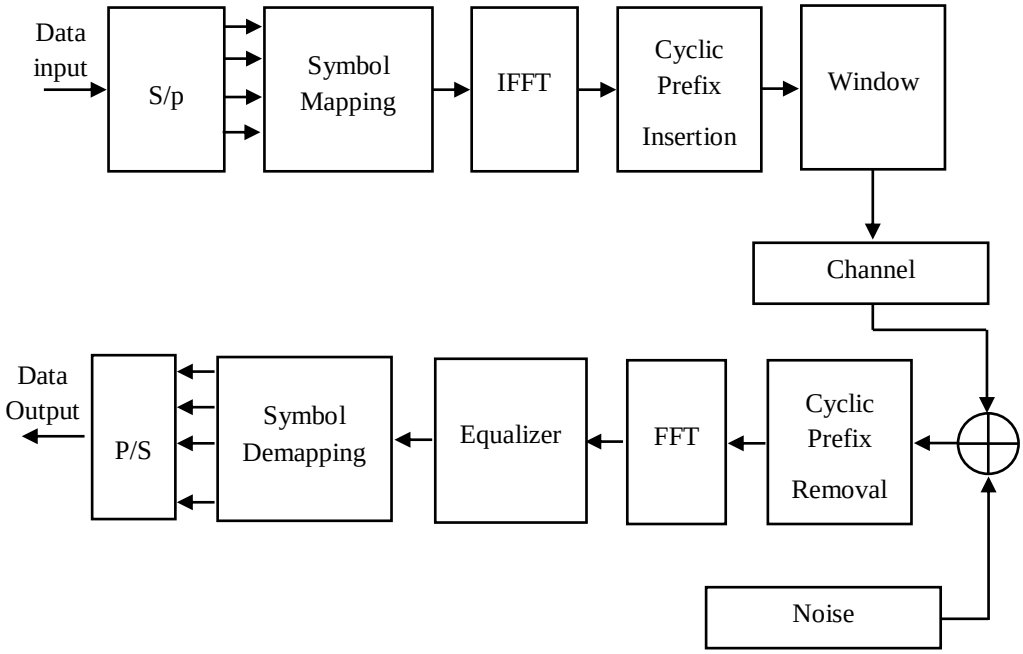

Figure 1.6: OFDM Transceiver Block Diagram

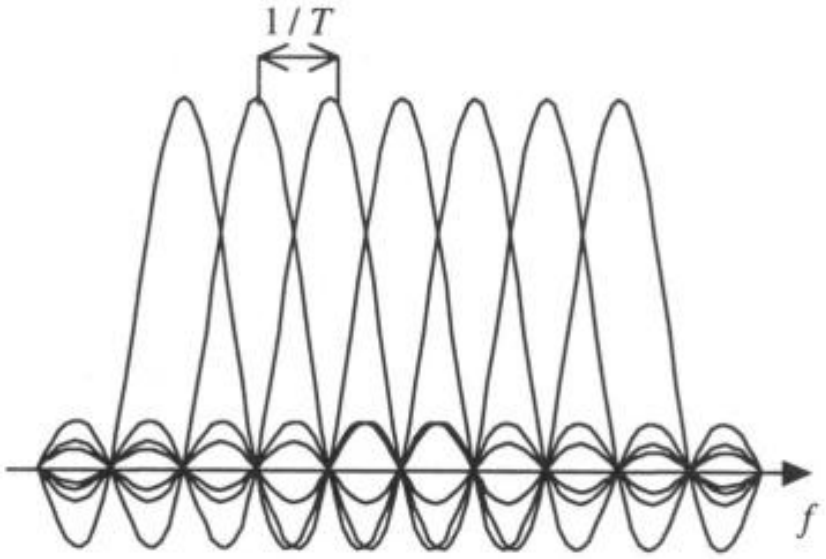

Figure 1.7: Spectrum Overlapped in OFDM [2]

A copy of the last $N_{IFFT} * 1/G$ samples is appended to the beginning of the symbol, called CP, which increases symbol duration so that multipath mitigation can be more likely achieved, (where N_{IFFT} is the IFFT size and G is a carefully chosen positive integer [34]).

After the samples of the symbol have been processed by the IFFT block, to avoid Inter Symbol Interference (ISI) and Inter Channel Interference (ICI), guard period samples must be formed by a cyclic extension of the symbol period. CP insertion is carried out by taking symbol samples from the end of every symbol and appending them to the front of the symbol, as shown in Figure 1.8 [27].

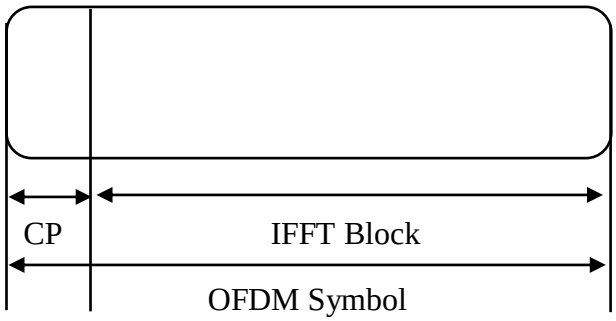

Figure 1.8: OFDM with Cyclic Prefix

1.4 Field Programmable Gate Array Technology

Field-programmable gate arrays (FPGAs) have become an extremely popular implementation technology since they were first introduced in 1989 by Xilinx. Many modern digital communication systems require a combination of high performance, low cost, and flexibility of design and compared to ASICS, this can be afforded by FPGAs (albeit the power consumption is higher and cost per device for high volumes is also higher). FPGAs will support designs at very high clock rates (approaching 1GHz at time of writing), and depending on the actual device selected offer very high levels of parallelism – note the latest and largest FPGA offers more than 2000 (two thousand) multipliers on a single chip, although at a very high per device cost. More affordable devices with perhaps order of 10 or a 100 multipliers per device are also available [35].

In a modern FPGA, the PHY layer design concept is that a single programmable chip can implement a complete transceiver system, i.e. the OFDM, the channelization, the equalisation, synchronisation, framing and so on are all implemented on the FPGA in hardware [36].

FPGAs are now driving DSP and communications in the direction of single chip designs, capable of implementing true software radio. Many digital signal processing algorithms/components, such as FFTs, Finite Impulse Response (FIR) filters, numerically controlled oscillators are very efficiently implemented by the FPGAs. In recent year these algorithms were built with ASICs or parallel DSPs [25-26] for very high volume applications, however as the power consumption of FPGAs drops, we may see the SDR enabling FPGA move from basestation to handset or user device and be capable of using one chip to implement all radio standards. (Note that modern smart phones (in the USA) can contain up to *seven* radios, and therefore *seven* transceiver chips including the likes of GSM/GPRS, cdmaONE, 3GPP, Bluetooth, LTE, 802.11, cdma2000).

FPGA Generic Forms

FPGAs belong to a group of devices called Field Programmable Logic (FPL), defined as programmable devices containing repeated fields of small switch-interconnectable logic blocks and elements. The logic structures of today's FPGAs consists of regular arrays of logic blocks (from a few 100 to a few 10000s) containing small look up tables, registers, multiplexers, and larger segments of the chip containing functionally components of block RAMs, and parallel multiplier/adder.

In 2011 the two largest FPGA vendors are Xilinx and Altera (around 95% of the market); other vendors include Lattice Semiconductor, Actel, Atmel and Achronix. All of these companies offer a wide variety of Intellectual Property (IP) core solutions for communication and DSP applications including FFT, FIR, Convolutional Encoder, Puncture and Depuncture, Viterbi Decoder, Interleaver and Deinterleaver and so on. Furthermore in recent years very easy to use high level block based tools such as Xilinx System Generator or Altera DSP Builder have been introduced [32].

Software Design Environments/Tools for FPGAs

The Xilinx System Generator tool, which runs under the Matlab/Simulink environment, has very obvious use for the simulation and design of an OFDM transmitter and receiver given the design environment and provision of FFT and IFFT blocks. (In this thesis we will refer to and use some of these blocks, however one aim herein is to have a generic FFT block that is reconfigurable for different standards and programmable for different FFT lengths and speeds, hence the need to design the FFT computation from first principles based on parameters extracted from PHY layer radio standards).

Xilinx also provide the ISE environment which we will use to synthesize and download the design to the targeted board, as well as to synthesize the VHDL code of FFT designs. The synthesis stage is a variable in any FPGA design, and synthesising with different vendor's tools may give different implementations and efficiencies. (For example a VHDL design could be targeted at a Xilinx FPGA Virtex 5, with the synthesis done either by say Xilinx XST/ISE or Synopsys Synplify – and one design may be more efficient than the other). However in this thesis all designs will be targeted at the FPGAs using the ISE tool, and the inference is that designs can be compared fairly. Similarly the actual FPGA device chosen has some effect on efficiency, but to address this variability ALL designs will be targeted at a Virtex 5 chip with sufficient hardware resources to easily accommodate all circuits and therefore to minimise any second order cost-effects (such as place and route issues, or resource exhaustion issues).

At the higher level of design, the Simulink tool provides a computation block based design environment for communication systems. It has widespread use in algorithm development, design and verification and the Xilinx System Generator can be used to easily implement PHY layer SDR type designs [37]. However, for lower level efficient implementations VHDL and Verilog languages are used to program the FPGAs. In this thesis, VHDL is the core language used to design first principles efficient, high-speed FFT for the OFDM core of 4G physical layer wireless networks.

1.5 Thesis Objectives and Contributions

This work has two core objectives leading to the two areas of main contribution:

The first is the research to develop efficient, high-speed, optimized Fast Fourier Transform/Inverse Fast Fourier Transform (FFT/IFFT) processors suitable for an array of 4G PHY layer standards implementation. For this a Radix 2 Decimation in Frequency (DIF) algorithm has been chosen and implemented from first principles on an FPGA. The designs are based on two architectures, one on a "butterfly processor" using multiply/adders and the second on a COordinate Rotational DIgital Computer (CORDIC) implemented butterfly. We can show the design offers variable FFT/IFFT size from 128 up to 2048 points (suitable for the entire range of current 4G standards). An optimized butterfly processor (compared to the standard butterfly available from Xilinx and other vendors) has also been introduced which is based around using two multiplier, rather than the more traditional four. To achieve this design two clock techniques have been used to control the calculation inside the FFTs. Both designs have been simulated, synthesized and implemented on the Virtex 5 Xilinx board (Note that the V5 was chosen as the standard reference FPGA to use in this work to give comparative costs). The resource areas, maximum frequency achieved by the designs and immunity to noise have been reported to all architectures.

The second objective is the design, validation and FPGA implementation of a dynamically programmable (i.e. software defined) OFDM transmitter which could form the core of the generic PHY layer of a 4G wireless software defined radio running on an FPGA. The design offers the current array of different types of modulation, such as Quaternary Phase Shift Keying (QPSK), 8PSK, 16 Quadrature Amplitude Modulation (16QAM and 64QAM) and has variable FFT/ IFFT size from 128 up to 2048 with a control circuit to switch between different FFT sizes. The design can generate different cyclic prefix sizes as required by the standards. The thesis will aim to show the efficiency of having a Software Defined Radio (SDR)

type front end PHY for 4G standards and compare to the costs/timings obtained from more standard tools used by other designers.

Therefore we conclude that that research introduces FFT/IFFT/OFDM systems that are novel, programmable, extendable and highly integrable into the physical layer of modern 4G radio systems likely to be seen over the next few years to allow one radio front end for all implemented standards (e.g., the single smart phone running 802.16, 802.11z, LTE, LTE-advanced, 3G, and so on).

1.6 Thesis Outline

This research looks at the implementation and evaluation of FFT algorithms and the physical layers of wireless networks. It is organized into seven chapters as follows:

Chapter1 : serves as an introduction to the thesis. Descriptions of 4G wireless standards and OFDM algorithms and tools for implementation on FPGA are given.

Chapter2 : provides a detailed analysis of the FFT and CORDIC algorithms, and architectures that can be implemented on an FPGA.

Chapter3: reviews previous work, in relation to the implementation of FFT algorithms and the OFDM transceiver for different wireless standards on FPGA.

Chapter 4 : discusses the FFT implementation on an FPGA based on Butterfly architecture.

Chapter 5: provides a detailed analysis and discussion of FFT implementations on FPGAs based on CORDIC.

***Chapter 6* :** describes a dynamic OFDM transmitter implementation on FPGAs.

Chapter 7: Summarises the results, draws conclusions, and highlights paths for further research in this area.

2 Fast Fourier Transform (FFT) and Coordinate Rotation Digital Computer (CORDIC) Algorithms

2.1 Introduction

Generic Digital Signal Processing (DSP) has many widespread applications in many different areas of science and engineering, with digital communications being one of the key fields. Many discrete-time digital communication systems are based on the Fast Fourier Transform (FFT) for analysis, design and implementation [38], and FFTs are widely used in satellites, radars, wideband digital receivers and so on. Whereas many applications have fixed length FFT calculations to implement, a number of authors have developed reconfigurable FFTs for many real-time applications [39]. One important contributor to the OFDM transmitter's low cost is the ability to perform the mapping of an input data stream

to individual subcarriers via the use of an inverse FFT in the transmitter, and by an FFT in the receiver.

This chapter reviews the FFT algorithms and architectures relevant to the research work presented later in this thesis. The Decimation in Frequency (DIF) and Decimation in Time (DIT) variations of the FFT are considered, and common forms of the FFT for hardware implementation are reviewed: Radix-2, Radix-4, Radix-8, and Split Radix. There follows a discussion of hardware implementation issues, such as serialisation and pipelining, as applied to the FFT. As the theme of this research work is a serial version of a Radix-2, DIF form of the FFT, the discussion is focussed accordingly.

Two methods of implementing the FFT are the butterfly technique, where a "butterfly engine" is used to perform the required complex multiplications, and the CORDIC algorithm. The CORDIC algorithm is a shift-and-add technique capable of calculating a variety of trigonometric and other functions, but in this context is used to rotate a vector in the complex plane, effectively performing the complex multiplication in the FFT. Appropriate background on the CORDIC algorithm is presented, including its principles of operation, parameters and modes.

The fundamental aim of this research is to develop fast and efficient FFT structures, and therefore it is useful to review here the metrics by which these aspects are evaluated. In particular, parameters relating to execution speed and resource utilisation are explained, and definitions are also provided for the system clock, throughput, latency and sampling rate.

2.2 Fast Fourier Transform

The Discrete Fourier Transform (DFT) is a key component of many systems used in the fields of engineering and science. Many of these applications employ the DFT for spectral analysis; for instance, signal processing, voice analysis, and data acquisition. In wireless communications, the DFT can also be used to perform modulation and demodulation in multi-carrier systems.

The DFT is a useful and widely used computation, and therefore has received considerable attention from the research community. In particular, algorithms have been developed to minimise the computational complexity of implementing the DFT, and to create efficient hardware structures for real time implementation.

The algorithm introduced by Cooley and Tukey in 1965 is one of the most common methods of realising the DFT [40-41], and achieves a significant computational saving compared to the direct implementation. In fact, the original DFT requires a number of operations proportional to N^2, where N is the number of points in the FFT, whereas the Cooley-Tukey algorithm requires only computations of the order of $N1og_2N$. This optimised calculation is known as the Fast Fourier Transform (FFT) [42].

The inverse Fourier transform can also be implemented using the Cooley-Tukey algorithm, resulting in the Inverse Fast Fourier Transform (IFFT). The computations performed by the FFT and IFFT are very similar, and consequently can be implemented using the same architecture with minor modifications. There are two differences between the FFT and its inverse: firstly, for the FFT the twiddle factor is equal to $e^{-j2\pi/N}$ while for the IFFT it is equal to $e^{j2\pi/N}$, where N is the FFT size. For more details about FFT and DFT fundamentals see [43] and equations below. Secondly, the normalization factor $1/N$. This is shown in Equations (2.1),(2.2),(2.3) and (2.4) [43-44].

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \tag{2.1}$$

$$x(k) = \frac{1}{N} \sum_{n=0}^{N-1} X(n) W_N^{-nk} \tag{2.2}$$

$$W_N^{nk} = \cos\left(\frac{2\pi kn}{N}\right) - j\sin\left(\frac{2\pi kn}{N}\right) \tag{2.3}$$

$$W_N^{-nk} = \cos\left(\frac{2\pi kn}{N}\right) + j\sin\left(\frac{2\pi kn}{N}\right) \tag{2.4}$$

Where *x(n)* is the time domain discrete input signal. *X (K)* is the DFT and it represent the frequency-domain of *x(n). n* represents the discrete-time domain index. *K* is the normalized frequency-domain index.

The Cooley-Tukey algorithm can be implemented as both decimation-in-time and decimation-in-frequency fast algorithms[43] . These variations are considered in the next section.

2.2.1 FFT Decimations

The FFT algorithm has two types of decimation, the Decimation in Frequency (DIF) and the Decimation in Time (DIT). The DIF has some advantages over DIT and in particular for issues related to finite word length effects. A truncation noise is necessarily introduced by the multiplication when implemented with fixed point.

The word length of the input data grows during the calculation of the FFT. The butterfly calculation includes complex multiplication, addition and subtraction. Adding /Subtracting two *N* bits numbers will results in up to *N+1* bits word length. Multiplying two *N* bits numbers can produce up to *2N* bits number. The bit growth occurs through all stages of the FFT. An un-scaled strategy is used to control the word length grows inside the FFT. A growth of one bit is accounted for at the output of each stage, with the remainder truncated (i.e. after the multiplication of two *N* bit numbers, a *2N* bit product is obtained but truncated back to *N* bits). Using the DIF slightly reduces the truncation noise and the complexity of the whole system. Note that the butterfly topology in the DIF sets the truncation stage to be after one of the butterfly outputs while in the DIT the two butterfly outputs are affected by the truncation noise as shown in Figure 2.1 and Figure 2.2 [45]For this reason, in this work, the DIF algorithm is chosen. In the two main sequential architectures that have been designed and investigated in this thesis both are based on the DIF butterfly engine.

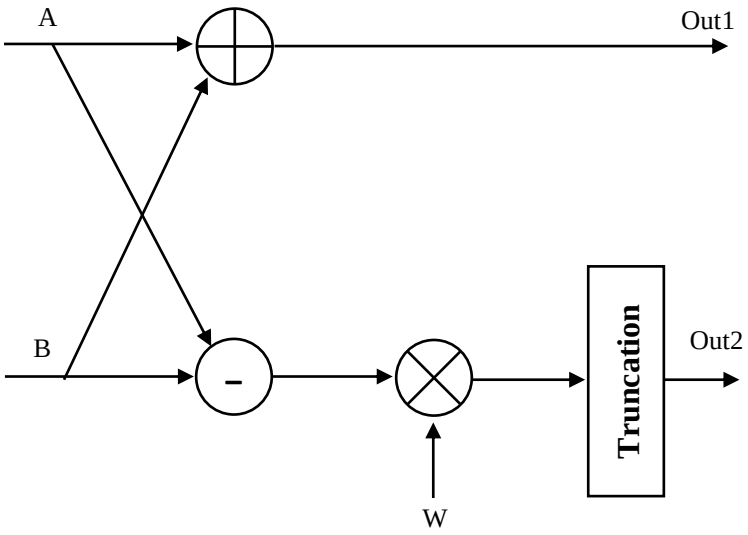

Figure 2.1: DIF Butterfly Topology

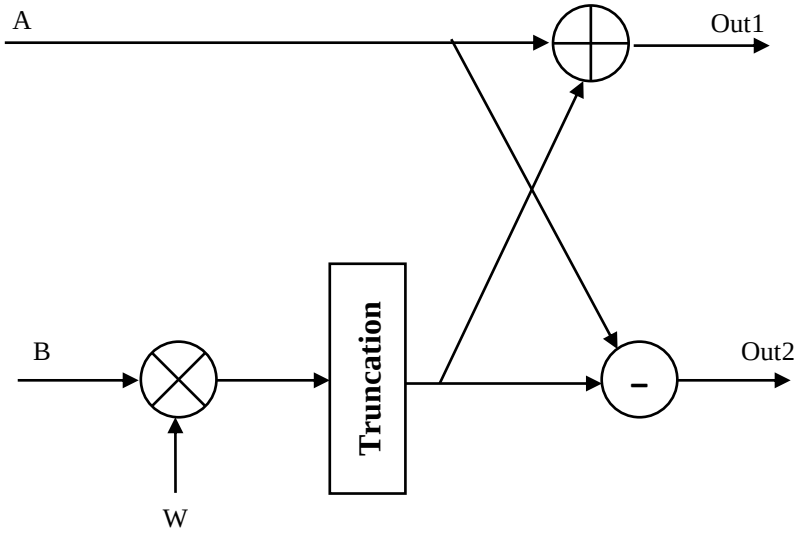

Figure 2.2: DIT butterfly Topology

To derive the classic FFT algorithm, the standard DFT is divided into sequences of smaller DFTs [38], and this decomposition can be based on either time or frequency, thus yielding the decimation-in-time or decimation-in-frequency. In this section, decimation-in-frequency (DIF) and decimation-in-time (DIT) are reviewed as applied to the Radix-2 FFT; other radix algorithms can implemented, along with prime factor methods, however in this thesis, to derive a pragmatic, programmable and implementable design we focus on radix-2.

Decimation in Frequency – DIF

The DIF form of the FFT separates the *N*-sample input sequence into two arrays, each of length *N*/2: the first consists of the first *N*/2 data samples, while the other comprises the last *N*/2 data samples. The calculation is then performed in two sections, as shown in Equation (2.5).

$$X(k) = \sum_{n=0}^{N/2-1} x(n) W_N^{kn} + W_N^{Nk/2} \sum_{n=0}^{N/2-1} x\left(n + \frac{N}{2}\right) W_N^{kn} \qquad (2.5)$$

$W_N^{Nk/2}$ can be expressed as $(-1)^k$ since $e^{-j2\pi kN/2N} = (\cos(\pi) - j*\sin(\pi))^k = (-1)^k$ as in Equation (2.6):

$$X(k) = \sum_{n=0}^{N/2-1} \left(x(n) + (-1)^k x\left(n + \frac{N}{2}\right) \right) W_N^{kn} \qquad (2.6)$$

At this point, *X(k)* can be decomposed into even and odd-numbered samples. The altered expressions can be seen in Equations (2.7) and (2.8) for the general case of an *N*-point FFT. *X(k)* of Equation (2.6) can be split into even and odd samples as shown in Equations (2.7) and (2.8). This is now two *N*/2 point DFTs and this can be

repeatedly decimated until there is only a 2 point DFT. This is illustrated in the flow graph below.

Figure 2.3 provides a signal flow graph to illustrate how this DIF approach would be applied to an 8-point FFT [46]. The computation of the 8-point FFT is divided into three stages, and the input samples to stage 1 in Figure 2.3 are in normal order [*x(0), x(1), x(2).....x(7)*].

In stage one, each sample is processed with its *N/2* sample, i.e., 0, 4, 1, 5 and so on. In the second stage each sample from the previous stage is processed with its *N/4* offset sample and in the final stage these are processed with the *N/8* offset samples. The output of the final stage is reordered.

$$X(2k) = \sum_{n=0}^{N/2-1} \left(x(n) + x\left(n + \frac{N}{2} \right) \right) W_{N/2}^{kn}, \quad 0 \le k \le \frac{N}{2} - 1 \tag{2.7}$$

$$X(2k+1) = \sum_{n=0}^{N/2-1} \left(x(n) - x\left(n + \frac{N}{2} \right) \right) W_{N/2}^{kn} \, W_N^n \ , \quad 0 \le k \le \frac{N}{2} \text{-} 1 \tag{2.8}$$

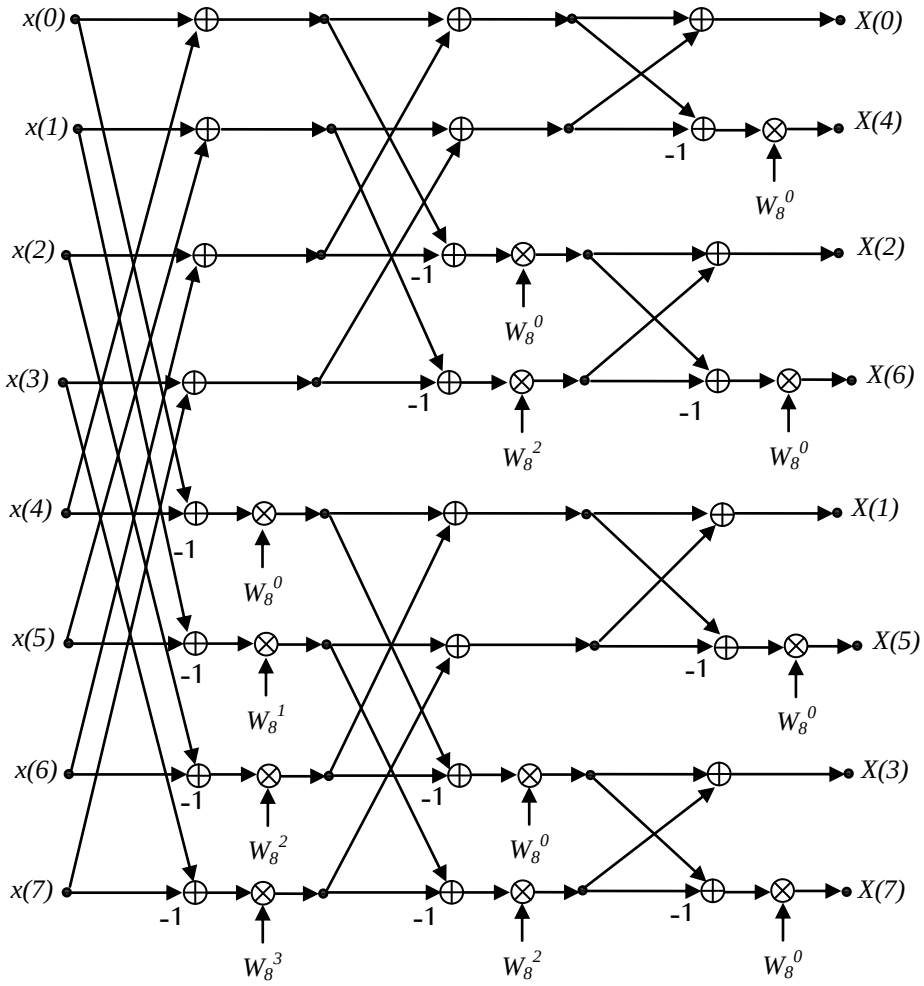

Figure 2.3: 8-point Decimation in Frequency Algorithm

Decimation in Time (DIT):

In the DIT form of the FFT, the *N*-sample window of the input signal $x(n)$ is also divided into two equal sections, but by a different method than the DIF form. In the DIT form of the FFT algorithm, the first section comprises the odd indexed samples and the second the even indexed samples. This is shown by Equations (2.9), (2.10), (2.11) and (2.12), where *x1(r)* represents the even indexed samples of $x(n)$, and *x2(r)* represents the odd indexed samples [47].

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{Kn}, \qquad 0 \le \mathrm{K} \le N-1 \tag{2.9}$$

$$X(K) = \sum_{n(even)=0}^{N-1} x1(n) W_N^{Kn} + \sum_{n(odd)=0}^{N-1} x2(n) W_N^{Kn} \tag{2.10}$$

$$\begin{pmatrix} x1(r) = x(2r) \\ x2(r) = x(2r+1) \end{pmatrix}, \mathrm{r} = 0,1,.....N/2-1 \tag{2.11}$$

$$X(K) = \sum_{r=0}^{(N/2)-1} x(2r) W_N^{2Kr} + \sum_{r=0}^{(N/2)-1} x(2r+1) W_N^{K(2r+1)} \tag{2.12}$$

In Equation (2.13), the new index *m* is created. A signal flow graph of an 8-

$$X(K) = \sum_{m=0}^{(N/2)-1} x1(2m) W_N^{2Km} + W_N^K \sum_{m=0}^{(N/2)-1} x2(2m+1) W_N^{2Km} \tag{2.13}$$

point DIT FFT is shown in Figure 2.4.

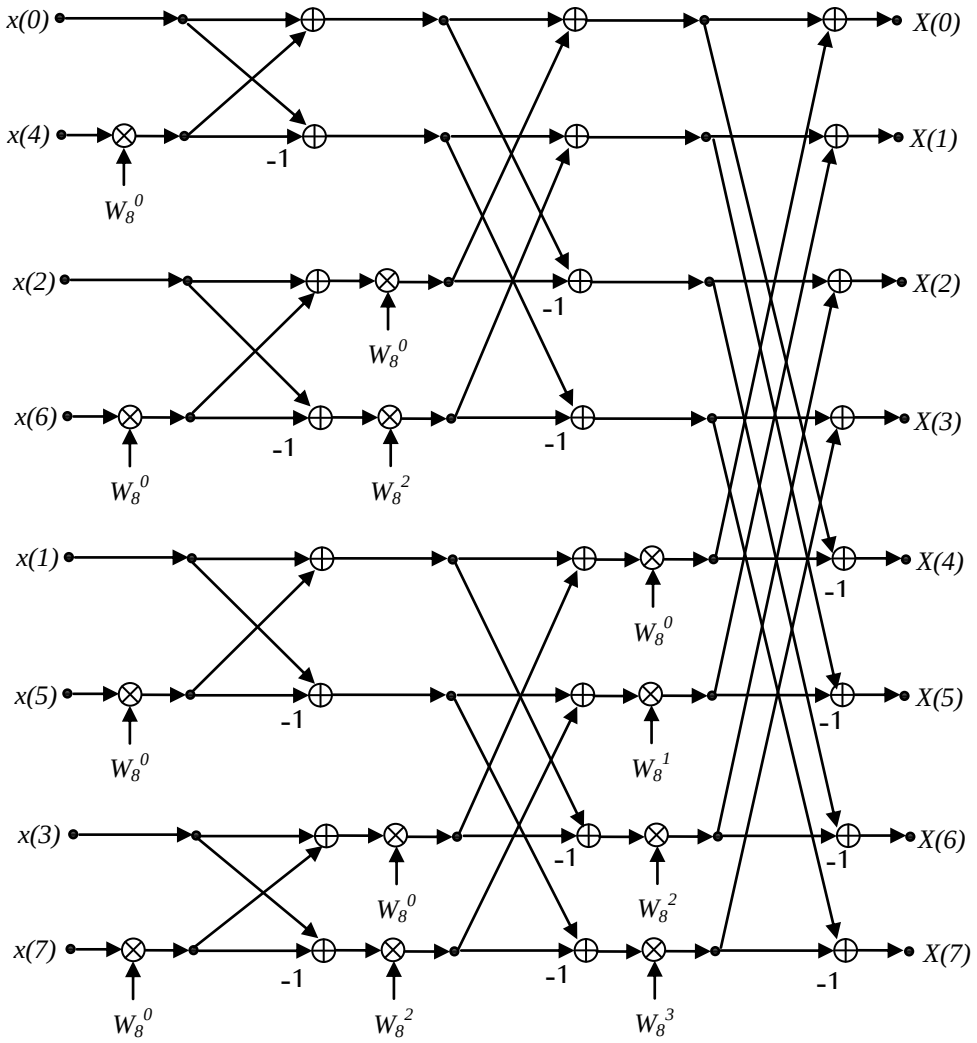

Figure 2.4: 8-point Decimation in Time Algorithm

In summary, the DIF and DIT algorithms calculate the same result**,** but by different methods. In the DIF FFT, the output is calculated by splitting the output sequence *X(n)* into odd and even indexed elements, while in DIT, FFT algorithms are obtained by splitting the input sequence $x(n)$ into odd and even indexed elements. FFT reorders the data from normal to bit reversed order. This is shown in Table 2.1 [38, 48].

Table 2.1: 8-Point Bit Reversed Order

Decimal	Binary	Bit reverse	Decimal
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

2.2.2 FFT Radices

For a 2^n-point DFT, there are several possible ways of implementing the algorithm as an FFT, using the butterfly method. These are Radix-2, Radix-4, Radix-8, and Radix-16 and Split-Radix algorithms [49]. Based on the butterfly size, the FFT is divided into a collection of smaller DFT points. Two points in Radix-2, four points in Radix -4, sixteen points for Radix-16 and four points for Split-Radix. Figure 2.3 and Figure 2.4 shows an FFT implementation based on Radix-2. The FFT output is calculated by using a number of 2-point DFT called a Butterfly. The minimum number of point that the FFT is split into represents the radix. Radix-2 has the

smallest butterfly unit size, and is the most flexible method of implementing the FFT. The total number of arithmetic operations can be reduced by using another radix, but this increases the complexity of the architecture and reduces its flexibility [50].

In the Radix-4 algorithm, the *N*-point DFT is decomposed into four *N*/4-point DFTs, each of which are then broken down into smaller point DFTs, as shown in Equation (2.14).

$$X(K) = \sum_{n=0}^{N-1} x(n) W_N^{Kn} = \sum_{n=0}^{N/4-1} x(n) W_N^{Kn} + \sum_{n=N/4}^{N/2-1} x(n) W_N^{Kn} + \sum_{n=N/2}^{3N/4-1} x(n)\, W_N^{Kn} + \sum_{n=3N/4}^{N-1} x(n) W_N^{Kn} \tag{2.14}$$

According to the periodicity and symmetry properties of the twiddle factor, Equation (2.14) can be rewritten as shown in Equation (2.15), where the summation is performed over the index range from 0 to *N*/4-1. The data segment *x(n)* is changed to meet the same range of Equation (2.14) to be *x(n)*, *x(n+N/4)*, *x(n+N/2) and x(n+3N/4)*, which also affects the twiddle factor.

$$X(K) = \sum_{n=0}^{N/4-1} [x(n) + (-j)^K x(n + \frac{N}{4}) + (-1)^K x(n + \frac{N}{2}) + (j)^K x(n + \frac{3N}{4})] W_N^{Kn} \tag{2.15}$$

The twiddle factor in Equation (2.15) depends on *N*. By dividing *K* into four subgroups, *K*= *4m*, *K*= 4*m*+1, *K*= 4*m*+2, and *K*= 4*m*+3, where *m* = 0, 1, 2, ..., *N*/4-1, the Radix-4 butterfly operation is represented by Equations (2.16), (2.17), (2.18) and (2.19) [41, 51-52].

$$X(4m) = \sum_{n=0}^{N/4-1} [x(n) + x(n+\frac{N}{4}) + x(n+\frac{N}{2}) + x(n+\frac{3N}{4})] W_{N/4}^{mn} \quad (2.16)$$

$$X(4m+1) = \sum_{n=0}^{N/4-1} [x(n) - jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) + jx(n+\frac{3N}{4})] W_N^n W_{N/4}^{mn} \quad (2.17)$$

$$X(4m+2) = \sum_{n=0}^{N/4-1} [x(n) - x(n+\frac{N}{4}) + x(n+\frac{N}{2}) - x(n+\frac{3N}{4})] W_N^{2n} W_{N/4}^{mn} \quad (2.18)$$

$$X(4m+3) = \sum_{n=0}^{N/4-1} [x(n) + jx(n+\frac{N}{4}) - x(n+\frac{N}{2}) - jx(n+\frac{3N}{4})] W_N^{3n} W_{N/4}^{mn} \quad (2.19)$$

The Split-Radix FFT is a combination of two FFT radices: Radix-2 and Radix-4. Radix-2 is used to obtain the even DFT results of *N* inputs, whereby it is found that in the Radix-2 DIF the computation of even samples is independently of odd samples, while Radix-4 is used to compute the DFT of odd samples. To reduce the number of computations, a Split Radix uses both Radix-2 and Radix-4 decomposition in the same FFT algorithm. The Split-Radix algorithm is appropriate for software design, but has disadvantages for hardware design [43, 49].

The Radix-2 and Radix-4 implementations of the FFT are widely used in hardware design due to their relative simplicity. Radix-8 has the advantage of fewer multiplications and reduced memory access, resulting in lower power consumption, but at the expense of a more complex butterfly element and control unit [49].

2.2.3 FFT Architectures

Two popular styles of hardware architecture are used to implement the FFT for OFDM systems: the pipeline-based architecture, and the sequential architecture [53] Implementation of the pipelined architecture requires more hardware resources than the equivalent sequential architecture. The pipeline architecture uses a single

butterfly in each stage. It is not fully parallel. The fully parallel architecture is very expensive as it needs *N/2* butterflies in each stage. The sequential architecture uses a single butterfly unit, with one or two memory units to store the results computed by the butterfly, and a fixed value memory (or ROM) is required to store the twiddle factor coefficients. In the sequential architecture, the butterfly unit is time shared and thus the hardware cost of implementation is lower, but the throughput is reduced accordingly [54].

Even assuming that a sequential architecture is chosen, there are variations possible on the detailed implementation of the FFT processor, for example the choice of adopting a single or a double memory. The memory is used to store input, output and intermediate values calculated during computing the FFT. Single RAM offer area more than two RAM design but slower in speed as it need double clocks to complete each butterfly output [55]. A block diagram of a sequential architecture is shown in Figure 2.5.

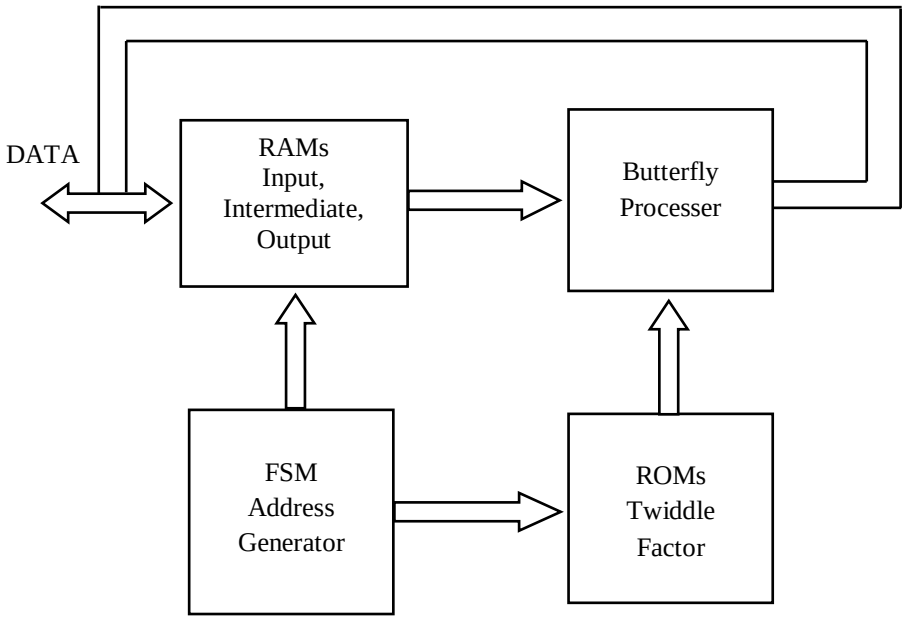

Figure 2.5: Sequential FFT Architecture Using Single Butterfly

Next, some pipeline architectures are introduced. The Radix-2 Multi-path Delay Commutator (R2MDC) is a straightforward method of implementing the Radix-2 FFT algorithm using a pipelined architecture. The data stream is divided into two parts. A delay line is used between stages in order to manage the butterfly processing in the correct order, as shown in Figure 2.6 [56-57]. As shown for an 8-point FFT, it is required a butterfly in each stage so that gives three butterflies. To manage the orders of input samples to each Butterfly, a delay register and a commutative switch are required, with system block separated by a one clock period delay.

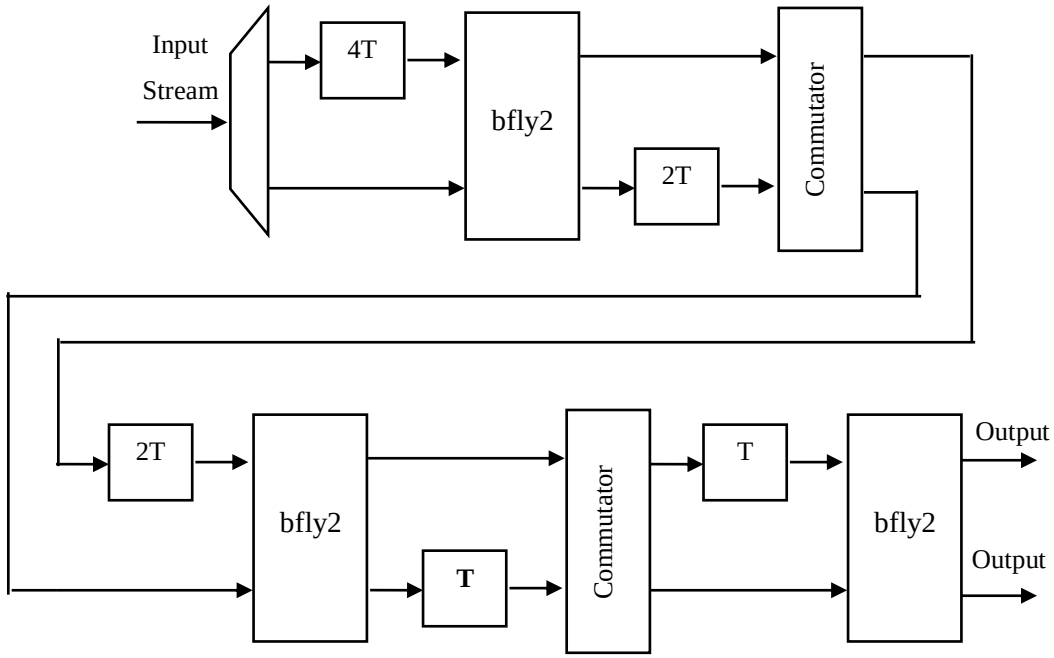

Figure 2.6: 8-point R2MDC

Another pipeline architecture called Radix-2 Single-path Delay Feedback (R2SDF) stores one output of each butterfly in a feedback shift register. The block diagram of R2SDF architecture is shown in Figure 2.7 [58].

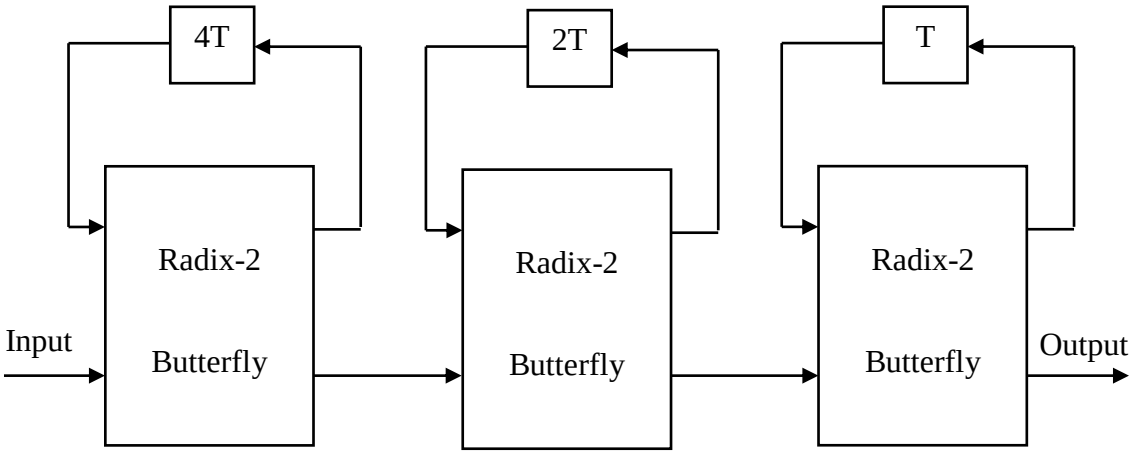

Figure 2.7: R2SDF Architecture

2.3 Coordinate Rotational Digital Computer (CORDIC)

The Coordinate Rotational Digital Computer (CORDIC) algorithm was proposed by J.E. Volder [59] as a useful and flexible approach to evaluate mathematical (and mainly trigonometric) functions. The CORDIC is capable of performing divisions, square roots, trigonometric functions (cosine and sine), and inverse-trigonometric functions (inverse tangent) [60]. The core implementation operations are based on the rotation of a vector using only additions and shifts operation [61], which makes CORDIC very suitable for simple hardware realisation for both ASIC and FPGA implementations [62]. For implementation of the FFT and IFFT the requirement to calculate sine and cosine values can be effectively performed by the CORDIC instead of using a look-up table, or using some other series expansion calculation of sine and cosine. Hence, the CORDIC is an important algorithm in the context of FFTs and IFFTs.

The generic CORDIC (Figure 2.8) has three input ports, X_{in}, Y_{in}, and $Z_{in,}$ and three output ports, X_{out}, Y_{out} and Z_{out}. X and Y for input and output represent the vector in two dimensions while Z represents an angle. CORDIC has two operating modes, namely the rotation mode and the vectoring mode. In rotation mode, X_{in}, Y_{in} is the

initial vector location in the Cartesian plane and Z_{in} is the angle by which it requires to be rotated. In vectoring mode, the X_{in}, Y_{in} vector is rotated to the x-axis by choosing rotations to drive the Y_{in} value to zero.

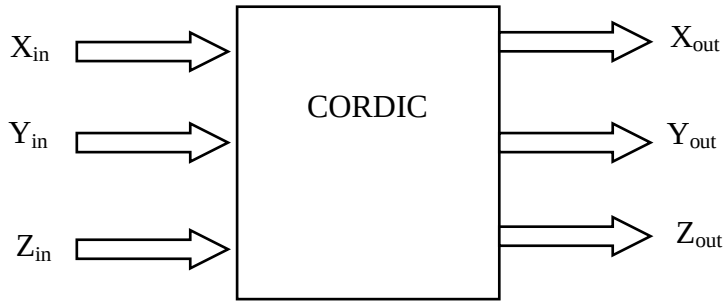

Figure 2.8: General CORDIC Block Diagram

In the FFT, the rotation mode CORDIC is used to implement the twiddle factor multiplication, where this is equivalent to a rotation of a 2-D input vector by the twiddle factor phase.

CORDIC algorithms are very suitable for twiddle factor multiplication and an FFT based on CORDIC can be constructed virtually free of multipliers [62]. In this research, we will show in later chapters how the parallel CORDIC architecture has been used to implement the multiplication within the FFT.

The CORDIC algorithm is based on the principles of two-dimensional geometry [63], and was first published by Volder [59] in 1959 as a procedure for efficiently implementing trigonometric functions, and was subsequently extended by Walther [64] to compute other functions, including multiplication, division, square root, and logarithmic and hyperbolic functions [65]. CORDIC is also capable of performing polar-to-Cartesian and Cartesian-to-polar coordinate system conversions.

The method of CORDIC is to iterate through a set of progressively smaller vector rotations towards an arbitrary angle. This is achieved using a series of shift and add operations, which can be implemented in hardware at low cost [66]. The CORDIC algorithm is therefore very suitable for implementation on FPGAs [67].

The CORDIC technique can play an important role in OFDM systems and can be found in a number of places. For example it can be used to determine and compensate for frequency offset, on calculate the division in the channel estimation stage. The linear vectoring mode is used to calculate the division in the channel estimation stage and as progressed in this document it can perform IFFT and FFT butterfly computations to modulate and demodulate data onto the OFDM subcarriers [68].

2.3.1 CORDIC Algorithm

The CORDIC algorithm has two modes of operation: the Rotation and Vectoring modes. It can also be used in three different coordinate systems: linear, circular, and hyperbolic [60]. In this thesis Rotation mode circular coordinate CORDIC is used to calculate Twiddle factor multiplication.

The CORDIC algorithm is an iterative procedure requiring simple arithmetic operations. Additions and binary shifting are the main operations required to implement the algorithm. Thus, it has no direct multiplications (other than the final scaling stage multiplier), nor does it have explicit square roots or divides [69].

The CORDIC algorithm achieves vector rotation by an arbitrary angle using a series of *m* micro-rotations by basic angles [70]. At every iteration (indexed by *i*, where i = 0, 1, 2… m-1), the vector is rotated by the angle $arctan(2^{-i})$. A small angular error remains at the end of the series of micro-rotations, but this approaches zero as $m \rightarrow \infty$ [71].

In Rotation mode, the original vector and desired angle of rotation are provided as inputs to the CORDIC processor, which then rotates the original vector through the specified angle to a new position. This forms the primary output of the CORDIC unit.

In Vectoring mode, the CORDIC processor rotates the input vector towards the X axis. The magnitude and angular position of the original vector are computed, and these form the outputs.

To demonstrate CORDIC, consider the example depicted in Figure 2.9. Here, the vector V_1 (X_1,Y_1) is rotated by an angle Φ, from its original position α, to obtain the

new vector V_2 (X_2,Y_2). To achieve this, the total rotation is divided to several micro-rotations, each of the fixed angle *arctan*(2^{-i}). These angles of rotation are stored in a ROM, and subtracted from or added to the accumulated angle of rotation, θ, through the series of rotations [61, 72]. The angular inputs and outputs from the CORDIC unit are denoted by the symbol *Z*.

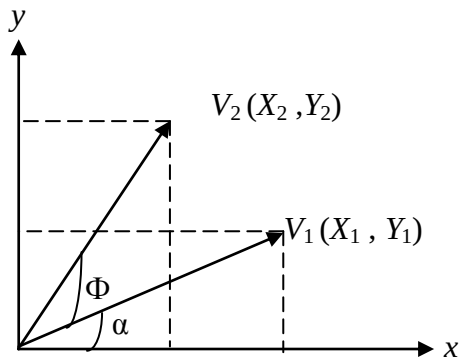

Figure 2.9 : Vector Rotation

The new vector position can be calculated using trigonometry, as in Equations (2.20), (2.21), (2.22), (2.23), (2.24) and (2.25).

$$X_{i+1} = r\cos(\alpha+\phi) = r(\cos\alpha\cos\phi - \sin\alpha\sin\phi) \quad (2.20)$$

$$X_{i+1} = X_i.\cos\phi - Y_i.\sin\phi \quad (2.21)$$

$$Y_{i+1} = r\sin(\alpha+\phi) = r(\sin\alpha\cos\phi + \cos\alpha\sin\phi) \quad (2.22)$$

$$Y_{i+1} = Y_i.\cos\phi + X_i.\sin\phi \quad (2.23)$$

The key of the CORDIC algorithm is to restrict rotations to angles of ϕ, where $\tan^{-1}\phi = 2^{-i}$. The tan function is generated due to taking $\cos\phi$ as common as shown in Equations below.

$$X_{i+1} = \cos\phi(X_i - Y_i \cdot 2^{-i}) \tag{2.24}$$

$$Y_{i+1} = \cos\phi(Y_i + X_i \cdot 2^{-i}) \tag{2.25}$$

By omitting the cosϕ the computation by simple and this can be equalized by multiplying the final output value by scale value to make the rotation correct.

$$X_{i+1} = (X_i - Y_i \cdot d_i \cdot 2^{-i}) \tag{2.26}$$

$$Y_{i+1} = (Y_i + X_i \cdot d_i \cdot 2^{-i}) \tag{2.27}$$

The value d_i is the direction of rotation, chosen to be either +1 or -1 as appropriate to the required direction of rotation.

$$Z_{i+1} = (Z_i - d_i \cdot \arctan 2^{-i}) \tag{2.28}$$

$$d_i = \begin{cases} -1 & \text{if } Z_i \le 0 \\ +1 & \text{if } Z_i \ge 0 \end{cases} \tag{2.29}$$

The CORDIC equations can be expressed in a generalised form applicable to any of the three coordinate systems, as given by Equations (2.30), (2.31) and (2.32). Note that the values of e and μ are chosen according to the desired coordinate system.

$$X_{i+1} = (X_i - \mu d_i \cdot 2^{-i} \cdot Y^i) \tag{2.30}$$

$$Y_{i+1} = (Y_i + d_i 2^{-i} X_i) \tag{2.31}$$

$$Z_{i+1} = Z_i - d_i e^i \tag{2.32}$$

For circular coordinates:

$$\mu = 1 \quad , \quad e^i = \tan^{-1} 2^{-i} \tag{2.33}$$

For linear coordinates:

$$u = 0 \quad , \quad e^i = 2^{-i} \tag{2.34}$$

For hyperbolic coordinates:

$$\mu = -1 \quad , \; e^i = \tanh^{-1} 2^{-i} \tag{2.35}$$

In this thesis only one coordinate system of CORDIC will be used, namely the circular coordinates – hence no further review of hyperbolic and linear will be presented.

2.3.2 CORDIC Errors

The CORDIC algorithm is subject to two sources of error, and these can be analysed mathematically [73]. The first is due to the finite number of CORDIC iterations, which leave a residual angular error after the last iteration; this error is referred to as the (angle) approximation error. The second source of error is the finite-precision arithmetic used to represent the signals, which results in a conventional rounding error [73]. These two errors are related given that whereas each additional rotation will input more angle accuracy, each angle is represented to a finite number of binary digits, and as such introduced a low level of round-off noise.

2.3.3 CORDIC Based FFT

The FFT algorithm can be implemented using the CORDIC technique to calculate the butterfly processor outputs, rather than the direct implementation of the butterfly processor using appropriate arithmetic and stored cosine and sine values [74]. To explain this, the Radix-2 butterfly decimation in frequency (DIF) processor is considered below.

One of the motivations for using the CORDIC method is that, for large FFT sizes, considerable memory is required for storing the twiddle factors. For every multiplication by a twiddle factor, the multipliers need two values to be stored, one for the real term and the other for the imaginary term of the twiddle factor. To avoid the need for large amounts of memory in larger FFTs, a complex multiplier based on the CORDIC algorithm can be employed [75]. Although this thesis has a focus on implementing FFTs on FPGAs (which invariably has available memory), there are also other semi-custom technologies (from companies like e-ASIC [76]) which have no direct block RAM available, and either function with distributed memory, or would favour CORDIC type strategies to calculate twiddle factors on-line rather than storing in memory.

The Radix-2 butterfly processor is shown in Figure 2.10, where $A_r + jA_i$, and $B_r + jB_i$, are the two complex inputs to the butterfly; and $C_r + jC_i$, and $D_r + jD_i$, are the two complex outputs of the butterfly; $W_r + jW_i$ is the twiddle factor; and $X_r + jX_i$ is the output of the subtractor.

To perform multiplication by the twiddle factor in real time, two ROMs are used to store the twiddle factors, while four multipliers and two adders are required to perform the arithmetic.

In an FFT processor based on CORDIC, the multiplication of $X_r + jX_i$ by the twiddle factor can be implemented using CORDIC. The twiddle factor multiplication is the rotation of a 2-D vector $(X_r + jX_i)$ by the phase of the twiddle factor, which is given by $arctan\ (W_i/W_r)$ [62].

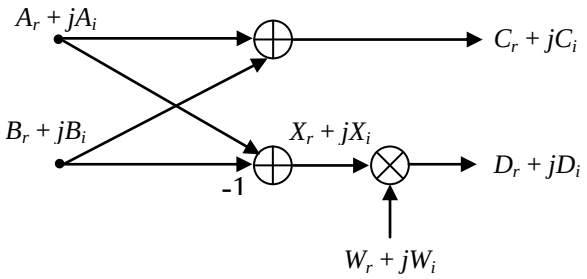

Figure 2.10: Radix-2 Butterfly Flow graph

In the FFT, the twiddle factor multiplication by a complex vector can be expressed in matrix form as shown in Equation (2.36), where X and W represent two complex vectors, of which W is the twiddle factor, and θ is the phase of the twiddle factor.

$$X \times W = \begin{pmatrix} X_{re} \\ X_{im} \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{2.36}$$

$$X \times W = \begin{pmatrix} X_{re} \\ X_{im} \end{pmatrix} \prod_{q=0}^{m} \cos\theta_q \prod \begin{pmatrix} 1 & -\tan\theta_q \\ \tan\theta_q & 1 \end{pmatrix} \tag{2.37}$$

CORDIC can implement Equation (2.36) in the usual way, by implementing a series of rotations of fixed angles to accomplish the overall rotation of the angle θ, which corresponds to the phase of the twiddle factor. The angle θ is divided to sub-angles θ_q. S_q represents the direction of summation as either addition or subtraction. The substitution of Equation (2.39) into (2.37) gives Equation (2.38). For simplicity the multiplication of the cosine term is replaced by 0.6073 constant as shown in Equation (2.38) assuming the number of iterations are ∞. For iterations less than ∞ which is the practical case this constant can be calculated [77].

$$X \times W = 0.6073 \begin{pmatrix} X_{re} \\ X_{im} \end{pmatrix} \prod_{q=0}^{m} \begin{pmatrix} 1 & -S_q 2^{-q} \\ S_q 2^{-q} & 1 \end{pmatrix} \tag{2.38}$$

$$\theta = \sum_{q=0}^{m} S_q . \theta_q = \sum_{q=0}^{m} S_q . \arctan(2^{-q}) \qquad (2.39)$$

2.4 The Fundamental DSP System - Definitions

The general block diagram of a digital signal processing (DSP) system is shown in Figure 2.11. In this section we will review some of the terminology used later in the thesis (on sampling rates, clock rates, quantisation, throughput etc). The signal is converted to a digital signal using an analogue-to-digital (A/D) converter. It is processed in a DSP system, with an FPGA or DSP processor, before being converted back to an analogue signal [78].

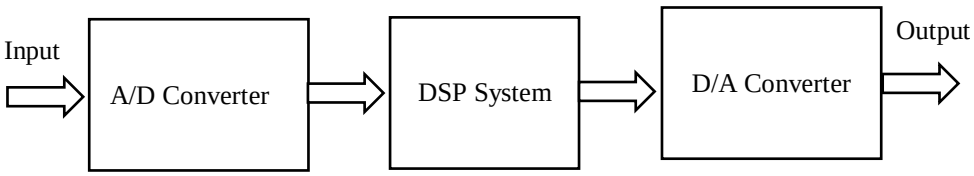

Figure 2.11 : Basic DSP System

The A/D converter includes two steps: *sampling* and *quantisation*. In the sampling step, the analogue signal is sampled every T_s seconds. The sampling rate f_s is defined in Equation (2.40) and expressed in units of Hertz (or samples per second).

$$f_s = \frac{1}{T_s} \qquad (2.40)$$

Quantisation is the process of representing the sampled value by *B*-bits. An appropriate choice of sampling rate is determined by the Nyquist sampling theorem. When implementing a DSP system, the designer needs to ensure that

samples are processed at the correct sampling rate, in order to implement the algorithm accurately [78-79].

Another term that represents speed is *throughput*. This refers to the amount of data that is processed per clock cycle, and is expressed in samples per second [80]. The term "throughput" is similar to "sampling rate", but is usually used when referring to the processing undertaken by a particular component of a DSP system. Finally, the *clock rate* is the operating speed of the system implementation, which may be greater than the sampling rate [78].

The best way to reduce the area of a design is to roll up the pipeline area to share logic resources. Sharing logic resources often requires special control circuitry. In the FFT design, single butterfly has been pipelined and shared many times to complete the calculation. Pipeline technique is used to increase throughput and achieve maximum performance. When the loop is unrolled to create a pipeline, more resource area is required to hold intermediate values and replicate computational structures that need to run in parallel [80].

When implementing an FFT processor on FPGA, a common approach is to share a single butterfly structure over time, hence considering the "processor" to be a butterfly, then sequentially use the "processor" to implement each butterfly in the overall FFT. Additionally in this sequential structure, a two-port RAM is used to store the intermediate data computed , and a memory is required to store the twiddle factors (if not being calculated), and an address generator and control logic are required. For computing a large number of point FFT , it is clear that the area or cost of the single-butterfly architecture (i.e. a sequential implementation) is significantly lower than a fully parallel implementation [81], albeit it will have a lower throughput.

In the sequential FFT architecture design, $N/2$ butterfly operations are included at every stage, and one butterfly unit can be used to perform all of them sequentially [82]. This lead to increase the computation time (latency) required for the FFT input vector. In this cause the overall throughput of the design is decreased.

2.5 FPGA Design Steps

The FPGA design steps are shown in Figure 2.12. At the Design Entry stage in an FPGA implementation, the designer creates design files using a schematic editor, or a Hardware Description Language (normally Verilog or VHDL). In this work, design of an FFT has been undertaken in VHDL using the Modelsim software development tool, while Xilinx block based System Generator has been used to develop the OFDM design [7, 9]. The System Generator allows designs to be made based on configuring of parameterisable blocks to implement DSP components such as filters, FFTs and so on, as well as simple elements such as multipliers and adders. After the design entry, the next step is synthesis. In this step, the VHDL and System Generator files are analysed and descriptions created at a lower level of logic abstraction, using a library of primitives. The XST tool from Xilinx has been used for this stage [83]. In the Partition stage, a particular physical element is assigned to each logic element. The Place level maps logic into specific locations in the target FPGA chip. At the Route level are the connections of the mapped logic. In the programming file generation step, a bit-stream file is generated to program the device. The subsequent device programming step downloads the bit-stream file to the FPGA, thus configuring it to implement the designed circuit.

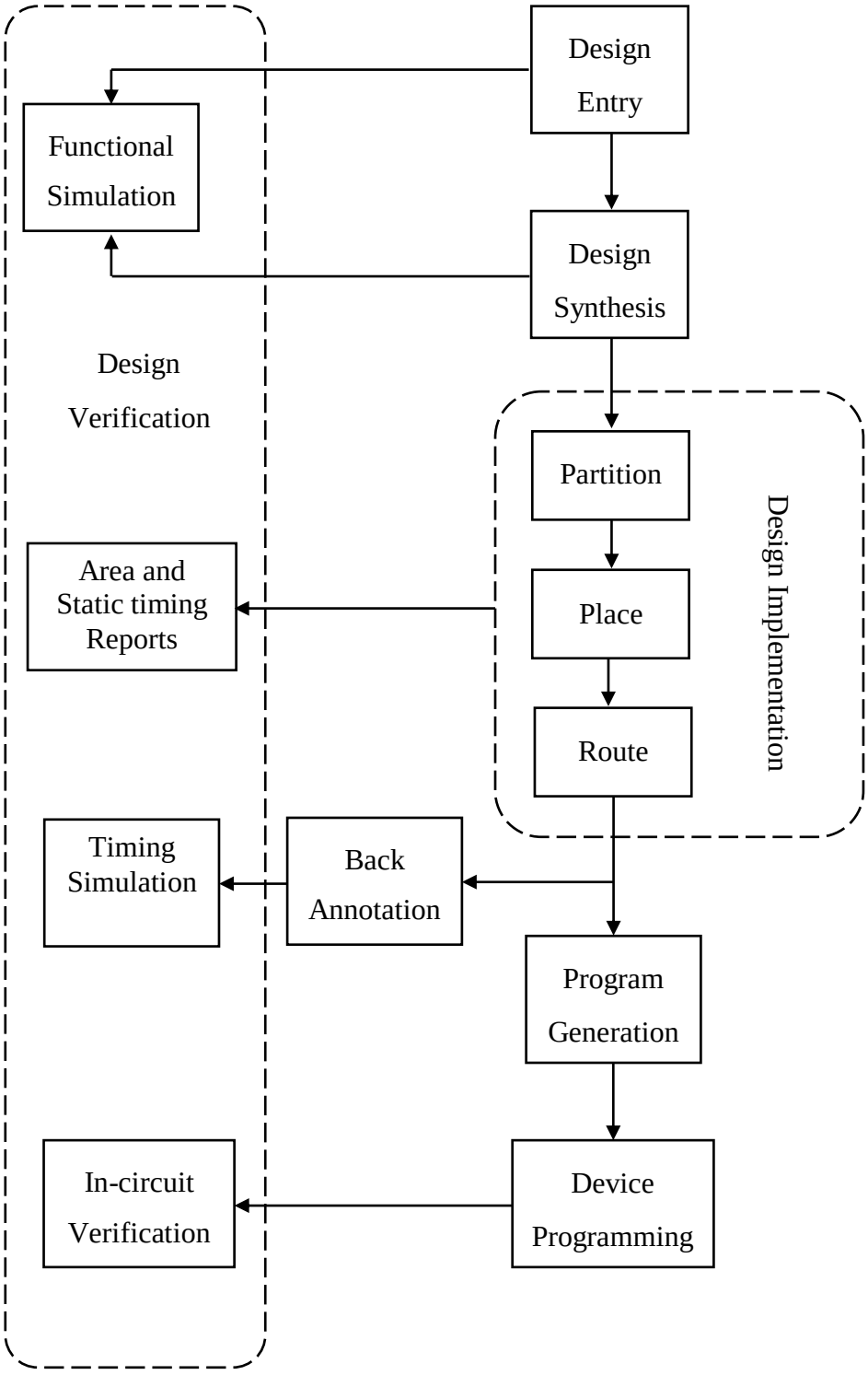

Figure 2.12: FPGA Design Step

Design verification can be undertaken via simulation, and simulation can be done at different levels. Various reports are generated to verify implementation results, such as maximum frequency and resource utilisation. The translate, map, and place and route processes are commonly referred to as design implementation [84-85]. The classic Xilinx design flow above was the process used in this thesis to generate the results for many of the different architectures.

2.5.1 Virtex 5 Technology

The Virtex 5 FPGA family from Xilinx [86] has many advanced features, and different variations of the device are offered. These features include multiplier-adder blocks (the so called "DSP48") which is a pipelinable unit capable of being configured for operation at the FPGA clock rate, performing a multiply-accumulate (MAC) on 25 bit and 18 bit input data (the multiplier is 25 x 18 bits). The *LX family* has been optimized for high-performance logic. The *LXT* has been optimized for high-performance logic with low-power serial connectivity. The *SXT* has been optimized for DSP and memory-intensive applications with low-power serial connectivity.

The current Xilinx Virtex 5 family can be clocked at 550 MHz [78] and consists of a classic FPGA fabric of gates, flip-flops, LUT (look up tables) and arithmetic blocks (DSP48s). The basic logic elements and Configurable Logic Block of Virtex 5 are illustrated in Figure 2.13 and Figure 2.14. The basic logic elements consist of a 6-input look-up table (LUT) that can be configured as a small RAM, called a distributed RAM, a configurable flip-flop/latch, and multiplexers to control the combinational logic output and the registered output (flip-flop/latch input). Fast carry logic is included to perform special logic and arithmetic functions using the slices. The slices consist of two basic elements grouped together, with each pair of two slices grouped to create Configurable Logic Block (CLB). The CLB is connected to programmable routing resources by a switch matrix. Identical CLBs are tiled in columns and rows in the device [87].

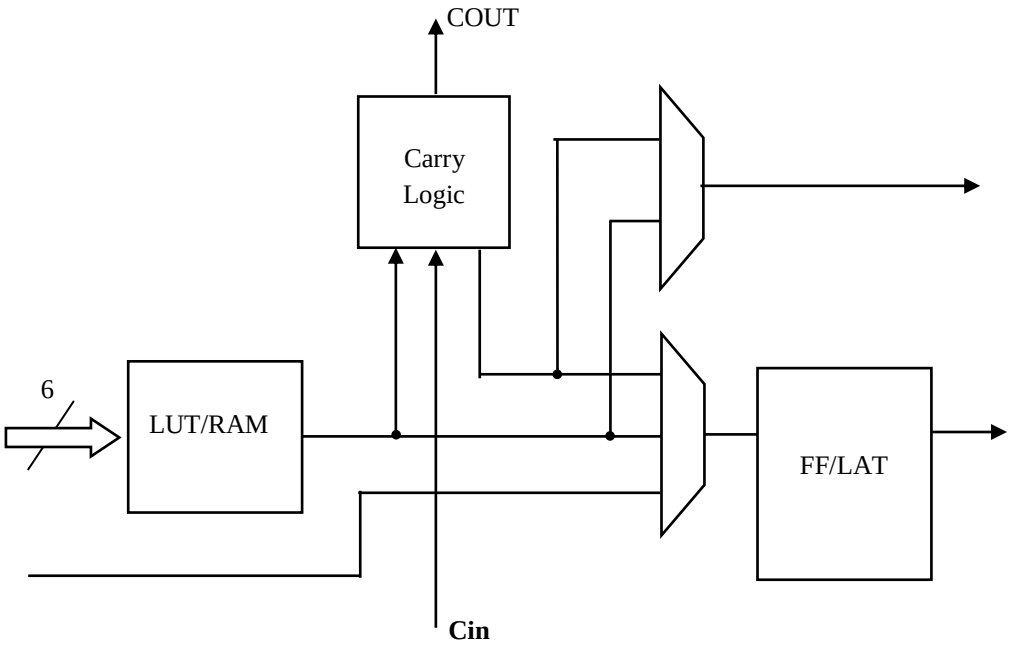

Figure 2.13: Basic Logic Element of Virtex 5 FPGA

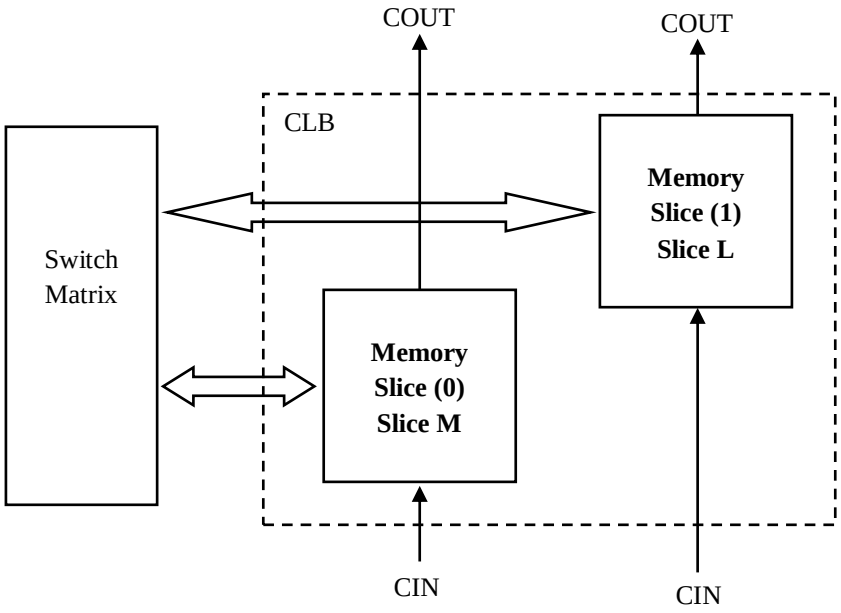

Figure 2.14: Virtex 5 Configurable Logic Block

The block RAM is a true dual-port RAM. In this case both ports can access any memory location at any time. The dual port memory stores up to 36 Kbits of data and

can be configured as either two independent 18 kb RAMs or one 36 kb RAM. Block RAMs are placed in columns, and the total amount of block RAM memory depends on the size of the device [88].

The Virtex 5 family introduced the DSP48E slice, a schematic of which is shown in Figure 2.15. The new DSP slice increased the multiplier input width to 25 x 18 bits, which compares to 18 x 18 bits in Virtex-4 devices. This offers more flexibility and easier implementation of DSP algorithms.

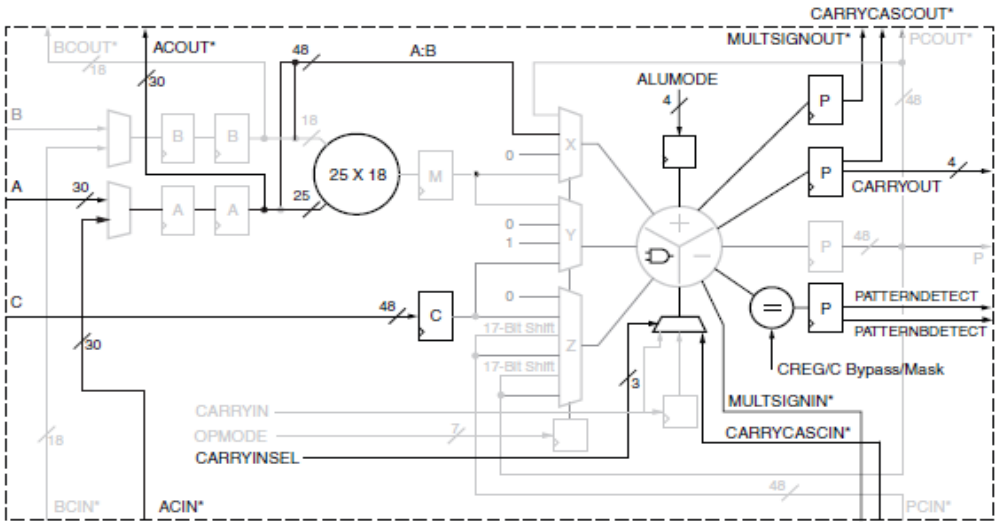

Figure 2.15:Xilnx Virtex 5 Family DSP48E Slice[1]

2.6 Summary

In this chapter we have reviewed the core computation required of the FFT and introduced the FPGA architecture and CORDIC. In order to now implement efficient and high speed FFTs we need critically evaluate the different algorithms, arithmetic architectures, and FPGA structures in order to aim to derive the most efficient implementation that will use resources optimally and where possible implement an FFT that is applicable for performing OFDM on the various radio standards mentioned in earlier chapters.

3 FPGA Implementations of High Speed FFTs

This chapter reviews and evaluates some relevant and recent work in the areas of FPGA implementation of FFT and IFFT algorithms, and OFDM for wireless networks on FPGAs. In Section 3.1, a collection of FFT implementations optimised for area is presented. While Section 3.2 presents FFT optimised for speed, Section 3.3 focuses on OFDM transceivers for different wireless standards. Section 3.4 summarises this chapter.

3.1 FFT Optimised For Area

In this section, FFTs that use minimum resource area are presented. This work is based on a sequential architecture FFT that relies on a single butterfly.

Xin Xiao [61] presents an FFT implementation based on CORDIC which reduces the memory required in an FFT architecture by eliminating the memory required to store the CORDIC angles . The design is applicable for FFT processors of any radix. The CORDIC algorithm is designed to eliminate the need to store the twiddle factor values or angles in memories, and instead generates the twiddle angles successively

using an accumulator. In this technique, the memory requirements of the whole FFT are reduced by more than 20%. The angle of CORDIC that is stored in ROM is generated by a simple circuit to save resource area, but the design still requires ROMs to store input and output data.

The Rotation mode CORDIC operation for an iteration *i* is summarised by Equations (3.1), (3.2), (3.3) and (3.4), where (x_i, y_i) is the initial location of the vector in Cartesian coordinates, and (x_{i+1}, y_{i+1}) is the new location after a rotation through the angle $arctan(2^{-i})$. For each iteration, the direction of rotation depends on the sign of d_i in Equation (3.3). The angle Z_{i+1} is the angle after the iteration, as stated in Equation (3.4).

$$x_{i+1} = (x_i - y_i \cdot d_i \cdot 2^{-i}) \tag{3.1}$$

$$y_{i+1} = (y_i + x_i \cdot d_i \cdot 2^{-i}) \tag{3.2}$$

$$d_i = \begin{pmatrix} -1 & \text{if} & z_i < 0 \\ +1 & \text{if} & z_i > 0 \end{pmatrix} \tag{3.3}$$

$$z_{i+1} = \left(z_i - d_i . \arctan 2^{-i}\right) \tag{3.4}$$

An interesting feature of this work is the method of generating the twiddle factor angles. The angle generator circuit is implemented by a circuit composed of an accumulator, a register and a latch, as shown in Figure 3.1. The $2\pi/N$ is the fundamental angle feed to the adder, where N is the FFT size. The sequential addition of the fundamental CORDIC angle is stored in flip flop registers and this generates all possible angles required by CORDIC to implement the multiplication by the twiddle factors. At each stage of the FFT computation, the generated angle streams need to change. The latch and its control signal are responsible for enabling and disabling the adder output based on the FFT stage.

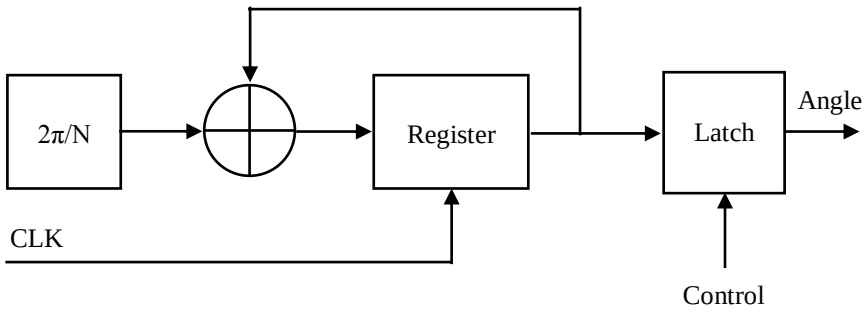

Figure 3.1 : Angle Generator for CORDIC

Bingrui Wang [45] focuses on a 64-point Radix-2 FFT processor, implemented on FPGA and targeted at WLAN (Wireless Local Area Network) applications. The design uses a DIF algorithm, and succeeds in reducing the number of multiplications to three instead of four. The complex multiplication by the twiddle factor requires four real multipliers. This number is reduced to three by using the simplification shown below. To illustrate the outputs of this paper, Equations (3.5) and (3.6) represent two complex numbers which, in the FFT algorithm, represent the data sample (*z1*) being multiplied by the twiddle factor (*z2*). By using factorization, the number of real multiplications is reduced from four to three as shown in Equations (3.7), (3.8) and (3.9). This simplification can be easily implemented in hardware.

$$z1 = x1 + jy1 \tag{3.5}$$

$$z2 = x2 + jy2 \tag{3.6}$$

$$z = z1 \times z2 = x + jy \tag{3.7}$$

$$x = x1(x2 + y2) - y2(x1 + y1) \tag{3.8}$$

$$y = x1(x2 + y2) - x2(x1 - y1) \tag{3.9}$$

Equations (3.8) and (3.9) are holding the real and imaginary parts of the multiplication. The first terms of both these equations are identical, and the result can be obtained by one multiplier, therefore reducing the number of to three. A pipelined FFT architecture is used to improve the throughput of the design. The architecture consists of four units: a control unit, a butterfly unit, two dual port RAMs, and an address generation unit. Verilog hardware description language is used within the Quartus II development environment to create the design, which is targeted at the EP2C70F896C6 part from Altera's Cyclone II family. The resource utilisation of the design is summarised in Table 3.1.

Table 3.1: Resource Utilisation of Pipelined 64-point FFT on Cyclone II Device

total logic	562
total pins	563
total embedded multipliers	48
Clock frequency	31.69 MHz

The novel Radix-2 FFT processor based on FPGA meets the requirements of the 802.11g WLAN standard. The design has a low clock frequency, and occupies a large amount of resources but it is using three multipliers instead of four for each complex multiplication.

The work of J. Viejo, reported in [89], is a methodological comparison of FFT/IFFT implementations on FPGA. Three methods are used: VHDL coding (VC), System-level tools at RT level (STR), and System-level tools at macro block level (STM). The first method is the VHDL coding method, and an FFT is designed with Radix-8 butterfly, RAMs, ROMs and control unit. In this method *only* VHDL code is used to create the design. The verification of the design is carried out using a combination of Simulink and Modelsim, using the Black Box facility of the Xilinx System Generator tool. Using the system levels tools at RT level, the System Generator blockset from Xilinx and VHDL code are jointly used to design the FFT/IFFT. In the last method, only System Generator's FFT block is used. In this methodology, it is only necessary to design an interface that adapts the input/output signals of the FFT block to the module interface. A comparison of accuracy with the number of clock cycles required is shown in Table 3.2. As shown in the table below the implementation of 64 point FFT with 26 bits data (13 bits for the real part and13 bits for the imaginary one) are tested to find which better accuracy with different style of implementation.

Table 3.2: Mean Error for VC, STR, STM Methods

	VC	STR	STM
Clock Cycles	291	291	262
Mean Error	1.0%	1.0%	2.1%

These results show that the VC and STR methods are more accurate than the STM method. However, the STM method requires the fewest clock cycles to complete the FFT calculation. This is because Xilinx implement the FFT architecture with minimum clock cycles. The designs were synthesised and implemented using Xilinx ISE, targeting a Virtex-II XC2V2000 FPGA, and the results are shown in Table 3.3.

Table 3.3: Resource Utilisation on Virtex-II XC2V2000 FPGA for VC, STR, STM Methods

Parameters	VC	STR	STM
Slices	1187 (23%)	1188 (23%)	1393 (27%)
Flip Flops	624 (6%)	624 (6%)	2041 (19%)
4 Input LUT	1984 (19%)	2030 (19%)	1380 (13%)
Bonded IOBs	58 (33%)	58 (33%)	57 (33%)
Block RAMs	2 (5%)	4 (10%)	3 (7%)
MULT18x18	12 (30%)	4 (10%)	7 (17%)
GCLKs	1 (6%)	1 (6%)	1 (6%)
Maximum operation frequency	39.53 MHz	40.15 MHz	122.65 MHz

The hardware implementation results show that VC and STR require about 4% fewer slices relative to STM. STM achieves the highest maximum operating frequency (122 MHz compared to 40 MHz for VC and STR). Based on this, VHDL code design can offer lower resource area while depending on Xilinx core can meet the high speed.

In the work of T.Y. Sung [90], an OFDM system is developed which supports FFT ranges from 2048 to 8192, in accordance with European digital video/audio broadcasting standards [77, 91]. In this work, 2048, 4096 and 8192 point FFT/IFFT processors are designed. The author presents an efficient, CORDIC-based Split-Radix FFT architecture suitable for the OFDM system under consideration. The processor is shown to perform an 8192-point FFT/IFFT every 138 ms, and a 2048-point FFT/IFFT every 34.5 ms, which exceeds the orthogonal frequency division multiplexing symbol rate.

A Split-Radix butterfly processor is used for the FFT calculation, and CORDIC is used to perform multiplication. This technique reduces the ROM size required for storing the twiddle factors. Figure 3.2 shows the proposed FFT architecture. To avoid conventional multiplier in the butterfly a rotational mode CORDIC is used and the required angles for CORDIC are generated to reduce the ROM required by the

design. In this paper the author also introduced a modified pipelined CORDIC arithmetic unit. The number of iterations or stages of the CORDIC processor is determined to be 12.

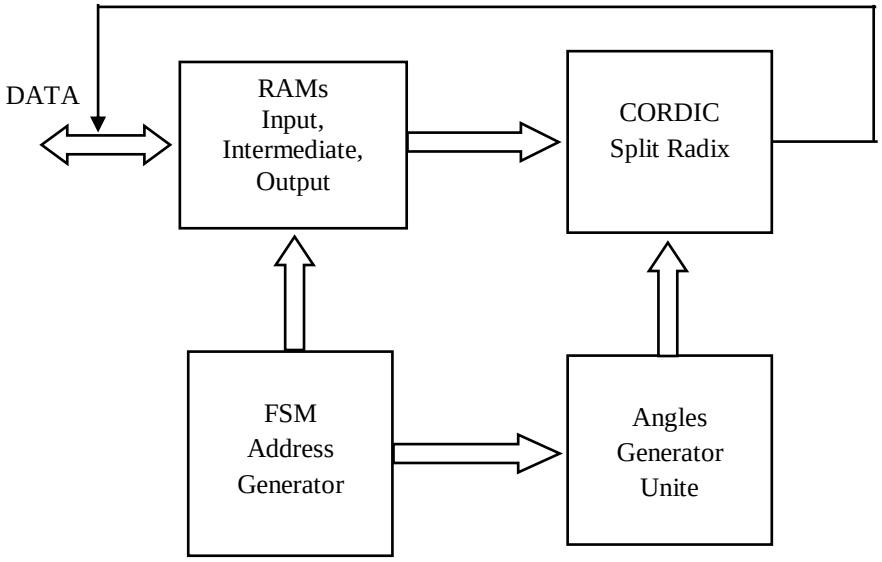

Figure 3.2: Sequential FFT Architecture Based on CORDIC with Split-Radix

In Fangming Liu [47], a 32-point, Radix-2 DIT FFT is introduced and a comparison made between three different butterfly implementations. The first is the traditional butterfly unit with four multipliers for the complex multiply; the second uses an alternative method of calculating the result it is similar to one used in [45], and has three multipliers; while the third uses only two hardware multipliers, taking two clock cycles to implement the four-multiplier operation. A recursive architecture is used, which contains the butterfly unit, data storage unit and address generator unit. The design is coded in VHDL and synthesized using Altera's Quartus II tools.

The 32-point FFT is designed with 8-bit precision using only 280 logic elements, and can be clocked at a frequency of 100 MHz. The implemented FFT architecture is shown in Figure 3.3.

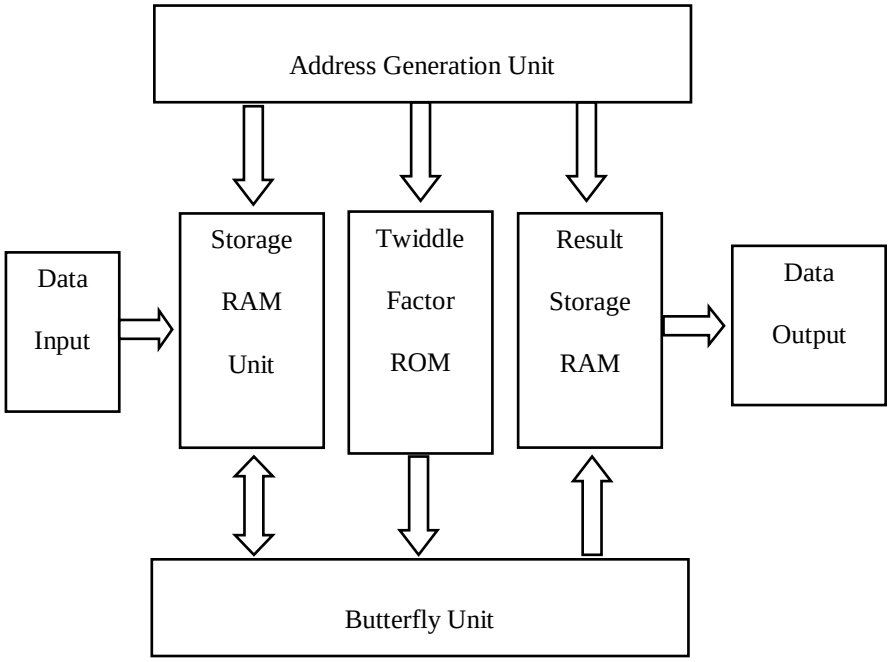

Figure 3.3 : Recursive FFT Architecture

3.2 FFT Optimised For Speed

In this section, FFT based on pipelined architectures are presented. The pipelined architecture can achieve the FFT calculation faster than the sequential architecture due to using a butterfly in each stage of calculation instead of using only one butterfly for all stages.

K. Harikrishna [32] introduces an FFT and IFFT based on the Radix-2^2 butterfly processor. The design uses an R2^2SDF Single path Delay Feedback pipelined architecture. The author makes a hardware utilisation comparison for different architectures, as shown in Table 3.4.

Table 3.4: FFT Architecture Requirements for multiplications, additions, memory and controls

Parameters	Multiplier	Adder	Memory	Control
R2MDC	$2(\log_4 N-1)$	$4\log_4 N$	3N/2 -2	Simple
R2SDF	$2(\log_4 N-1)$	$4\log_4 N$	N-1	Simple
R4SDF	$\log_4 N-1$	$8\log_4 N$	N-1	Medium
R4MDC	$3(\log_4 N-1)$	$8\log_4 N$	5N/2 -4	Simple
R4SDC	$\log_4 N-1$	$3\log_4 N$	2N-2	Complex
$R2^2SDF$	$\log_4 N-1$	$4\log_4 N$	N-1	Simple

Verilog code for an $R2^2$ SDF decimation in frequency 1024-point FFT/IFFT has been written. The design is targeted at a Spartan 3 FPGA. The resources occupied by the design are shown in Table 3.5. The maximum frequency for the design is 92.366 MHz. In this work, the resource area occupied by the design is high compared to Xilinx or the FFT introduced in this thesis.

Table 3.5: Resources Occupied by 1024 $R2^2$ SDF on Spartan 3

Logic Utilization	Used
No. of Slices	3155
No. of Slice Flip Flops	1514
No. of 4 input LUTs	5916
No. of bonded IOBs	32
No. of Mult18x18s	16
No. of GCLKs	1

The contribution of Zahra Haddad [36] is to introduce a design for a 1024-point Radix-2 DIF FFT. The FFT processor is shown to perform the FFT calculation in 5120 clock cycles. Of interest in the design is the fact that it uses one block dual port RAM, and completes the FFT calculation in 5120 clock cycles instead of 10240. The block RAM stores the calculated values in complex form, with 16-bits for real and 16-bits for imaginary (thus the RAM is 32-bit precision). The design is targeted at the Virtex-4 LX25 FPGA from Xilinx. The resources occupied by the design are shown in Table 3.6. Notably the design uses large numbers of block RAMs and

DSP48 multipliers compared to the designs introduced in this thesis and the Xilinx FFT core.

Table 3.6: Resource Utilisation of 1024-point Radix-2 FFT on Virtex-4 LX25 FPGA

Logic Utilization	Used
Number of Slice Flip Flops	38
Number of occupied Slices	2472
Number of 4 input LUTs	10841
Number used as 32x1 RAMs	2048
Number of DSP48s	10

Abdullah, S. S [92], A 1024-point Radix-2 DIF FFT based on the CORDIC processor is presented. It is pipelined, single path delay feedback architecture, featuring a simple controller and efficient pipelining. The design has no multipliers because it is based on CORDIC, and can reach a speed of up to 198 MHz.

In the work of Sheng Li, presented in [51], a 1024-point Radix-4 DIF FFT processor is implemented on a Xilinx Virtex II pro 70 FPGA. VHDL has been used in the development of the core. The design is divided into five pipelined stages, and each stage includes a butterfly and RAM memory. The design uses 30 block RAMs and 12 multipliers, and its maximum frequency is 164 MHz.

Also of interest is an FFT architecture for Radix-2 DIF which can produce two FFT transform samples every clock cycle by Christos Meletis in [82]. The architecture requires $N\log_2 N$ multipliers, $2N\log_2 N$ complex adders, and $N+2\log_2 (N-2)$ delay elements to compute an N-point FFT in $N/2$ clock cycles. A block diagram of the suggested pipelined architecture is shown in Figure 3.4. Each stage includes a butterfly and two shuffling units which are responsible for managing the delays required between stages. These delays work as pipelined registers as well.

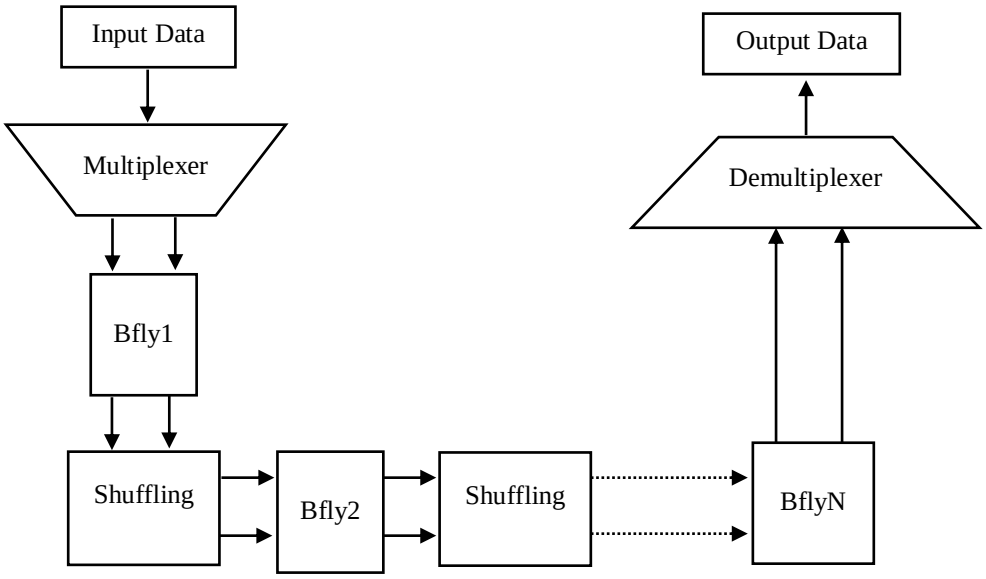

Figure 3.4 : Pipelined FFT Architecture for two outputs for each clock

3.3 OFDM Transceiver Design

In this section, OFDM transceiver design based on FPGA are presented. Many wireless standard use OFDM as modulation and multiplexing technique.

The work of Mª José Canet in [93] describes the design and implementation of an intermediate frequency (IF) OFDM transceiver based on FPGA. The Xilinx XC3S400-4 Spartan III board is used in the design, which supports Hiperlan 2 and IEEE802.11a/g WLAN standards. This design generates baseband OFDM symbols with a 20MHz sampling rate, and the baseband OFDM symbol is upconverted to a 120 MHz sampling rate. The upconverter comprises two interpolating filters of factors 2 and 3, and a quadrature mixer. At the receiver side, the downconverter consists of a mixer and decimator. A decimation by 3 from a sampling rate of 50MHz is used obtain a 20MHz baseband OFDM symbol. The design supports several modulation schemes, covering BPSK to 64-QAM. The baseband OFDM symbol is built using a 64-point IFFT, and 16 samples of cyclic prefix. The FFT/IFFT processors consist of 3 dual port memories and Radix-2 DIF butterfly processor. Two of the dual port memories are used to store the FFT/IFFT input, intermediate and output data, while the other memory stores the twiddle factor coefficients.

An autocorrelation circuit is used for frame detection: the received signal is correlated with a version of itself delayed by 16 samples. OFDM is sensitive to carrier frequency offset (CFO). The estimated frequency offset calculated by Equation (3.10) is used to remove the CFO:

$$f_o = \frac{\angle(R)}{2.\pi.N_c.T} \tag{3.10}$$

– where T is the sampling period and N_c is 16. The angle R is calculated by a circular vectoring mode CORDIC processor.

Channel estimation is performed in the frequency domain. The design is good for FPGA implementation, because the resource cost is low (as shown in Table 3.7), and it could be updated to other standards.

Table 3.7: Resource Utilisation of an OFDM transceiver for IEEE802.11a/g WLAN standards

Circuits	Slices	BRAMs	Multipliers
Up converter	273	0	0
Down converter	140	0	0
Mapping/Demapping	82	0	0
FFT/IFFT	340	3	3
Autocorrelator	431	0	9
CORDIC	363	0	2
Channel estimation	85	0	0
Control	95	1	0

In work by Garcia, J. Q. [30], the authors present an FPGA design of an OFDM modulator for IEEE 802.11a standards. Xilinx System Generator is used as a design tool. The mapping circuit offers BPSK, QPSK, 16-QAM and 64-QAM modulations, and is realised by a number of memories which store the constellation of each scheme. Two multiplexers are used for selection between different modulation

schemes one multiplexer for real and the other for imaginary The output is stored in two First In First Out (FIFO) buffers, and an Interleaver circuit is used to combine data, pilots and zero pads (a multiplexer and a counter are used to construct the interleaver). A Radix-4 IFFT and a couple of FIFOs are used to generate the OFDM symbol. The resources occupied by the design are summarised by Table 3.8. An advantage of the design is that it could be updated to other standards.

Table 3.8: Resource Utilisation of OFDM Modulator for IEEE 802.11a standards

Parameter	Used
Number of Slices	1678
Number of Slice Flip Flops	2353
Number of 4 input LUTs	2814
Number of bonded IOBs	29
Number of BRAMs	12
Number of GCLKs	1

The work reported by Garcia, J. Q. [94] is dedicated to the physical layer of the IEEE 802.16 OFDM modulator. It is smaller than the work in [30], and provides an update to meet the requirements of IEEE 802.16. The resources occupied by the design are summarised in Table 3.9. The design has used more resources than the previous design [30] due to the FFT size, which is 256-point for IEEE 802.16.

Table 3.9: Resource Utilisation of OFDM Modulator for IEEE 802.16 standards

Parameter	Used
Number of Slices	2614
Number of Slice Flip Flops	3566
Number of 4 input LUTs	4304
Number of bonded IOBs	29
Number of BRAMs	12
Number of GCLKs	1

In Xu, Jinsong [95], a Multi Band MB-OFDM is part of the physical layer of Wireless Personal Area Network (WPAN) IEEE 802.15.3a. Work is presented on the design and implementation of a MB-OFDM transmitter on FPGA for IEEE 802.15.3a. The VHDL language is been used to implement the design on a Xilinx Virtex 2 FPGA. The transmitter chain consists of scrambler, encoder, and puncture block; bit interleaving, QPSK and IFFT. A Radix-2 DIT butterfly processor is used in the FFT core. The multiplication is implemented using the CORDIC algorithm. A summary of the resource utilisation is given in Table 3.10, showing that the design is relatively expensive as it use large number of slices, flip flops, look up tables and multipliers.

Table 3.10: Resource Utilisation of an OFDM Modulator for IEEE 802.15.3a standards

Parameters	Used
Number of Slices	2885
Number of Flip Flops	3694
Number of 4 in LUTs	4811
Number of GCLKs	1
Number of DSP48s	12

The work of Aifeng Ren presented in [96] reports on the design of an OFDM transceiver based on FPGA. Intellectual property (IP) cores from Altera have been used to realise the design, which focuses on the baseband structure of OFDM. The paper describes different types of Forward Error Correction (FEC) IP cores from Altera, including the Reed-Solomon encoder/decoder,Convolutional Encoder/Viterbi Decoder, and Turbo encoder/decoder. The performances of the turbo encode/decode IP core is shown in Table 3.11.

Table 3.11: Resource Utilisation of Forward Error Correction using Altera IP

Device	Logic Elements (LES)	Memory (Bits)	Frequency (MHz)
EP1S10F780C6	7517	73216	95
EP1C20F400C7	7517	73216	83

For the interleaver / deinterleaver section of the design, the convolutional interleaver/Deinterleaver provides reduced delay and lower memory usage compared to the block interleaver/deinterleaver. A performance comparison of the constellation mapper/demapper IP-core is also made, and the results generated using Quartus II 8.0 software targeting the EP1C20F400C7 device is as shown in Table 3.12. A Radix-4 DIF design is used for the IFFT transmitter and FFT receiver. The performance characteristics of the FFT IP core targeted at two different FPGA devices are given in Table 3.13. The proposed architectures are suitable for rapid design of fourth generation wireless communication devices.

Table 3.12: Resource Utilisation of Constellation Mapper/De-mapper IP Core from Altera

Demodulation Scheme	Decoding Scheme	Eb/No (dB)	Logic Elements (LEs)	Frequency (MHz)
BPSK	Binary Decoding	15	347	274.53
8-PSK	User-Defined	15	452	256.87
16-QAM	Gray Decoding	15	376	245.22
256-QAM	User-Defined	15	413	228.50

Table 3.13: Radix-4 FFT IP Function from Altera

Device	Points	Logic Elements	Frequency (MHz)	Clock Cycle Count	Transform Time (us)
EP1S10F780C6	512	4510	255.62	512	1.03
EP1C20F400C7	512	4671	243.18	512	2.0

The work described by Manavi, F. [97] is the development of an OFDM modem for the IEEE 802.11a standard, based on FPGA. A synchronization circuit for packet detection and time synchronization is created. The design process involves a number of steps. Firstly, a floating-point model is designed using Cadence Signal Processing Worksystem (SPW). The floating-point model is then transformed to a fixed-point model in SPW. Finally, the fixed-point blocks are replaced with Hardware Design System (HDS) blocks, from which VHDL code can be generated automatically.

The resources required for the transmitter and receiver are shown in Table 3.14. For the synchronizer circuit, the occupied resources are shown in Table 3.15. Xilinx Virtex 2 is used to target the design.

This work represents rapid prototyping of the OFDM algorithm. The design is modelled in floating and fixed point, and meets the requirements of the IEEE 802.11a standard.

Table 3.14: Transmitter and Receiver Resource Utilisation for OFDM modem for IEEE 802.11a standard

	Slices	RAM Blocks	Total Gates
Transmitter	1115	10	690048
Receiver	1150	10	690533

Table 3.15: Synchronizer Resource Utilisation for OFDM modem for IEEE 802.11a standard

	Slices	Multiplier Blocks	Total Gates
Synchronizer	1409	18	97003

3.4 Summary

In this chapter we have reviewed a number of architectures for implementation of high speed FFTs. As discussed FFTs are important for the implementation of OFDM based architectures and the number of data points is always a power of 2, as specified in the appropriate standards documents for different radios (LTE, WiMax, Wi-Fi etc). In the above work, the aim of most authors and designers is to present FFTs that use minimum resources, and will achieve the necessary clock frequencies. This is of course a function of a number of things – first, the actual FPGA being used (technology, speed grade and so on), and the efficiency of the architecture. In this thesis we target the former and aim to produce architectures that will achieve speeds to implement various radio standards, but at a minimum cost. Hence in chapter 5, we will compare some "off-the-shelf" FFTs, with some custom designed architecture in order to demonstrate that careful optimisation will lead to faster and cheaper architectures.

4 Fast Fourier Transform Implementation on FPGA Based on Butterfly

4.1 Introduction

In this chapter, a full description, analysis, and hardware implementation of FFT/IFFT based on the butterfly operation is presented. The discussion focuses on the Radix-2 FFT algorithm. The FFT function and its inverse are implemented within the same architecture, in order to demonstrate that the design is indeed capable of implementing both functions as required by general OFDM transceivers.

Prior to implementing the FFT architecture, a MATLAB script was developed to provide golden reference floating-point and bit-accurate models for the forward and inverse FFT, and presents the benchmark for debugging and validating the design.

The custom VHDL code for the FFT and inverse FFT was created and tested in the ModelSim environment, and the code was simulated and compared with the golden reference MATLAB FFT implementation. Thereafter, the Mean Squared Error

(MSE) was found, together with the signal to noise ratio for the implementations of QAM transceivers. All designs were synthesized using the Xilinx ISE tool, and the resource utilisation and speed are reported as parameters of merit.

4.2 FFT Butterfly Processor Implementation

In this section, the butterfly-based, Decimation in Frequency Radix-2 FFT is introduced. It has a sequential architecture consisting of a single butterfly engine, two dual port RAMs, two ROMs and control unit.

Two types of butterflies are introduced here for analysis: the first is the serial butterfly FFT (to be discussed in Section 4.3.1), whereas the second is the serial pipelined butterfly FFT (covered in Section 4.3.2). Both butterflies use two multipliers rather than the usual four of standard Xilinx FFT cores, and one of them is highly pipelined to achieve higher speed. We will demonstrate that the sequential architecture (serial architecture that use only single butterfly) offers a reduction in resource utilisation compared to a pipelined architecture, and this work focuses on minimising the area occupied by the design. In the pipelined FFT implementation each stage requires a butterfly as shown in section 2.2.3.

4.2.1 FFT Entity

The FFT entity represents the input and output ports available for the user interface to the FFT, as defined by the VHDL design. A graphical representation of the FFT entity is provided in Figure 4.1 . It is a complex input FFT, where `fft_in_re` represents the real input and `fft_in_im` represents the imaginary input. Both inputs are specified as the "std_logic_vector" type with variable length, thus supporting inputs of any word length. The operation mode can selected as either the forward or inverse Fast Fourier Transform, using a pair of input signals. `FFT_IFFT` and `FFT_IFFT_we` are decided on, where `FFT_IFFT` and `FFT_IFFT_we` are

std_logic one bit assigned, with logic one for FFT, logic zero for IFFT. This will be active when `FFT_IFFT_we` is set to one as shown in Figure 4.1.

The operation of the FFT proceeds in three steps: reading the input vector, processing, and unloading the output. The `start` signal, which is of type std_logic, is activated (to logic 1) and this initiates the reading of input data. The length of the input vector is equal to the FFT size. After processing, the `unload` signal, which is also of type std_logic, is activated (to logic 1) to begin the writing of data to the output. All operations within the FFT core are synchronous to the clock, which is supplied via the std_logic signal `clk_main`.

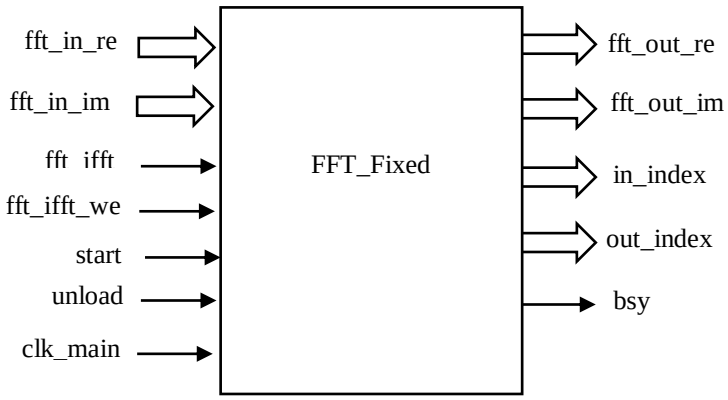

Figure 4.1: FFT Entity Block Diagram

The output ports `fft_out_re` and `fft_out_im` provide the results generated by the FFT core. These ports are both of type std_logic_vector, and in each case their length is defined as the input wordlength, plus $log_2(NFFT)$, where *NFFT* is the FFT size. This represents unscaled FFT fixed point precision, allowing wordlength growth of one bit at each stage. Equation (4.1) shows the maximum wordlength growth based on unscalled technique. There are different ways to have control on wordlength growth, the unscalled is the cheapest way. For example for a 1024-point FFT with input vector word length 16 bits, the *Maximum_Worldlength* would be 26 bits.

$$Maximum\_Wordlength = Input\_Vector\_Wordlength + \log_2(NFFT) \quad (4.1)$$

Three additional output ports are provided for monitoring the status of the core. The first two are `in_index` and `out_index`, which are of type std_logic_vector and dimension *log2(NFFT)*. These represent count values corresponding to the input and output data indices. The `bsy` port is a 1-bit flag (std_logic type), which is set high during the processing phase. `in_index` begins counting when `start` port is logic one, while `out_index` begins counting when `unload` is logic one. Both ports are used to monitor the input and output complex vector position to the FFT.

This chapter will introduce two FFT/ IFFT implementations: serial butterfly and serial pipelined butterfly.

4.2.2 FFT Based Butterfly Architecture

The architecture considered is a sequential, butterfly-based FFT/IFFT. This architecture is particularly suitable for low-cost design on an FPGA. It consists of two RAMs, two ROMs, a radix-2 butterfly, a Finite State Machine (FSM), and an address generator unit. A block diagram of the sequential butterfly architecture is shown in Figure 4.2 and this architecture has been implemented via "golden reference" MATLAB scripts and custom VHDL code. In the following sections, descriptions of each part in the FFT architecture are given.

Note that in the direct implementation of the FFT algorithm, the design requires a large number of butterflies, as given in Equation (4.2).

$$Butterfly\_numbers = \left(\frac{NFFT}{2}\right)\log_2(NFFT) \quad (4.2)$$

where *Butterfly_numbers* is the number of butterflies required to calculate the FFT. As stated in Equation (4.2), this depends on the FFT size (*NFFT*).

The number of butterflies required by the FFT is large as the FFT size increase. The advantage, of course, of the sequential architecture is that it uses only one butterfly and thus minimises resource utilisation (albeit is slower than a parallel implementation) but it also requires smart control and scheduling to be efficient. Hence a controller Finite State Machine (FSM) and an address generator unit are required for this purpose.

To review the general operation of the architecture in Figure 4.2, noting that the input vector first is loaded into RAM and the address generator then invokes a pair of values for user over two consecutive clocks, and the appropriate twiddle factor is read from the ROM. The numerical calculation is then performed using one butterfly processor, and the calculated values are written back to the same addresses in the RAM. This is done sequentially for all stages of the FFT design. One immediate disadvantage of this architecture is its large latency in evaluating the output, which implies a substantial time delay required for the sequential FFT architecture to conclude its calculation. Hence a "double clock" technique has been used to overcome the latency issue, which will be presented in Chapter 5.

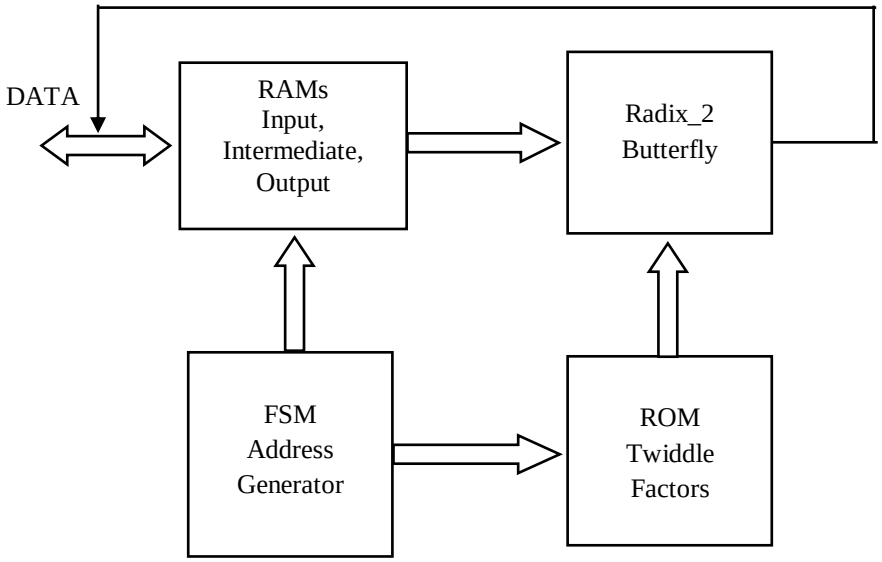

Figure 4.2: Sequential FFT Architecture Based on Butterfly

4.2.3 FFT RAMs

FPGAs provide two types of RAM: Block RAM and distributed RAM. The Block RAM is a dedicated memory block, while the distributed RAM is built from look-up tables distributed over the entire logic fabric. For large FFT designs, it is usually more efficient to use the Block RAMs (BRAMSs), however in many DSP systems there is not enough block RAM and hence distributed RAM may often be used. In Xilinx Virtex 5 (xc5vlx110t-3ff1136) devices used in this work, there are 148 BRAMS. A block diagram for the RAM that has been used in the design is shown in Figure 4.3. This RAM is used to store the input, intermediate and output data. One RAM is used for the real part, and the other for the imaginary part.

Referring to the interface as depicted in Figure 4.3 , `din_a` is the input port, which is used to write data to the RAM. It receives its first input from the interface and then from the output of the butterfly component until the calculation is finished. The input `addr_a` provides the address location at which data is stored in the RAM when the write enable port, `we`, is high. While the address specified by `addr_a` is used for write only, the `addr_b` port is used to specify the address for read only operations,

which result in the output data presented through the `dout_b` port. The values supplied to both the `addr_a` and `addr_b` ports are obtained from the address generation unit (to be covered in Section 4.6).

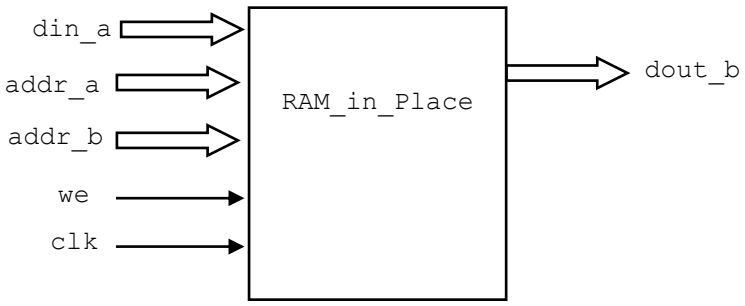

Figure 4.3: FFT in place RAM Block Diagram

In this design, an "in place" algorithm has been implemented. The term "in place" means that during the execution of the FFT, the same memory can be reused to store the results of intermediate calculations. This method is used to reduce the resource utilisation of the design.

The area occupied by the RAMs, i.e. the number of Block RAMs required by the FFT, is proportional to the precision of the input, and to the FFT size. The depth of the Block RAM is equivalent to the FFT size, while its precision is set to the maximum wordlength of the FFT output, as specified in Equation (4.3) below,

$$RAM\_precision = FFT\_input\_precision + \log_2(NFFT) \qquad (4.3)$$

where *RAM_precision* is equal to the maximum output wordlength required for the FFT.

As the input precision of the FFT is less than the output precision, sign extension is used to permit constant wordlength operations.

All of the above description applies to both the real and imaginary RAMs.

4.2.4 FFT ROMs

Two ROMs are assigned to store the twiddle factor values; one each for the real and imaginary parts. The complex twiddle factor is obtained from the two ROMs, and provided as an input to the butterfly processor, along with two complex data samples. Equation (4.4) is used to calculate the real part of the twiddle factor, *W_re*, while Equation (4.5) is used to calculate the imaginary part, *W_im*.

$$W\_re(n) = \cos\left(\frac{2\pi n}{NFFT}\right), n = 0,1,2,....., NFFT - 1 \qquad (4.4)$$

$$W\_im(n) = \sin\left(\frac{2\pi n}{NFFT}\right), n = 0,1,2,.....NFFT - 1 \qquad (4.5)$$

In these equations, *n* is the index which ranges from 0 to *NFFT-1*, where *NFFT* is the size of the FFT. A MATLAB script has been used to generate the values of *W_re* and *W_im* according to Equations (4.4) and (4.5), and these values have been stored in hexadecimal form in two ROMs. The precision is the number of bits used to represent the twiddle factor; for example using the 8 bit format <1, 7>, one integer bit and seven fractional bits are used to represent the twiddle factor, values in the range -1 to +0.99999 can be expressed. The precision of the twiddle factor has an impact on the FFT accuracy (signal to noise ratio) as discussed in Section 4.9.

Block RAMs are used to store the twiddle factor values; the number of Block RAMs required depends on the FFT size (*NFFT*) and the precision of *W_re* and *W_im*. A block diagram of the ROM entity is shown in Figure 4.4. The input port `addr` is of type std_logic_vector, and its word length is $log_2(NFFT)$. The address length depends on the maximum twiddle factor stored in the ROMs. The `en` port is of type std_logic, and represents an active-high enable (note that the ROM is enabled during the calculation of FFT stages, and is disabled during the input and output phases). The ROM is clocked by `clk_main`, or `clk_cal` when high throughput FFT is required, as described in Section 5.7. A more accurate FFT will require more precise

representation of the twiddle factor, potentially requiring more Block RAMs on the FPGA. Note that all of the above description applies to both the real and imaginary ROMs.

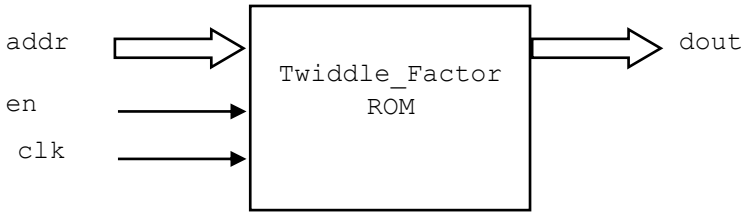

Figure 4.4 : ROM Twiddle Factor Entity

4.3 Butterfly Radix-2

In this section, a description of the traditional 4-multiplier implementation of a single butterfly is given. This is full parallel butterfly, as shown in Figure 4.5.

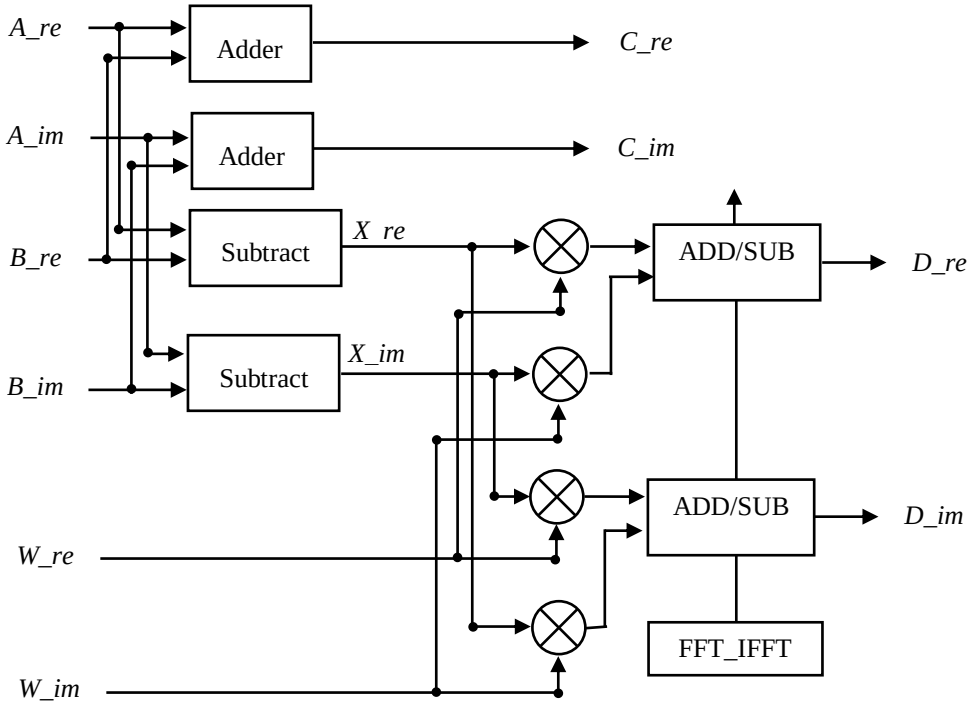

Figure 4.5: Radix-2 Butterfly Direct Implementation

This butterfly calculation is performed several times in the FFT, according to Equation (4.2), and in a fully serial implementation a single butterfly unit can be time-shared to perform all of the calculations. In the remainder of this section, the operation of an individual, directly implemented butterfly processor is described (later, Sections 4.3.1 and 4.3.2 will detail an alternative design of the butterfly, which uses fewer resources).

Each butterfly operation requires two complex data values and one complex twiddle factor. The inputs to the butterfly are denoted by *A* and *B*, and the twiddle factor by *W*. Their real and imaginary parts are indicated by the suffixes *_re* and *_im*, respectively, hence *A*, *B* and *W* are given by Equations (4.6), (4.7) and (4.8).

$$A = A\_re + jA\_im \tag{4.6}$$

$$B = B\_re + jB\_im \tag{4.7}$$

$$W = W\_re + jW\_im \tag{4.8}$$

The complex quantities *C* and *D* are calculated by the butterfly as given in Equations (4.9), (4.10), (4.11) and (4.12).

$$C\_re = A\_re + B\_re \tag{4.9}$$

$$C\_im = A\_im + B\_im \tag{4.10}$$

$$X\_re = A\_re - B\_re \tag{4.11}$$

$$X\_im = A\_im - B\_im \tag{4.12}$$

Therefore, implementing the calculation of *C_re* and *C_im* requires one adder each, while *X_re* and *X_im* require one subtract each.

To complete the calculations, Equations (4.13) and (4.14) are used in the case of the forward FFT, while (4.15) and (4.16) are used for the inverse FFT. These equations

correspond to multiplication with the twiddle factor, and show that a total of four multiplication operations are required for both the forward and inverse FFTs.

$$D\_re = X\_re \times W\_re + X\_im \times W\_im \quad (4.13)$$

$$D\_im = X\_im \times W\_re - X\_re \times W\_im \quad (4.14)$$

$$D\_re = X\_re \times W\_re - X\_im \times W\_im \quad (4.15)$$

$$D\_im = X\_im \times W\_re + X\_re \times W\_im \quad (4.16)$$

A key development in this thesis is to investigate butterflies with only two real multipliers, and which use CORDIC (Chapter 5) to perform the multiplication so that the design can be implemented without explicitly using multipliers, and thus achieving a variety of designs that can run on different FPGA structures and can be chosen and balanced according to available resources and speed requirements.

By just negating the sign of the twiddle factor it is well known that we can use the FFT architecture to calculate the inverse FFT, thus using a single architecture with a selectable control line to negate or otherwise the twiddle factor. Clearly this is another method of minimising the overall resource cost.

As shown in Figure 4.5, the direct implementation of the butterfly required six adder/subtractors and four multipliers all of appropriate word lengths corresponding to a pre-specified word length to achieve a certain level of desired accuracy. In the next sections, two new butterfly implementations are introduced to obtain minimum resource for the FFT.

4.3.1 Radix-2 Butterfly Serial Implementation

One of the options considered is a modified butterfly architecture which reduces resource consumption. The new architecture has reduced the number of multipliers and Block RAMs to two each – this will clearly have some impact on the maximum achievable speed, however the decision of the designer is to use the FFT that can achieve the appropriate sample rate of the wireless standard, and at the minimum cost (so to confirm - using the off-the-shelf low cost or "efficient" FFT cores is not always the right choice). The direct implementation of the butterfly, as presented in Section 4.3, required for example four RAMs to read and write the data, and four multipliers: this constitutes a fully parallel implementation of the butterfly processor. The alternative here is a "serial" butterfly, which could be utilised to reduce the resource cost. The block diagram of the serial Radix-2 butterfly is shown in Figure 4.6.

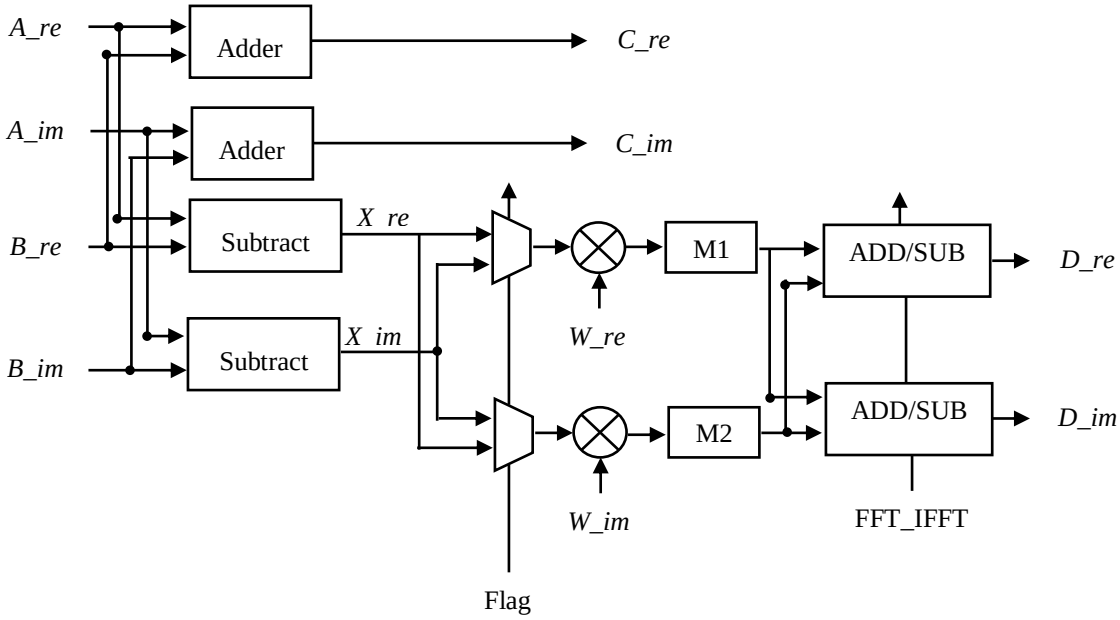

Figure 4.6: Serial Butterfly Radix-2

In this technique, *A_re* and *B_re* are stored in the same RAM (the "real RAM"), and likewise, *A_im* and *B_im* are stored in a single RAM (the "imaginary RAM"); in this case only two RAMs are used in total. As both *A_re* and *B_re* need to be available for calculation at the same time, two clock cycles are required to read, and to write, the calculated butterfly values. A control signal of length two clock cycles is used for co-ordination purposes. Using this style of memory access, it is not necessary to use four multipliers, as they would not be fully utilised. Therefore, an architecture which uses two RAMs and two multipliers is possible.

The calculations of *C_re, C_im, X_re, X_im* are implemented using the same resources as required for the direct implementation, taking two clock cycles. Multiplexers are used to select inputs for the two multipliers, which in each case are one of the outputs from the subtractors, *X_re* and *X_im*. A control signal, "Flag", is used to make the choice and this takes two clock cycles. In the first multiplexer, while Flag is equal to one, the input *X_re* is selected, and while Flag is zero, the input *X_im* is selected and vice versa in the second multiplexer. The output of multiplexers, which are *X_re, X_im* and *X_im, X_re,* are multiplied by the twiddle factor *W_re* and *W_im*. M1 holds *W_re* multiplied by *X_re* and *X_im* while M2 holds *W_im* multiplied by *X_re* and *X_im* successively as shown in Equations (4.17) and (4.18).

$$\begin{aligned} M1 &= W\_re \times X\_re \quad \text{when Flag} = 1 \\ M1 &= W\_re \times X\_im \quad \text{when Flag} = 0 \end{aligned} \tag{4.17}$$

$$\begin{aligned} M2 &= W\_im \times X\_im \quad \text{when Flag} = 1 \\ M2 &= W\_im \times X\_re \quad \text{when Flag} = 0 \end{aligned} \tag{4.18}$$

In this case, full utilization of the multipliers has been achieved, and the set of outputs produced corresponds to the results of Equations (4.13), (4.14), (4.15) and (4.16). The "Add/Sub" blocks perform addition or subtraction based on the choice of

transform, where they accept the outputs of *M1* and *M2*, and perform addition or subtraction as required to compute the FFT or IFFT (the FFT is generated by adding *M1* and *M2*, while the IFFT is generated by subtracting *M1* and *M2*).

The important point to mention is the word length growth during the multiplication, as it can go up with each calculation. To limit this growth, truncation techniques are used. For sixteen bits input precision the RAMs are set to be 27 bits in the case of *NFFT*, equal to 2048 based on Equation (4.3). In the twiddle factor case with eight bit precision, the output of multipliers grows to 35 bits so it is truncated to 27 bits again. The next section introduces another variation of the serial butterfly, which has the advantage of using more pipeline registers, so as to increase the frequency at which the design can operate successfully on the FPGA.

4.3.2 Radix-2 Butterfly Serial Pipelined Implementation

In this section, a pipelined butterfly architecture is presented which can support a higher FPGA clock frequency than the serial butterfly of Section 4.3.1.

The pair of RAMs used in the architecture store the real part of the input vector in the form:

[*A_re(0), B_re(0), A_re(1), B_re(1), ...A_re(NFFT-1), B-re(NFFT-1)*]

and the imaginary part as:

[*A_im(0), B_im(0), A_im(1), B_im(1), ... A_im(NFFT-1), B_im(NFFT-1)*].

This means that *A_re* and *B_re* appear one after the other delayed by one clock. The values of *A* and *B* appear sequentially and need to be processed in the same time to calculate *C_re* and *C_im*. To do this, a Flag signal, which takes two clock cycles, is

used to extend the times of *A_re, B_re, A_im, B_im*. *A* vector is [*(A_re(0)A_im(0), A_re(1)A_im(1), ...A_re(NFFT-1)A_im(NFFT-1)*] and *B* vector is [*(B_re(0)B_im(0), B_re(1)B_im(1), ...B_re(NFFT-1)B_im(NFFT-1)*]. To find out

the *X_re* and *X_im* in Equations (4.11) and (4.12), only one subtract is required in this case as the input is complex. Two ROMs are used to store the twiddle factor values, one for real twiddle factors *W_re* and the second for *W_im*. Two multiplexers are used to generate complex twiddle factors, two multiplied by the complex *X*, to have all the combinations required to calculate *C_re* and *D_re* as shown in Figure 4.7, Figure 4.8 and Figure 4.9.

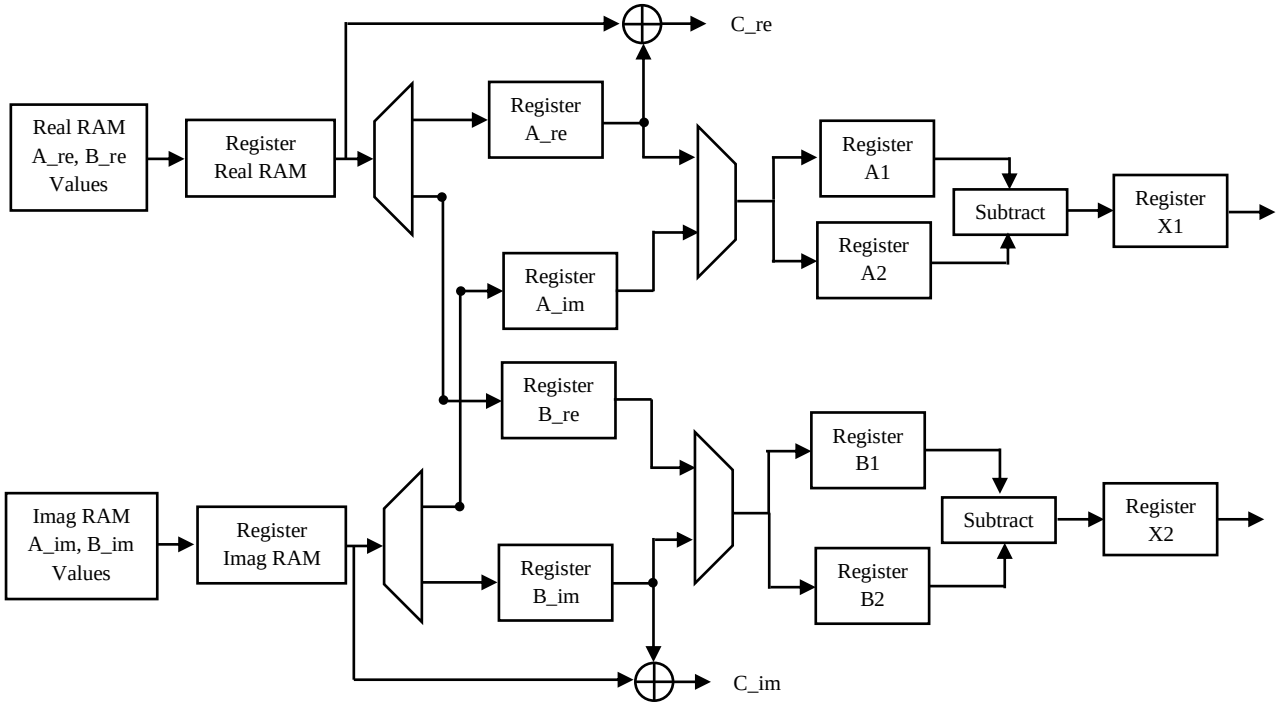

Figure 4.7: Pipelined Butterfly Part A

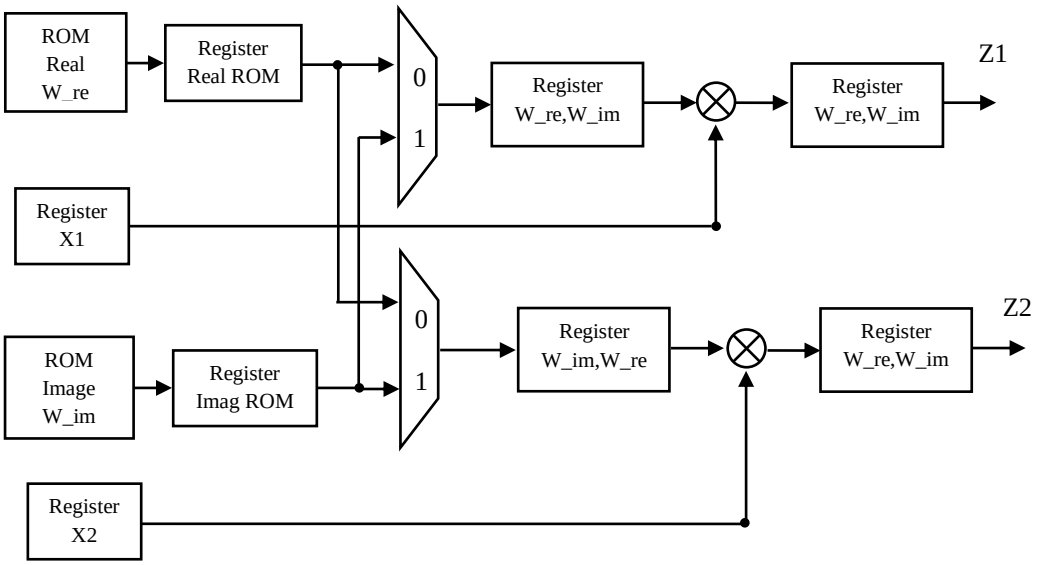

Figure 4.8: Pipelined Butterfly Part B

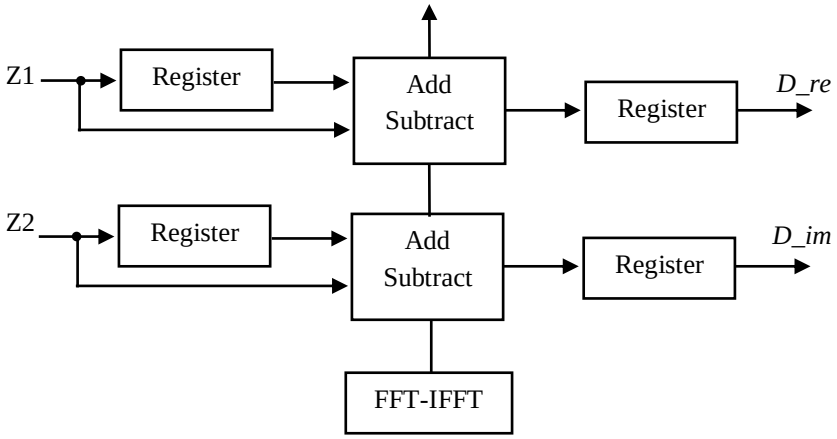

Figure 4.9: Pipelined Butterfly Part C

4.4 FFT Logic Control Unit

The logic control unit is responsible for co-ordinating the operation of the FFT processor (with variations depending on whether a forward or inverse transform is required), and the scheduling of reading inputs and writing outputs. As reviewed in Section 4.2.1 a 1-bit input is used to choose between the FFT or IFFT modes of operation. The add/sub blocks in the previous butterfly sections are set to add or subtract in response to this input (and this is achieved using `if statements` in the VHDL code). Referring to the equations presented previously, the effect is to switch between Equations (4.13**)** and (4.14**)** which implement the FFT, and (4.15**)** and (4.16**)** which implement the IFFT. The FFT/IFFT functionality is implemented either using the serial butterfly, or the serial pipelined architecture as presented in Sections 4.3.1 and 4.3.2, respectively. Note that this is an efficient way to implement the two functions (FFT and IFFT) within a single architecture; another possibility would be to store (Wr + jWi) and (Wr-jWi) twiddle factors but this would double the cost of the ROM.

In the logic control unit, initial operation is prompted by the 1-bit input `start` going high, at which point a counter begins to increment from 0 up to (*NFFT-1*), and this corresponds to the index of the input sequence `in_index`. The signal `in_index` is used to address the RAMs which store the complex input vector as well as the twiddle factors, while the FFT is processing the input data.

The logic control unit sets the `bsy` port to logic 1 while processing, and to logic 0 when the calculation is finished. During the calculation phase `bsy` stack is logic one. The number of clock cycles that `bsy` is logic one represents the latency of the FFT required to produce the output. Latency is an important performance metric and is evaluated in the FFT testing and a significant latency can negatively affect the ability of the FFT to process a high OFDM sampling rate.

The 1-bit input, `unload`, is used to prompt writing of the calculated outputs of the transform to the output ports. When logic 1 is received on `unload`, the logic control unit checks whether the calculation is finished. If the calculation has completed, the output of the FFT is written to the output port of the FFT unit by generating a counter, `out_index`, that counts from 0 up to (*NFFT*-1), and addresses RAMs holding the computed output. The logic control unit works synchronously with the finite state machine, as described in Section 4.5.

4.5 Finite State Machine

The Finite State Machine (FSM) built in the core is responsible for management, control and operation synchronous with the logic control unit.

The FSM has five states: S0, S1, S2, S3 and S4. The state machine starts in the S0 state, which allows the FFT core to read the input data from the interface. During this time, no processing of data takes place. In the S0 state, the status of the `in_index` signal generated by the logic control unit is checked, and when `in_index` = *NFFT* – 1, the FSM transitions to state S1.

The S1 state is dependent on other processes in the FFT to generate internal counters and look-up tables for the number of stages required to calculate the FFT and the length that each stage requires. Each stage requires *NFFT* clock cycles to achieve the butterfly operation between the two complex inputs to the butterfly. The output is back to the same address of RAMs that it was collected from. Each stage takes NFFT instead of *NFFT/2* cycles because this is serial implementation of the FFT, offering 50% of the resources area on the FPGA.

It requires considerable effort to manage and control the operation inside the FFT. To keep monitoring the calculation process in the butterfly in each stage, a Flag signal is generated, taking two clocks. This is used to achieve the multiplexing necessary for the butterfly registers to implement serially, producing the *NFFT* cycles for each stage instead of *NFFT/2* in the normal case. The number of stages is equal to

$log_2(NFFT)$. For reasons related to delays and pipelined stages, the FFT needs a few extra clocks, not to exceed 10, to complete the calculation of the stages. To monitor this state S2 is assigned and a register with size equal to 11 bits is used to fix that. When this register bit six goes high the calculation is finished. S3 state is one clock before the output finishes. S4 state is when the output is ready. At this point the FFT returns to state S0 to dump the output to the interface. The variable out_index is used to count and control this process, no calculation operation is performed in state S0. The flow state diagram of the FSM is shown in Figure 4.10.

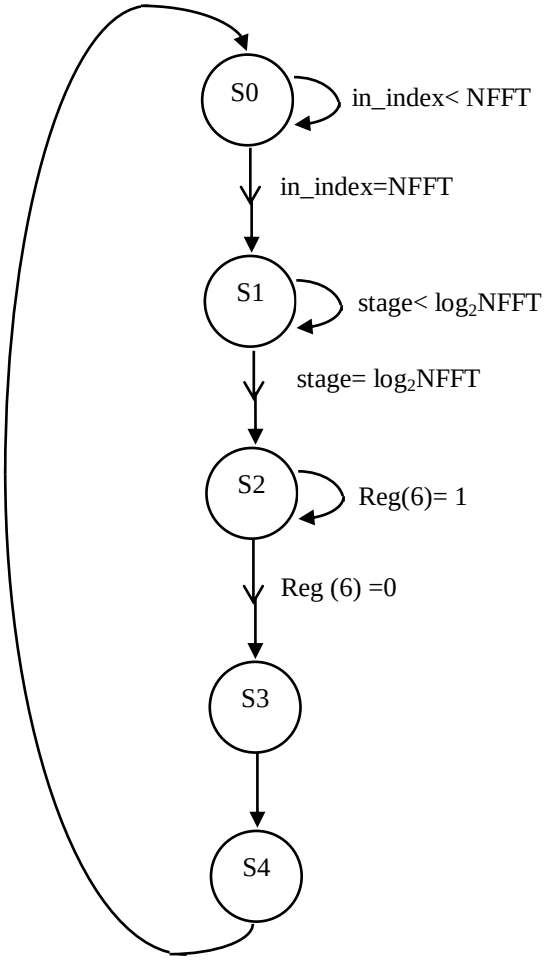

Figure 4.10: Finite State Machine FFT

4.6 Address Generation Unit

The address generation unit is responsible for generating the addresses for both pairs of RAMs and ROMs. The RAMs are used to store the input, output and intermediates values during the FFT stages calculations, while the ROMs are used to store the complex twiddle factor. The orders of reading values from RAMs and ROMs differ based on the stage of the calculation, and hence the implementation of the address generation unit is non-trivial.

As mentioned in Section 4.4, there are three basic phases of operation, namely: reading the data from the interface, processing the data, and dumping the output to the interface. When reading and writing data, the RAM address is generated by the `in_index` and `out_index` counters, after the data has entered the RAMs as shown in Figure 4.11**,** where data *x(n)* is represented as a one-dimensional array. *NFFT* can be factored as a product of two integers as shown in Equation (4.19**)**.

$$NFFT = N1 \times N2 \tag{4.19}$$

NFFT may be expressed as the product of *N*1 and *N*2, where *N*1 and *N*2 are both positive integers.

Equation (4.20) is used to generate the index mapping for the RAMs.

$$Add\_A = k1 + N1k2 \begin{pmatrix} 0 \le k1 \le N1-1 \\ 0 \le k2 \le N2-1 \end{pmatrix} \tag{4.20}$$

$$Add\_B = Add\_A + step \tag{4.21}$$

n	0	1		N-1
x(n)	x(0)	x(1)	.....	x(N-1)

Figure 4.11: Input data index

$$step = \frac{NFFT}{2^{stage+1}} \tag{4.22}$$

Equation (4.23) is used to generate the address for accessing the ROMs storing the twiddle factor.

$$Add\_w = \left(k1 \bmod \frac{NFFT}{2^{stage}} \right) 2^{stage} \tag{4.23}$$

A flow chart describing the operation of the address generation unit is provided in Figure 4.12. The algorithm is efficient in terms of implementation, and can be applied to any FFT/IFFT size; this is defined in the VHDL code.

V is the maximum number of stages for the FFT, i.e. 11 stages for 2048, 10 stages for 1024, 9 stages for 512, 8 stages for 256 and 7 stages for 128. A counter is used to increment through the FFT processing stages, and this increment is performed every *NFFT* clock cycles, as was explained in Section 4.5. The stage counter, *stage*, is compared with the maximum number of stages, *V*, and if they are equal then this indicates that the FFT processor has completed the calculation.

*N*1 represents the distance between the *A_re* and *B_re*, or *A_im* and *B_im*, RAM memory locations. Its value depends on the FFT size (*NFFT*) and the current stage of the computation, i.e. the value of stage.

Recalling Figure 4.11, where *x*(0) represents *A_re*(0)+*jA_im*(0). This value is processed in the butterfly along with *x*(4), which represents *B_re*(4)+*jB_im*(4). The next value *x*(1) is processed with *x*(5) and so on. At stage zero of the calculation, the difference between values is four in the case of eight points FFT; which is equivalent *NFFT/2*. At the next stage, the difference between the processed samples changes to two, and in the final stage, to one. Within the VHDL implementation of the address generation unit, *N*1 is a look-up table which carries all possible step differences between the samples that are processed, based on the FFT size and stage.

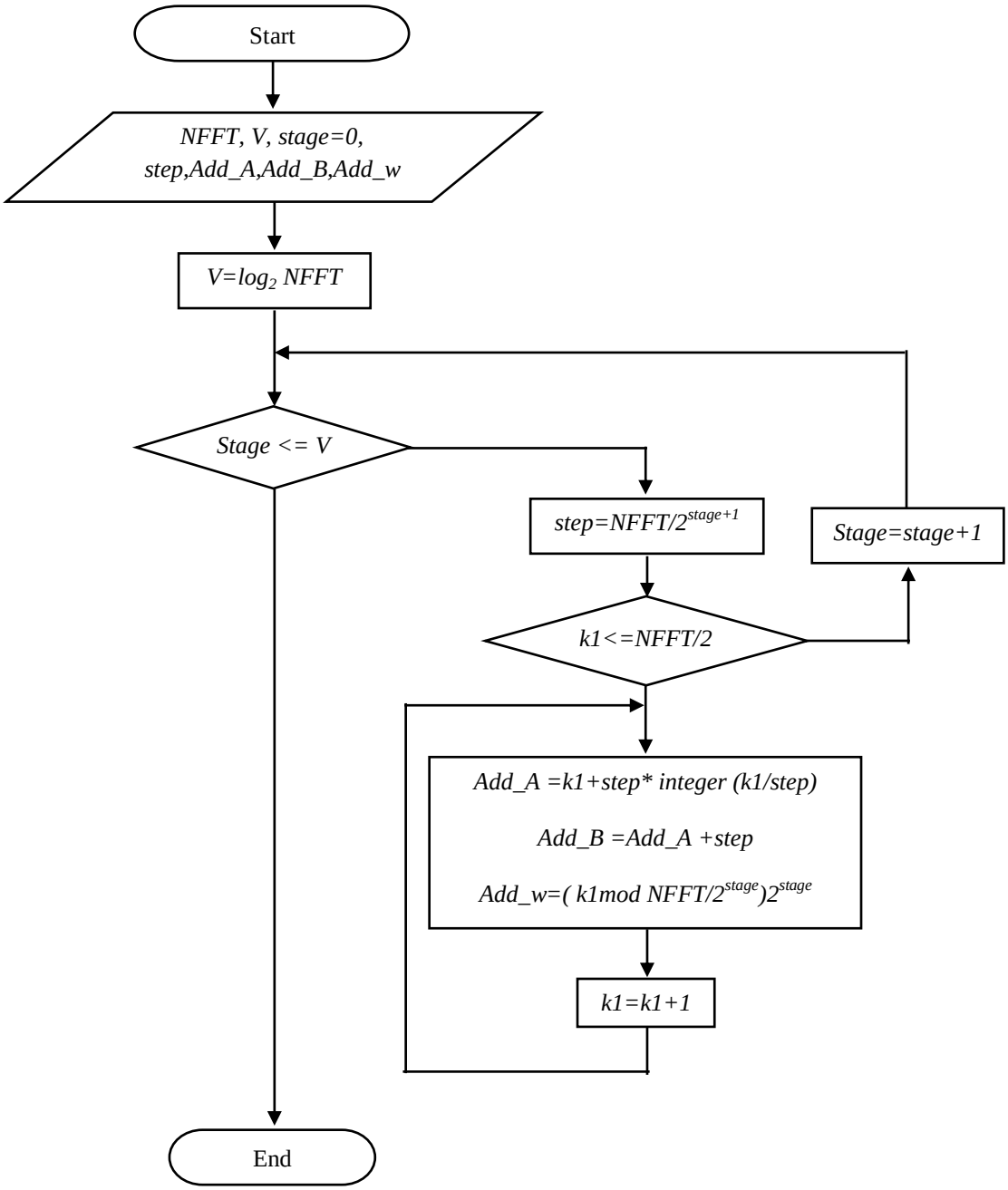

Figure 4.12: Address Generation Flow Chart

4.7 FFT Based Butterfly MATLAB Scripts

For large FFT sizes, it is necessary to use a floating-point model for debugging and validation purposes, and MATLAB scripts have been written for the architecture shown in Figure 4.2.

The bit-accurate model has been derived from the floating-point model. The bit-accurate model is an integer valued representation of the design. This can be compared to the signed integer representation which may be viewed in the HDL waveform, for example in Modelsim or Isim.

The scripts consist of a main file and functions. A flow chart of the design is shown in Figure 4.14.

The complex input vector function is responsible for generating a complex random input vector to the FFT. Its length is equal to the FFT size. The generated complex numbers are floating point. To obtain an integer representation, the values are multiplied by 2^{no_bit-1}, where *no_bit* is the number of bits used to represent the input to the FFT. For sixteen bits the precision is 32768, and so on.

The Twiddle Factor Generator function is responsible for generating the real and imaginary parts of the twiddle factors. The values have been generated based on Equations (4.4) and (4.5). The floating point values are converted to Hexadecimal and stored in the ROMs within the VHDL design. The Address Generator function is responsible for generating the addresses for the buffers that store the complex input vector and the twiddle factors. The butterfly function is responsible for implementing the Radix-2 butterfly.

The output of the FFT is in bit-reversed order. A MATLAB function is used to generate it in the natural order. If the IFFT is calculated, then the output is divided by the FFT size.

The floating point model is tested to prove its accuracy as shown in Figure 4.13. MATLAB's built in FFT function is considered as a golden reference model, and the signal to noise ratio demonstrates that the floating point model is working properly. The average signal power, average error power (noise power) and the signal to noise ratio are calculated according to Equations (4.24), (4.25) and (4.26). The signal *y_Matlab(n)* of Matlab's built-in function, while *y_float(n)* is the floating point model output.

$$average\_signal\_power = \frac{\sum_{n=0}^{NFFT-1} (y\_Matlab(n))^2}{NFFT} \qquad (4.24)$$

$$average\_error\_power = \frac{\sum_{n=0}^{NFFT-1} (y\_Matlab(n) - y\_float(n))^2}{NFFT} \qquad (4.25)$$

$$signal\_to\_noise = 10\log_{10}\left(\frac{average\_signal\_power}{average\_error\_power}\right) \qquad (4.26)$$

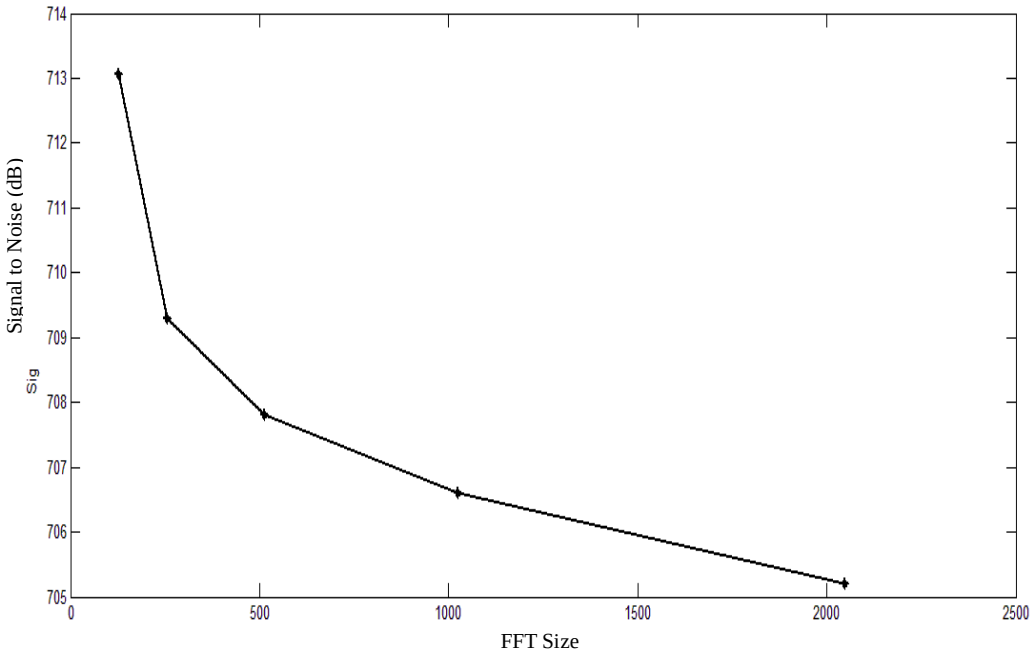

Figure 4.13 : Floating-Point accuracy of FFT based on Butterfly

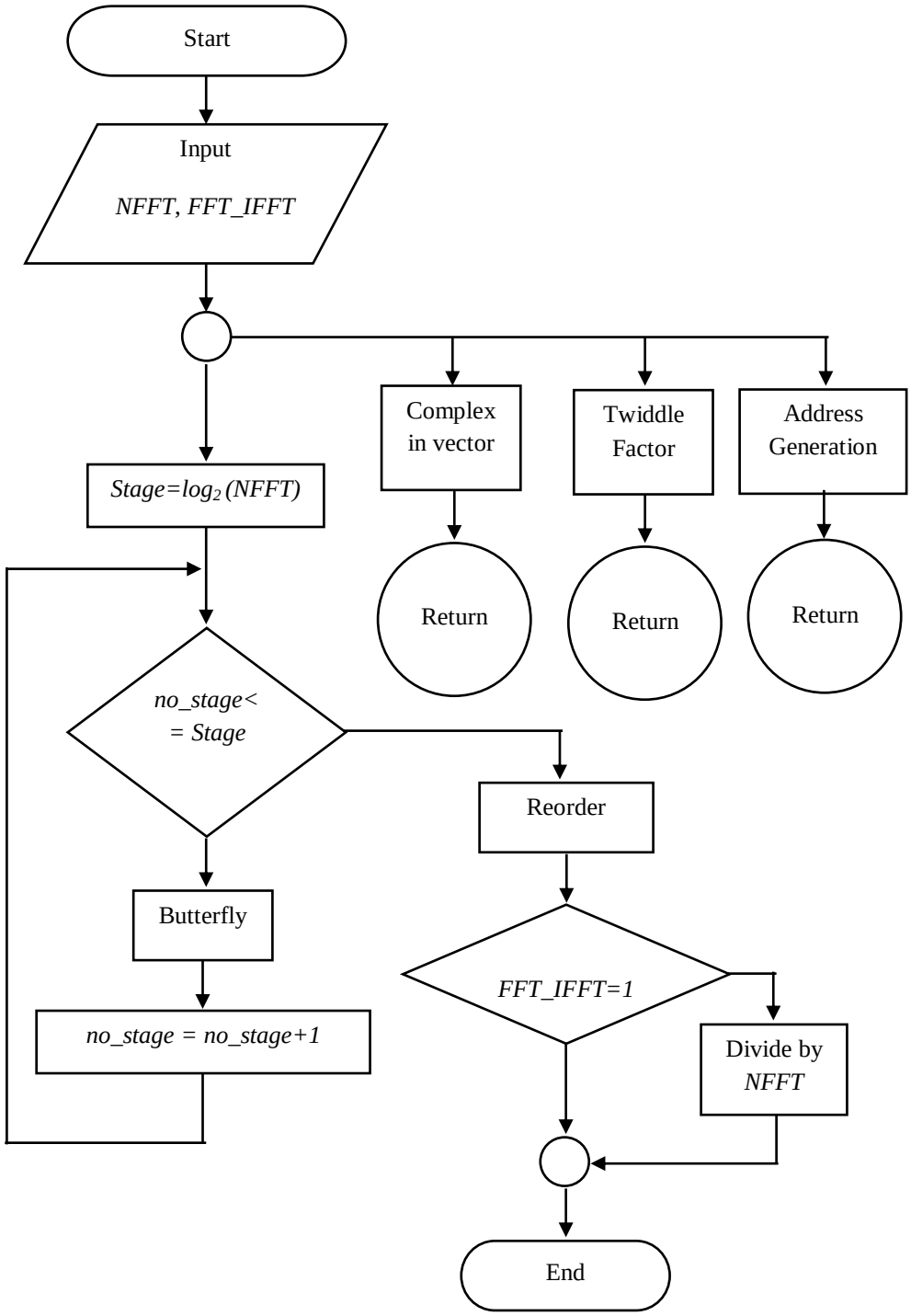

Figure 4.14: Matlab Flow Chart for FFT Based on Butterfly

4.8 FFT Based Butterfly Test

VHDL code was written for the FFT architecture shown in Figure 4.2 and the design was synthesised using the Xilinx ISE tool. The resources occupied by the design are summarised in Table 4.1 for the serial butterfly implementation, and Table 4.2 for the serial pipelined butterfly. The designs have been tested with 128, 256, 512, 1024 and 2048 FFT sizes. The input complex vector precision is <1, 15>, i.e. one integer and fifteen fractional bits. The twiddle factor precision is <1, 7>, i.e. one integer and seven fractional bits. The output of the complex FFT is permitted to grow to <12, 15> bits.

Table 4.1: Resource Area of Serial Butterfly FFT Architecture on Virtex 5 X110t

Parameters	FFT SIZE				
	128	256	512	1024	2048
Flip Flops	450	473	496	600	625
LUTs	652	702	752	873	992
Slices	276	284	324	364	444
DSP48Es	2	2	2	2	2
RAMB18	2	2	2	2	2
Maximum frequency (MHz)	197.316	191.314	186.359	142.025	144.718
Latency (clock cycles)	896	2048	4608	10240	22528

Table 4.2: Resource Area of Serial Pipelined Butterfly FFT on Virtex 5

Parameters	FFT SIZE				
	128	256	512	1024	2048
Flip Flops	434	456	478	619	646
LUTs	616	641	676	767	826
Slices	260	272	290	336	348
DSP48Es	2	2	2	2	2
RAMB18	4	4	4	4	4
Maximum frequency (MHz)	233.754	221.043	179.244	217.817	192.160
Latency (clock cycles)	896	2048	4608	10240	22528

The Mean Square Error (MSE) was tested, i.e. a comparison was made between the output of MATLAB's FFT function, and the output of the FFT as implemented in VHDL code *(y_VHDL).* A random input complex vector was used as the input to both, and the average error power is calculated, divided by the FFT size as in Equation (4.27).

$$average\_error\_power = MSE = \frac{\sum_{n=0}^{NFFT-1} (y\_Matlab(n) - y\_VHDL(n))^2}{NFFT} \quad (4.27)$$

The signal to noise ratio was calculated by dividing the average power of the MATLAB FFT *(y_Matlab)* output by the average error power (MSE), as shown in Equations (4.28) and (4.29).

$$average\_signal\_power = \frac{\sum_{n=0}^{NFFT-1} (y\_Matlab(n))^2}{NFFT} \quad (4.28)$$

$$signal\_to\_noise = \frac{average\_signal\_power}{average\_error\_power} \qquad (4.29)$$

The MSE and SNR for both FFTs (based on the butterfly architecture) are shown in Table 4.3 and Table 4.4, for different FFT sizes. The error is seen to increase with the FFT size. The input vector precision is 16 bits, with bits <1, 15>, while the twiddle factor precision is 8 bits, also denoted as <1, 7> bits. There are slight differences in the SNR of the two architectures due to the different input sequence vectors that are applied.

Table 4.3: The Mean Square Error and the Signal to Noise Ratio for Serial Butterfly Architecture

Parameters	FFT SIZE				
	128	256	512	1024	2048
Mean Square Error	0.0832	0.17	0.555	1.237	3.02
Signal to Noise Ratio	509.55	479.61	321.94	273.96	228.12
Signal to Noise Ratio (dB)	62.3353	61.7297	57.7437	56.1298	54.2987

Table 4.4: Mean Square Error and Signal to Noise Ratio for Serial Pipelined Butterfly FFT Architecture

Parameters	FFT SIZE				
	128	256	512	1024	2048
Mean Square Error	0.0714	0.19	0.55	1.14	3.021
Signal to Noise Ratio	543.12	442.054	321.94	277.724	228.126
Signal to Noise Ratio (dB)	62.9733	60.9143	57.7437	56.2663	54.2813

4.9 Impact Effect of Twiddle Factor Precision on Signal to Noise Ratio

This research focuses on fixed point design for FPGA implementation. The FFT based on the serial butterfly, and the FFT based on the serial pipelined butterfly, both have equal response to the signal to noise ratio test, as shown in Table 4.3 and Table 4.4 .

This section describes how a test for one of those architectures was performed to evaluate the effect of twiddle factor precision on the accuracy of the FFT. The twiddle factor is generated as in Equations (4.30) and (4.31).

$$w\_re\_\text{int} = fix(w\_re\_fl \times (2^{Twddl\_prcision} - 1) / \max(abs(w\_re\_fl))) \quad (4.30)$$

$$w\_im\_\text{int} = fix(w\_im\_fl \times (2^{Twddl\_prcision} - 1) / \max(abs(w\_re\_fl))) \quad (4.31)$$

The basic idea is to convert the floating point number to a Q format number. Q format numbers are fixed point numbers that are stored and operated upon as regular binary numbers (i.e. signed integers), where *w_re_int* and *w_im_int* are signed integers for the twiddle factor in Q format. To convert a number from floating point to Q format, it is multiplied by 2^n and rounded to the closest integer, where *n* is the number of the bits used to represent the floating point number in fixed point format; here this is denoted by *Twddl_prcision*.

The natural binary representation of an *n*-bit word length is an unsigned integers from 0 to 2^n*-1*. The range of the twiddle factor is from -1 to 1, hence a signed integer representation is required; two's complement is used to represent signed integers. In 2's complement representation, an *n*-bit word represents integers from -2^{n-1} to $2^{n-1}-1$. To find this, the floating point number is multiplied by power two of the twiddle factor precision subtracted from one. To improve the accuracy the value is divided by the maximum absolute value of the floating point numbers.

To evaluate the effect of the word length of the twiddle factor on the accuracy of the FFT, several twiddle factor precisions were specified for the FFT, and the signal to noise ratio was calculated.

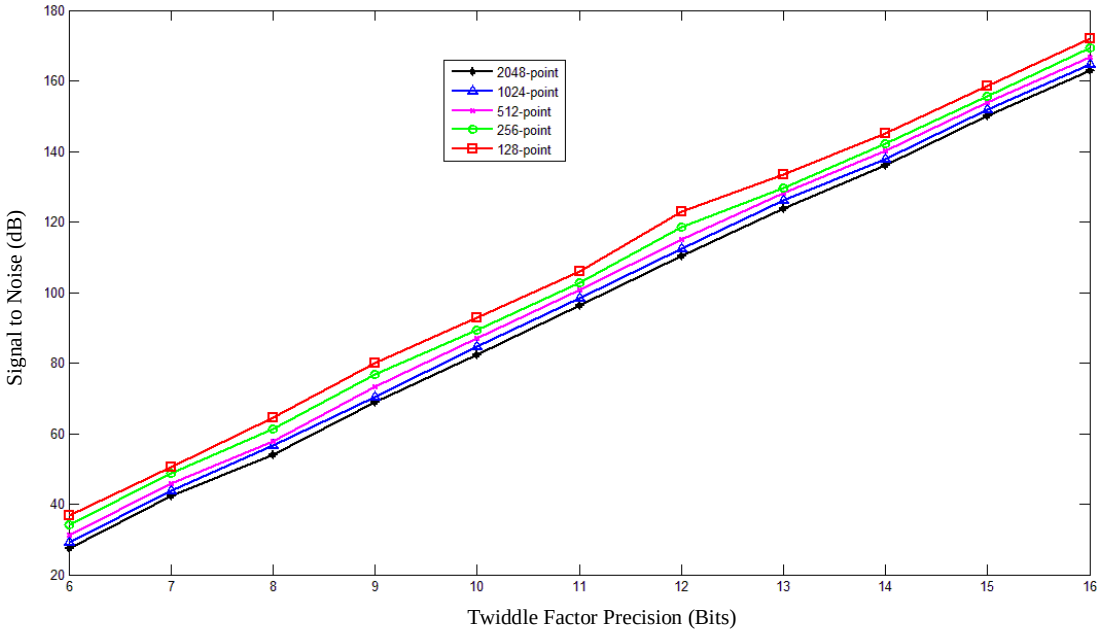

Figure 4.15: Signal to Noise Ratio for Various Twiddle Factor Precisions (Serial Butterfly FFT)

From Figure 4.15, two points are apparent. The first is that, as the FFT size increases, the signal to noise ratio decreases. The second is that, as the twiddle factor wordlength increases, the signal to noise ratio increases.

The FFT was also tested with a QAM modulation scheme as shown in Figure 4.16, where the unit under test was a fixed point IFFT core specified in VHDL code with variable twiddle factor precision. The constellation diagrams for each level of precision are provided, and these correspond to the precisions considered in Figure 4.15.

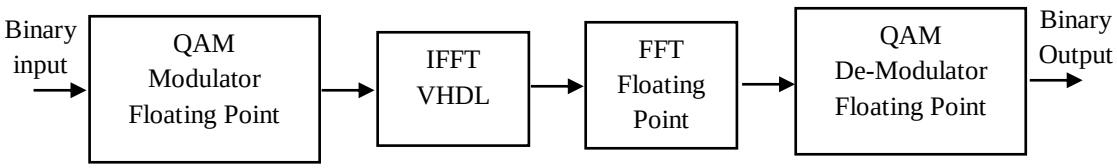

Figure 4.16: Variable Twiddle Factor IFFT Test with QAM

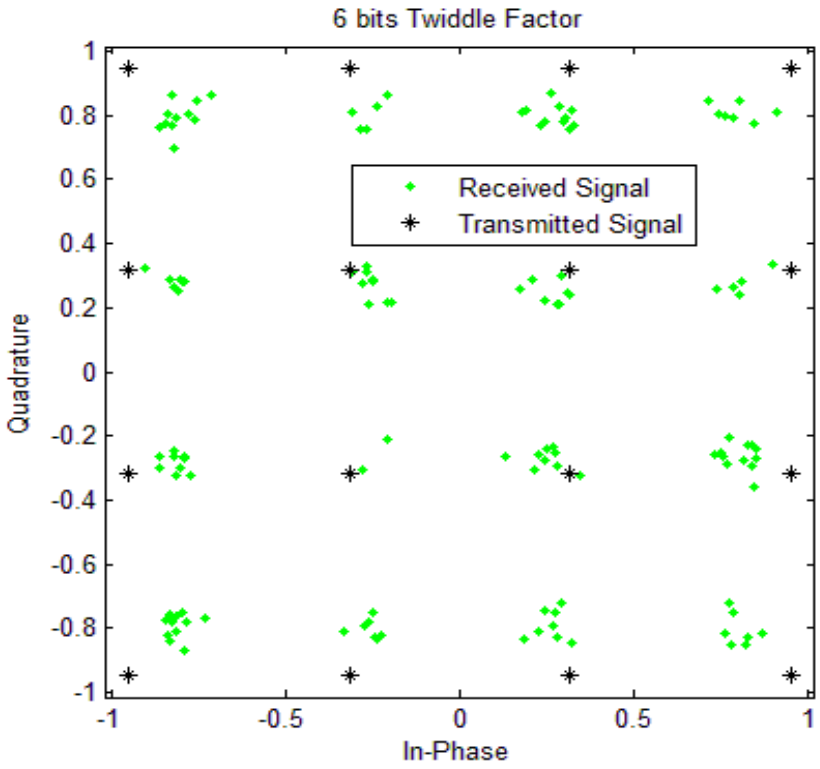

Figure 4.17: QAM Constellation Diagram (6 bit Twiddle Factor)

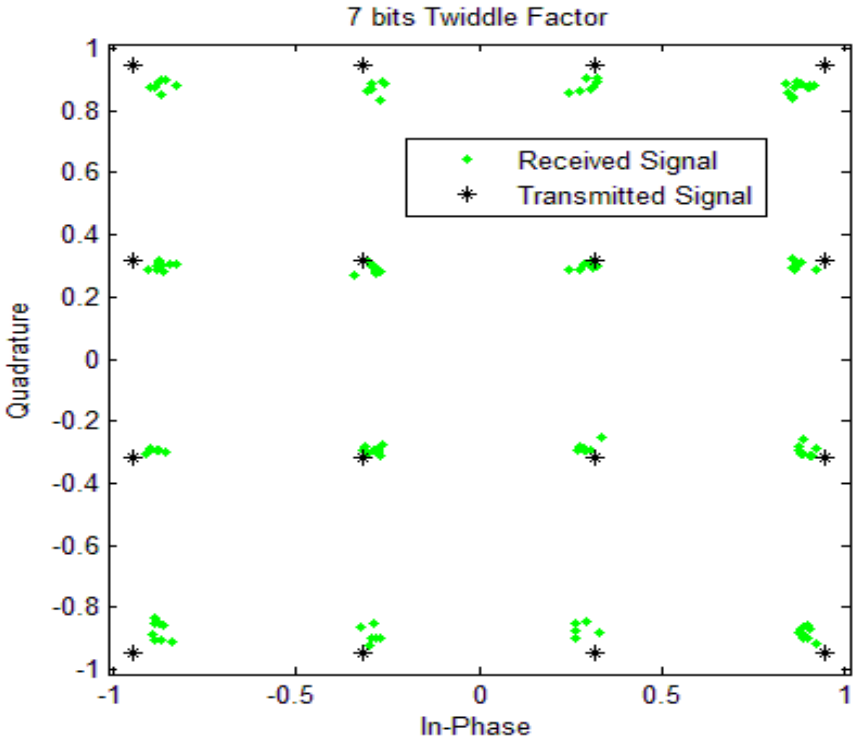

Figure 4.18: QAM Constellation Diagram (7 bit Twiddle Factor)

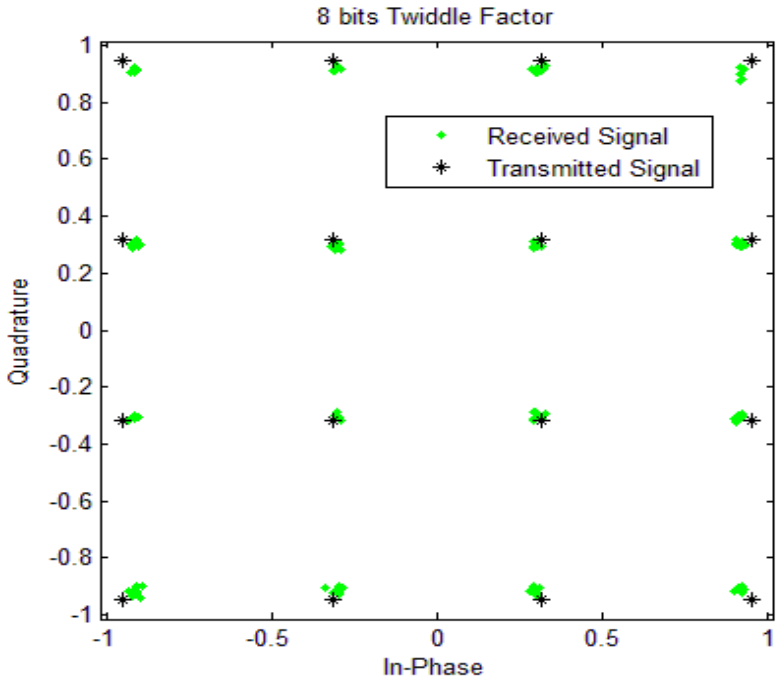

Figure 4.20: QAM Constellation Diagram (8 bit Twiddle Factor)

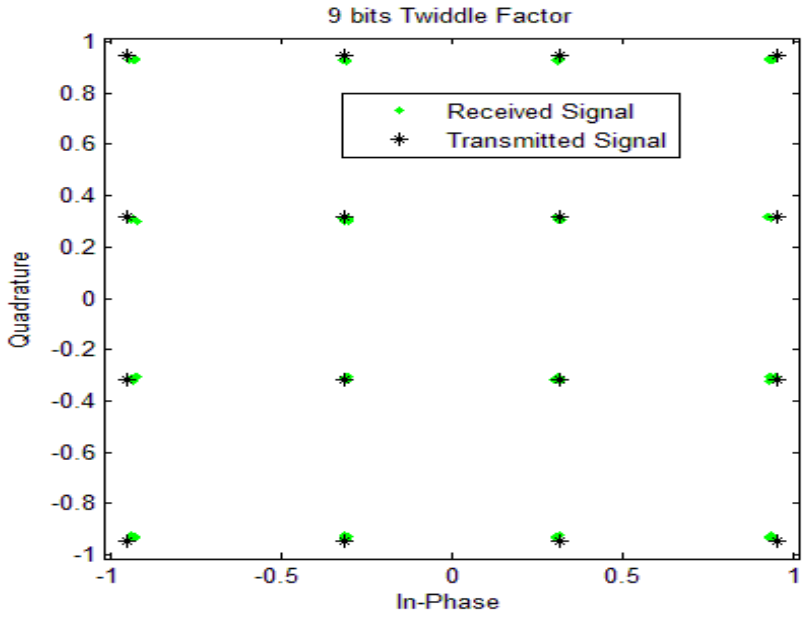

Figure 4.19: QAM Constellation Diagram (9 bit Twiddle Factor)

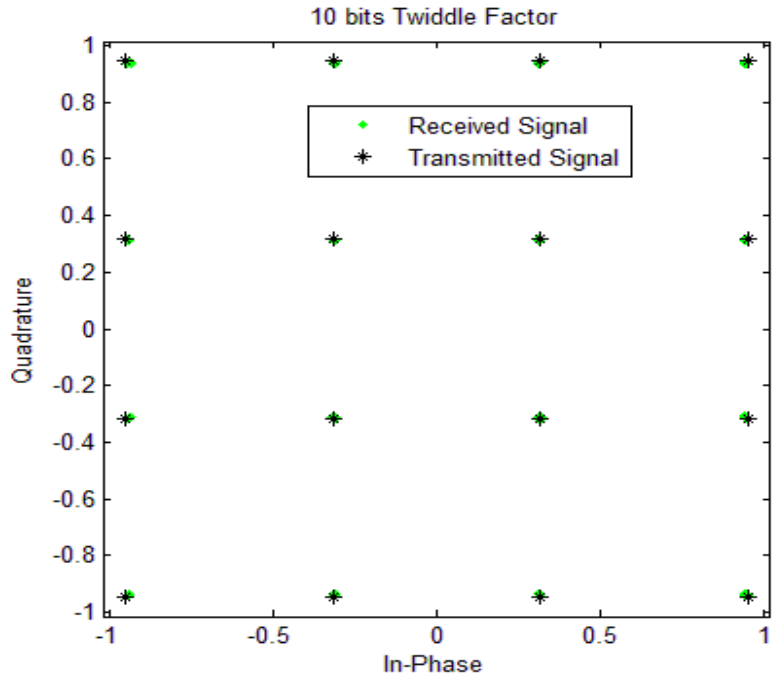

Figure 4.21: QAM Constellation Diagram (10 bit Twiddle Factor)

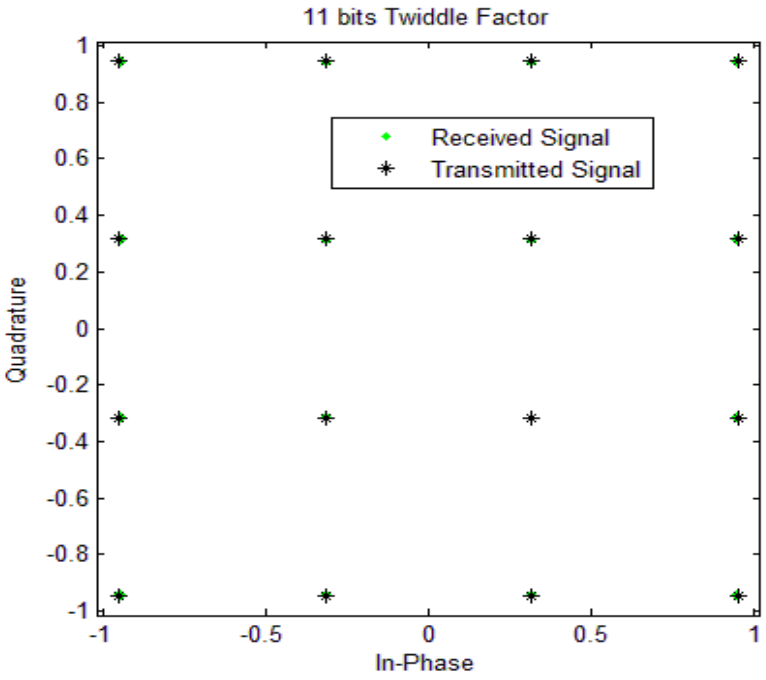

Figure 4.22: QAM Constellation Diagram (11 bit Twiddle Factor)

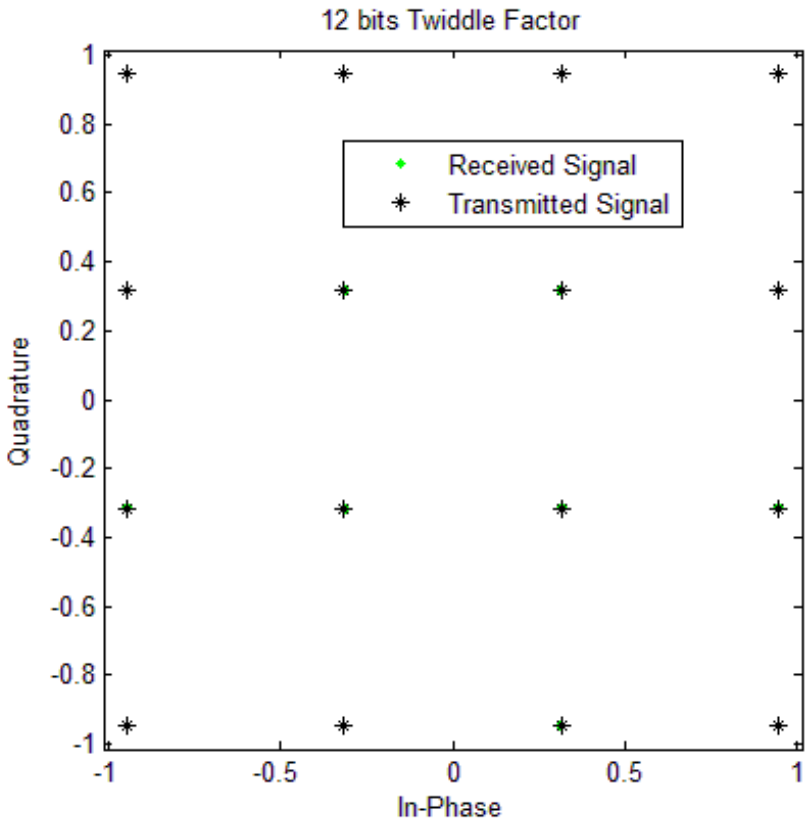

Figure 4.23: QAM Constellation Diagram (12 bit Twiddle Factor)

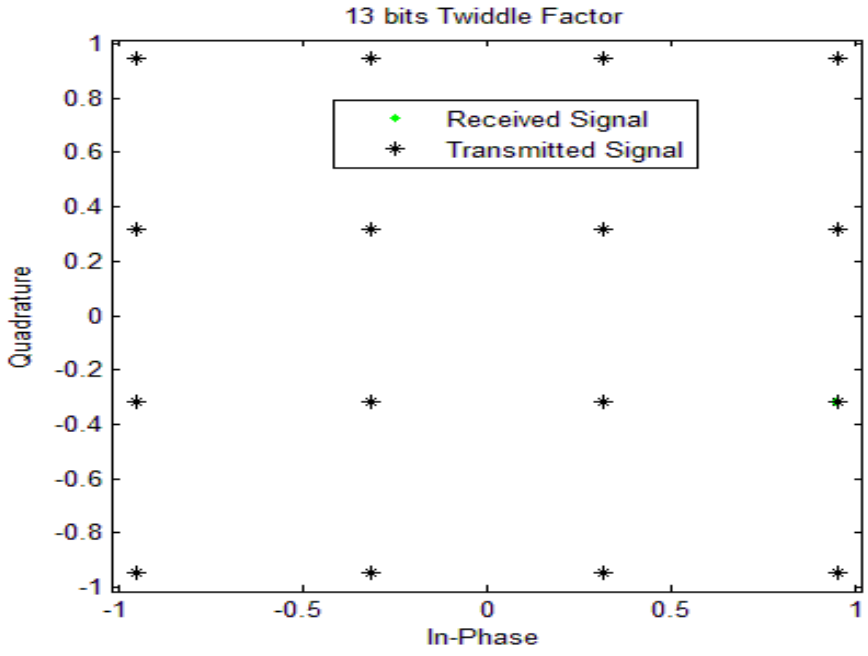

Figure 4.24: QAM Constellation Diagram (13 bit Twiddle Factor)

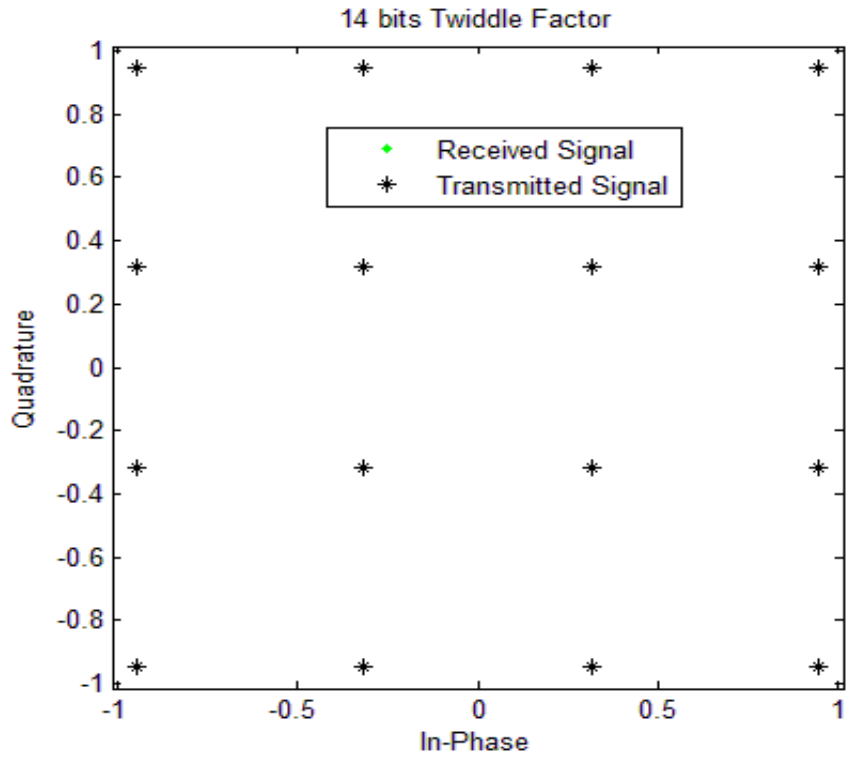

Figure 4.25: QAM Constellation Diagram (14 bit Twiddle Factor)

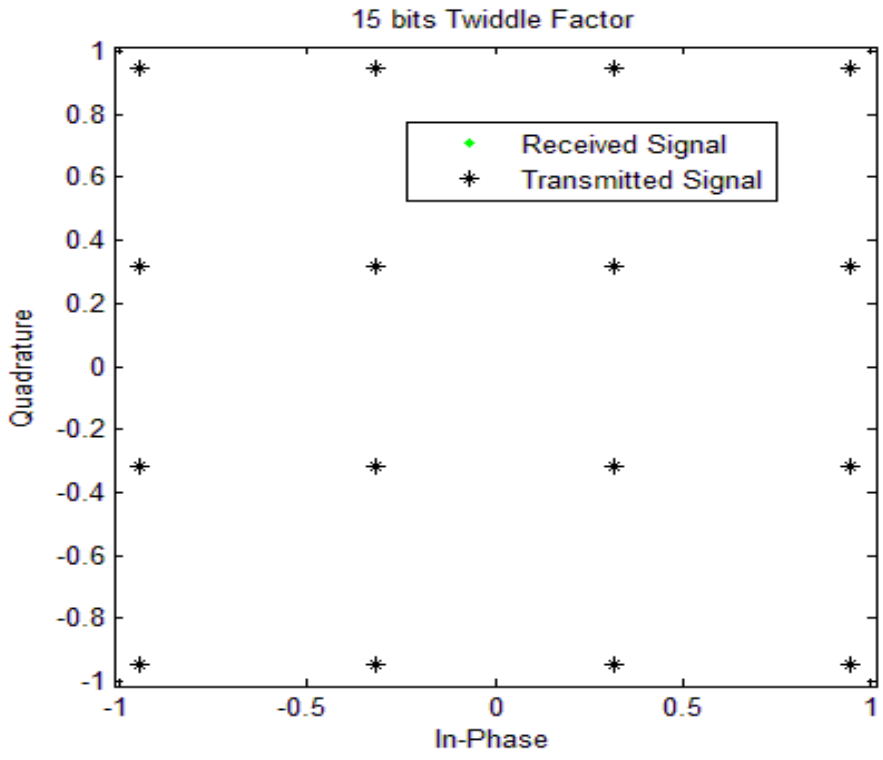

Figure 4.26: QAM Constellation Diagram (15 bit Twiddle Factor)

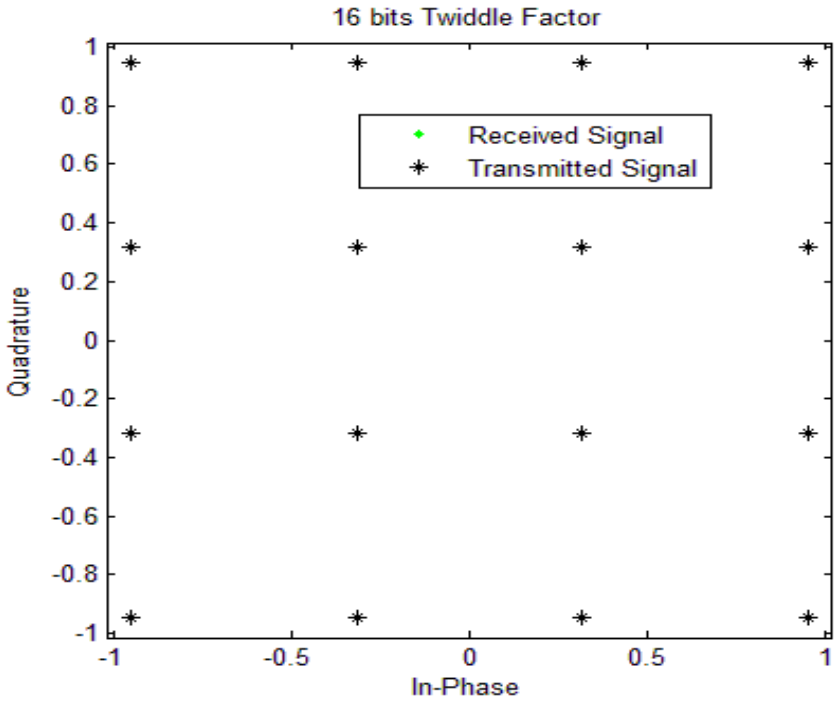

Figure 4.27 : QAM Constellation Diagram (16 bit Twiddle Factor)

In this test, a variable twiddle factor precision is used to explore the twiddle factor word length effect on the efficiency of the FFT. The received samples are represented by green diamonds, while the transmitted symbols are denoted by black star. The test uses floating point models for both the QAM transmitter and receiver and for the FFT. The test unit is the VHDL IFFT fixed point model as shown in Figure 4.16, which is the only possible error source in the test. In the case of using a fixed point VHDL model for both the FFT and IFFT, the error is doubled. The worst case for the IFFT calculation is when the twiddle factor precision is 6 bits <1, 5> as Figure 4.17. While the ideal case for the IFFT calculation is when the twiddle factor precision is 16 bits <1, 15> as shown in Figure 4.27, we can conclude from Figure 4.15 and the constellation diagrams that as the twiddle factor word length decreases, the efficiency of the FFT decreases, and as the twiddle factor word length increase, the FFT efficiency increases as well. In the hardware design, this is related to cost, and hence an 8 bits twiddle factor is preferred as a trade-off for the FFT efficiency and complexity.

5 Fast Fourier Transform Implementation on FPGA Based on CORDIC

5.1 Introduction

In this Chapter, an FFT implementation based on the CORDIC algorithm is presented. The CORDIC is used to perform complex multiplication with shift-add processes. The FFT architecture is modified to generate the phase angle in section 5.4. This technique can offer resource area instead of using two large ROMs to store phase angles. In section 5.6, the FFT based on Butterfly and FFT based on CORDIC are compared based on the Xilinx 7.1 FFT core. The designs are upgraded to use two clocks to improve the throughput in order top to meet the requirement of 4^{th} generation standards in section 5.7.

5.2 FFT Based on CORDIC

Another approach, based on CORDIC, has been used to implement the FFT function and its inverse. In this approach, CORDIC is used to perform multiplication. The aim is to design an FFT without any explicit multiplier. The architecture developed is for the radix-2 decimation-in-frequency FFT.

Sequential architecture has been used as shown Figure 5.1. It consists of two RAMs, a ROM, a CORDIC processor, an FSM, and an address generator unit. The architecture has been implemented using MATLAB scripts and VHDL code. The design has the interface shown previously in Figure 4.1.

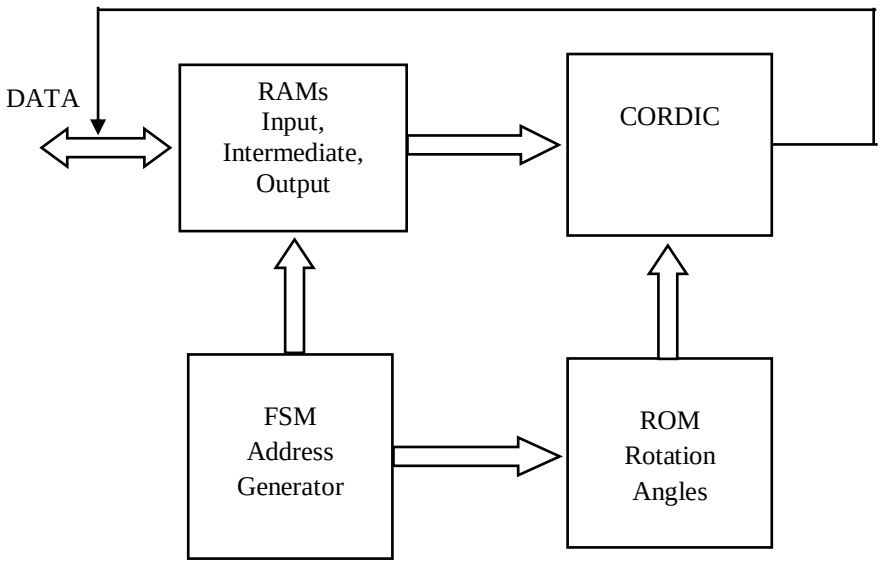

Figure 5.1: Sequential FFT Architecture Based on CORDIC

5.2.1 CORDIC ROM

A single ROM is used to store the angles of twiddle factor rotation. Equation (5.1) is used to calculate the angles.

$$\theta(n) = \left(\frac{2\pi n}{NFFT} \right) \quad , n = 0,1,2,\ldots\ldots\ldots\ldots, \; NFFT - 1 \qquad (5.1)$$

A MATLAB script was used to implement Equation (5.1), and the calculated values stored in Hexadecimal format in the ROM. The precision used was <1, 15>, i.e. one integer and fifteen fractional bits. The ROM size is $log_2(NFFT)$. For the IFFT, the angle is inverted. In this case, the same FFT architecture is used to perform the FFT and its inverse.

5.2.2 CORDIC For Radix-2

The CORDIC processor is used to perform multiplication. The CORDIC performs multiplications by rotating the input vector by discrete angles. A pipelined parallel architecture is used to create the CORDIC processor inside the FFT. The CORDIC is operated in rotation mode with eleven cells. A block diagram of the CORDIC architecture is shown in Figure 5.2 . The angles that CORDIC needs to rotate by are larger than 90^o, so a mapping and de-mapping circuit is required. The *X_re* and *X_im* values are as given in Equations (4.11) and (4.12).

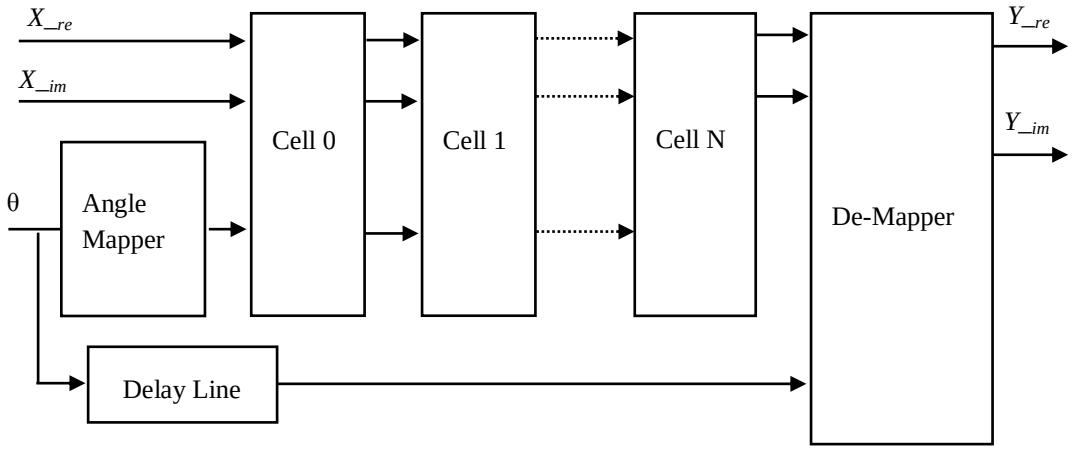

Figure 5.2: CORDIC for FFT

5.2.3 CORDIC Scaling Factor Implementation

The output of the CORDIC processor is multiplied by a constant gain of 0.6073 as given in Equation (2.38). A shifts and adds operation is used to implement this multiplication, and as a result the synthesised design has no multipliers.

5.3 FFT Based CORDIC Test

VHDL code was written for the FFT architecture shown in Figure 5.1, and the design was synthesised using the Xilinx ISE tool. The resources occupied by the design are summarised in Table 5.1. The design has been tested with 128, 256, 512, 1024 and 2048 FFT sizes. The complex input vector precision is <1, 15>, i.e. one integer bit and fifteen fractional bits. The angles used by CORDIC are stored in a ROM with <1, 15> bits precision. The complex output of the FFT is permitted to grow to <12, 15> bits. The CORDIC-based FFT has no multipliers. The maximum clock frequency the design can attain is around 200 MHz, which is sufficient for many OFDM applications.

Table 5.1: Resources utilisation of fully parallel, pipelined CORDIC FFT Architecture on Virtex 5 X110T

Parameters	FFT SIZE				
	128	256	512	1024	2048
Flip Flops	1,122	1,165	1,208	1,251	1,294
LUTs	1,453	1,515	1,570	1,628	1,697
Slices	481	482	504	528	560
DSP48Es	-	-	-	-	-
RAMB18	3	3	3	3	5
Maximum frequency (MHz)	216.076	228.206	181.324	204.082	199.720
Latency (clock cycles)	896	2048	4608	10240	22528

The MSE and SNR of the design were evaluated, and the relevant figures are given in Table 5.2. The error increases with the FFT size.

Table 5.2: Mean Square Error and Signal to Noise Ratio for parallel CORDIC FFT

Parameters	FFT SIZE				
	128	256	512	1024	2048
Mean Square Error	0.056	0.16	0.369	0.856	2.09
Signal to Noise Ratio	678.25	579.501	485.84	399.48	334.65
Signal to Noise Ratio (dB)	65.1952	63.6217	61.8588	59.9016	58.1309

The floating point model is tested to prove its accuracy as shown in Figure 5.3. The FFT Matlab built in function is considered as a golden model of the signal to

noise ratio that show the floating point model is working properly. The SNR figures obtained indicate that the floating point model is accurate.

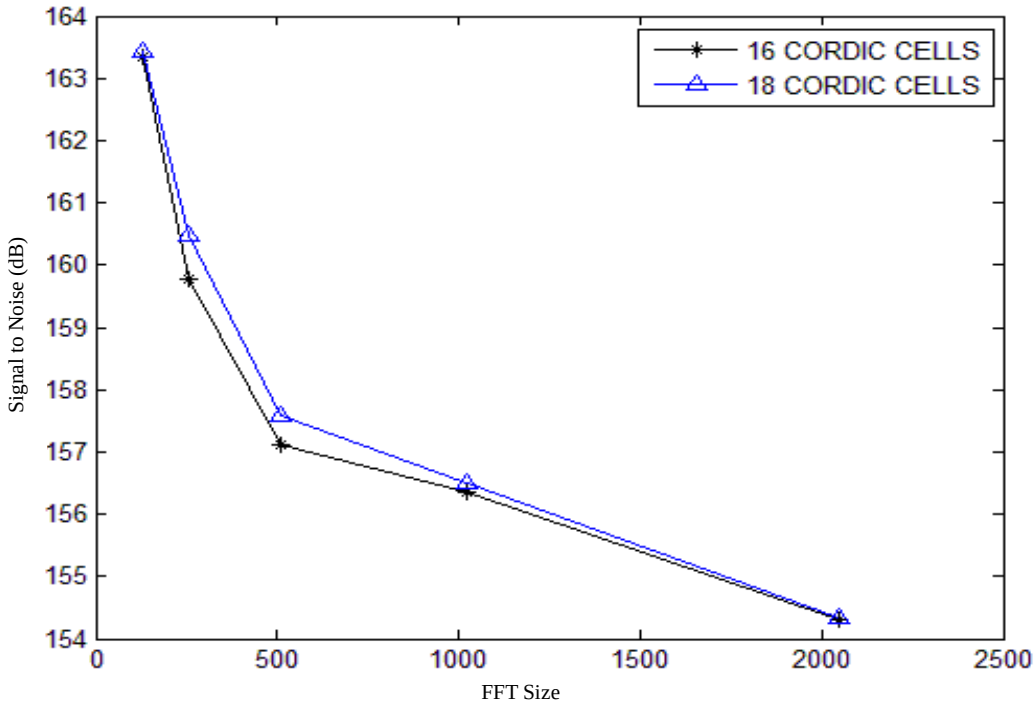

Figure 5.3: CORDIC Floating point Model Test Compared With MATLAB FFT Function

5.4 Upgraded FFT based CORDIC with Generated Angles

In the CORDIC-based FFT design in Section 5.2.1, a look-up table is used to store the twiddle factor angles that are used to rotate the input vectors to implement multiplication. This section describes a circuit which was designed to generate these angles, instead of storing them, as depicted in Figure 5.4 . The FFT twiddle factor angles follow Equation (5.1) in Section 5.2.1, and are equivalent to a constant value equal to $2\pi/NFFT$, multiplied by n, where $NFFT$ represents the FFT size, and n is the index, i.e. an integer ranging from 0 up to $NFFT$-1. These values were stored in a ROM in the previous CORDIC-based FFT design.

For a more efficient design that can offer a resource cost reduction, the angles may be calculated by generating the values of n using a controlled counter, and multiplying them by a constant. The controller circuit is responsible for constraining

the counter values to be within the specific range required by each stage of the FFT. The resource utilisation of the CORDIC-based FFT with generated angles is summarised in Table 5.3. A shift and add technique is used to implement the multiplication, rather than using an explicit multiplier. The design saves one Block RAM compared to the CORDIC-based FFT with stored angles.

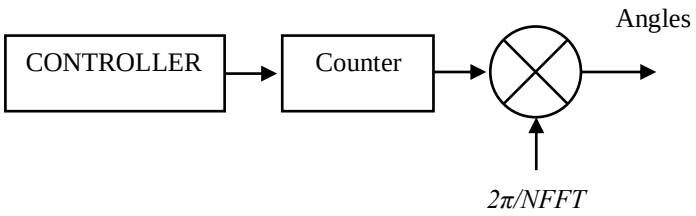

Figure 5.4: FFT Twiddle Factor Angles Generation

Table 5.3 : Resource Utilisation of FFT based on CORDIC with Generated Angles on Virtex 5

Parameters	FFT SIZE				
	128	256	512	1024	2048
Flip Flops	1,232	1,288	1,353	1,386	1,436
LUTs	1,593	1,679	1,759	1,796	1,867
Slices	504	542	595	583	594
DSP48Es	-	-	-	-	-
RAMB18	2	2	2	2	2
Maximum frequency (MHz)	209.468	211.506	208.377	209.952	223.714
Latency (clock cycles)	896	2048	4608	10240	22528

The CORDIC-based FFT with generated and stored angles was tested using the circuit shown in Figure 5.5. Binary data was applied to a floating point QAM modulator to generate test symbols for the FFT designed in VHDL. This QAM constellation was taken as a reference of accuracy, as shown in the figures which follow.

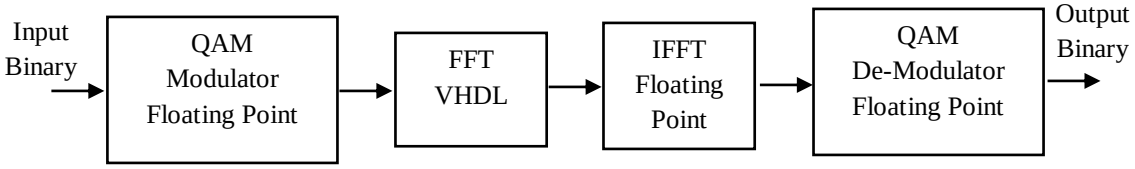

Figure 5.5: FFT Based CORDIC Test with QAM

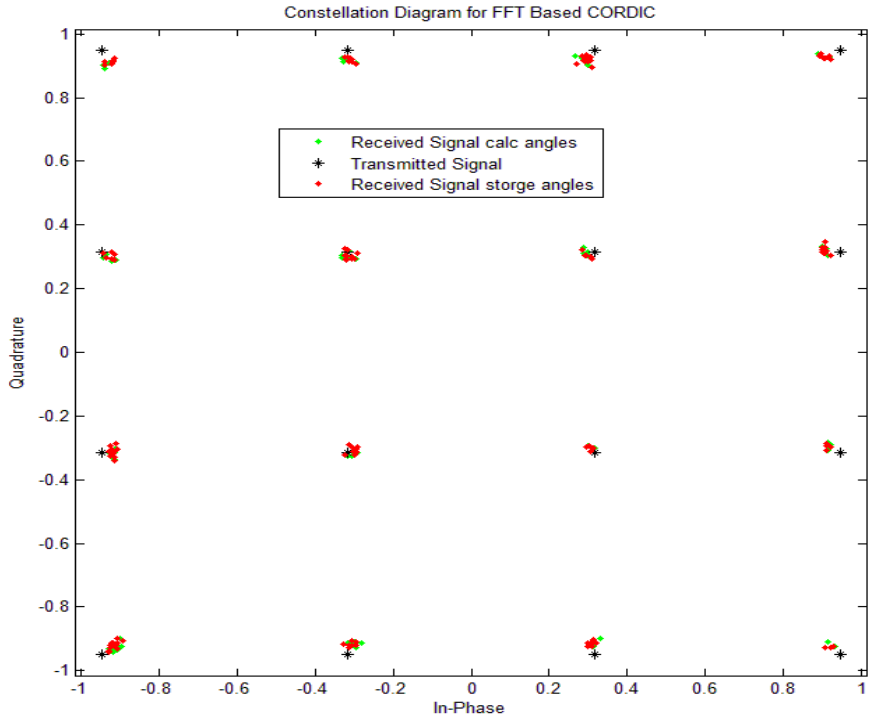

Figure 5.6: Constellation Diagram of 128 Point FFT based on CORDIC

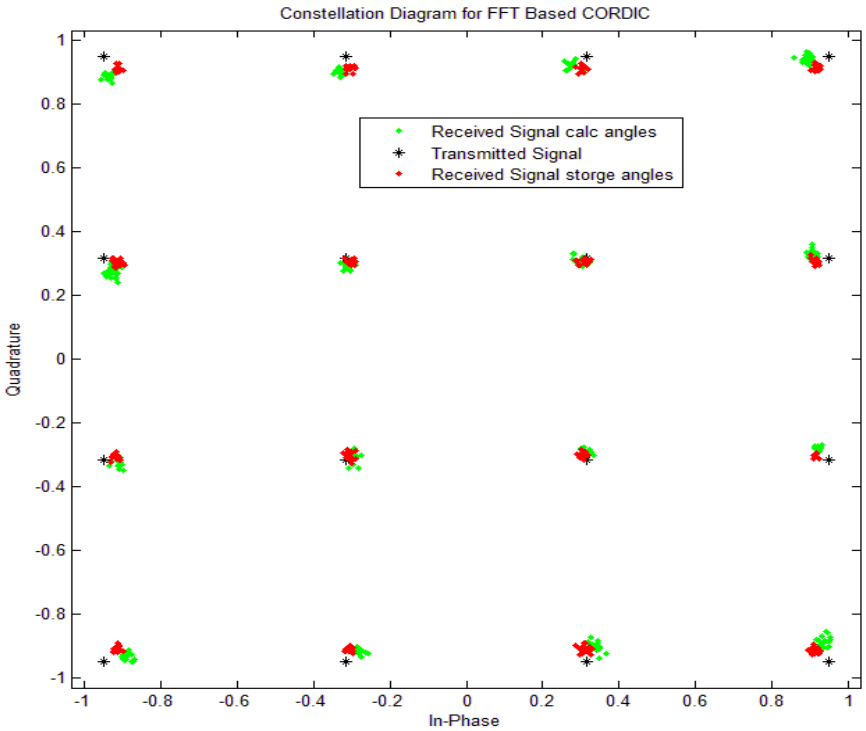

Figure 5.7 : Constellation Diagram of 256 Point FFT based on CORDIC

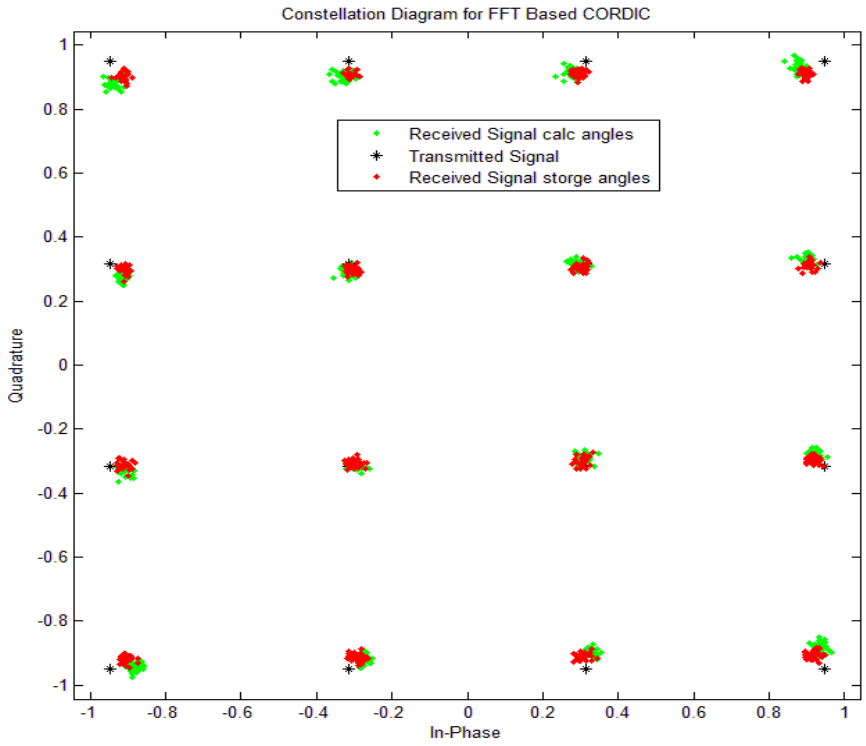

Figure 5.8: Constellation Diagram of 512 Point FFT based on CORDIC

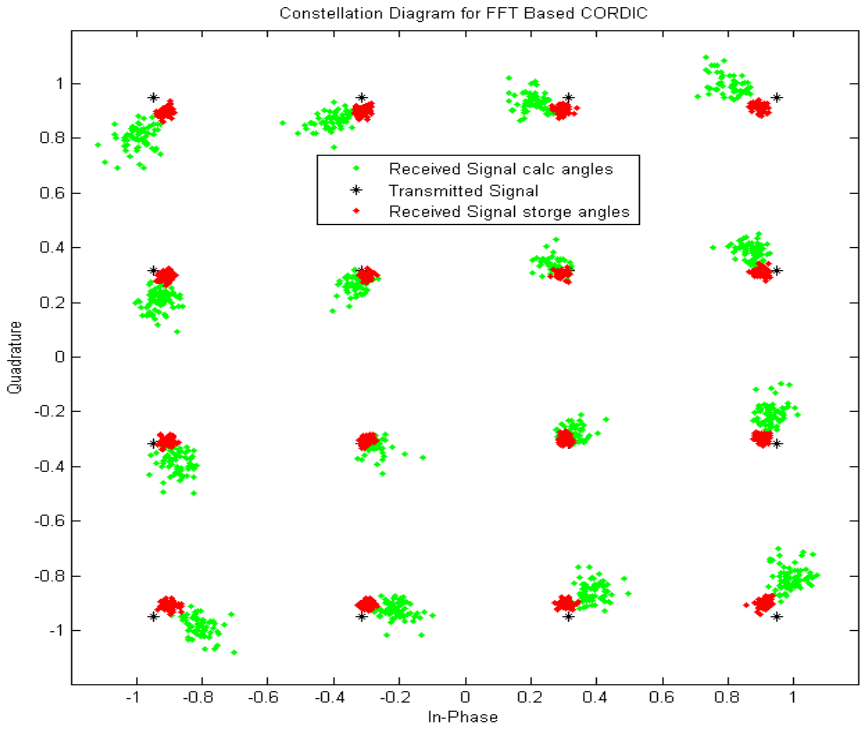

Figure 5.9: Constellation Diagram of 1024 Point FFT based on CORDIC

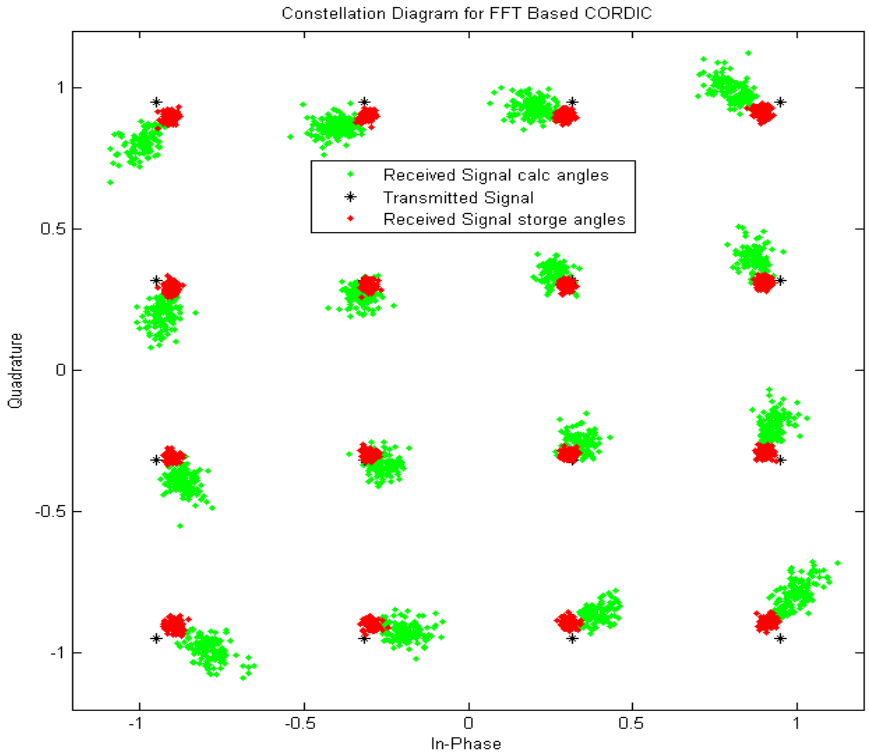

Figure 5.10: Constellation Diagram of 2048 Point FFT based on CORDIC

In this test, two FFT based CORDIC architectures are compared. The first one uses a look-up table to stores the twiddle factor angles required by CORDIC. Its received signal is represents by red squares. The second architecture calculates the twiddle factor based on Figure 5.4. Its received signal is represented by green squares. The black stars denote the values of the transmitted signal. The generated angles do not match the stored angles, and give different results in the test. The mismatch between the two techniques is due to the accuracy of the generated sequence of the counter in Figure 5.4. The stored angle method is more accurate than the generated angles as shown in Figure 5.6, Figure 5.7, Figure 5.8, Figure 5.9 and Figure 5.10. As the FFT size increases, the error increases due to the increase in the number of stages. Whilst the test uses floating point models for QAM transmitter and receiver as well as for the IFFT, VHDL FFT test unit relies on a fixed point model and is therefore the only possible source of error. In the case of using a fixed point VHDL model for both the

FFT and IFFT, the error doubles. With a higher modulation index such as 64QAM, it has been observed that the error can increase further.

5.5 The Effect of CORDIC Iterations on Signal to Noise Ratio

The principles of CORDIC algorithms have been explained in Section 2.3.1. Each iteration in CORDIC represents a cell. As the number of the cells increases, the accuracy of the calculation increases. In this section, the FFT based on CORDIC was tested to evaluate the relationship between the number of cells and the resulting SNR. The SNR is calculated as previously in (4.27),(4.28) and (4.29), between Matlab's floating point FFT function and VHDL code FFT but based on different numbers of cells. The range of the cells is selected in the range between six and 16. The SNR was seen to increase dramatically from around 60dB for six CORDIC cells, to around 150dB for eleven cells. Beyond that, at twelve cells and above, the effect of increasing the number of iterations was less significant, reaching a maximum of around 160dB. The reason for this phenomenon is that the angle cells are represented with 8 only bits, which makes the last five cells ineffective. Based on this observation, the number of cells was therefore limited to eleven. A graph of SNR against CORDIC iterations – which is equivalent to the number of cells -- is shown in Figure 5.11.

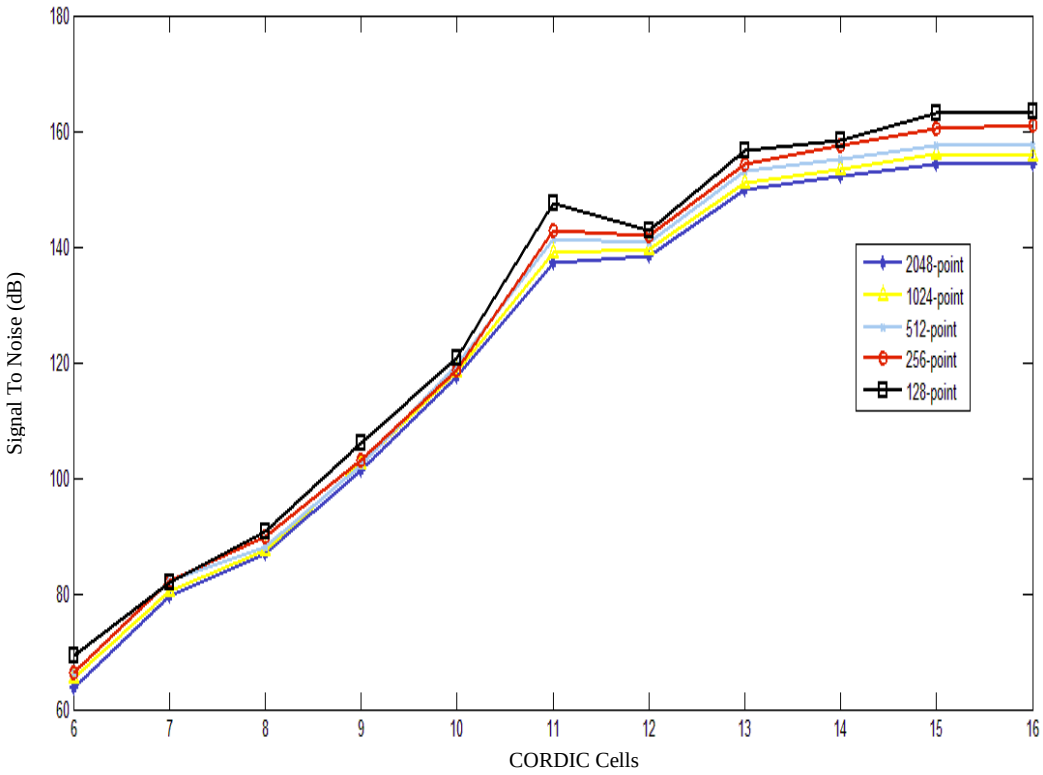

Figure 5.11: Variation of SNR with Number of CORDIC Cells

5.6 Comparison to Xilinx FFT Version 7.1

Xilinx LogiCORE offers Intellectual property (IP) cores that can be synthesised and implemented on Xilinx FPGAs. FFT v7.1 is an IP core for the Fast Fourier Transform and its inverse. It has different architectures, but the architecture corresponding to the FFT being researched is the Radix-2 burst I/O architecture. The latest version of the core is 7.1, and the results of synthesising this core are as shown in Table 5.4.

These results have been generated for several different FFT sizes, with the complex input vector set to sixteen bit precision <1, 15>, while the twiddle factor precision is set to eight bits <1, 7>. The output order is selected as natural order, and the output precision is selected as growth with no scaling, and carries all significant integer bits to the end of the computation, which is as defined in Equation (4.3).

The FFT 7.1 core is compared to the serial butterfly FFT, serial pipelined FFT and CORDIC FFT architectures. In terms of the number of flip flops used, the serial pipelined architecture is seen to be the cheapest, as shown in Figure 5.12, while the serial butterfly uses the next fewest, followed by the Xilinx FFT 7.1 core. The CORDIC-based FFT implements the CORDIC cells in a fully parallel, pipelined architecture, and therefore uses the most flip flops. Notably the serial pipelined and serial FFT architectures use around 50% of the flip flops required by the Xilinx FFT 7.1 core, and as the FFT size increases, the flip flops required increase dramatically.

The flip flop is one of the critical points in the design on Virtex 5 as its number is less than that offered by Virtex 4. The pipelined butterfly architecture maintains its advantages over other architectures, in terms of the number of look-up tables (LUTs) occupied, while the CORDIC FFT architecture requires the largest number of LUTs, as shown in Figure 5.13. The number of occupied slices for the serial pipelined butterfly and serial butterfly architectures are lower than for the Xilinx FFT 7.1 core, as shown in Figure 5.14.

The CORDIC FFT architecture has an advantage over the other styles of FFT implementation, as it does not use any multipliers; whereas the serial pipelined and serial butterfly architectures both use two multipliers, and the FFT 7.1 core requires up to six multipliers, depending on the size of the FFT. The serial butterfly implementation uses the fewest of Block RAMs of all the architectures considered, while the FFT 7.1 core uses up to eight, again depending on the FFT size.

Table 5.4: Resource Utilisation of FFT 7.1 on Virtex 5 X110t

Parameters	FFT SIZE				
	128	256	512	1024	2048
Flip Flops	959	835	1,149	1,202	1,258
LUTs	712	645	849	891	932
Slices	347	319	403	427	490
DSP48Es	3	4	6	6	6
RAMB18	4	4	4	5	8
Maximum frequency MHZ	392.619	459.348	350.385	273.598	292.141

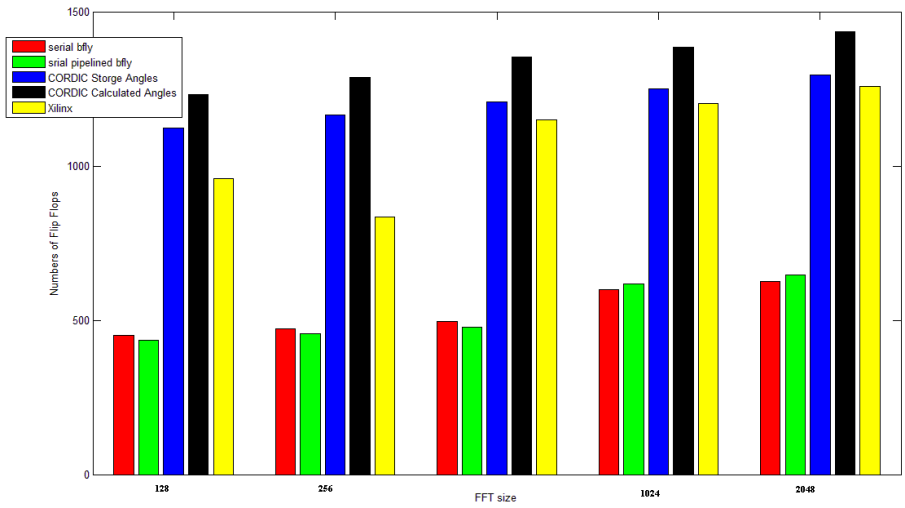

Figure 5.12: Number of Flip Flops for various FFT Architectures

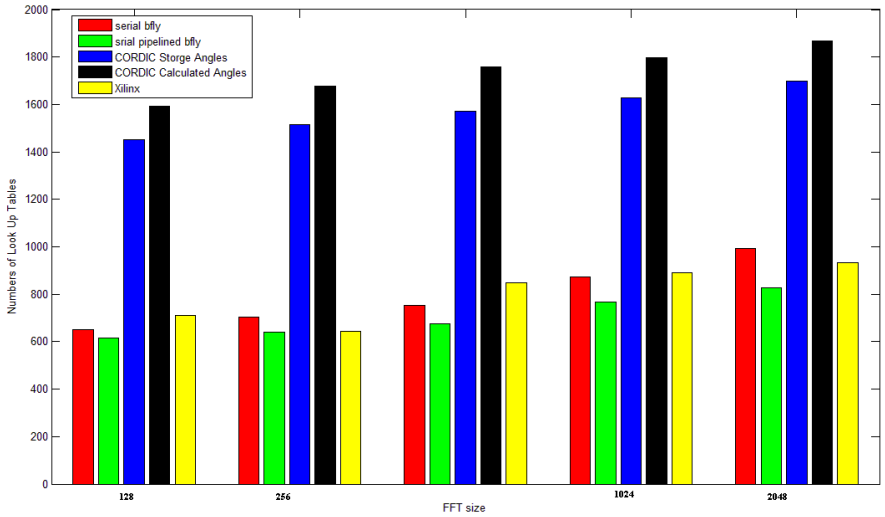

Figure 5.13: Number of Look-Up Tables for various FFT Architectures

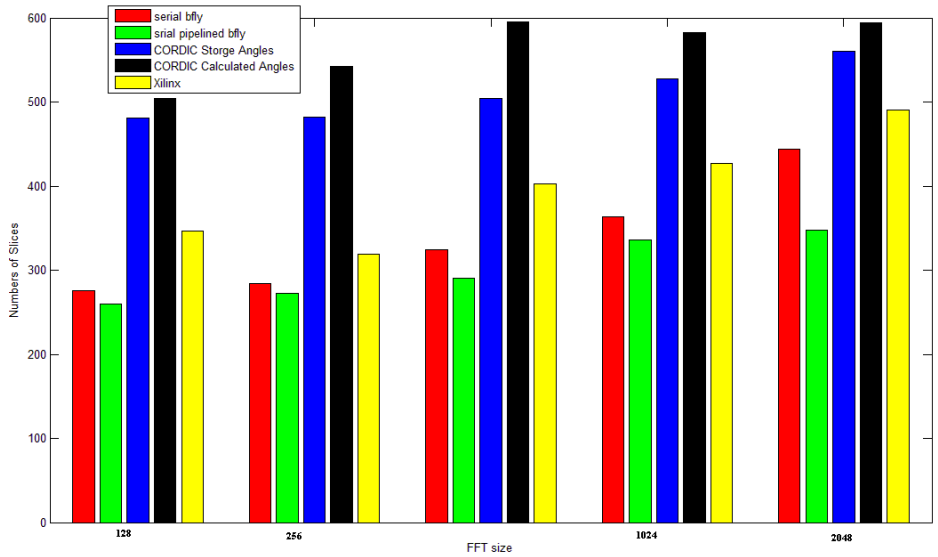

Figure 5.14: Number of Slices for various FFT Architectures

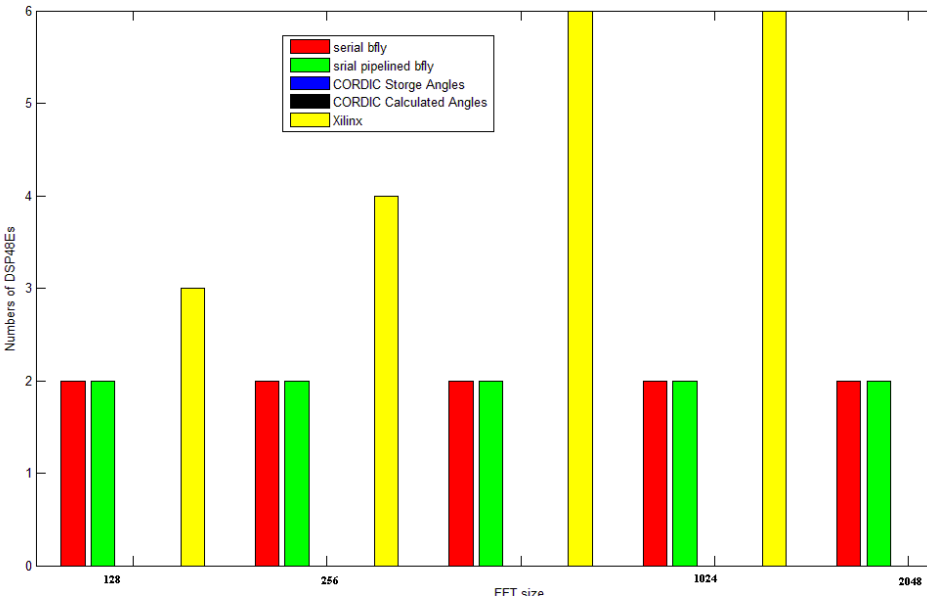

Figure 5.15:Number of DSP48Es various FFT Architectures

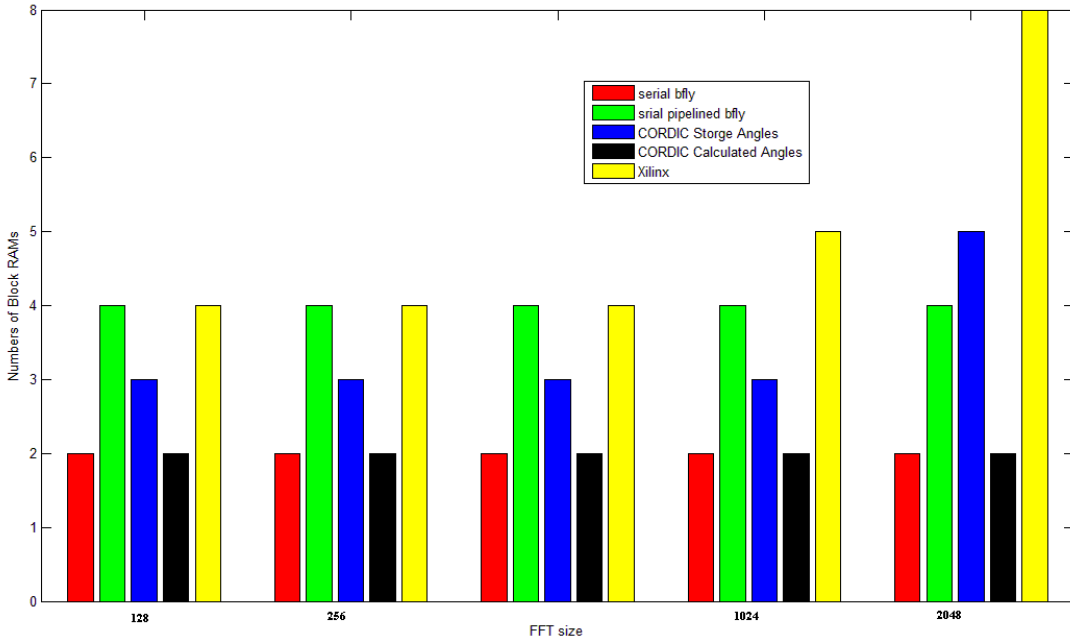

Figure 5.16: Number of Block RAMs for various FFT Architectures

5.7 Upgrade with Two Clocks for OFDM Requirements

The FFT is widely used to implement OFDM modulation and demodulation. Each wireless standard has defined OFDM parameters, as was discussed in Chapter 2. To allow both the FFT based on Butterfly, and the FFT based on CORDIC, to meet the requirements of the OFDM standard, the design was upgraded to use two clocks, one to read and write data from the interface (`clk_main`) and the other to perform the calculation inside the FFT (`clk_cal`), as shown in Figure 5.17. The interface clock, `clk_main`, is set to the sampling or chip rate, according to the OFDM parameters of the wireless standard, while `clk_cal` is set to the maximum frequency the clock can achieve.

According to the operational steps of the FFT, there are five states, as mentioned in 4.5. The first state, S0, is assigned to read and write data from the interface. S0 is responsible for generating the position of the input and output data, *in_index* and *out_index*. When this state is set to work with `clk_main`, the input and output can match the sampling rate of the wireless standards. On the other hand, all of the calculation processing undertaken inside the FFT (as controlled by states S1, S2, S3 and S4) uses the faster `clk_cal` to decrease the latency of the calculation.

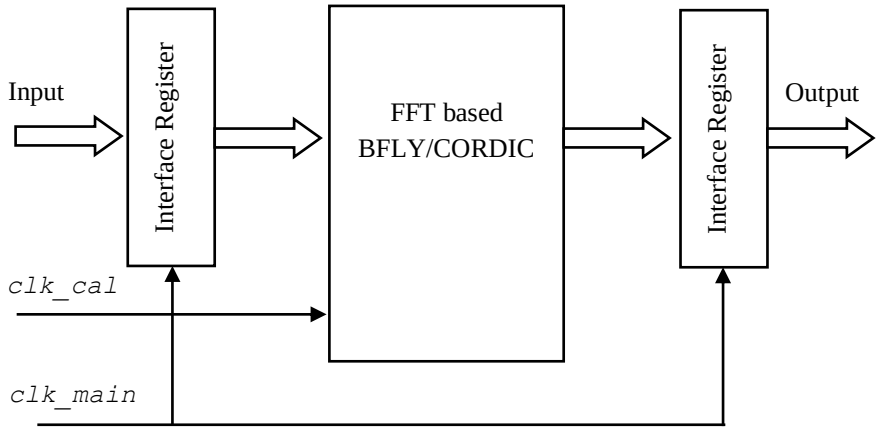

Figure 5.17: FFT for OFDM

The latency improvement achievable for a sequential architecture through the use of two clocks can be illustrated as in Figure 5.18. In this example, the OFDM symbol of IEEE 802.20 is considered. The chip rate for a 2048-point FFT is 20MHz. In case A, the FFTs based on CORDIC, serial butterfly and serial pipelined, all operate at the sampling rate. To calculate the output frame 22528 clock cycles are needed, and this is equivalent to 1.1ms for completing the calculation**.** In case B, the FFTs operate with two clocks: the first, which is responsible for reading from and writing to the interface, is operated at 20MHz, while the second, which is responsible for calculation inside the FFTs, is operated at 180MHz. In this case the output frame can be calculated in 125.1μs.

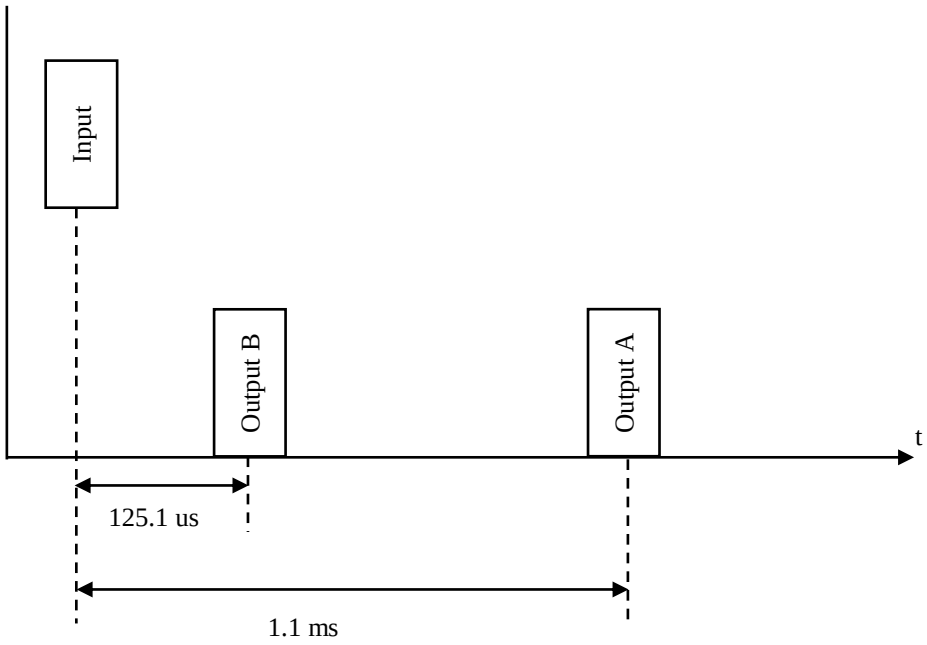

Figure 5.18: Latency Improvement for FFT

The achievable OFDM symbol rate decreases as the FFT size, *NFFT*, increases. To evaluate which FFT architectures can meet the requirements of 4G wireless standards, Equation (5.2) is used to calculate the maximum OFDM Symbol rate that the FFT can process.

$$Sampling\_Rate = \frac{clk\_cal \times NFFT}{Total\_Clock\_Cycles} \qquad (5.2)$$

In Equation (5.2), *clk_cal* is the maximum clock frequency that can be used; it should not exceed the maximum frequency stated in Table 4.1, Table 4.2, or Table 5.1, depending on the architecture being evaluated. Also, *Total_Clock_Cycles* is equal to the latency, and *NFFT* is the FFT size. The three FFT architectures (FFT based on serial butterfly, FFT based on serial pipelined butterfly, and FFT based on CORDIC), are considered to determine whether each can meet the requirements of the 4G wireless standards IEEE 802.20, IEEE 802.16e, 3GPP2_UMB and LTE, as shown below.

Setting *clk_cal* to the maximum frequency will show the maximum OFDM symbol rates that can be processed.

Table 5.5: Performance of FFT based on Serial Butterfly Architecture, with respect to 4G Wireless Standards

NFFT	Clk_cal (MHz)	Sampling Rate (MHz)	IEEE802.20	UMB	IEEE 802_16e	LTE
128	196	28	Yes	Yes	Yes	Yes
256	191	23	Yes	Yes	Yes	Yes
512	186	20	Yes	Yes	Yes	Yes
1024	142	14	Yes	Yes	Yes	yes
2048	144	13	No	No	No	No

Table 5.6: Performance of FFT based on Serial Pipelined Butterfly, with respect to 4G Wireless Standards

NFFT	Clk_cal (MHz)	Sampling Rate (MHz)	IEEE802.20	UMB	IEEE 802_16e	LTE
128	233	33	Yes	Yes	Yes	Yes
256	221	27	Yes	Yes	Yes	Yes
512	179	19	Yes	Yes	Yes	Yes
1024	217	21	Yes	Yes	Yes	Yes
2048	192	17	No	No	No	No

Table 5.7: Performance of FFT based on CORDIC, with respect to 4G Wireless Standards

NFFT	Clk_cal (MHz)	Sampling Rate (MHz)	IEEE802.20	UMB	IEEE 802_16e	LTE
128	216	30	Yes	Yes	Yes	Yes
256	228	28	Yes	Yes	Yes	Yes
512	181	20	Yes	Yes	Yes	Yes
1024	204	20	Yes	Yes	Yes	Yes
2048	198	18	No	No	No	No

Table 5.8: Performance of FFT based on CORDIC with Generated Angles, with respect to 4G Wireless Standards

NFFT	Clk_cal (MHz)	Sampling Rate (MHz)	IEEE802.20	UMB	IEEE 802_16e	LTE
128	209	29	Yes	Yes	Yes	Yes
256	211	26	Yes	Yes	Yes	Yes
512	208	23	Yes	Yes	Yes	Yes
1024	209	20	Yes	Yes	Yes	Yes
2048	223	20	Yes	Yes	Yes	NO

Table 5.9: Performance of Xilinx FFT 7.1 Core with respect to 4G Wireless Standards

NFFT	Clk_cal (MHz)	Sampling Rate (MHz)	IEEE802.20	UMB	IEEE 802_16e	LTE
128	392	112	Yes	Yes	Yes	Yes
256	459	114	Yes	Yes	Yes	Yes
512	350	77	Yes	Yes	Yes	Yes
1024	273	54	Yes	Yes	Yes	Yes
2048	292	53	Yes	Yes	Yes	Yes

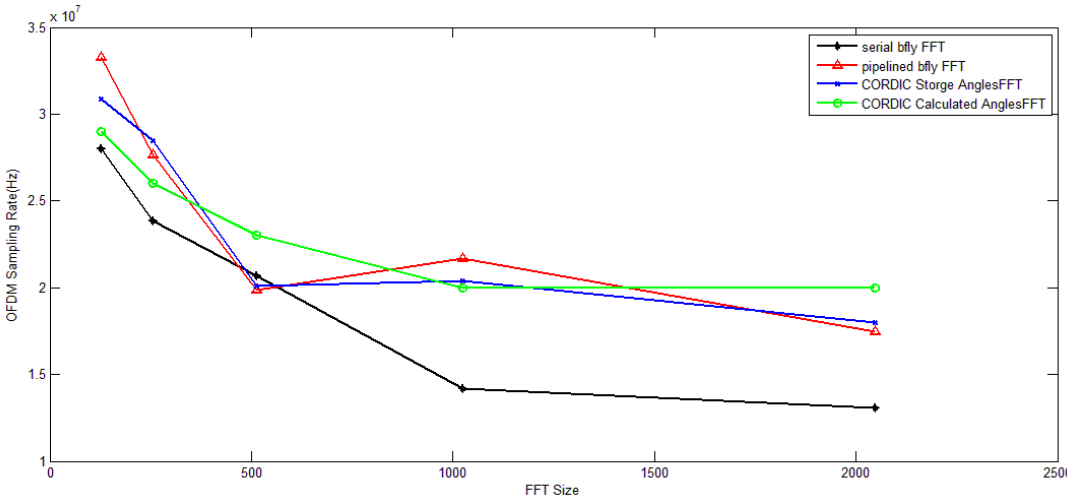

Figure 5.19: Comparison of Maximum Sampling Rates Supported by Considered FFT Architectures

6 Orthogonal Frequency Division Multiplexing Transmitter on FPGA Using Xilinx System Generator

6.1 Introduction

OFDM modulation is part of the physical layer of most recent wireless networks. This chapter considers the design of an OFDM transmitter. The OFDM scheme can support different mapping schemes, including QPSK, 8PSK, 16QAM and 64QAM. In section 6.2 the different modulation schemes are introduced. In section 6.3, the design features a dynamic IFFT that can dynamically change its size from 128 point up to 2048 point. The design can achieve variable lengths of cyclic prefix, as defined in the standards considered. This corresponds to the requirements of IEEE 802.20

and 3GPP_UMB wireless networks, and also covers most other standards. Figure 6.1 shows the block diagram of the OFDM transmitter.

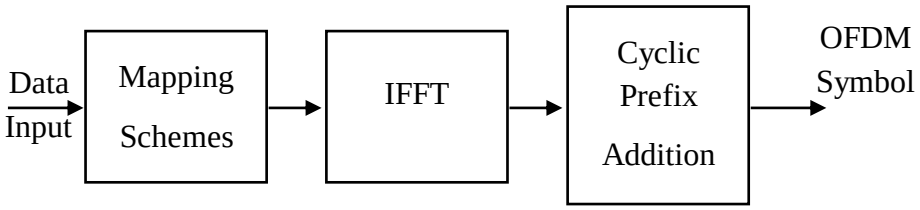

Figure 6.1: Block Diagram of OFDM Transmitter

Therefore in summary, this design has three variable aspects: the first is in the modulation scheme, the second is in the IFFT size, and the third is the cyclic prefix length.

System Generator and ISE (Integrated Software Environment) tools from Xilinx were used in the design, implementation and synthesis of the OFDM system. The design was targeted at a Xilinx Virtex 5 device, and all of the variable features were tested. The resources utilisation for each part of the design is given.

A floating point model was created in the form of MATLAB scripts for debugging and validation purposes. The mean square error was calculated.

6.2 Mapping Schemes

The QPSK, 8PSK, 16QAM, 64 QAM modulation schemes were implemented on the FPGA, with each modulator consisting of a serial to parallel converter, and pairs of ROMs. The number of bits involved in the serial to parallel conversion is set according to the modulation type: 2 bits for QPSK, 3 bits for 8PSK, 4 bits for 16QAM, and 6 bits for 64QAM. A pair of ROMs is assigned to each modulation scheme: one to store the in-phase values, and the other to store the quadrature values. Two multiplexers are assigned to choose between modulation schemes, to manage

which task a control signal with two bits is used**.** A block diagram of the variable modulation scheme is shown in Figure 6.2.

The resource utilisation of the modulator is shown in Table 6.1. The design has sixteen bit precision < 1, 15 >, i.e. one integer bit and fifteen fractional bits. The report shows that the area occupied by the circuit is small and that it can achieve very high speed.

The MSE is calculated as shown in Table 6.2. For QPSK, 8PSK and 16QAM the MSE is acceptable but for 64QAM is significantly increased due to the fixed point limitation <1, 15> which restricts amplitude values to the interval (-1, 0.9999). With 64QAM having values > 1, this inevitably leads to an increase in MSE.

For validation purposes, the signal constellations produced by the fixed point design for each modulation scheme are given. Figure 6.3 shows the constellation diagram for QPSK modulation, Figure 6.4 shows the 8PSK constellation diagram; Figure 6.5 the fixed 16QAM constellation diagram; and finally Figure 6.6, the 64QAM constellation diagram.

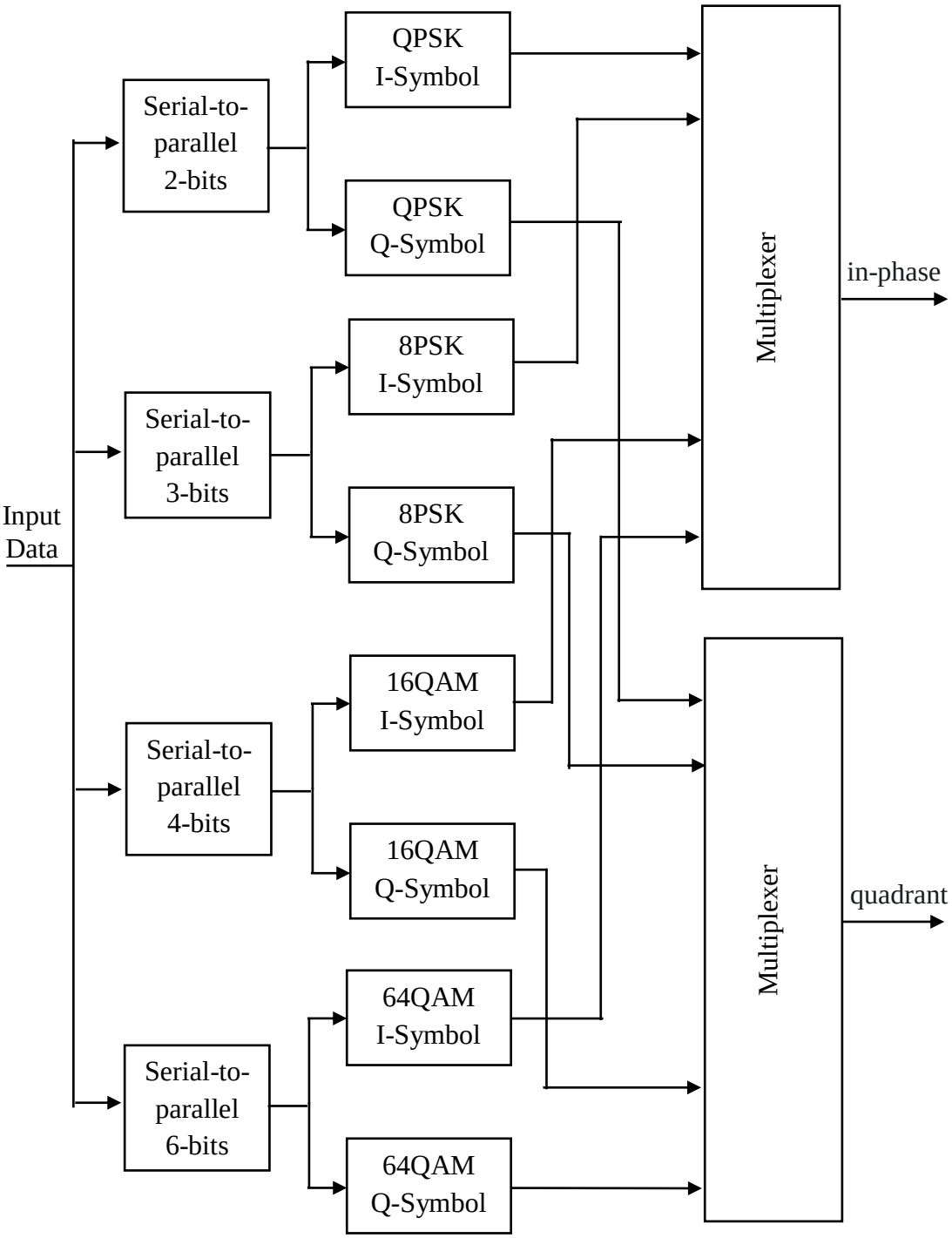

Figure 6.2: Variable Modulation Schemes Block Diagram

Table 6.1: Resource Utilisation and Performance of Modulator

Parameters	Used on Virtex 5
Number of Slice Registers	147
Number of Slice LUTs	89
Number of occupied Slices	79
Maximum frequency	484.262 MHz

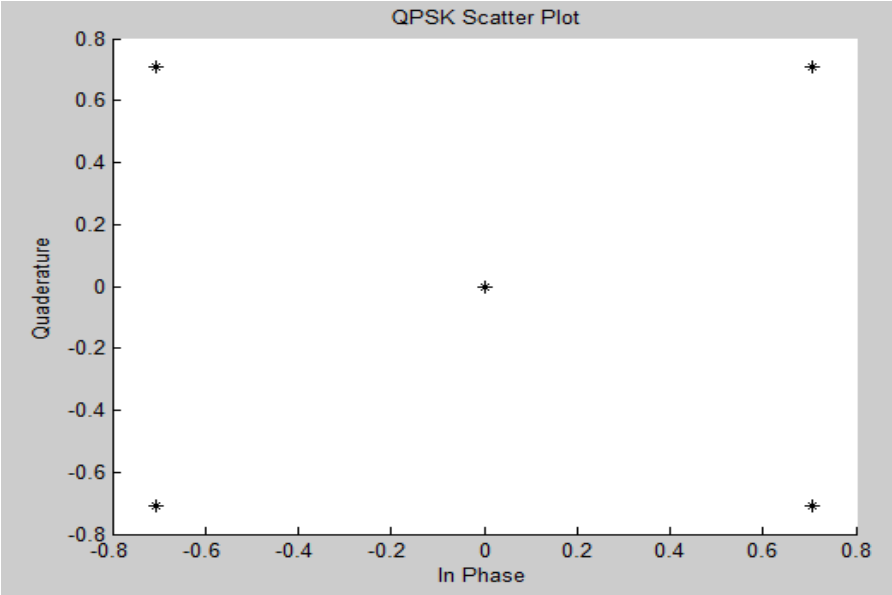

Figure 6.3: Fixed Point QPSK Constellation Diagram

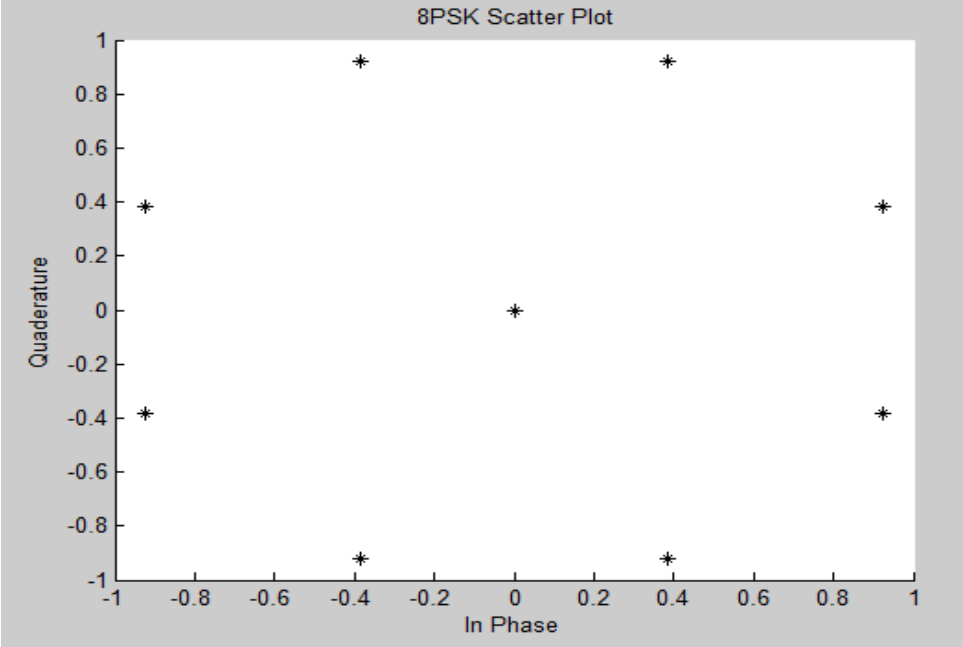

Figure 6.4: Fixed Point 8PSK Constellation Diagram

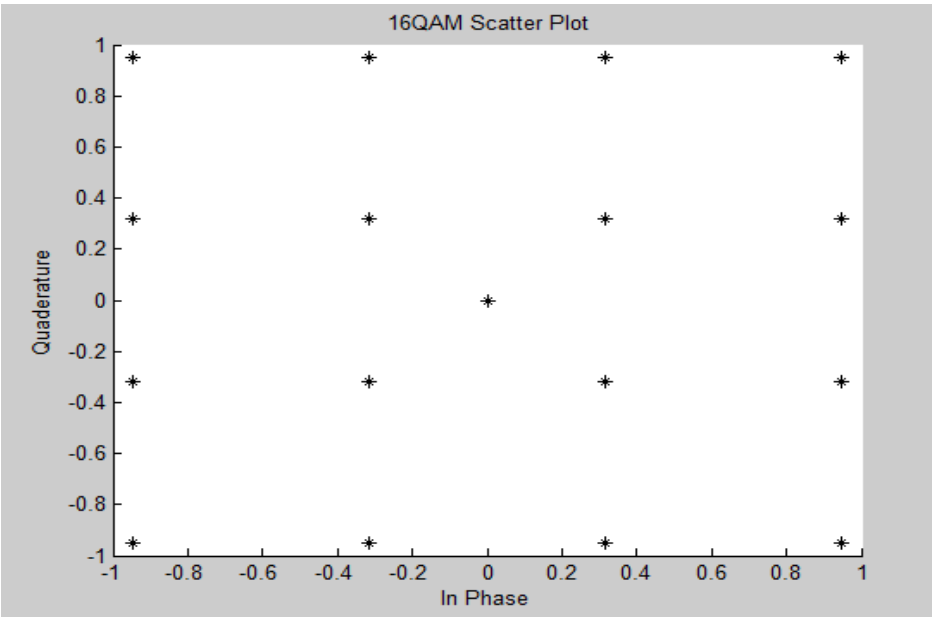

Figure 6.5: Fixed Point 16QAM Constellation Diagram

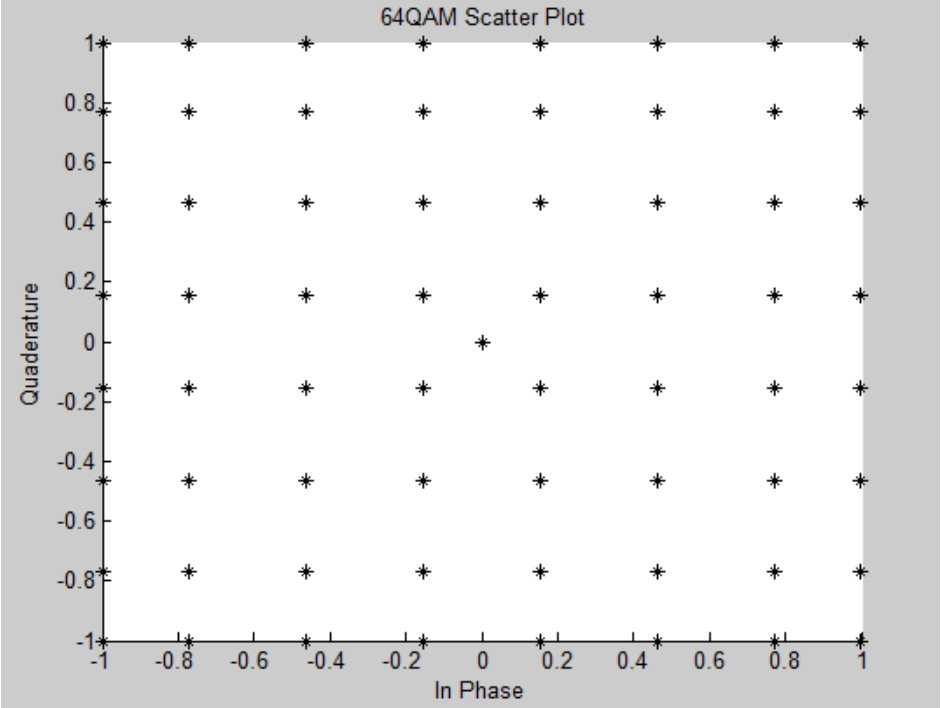

Figure 6.6: Fixed Point 64QAM Constellation Diagram

Table 6.2 : Mean Squared Error for Supported Modulation Schemes

Modulation Schemes	Mean Square Error
QPSK	5.95783329002397e-11
8PSK	1.05208187103746e-10
16QAM	6.26153516768136e-11
64QAM	0.00148347991492249

6.3 Reconfigurable OFDM Transmitter

The new generation of standards require variable length Inverse Fast Fourier Transforms, and the addition of variable length cyclic prefixes. The output produced by the modulator is presented to the variable length IFFT.

For this purpose, the Xilinx FFT core, version 7.1 is used, offering variable FFT size and variable cyclic prefix length. Figure 6.7 gives the block diagram of the reconfigurable OFDM transmitter. The IFFT has a dynamic range from 128 up to 2048 points, and this is set using the unsigned vector *IFFT_size*, with length $log_2(NFFT)$. Table 6.3 gives the values for setting the IFFT size. The addition of the cyclic prefix is set using the unsigned vector *Cyclic_Prefix_size*, also with length $log_2(NFFT)$.

The inverse FFT was synthesized with variable length ranging up to 2048, and with the input complex precision specified as sixteen bits, in the form <1, 15>. The resource utilisation of the design is summarised in Table 6.4. Matlab floating point flow graph of the OFDM transmitter is shown in Figure 6.8.

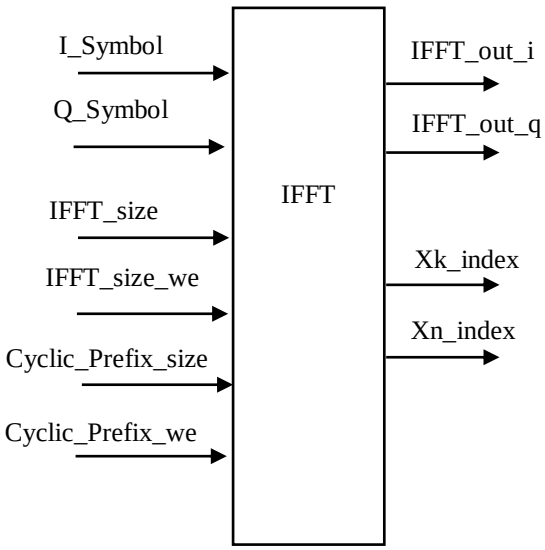

Figure 6.7: Block Diagram of variable IFFT

Table 6.3: *IFFT_size* Selection Table

IFFT_size setting	IFFT
11	2048
10	1024
9	512
8	256
7	128

Table 6.4: Resource Utilisation of Configurable IFFT

Parameters	Used
Number of Slice Registers	1,349
Number of Slice LUTs	1,117
Number of occupied Slices	529
Number of BlockRAM/FIFO	8
Number of DSP48Es	6
Maximum frequency	372.024 MHz

Table 6.5: Mean Squared Error of Reconfigurable OFDM Transmitter

IFFT	Mean Square Error			
	QPSK	8PSK	16 QAM	64 QAM
128	7.902e-08	7.269e-08	7.136e-08	1.142e-05
256	5.067e-08	5.580e-08	4.604e-08	6.287e-06
512	2.985e-08	2.926e-08	2.968e-08	2.914e-06
1024	1.72e-08	1.744e-08	1.862e-08	1.425e-06
2048	1.03e-08	9.507e-09	1.027e-08	7.00e-07

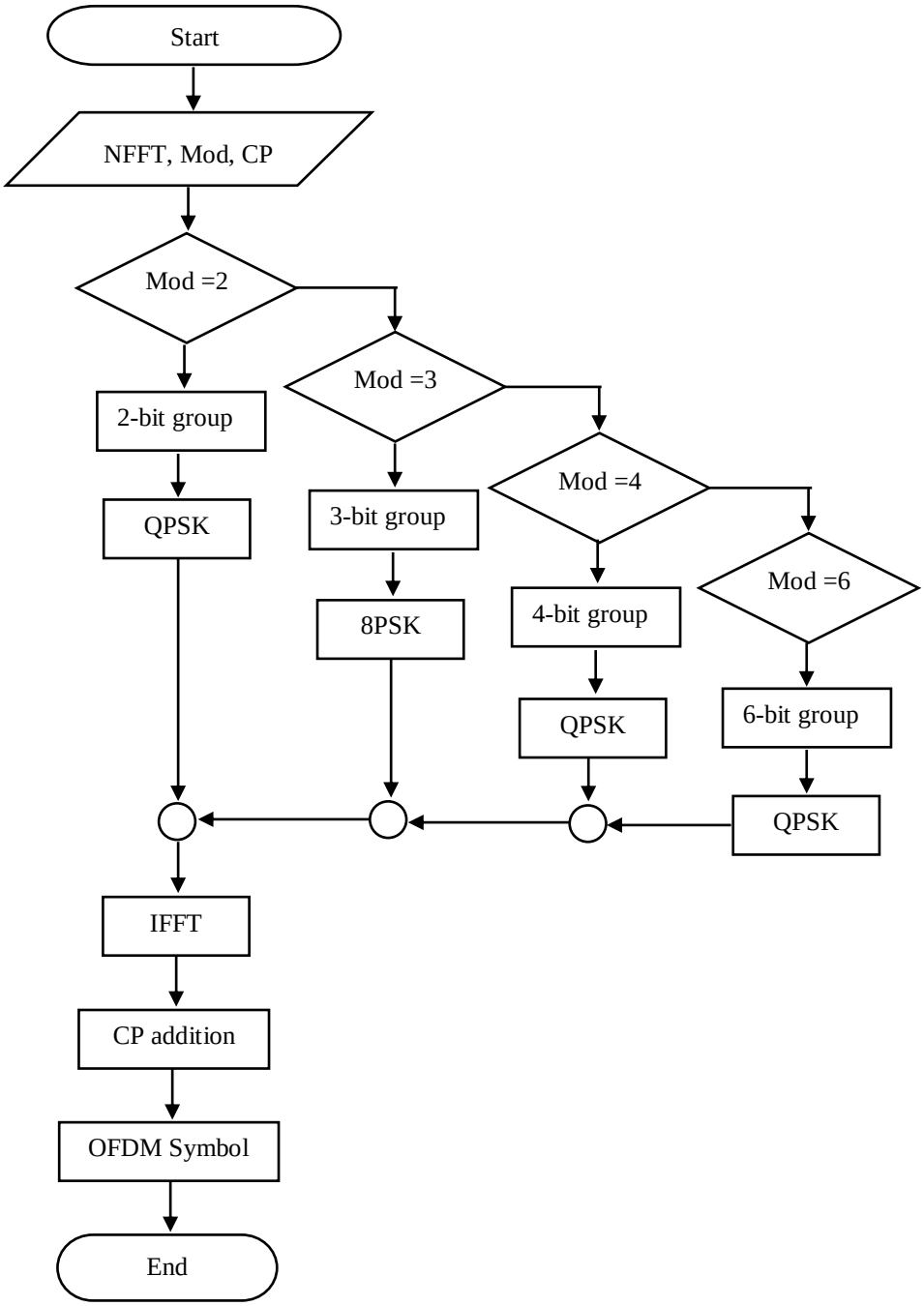

Figure 6.8: Matlab Script Transmitter Flow Chart

7 Conclusion and Discussion

7.1 Introduction

This research has been applied to the study, design and analysis of efficient sequential Fast Fourier Transform Architectures aimed at implementation on high speed FPGA devices. Many 3G and most 4G wireless networks now use the FFT as part of the physical layer of modulation and multiplexing. This research has focused on how to meet the requirements of the 4G FFT implementation for OFDM algorithms based on using FPGA hardware devices that will use minimum resource area but have a speed capability that they can implement all FFT lengths required in the various 4G implementations. A key effort in this research has been to find the best way of optimizing and/or minimising the resources area required on the FPGA, but maintaining performance. Noting that many designers using FPGAs for 4G will likely choose to use off-the-shelf FFT and IFFT components from FPGA vendors the thesis has aimed to show this may not always be best suited and clear note must be

taken of the actual 4G PHY layer standard, wordlength and resolution required, and also the actual resources available, i.e. is there memory blocks, or multipliers, etc available.

The research work has designed architectures working with fixed point and comparing against golden reference implementations which were specifically written in floating point models for the various design validation and verification.

In the course of the research two specific architectures have been used to calculate the FFTs. One depends on an efficient serialised butterfly using two multipliers and the second is based on CORDIC to perform the multiplies and various trigonometric calculations. A comparison between these architectures and the recent version of Xilinx FFT 7.1 core was been performed to highlight the advantages and disadvantages of the design and provide researchers with ideas as to which points can be modified.

The OFDM transmitter designed was based on the Xilinx System Generator software tool, and aimed to be reconfigurable with variable FFT size, cyclic prefix and modulation schemes and as such be full adaptable to any OFDM aspect of modern 4G PHY layer wireless standards.

7.2 Summary Contributions of the Research

The trade-off among the resources area, speed and accuracy of FPGA design is one of the important topics that have been studied in this research. To obtain better results, three architectures for FFTs have been introduced and researched. One used serial butterfly (section 4.3.1), the second used serial pipelined butterfly (section 4.3.2), while the third used CORDIC (Section 5.2). At the same time, an OFDM transmitter with reconfigurable property has been designed (Chapter 6). A summary of main contributions made by this research and design work are given below.

7.2.1 FFT Based Serial Butterfly

The best way to achieve minimum resource area in the FPGA design is to implement the required design serially – assuming the required processing speeds can be attained. Therefore to gain full advantage of the serial implementation, the butterfly and the entire architecture of the FFT have been implemented serially. Given that the design will slow down in the calculation speed but use less resource area, it is important for optimization purposes to think of the architecture serially. In the serial butterfly situation, not only is the butterfly implemented serially but the design uses only a pair of RAMs and ROMs. In the eight bits twiddle factor case study, the FFT based on serial butterfly utilized flip flops between 450 for 128 points FFT up to 625 flip flops in 2048 points FFT, as was shown in Table 4.1. The number by which the flip flops increase is only 175, which is very efficient compared to the off the shelf option from Xilinx, where the number of flip flops for 128 points is 959, and for 2048 points 1258 (Table 5.4), i.e., an increase of 299. In the serial butterfly, the look-up tables start from 652 for 128 points and rise to 992 for 2048 points, an increase of 340. In the Xilinx case the LUTs start at 712 for 128 points and end with 932 for 2048 points, i.e. a cost increase increments are 220. Here Xilinx maintained the increments better than serial butterfly, but this occurred because the serial butterfly used minimum block RAMs compared to all the other architectures in this research. Where Xilinx goes up from four in 128 points to eight in 2048 points, the serial butterfly keeps two for different FFT sizes. In the DSP48Es multipliers numbers, the serial butterfly shared two multipliers to calculate the FFT efficiently in terms of resource area. So it uses only two throughout the varying sizes of the FFT. In the Xilinx case, the multipliers start with 3 in 128 points and grow to six in 2048 points. In Slices terms, the serial butterfly starts with 276 for 128 points and ends with 444 for 2048, giving an increment of around 177 slices. In the Xilinx case, it starts with 347 for 128 points and goes up to 490 for 2048. This gives an increment of 143. It seems to maintain the number of slices, but is bigger in values.

In respect of speed, Xilinx has the best design as it is optimized for Virtex 5 boards where it starts from 392 MHz for 128 points, while serial butterfly is at about 197

MHz. It could be half; this is because of the serial nature of the design and because Xilinx is optimized for Xilinx boards. As the FFT size increases the speed goes down: for 2048 point Xilinx gives 292 MHz, while the serial butterfly gives 144 MHz. For latency, Xilinx is twice as good since it needs half the number of clocks to complete the calculations, taking 448 clock cycles for 128 points and going up to 11264 for 2048 points. In the serial FFT butterfly it is double these values.

In the serial butterfly architecture, the main advantage is that it minimizes the resources. The numbers of Flip Flops, Look-up Tables, Slices and Block RAMs are much lower than for the Xilinx Core generator solution. It uses only two multipliers to implement the Radix-2 butterfly. For the wireless 4G application case study, in section 5.7, Table 5.5 shows the maximum OFDM sampling rate that the serial FFT can perform, the networks and FFT size that can be used. The requirements of the standards for OFDM symbols are mentioned in chapter 1, Table 1.1, Table 1.2 and Table 1.3. The sampling rate is the speed that the FFT design is required to meet. For the serial butterfly the maximum speed at which the sampling OFDM symbol can be processed is given in Table 5.5. For IEEE802.20, UMB and IEEE 802.16e standards, the serial butterfly can meet the OFDM requirements up to 1024 FFT points. For LTE this is possible for up to 512 points.

7.2.2 FFT Based Serial Pipelined Butterfly

In the previous section, serial butterfly architecture has been discussed. One of the important parameters that the FFT for OFDM needs to meet is the sampling rate. This parameter depends on the speed that can be achieved by the design. The serial *pipelined* butterfly is designed to improve the speed of the design and keep the cost as low as possible. The key development achieved here is in the butterfly engine. In serial pipelined butterfly for eight bits twiddle factor precision, the speed of the design is 233 MHz for 128 point, compared to 192 MHz in serial butterfly: an improvement of around 41 MHz. For 256 points it is 30 MHz, for 512 points 11MHz, and for 1024 points 70 MHz. At this size, the improvement towards

meeting the requirements of all 4G wireless networks with FFT goes up to 1024, as shown in Table 5.6, where the LTE standard requirement is met at 1024 FFT points.

In the resource area analysis, the main difference between the serial pipelined butterfly architecture and serial butterfly architecture is the number of slices and block RAMs occupied by the designs. Here, the number of slices starts with 260 for 128 points for serial pipelined butterfly and goes up to 348 for 2048 points; while in serial butterfly, it starts from 276 at 128 points and goes up to 444 at 2048. But the main difference appears in the number of block RAMs used to achieve the architectures, with the serial using only two compared with four in the serial pipelined case.

With respect to the Xilinx study case, the flip flop numbers used by serial pipelined butterfly reach 646 at 2048, while in Xilinx they rise to 1258 at 2048, which is about double. For the look-up tables the range starts from 616 for 128 points and ends with 826 at 2048 points, while in Xilinx it ranges from 712 up to 932. The difference is around one hundred LUTs in each case. The number of slices occupied by serial pipelined butterfly is from 260 up to 348 while in Xilinx it starts from 347 and goes up to 490. In Xilinx the loss level (347) at 128 equals the serial pipelined higher level (348). For DSP48Es, the serial pipelined butterfly is designed to use only two multipliers, and this covers the whole range of FFTs under research, while in Xilinx it starts from three and goes up to six. The block RAM numbers give it another advantage over Xilinx, since it consistently utilizes four while Xilinx ranges from four to eight. For the maximum frequency that can be achieved on the FPGA board, The Xilinx design though does have advantages over the serial pipelined butterfly, the clocking range for Xilinx being 392, 495, 350, 273, 292 MHz, (for FFT orders of 11, 10, 9, 8, 7 respectively) while for serial pipelined it is 233, 221, 179, 217, 192 MHz. The latency of the design in Xilinx is better as well, since it needs half the number of clocks required by serial pipelined butterfly.

In the serial pipelined butterfly architecture, the main advances are that it increases the speed of operations and keeps the resource area small compared to serial butterfly. It uses only two multipliers and pipelines the input and output and each

operation inside the butterfly very well. It uses only two multipliers to implement the Radix-2 butterfly. It can operate in OFDM with sampling rate up to 33 M sample/second and meets the requirements of all 4G wireless networks with FFT lengths up to 1024.

7.2.3 FFT Based Full Parallel CORDIC

This part of the research was focused on FFT sequential architecture. To discover more about this architecture and its ability to meet the requirements of 4G wireless standards, the engine that performs the calculation has been changed with CORDIC used instead of butterfly. The changed requirements have been studied, analyzed and compared with FFT based on butterfly. The twiddle factor values in FFT based butterfly that were stored in a couple of ROMs (section 4.2.4) have changed to twiddle factor angles as in section 5.2.1. This has not extended the resource area on FPGA for the small FFT size, but the twiddle factor values are between (-1, 0.999) while the twiddle factor angles are between 0 and 360 degrees. This leads to more bits required to represent the values on the twiddle factor angles, from eight for twiddle factor values to sixteen on twiddle factor angles. Result as the angles are real while the values of twiddle factor are complex. The block RAMs used in the design are shown in Figure 5.16. At large numbers FFT 2048, the block RAMs number in the CORDIC case goes up to five as a reflection of previous effects. The CORDIC engine is implemented fully parallel, and uses eleven pipelined cells (to obtain enough iterative resolution) based on Figure 5.11. It was noted that increasing the number of cells beyond 11 did not improve the signal to noise ratio. The CORDIC required Mapper, Demapper pulse CORDIC cells. The resource area analysis shows that CORDIC needs more flip flops compared with the other design (Figure 5.12) and reach 1200 flip flops in 2048 point FFT. The number of look-up tables and slices are high as was shown in Figure 5.13 and Figure 5.14. However the very clear advantage of CORDIC FFT is that it needs no multipliers; since the CORDIC implements the multiplication by add shift process (Figure 5.15). The design can operate on 30 M sample/second and meet 4G wireless networks up to

1024 FFT points as in Table 5.7. Another point to note is that it gives a better signal to noise ratio compared to the above architectures given the architecture chosen does in fact operate with higher wordlength.

7.2.4 FFT Based CORDIC Calculated Angles

On this architecture one key advantage and useful attributes were demonstrated. It uses just one block ROM compared with the storage angles required by the FFT CORDIC design. The design also uses no multipliers (Figure 5.15). In overall cost the design uses more Flip flops, Look up tables and Slices that the other architectures, however a specific advantage is it can be use for variable size FFT more easily that other architectures.

7.2.5 Reconfigurable OFDM Transmitter

This OFDM transmitter architecture has the ability to control three important parameters in the OFDM system. The transmitter can change the modulation schemes to QPSK, 8PSK, QAM and 64 QAM. The values of constellation signals are stored in distributed RAMs instead of block RAMs, as their size is not large, making the former more efficient to use (Table 6.1). A couple of multiplexers are used to collect the real and imaginary values of the symbol generated by the modulators. The second and third levels of reconfigurable parameters are offered by Xilinx FFT 7.1. The FFT size and cyclic prefix can be set directly in the IP core. The mean square error test for the design compared with floating point for the OFDM transmitter demonstrated good performance, as in Table 6.2 and Table 6.5.

7.2.6 Floating Points Models Verification and Validation

Verification and Validation (V&V) form an essential step in the progress of successful research. Verification and validation were performed at all stages to prove that the sequential FFT models are programmed correctly. The procedure can be used to estimate many parameters in the design and in debugging the VHDL codes. The signal to noise ratio parameter is used to prove the accuracy of the FFT based on butterfly and CORDIC architectures. Figure 4.13 shows the FFT based butterfly signal to noise ratio test comparing the double precision floating point FFT with the FFT of Matlab built in function. The figure shows that the model is programmed correctly and has a high signal to noise ratio, around 715 dB on 128 point, decreasing to 700 dB as the FFT size goes up to 2048. The second aim of the V&V floating point model is to ensure that the VHDL model has been implemented properly. For this purpose a bit-accurate model is driven that converts the floating point values to two complement (signed integer values) for use in V&V.

In the sequential FFT based on CORDIC, the floating point model is used. The algorithm is programmed correctly as shown in **Figure 5.3**. The signal to noise ratio test shows 163 dB at 128 FFT point and 154 dB at 2048. This model is used to verify and validate the VHDL code that has been written correctly. To obtain a match with the signed integers shown in Modelsim or Isim software packages from Xilinx, the floating point model values are changed to tow's complement.

The verification process of the FFT models ensures that mistakes have not been made in implementing them by using the VHDL codes. Generating random values, creating VHDL test benches, and simulating the test benches including FFT models have been achieved successfully. The VHDL test bench is used to the random values generated by floating point models and fed into the FFT VHDL entity (unit under test); it then collects the FFTs output as in section 4.8 and section 5.3.

7.3 Conclusion

In the signal processing field, many algorithms play a substantial role in the development of wireless communication. Among these algorithms, this thesis has focused on the Fast Fourier Transform. The FFTs have been used as an efficient means of implementing multicarrier modulation and demodulation for 4G wireless networks. The OFDM is a multicarrier modulation used as part of the physical layers to obtain air interface between the base station and the access terminal.
Many design tools have been used to implement the signal processing algorithms and the FFTs. The programmable DSPs processors, ASIC and the FPGAs are the most common tools in this field. The FPGA makes it possible to combine the advantages of the DSP processor and ASIC. This project aims at efficient design of sequential FFTs based on FPGA for 4G wireless networks

There are many architectures for FFTs that can be implemented on FPGAs with differing features and requirements. Two things are important and need to be kept in balance: cost and efficiency. This research aimed to give meaningfull and real time implementable results with minimum cost. The most common FFT architectures are the sequential and pipelined. This research has used the sequential architecture to achieve minimum cost and meet the requirements of the 4G wireless networks on FPGA boards (section 5.7). For the sequential architecture, Radix-2 FFT decimation-in-frequency has been chosen for its simplicity of implementation on the FPGA. To obtain better compression and result, the FFT has been implemented based on butterfly and CORDIC.

This thesis focuses on the use of FPGAs in two applications. One is for FFT algorithms, which absorb the most interest. The second is for the OFDM system design using System Generator. Many parameters have been taken into consideration while working on the FPGA. Achieving high speed and minimizing the resource area were the most important. For speed, three factors are taken into account: throughput, latency and the critical path. The approach of adding pipeline registers has been used to reduce the critical path. This approach is powerful for increasing the design speed.

The two clocks technique has been used to reduce the latency and increase the throughput. The resource area is reduced by using pipelined sequential architecture.

References

[1] Xilinx, "Virtex-5 FPGA XtremeDSP Design Considerations User Guide Virtex-5 FPGA User Guide, v3.2,," September, 2008.

[2] B. R. S. Ahmad R. S. Bahai, *Multi-Carrier Digital Communications Theory and Applications of OFDM*. New York: Kluwer Academic Publishers, 2002.

[3] W. Y. Bolton, Xiao Guizani, M., "IEEE 802.20: mobile broadband wireless access," *Wireless Communications, IEEE,* vol. 14, pp. 84-95, 2007.

[4] A. K. Greenspan, M. Tomcik, J. Canchi, R. Wilson, J., "IEEE 802.20: Mobile Broadband Wireless Access for the Twenty-First Century," *Communications Magazine, IEEE,* vol. 46, pp. 56-63, 2008.

[5] M. Merlyn, "FPGA implementation Of FFT processor with OFDM transceiver," in *Signal and Image Processing (ICSIP), 2010 International Conference on*, 2010, pp. 483-489.

[6] J. M. L. Rudagi, R. Patil, P. Biraj, N. Nesaragi, N., "An Efficient 64-point Pipelined FFT Engine," in *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on*, 2010, pp. 204-208.

[7] Xilinx, "System Generator for DSP User Guide," 2010.

[8] Xilinx, "Synplify Pro™ User Guide and Tutorial," 2001.

[9] Xilinx, "ModelSim XE User's Manual," 2000.

[10] S. A. Ltd, "Signal Processing and Communication Notes Book," ed: Steepest Ascent Ltd

[11] FAROOQ KHAN, *LTE for 4G Mobile Broadband Air Interface Technologies and Performance*. Cambridge: Cambridge University Press, 2009.

[12] Vijay K. Garg, *WIRELESS COMMUNICATIONS AND NETWORKING*. Boston: Elsevier Inc., 2007.

[13] D. J. W. Andrew S. Tanenbaum, *Computer Networks,* 5th Edition ed. Boston: Prentice Hall, 2011.

[14] A. Jalili, *et al.*, "Performance Evaluation of IEEE 802.20 PHY Layer," presented at the 2009 International Conference on Computer Engineering and Technology, Vol Ii, Proceedings, Los Alamitos, 2009.

[15] M. Wang, "Ultra Mobile Broadband Technology Overview," in *Communication Networks and Services Research Conference, 2008. CNSR 2008. 6th Annual*, 2008, pp. 8-9.

[16] 3GPP2, "Physical Layer for Ultra Mobile Broadband (UMB) Air Interface Specification," August 2007.

[17] "IEEE Standard for Local and Metropolitan Area Networks Part 20: Air Interface for Mobile Broadband Wireless Access Systems Supporting Vehicular Mobilityphysical and Media Access Control Layer Specification," *IEEE Std 802.20-2008,* pp. 1-1039, 2008.

[18] W. forum. WiMAX's technology for LOS and NLOS environments.

[19] J. M. Gazi Ahmed, Imran Hasan "Performance Evaluation of IEEE 802.16e (Mobile WiMAX) in OFDM Physical Layer," Master, Department of Telecommunication, Blekinge Institute of Technology, August 2009.

[20] F. Wang, *et al.*, "IEEE 802.16e system performance: analysis and simulations," in *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on*, 2005, pp. 900-904 Vol. 2.

[21] A. Ghosh, *et al.*, "LTE-advanced: next-generation wireless broadband technology [Invited Paper]," *Wireless Communications, IEEE,* vol. 17, pp. 10-22, 2010.

[22] J. Berkmann, *et al.*, "On 3G LTE Terminal Implementation - Standard, Algorithms, Complexities and Challenges," in *Wireless Communications and Mobile Computing Conference, 2008. IWCMC '08. International*, 2008, pp. 970-975.

[23] Carl R. Nassar, *et al.*, *Multi-Carrier Technologies For Wireless Communication*. New York: Kluwwe Academic Publishers, 2002.

[24] C. Xiaoxin and Y. Dunshan, "Digital OFDM transmitter architecture and FPGA design," in *ASIC, 2009. ASICON '09. IEEE 8th International Conference on*, 2009, pp. 477-480.

[25] S. Weinstein and P. Ebert, "Data Transmission by Frequency-Division Multiplexing Using the Discrete Fourier Transform," *Communication Technology, IEEE Transactions on,* vol. 19, pp. 628-634, 1971.

[26] J. Haining, *et al.*, "Design of an efficient FFT Processor for OFDM systems," *Consumer Electronics, IEEE Transactions on,* vol. 51, pp. 1099-1103, 2005.

[27] Q. Wang, *et al.*, "Efficient Implementation of Synchronization in OFDM System Based on FPGA," in *Advanced Communication Technology, The 9th International Conference on*, 2007, pp. 178-181.

[28] W. Y. Zou and W. Yiyan, "COFDM: an overview," *Broadcasting, IEEE Transactions on,* vol. 41, pp. 1-8, 1995.

[29] K. E. Nolan, "Reconfigrabale OFDM Systems," Doctor of Philosophy, Electronic and Electrical Engineering, University of Dublin, Dublin, 2005.

[30] J. Q. Garcia, *et al.*, "On the design of an FPGA-based OFDM modulator for IEEE 802.11a," in *2005 2nd International Conference on Electrical & Electronics Engineering*, New York, 2005, pp. 114-117.

[31] Zhengdao, *et al.*, "OFDM or single-carrier block transmissions," *Communications, IEEE Transactions on,* vol. 52, pp. 380-394, 2004.

[32] K. Harikrishna, *et al.*, "An FFT Approach for Efficient OFDM Communication Systems," in *Signal Acquisition and Processing, 2010. ICSAP '10. International Conference on*, 2010, pp. 117-119.

[33] C. Dick and F. Harris, "FPGA implementation of an OFDM PHY," in *Signals, Systems and Computers, 2003. Conference Record of the Thirty-Seventh Asilomar Conference on*, 2003, pp. 905-909 Vol.1.

[34] S. Abbas, *et al.*, "Implementation of OFDM Baseband Trasmitter Compliant IEEE Std 802.16d on FPGA," in *Communication Software and Networks, 2010. ICCSN '10. Second International Conference on,* 2010, pp. 22-26.

[35] C. Ebeling, *et al.*, "Implementing an OFDM receiver on the RaPiD reconfigurable architecture," *Computers, IEEE Transactions on,* vol. 53, pp. 1436-1448, 2004.

[36] Z. H. Derafshi, *et al.*, "A High Speed FPGA Implementation of a 1024-Point Complex FFT Processor," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, 2010, pp. 312-315.
[37] M. Serra, *et al.*, "IFFT-FFT core architecture with an identical stage structure for wireless LAN communications," in *Signal Processing Advances in Wireless Communications, 2004 IEEE 5th Workshop on*, 2004, pp. 606-610.
[38] X. S. Xiangyang Liu1, and Yuke Wang1, "Performance Evaluation on FFT Software Implementation," in *Proceedings of the International MultiConference of Engineers and Computer Scientists*, Hong Kong, March 18 - 20, 2009.
[39] Q. Wu and H. Chen, "The Design of a Large Point Reconfigured FFT Based on FPGA," in *Intelligent Information Technology and Security Informatics, 2009. IITSI '09. Second International Symposium on*, 2009, pp. 64-67.
[40] S. Bouguezel, *et al.*, "An improved radix-16 FFT algorithm," in *Electrical and Computer Engineering, 2004. Canadian Conference on*, 2004, pp. 1089-1092 Vol.2.
[41] K. M. Nadehara, T. Kuroda, I, "Radix-4 FFT implementation using SIMD multimedia instructions," presented at the Acoustics, Speech, and Signal Processing, 1999. ICASSP '99. Proceedings., 1999 IEEE International Conference, 1999.
[42] A. M. Despain, "Very Fast Fourier Transform Algorithms Hardware for Implementation," *Computers, IEEE Transactions on*, vol. C-28, pp. 333-341, 1979.
[43] John G. Proakis and Dimitris G. Manolakis, *Digital Signal Processing Principles, Algorithms, and Applications*. USA: Prentice Hall, Inc., 1996.
[44] L. Hsin-Lei, *et al.*, "A novel pipelined fast Fourier transform architecture for double rate OFDM systems," in *Signal Processing Systems, 2004. SIPS 2004. IEEE Workshop on*, 2004, pp. 7-11.
[45] W. Bingrui, *et al.*, "Design of Pipelined FFT Processor Based on FPGA," in *Computer Modeling and Simulation, 2010. ICCMS '10. Second International Conference on*, 2010, pp. 432-435.
[46] R. T. BONE, "FPGA DESIGN OF A HARDWARE EFFICIENT PIPELINED FFT PROCESSOR," SCHOOL OF GRADUATE STUDIES, Wright State University, 2005.
[47] L. Fangming and P. Xiaozhong, "Research on implementation of FFT based on FPGA," in *Computer Application and System Modeling (ICCASM), 2010 International Conference on*, 2010, pp. V7-152-V7-155.
[48] X.-f. Wang and Z.-l. Hou, "Design and implement of FFT processor for OFDMA system using FPGA," in *Mechanical and Electronics Engineering (ICMEE), 2010 2nd International Conference on*, 2010, pp. V2-297-V2-299.
[49] Z. Kai, *et al.*, "An ultra high-speed FFT processor," in *Signals, Circuits and Systems, 2003. SCS 2003. International Symposium on*, 2003, pp. 37-40 vol.1.
[50] K. S. Hemmert and K. D. Underwood, "An analysis of the double-precision floating-point FFT on FPGAs," in *Field-Programmable Custom Computing Machines, 2005. FCCM 2005. 13th Annual IEEE Symposium on*, 2005, pp. 171-180.

[51] L. Sheng, *et al.*, "A broadband FFT processor core based on FPGA," in *Antennas, Propagation and EM Theory, 2008. ISAPE 2008. 8th International Symposium on*, 2008, pp. 1546-1549.
[52] S. Zhijian, *et al.*, "The Design of Radix-4 FFT by FPGA," in *Intelligent Information Technology Application Workshops, 2008. IITAW '08. International Symposium on*, 2008, pp. 765-768.
[53] M. Petrov and M. Glesner, "Optimal FFT architecture selection for OFDM receivers on FPGA," in *Field-Programmable Technology, 2005. Proceedings. 2005 IEEE International Conference on*, 2005, pp. 313-314.
[54] Z. Qihui and M. Nan, "A Low Area Pipelined FFT Processor for OFDM-Based Systems," in *Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on*, 2009, pp. 1-4.
[55] K. Jen-Chih, *et al.*, "Implementation of a programmable 64∼2048-point FFT/IFFT processor for OFDM-based communication systems," in *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, 2003, pp. II-121-II-124 vol.2.
[56] H. Shousheng and M. Torkelson, "A new approach to pipeline FFT processor," in *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, 1996, pp. 766-770.
[57] Y. M. W. Li, and L. Wanhammar, "Word Length Estimation for Memory Efficient Pipeline FFT/IFFT Processors," presented at the the Int. Conf. on Signal Processing Applications & Technology (ICSPAT), Nov., 1999.
[58] A. M. Sule, "Design of Pipeline Fast Fourier Transform Processors using 3 Dimensional Integrated Circuit Technology," Computer Engineering, North Carolina State University, Raleigh, North Carolina, 2007.
[59] J. E. Volder, "The CORDIC Trigonometric Computing Technique," *Electronic Computers, IRE Transactions on*, vol. EC-8, pp. 330-334, 1959.
[60] J. Yang, *et al.*, "A New Design and Implementation of the Butterfly Unit on FPGA," in *Image and Signal Processing, 2009. CISP '09. 2nd International Congress on*, 2009, pp. 1-6.
[61] X. Xin, *et al.*, "Reduced memory architecture for CORDIC-based FFT," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, 2010, pp. 2690-2693.
[62] Y. Cheng-Ying, *et al.*, "Efficient CORDIC Designs for Multi-Mode OFDM FFT," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, 2006, pp. III-III.
[63] P. K. Meher, *et al.*, "50 Years of CORDIC: Algorithms, Architectures, and Applications," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 56, pp. 1893-1907, 2009.
[64] J. S. Walther, "A Unified Algorithm for Elementary Functions," presented at the Spring Joint computer conf., 1971.
[65] Scott Hauck and Andr´e DeHon, *RECONFIGURABLE COMPUTING THE THEORY AND PRACTICE OF FPGA-BASED COMPUTATION*. BOSTON: Elsevier Inc, 2008.

[66] T. Adiono and R. S. Purba, "Scalable pipelined CORDIC architecture design and implementation in FPGA," in *Electrical Engineering and Informatics, 2009. ICEEI '09. International Conference on*, 2009, pp. 646-649.
[67] L. Huan and X. Yan, "Modified CORDIC Algorithm and Its Implementation Based on FPGA," in *Intelligent Networks and Intelligent Systems (ICINIS), 2010 3rd International Conference on*, 2010, pp. 618-621.
[68] O. A. Alim, *et al.*, "FPGA implementation for an optimized CORDIC module for OFDM system," in *Computer Engineering & Systems, 2008. ICCES 2008. International Conference on*, 2008, pp. 21-26.
[69] D. Timmermann, *et al.*, "A programmable CORDIC chip for digital signal processing applications," *Solid-State Circuits, IEEE Journal of*, vol. 26, pp. 1317-1321, 1991.
[70] B. Lakshmi and A. S. Dhar, "High speed architectural implementation of CORDIC algorithm," in *TENCON 2008 - 2008 IEEE Region 10 Conference*, 2008, pp. 1-5.
[71] M. Kuhlmann and K. K. Parhi, "A novel CORDIC rotation method for generalized coordinate systems," in *Signals, Systems, and Computers, 1999. Conference Record of the Thirty-Third Asilomar Conference on*, 1999, pp. 1361-1367 vol.2.
[72] J. Valls, *et al.*, "Efficient mapping of CORDIC algorithms on FPGA," in *Signal Processing Systems, 2000. SiPS 2000. 2000 IEEE Workshop on*, 2000, pp. 336-345.
[73] Y. H. Hu, "The quantization effects of the CORDIC algorithm," *Signal Processing, IEEE Transactions on*, vol. 40, pp. 834-844, 1992.
[74] A. M. Despain, "Fourier Transform Computers Using CORDIC Iterations," *Computers, IEEE Transactions on*, vol. C-23, pp. 993-1001, 1974.
[75] Sang Yoon Park, *et al.*, "DESIGN OF 2W4WSK-POINT FFT PROCESSOR BASED ON CORDIC ALGORITHM IN OFDM RECEIVER," presented at the PACRIM. 2001 IEEE Pacific Rim Conference Pacific, 2001.
[76] eASIC. Available: http://www.easic.com/
[77] R. M. Jiang, "An Area-Efficient FFT Architecture for OFDM Digital Video Broadcasting," *Consumer Electronics, IEEE Transactions on*, vol. 53, pp. 1322-1326, 2007.
[78] Roger Woods, *et al.*, *FPGA-based Implementation of Signal Processing Systems*. United Kingdom: John Wiley & Sons, Ltd, 2008.
[79] E. Lai, *Practical Digital Signal Processing for Engineers and Technicians*. London: all Newnes publications, 2003.
[80] Steve Kilts, *Advanced FPGA Design Architecture, Implementation, and Optimization*. Hoboken, New Jersey: John Wiley & Sons, Inc.,, 2007.
[81] T. Sansaloni, *et al.*, "Area-efficient FPGA-based FFT processor," *Electronics Letters*, vol. 39, pp. 1369-1370, 2003.
[82] Christos Meletis, *et al.*, "High-Speed Pipeline Implementation of Radix-2 DIF Algorithm," in *World Academy of Science, Engineering and Technology 2 2005*, 2005.
[83] Xilinx, "ISE Design Suite Software Manuals and Help - PDF Collection," 2010.

[84] G. R. J. JEAN-PIERRE, GUSTAVO D. SUTTER, *SYNTHESIS OF ARITHMETIC CIRCUITS*. New Jersey.: John Wiley & Sons, Inc, 2006.
[85] P. P.Chu, *FPGA Prototyping By VHDL Examples*. New Jersey: John Wiley & Sons, 2008.
[86] Xilinx. Available: http://www.xilinx.com/
[87] Bradley F. Dutton and C. E. Stroud, "Built-In Self-Test of Configurable Logic Blocks in Virtex-5 FPGAs," presented at the Proc. IEEE Southeastern Symp. on System Theory, 2009.
[88] H. Sarmento, "Virtex platforms comparison," 2008.
[89] J. Viejo, *et al.*, "Implementation of a FFT/IFFT module on FPGA: Comparison of methodologies," presented at the 2008 4th Southern Conference on Programmable Logic, Proceedings, New York, 2008.
[90] T. Y. Sung, "Memory-efficient and high-speed split-radix FFT/IFFT processor based on pipelined CORDIC rotations," presented at the Iee Proceedings-Vision Image and Signal Processing, 2006.
[91] Walter Fischer, *Digital Video and Audio Broadcasting Technology*. Heidelberg: Springer-Verlag Berlin Heidelberg, 2008.
[92] S. S. Abdullah, *et al.*, "A high throughput FFT processor with no multipliers," in *Computer Design, 2009. ICCD 2009. IEEE International Conference on*, 2009, pp. 485-490.
[93] M. J. Canet, *et al.*, "FPGA implementation of an if transceiver for OFDM-based WLAN," presented at the 2004 Ieee Workshop on Signal Processing Systems Design and Implementation, Proceedings, New York, 2004.
[94] J. Garcia and R. Cumplido, "On the design of an FPGA-based OFDM modulator for IEEE 802.16-2004," in *2005 International Conference on Reconfigurable Computing and FPGAs*, Los Alamitos, 2005, pp. 153-156.
[95] J. Xu, *et al.*, "Implementation of MB-OFDM Transmitter Baseband Based on FPGA," in *Circuits and Systems for Communications, 2008. ICCSC 2008. 4th IEEE International Conference on*, 2008, pp. 50-54.
[96] A. Ren, *et al.*, "FPGA implementation of an OFDM modem," in *Wireless Mobile and Computing (CCWMC 2009), IET International Communication Conference on*, 2009, pp. 761-764.
[97] F. Manavi and Y. R. Shayan, "Implementation of OFDM modem for the physical layer of IEEE 802.11a standard based on Xilinx Virtex-II FPGA," in *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, 2004, pp. 1768-1772 Vol.3.