Circuit and Systems
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ens.ewi.tudelft.nl/

2008

MSc THESIS

ASIC FFT Processor for MB-OFDM UWB System

Nuo Li

Abstract

The physical layer (PHY) standard of Multi-band Orthogonal Frequency Division Multiplexing (MB-OFDM) Ultra Wideband (UWB) system was defined by ECMA International. In this standard, the data sampling rate from the analog-to-digital converter to the physical layer is up to 528 Msample/s. Therefore, it is a challenge to realize the physical layer of the UWB system-especially the components with high computational complexity in Very Large Scale Integration (VLSI) implementation. Fast Fourier Transform (FFT) block is one of these components. FFT plays an important role in Multi-band OFDM UWB system, which is the demodulation block of OFDM signals.
The purpose of this project is to design an Application Specific Integrated Circuit (ASIC) FFT solution for this system. The specification is defined from the system analysis and literature research. All the design choices and considerations are concluded and explained. Based on the algorithm and architecture analysis, a novel Radix2^2Parallel processor is proposed, which is a small-area and low-power-consumption solution for MB-OFDM UWB system. Both Field Programmable Gate Array (FPGA) and ASIC targeted synthesis results of this architecture are presented.

CAS-MS-2008-05

Faculty of Electrical Engineering, Mathematics and Computer Science

ASIC FFT Processor for MB-OFDM UWB System

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

MICROELECTRONICS

by

Nuo Li
born in Ningxia, China

Circuit and Systems
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

ASIC FFT Processor for MB-OFDM UWB System

by Nuo Li

Abstract

The physical layer (PHY) standard of Multiband Orthogonal Frequency Division Multiplexing (MB-OFDM) Ultra Wideband (UWB) system was defined by ECMA International. In this standard, the data sampling rate from the analog-to-digital converter to the physical layer is up to 528 Msample/s. Therefore, it is a challenge to realize the physical layer of the UWB system-especially the components with high computational complexity in Very-Large-Scale Integration (VLSI) implementation. Fast Fourier Transform (FFT) block is one of these components. FFT plays an important role in Multiband OFDM UWB system, which is the demodulation block of OFDM signals.

The purpose of this project is to design an Application Specific Integrated Circuit (ASIC) FFT solution for this system. The specification is defined from the system analysis and literature research. All the design choices and considerations are concluded and explained. Based on the algorithm and architecture analysis, a novel Radix2^2Parallel processor is proposed, which is a small-area and low-power-consumption solution for MB-OFDM UWB system. Both Field Programmable Gate Array (FPGA) and ASIC targeted synthesis results of this processor are presented.

Laboratory	:	Circuit and Systems
Codenumber	:	CAS-MS-2008-05

Committee Members :

Advisor:	Nick van der Meijs, CAS, TU Delft
Chairperson:	Edoardo Charbon, CAS, TU Delft
Member:	René van Leuken, CAS, TU Delft
Member:	Arjan van Genderen, CE, TU Delft

To my grandmother in heaven
To my grandparents, parents and brother for their endless love and support...

Anything is possible. You can be told that you have a 90-percent chance or a 50-percent chance or a 1-percent chance, but you have to believe, and you have to fight.

Pain is temporary. It may last a minute, or an hour, or a day, or a year, but eventually it will subside and something else will take its place. If I quit, however, it lasts forever.

Lance Armstrong quote

Contents

List of Figures

List of Tables

Acknowledgments

During the past one year, many people gave me help and support. Without their help, the work described in this thesis could not have been accomplished.

I gratefully thank all members in Circuit and System Group. I learned a lot from them. I feel very honored to have done my master thesis in this group.

First and foremost, I would like to show my appreciation to Nick van der Meijs, the kindest supervisor I have ever known. From the beginning, he tried hard to help me find a project that was consistent with my interest. Without him, this project would not exist. He has been fully supporting this project work and using all the resources he could use to help me solve problems. He has been constantly guiding and teaching me to use the scientific methods and follow the right direction. Specially thank his encouragement, comforting, and serious attitude to the science. Meanwhile he also helped me to improve the writing and presentation skills. In total, I am deeply indebted to him. Thank you, Nick.

I would like to thank René van Leuken for teaching me the ASIC circuit design flow and helping me solve a lot of difficulties during the design process. What's more, he introduced a lot of resources to me which helped me a lot.

I am deeply indebted to Huib Lincklaen Arriëns. His kind and patient help really touches me. Every time when I asked for help, he always tried his best to solve the problems. Without him, I would take much more time in the darkness. It is his valuable suggestions that helped me conquer many barriers.

I especially thank Alexander de Graaf for helping me solve the synthesis problems and teaching me the Design Compiler tool usage.

I would like to thank Patrick Dewilde and Geert Leus for teaching me FFT and OFDM principles. And especially thank Geert Leus for helping me understanding the UWB principle.

I would like to thank Alle-Jan van der Veen for helping me the OFDM problems. And what' more, thank his diligent working attitude and words "every project has something to be improved" for inspiring me doing my project.

I would like to thank Antoon Frehe for helping me fixing the computer and software problems and providing the very nice lab environment.

I would like to thank Secretary Laura Bruns for constantly support and help.

I would like to thank Helena Sarmento and Nuno Rodrigues from Instituto Superior Ténico for answering me project related questions and telling me which parts should be focused on.

I especially thank Ramesh Chidambaram for discussing with me about the project problems by email and his very valuable suggestions.

I would like to thank all my colleagues in CAS lab. I especially thank Alejandro for the valuable help of debugging VHDL codes and the discussion of project problems. Thank Robert's help on the FGPA tool usage. I would like to thank Shiva and Benjamin Frey for the help of Latex. Special thanks to Snehal, Claudio , Fransesco, and Jose for their warm company.

I especially thank Feifei and Xiong for revising my thesis report. I would like to thank all my friends for the happy times. I have been very enjoying the past two years in Delft. Thank you all!

The last but not the least, I would like to thank something called interest for accompanying me to conquer the difficulties and letting me not give up!

Nuo Li
Delft, The Netherlands
October 12, 2008

1 Introduction

1.1 Motivation

Ultra-Wideband (UWB) Technology brings the convenience and mobility of wireless communications to high-speed interconnects in devices through out the digital home and office [11]. Instead of wired connection, this technology enables wireless connection for transmitting video, audio, and other data with high data speed and low power consumption. In February 2002, the Federal Communications Commission (FCC) in USA issued the ruling that Ultra-Wideband (UWB) could be used for data communication. Since then, UWB has became a hot research topic and plenty of research results have been published. Multiband-OFDM standard is one solution for UWB technology. A proposal for Multi-band OFDM UWB standard is published by IEEE 802.15 3a study group [12]. After IEEE 802.15 3a was withdrawn in the Spring of 2006, Multiband-OFDM has been controlled by ECMA International. In December 2007, the second revised version Standard ECMA-368: High Rate Ultra Wideband PHY and MAC Standard' was released, which specified physical layer (PHY) and medium access control layer (MAC) of the UWB technology based on Multiband-OFDM [5].

However, some key issues need to be solved for designing CMOS based Multiband-OFDM UWB solution in support of the low power requirement. One of the issues focuses on its FFT (Fast Fourier Transform) block, which takes 25% design complexity of the total digital baseband transceiver [1]. Although many results have already been published in this research area in the past few years [13][14][15][9][4], some key problems still exist and need to be improved for the speed, area and power consumption consideration. Based on ECMA-368, for the requirement of Multiband-OFDM system, this FFT processor should work on a few hundred MHz, which makes it difficult to implement. And since this system targets for the wireless portable devices, small area and low power consumption are also imperative. Therefore, this thesis focuses on the area and power consumption improvement under the ECMA-368 standard requirements.

1.2 The related work

The FFT processor design can be traced back long time ago. The basic FFT algorithms and architectures can be found from book [16] [6]. The "Split-Radix" algorithm was implemented by Duhamel [17]. The Radix 2^2 algorithm was provided in [8], which was a low computation solutions for FFT processor. Parallel-Pipeline FFT processors for VLSI implementations were analyzed by Wold, et al [18]. Lin, et al proposed four-parallel 1-GS/s FFT processor for UWB applications [15]. Chidambaram, et al presented an application specific instruction-set processor (ASIP) for OFDM-UWB [9]. A Field-Programmable Gate Array (FPGA) FFT module was provided for Multiband-OFDM receiver in 2007 [4]. In April, 2008, a two-parallel Radix 2^4 FFT processor was proposed [10].

1.3 Project goal

This project aims at designing ASIC (Application Specific Integrated Circuit) FFT processor for Multiband-OFDM UWB system. In order to achieve this goal, several steps need to be followed.

The first step is to find the specifications for this FFT processor, which are determined by the Multiband OFDM UWB standard. The step requires the analysis on OFDM and UWB technology and the constraints of its FFT processor.

After defining the specifications, optimized FFT algorithm and architecture should be used for these specifications. There are a large number of FFT algorithms and architectures in the signal processing literature [7]. Therefore, the state of art algorithms and architectures should be analyzed and compared. Based on different algorithms and architectures, different power consumptions, area and speed of the processor will be achieved. So their ASIC suitability should be analyzed and the effort should be focused on the choosing algorithms and architectures and optimization. Furthermore, the improvement space should be analyzed and the architecture should be further optimized.

The proposed algorithm and architecture should be validated by Matlab simulation before implementation. After that, this circuit needs to be implemented with VHDL. The synthesis step is followed by using both Synplify Pro targeted for FPGA and Design Compiler for ASIC. The synthesis results will be compared with other published FFT processor results.

1.4 Synopsis

Chapter 2 begins with an introduction to the UWB technology and OFDM signal, with a focus on FFT. Multiband-OFDM UWB is introduced and its OFDM and FFT related standard defined by [5] is strengthened. The remainder of this chapter focuses on the FFT requirements of Multiband-OFDM UWB.

Chapter 3 introduces the FFT concept and algorithms. Several conventional FFT algorithms are analyzed and new Radix 2^2 and 2^x algorithms are also discussed, which are suitable for ASIC FFT implementation. At the end, these algorithms are compared for utilization of ASIC implementation.

Chapter 4 analyzes and compares different FFT processor architectures. High level speed and area of these structures are presented. The imperative design choices and considerations are also discussed and analyzed here. Data format and scaling , twiddle factors, and the interface handling are analyzed and chosen with regard to the system requirements.

Chapter 5 presents the improvement based on the R2^2SDF structure. A novel parallel-pipeline structure called Radix2^2Parallel is introduced, which is a small-area and low-power-consumption solution for Multiband OFDM UWB systems. The remainder of this chapter describes design details and the VHDL implementation of this structure.

Chapter 6 describes the verification of this structure and test bench method. Both FPGA and ASIC targeted synthesis results are presented. At the end, the comparison of this structure with other published results is provided.

Chapter 7 summarizes the work achieved and suggests the future work areas.

The Principle of MB-OFDM UWB System and Its Requirements for FFT Processor 2

2.1 UWB system

Ultra-wideband (UWB) is a radio technology that can be used at very low energy levels for short-range high-bandwidth communication by using a large portion of the radio spectrum [19]. The Federal Communications Commission (FCC) in the USA has defined UWB signal in terms of the band width of emitted signal from an antenna which should exceeds 500 MHz or 20% of the center frequency. The FCC has also allocated 7.5 GHz frequency spectrum for unlicensed use of UWB devices in the 3.1 to 10.6 GHz frequency band, as shown in Figure 2.1.

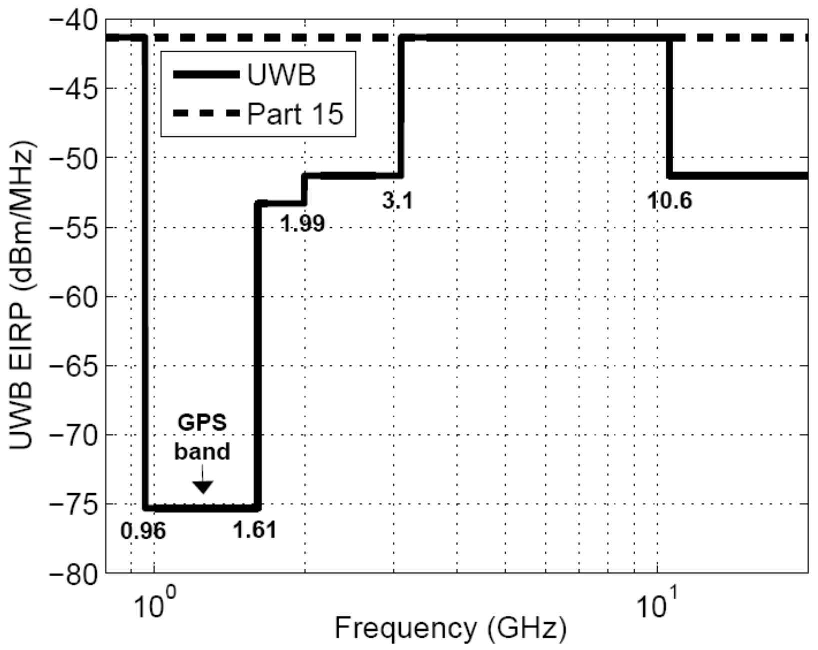


Figure 2.1: FCC spectrum mask for UWB devices indoor transmission, from [1]

As shown in the figure, for UWB operating in the 3.1-10.6 GHz band, the power spectral density emission limit is - 41.3dBm/MHz, which is significantly lower than other bands of the spectrum.

2.2 OFDM principle

There are many books and articles which describe how the orthogonal frequency-division multiplexing (OFDM) system works, such as [20] [3]. Here the FFT and IFFT modulation of OFDM is focused on, which is closely related to our FFT processor design.

The concept of orthogonal frequency-division multiplexing (OFDM) dates back to the mid 60's, when Chang provided the principle of transmitting messages simultaneously through a linear band-limited channel without interchannel interference (ICI) and intersymbol interference (ISI) [20]. Saltzberg performed an analysis of the performance and concluded, "The efficient parallel system needs to concentrate more on reducing cross talk between the adjacent channels rather than perfecting the individual channel itself because imperfection due to cross talk tends to dominate [20]." Later on, a major contribution to OFDM was given in 1971, which is Weinstein and Ebert [21] presented the use of discrete Fourier transform (DFT) to perform the baseband modulation and demodulation.

2.2.1 OFDM signal

The fundamental principle of the OFDM concept is to split one high rate data symbol into a set of independent smaller symbols [2]. Each subsymbol is modulated on a separate subcarrier. Meanwhile, these subsymbols are transmitted simultaneously in time domain.

Signal Form
Here the OFDM signal formula is directly introduced by Equation (2.1).

$$s(t) = \begin{cases} \sum_{n=0}^{N-1} x_n e^{j2\pi f_n t} & 0 \leq t \leq T \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where, the x_n stands for the information which needs to be transmitted. Normally, they are modulated by using the Phase-Shift Keying (PSK) and Quadrature Amplitude Modulation (QAM), which determines how many bits can be assigned per time frame. Here Binary Phase-Shift Keying (BPSK) is shown as an example, which is the simplest PSK digital modulation technique used for digital data transmission. The constellation diagram of BPSK is shown in Figure 2.2, in which only 1 bit can be modulated. It means, if there are N subcarriers, then N bits can be transmitted per frame.

Subcarrier
$e^{j2\pi f_n t}$ in equation (2.1) stands for the subcarriers mentioned above. Figure 2.3 presents their 'pure harmonic'waveform.

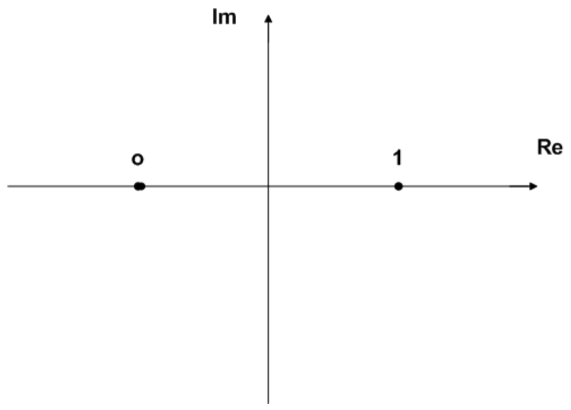


Figure 2.2: The constellation diagram for BPSK

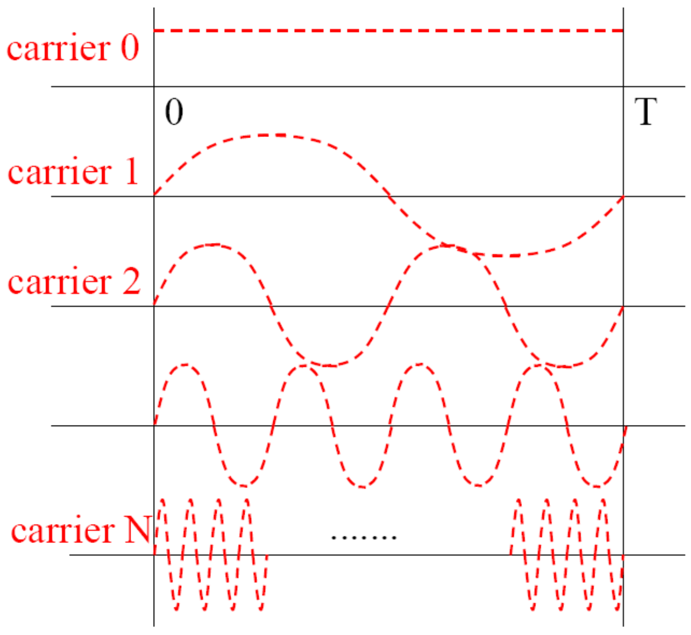


Figure 2.3: The subcarrier wave form of OFDM, from [2]

As shown in the figure, the carrier 0 is the DC constant at frequency zero, and the frequency f_1 of carrier 1 is 'the basic frequency', by which the basic frame interval T is determined ($T = \frac{1}{f_1}$). The other carriers are positioned at regular frequency intervals: $f_k = kf_1$.

The orthogonal property

The orthogonal property is the key property of OFDM. This property reduces the crosstalk between subcarriers, which allows to overlapping multicarriers in order to maximize the spectrum utilization. This is the biggest difference between OFDM and the conventional frequency-division multiplex system where the guard frequency bands are required between carriers.

The symbols $x_n e^{j2\pi f_n t}$ of the modulated signal have the property that they are 'orthogonal' on each other in the interval [0,T). For the usual inner product of complex functions on an interval, it can be defined by Equation (2.2).

$$< x_m, x_n >= \int_0^T (x_m e^{j2\pi f_m t})(x_n e^{j2\pi f_n t}) \tag{2.2}$$

If the product is equal to zero, the two signals are orthogonal. Here, for any two subcarriers mentioned above, as long as $m \neq n$, the product is zero. So all these subcarriers are mutually orthogonal. In Figure 2.3, the orthogonal subcarriers is shown.

Cyclic-prefixing

If there are no ISI and ICI caused by transmission, the orthogonality of subcarriers can be maintained, and the subcarriers can be separated by FFT at the receiver side. However, due to channel distortions, such as multipath delay, ISI and ICI can be introduced. To eliminate ISI completely, a guard time is introduced for each symbol. As long as the guard time is longer than the channel impulse response, the ISI can be eliminated. To eliminate ICI, the OFDM symbol is cyclically extended in the guard time. The cyclic-prefixing is shown in Figure 2.4.

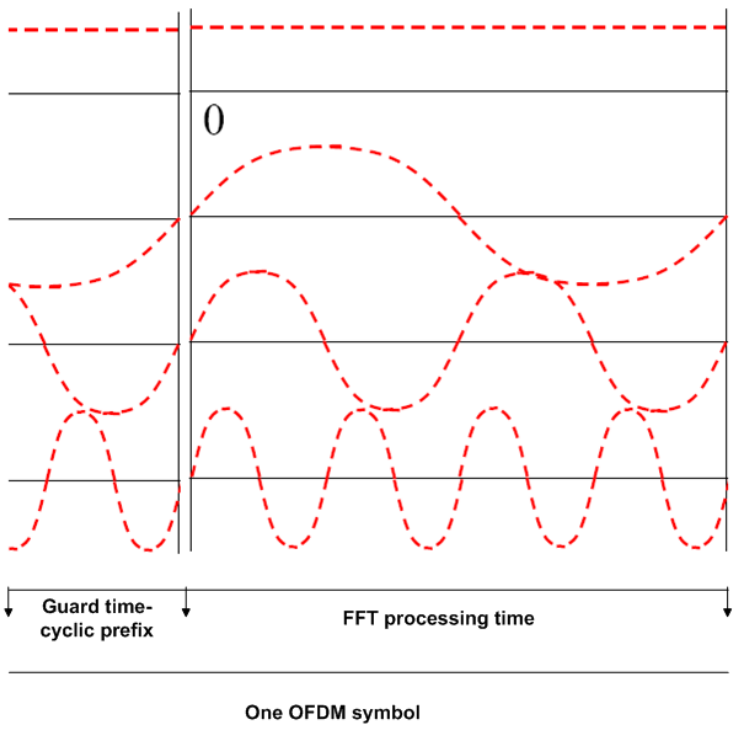


Figure 2.4: The cyclic-prefix of OFDM symbol

FFT presentation in OFDM
By observing the OFDM signal presented in Equation (2.1), it can be found that the equation looks very similar to the Fourier transform. If this waveform is sampled based on time slot $\frac{T}{N}$, during the period [0,T), N sample values can be obtained and their summation is shown in Equation (2.3).

$$X[k] = \sum_{n=0}^{N-1} x_n e^{j2\pi f_k n \frac{T}{N}} = \sum_{n=0}^{N-1} x_n e^{j2\pi k \frac{1}{T} n \frac{T}{N}} = \sum_{n=0}^{N-1} x_n e^{jkn \frac{2\pi}{N}} \tag{2.3}$$

which is exactly the expression of the Inverse Discrete Fourier Transform (IDFT) on the series $x(n)$. This means IDFT can be used to construct the OFDM signals.

2.2.2 OFDM System

The general OFDM system structure is shown in Figure 2.5. First, the incoming serial data go through the serial to parallel converter and are grouped into x bits each to form a complex number [3]. The number x is decided by the constellation mapping, such as BPSK. Next, the IFFT is used to modulate the complex numbers. After this, the numbers are converted to a series in time using a so called time-division multiplexer (TD-MUX) and the cyclic prefixes are added to avoid intersymbol interference (ISI) caused by multipath distortion [2]. By using the digital to analog converter, the discrete symbols are converted to the actual modulating analog signal. After going through the low-pass filter and Radio Frequency (RF) transmitter, the OFDM signal will be transmitted. At the receiver side the opposite operations take place. After the RF receiver and low pass filter, the analog to digital converter converts the analog signal to digital symbols. The cyclic prefix is removed and the symbols are parallelized in demultiplexer. At this time, the FFT processor is used to process the symbols and represents the original samples. The outputs of the FFT processor go to the deinterleaver and viterbi decoder to get further processed and decoded.

2.3 Multiband-OFDM for UWB system

Multiband-OFDM is an OFDM specification for UWB [4]. Multiband-OFDM was first proposed by Texas Instruments and was sponsored by Texas Instruments as a member of the Multiband OFDM Alliance, which is now part of WiMedia [22]. In December 2002, IEEE 802.15 3a was set up to develop a high data rate UWB PHY amendment for the IEEE 802.15 3WPAN standard. However, there are two different proposals to operate the wireless UWB, Multiband-OFDM UWB proposed by WiMedia, and Direct Sequence-UWB (DS-UWB), supported by UWB Forum. Because of the disagreements between the two groups, IEEE 802.15 3a was withdrawn. Nevertheless, a proposal for Multi-band OFDM UWB standard is published [12]. In December 2005, ECMA International, an international, private (membership-based) and non-profit standards organization for information and communication systems, published ‘ Standard ECMA-368: High Rate Ultra Wideband PHY and MAC Standard’, which specified physical

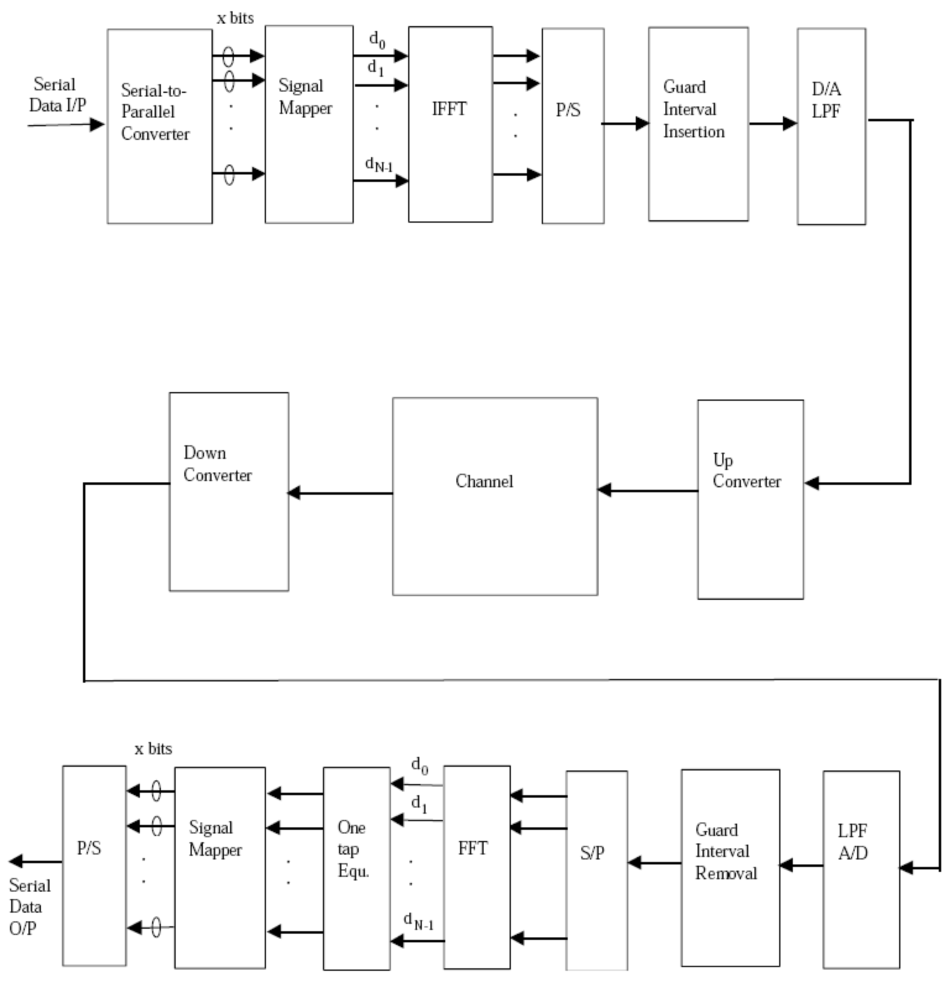


Figure 2.5: The general OFDM system, from [3]

layer (PHY) and medium access control layer (MAC) of the UWB technology based on Multiband-OFDM. In December 2007, the second revised version was released [5].

The ECMA-368 [5] uses the spectrum between 3.1-10.6 GHz supporting transmission data rate up to 480 Mb/s. The MB-OFDM is used to transmit data. The spectrum is divided into 14 bands, each with a bandwidth of 528 MHz, as shown in Figure 2.6. For the first 12 bands, every 3 bands form one band group and there are 4 such band groups. The last two bands are grouped into a fifth band group. A communication channel can either employ only one band or hop between the three bands of a band group [4].

Constellation Mapping and Data Rate for Communication
Based on the requirement of the standard, for data rates of 200 Mb/s and lower, the binary data should be mapped onto a QPSK constellation. For data rate of 320 Mb/s

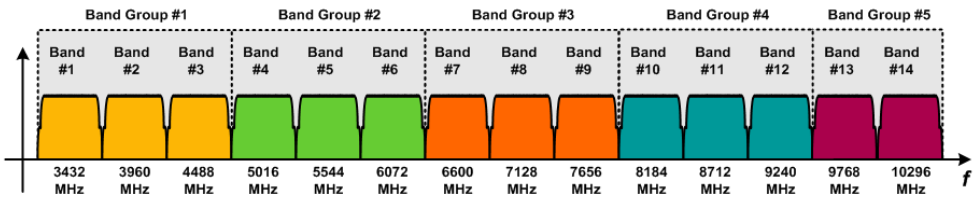


Figure 2.6: MB-OFDM band group allocation, from [4]

and higher, the binary data should be mapped onto a multi-dimensional constellation using a dual-carrier modulation (DCM) technique. Table 2.1 shows the data rates with required constellation mapping.

Table 2.1: Data rates for communication with required constellation mapping, from [5]

Data Rate(Mb/s)	Modulation	Coding Rate(R)
53,3	QPSK	$\frac{1}{3}$
80	QPSK	$\frac{1}{2}$
106.7	QPSK	$\frac{1}{3}$
160	QPSK	$\frac{1}{2}$
200	QPSK	$\frac{5}{8}$
320	DCM	$\frac{1}{2}$
400	DCM	$\frac{5}{8}$
480	DCM	$\frac{3}{4}$

QPSK Mapping
The QPSK is to divide the binary serial input data into groups of two bits and converted the group into a complex number representing one of the four QPSK constellation points, as shown in Figure 2.7.

Dual-carrier Modulation (DCM) [5]
The binary serial input data should be divided into groups of 200 bits and converted into 100 complex numbers using a technique called dual-carrier modulation. The conversion is performed as follows. First, the 200 bits are grouped into 50 groups of 4 bits. After this, each group of 4 bits is mapped onto a four dimensional constellation and converted into two complex numbers. The complex numbers are normalized in order for modulation.

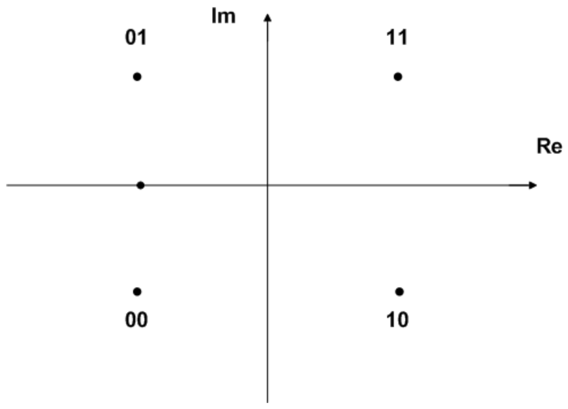


Figure 2.7: The QPSK bit coding

Transmitted Signal
The presentation of the transmitted RF signal is shown in Equation (2.4).

$$S(t) = Re\{\sum_{n=0}^{N_{packet}-1} s_n(t - nT_{SYM})e^{(j2\pi f_c(q(n))t)}\} \tag{2.4}$$

Where Re() represents the real part of the signal, T_{SYM} is the symbol length, N_{packet} is the number of symbols in the packet, $f_c(m)$ is the centre frequency for the m_{th} frequency band, q(n) is a function that maps the n_{th} symbol to the appropriate frequency band, and $s_n(t)$ is the complex baseband signal representation for the n_{th} symbol, which must satisfy the following property: $s_n(t) = 0$ for $t \notin [0, T_{SYM})$.

Figure 2.8 is an example of the transmitted RF signal using three frequency bands. It can be seen the first three symbols transmitted in three different bands and the next three symbols repeated. It can be seen that a zero-padded suffix is added at the output of IFFT.

2.4 The requirement of FFT for Multiband OFDM system

The FFT related requirement defined by the ECMA International is shown in Table 2.2.

As shown in the table, the required sampling frequency is 528MHz and the total number of subcarriers, which determines the FFT size, is 128. Based on [1], the FFT size selection is dependent on many factors. To consider the chip area, the FFT size should be as small as possible. On the contrary, the use of a smaller FFT size increases the overhead due to cyclic prefixing and degrades the range [1]. The size of 128 point is considered as a trade-off choice. In these subcarriers, the number of data subcarriers is 100 and the number of pilot subcarriers and guard subcarriers are 12 and 10 separately. The time period available for the IFFT and FFT is 242.42ns, which is the inverse of

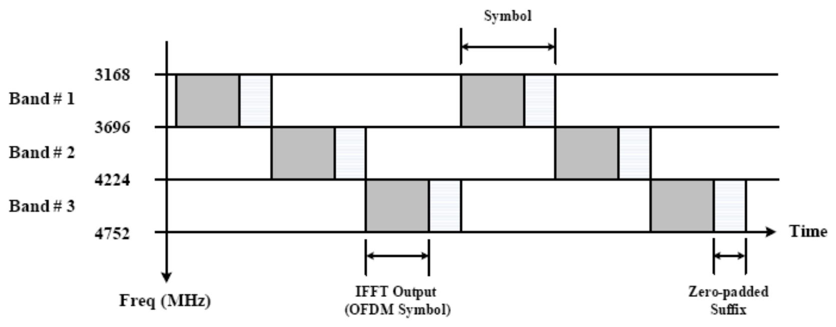


Figure 2.8: Realization of a transmitted signal using three bands, from [5]

Table 2.2: The FFT related requirements of ECMA

Parameter	Description	Value
f_s	Sampling frequency	528MHz
N_{FFT}	Total number of subcarriers(FFT size)	128
N_D	Number of data subcarriers	100
N_P	Number of pilot subcarriers	12
N_G	Number of guard subcarriers	10
N_T	Total number of subcarriers used	122
N_f	Subcarrier frequency spacing	4.125MHz
T_{FFT}	IFFT and FFT period	242.42ns
N_{ZPS}	Number of samples in zero-padded suffix	37
T_{ZPS}	Zero-padded suffix duration in time	70.08ns
T_{SYM}	Symbol interval	312.5ns
F_{SYM}	Symbol rate	3.2MHz
N_{SYM}	Total number of samples per symbol	165

sampling frequency $T_{FFT} = \frac{1}{f_s}$. There are 37 zero padded suffix samples, which take 70.08ns. So the total symbol interval is 312.5ns ($T_{SYM} = T_{FFT} + T_{ZPS}$) and the total number of samples per symbol is 165.

Time constraint analysis

There are some conflicts in the time constraint of FFT processing time. One opinion is that the symbol interval 312.5ns should be used as the FFT period [9] [15]. The other one is that the FFT period 242.42ns as specified by the standard should be used as the time constraint [4]. The key point of this problem is whether the zero padded suffix samples period can be used for processing in the time constraint. To analyze this problem, the OFDM system needs to be checked. The structure of Figure 2.5 is zoomed in and the FFT area is focused, as shown in Figure 2.9.

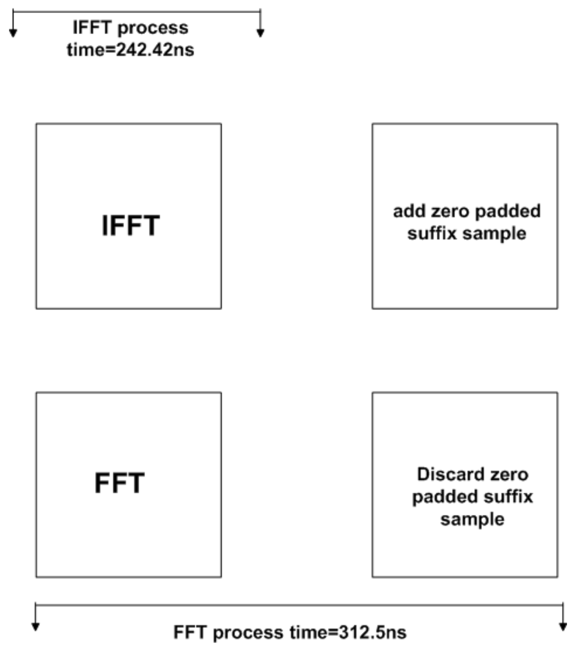

IFFT process time=242.42ns
IFFT
add zero padded suffix sample
FFT
Discard zero padded suffix sample
FFT process time=312.5ns

Figure 2.9: The FFT time constraint analysis

In the figure, it can be seen, at transmitter, the IFFT uses 242.42ns to process the data and after the processing, the zero padded suffix samples are added. However, at the receiver, the FFT processor can use 312.5ns to process the received data. The reason is that, after discarding the zero padded suffix sample, the time slot of two successive received data symbols is still 312.5ns. In order for both FFT and IFFT utilization, tighter time slot 242.42ns is employed in the following design.

Numerical Precision Choice
The data bits choice is a critical issue for FFT processor design. The trade-off between chip area consideration and signal to quantization noise ratio (SQNR) directly determines the choice. Here, only the quantization and the saturation effects related SQNR is considered. This key issue should be simulated and briefly decided before the processor design. After the processor design finished, the SQNR should be simulated again, which is further dealt with in Chapter 6. The reason is that the architecture of the processor, which affects the saturation and rounding effects, also changes the SQNR.

The first decision need to be made is the data type choice, the floating points or the fixed points. Actually, most of the ASIC DSP chips use fixed point, because the floating points are too costly and complex to implement. By considering the speed, area and economy, the practical UWB receiver is likely to employ a fixed point FFT [23]. By using fixed points to implement the digital system, it inevitably involves quantization of signals and coefficients in the system [24]. The first quantization error is due to the coefficients and input data quantization. In FFT processors, this means the input data and the twiddle factors W_N^r should be quantized in order to fit for the fixed data type. Another quantization error is caused by the signal rounding and overflows during the arithmetic calculation of the internal stages. These quantization errors are analyzed in [25]. This paper evaluates the variance of the quantization errors of fixed-point computation of the FFT. The complex multiplication in each stage, which requires four real multiplications, is considered as the main factor to cause the quantization errors. There can also be saturation effects caused by addition in each stage to produce the quantization errors. To eliminate this, proper scaling in each stage is used during the realization.

For the Multiband OFDM UWB system, Sherratt, et al [23] discussed the requirements on the numerical precision for practical consumer electronic solution. Based on the performance by using channel model and compared with floating points analysis, he concludes 8 bits are a good practical implementation for the data into and out of the FFT core, and 11 points gives good approximation for the internal data representation.

Nevertheless, brief Matlab simulation of the SQNR caused by the fixed point saturation and rounding of inner stages is necessary before the implementation of the FFT processor. Here, common radix 2 Decimation in Time (DIT) FFT algorithm is used to simulate fixed point 128 data FFT. For the seven stages, each stage performs a right shift, which means the FFT computation has a scaling factor of $\frac{1}{2}$ each stage. The floating point and 15 bits fixed point FFT outputs are shown in Figure 2.10. The input signals shown in the upper figure are the QPSK modulated OFDM signals and its fixed point repesentation. Because of the modulation property and IFFT processing, the values of original OFDM signals are all below 1. The maximum value can be obtained when all the carrier signals are coherently added. In fact, from many time simulations, it is found that the absolute values of most QPSK modulated OFDM signals are in the range of 0$\sim$0.2. The lower figure shows both of the floating and fixed point FFT outputs. It is shown that the fixed point FFT causes quantization errors.

Above 25dB SQNR is considered acceptable in Multiband-OFDM system [26]. Different bits are used to simulate in FFT in order to decide the proper number of bits that need to be used. It is found that the SQNR of 15 bits is mostly above 40dB, which is shown in Figure 2.11. Nevertheless, when the 14 bits are used as fixed point to simulate, at some points, only 20dB SQNR can be achieved, which is shown in Figure 2.12. Therefore, 15 bits are used as fixed point format to design the FFT processor.

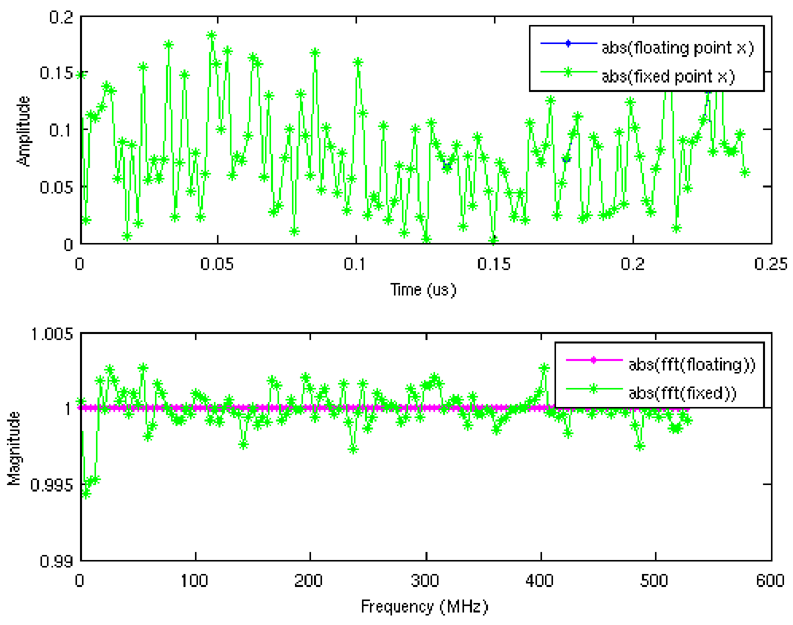


Figure 2.10: The comparison of the floating point FFT with the fixed point FFT

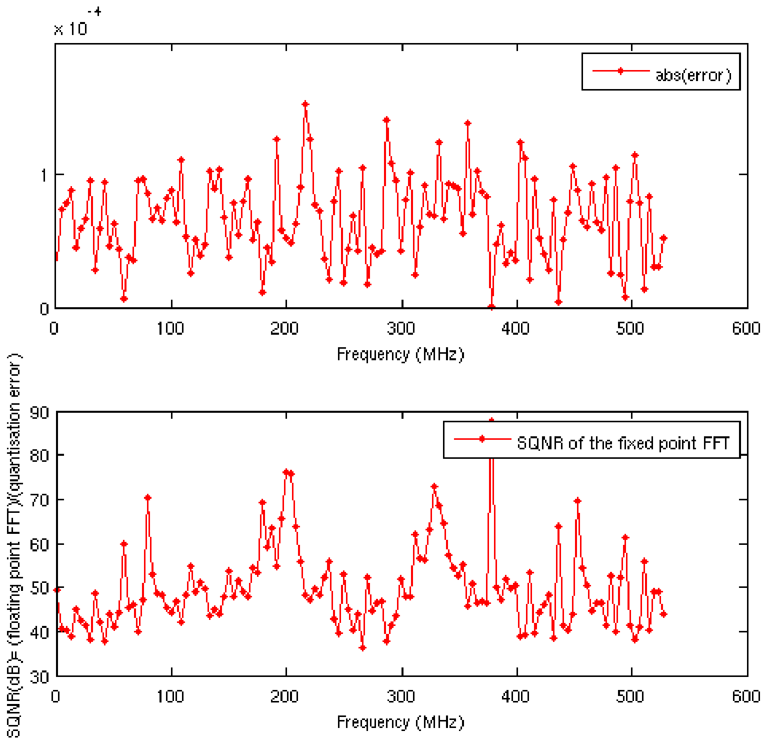


Figure 2.11: The Error and SQNR analysis of 15 bits fixed point FFT

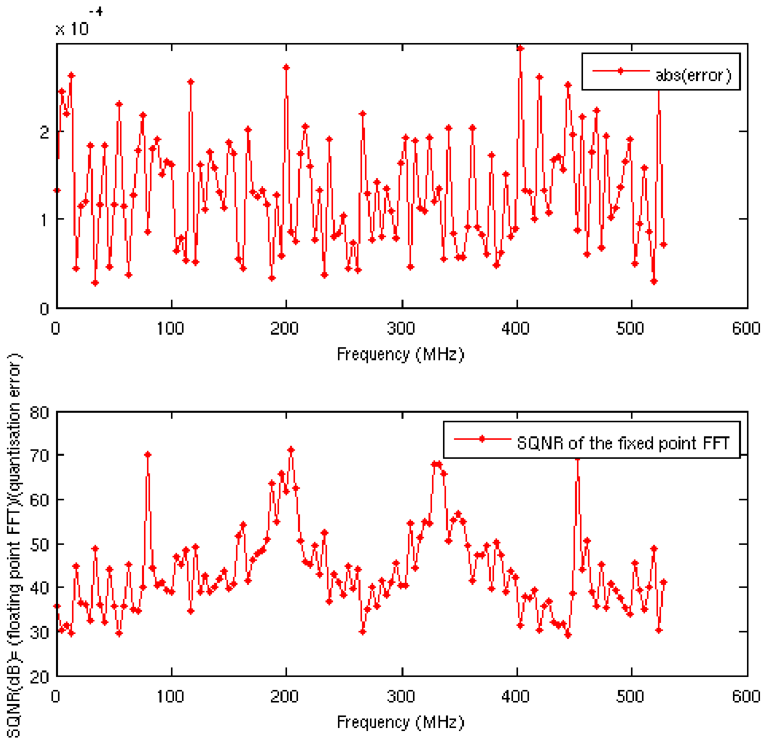


Figure 2.12: The Error and SQNR analysis of 14 bits fixed point FFT

3 Fast Fourier Transform and Its Different Algorithms

As shown in the previous chapter, in MB-OFDM UWB system, the FFT takes an important role in the total base-band circuit. Based on the system requirement, 128 point FFT need to be processed by using proper algorithm, which should be optimal for ASIC implementation. This chapter focuses on the different algorithms used to realize the Fast Fourier Transform. The advantages and disadvantages of these algorithms are also analyzed.

3.1 The Discrete Fourier Transform

The DFT of $x(n)$, written as $X(n)$, is defined in Equation (3.1), which is an N-point sequence.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-i2\pi nk/N}, \qquad k = 0, 1, ..., N-1 \tag{3.1}$$

The $x(n)$ and $X(n)$ are, in general, complex. The indices n and k are integers.

Here, W_N, which is called twiddle factor, is introduced by Equation (3.2)

$$W_N = e^{-i2\pi/N} = cos(\frac{2\pi}{N}) - \mathtt{i}sin(\frac{2\pi}{N}) \tag{3.2}$$

W_N^{nk} can be used instead of $e^{-i2\pi nk/N}$, so the equation above could be written as

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad k = 0, 1, ..., N-1 \tag{3.3}$$

3.2 The Fast Fourier Transform and its algorithms

The Fast Fourier Transform (FFT) [27] [6] is an efficient algorithm for computing the DFT. It is based on the fundamental principle of decomposing the computation of the discrete Fourier transform of a sequence of length N into successively smaller discrete Fourier transforms. The FFT generates the same result as DFT, however the computation complexity for N numbers is reduced from $O(N^2)$ to $O(Nlog(N))$.

After Cooley and Tukey [27] publishing the FFT algorithm for faster computation of discrete Fourier transform, several fast computation algorithms were proposed. Based on different decomposition, different algorithms could be obtained for computing the discrete Fourier transform. Here some of the algorithms are introduced.

3.2.1 Decimation-in-time FFT Algorithms

From the definition in [6], algorithms which decompose the sequence $x[n]$ into successively smaller subsequences, are called decimation-in-time algorithms.

In order to explain this algorithm clearly, the Radix-2 DIT algorithm is derived here to demonstrate the decomposition.

From the DFT formula, $X(k)$ is given by

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}, \qquad k = 0, 1, ..., N-1 \tag{3.4}$$

In Equation (3.4), $x(n)$ could be separated into even-numbered points part and odd-numbered points part. So the formula above can be rewritten as

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1) W_N^{(2r+1)k}, \quad k = 0, 1, ..., N/2-1 \tag{3.5}$$

$$= \sum_{r=0}^{N/2-1} x(2r) W_N^{2rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) W_N^{2rk} \tag{3.6}$$

$x(n)$ in the equation above is divided into even sequence $x(2r)$ and odd sequence $x(2r+1)$. It is also known as $W_N^2 = W_{N/2}$ since

$$W_N^2 = e^{-2i2\pi/N} = e^{-i2\pi/(N/2)} = W_{N/2} \tag{3.7}$$

Consequently, Equation (3.6) can be written as

$$X(k) = \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^{rk}, \quad k = 0, 1, ..., N/2-1 \tag{3.8}$$

The data flow of Equation (3.8) is shown in Figure 3.1.

By analyzing this figure, for each divided $\frac{N}{2}$-point DFT, the calculation operation requires $O((\frac{N}{2})^2)$ complex multiplications and complex additions. Then, to combine the parts together, $O(N)$ complex multiplications and complex additions are also needed to get the N-point DFT. So in total, for once divided N-point DFT, $N + 2 \times (\frac{N}{2})^2$ complex multiplications and complex additions are needed. Compared with original N-point DFT, which needs $O(N^2)$ complex multiplications and complex additions, for large N,

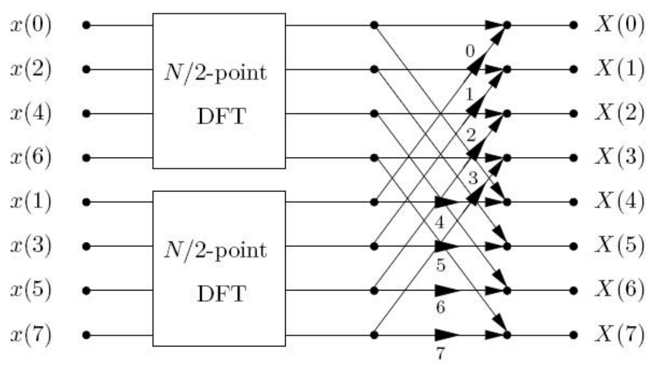


Figure 3.1: the DIT decomposition of an N-point DFT computation into two (N/2)-point DFT computations, from [6]

nearly 50% reduction in the number of operations is obtained.

Here by using the symmetry and periodicity of the twiddle factor W_N, the computation can be reduced further. Figure 3.2 shows one butterfly structure, which has two input values in first stage, and the two output values obtained by multiplying each input value with twiddle factors W_N^r and $W_N^{r+\frac{N}{2}}$ separately and adding these values with another input value.

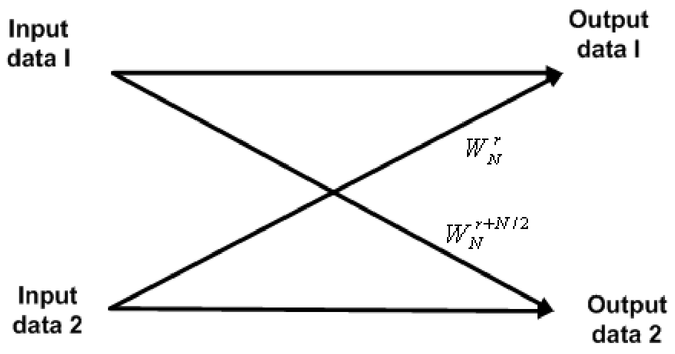


Figure 3.2: Butterfly with full twiddle factors in DIT algorithm

The twiddle factor $W_N^{r+\frac{N}{2}}$ can be rewritten as

$$W_N^{r+\frac{N}{2}} = W_N^{\frac{N}{2}} W_N^r = -W_N^r \tag{3.9}$$

since

$$W_N^{\frac{N}{2}} = e^{-j(\frac{2\pi}{N})\frac{N}{2}} = e^{-j\pi} = -1 \tag{3.10}$$

So Figure 3.2 can be changed in the figure below:

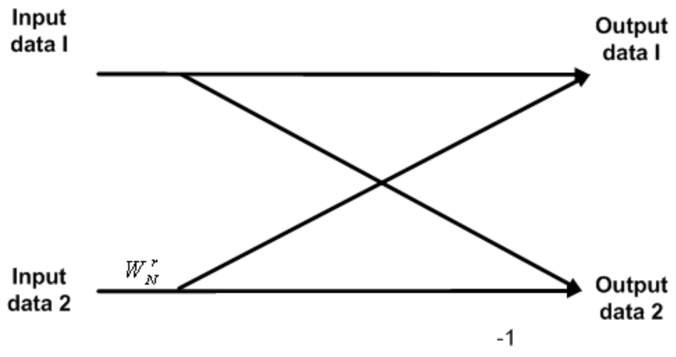


Figure 3.3: Butterfly with simplified twiddle factors in DIT algorithm

The structure consists of one complex multiplier and two complex adders which will take smaller area for implementation than two complex multipliers and adders.

For each $\frac{N}{2}$-point DFT, if it still can be successively divided by 2, then above steps can be used again for these $\frac{N}{2}$-point DFT. Figure 3.4 shows the data flow of 8 point DIT FFT.

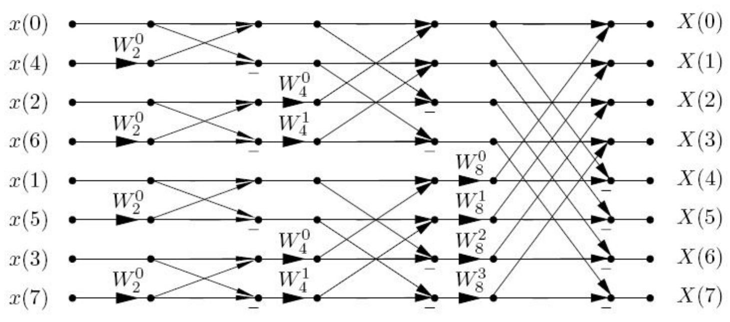


Figure 3.4: 8 Point DIT FFT data flow, from [6]

From the derivation above, it is shown, for the radix-2 DIT algorithms, the computation operations are $O(NlogN)$ which is much smaller than $O(N^2)$ in the original N-point DFT.

3.2.2 Decimation-in-Frequency FFT Algorithm

Compared with Decimation-in-Time FFT algorithm, dividing the outputs of DFT in smaller sequence can also be considered.

As the same method being used in 3.2.1, the Radix-2 DIF algorithm function is also illustrated by using the algorithm derivation.

First, the DFT formula is recalled here

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \qquad k = 0, 1, ..., N-1 \tag{3.11}$$

The even-numbered frequency samples are taken from Equation (3.11), which is

$$X(2r) = \sum_{n=0}^{N-1} x(n)W_N^{2nr} \tag{3.12}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{2nr} + \sum_{\frac{N}{2}}^{N-1} x(n)W_N^{2nr}, \qquad r = 0, 1, ..., \frac{N}{2} - 1 \tag{3.13}$$

The second summation from $\frac{N}{2}$ to N can be changed into the summation from 0 to $\frac{N}{2}$. So the equation above can be expressed as

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} x(n)W_N^{2nr} + \sum_{n=0}^{\frac{N}{2}-1} x(n + \frac{N}{2})W_N^{2r(n+\frac{N}{2})}, \qquad r = 0, 1, ..., \frac{N}{2} - 1 \tag{3.14}$$

By using the property of W_N^{2nr},

$$W_N^{2r(n+\frac{N}{2})} = W_N^{2rn}W_N^{rN} = W_N^{2rn} \tag{3.15}$$

Equation (3.14) can be expressed as

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} (x(n) + x(n + \frac{N}{2}))W_{\frac{N}{2}}^{2nr}, \qquad r = 0, 1, ..., \frac{N}{2} - 1 \tag{3.16}$$

Following in the same way, the odd-numbered frequency sequences can be expressed as

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} (x(n) - x(n + \frac{N}{2}))W_N^n W_{\frac{N}{2}}^{2nr}, \qquad r = 0, 1, ..., \frac{N}{2} - 1 \tag{3.17}$$

Now, Equation (3.16) and Equation (3.17) can be combined together to get the first order DIF derivation. This derivation is shown in Figure 3.5. First, The $x(n)+x(n+\frac{N}{2})$ and $x(n)-x(n+\frac{N}{2})$ is computed. Then, the $x(n)-x(n+\frac{N}{2})$ part is multiplied with the twiddle factor W_N^n. After this, the even and odd frequency sequences can be obtained by computing the following $\frac{N}{2}$ point DFT of the two sequences separately.

For each N/2 point DFT above, if it still can be successively divided by two, the above steps will repeat. A 8 point radix-2 decimation in frequency algorithm data flow is shown as Figure 3.6.

It is shown from the derivation above, for the radix-2 DIF algorithms, the computation operations are also $O(NlogN)$ which is the same as radix-2 DIT algorithm and is much smaller than $O(N^2)$ in the original N-point DFT.

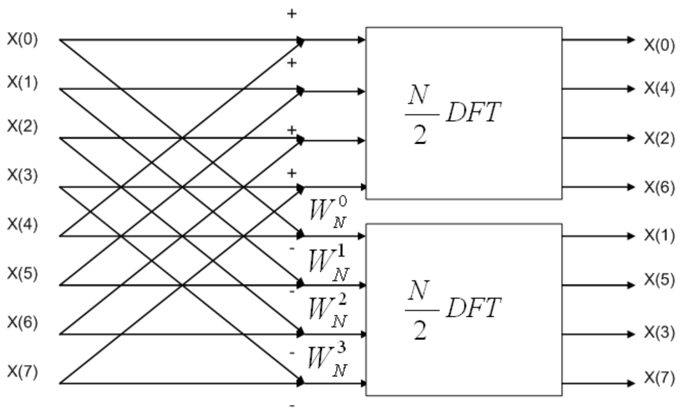


Figure 3.5: 8 Point Radix-2 divided once DIF data flow

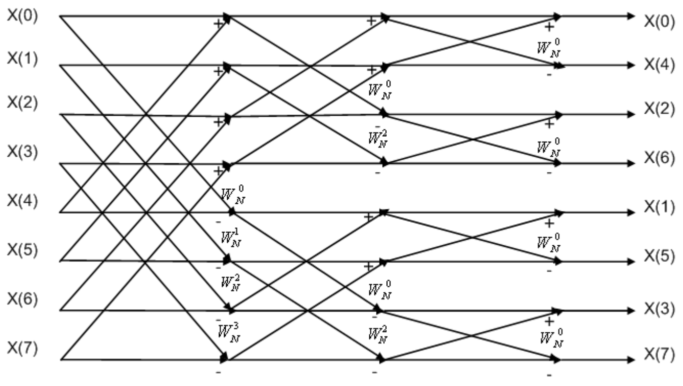


Figure 3.6: 8 Point Radix-2 DIF algorithm data flow

3.2.3 Radix 4 and higher radix Algorithms

3.2.3.1 Radix 4

If the data sequence N is a power of 4, the same method in Radix-2 algorithms can be used to divide the sequence by 4 in order to let all the elementary computations be 4 point DFT's. Both the DIT and DIF algorithms can be used for radix 4. A 16 point DIF radix 4 algorithm data flow is shown in Figure 3.7 to illustrate the function of radix 4 algorithm.

In the figure, the numbers on the left side mean input data and the numbers on the right side are output data which are in bit reversed order. In the first column butterflies, every four input data are processed in the same butterfly and four output data are produced. We can zoom in to one radix 4 butterfly in order to check its function and compute operations.
As shown in the Figure 3.8, One radix 4 butterfly needs 3 multipliers and 12 adders, whereas radix 2 butterfly needs 4 multipliers and 8 adders. Therefore, radix 4 butterfly

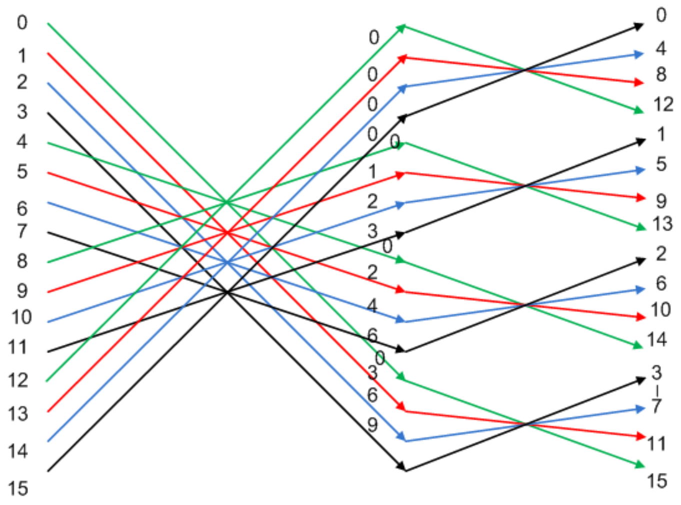


Figure 3.7: Radix 4 DIF Data flow

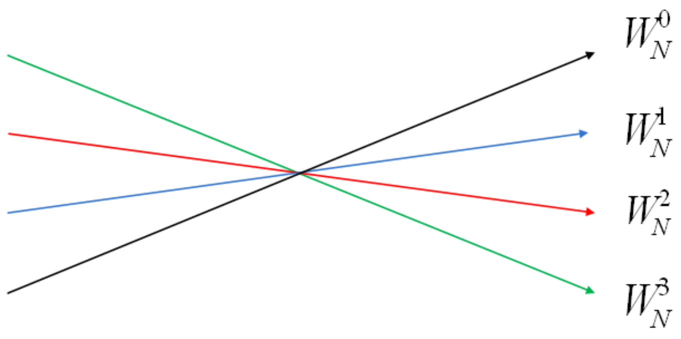


Figure 3.8: One Radix 4 DIF Butterfly

reduces 1 multiplier but requires 4 more adders. For ASIC implementation, 1 multiplier takes much bigger areas than 4 adders. Therefore radix 4 algorithm has the advantage in this perspective. However, in the meantime, the control and data flow design complexity is also increased.

3.2.3.2 Higher Radix Algorithms

As long as the input sequence is an order of r ($r > 4$), DFT can be divided into several radix r DFT blocks. It is called higher radix algorithm. Its main benefits are higher processing speed, less stages, and less arithmetic blocks. However, the design complexity increases correspondingly with the radix order. For example, in one butterfly processing time, r twiddle factors are required to be read in the processor simultaneously, which requires much more efforts on memory and time control than radix 2 algorithm. In fact, higher than radix 8 algorithms are not very often employed for ASIC implementation. Due to the complexity of the structure, the efficiency of the processor can not be improved.

3.2.4 Radix 2^2 Algorithm and Radix 2^x Algorithms

3.2.4.1 Radix 2^2 Algorithm [8] [28]

In 1996, He and Torkelson integrated the twiddle factor decomposition and proposed a so-called radix 2^2 algorithm. This algorithm has the same multiplicative complexity as radix 4 algorithm, but retains the butterfly structure of radix-2 algorithm [8].

To explain this algorithm, the first two steps of the decomposition of radix 2 DIF FFT is analyzed, and the Common Factor Algorithm (CFA) is used to illustrate. In equation (3.1), for the first two steps of the DIF decomposition, the n and k should be decomposed as

$$n =< \frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3 > N \tag{3.18}$$
$$k =< k_1 + 2k_2 + 4k_3 > N \tag{3.19}$$

$<>$ N means the total value of n and k is N. Substitute the decomposition above in Equation (3.1), it can be obtained that

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}\sum_{n_1=0}^{1} x(\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3)W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \tag{3.20}$$

$$= \sum_{n_3=0}^{\frac{N}{4}-1}\sum_{n_2=0}^{1}[B_{\frac{N}{2}}^{k_1}(\frac{N}{4}n_2 + n_3)W_N^{(\frac{N}{4}n_2+n_3)k_1}]W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} \tag{3.21}$$

Here,

$$B_{\frac{N}{2}}^{k_1} = x(\frac{N}{4}n_2 + n_3) + (-1)^{k_1}x(\frac{N}{4}n_2 + n_3 + \frac{N}{2}) \tag{3.22}$$

For normal radix 2 DIF algorithm, the expression in braces is computed first as a first stage in Equation (3.21). However, in radix 2^2 algorithm, the key idea is to reconstruct the first stage and second stage twiddle factors, which is shown in Equation (3.24)

$$W_N^{(\frac{N}{4}n_2+n_3)k_1}W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} = W_N^{Nn_2k_3}W_N^{Nn_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_N^{4n_3k_3} \tag{3.23}$$
$$= (-j)^{n_2(k_1+2k_2)}W_N^{n_3(k_1+2k_2)}W_N^{4n_3k_3} \tag{3.24}$$

By using the Equation (3.24), the Equation (3.21) can be changed into

$$X(k_1 + 2k_2 + 4k_3) = \sum_{n_3=0}^{\frac{N}{4}-1}[H(k_1, k_2, n_3)W_N^{n_3(k_1+2k_2)}]W_N^{4n_3k_3} \tag{3.25}$$

The $H(k_1, k_2, n_3)$ is expressed as

$$H(k_1, k_2, n_3) = [x(n_3) + (-1)^{k_1}x(n_3 + \frac{N}{2})] + (-j)^{(k_1+2k_2)}[x(n_3\frac{N}{4})) + (-1)^{k_1}x(n_3 + \frac{3N}{4})] \tag{3.26}$$

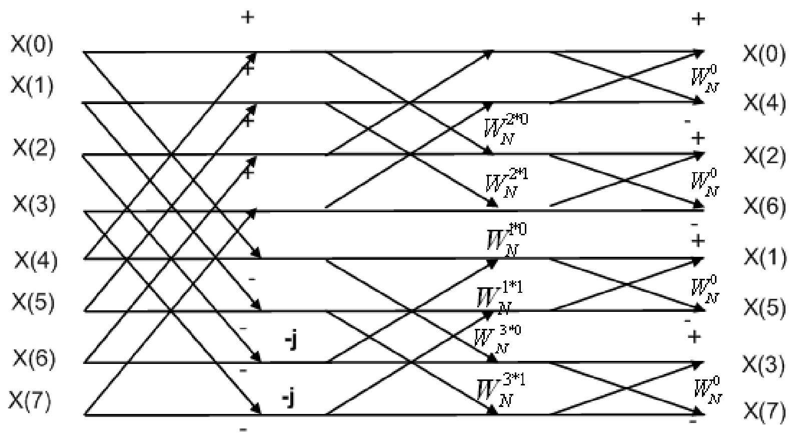


Figure 3.9: The application of Radix 2^2 algorithm for 8 point FFT

The application of this algorithm to 8 point FFT is shown in Figure 3.9. It is shown in the figure that the radix 2^2 algorithm was only used once, because 8 point DFT can only be decomposed once by radix 4. Therefore, radix 2^2 algorithm can only be used for the first two stages. For the last stage, normal radix 2 DIF algorithm is used. By using radix 2^2 algorithm, complex multiplication of the twiddle factor in the first stage is changed into multiplying (-j) which means just real-imaginary swapping and sign inversion. Therefore, one complex multiplier can be reduced for 8 point FFT during implementation.

3.2.4.2 Higher Radix 2^x Algorithms

Some researchers tried to combine more stages twiddle factors in order to reduce more complex multipliers. In 1999, He, et al [28] provided Radix 2^3 algorithm which combined 3 stage twiddle factors and reconstructed them. In 2005, Jung-Yeol OH and Myoung-Seob LIM [29] [30] provided the Radix 2^4 Algorithm, which reconstructs 4 stages twiddle factors. The main idea of this algorithm is shown as follows. The derivation details can be found in [29].

Figure 3.10 shows the 32 points Radix 2^4 Algorithm data flow. In the first 4 stages, the twiddle factor multiplication of the first and third stages becomes only real-imaginary swapping and sign inversion by using the algorithm. For the second stage, the twiddle factors can be reduced to W_{16}^0, W_{16}^1, W_{16}^2, W_{16}^3, which means constant multipliers with some control units can be employed instead of a complex multiplier for the twiddle factor multiplication . The complex multiplication is only required at the fourth stage. The implementation area can be reduced by employing this algorithm because the constant multiplier takes smaller area than complex multiplier in ASIC chips.

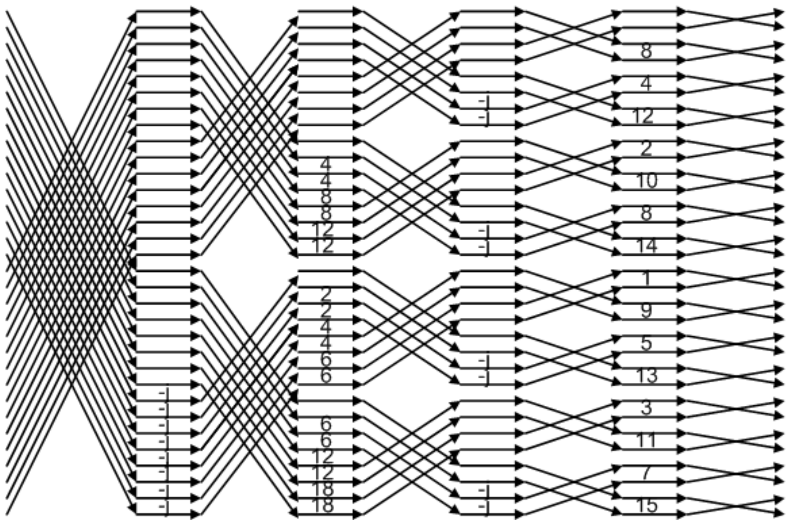


Figure 3.10: 32 Points Radix 2^4 DIF Algorithm data flow

3.2.5 Other FFT Algorithms

Split Radix
The key idea of this algorithm is to use different radices for different decimation-products of the sequence. For example, the odd indexed numbers part can use radix 2 algorithm and the even indexed numbers part can use the radix 4 algorithm. The purpose of doing this is to reduce the number of multipliers and adders. However, because of the irregular structure, The implementation is much more complicated than normal radix r algorithms.

Mixed Radix
If the radix r algorithm is used, normally log_r^N stages are required for implementation. Mixed radix algorithm means that different radix r butterflies for different stages are employed. If the radix is properly chosen for different stages based on the specifications of the FFT, the optimized design for speed and area can be obtained. However, for generic FFT, in which the process points need to be scaled, this algorithm is not suited because the scaling porperty is hard to handle by this algorithm.

Algorithms such as prime factor algorithm [31], Winograd Fourier Transform [32], etc have the advantage of requiring less multiplications. However, more additons are required than previously-discussed FFT. Because of the irregular structure determined by the algorithm, they are normally implemented for special usage.

3.3 Algorithms Comparison for ASIC Implementation

This chapter briefly reviewed the algorithms of Fast Fourier Transform. The radix 2 DIT and DIF algorithms have simple structure and clear data flow, which is easy to implement and is suitable for generic FFT implementation. Nevertheless, these algorithms need large memory to store data at inner stages, which is not efficient for the ASIC implementation. The radix 4 or higher radix algorithms need less multiplications than radix 2 algorithm. However, the data flow control is complex and more data need to be fetched at the same time, which will increase the area for implementation. This chapter also discusses the radix 2^2 and 2^x algorithms, which integrate the twiddle factor decomposition every two or X stages. The radix 2^2 algorithm has the same multiplicative complexity as radix 4 algorithm, but retains the butterfly structure of radix 2 algorithm, which is very suitable for ASIC implementation. Nowadays, based on literature research, there are two trends for FFT implementation of OFDM system, the mixed radix algorithms, like [4] and the pipeline structure based algorithms, like [13]. More detailed analysis is discussed in Chapter 4 combined with architecture choice.

FFT architectures Analysis and Design Choices 4

4.1 Overview of FFT Architectures

According to [16] [7], based on different structures, the FFT processors can be divided into memory based structure and pipeline-buffer structure.

4.1.1 Memory based structure

Single Memory
The single-memory architecture is the simplest memory system architecture. The simplest version of this structure is just one arithmetic block connected with one memory by bi-directional data bus as shown in Figure 4.1. In this situation, each butterfly input reads from the memory and writes back to the memory after processing. Therefore, for each of the log_r^N stages, the $\frac{N}{r}$ times butterfly calculation is required, where r is the radix number of selected algorithm. It is also possible to read all the data from the memory, which need parallel $\frac{N}{r}$ butterfly arithmetic blocks for processing. In this situation, N data width bus is to connect the memory and the arithmetic blocks and only one butterfly processing period is needed for each stage.

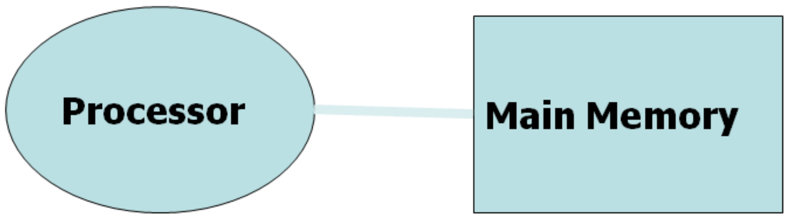


Figure 4.1: Single Memory Architecture Block, from [7]

Dual Memory
Connecting the arithmetic blocks with the two memory blocks is called dual memory structure as shown in Figure 4.2. Each memory block is connected to the arithmetic blocks separately. First, the data are read from one memory and processed in the arithmetics block. After processing, the data are written to the other memory. At the same time, the next data are read again from the first memory block and processed in a pipeline way. After finishing this processing stage, the same process repeats for the next stage. However, the data are now read from the second memory block and are written into the first memory block. This process looks like ‘ping pong’ between the two memory

blocks. The ASIP FFT processor designed by Ramesh Chidambaram [9] employs dual memory structure.

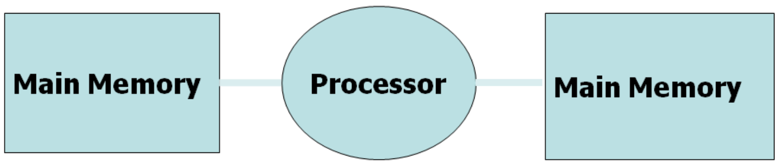


Figure 4.2: Dual Memory Architecture Block, from [7]

Array
The array architecture is to divide the whole processing into a number of independent processing elements with local buffers as shown in Figure 4.3 [7]. These independent processing elements are connected by different types of networks. This architecture is mainly used for large number data processor.

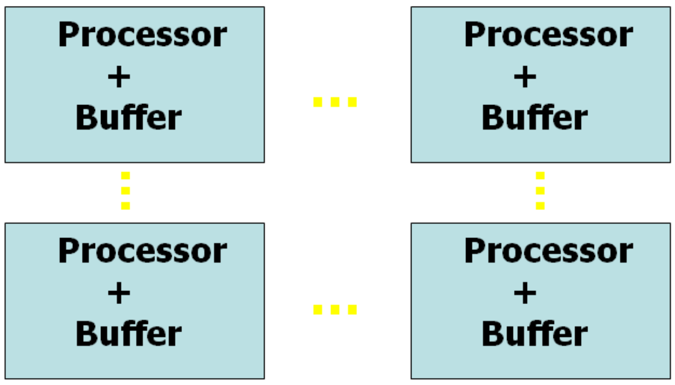


Figure 4.3: Array architecture block, from [7]

4.1.2 Pipeline-buffer Structure

Figure 4.4 shows an overview of the pipeline-buffer structure for FFT processor. It is considered as real time data processing architecture, in which the input data are read in series. Different pipeline architecture had been developed since 1970s [16]. He and Torkelson [8] summerized four main pipeline structures. Here, these structures are analyzed and compared based on [16] [8].

Multi-path Delay Commutator (MDC) [16] and Single-path Delay Feedback (SDF) [18] are the two main connection ways of each pipeline stages for FFT algorithm. Based on

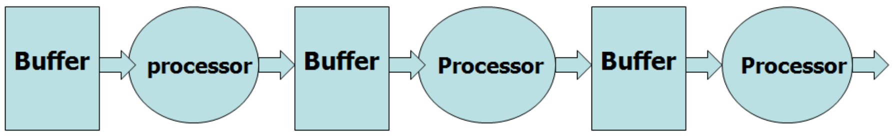


Figure 4.4: Overview of Pipeline Structure, from [7]

different radix decomposition, five main pipeline structures are R2MDC [16], R2SDF [18], R4SDF [33], R4MDC [16] and R4SDC [34].

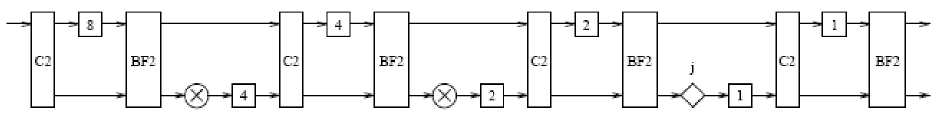
The most straightforward approaches of the pipeline implementation are R2MDC and R4MDC shown in Figure 4.5(i) and (iv). The input data have been separated into several data streams flowing forward to each pipeline stage. The outputs of these stages are also parallel and delayed to the next stages. The time delay between each parallel data need to be correctly matched, which is determined by the algorithms. It can be obtained that the memory utilization for R2MDC and R4MDC is 50% and 25% separately. The details are explained in [16].

Single-path Delay Feedback (SDF) combines the input memory and out memory of each stage. The input and output data of each stage share the same shift registers. In this way, the shift registers are more efficiently in R2SDF and R4SDF. Only one data path is employed to connect the stages as shown in Figure 4.5 (ii) and (iii). Moreover, the memory utilization of R2SDF and R4SDF is improved to 100% and 75% respectively. However the synchronization control of these structures is very complex because the butterfly computational unit is modified.

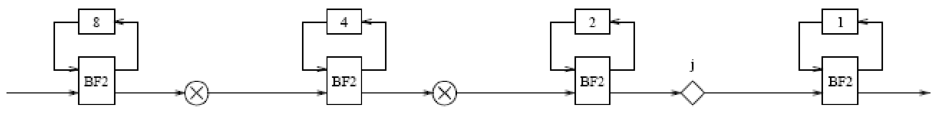
Radix 4 Single-path Delay Commutator [34] uses a programmable method to change the structure of normal radix 4 butterfly in order to reduce the complex multipliers required. And the memory is also reduced by a combined Delay-Commutator. Table 4.1 shows the comparison of the memory requirements of these structures.

Table 4.1: The Memory Requirement of Different Pipeline Structure

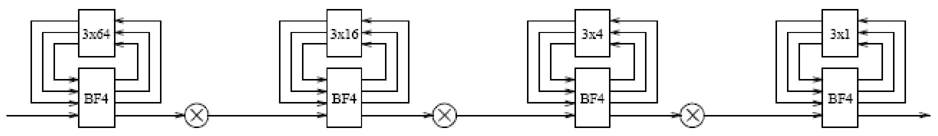
Structure	Memory Requirements
R2SDF	$N-1$ FFT number data
R2MDC	$\frac{3N}{2}-2$ FFT number data
R4SDF	$N-1$ FFT number data
R4MDC	$\frac{5N}{2}-4$ FFT number data
R4SDC	$2N-2$ FFT number data



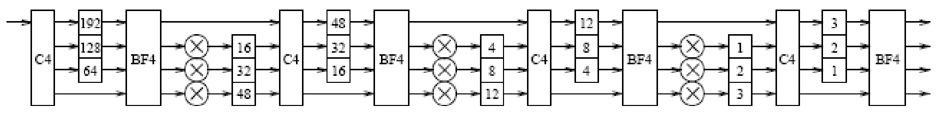
(i). R2MDC(N=16)



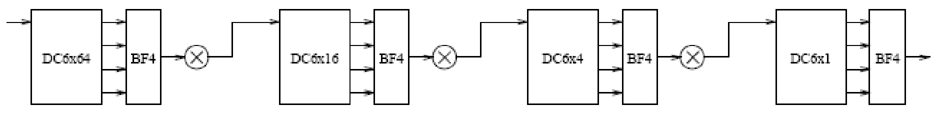
(ii). R2SDF(N=16)



(iii). R4SDF(N=256)



(iv). R4MDC(N=256)



(v). R4SDC(N=256)

Figure 4.5: The various schemes for pipeline FFT processor, from [8]

4.2 High Level Speed, Area and Power Analysis

Since the architecture is the main factor to determine the speed, area and power of FFT processor, deeper investigation and comparison of different architectures is necessary at high lever before the implementation. The analysis is based on architecture level which means the design details are not considered. Therefore the pipeline and shimming registers which are used for timing and delay control signals are not considered here and the multiplication is assumed to be finished at one clock cycle.

The 128 point FFT is used for analysis and the input data sampling rate is 528MHz, which are defined by [5]. In fact, this frequency is not suitable for implementation because the time slot is too short and more registers need to be added between combinational logic. Normally, the design area and power consumption increase exponentially as the frequency goes up and most consumer electronics products for UWB can not afford more than 300 MHz frequency. Thus 300MHz is used as the reference clock frequency to analyze the following architectures. One cycle period is approximate 3.3ns according to Equation (4.1).

$$T_{\texttt{clock period}} = \frac{1}{f_{\texttt{clock speed frequency}}} \tag{4.1}$$

At this stage, Only the number of memories, multipliers, adders is taken into account without referring to the real areas of each block. Because these information can only be available after synthesis.

The following parts are the analysis and comparison of different architectures.

Single Memory
For this architecture, one butterfly needs at least 3 clock cycles for reading from memory, processing, and writing back to memory. Thus approximately 10ns is required by one butterfly calculation. If radix 2 algorithm is used for this architecture, 448 radix 2 butterflies are needed for 128 point FFT because

$$\begin{aligned}\texttt{Butterflies required} &= \texttt{Stages numbers} \times \texttt{butterflies per stage} \\ &= 7 \times 64 \quad \texttt{for 128 points radix 2 algorithms} \\ &= 448\end{aligned} \tag{4.2}$$

The time slot for the FFT calculation is 242.42ns which is analyzed in Section 2.4. Thus, at most 24 clock cycles are available in this time slot ($24 = floor(\frac{242.42}{10})$). The required number of parallel butterflies is 19 which is calculated by Equation (4.3). This is unimplementable because 19 complex multipliers and 76 adders in total are required, which will take too much area in ASIC implementation.

$$\texttt{parallel number} = \frac{\texttt{total needed butterflies}}{\texttt{clock cycles available/clock cycles for one butterfly}} \tag{4.3}$$

Based on the analysis above, single memory structure need 128 data memories and 19 complex multipliers and 76 adders.

Dual Memory Structure
Dual memory structure can pipeline reading, processing, writing of one butterflies process. Therefore, only one clock cycle is needed for butterfly calculation theoretically. By using the radix 2 algorithm, the 448 butterflies are also needed for the FFT calculation. The parallel butterflies number is calculated in Equation (4.4).

$$\texttt{parallel number} = \frac{448}{72} = 7 \tag{4.4}$$

Two 128 data memories are required, which is determined by the structure. The number of multipliers and adders required are 7 and 28 respectively.

The radix 4 version single memory and dual memory structures need the same memories as radix 2. Slightly less multipliers are required, but more adders are required than radix 2.

R2MDC
For 128 point FFT, 7 stages are required for radix 2 algorithm based pipeline structure. In this structure, each stage needs one multiplier and four adders. So in total, 7 complex multipliers and 28 adders are needed for R2MDC. And based on the analysis in Section 4.1.2, the memory required is 190 data storage.

R2SDF
R2SDF needs the same stages as R2MDC, which means the same required multipliers and adders. However, the memory is reduced to 127 data storage.

R4MDC
It is not possible to implement 128 point FFT by only using R4MDC structure because 128 is not the power of 4. The only possible way to use R4MDC for 128 point is to employ mixed radix method which combines the radix 4 with radix 2 algorithms. The combination of 3 stages R4MDC with 1 stage R2MDC is used for 128 point. In this structure, the connection between R4MDC and R2MDC requires extra shift registers to delay the output of R4MDC. This method will increase the whole structure cost and the total latency. 10 multipliers and 157 data storage plus extra shift registers for delay are needed. Compared with other algorithms, it is not area and power efficient.

R4SDF
Like R4MDC, the combination of 3 stages R4SDF with 1 stage R2SDF is used for 128 point FFT. Because of the single path between each stage, the connection between R4SDF and R2SDF does not need extra shift registers. The required multipliers are 3 and the memory is 127 data storage. However, about 50 adders are needed for this structure.

R2^2SDF
The R2^2SDF structure uses the radix 2^2 algorithm to SDF architecture which is shown in Figure 4.6. The SDF architecture is the most memory efficient architecture and the radix 2^2 algorithm can save half of multipliers required compared with normal radix 2 algorithms. Both benefits can be perfectly integrated in R2^2SDF for 128 point FFT. The total multipliers are 3 and 127 data storages are needed.

The structures mentioned above are the main FFT structures available nowadays. Depending on different requirements, plenty of papers provide the modified versions of the structures analyzed here. Table 4.2 is the analysis summary. The required input buffers for different structure are not included.

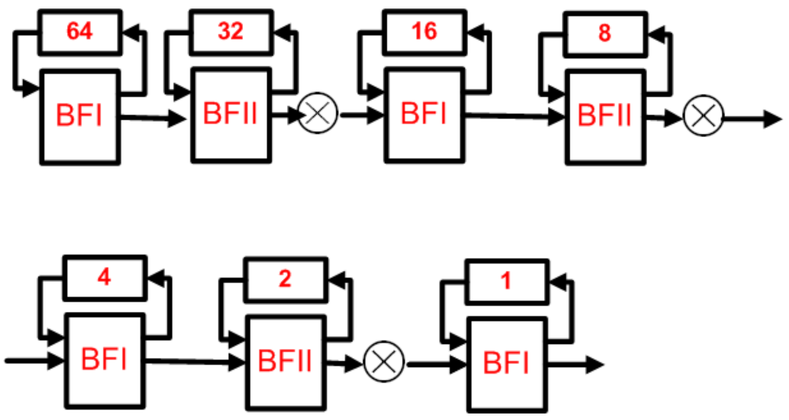


Figure 4.6: The radix 2^2 SDF Structure

Table 4.2: The Analysis of Different Pipeline Structure

Structure	Frequency	Memory Requirements	Complex Multipliers	Adders	design complexity
R2 single memory	300MHz	128 data storage	19	76	simple
R2 dual memory	300MHz	256 data storage	7	28	simple
R2SDF	528MHz	127 data storage	7	28	medium
R2MDC	528MHz	190 data storage	7	28	complex
R4SDF	528MHz	127 data storage	3	40	medium
R4MDC	528MHz	157 data storage plus extra the delay registers	10	40	complex
R2^2SDF	528MHz	127 data storage	3	28	complex

4.3 Design Choice

4.3.1 Algorithm and Architecture Choice

From previous section, it is shown that R2^2SDF structure is the best option in regard of the memories and arithmetic blocks utilization. Based on the requirement of MB-OFDM UWB system, neither single memory nor dual memory is suitable for ASIC implementation. The reason is that single memory structure needs parallel 19 butterflies which means the arithmetic blocks are too large for implementation. Dual memory structure needs 256 data memory block which is also not area efficient for ASIC implementation. Whereas, the pipeline structure is the most suitable choice for high throughput rate with affordable hardware cost. However, the frequency requirement for all pipeline structures is 528MHz which is the same as input data sampling rate. In fact, the 528MHz is unacceptable for current ASIC implementation based on the previous analysis. Therefore, it is crucial to reduce the working frequency if the pipeline structure is used. This difficult problem is discussed and solved in Chapter 5.

Regardless of the clock frequency, R2^2SDF structure is a reasonable choice for the FFT processor targeted for the MB-OFDM system. The first step for verifying this structure is to validate its function from Matlab simulation. The 128 point radix 2^2 algorithm based SDF structure FFT is simulated in Matlab and compared with the builtin FFT function of Matlab. The simulation result is shown in Figure 4.7. From this figure, it can be seen that there is only 10^{-16} order difference between the R2^2SDF structure and Matlab builtin algorithm. The difference is caused by twiddle factor reconstruction of Radix 2^2 algorithm.

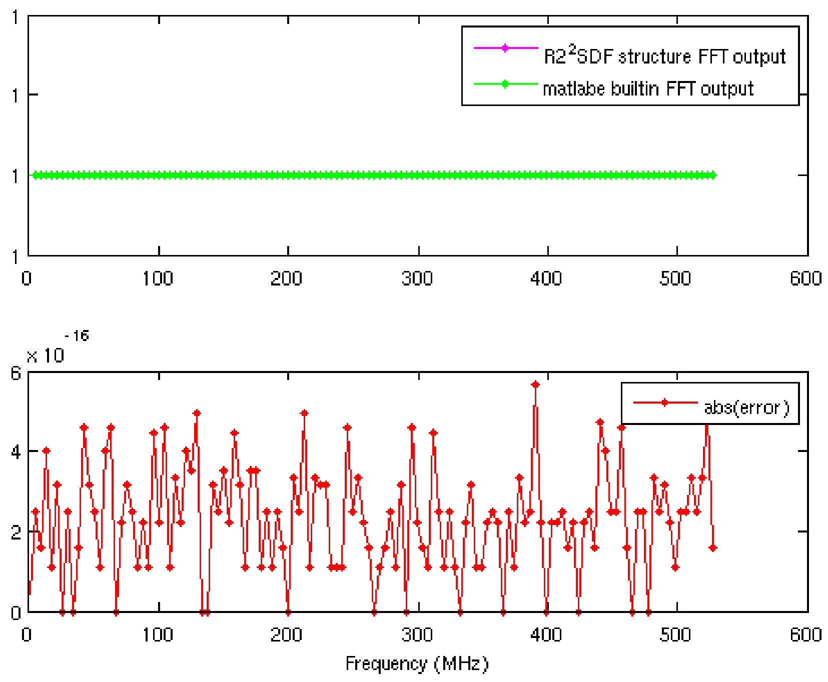


Figure 4.7: The R2^2 SDF structure simulation and comparison with Matlab builtin FFT algorithm

R2^2SDF based design choices is discussed in the following sections.

4.3.2 Fixed point data format

Two's Complement
The data presentation of this FFT uses two's complement. In two's complement, the first bit of a positive number is always 0, and the following bits is the standard binary value. The first bit of negative number is always 1, and the following bits can be obtained by the following steps. First, flip its standard binary positive value. Then, add 1 to the binary values.

Complex data
Due to the OFDM signal and twiddle factor property, the input data and twiddle factor are all separated into real and imaginary parts. Based on the requirements analyzed in Section 2.4, both real and imaginary parts should be 15 bits format.

For input data, because of the QPSK modulated OFDM signal property, all the values are in the range of -1 to 1. Therefore, keeping only one bit integer part is enough for processing in FFT processor. For other signal types, the value of input data can all be normalized into the range of -1 to 1 before processing.

For twiddle factor, because both the real and imaginary parts are also in the range of -1 to 1, the first bit of 15 bits is the sign bit and the rest 14 bits are used for fraction representation. Table 4.3 shows the range.

Table 4.3: The twiddle factor data format

Format	Binary	Decimal
General format	SXXXXXXXXXXXXXX	
Minimum value	100000000000000	-1.0
Maximum value	011111111111111	+0.9999999999999995
Minimum step size	000000000000001	+0.0000000000000005

4.3.3 Data scaling between each stages

There are two effects that require data scaling between each stage. The first one is the overflow and saturation effect. For R2^2SDF structure, this effect can only happen in the addition and subtraction of butterfly arithmetic blocks. Because there are additions and subtractions at each stage, $\frac{1}{2}$ division is used for each stage in this FFT design. This implementation uses arithmetic right shift. There is a possible optimization for this scaling. Armstrong, et al [35] provides a method to improve the scaling effect. However, for ASIC implementation, the data bus connecting between each stages can be customly defined. Thus, the width of the data buses can be different for different stage connections. In case that the overflow accours at specific stage, an extra MSB (Most Significant Bit) bus line needs to be added. Therefore, an optimal solution exists for data bus design in order to optimize the ASIC implementation and avoid overflow. Extra efforts need to be focused on the data value analysis to find the optimal solution in future.

The other effect is the rounding effect caused by multiplication. The multiplication of two 15 bits data will produce 30 bits product, which can not be passed to the next stage. So the trancating must be used to fit for the transmit bus and system requirement. This design truncates the 15 left most bits directly after the multiplication, which is another factor to cause quantization errors.

4.3.4 Twiddle factors design

There are two ways for twiddle factors design. One way is using ROM to store the fixed twiddle factors which were calculated in advance. The other way is generating twiddle factors by CORDIC (coordinate rotation digital computer). The CORDIC algorithm provides an efficient method of computing trigonometric functions by rotating a vector through some angle, specified by its coordinates [36]. Sarmiento, et al [36] and Despain, et al [33] employ CORDIC to calculate the twiddle factor multiplication for FFT processor. Zhang, et al [14] uses CORDIC to generate twiddle factor, which is a good way to combine the multiplication with calculating the twiddle factor for FFT. However, it is difficult to implementation for R2^2SDF structure. The reason is that Radix 2^2 algorithm changes the order of the twiddle factors, which will change the CORDIC structure and require more circuit blocks. It is not reasonable to implement from the area and power consumption point of view.

The proposed design employs the first method to handle the twiddle factors. However, a low power ROM design is used specially for R2^2 SDF, which is discussed in Section 5.2.3.1.

4.3.5 The input order and bit reversed order for the output

Many designs for FFT processor ignore the consideration of the interface with other baseband UWB system blocks. However, the analysis of this part is imperative for processor design. The reason is that the Input/Output (I/O) interface handling may take much area and power consumption for which ASIC chip is not affordable in some circumstances. And the choice of architecture is strongly affected by the interface issues.

For high performance real time processing, the input data are streaming into the processor. This means, for normal FFT processor, a serial to parallel data buffer needs to be installed at the input of the FFT processor. However, the pipeline structure FFT is very suitable for this data input type. Particularly, R2^2SDF structure artfully integrates input data buffer with first stage output data buffer. Nevertheless, this means the processor needs to work at the data sampling frequency, which is an obstacle for implementation. This problem is solved in Section 5.1.

Similar as normal DIF algorithm, the output of R2^{2}SDF structure is in bit reversed order. Therefore, extra registers are required for reordering the outputs into normal order. However, the problem can be solved if the channel estimation and equalization blocks, which is connected with the output of FFT processor, can be designed to process the input data in bit reversed order [9]. In this way, the output of FFT will serve as the input to the channel decoding stage. Actually, the following blocks after FFT are the deinterleaver and viterbi decoder. The complete solution to this problem should include these blocks and interconnections design consideration. Some effects have already been done for the bit reversed order output problem [13] [37]. However, deeper analysis of the baseband processor is necessary for this issue, which is beyond the thesis scope and needs further research support.

4.3.6 IFFT realization

IFFT (Inverse Fast Fourier Transform) calculation can be computed in FFT processor by just adding some small circuits [15].

IDFT (Inverse Discrete Fourier Transform) is presented in Equation (4.5).

$$X(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)W_N^{-nk}, \qquad n = 0, 1, ..., N-1 \tag{4.5}$$

By taking complex conjugate and multiplying N, Equation (4.5) can be written as

$$NX^*(n) = \sum_{k=0}^{N-1} X^*(k)W_N^{nk} \tag{4.6}$$

Changing $X^*(n)$ back to $X(n)$, the equation can be written

$$X(n) = (\frac{1}{N}\sum_{k=0}^{N-1} X^*(k)W_N^{nk})^* \tag{4.7}$$

From Equation (4.7), it is shown that two extra circuit blocks are required for IFFT calculation. The first one is to change the input data into its complex conjugate. The second one is to change the output of FFT processor into its complex conjugate. In this way, there is no requirement for changing twiddle factors from W_N^{nk} to W_N^{-nk} in FFT processor.

4.4 Summary

This chapter analyzes and compares different FFT processor structures from the architecture level. It is found that R2^{2}SDF structure is the optimum option for ASIC FFT implementation of MB-OFDM UWB system with regard of the memories and arithmetic blocks utilization.

Important design choices and considerations are also discussed and analyzed here. The data format and scaling, twiddle factors design, and the interface handling methods are all analyzed and chosen in regard of the system requirement. However, the frequency requirement problem is not yet fulfilled, which is solved in the next chapter.

5 Improvement and Implementation

From the analysis of previous chapter, it is shown that R2^2SDF requires 528 MHz clock frequency to process data. However, as explained in Section 4.2, this is unrealizable for baseband UWB processor. Here, a novel parallel-pipeline FFT processor structure is proposed based on the R2^2SDF architecture. The clock frequency of the proposed structure can be reduced to 157MHz, without much demand for chip area. In this way, the power consumption is dramatically reduced.

5.1 Improvement

To reduce the clock frequency while keeping the throughput, the most possible way is to parallelize the butterflies. However, parallelism will increase the area and power consumption. Therefore, a balance is required between the level of parallelism and the area and power consumption concern. After analysis, for pipeline structure, the first stage is the most clock-cycle-consuming stage, which takes half of the total latency. Thus, the first attempt is to parallel the butterflies only for the first stage. Nevertheless, by doing this, the data will overstock at the beginning of the second stage.

It is found from analysis, employing two-path parallelism in first six stages is proper for this structure. Because these six stages can process the even and odd input data separately and the last stage need to mix the even and odd data. The design details are described in Section 5.1.2. The shift registers and multipliers are used to compare between structures because they take most of the area of the processor. In the proposed design, the shift registers are the same as required by R2^2SDF. The number of the multipliers is changed from 3 to 5. Therefore, only 2 extra multipliers are required for this design, while the clock frequency reduced to half of the original one. If traditional input buffers are used, the clock frequency can be reduced to 157MHz.

In the following part, the improvement is described from the algorithm, architecture and implementation perspective respectively.

5.1.1 Algorithm Level

From the analysis of the radix 2^2 algorithm, it is found that the input data can also be separated to the odd and even parts. These odd and even parts are not mixed until the last stage. Here, 8 point FFT data flow is used to illustrate the idea, as shown in Figure

5.1.

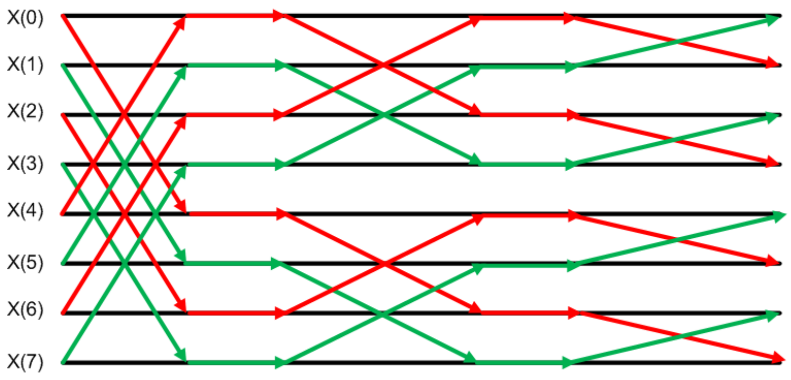


Figure 5.1: Radix 2^2 based parallel FFT algorithm data flow

The red line stands for the even input data flow while the green line stands for the odd input data flow. For the first and second stage, there is no cross between the right lines and green lines, which means the even and odd input data can be separately processed in the first and second stages. Only in the last stage, the red lines and green lines are crossed which means that the even and odd data should be mixed to process. It can also be seen that the twiddle factors are not changed by this parallel algorithm compared with Radix 2^2 algorithm. Thus the number of multiplication required is the same. However, due to parallelism, the position and order of the twiddle factor need to be changed. The 128 point parallel Radix 2^2 based algorithm data flow with twiddle factor position is shown in Figure 5.4 and Figure 5.5.

By analyzing the Figure 5.4 and Figure 5.5, the input data are separated into the even and odd data and processed. The twiddle factors of radix 2^2 algorithm are also separated into the even and odd parts in each stage and assigned to the even and odd data flow. The outputs of this algorithm are also changed and seem not in bit reversed order. However, in real time processing, the first output of the blue arrow lines will be produced at the same time as the first output of the red arrow lines as shown in the figures. If all the outputs are saved in this order, They will be in bit reversed order. The detailed structure information is described in the following parts.

Figure 5.2 is the Matlab simulation results of this parallel algorithm. The upper figure is the outputs comparison between this algorithm with the Matlab builtin one. It can be seen that the two output waves are matched. There are only on the order of 10^{-16} differences as shown in the lower figure, which is caused by the twiddle factors combination in radix 2^2 algorithm. Figure 5.3 is the 15 bits fixed point simulation results. The SQNR is above 40dB, which is satisfied the system requirement. The increase of the error compared with floating points, is caused by the quantization in the arithmetic block of FFT processor. The fixed point Matlab code is in Appendix D.

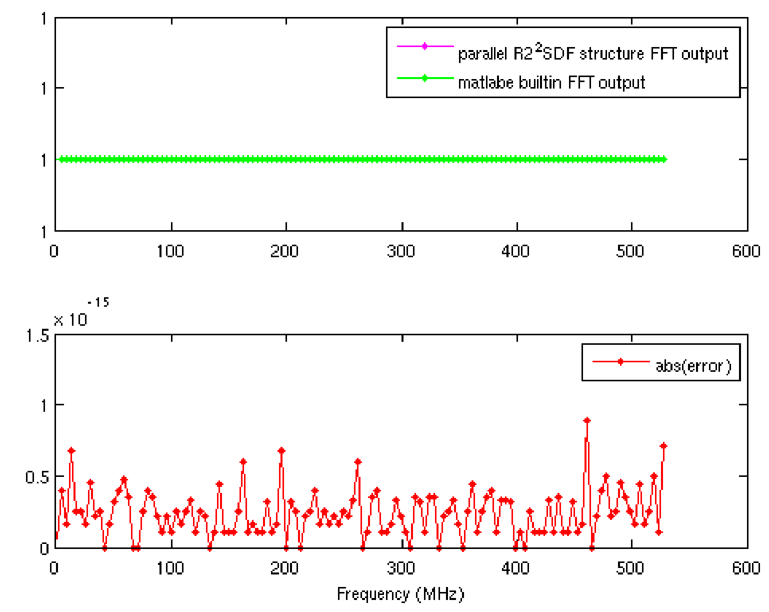


Figure 5.2: The comparison of parallel algorithm with Matlab builtin FFT

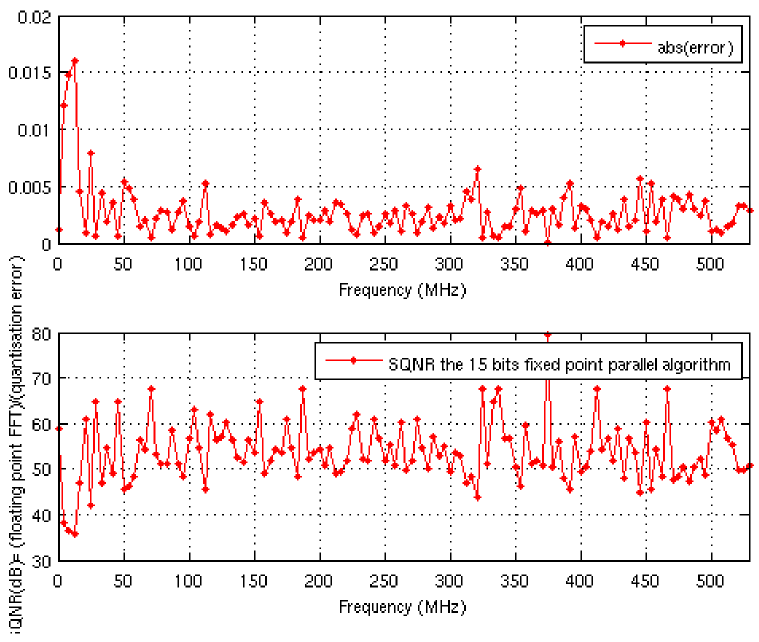


Figure 5.3: The error of 15 bits parallel radix 2^2 algorithm

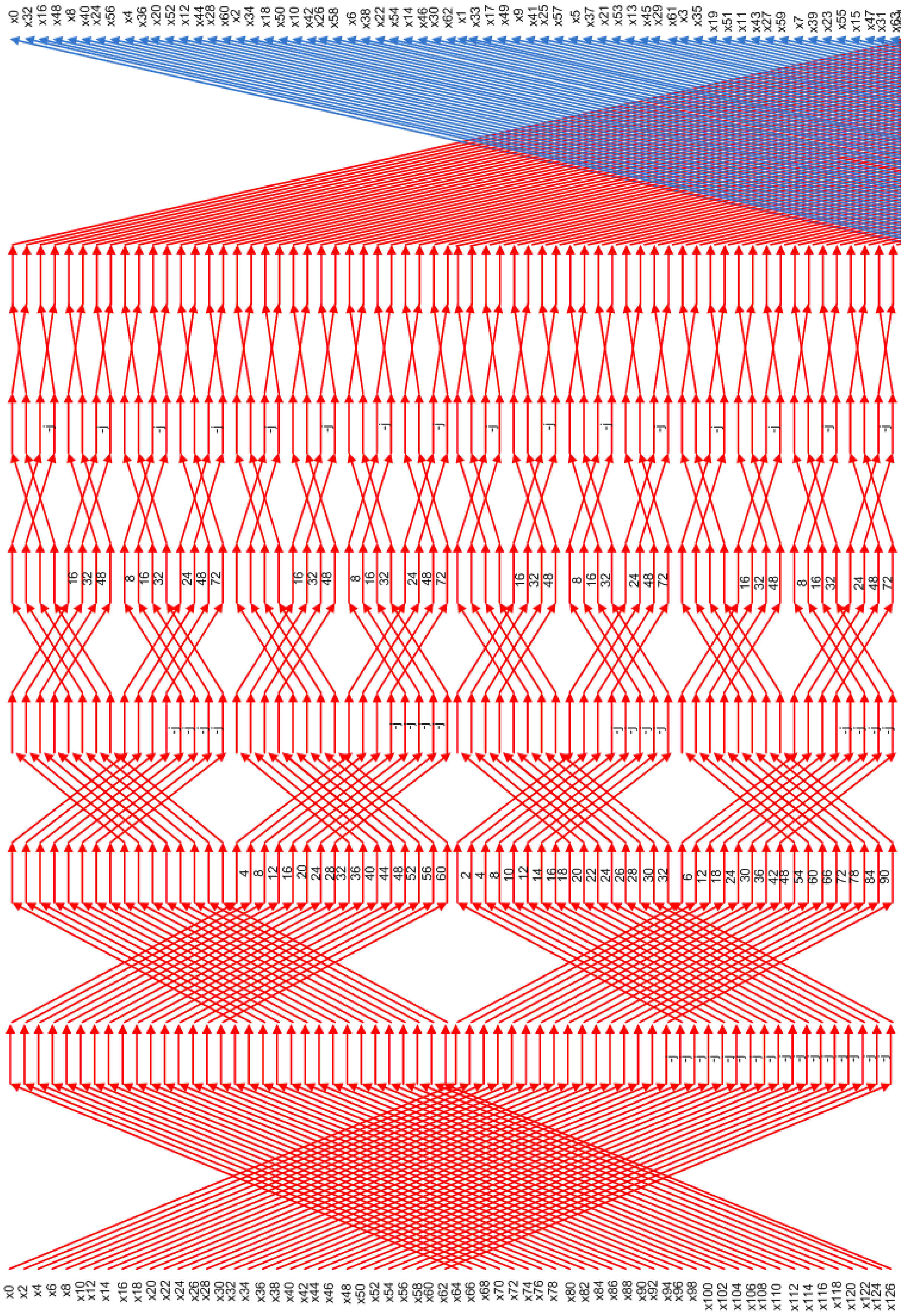


Figure 5.4: The 128 point parallel radix 2^2 based algorithm data flow first part

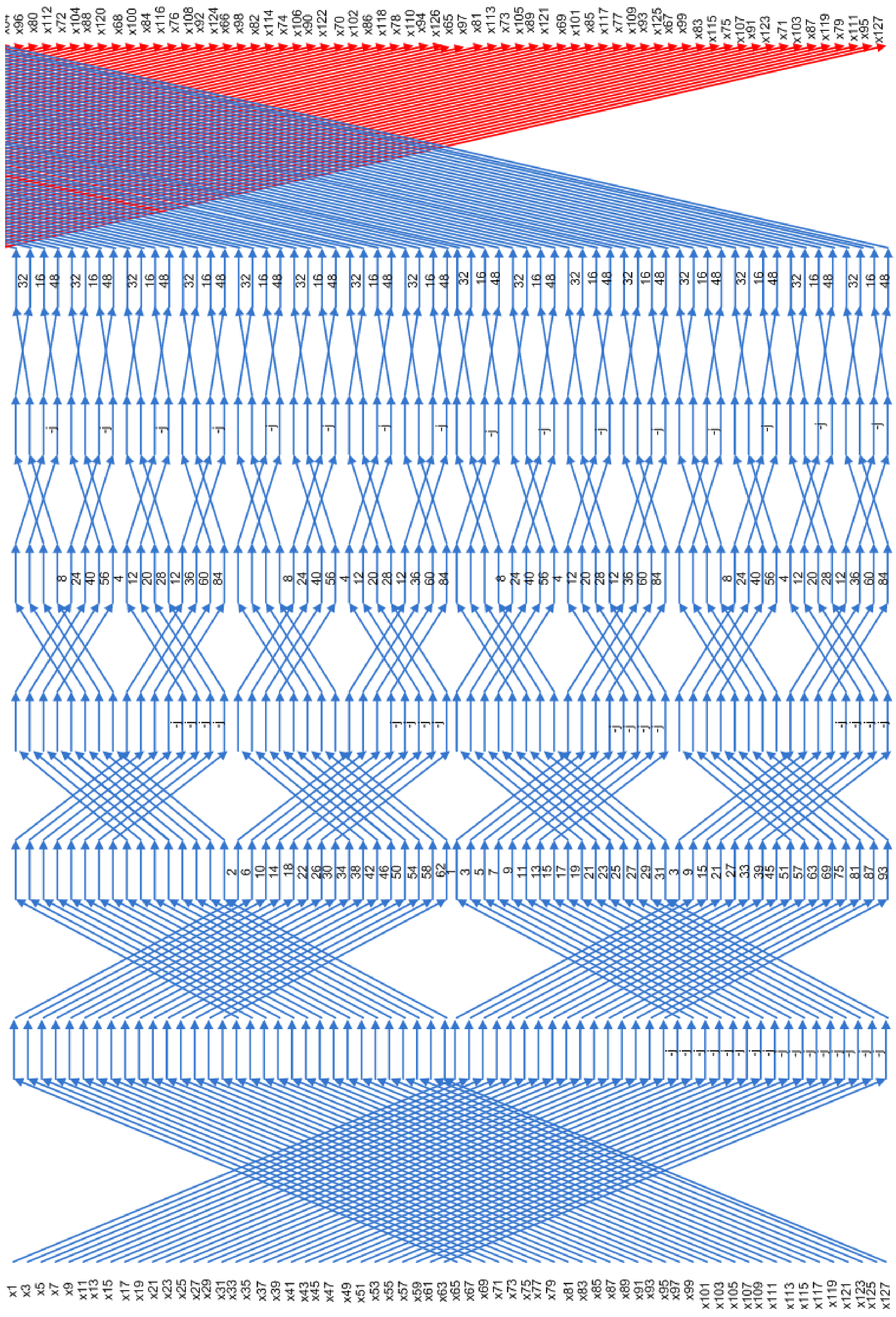

x96 x80 x112 x72 x104 x88 x120 x68 x100 x84 x116 x76 x108 x92 x124 x66 x98 x82 x114 x74 x106 x90 x122 x70 x102 x86 x118 x78 x110 x94 x126 x65 x97 x81 x113 x73 x105 x89 x121 x69 x101 x85 x117 x77 x109 x93 x125 x67 x99 x83 x115 x75 x107 x91 x123 x71 x103 x87 x119 x79 x111 x95 x127
32 16 48
-j
8 24 40 56 4 12 20 28 12 36 60 84
2 6 10 14 18 22 26 30 34 38 42 46 50 54 58 62 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 3 9 15 21 27 33 39 45 51 57 63 69 75 81 87 93
x1 x3 x5 x7 x9 x11 x13 x15 x17 x19 x21 x23 x25 x27 x29 x31 x33 x35 x37 x39 x41 x43 x45 x47 x49 x51 x53 x55 x57 x59 x61 x63 x65 x67 x69 x71 x73 x75 x77 x79 x81 x83 x85 x87 x89 x91 x93 x95 x97 x99 x101 x103 x105 x107 x109 x111 x113 x115 x117 x119 x121 x123 x125 x127

Figure 5.5: The 128 point parallel radix 2^2 based algorithm data flow second part

5.1.2 Architecture Level

From the previous analysis, there are some requirements for the architecture design. First, the input data need to be separated into the even and odd parts. Then two parallel data paths are controlled by the same counters in order to be processed synchronously. The input data of the last stage need to be switched and well controlled to send to butterfly and the last stage needs to be well organized to receive the input data and generate the output.

Figure 5.6 shows the architecture of the proposed radix2^2parallel based algorithm. It consists of multiplexers, circular buffers, ROM, complex multipliers, and butterfly units. First, the input data are streamed in and handled by multiplexer. The multiplexer is controlled by the control signal, which can separate the input data into even and odd parts. These data are processed in the even and odd parts of the architecture, where the red arrow lines stand for the data flow of even data and the blue lines stand for the odd data. There are three counters which produce the control signals and the addresses for reading the twiddle factor from the ROM. The even and odd parts of the architecture are all controlled by the same control signals. Therefore, the synchronization property is fulfilled for the whole structure, which is also important for the last stage processing. There are five complex multiplications in the architecture. In the sixth stage, the even part outputs do not need multiplication and twiddle factor storage, which can be found in Figure 5.6. The reason is that, after twiddle factor separation in this stage, all the twiddle factors in the even part become constant 1. Therefore, no multiplication is required.

As the same as Radix2^2SDF structure, there are seven stages, which are shown in Figure 5.6. For the parallel architecture, the whole structure can also be divided into the three big blocks, the last block, and the arithmetic blocks. The counters, which look like the brain of this structure, are the core of the big blocks. The first big block includes the first and second stages of the even and odd parts, and the first counter. The second big block includes the third and fourth stage of the even and odd parts, and the second counter. The third big block includes the fifth and sixth stage of the even and odd parts, and the second counter. The details are described in Section 5.2.1.3. The arithmetic blocks are composed of five ROMs and complex multipliers, which are described in Section 5.2.3.

The last block only includes the seventh stage. Because the odd and even data need to be commutated, two multiplexers are required to switch the data, which is shown in the Figure 5.6. However, by analyzing the scheduling of the last stage, it can be found that the first output data of the even part will be processed with the first output of the odd part. As long as the timing is matched, the even outputs will be processed with the odd ones correspondingly. Therefore, the two multiplexers are not necessary and only one butterfly in the last stage is required to process the data. The modified structure of the last stage and interface with previous stage is shown in Figure 5.7.

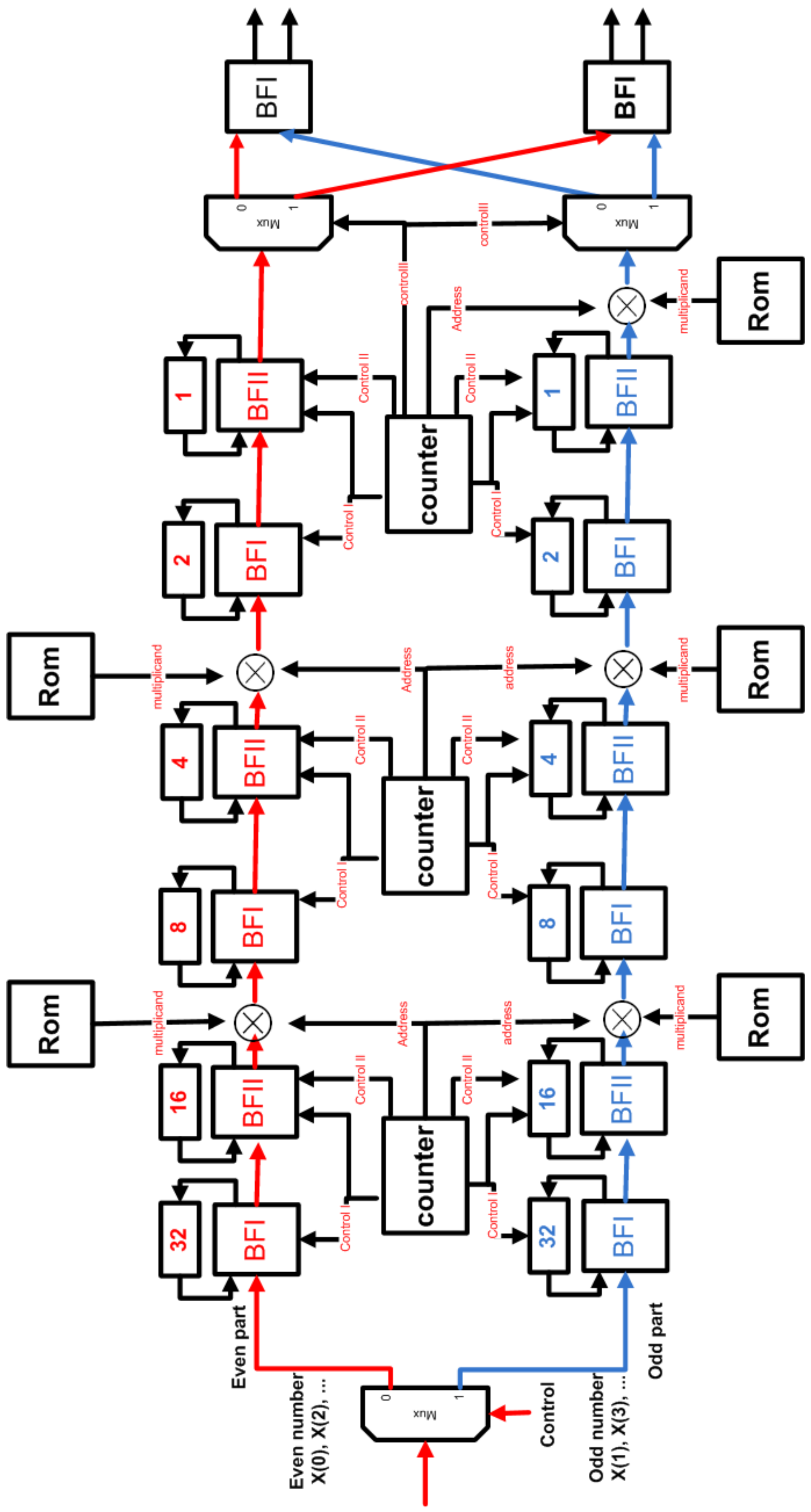


Figure 5.6: The parallel radix 2^2 based pipeline architecture

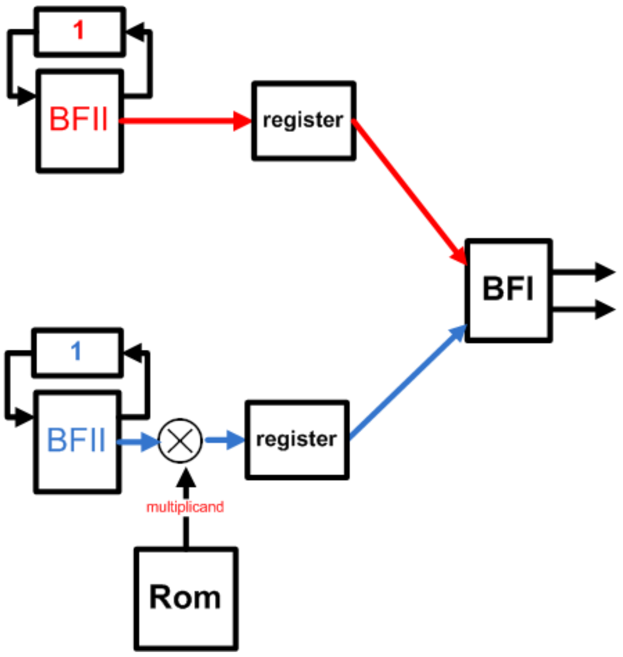


Figure 5.7: The change of last stage

5.1.3 Implementation Level

For implementation level, several aspects need to be considered. Firstly, the control signals need to be assigned in order to work synchronously. These control signals are produced from the counter. In the first draft of the design, seven counters are required for seven stages. However, this method is not area and power efficient. The control signals of the first and second stage and the addresses for the ROM storage can be provided by one counter. This method can be expanded to the following stages, so the number of the counters is reduced to 3. The synchronization between counters is also needed to be well designed to let the whole circuit work synchronously.

Secondly, the timing reconstruction is required to let the processor work at proper frequency. Due to the structure, extra registers are needed at the outputs of each stage and complex multiplier. At the same time, the timing and control signals should be adjusted because of the latency caused by the extra registers.

Thirdly, the problems caused by parallelism need to be handled. The steaming input data are separated by the multiplexer. The even parallel data need to be processed with the odd ones synchronously. And at the last stage, data should be well mixed to be sent to the last stage.

The detailed implementation information is described in the next section.

5.2 Implementation

5.2.1 Radix 2^2 SDF Pipeline Data Flow

The SDF pipeline data flow is described from the basic elements to the whole structure, which is composed of the three big blocks and the final block. There are two stages and a control counter in the big block. The butterflies and memory block are included in the stages. First the butterfly structure is introduced.

5.2.1.1 Butterfly

Two type of butterflies are used in this structure, which are butterfly I and butterfly II.

The design of butterfly I
For the Butterfly I, it consists of four 2-to-1 multiplexers and four adders. The schematic is shown in Figure 5.8.

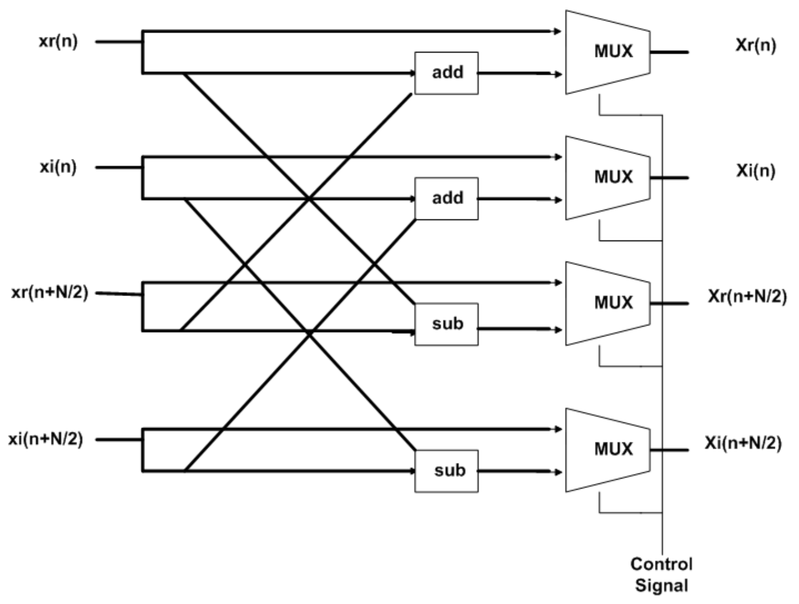


Figure 5.8: Radix 2^2 Butterfly Type I

As shown in the figure, the input data are complex data $x(n)$ and $x(\frac{N}{2}+n)$ which is $\frac{N}{2}$ data later than the $x(n)$. The data path is controlled by the multiplexer. When the control signal is one, the data will go though the path which does the calculation of addition or subtraction. Otherwise, when the control signal is zero, $x(n)$ and $x(\frac{N}{2}+n)$ will directly pass butterfly I and get out.

The design of butterfly II
Besides the four 2×1 multiplexers and four adders, Butterfly II also consists real and image parts switching and inversing because of the $-j$ multiplication which is required

by the Radix 2^2 algorithm. The schematic is shown in Figure 5.9.

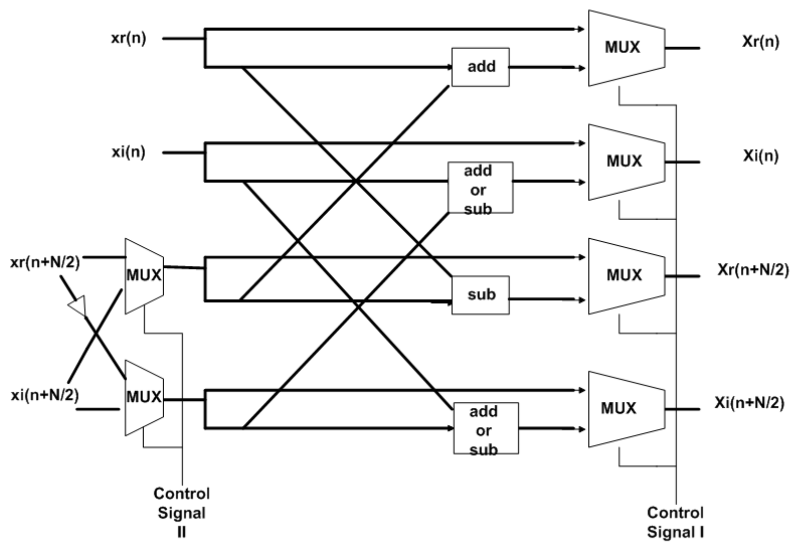


Figure 5.9: Radix 2^2 Butterfly Type II

Compared with Butterfly I, another control signal is introduced to the Butterfly II. this signal control the real and imaginary parts switching of the input data $x(\frac{N}{2}+n)$, and the add or subtract switching function is also controlled by this signal which is not shown in the figure. When control signal II is zero, this butterfly acts the same as Butterfly I, which means the data flow only depends on control signal I. When control signal II is one, $-j$ should multiply $x(\frac{N}{2}+n)$. $-j \times (x+jy)$ equals $y+j(-x)$. Because of the 2' complement data presentation, to handle the addition of $(-x)$, the x need to be inversed and the carry-in of the adder is changed to one. Considering the whole function, first, the real part of $x(\frac{N}{2}+n)$ should be inversed. Second, the real and imaginary part should be commuted, which means the real part goes through the imaginary data path and vise verse. Third, the second adder and subtractor in the figure should be changed to the subtractor and adder, which means that the carry-in of the adder needs to be changed.

Control signal Design
The two control signals mentioned above come from the synchronizing counter. Control signal I controls the outputs. In the first $\frac{N}{2}$ cycles, this control signal is zero, which means the two data pass to the shift register and get out. In the second $\frac{N}{2}$ cycles, the control signal is one, which means the two input data do the addition and subtraction to get the output value. The control signal II controls the input data. It controls the $-j$ multiplication with the input data at some periods. To easily show how this multiplication works, Figure 3.9 is called back here as Figure 5.10.

As shown in the figure, the $-j$ multiplication only happens when the last $\frac{N}{4}$ output data come out of the former stage. So this control signal can be the same as the control signal

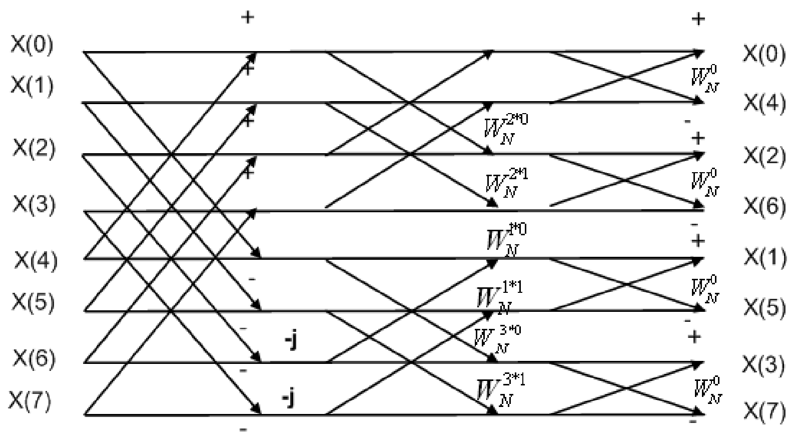


Figure 5.10: The application of Radix 2^2 algorithm for 8 point FFT

I of the first stage, which can separate the two second stages as shown in the middle of the figure 5.10. Table 5.1 shown the different action based on the two different control signals for Butterfly II.

Table 5.1: Butterfly Control Signal Truth Value

		Control Signal I	
		0	**1**
Control signal II	**0**	**input signals direct go through to the output**	**$-j$ multiplication with butterfly calculation**
	1	**input signals direct go through to the output**	**normal butterfly calculation**

5.2.1.2 Stage

In the big block, two stages and a counter are included. Here the two stages are introduced. Each stage is composed of a memory block and a butterfly. First, the memory block is introduced.

Shift Register
Based on the circuit requirement, the first input data need to wait $\frac{N}{2}$ cycles in order to be processed together with the $\frac{N}{2}+1$ data. Also, for the outputs of the butterfly, one of them needs to wait $\frac{N}{2}$ cycles in order to be processed by the next stage. Normally, to fulfill these functions, two shift registers are needed, one at the beginning and the other at the output as shown in the Figure 5.11.

These shift registers are controlled by the clock and synchronizing counter and follow the rule of first in first out (FIFO). When clock signal became 1, they read one data in

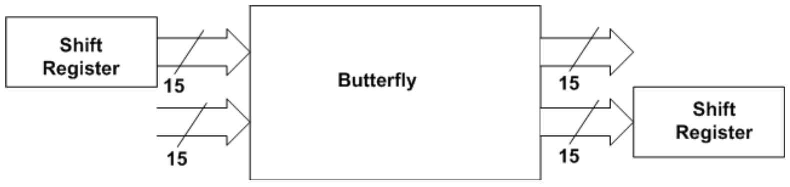


Figure 5.11: Shift Registers for Butterfly

and send one data out.

Inside the shift register, it is separated into two parts because of the complex data type. the upper part shifts the real part of the data, and the lower part shifts the image part of the data. Figure 5.12 shows the structure.

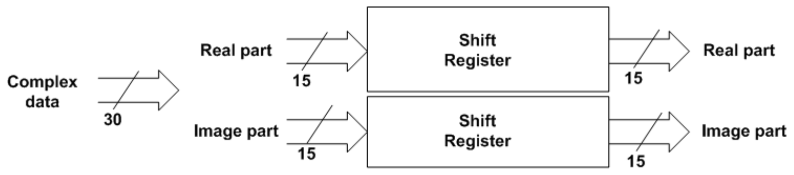


Figure 5.12: The structure of shift register

However, due to the SDF, the input shift register and output shift register are combined together as one shift register. This combination based on that the input and output data flows are perfect matched in time sequence.

Circular Buffer
Circular buffer is similar with shift register. However, from the power consumption point of view, circular buffer performs better because of the smaller number of flip flop switches during one clock cycles.

Same as shift register, circular buffers are also controlled by the clock and synchronizing counter and follow the rule of first in first out (FIFO). However, instead of all the data shifting at the up edge of the clock, only one data are read in and one data are read out. There are two pointers shifting around the buffers. One points the address where the data are read in. The other points the address where the data are read out. At each up edge of the clock, the pointers will shift one data right. When these pointers arrived at the most right point, they will start again from the beginning of the most left point. The reading pointer points one data right compared with the writing pointer. Both of them are controlled by the counter signals. Figure 5.13 shows the structure.

In the following part, shift registers are still used just for easy explanation. For real implementation, the circular buffers are used instead of shift registers.

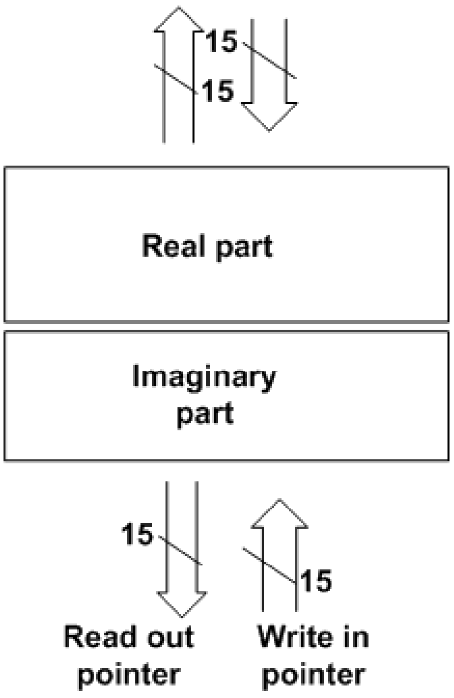


Figure 5.13: Circular Buffer

Stage 1 and Stage 2
Stage 1 is composed of the shift register and butterfly I. One of the outputs of butterfly I connects the input of shift register. The output of the shift register is connected to one input of the butterfly. Figure 5.14 shows the connection.

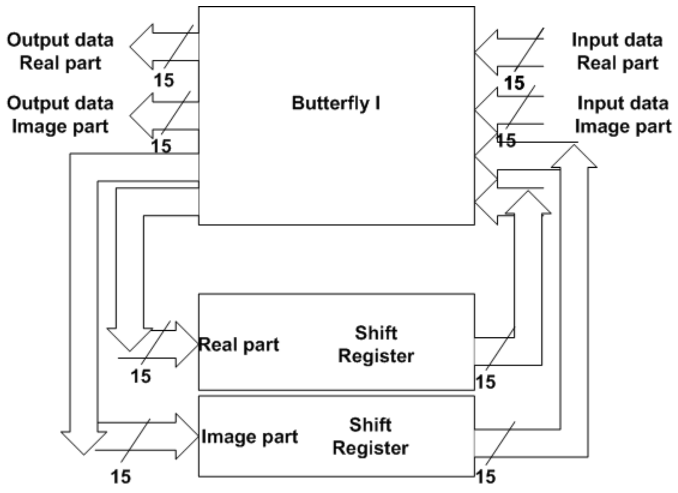


Figure 5.14: The first stage of radix 2^2 FFT processor

Stage 2 is similar to stage 1. The difference is that the butterfly I should be replaced with the butterfly II. The data flow of two stages is introduced here. First, the data come in the stage and go through the butterfly and are directly read in the shift register. After $\frac{N}{2}$ cycles, the first data come out from shift register and are processed with the new input data. One of the processed data go out of the stage, and the other go back to the shift register. Again after $\frac{N}{2}$ cycles, the processed data will come out of the shift register and go out of the stage. At the same time, new data are read in to the shift register and a new cycle process begins.

5.2.1.3 Block

As said above, for 128 point radix 2^2 structure, the total circuit can be separated into 7 stages. The first six stages can compose three big blocks and one stage at the end as the fourth block.

The big block
This block is composed of the counter, the two stages, and some registers in order to delay the signals and data. The main components are focused.

The counter
Only one counter is used to produce the control signals for both stage 1 and stage 2. The control signal waves are shown in the Figure 5.15.

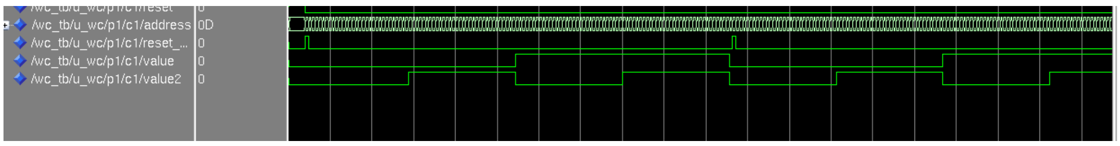

/wc_tb/u_wc/p1/c1/reset 0
/wc_tb/u_wc/p1/c1/address 0D
/wc_tb/u_wc/p1/c1/reset_... 0
/wc_tb/u_wc/p1/c1/value 0
/wc_tb/u_wc/p1/c1/value2 0

Figure 5.15: The counter control wave

As shown in the figure, there are two control signals which come from the output of the counter. The upper one is control signal I for stage 1 which controls butterfly I. The lower one is control signal I of the butterfly II in stage 2. At the same time, the upper signal as control signal II controls butterfly II. Besides producing the control signal, the counter will also generate the reset signal of next block, which is determined by the number of registers added between two blocks. The ROM address of the twiddle factors is also produced here.

The structure of the big blocks is shown in Figure 5.16.

The basic idea of the data flow of this block is that the first stage repeats after $\frac{N}{r}$ data, and the second stage repeats after $\frac{N}{2r}$ data, where r depends on which block it belongs to. First, control signal I is set to zero to let the $\frac{N}{2r}$ data be read into the first stage, and in the following $\frac{N}{2r}$ cycles, control signal I is set to one to enable the butterfly function.

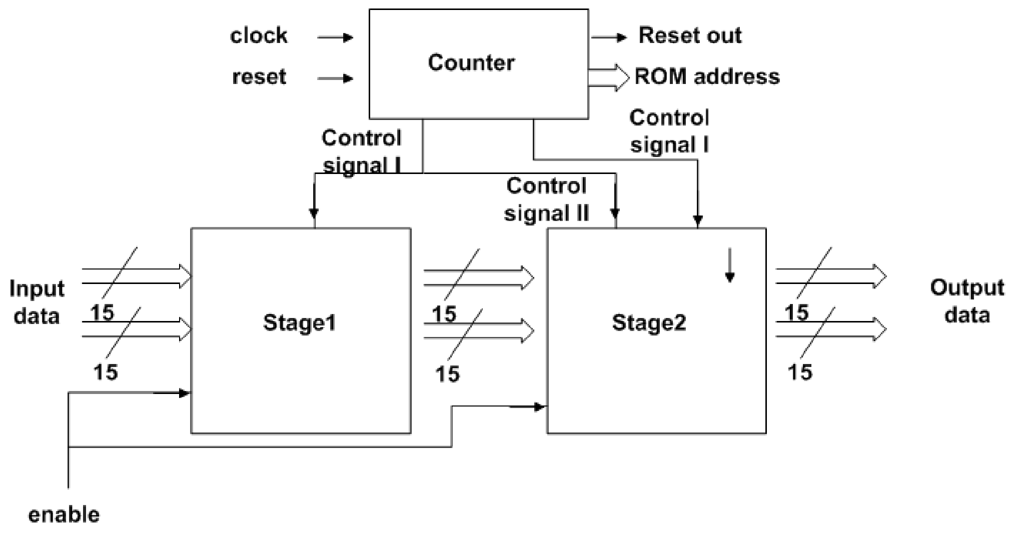


Figure 5.16: The big block structure

At the same time, the stage 2 read in the $\frac{N}{4r}$ data outputs of the stage 1 controlled by control signal II. The next $\frac{N}{4r}$ cycles, butterfly II of stage2 works and control signal II equals one. The data flow analysis is shown in the Figure 5.17.

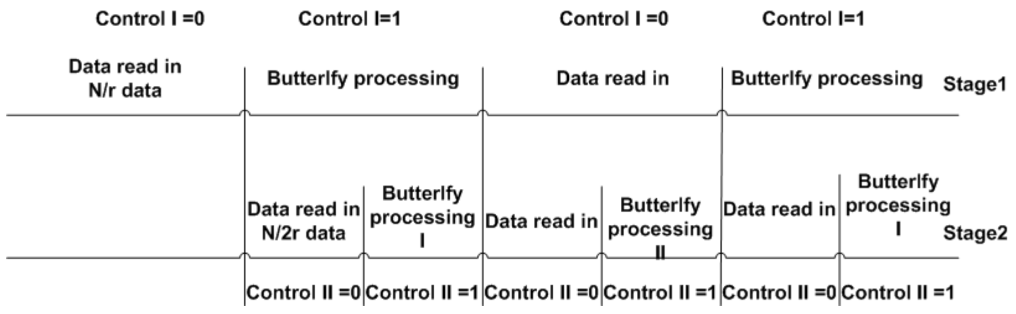


Figure 5.17: The data flow of the block

The fourth block
This fourth block is only composed of the counter, the registers and butterfly I. The counter of this block is the same as clock divider. Because the control signal, which is shown in Figure 5.18, changes after one clock cycle. This signal controls the butterfly I data flow. One output of butterfly I goes to the FFT output, and the other output waits in the registers after one clock cycle and then gets out of the processor.

The synchronization between counters
The whole structure is controlled by different counters. To make these counters working synchronously, reset signals are used between counters. Because there is latency caused by registers and arithmetic blocks among blocks, the same reset signal can not be used.

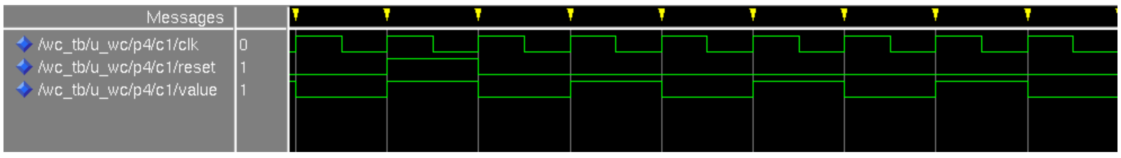


Figure 5.18: counter signals of the fourth block

The counter of former block is required to generate the reset signal for the following block. The difference of reset signals is determined by the latency between the two blocks.

5.2.2 Radix2^2Parallel Data Flow

Compared with Radix 2^2 SDF pipeline structure, some blocks are required to change or add for the structure. The additional blocks and changes are introduced in the order of the actual data flow.

5.2.2.1 The input multiplexer

Because the input data are streamed in, they need to be separated into the even and odd parts to be processed separately. The clock signal is used as the control signal of this multiplexer. In this way, the clock period of following parts is as long as twice of pervious clock period. All the even and odd outputs are stored in registers in order to synchronously processing in the following stages. The structure is shown in Figure 5.19. If there is an input memory block served, this multiplexer block is not needed and can be removed because the even and odd data can be directly read from the memory.

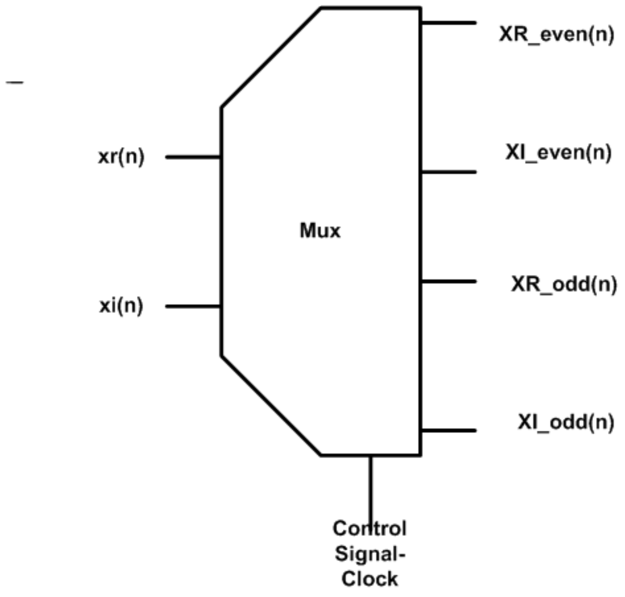


Figure 5.19: The input multiplexer

5.2.2.2 The parallel block

The three big blocks discussed in Section 5.2.1.3 are required to change to parallel ones. In this way, the counter in each block needs to control the four stages, as shown in Figure 5.20. The upper control signals are the same as the lower ones, which can guarantee the synchronization between the even and odd parts. And the arithmetic blocks are also needed to copy to the upper and lower parts. The addresses required by these arithmetic blocks are provided by the same counter.

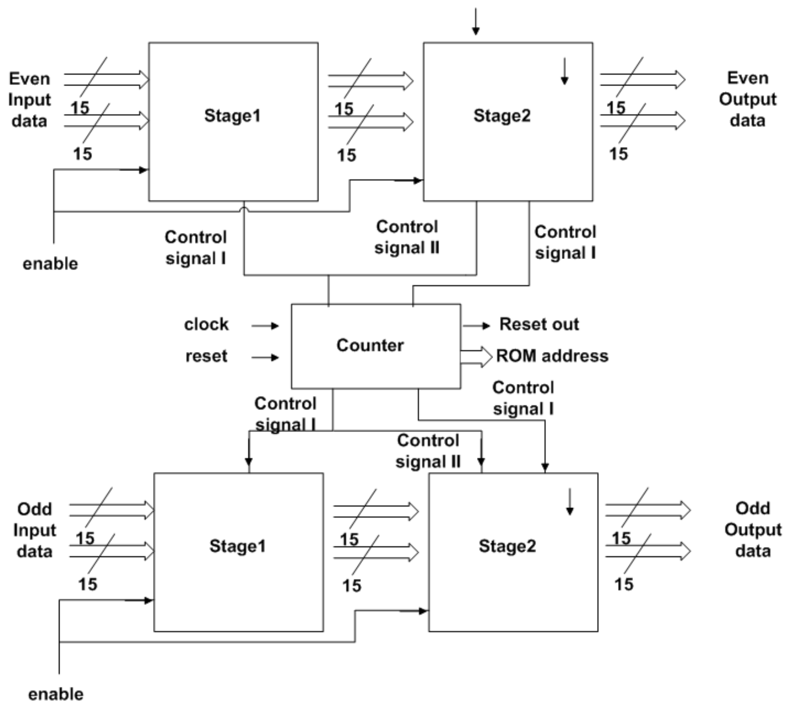


Figure 5.20: The parallel block

5.2.2.3 The last block

Different from the last block in R2^2SDF pipeline structure, the even part and the odd part are combined at this last block as shown in Figure 5.7. The data from the even part and the odd part are synchronously sent to the inputs of this last block. Inside the last block, butterfly I is used to process the data. The control signal of butterfly I is always one, which means all the inputs are combined to add or subtract. all the outputs of the butterfly are sent to the store registers, which is connected to the output pins.

5.2.2.4 Input buffer

For many FFT processors, input buffers are used to reorder the input data and also to reduce the clock frequency [15] [13]. In fact, most of the pipeline FFT processors employ this method for the tight time constraint of UWB systems. If this traditional way is employed in this processor, there are 128 data extra memory blocks required. Thus the clock frequency can be reduced to 157 MHz. However the area and power consumption will be increased.

5.2.3 ROM and Arithmetic Block Design

The arithmetic related blocks are composed of the ROM, the adder and subtractor, and the multiplier.

5.2.3.1 ROM Design

Traditionally, all the twiddle factors for multiplication are stored in a single ROM block. In the proposed design, this ROM block is separated into three small parts because of the structure requirement. If only single ROM block is employed, an address generator is required to provide the address for this ROM. The synchronization problem with the former blocks needs to be handled. However, from the Figure 5.4 and Figure 5.5, it can be found that the twiddle factors can be separated into three parts. For example, in the first complex multiplication block, twiddle factors from W_N^4 to W_N^{60} are considered as the first part. Twiddle factors from W_N^2 to W_N^{32} are cataloged as the second part. Twiddle factors from W_N^6 to W_N^{90} are cataloged as the third part. In this way, the ROM can be separated into three parts. The advantage of this method is that the address of these ROM blocks can be generated from the counter of the previous block. Two control signals are generated from this counter to control these ROM blocks. In this way, the ROM can be switched off when they are not used.

Figure 5.21 shows the structure of the ROM. There are two control signals generated by the counter, which is the same as the control signals for stage1 and stage2 of the big block. These control signals decide which ROM produces the twiddle factors. The three ROM blocks share the same address from the counter. In this way, the whole ROM block work synchronously with the outputs of the previous block.

5.2.3.2 Addition and Subtraction

The simplest way to implement addition and subtraction is to use the addition sign '+' and subtraction sign '-' in VHDL. The synthesis tools automatically map '+' and '-' operators onto an appropriate adder to meet timing constraints while minimizing area [38]. However, it is better to choose the addition and subtraction structure individually based on the chip area and latency analysis.

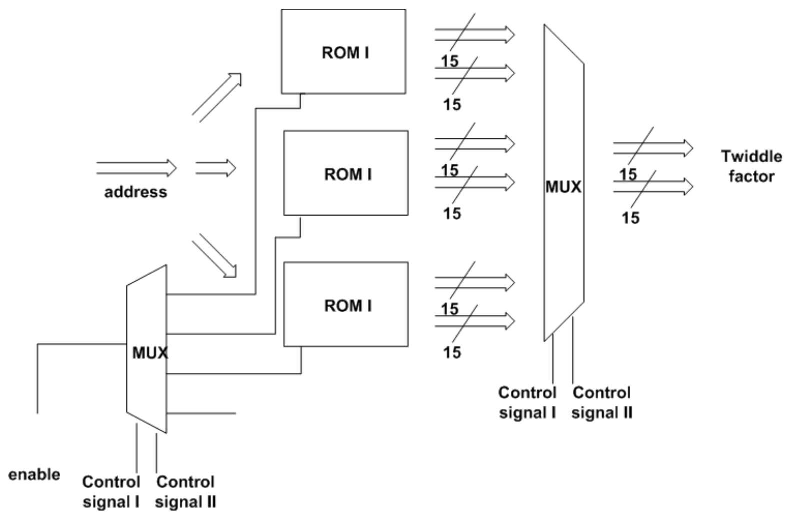


Figure 5.21: The ROM structure

The principle of adder [39]
Since subtractor has nearly the same hardware as adder, this part focuses on the principle of adder. The full adder block is shown in Figure 5.22. There are three inputs: Input I, Input II and Carry in, and two outputs: Sum and Carry out.

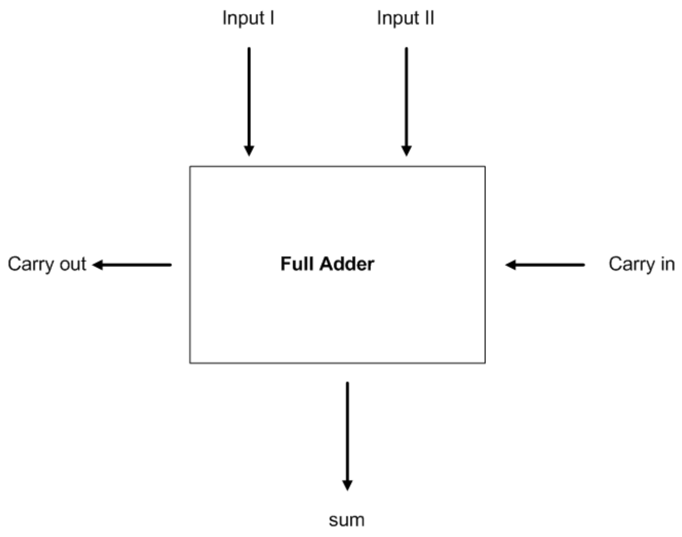


Figure 5.22: Full adder block

The simplest but slowest implementation of n bits adder is ripple carry adder, which requires n full adders. The carry out of each full adder is connected with the carry in of the next full adder.

The Comparison of different adders

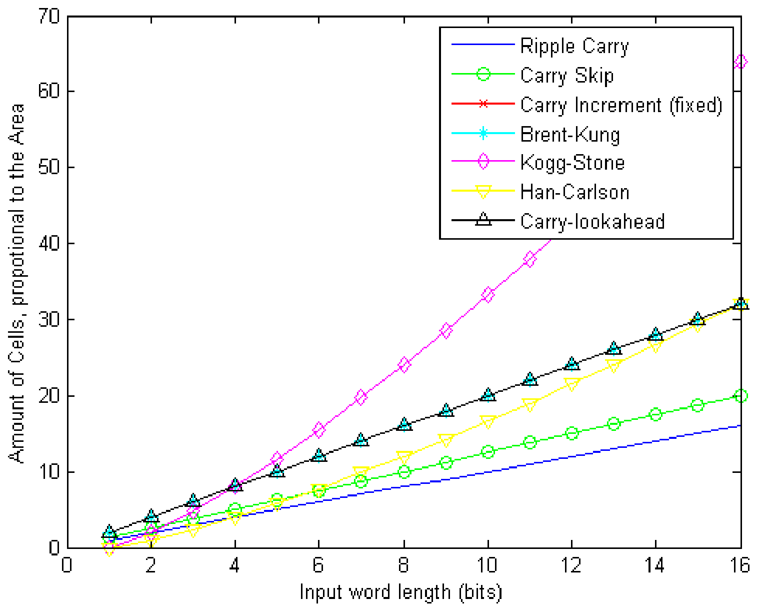


Figure 5.23: The area comparison of different adder structures

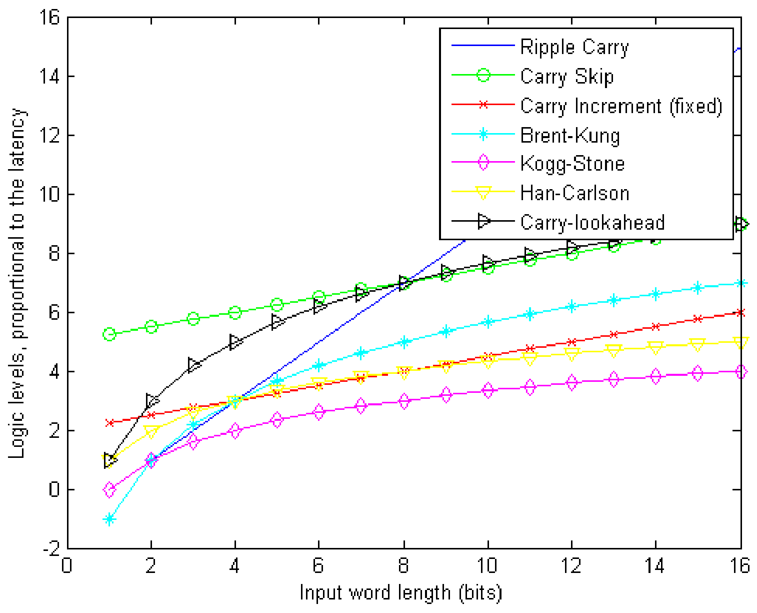


Figure 5.24: The latency comparison of different adder structure

Due to the trade-off between area and speed, it is necessary to compare different addition structures. The analysis of the area and speed performances of different adders based on book [38] and [39] are shown in Figure 5.23 and Figure 5.24. The speed analysis is based on the number of logic levels of the critical path and the area analysis is based on how many cells are required.

Because of the tight time constraints of this FFT processor, the optimal choice is to choose one of the fastest adders. As shown in Figure 5.23 and Figure 5.24, the Han-Carlson adder is chosen for this design. The reasons are that it is one of the fastest adders and its area is near the average level. The VHDL implementation is generated from [40].

Subtraction
The relationship of 2's complement subtraction and addition is

$$A - B = A + \bar{B} + 1 \tag{5.1}$$

So the subtraction can be implemented by slightly changes of adder. Figure 5.25 shows the changes. One operand is inverted and the carry input is set to one.

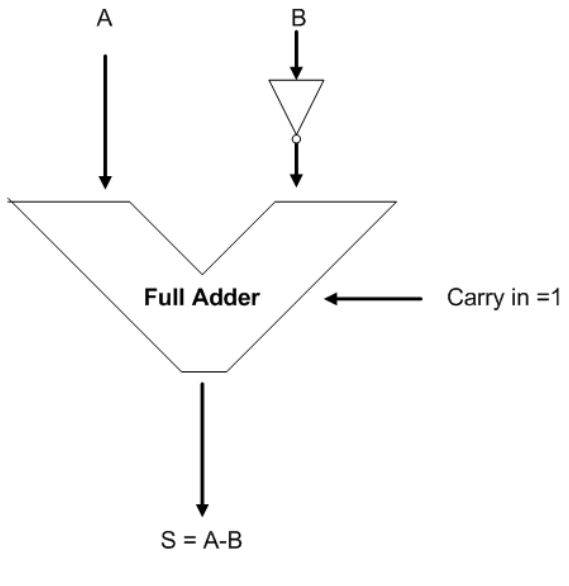


Figure 5.25: The subtraction block

However, in butterfly II of this processor, the switch between addition and subtraction is required. So the add/sub block is required. Figure 5.26 shows the structure of this block. Extra XOR gates are required and are controlled by the control signal of butterfly II.

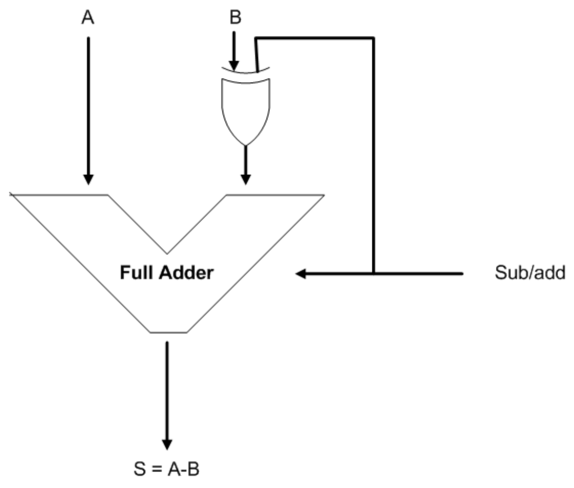


Figure 5.26: The add/sub block

5.2.3.3 Multiplication

The principle of multiplier [39][7]
The block of multiplier principle is shown in Figure 5.27. There are two inputs: multiplicand and multiplier and one output: product.

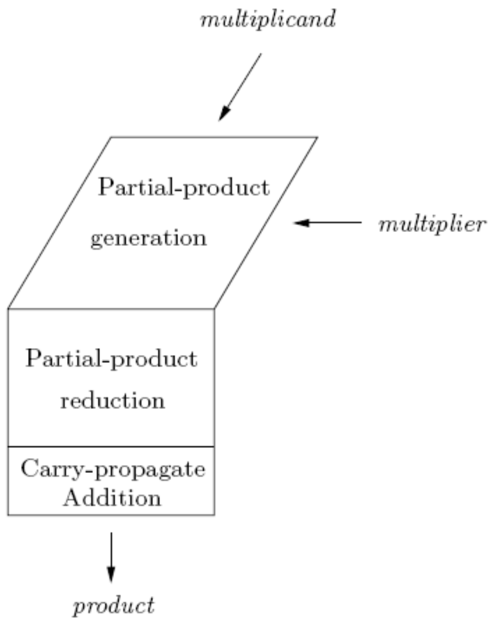


Figure 5.27: The principle of Multiplier [7]

The first stage is Partial Product Generation (PPG). In PPG, the partial products are generated from the multiplicand and multiplier. It is a $N * M$ matrix, where N is the number of digits in the multiplier and M is the number of digits in the multiplicand.

The second stage is partial Product Accumulator (PPA). PPA stage is actually a multi-operand adder to add groups of partial product bits. There are various hardware algorithms for multi-operand adder. The area and latency of multiplier are different based on different structures. The following part compares different partial product adding algorithms in order to choose the optimal one for this processor. The outputs of this stage are two rows of partial products.

The last stage is a two-operand adder to add the two rows of partial product. The output of this stage is the final product of the multiplier.

The comparison of different multipliers
Similar to addition, multiplication can employ '*' during VHDL implementation. The synthesis tools automatically map the operator onto an appropriate multiplier to meet timing constraints while minimizing area. However, because multiplier is one of the largest area-consuming blocks and its latency is probably the critical path of the whole processor, the selection is critical for the whole circuit design. Therefore, the comparison is necessary for the processor design.

The comparison steps are as follows. First, different multiplier implementations are generated from [40]. Five different type multipliers are generated, which are simple PPG array tree multiplier, simple PPG Wallace multiplier, simple PPG Dadda muliplier, Booth-Wallace multiplier, simple PPG (4 : 2) compressor multiplier, and simple PRG redundant binary addition multiplier. The details of these multiplier structures are shown in Table 5.2.

Table 5.2: The details of these multiplier structures

stage	array	Wallace	Dadda	Booth-Wallace	(4 : 2) compressor	redundant binary addition
PPG	simplePPG	simplePPG	simplePPG	booth recoding	simplePPG	simplePPG
PPA	array tree	wallace tree	Dadda tree	wallace tree	(4 : 2) compressor tree	redundant binary addition
Final stage adder	Brent-kung	Brent-kung	Brent-kung	Brent-kung	Brent-kung	Brent-kung

Each type of multipliers have four different bits types, which are 8 bits, 12 bits, 15 bits, and 18 bits. Each multiplier has been synthesized by Synopsys Design Compiler using

Faraday 90nm standard cell library, which is tailored for UMC' 90 nm logic LL-RVT (lowK) process. This library is also the library used for the FFT processor synthesis. Therefore, the simulation data is useful for this design. The results of area and latency are shown in Figure 5.28 and Figure 5.29.

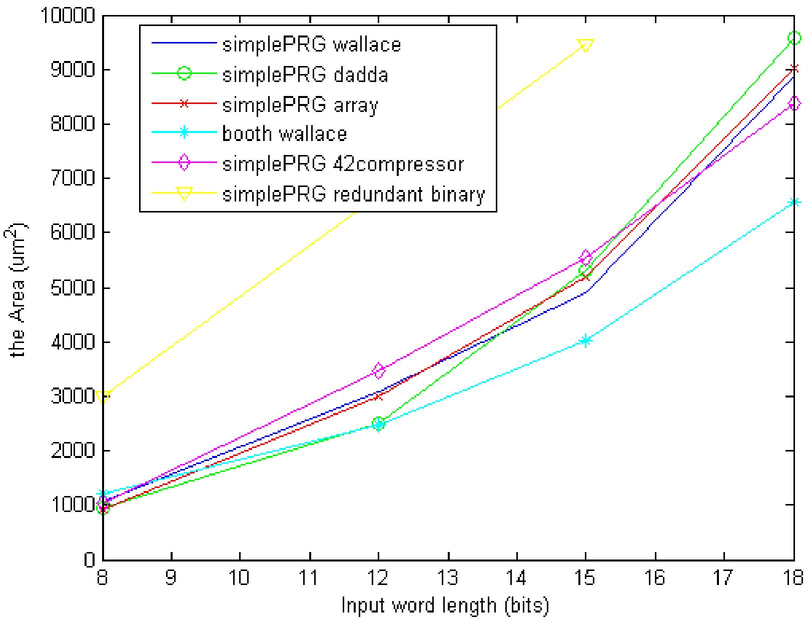


Figure 5.28: The area comparsion of different multipliers

From the figure, it is shown that Booth-Wallace multiplier is the optimal one among these multipliers at 15 bits point. Thus, it is chosen as the multiplier of this FFT processor. The latency of this 15 bits multiplier is 3.05 ns which is less than one clock cycle period of the processor. Therefore, there is no pipelining required for multiplier. The multiplier used by this processor is directly generated from [40].

However, if higher clock frequency is required for the processor, pipeline stages should be added into the multiplier, which is required for custom design. Another advantage of the custom-design multiplier is smaller area. Because truncating half output bits of multiplier is employed in the processor design based on the data bits requirement, some parts of PPA addition tree are not required, which will further reduce the area.

The complex multiplier

Because of the complex input data, the complex multiplier is required for processing. The formula of complex multiplication is

$$(A + jB) * (C + jD) = A * C + jA * D + jB * C - A * D \tag{5.2}$$

The complex multiplier is composed of four real multipliers and two adders. The

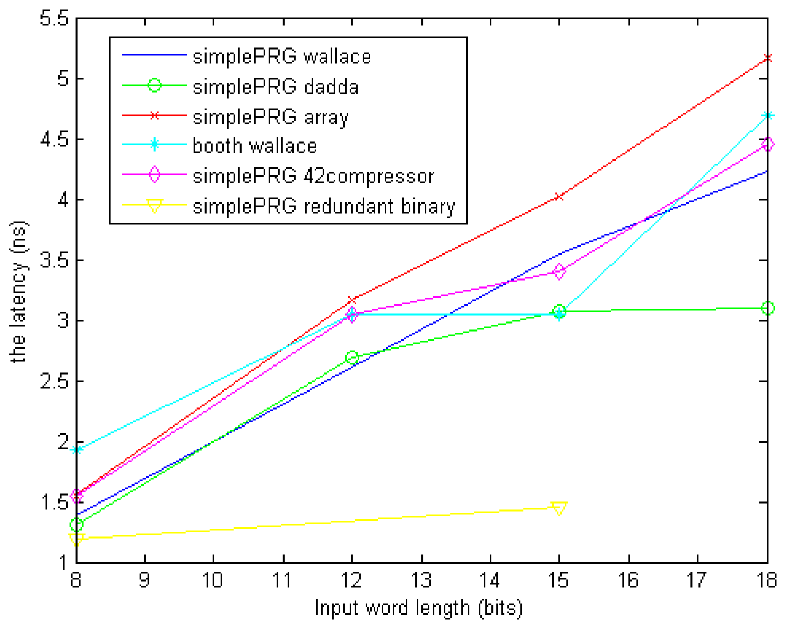


Figure 5.29: The speed comparsion of different multipliers

schematic is shown in Figure 5.30.

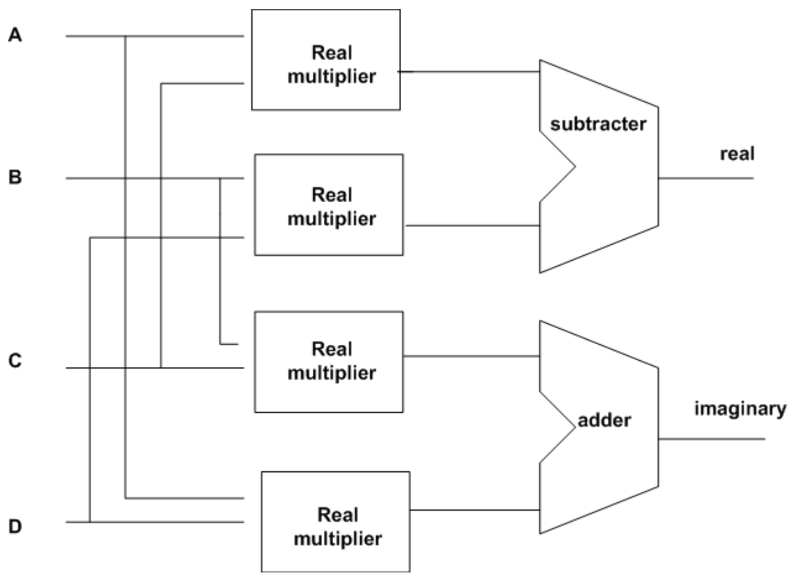


Figure 5.30: Complex Multiplier Block

5.3 Summary

A novel parallel-pipeline structure is invented, which is called Radix2^{2}Parallel. It is a small-area and low-power-consumption solution for Multiband OFDM UWB systems. The detailed implementation information is presented. The last part described the arithmetic block choices based on the performance analysis of Design Compiler.

Verification and ASIC Implementation 6

In this chapter, the results obtained from current design are enumerated. First the test bench is introduced and the results are compared with Matlab simulation results. Then, the detailed synthesis results which targets for FPGA and ASIC design are presented. The comparison with other published FFT processors is presented at the end.

6.1 Verification

6.1.1 Test Bench

In this test bench, first the input data are generated by Matlab. These data are sent to both Matlab simulation environment and Modelsim environment. The I/O connection between Matlab and Modelsim is described in Appendix B. After simulation, Matlab reads the data from Modelsim outputs and both simulation results from Matlab and Modelsim are compared. Figure 6.1 shows the structure of the test bench.

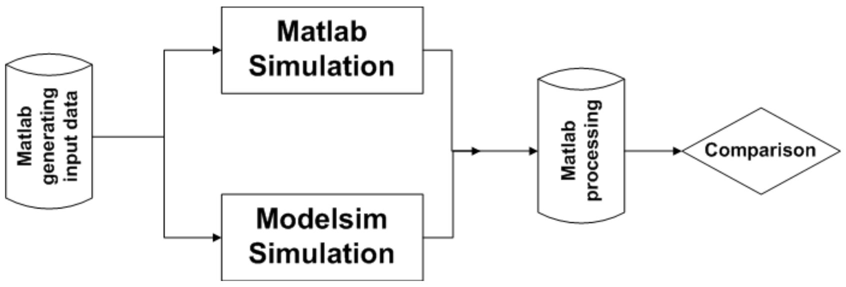


Figure 6.1: Test Bench Structure

6.1.2 Simulation Results

Figure 6.2 shows the errors between the modelsim outputs and the Matlab simulation results. It is shown that there is no error for the proposed FFT processor with the Matlab fixed point Radix2^2Parallel simulation, which means the proposed FFT processor achieves the expected accuracy.

Figure 6.3 shows the errors between the proposed processor and the Matlab builtin FFT. It is shown that the SQNR of the proposed FFT is above 40 dB, which is nearly the same as the fixed point simulation in Section 2.4. This result achieves the numerical precision requirement.

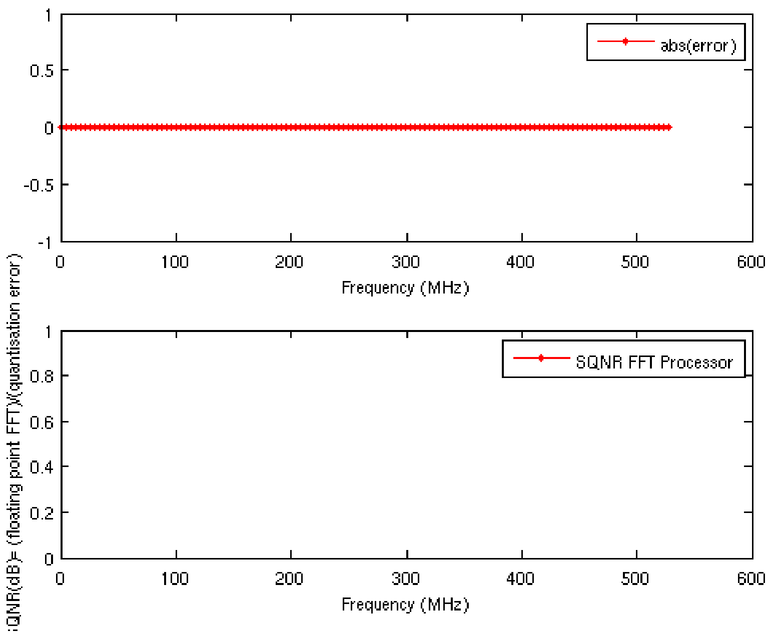


Figure 6.2: The result comparison between the processor and Matlab simulation

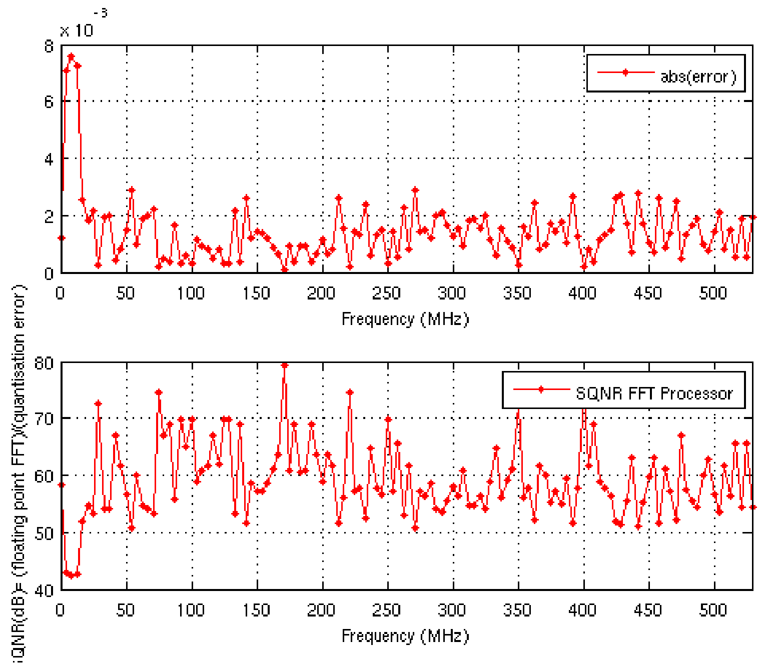


Figure 6.3: The result comparison between the processor and Matlab builtin FFT

6.2 Synthesis

Synthesis is the process of taking a design written in a hardware description language, such as VHDL, and compiling it into a netlist of interconnected gates which are selected from a user-provided library of various gates [41]. In the following part, Both FPGA and ASIC targeted synthesis results are presented.

6.2.1 Conclusion of the Radix2^2Parallel Architecture Requirements

The Radix2^2Parallel Architecture Requirements are concluded here in order to specify the synthesis environments.

6.2.1.1 Components Requirements

Most of the required components are analyzed in the previous chapter. The analysis in this section is focused on the components which affect the clock frequency.

In theory, for this parallel structure, the latency is 64 clock cycles, which is defined by the time slot from the first data coming into the processor until the first data coming out of the processor. However, if the input memory block is employed, the first stage can directly get the input data from the memory block. Therefore, the latency is reduced to 32 clock cycles because the latency of the first stage does not need to be counted. In reality, besides all the shift registers used to store the inner-stage data, there are registers to store the output data of the big blocks and multipliers. So there are $6 \times (2 \times 15 \times 2)$ bits registers required. In this way, the latency of the whole circuits will be increased by 6 clock cycles.

The main components used by R2^2SDF and Radix2^2Parallel architectures are shown in Table 6.1.

Table 6.1: The components of R2^2SDF and Radix 2^2Parallel

Number	R2^2SDF	Radix2^2Parallel
Complex Multiplier	3	5
Adder	28	56
Shift Register (bits)	$127 \times (15 \times 2)$	$128 \times (15 \times 2)$
Shimming Register (bits)	$6 \times (15 \times 2)$	$6 \times (2 \times 15 \times 2)$

6.2.1.2 Clock Frequency Requirement

The time constraint is 242.42 ns as described in Section 2.4 and the latency is 38 clock cycles, which is analyzed in the previous paragraph. Therefore, the estimated time period is

$$\frac{\texttt{time constraint}}{\texttt{clock cycles required}} = \frac{242.42ns}{38\texttt{cycles}} = 6.38\texttt{ns} \tag{6.1}$$

which means the required clock frequency is 157 MHz.

6.2.2 FPGA targeted results

The whole design is synthesized by Synplify Pro which is used for FPGA implementation. The purpose to use Synplify Pro is to check the function of the proposed processor. The target implementation FPGA is Xilinx Virtex4. The reason to use this FPGA is that the synthesis results can be compared with [4] to show the advantages of the proposed architecture. Because Virtex4 FPGA includes DSP48 components, the "+", "-", and "*" signs are used in VHDL coding, which are directly mapped to DSP48 components.

6.2.2.1 Timing analysis

The estimated period is 9.654ns and the corresponding frequency is 103.6MHz. The critical path delay is the multiplication components which are implemented by DSP48 components of Virtex 4. The total path delay (propagation + setup) of 9.654ns is 9.645 (99.9%) logic and 0.009 (0.1%) route.

By adding another registers at each multiplier output, and using retiming and pipelining functions of Synplify Pro, the estimated period is reduced to 5.978 ns and the corresponding frequency is 167.3MHz.

6.2.2.2 Resource usage analysis

The resource usage of Virtex4 is shown in Table 6.2.

Table 6.2: The resource usage of Virtex4

Logic Utilization	Used
Total Number of Slice Registers	1052
*Number used as Flip Flops	662
*Number used as Latches	390
Total Number of 4 input LUTS	3600
Number of DSP48s	20
Total equivalent gate count for design	59,479
Additional JTAG gate count of IOBs	4512

6.2.2.3 Comparison with [4]

Nuno Rodrigues, et al proposed a FFT FPGA implementation to demodulate OFDM in a UWB receiver [4], which was published in September, 2007. A mixed radix (radix 4 and radix 2) algorithms are used and 8 radix 4 butterflies are paralleled in their implementation. Table 6.3 is the comparison between their design with the proposed implementation.

Table 6.3: The comparison with [4]

	[4]	proposed implementation
Data bits	11	15
Total Number Slice Registers	7390	1052
*Number used as Flip Flops	3860	662
Total Number of 4 input LUTS	12749	3600
Number of DSP48s	48	20

From the comparison table, it is shown that the used resources of the proposed design are far less than the implementation in [4]. The reason is that [4] paralleled 8 radix 4 butterflies, which means 24 (3×8) complex multipliers are required. So 96 DSP48 components (24×4 real multipliers) are required for Virtex4. In their Virtex4 type, only 48 DSP48s are provided, which means extra logic blocks are needed to implement the left 48 multipliers. Also, a large amount of memories are required to store the inner stages data of their design. In contrast, our implementation just requires 5 complex multipliers.

6.2.3 ASIC targeted results

The design is synthesized by Synopsys Design Compiler which is targeted for ASIC implementation. The synthesis library is Faraday 90nm standard cell library [42], which is tailored for UMC' 90 nm logic LL-RVT (lowK) process.

For this library, the supply voltage of the core cells is 0.9V $\sim$ 1.1V. 1V supply voltage is used in the power analysis. The power consumption is 5.0nW/MHz/gate. One gate delay is 18.2 ps, which is measured from 101-stage NAND ring in the typical process and operating under 1.0V, 25 °C. One gate size is 4.7 μm^2 and gate density is 400k gates/mm^2.

6.2.3.1 Timing analysis

The 6.38 ns is used as time period for the timing analysis. It is shown from the analysis results that the critical path timing requirement is less than the required clock cycle time slot. Therefore, there is no pipelining requirement.

From Design Compiler analysis, the critical path of this processor is the whole multiplication process. Table 6.4 shows the detailed critical path and the timing requirement.

The required timing period is 6.38ns. However, the clock uncertainty is 0.2 ns and the library setup time is 0.45 ns. Therefore, the data required time is $6.38 - 0.2 - 0.45 = 5.73ns$. The slack is $5.74 - 4.99 = 0.74ns$. It is positive, which means the processor qualified the time constraint.

Table 6.4: The timing requirement of critical path

Critical Path	Timing requirement (ns)	Explanation
c1/counter_clk	0	Address generated from counter
c1/counter_output	0.98	Address generated from counter
m1/ROM_address	0	twiddle factor generation
m1/ROM_DATA_OUT	1.32	twiddle factor generation
m1/Multiplier_IN	0	Multiplication
m1/Multiplier_OUT	2.54	Multiplication
m1/MUltiplier_OUT	0	saving data
m1/Output_register	0.24	saving data
data arrival time	4.99	Total time requirement

6.2.3.2 Area Analysis

The total area of Radix2^2Parallel is 270925 μm^2. The combinational circuit takes 164468 μm^2 and the non combinational circuit takes 106457 μm^2.

Table 6.5 shows the detailed area analysis of each block for both R2^2SDF and Radix 2^2Parallel.

As shown from the table, the total area increases by 0.052 mm^2 (51923 μm^2) after parallelism (23.7% of R2^2SDF). The combinational circuit increases by 0.046 mm^2. The noncombinational circuit increases by 0.006 mm^2. These are caused by spliting the shift registers, adding extra shimming registers and extra multipliers and adders.

For further analysis of parallelism effects, Table 6.6 shows the detailed area analysis of the biggest block - Block 1.

As shown from the table, the memory part is the most area consuming part. The area of the total memory part is 126450 μm^2, which is 98 % of the total area of Block I. For the same 64 data storage, the area increases by 0.018 mm^2 (28 %) by the parallelism effects. The area of 32 data storage increases by 26 % after parallelism. The combinational components (Butterfly) are doubled. However, for the following stages, this effect becomes smaller because of less data storage in the following stages.

Table 6.5: The detailed area information

Block	$R2^2SDF$		$Radix2^2Parallel$	
	Area (μm^2)	Gate Count	Area (μm^2)	Gate Count
Input Mux	0	0	3271	696
Multiplication 1 up	0	0	15955	3395
Multiplication 1 down	17114	4641	16051	3415
Multiplication 2 up	0	0	15037	3199
Multiplication 2 down	15324	3260	14894	3170
Multiplication 3 down	10837	2305	10635	2263
Block 1	122133	25985	128805	27405
Block 2	35559	7566	42023	8941
Block 3	13709	2916	21046	4477
Block 4	3443	733	1738	370
Combinational	118458	25203	164468	34993
Non combinational	100544	21392	106457	22650
Whole circuit	219002	46594	270925	57643

Table 6.6: The area analysis of Block I

Block 1	$R2^2SDF$	$Radix2^2Parallel$	
	Area (μm^2)	Parallel Place	Area (μm^2)
stage 1	63607 (64 data storage)	up	41006(32 data storage)
		down	40807(32 data storage)
stage 2	35360 (32 data storage)	up	22276(16 data storage)
		down	22361(16 data storage)
Butterfly I	2249	up	2737
		down	2738
Butterfly II	2956	up	3025
		down	3105

6.2.3.3 Power Analysis

The power consumption of $Radix2^2Parallel$ is 346 mW, which is calculated by Design Compiler at 157 MHz. The same as the area analysis, the power analysis is also described hierarchically, which is shown in Table 6.7.

The same clock frequency is used in both of architectures. It is shown the $Radix2^2Parallel$

Table 6.7: Power Consumption analysis of Radix 2^2 Parallel

Block	R2^2SDF	Radix2^2Parallel
Power Consumption	(mW) (@157 MHz)	(mW)(@157MHz)
Input Mux	0	7.239
Multiplication 1 up	0	3.172
Multiplication 1 down	3.201	3.183
Multiplication 2 up	0	3.055
Multiplication 2 down	3	2.75
Multiplication 3 down	2.694	2.959
Block 1	237.114	258.355
Block 2	67.036	68.309
Block 3	23.650	21.146
Block 4	2.808	3.54e-02
Whole circuit	341.199	374.225

Table 6.8: Power Comparison

Block	R2^2SDF	Radix2^2Parallel
Power Consumption	(mW) (@157 MHz)	(mW)(@78MHz)
Input Mux	0	6.705
Multiplication 1 up	0	1.587
Multiplication 1 down	3.201	1.604
Multiplication 2 up	0	1.527
Multiplication 2 down	3	1.485
Multiplication 3 down	2.694	1.4
Block 1	237.114	128.607
Block 2	67.036	34.037
Block 3	23.650	10.534
Block 4	2.808	1.86e-02
Whole circuit	341.199	185.518

consumes slightly extra power compared with R2^2SDF. About 33 mW extra power is caused by parallelism. However, for the same throughput performance, the clock frequency of R2^2SDF should be as twice as the one of Radix2^2Parallel. Therefore, a new comparison is shown in Table 6.8. It is shown the proposed Radix2^2Parallel processor consumes much less power than R2^2SDF, which is only 54.37% of the R2^2SDF for the same performance.

6.2.3.4 Comparison

The comparison with [9]

An application specific instruction-set processor (ASIP) was developed for UWB-OFDM FFT calculation [9]. Table 6.9 is the comparison details.

Table 6.9: Comparison with [9]

	[9]	proposed implementation
Technology (nm)	120	90
clock frequency (MHz)	336	157
Cycles count	105	38
Word length (bits)	16	15
Area (mm^2)	2.44	0.27
Power	1.78 Watts	0.374 Watts

The comparison with [10]

Table 6.10: Comparison with [10]

	[10]	proposed implementation
Technology	0.18 μm ,1.8V	90 nm , 1V
clock frequency (MHz)	450	157
Parallel data format	2 data-path	2 data-path
Algorithm	Radix 2^4	Radix 2^2
Word length (bits)	10	10
Complex multipliers	2+0.41	5
Registers	190	128
Gates	70000	38540 (181140 μm^2)
Power	unknown	171.969 mW

During the implementation stage of our processor, a revised version paper was published which employed the similar parallel structure [10]. However, there are some key differences between these two architectures, which is described in Appendix A.

Because [10] employed 10 bits data format for the processor, the proposed processor is also changed to 10 bits for comparing the difference of both architectures. The performance analysis is shown in Table 6.10.

6.3 Summary

This chapter shows the validation of proposed processor. Both FPGA and ASIC targeted results are analyzed. Table 6.11 shows the results of Radix2^2Parallel processor for both 10 bits and 15 bits data type.

Table 6.11: The performance of Radix2^2Parallel processor

Radix2^2Parallel	10 bits	15 bits
Area (μm^2)	181140	270925
Power (mW)	171.969	374.225

Conclusions 7

7.1 Summary of Results

Present work involves the ASIC design flow from defining the processor specifications till getting the synthesis results. It can be obtained from the results that the proposed architecture is suitable for ASIC implementation and dramatically reduces the area and power consumption. The key contributions of this thesis work are summarized as follows:

- Firstly, the OFDM and UWB systems are analyzed. Combined with literature research and system simulation, the specifications of the FFT processor are defined.
- The FFT algorithms are fully reviewed and analyzed based on the multiplicative complexity. The purpose is to find the most suitable one for UWB system requirements and ASIC implementation. It is shown that Radix 2^x algorithms are the best ones for ASIC implementation.
- Different FFT processor structures are compared on the architecture level. It is found that R2^2SDF structure is the optimum option for ASIC FFT implementation of Multiband UWB system with regard to the memories and arithmetic blocks utilization. Important design choices and considerations are also analyzed and concluded.
- A novel parallel-pipeline structure is proposed, which is called Radix2^2Parallel. It is a small-area and low-power-consumption solution for MB-OFDM UWB systems.
- The proposed FFT solution is verified. Both FPGA and ASIC targeted synthesis results are presented.

7.2 Further Work

Although the objectives of the thesis have been attained, there are a few fundamental recommendations for further research.

- From the Design Compiler analysis, the power of the processor is mainly consumed by Memory blocks. Although the circular buffers are employed instead of shift registers, there are few effects for power reduction. Therefore, further research should be focused to reduce the power consumption of memory blocks.

- As discussed in Section 4.3.3, an optimal solution exists for data bus design in order to optimize the ASIC implementation and avoid overflow. Therefore, extra efforts need to be focused on the data value analysis to find the optimal solution. Combined with MB-OFDM system analysis, the overflow possibility in each stage needs to be determined. In this way, the optimal data bits in each stage can be defined for ASIC implementation.

- The connections of FFT block with the whole baseband processor should be analyzed in order to optimize the input and output memory block of FFT processor. Because the outputs of Radix2^2Parallel are in bit reversed order, the analysis should be focused on how to handle the connections with the following components - deinterleaver and viterbi decoder.

- The next attempt should be integrating the whole baseband processor with a single ASIC Chip. Deeper analysis of the baseband processor is necessary for this issue. More efforts are required for integrating all the blocks.

A Comparison Between the Proposed FFT Processor with [10]

During the implementation stage of our processor, a revised version paper was published which employed the similar parallel structure [10] (the first version [43]). However, there are some key differences between these two architectures, which result in different performances. The similarity and difference between the two processors are discussed as follows.

A.1 Similarity

The architecture of the proposed processor was developed independently during the whole design process. The architecture proposed by [10] was only noticed in September, 2008.

Both of the architectures employs the two path parallel-pipeline structure and separates the input data into even and odd data. However, the idea and reason of this separation is more detailed and clearer illustrated in this thesis. What's more, this data separation in the proposed processor is handled more artfully, which reduces the required registers.

A.2 Difference

The main differences between these two architectures are discussed as follows.

Different Clock Frequency

The proposed architecture reduces the clock frequency to 157 MHz, whereas the clock frequency of [10] is 450MHz. In fact, our purpose of employing the two path parallel way is to reduce the clock frequency, which can reduce the power consumption dramatically (this conclusion can be obtained from power analysis in Section 6.2.3.3). However, if the circuit is required to work at 450MHz, the areas and power will increase because the signals are captured in smaller time-slots, which means more flip-flops need to be added in the combinational circuit.

Different Registers

Only 128 complex words registers are required in the proposed architecture. However, 190 complex words registers are required in [10]. From the results in Section 6.2.3, it is the registers that consume the most area and power. 10 bits 62 complex words registers consume 42207 μm^2 area (23.3% of the proposed area) and 76.121 mW power (44% of the proposed power) (Design Compiler simulation using Faraday 90nm standard cell library [42]). Therefore, it is a huge difference between the two architectures from the area and power consumption points of view.

The last stage

The architecture of [10] employs the two path parallel structure for the whole pipeline structure. About 64 complex words registers are required for their last stage. However, in the proposed architecture, the last stages are artfully combined into a single stage as shown in Figure 5.7. Only 2 complex words registers are required in this stage, which reduces the area and power consumption.

A.3 Performance Comparison

From the analysis, it is shown that there are some key differences between the two architectures. Table 6.10 is rewritten here to show the performance differences.

Table A.1: Comparison with [10]

	[10]	proposed implementation
Technology	0.18 μm ,1.8V	90 nm , 1V
clock frequency (MHz)	450	157
Parallel data format	2 data-path	2 data-path
Algorithm	Radix 2^4	Radix 2^2
Word length (bits)	10	10
Complex multipliers	2+0.41	5
Registers	190	128
Gates	70000	38540 (181140 μm^2)
Power	unknown	171.969 mW

B I/O between Matlab and VHDL

Based on the test bench requirement, the data transmission is required between Matlab and VHDL. Therefore, the Input/Output (I/O) connections of these two language should be analyzed. In this appendix, One connection possibility is realized and the design details are shown below.

The data type of both environments are set to binary for transmission. The first part is the Matlab environment I/O design.

B.1 Matlab I/O Design

B.1.1 Output the data generated from Matlab

```
%Matlab Output generated
%Output the matlab data
%save as txt file format
output = outputsave (X)

% define fixed point format
wordlength = 15;
frac_input = 14;

% change to fixed data format
%real part of original data
xre = real(x0);
fi_xre = fi (xre,1,wordlength, frac_input);
%imaginary part of original data
xim = imag(x0);
fi_xim = fi (xim,1,wordlength, frac_input);

% change to binary format
fi_bin_xre= bin (fi_xre);
fi_bin_xim = bin (fi_xim);

% output data and save in txt format
fid = fopen(’input_data.txt’, ’wt’);
for i = 1:n, fprintf( fid, ’ A ¡= ”%s”;\n B ¡=”%s”;\n wait for 10 ns ;
\n’, fi_bin_xre(i,:),fi_bin_xim(i,:) ); end
```

```
fclose(fid);
```

B.1.2 Reading the data produced by Modelsim Simulation

```
%Matlab input reading
%The data to be read are in binary format and saved in txt file
% reading from the txt file
fid = fopen(’outfilere.txt’, ’r’);
re = fscanf(fid, ’%s’, [15 inf]);
fclose(fid);

%rechange the data format
re = re’;
re_length = length(re);

% fixed point convertion
% define fixed point format
wordlength = 15;
frac_input = 14;
outputre = zeros(re_length,1);
outputre_fi = fi(outputre,1, wordlength,frac_input);
outputre_fi.bin = re;

% the last 128-1 data are the fft output data
ree = zeros(n,1);
ree(1:n) = outputre_fi(re_length-128:re_length-1);
```

B.2 VHDL I/O Design

```
– output the data generated by VHDL
– there are real and imaginary part
– the process cotrolled by the clock
– the data are ouput as txt file

process (T_clk,c) is
– creat output txt file
file outfilere : TEXT open write_mode is ”outfilere.txt”;
– variable buffer
variable bufre:line;
– data read to the buffer and saved to the output txt file

begin
if (T_clk’event and T_clk =’1’) then
```

```
write (bufre, to_bitvector(c));
writeline(outfilere,bufre);
end if;
end process;

process (T_clk,d) is
file outfileim : TEXT open write_mode is "outfileim.txt";
variable bufim:line;

begin if (T_clk'event and T_clk ='1') then
write (bufim, to_bitvector(d));
writeline(outfileim,bufim);
end if;
end process;
```

The Synthesis Schematics C

In this appendix, the synthesis schematics from synthesis tools Synplify Pro and Synopsis Design Compiler are shown respectively.

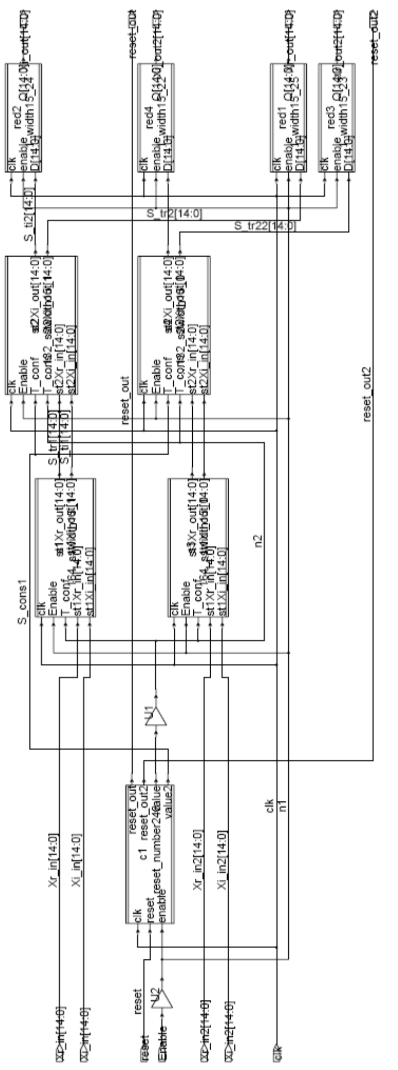


Figure C.1: The Big Block Structure of Radix2^2Parallel (Design Compiler)

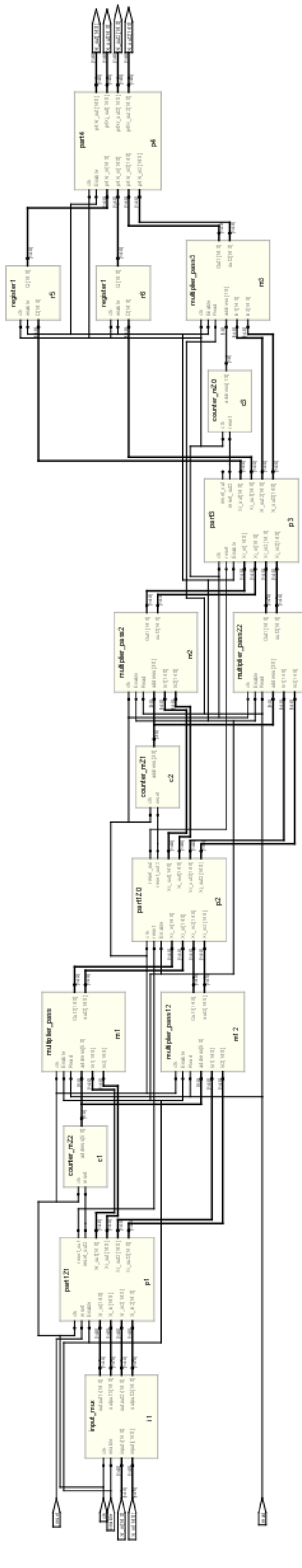


Figure C.2: Synthesis Schematic of Radix2^{2}Parallel (Synplify Pro)

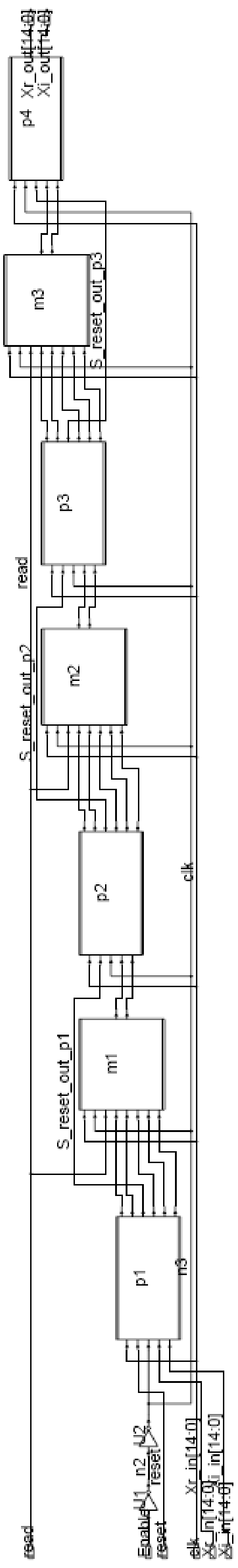


Figure C.3: Synthesis Schematic of R2^2SDF (Design Compiler)

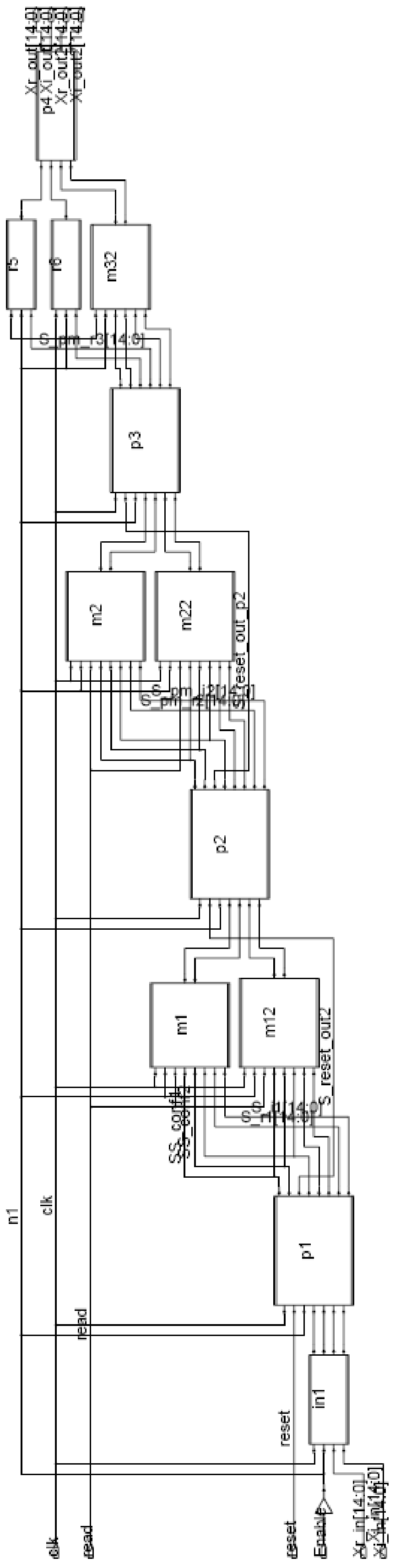


Figure C.4: Synthesis Schematic of Radix2^2Parallel (Design Compiler)

Matlab Code of the Radix2^{2}Parallel Based Algorithm

D

For better understanding of the Parallel Radix 2^2 Based Algorithm, the Matlab code is attached here.

```
function (output) = parallel(X0)
%parallel radix 2^2 based FFT algorithm;
%input data 128 point complex fomat;
%writen by nuoli;
%revised in 08.09.2008;

% change the input data to fixed point format;
%real part;
wordlength = 15;
xre = real(x0);
fi_xre = fi (xre,1,wordlength, wordlength-1);
%imaginary part;
xim = imag(x0);
fi_xim = fi (xim,1,wordlength, wordlength-1);

%generate the twiddlefactor;
w1 = radix22twiddles_part1(n);
w1_re = real(w1);
w1_im = imag(w1);
w1_re_fi= fi(w1_re, true, wordlength, wordlength-1);
w1_im_fi= fi(w1_im, true, wordlength, wordlength-1);
w2 = radix22twiddles_part2(n);
w2_re = real(w2);
w2_im = imag(w2);
w2_re_fi= fi(w2_re, true, wordlength, wordlength-1);
w2_im_fi= fi(w2_im, true, wordlength, wordlength-1);
w3 = radix22twiddles_part3(n);
w3_re = real(w3);
w3_im = imag(w3);
w3_re_fi= fi(w3_re, true, wordlength, wordlength-1);
w3_im_fi= fi(w3_im, true, wordlength, wordlength-1);

% fi_datatype define;
% keep the most signficant data for product;
% product word length keep 30;
% keep the most signficant data for full precision;
% product word length keep 16;
F = fimath;
F.ProductMode = 'KeepMSB';
F.ProductWordLength = 30;
F.SumMode = 'KeepMSB';
F.SumWordLength = 16;
F.OverflowMode = 'saturate';
F.RoundMode = 'floor';
F.CastBeforeSum = false;

% Change to fixed input and twiddle factor;
fi_xre.fimath = F;
fi_xim.fimath = F;
w1_re_fi.fimath = F;
w1_im_fi.fimath = F;
w2_re_fi.fimath = F;
w2_im_fi.fimath = F;
w3_re_fi.fimath = F;
w3_im_fi.fimath = F;

%inverse the data;
w1_re_fi = w1_re_fi';
w1_im_fi = w1_im_fi';
w2_re_fi = w2_re_fi';
w2_im_fi = w2_im_fi';
w3_re_fi = w3_re_fi';
w3_im_fi = w3_im_fi';

% change to even and odd part;
% the even part in this Matlab code means the Odd part shown in Figure 5.6;
% the odd part in this Matlab code means the Even part shown in Figure 5.6
for m =1:64
% odd part of real input data;
xred(m) = fi_xre(2*m-1);
% even part of real input data;
xree(m) = fi_xre(2*m);
% odd part of imagnary input data;
ximd(m) = fi_xim(2*m-1);
% even part of imagnary input data;
xime(m) = fi_xim(2*m);
end

    %twiddle factor even and odd part change
    w1length = length(w1);
w2length = length(w2);
w3length = length(w3);

    %First block
for m =1:w1length/2
% realodd part of twiddle factor w1
w1_re_fid(m) = w1_re_fi(2*m-1);
% image odd part of twiddle factor w1
w1_im_fid(m) = w1_im_fi(2*m-1);
% real even part of twiddle factor w1
w1_re_fie(m) = w1_re_fi(2*m);
% image even part of twiddle factor w1
w1_im_fie(m) = w1_im_fi(2*m);
end
%second block
for m =1:w2length/2
% realodd part of twiddle factor w2
w2_re_fid(m) = w2_re_fi(2*m-1);
% image odd part of twiddle factor w2
w2_im_fid(m) = w2_im_fi(2*m-1);
% real even part of twiddle factor w2
w2_re_fie(m) = w2_re_fi(2*m);
% image even part of twiddle factor w1
w2_im_fie(m) = w2_im_fi(2*m);
end
%third block
for m =1:w3length/2
% realodd part of twiddle factor w3
w3_re_fid(m) = w3_re_fi(2*m-1);
% image odd part of twiddle factor w3
w3_im_fid(m) = w3_im_fi(2*m-1);
% real even part of twiddle factor w3
w3_re_fie(m) = w3_re_fi(2*m);
% image even part of twiddle factor w3
w3_im_fie(m) = w3_im_fi(2*m);
end

%odd part input data computation;
% the odd part in this Matlab code means the Even part shown in Figure 5.6
%part 1;
(p1s1_red,p1s1_imd) = ripart1stage1(xred,ximd);
(p1s2_red,p1s2_imd) = ripart1stage2(p1s1_red,p1s1_imd);
(p1_red,p1_imd) = complexmultiplier(p1s2_red,p1s2_imd, w1_re_fid, w1_im_fid);
```

```
%part 2;
N=n/2;
for o=0:3
(p2s1_red(1+o*N/4:o*N/4+N/4),p2s1_imd(1+o*N/4:o*N/4+N/4))
=ripart1stage1(p1_red(1+o*N/4:o*N/4+N/4),p1_imd(1+o*N/4:o*N/4+N/4));
(p2s2_red(1+o*N/4:o*N/4+N/4),p2s2_imd(1+o*N/4:o*N/4+N/4))
=ripart1stage2(p2s1_red(1+o*N/4:o*N/4+N/4),
p2s1_imd(1+o*N/4:o*N/4+N/4));
(p2_red(1+o*N/4:o*N/4+N/4),p2_imd(1+o*N/4:o*N/4+N/4)) =
complexmultiplier(p2s2_red(1+o*N/4:o*N/4+N/4),p2s2_imd(1+o*N/4:o*N/4+N/4),
w2_re_fid, w2_im_fid);
end

%part 3;
N=n/2;
for o=0:15
(p3s1_red(1+o*N/16:o*N/16+N/16),p3s1_imd(1+o*N/16:o*N/16+N/16))
=ripart1stage1(p2_red(1+o*N/16:o*N/16+N/16),p2_imd(1+o*N/16:o*N/16+N/16));
(p3_red(1+o*N/16:o*N/16+N/16),p3_imd(1+o*N/16:o*N/16+N/16))
=ripart1stage2(p3s1_red(1+o*N/16:o*N/16+N/16),
p3s1_imd(1+o*N/16:o*N/16+N/16));
% the third part multiplication is not required here, because the twiddle fators become
constant 1;

% even part input data calculation;
% the even part in this Matlab code means the Odd part shown in Figure 5.6;
%part 1;
(p1s1_ree,p1s1_ime) = ripart1stage1(xree,xime);
(p1s2_ree,p1s2_ime) = ripart1stage2(p1s1_ree,p1s1_ime);
(p1_ree,p1_ime) = complexmultiplier(p1s2_ree,p1s2_ime, w1_re_fie, w1_im_fie);

%part 2;
for o=0:3
(p2s1_ree(1+o*N/4:o*N/4+N/4),p2s1_ime(1+o*N/4:o*N/4+N/4))
=ripart1stage1(p1_ree(1+o*N/4:o*N/4+N/4),p1_ime(1+o*N/4:o*N/4+N/4));
(p2s2_ree(1+o*N/4:o*N/4+N/4),p2s2_ime(1+o*N/4:o*N/4+N/4))
=ripart1stage2(p2s1_ree(1+o*N/4:o*N/4+N/4),
p2s1_ime(1+o*N/4:o*N/4+N/4));
(p2_ree(1+o*N/4:o*N/4+N/4),p2_ime(1+o*N/4:o*N/4+N/4)) =
complexmultiplier(p2s2_ree(1+o*N/4:o*N/4+N/4),p2s2_ime(1+o*N/4:o*N/4+N/4),
w2_re_fie, w2_im_fie);
end

%part 3;
for o=0:15
```

```
(p3s1_ree(1+o*N/16:o*N/16+N/16),p3s1_ime(1+o*N/16:o*N/16+N/16))
=ripart1stage1(p2_ree(1+o*N/16:o*N/16+N/16),p2_ime(1+o*N/16:o*N/16+N/16));
(p3s2_ree(1+o*N/16:o*N/16+N/16),p3s2_ime(1+o*N/16:o*N/16+N/16))
=ripart1stage2(p3s1_ree(1+o*N/16:o*N/16+N/16)
,p3s1_ime(1+o*N/16:o*N/16+N/16));
(p3_ree(1+o*N/16:o*N/16+N/16),p3_ime(1+o*N/16:o*N/16+N/16)) =
complexmultiplier(p3s2_ree(1+o*N/16:o*N/16+N/16),p3s2_ime(1+o*N/16:o*N/16+N/16),
w3_re_fie, w3_im_fie);
end

%combine the odd and even part
for i=1:64
p3_re(2*i) = p3_ree(i);
p3_re(2*i-1) = p3_red(i);
end

   for i=1:64
p3_im(2*i) = p3_ime(i);
p3_im(2*i-1) = p3_imd(i);
end

%part4 calculration
N=n;
for o=0:63
(p4_re(1+o*N/64:o*N/64+N/64),p4_im(1+o*N/64:o*N/64+N/64)) =
ripart1stage1(p3_re(1+o*N/64:o*N/64+N/64),p3_im(1+o*N/64:o*N/64+N/64));
end
p4_re = fi_bitreverse(p4_re,n);
p4_im = fi_bitreverse(p4_im,n);
p4 = complex(p4_re,p4_im);

% matlab bultiin algorithm;
y0 = fft(x0);
y0= y0./n;

%output comparsion;
figure(1)
plot(f,abs(double(p4)),'m.-',f,abs(double(y0)),'g.-')

figure(2)
% plot the abs error of the outputs of the two FFT.
subplot(211)
realerr = real(double(p4(:))-double(y0(:)));
imagerr = imag(double(p4(:))- double(y0(:)));
abserr = abs(double(p4(:))-double(y0(:)));
```

```
plot(f,abserr,'r.-');
legend('abs(error)')
xlabel('Frequency (MHz)')

% plot the snr of the signal to noise
subplot(212)
compare_y0_err =abs(double(y0(:)))./abserr(:);
snr_fi = 20*log10(compare_y0_err(:));
plot(f,snr_fi,'r.-');
legend('SQNR of the fixed point FFT ')
xlabel('Frequency (MHz)')
ylabel('SQNR(dB)= (floating point FFT)/(quantisation error)')
================================================
========= LOCAL FUNCTIONS ====== LOCAL FUNCTIONS ====
================================================

function (y_re,y_im) = ripart1stage1(x_re,x_im)
% function for part1 stage1, x_re and x_im are the real and image part of;
% input signal;

n = length(x_re);
% real part output;
y_re = zeros(1,n);
% image part output;
y_im = zeros(1,n);

% fixed point format;
wordlength = 15;
y_re = fi (y_re,1,wordlength, wordlength-1);
y_im = fi (y_im,1,wordlength, wordlength-1);

for q=1:n/2
y_re(q) = x_re(q)+x_re(q+n/2);
y_im(q) = x_im(q)+x_im(q+n/2);

    y_re(q+n/2)=x_re(q)-x_re(q+n/2);
y_im(q+n/2)=x_im(q)-x_im(q+n/2);
end

================================================
function (y_re,y_im) = ripart1stage2(x_re,x_im)
% function for part1 stage2;

n = length(x_re);
% real part output;
```

```
y_re = zeros(1,n);
% image part output;
y_im = zeros(1,n);

% fixed point format;
wordlength = 15;
y_re = fi (y_re,1,wordlength, wordlength-1);
y_im = fi (y_im,1,wordlength, wordlength-1);

for m=1:n/4
temp_re=x_im(m+(n*3)/4);
temp_im=0 - x_re(m+(n*3)/4);
x_re(m+(n*3)/4)=temp_re;
x_im(m+(n*3)/4)=temp_im;
end

for q=1:n/4
y_re(q) = x_re(q)+x_re(q+n/4);
y_im(q) = x_im(q)+x_im(q+n/4);
y_re(q+n/4)=x_re(q)-x_re(q+n/4);
y_im(q+n/4)=x_im(q)-x_im(q+n/4);
end
for q=1:n/4
y_re(q+n/2) = x_re(q+n/2)+x_re(q+(3*n)/4);
y_im(q+n/2) = x_im(q+n/2)+x_im(q+(3*n)/4);
y_re(q+(3*n)/4)=x_re(q+n/2)-x_re(q+(3*n)/4);
y_im(q+(3*n)/4)=x_im(q+n/2)-x_im(q+(3*n)/4);
end

==========================================
function (y_re,y_im) = complexmultiplier(x_re,x_im, w_re_fi, w_im_fi)
% complex multiplication;

n = length(x_re);
temp1 = zeros(n,1);
temp2 = zeros(n,1);
temp3 = zeros(n,1);
temp4 = zeros(n,1);

%fixed point format;
wordlength = 15;
temp1 = fi(temp1,1,wordlength, wordlength-1);
temp2 = fi(temp2,1,wordlength, wordlength-1);
temp3 = fi(temp3,1,wordlength, wordlength-1);
temp4 = fi(temp4,1,wordlength, wordlength-1);
```

```
% fi_datatype define;
% keep the most signficant data for product;
% product word length keep 15 ;
% keep the most signficant data for full precision;
% product word length keep 16;

F = fimath;
F.ProductMode = 'KeepMSB';
F.ProductWordLength = 30;
F.SumMode = 'KeepMSB';
%'FullPrecision';
F.SumWordLength = 16;
F.OverflowMode = 'saturate';
F.RoundMode = 'floor';
F.CastBeforeSum = false;

x_re.fimath = F;
x_im.fimath = F;

%real multiplication temp1 = x_re.*w_re_fi;
temp2 = x_re.*w_im_fi;
temp3 = x_im.*w_re_fi;
temp4 = x_im.*w_im_fi;

y_re = temp1 - temp4;
y_im = temp2 + temp3;
```

Bibliography

[1] A. Batra, J. Balakrishnan, G.R. Aiello, J.R. Foerster, and A. Dabak, "Design of a multiband ofdm system for realistic uwb channel environments," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 52, no. 9, pp. 2123–2138, Sept. 2004.

[2] Geert Leus, "Lecture notes of digital signal processing course in delft universtiy of technology," 2007.

[3] W.Y. Zou and Yiyan Wu, "Cofdm: an overview," *Broadcasting, IEEE Transactions on*, vol. 41, no. 1, pp. 1–8, Mar 1995.

[4] N. Rodrigues, H. Neto, and H. Sarmento, "A ofdm module for a mb-ofdm receiver," *Design & Technology of Integrated Systems in Nanoscale Era, 2007. DTIS. International Conference on*, pp. 25–29, Sept. 2007.

[5] *Standard ECMA-368: High Rate Ultra Wideband PHY and MAC Standard 2nd Edition.*

[6] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck, *Discrete-time signal processing*, Prentice-Hall, 1999.

[7] Bevan M.Baas, "An approach to low-power, high-performance fast fourier transform processor design," Ph.D. dissertation, Stanford University, 1999.

[8] Shousheng He and M. Torkelson, "A new approach to pipeline fft processor," *Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International*, pp. 766–770, Apr 1996.

[9] Ramesh Chidambaram, "A scalable and high-performance fft processor, optimized for uwb-ofdm," M.S. thesis, Delft University of Technology, 2005.

[10] Jeesung LEE and Hanho LEE, "A High-Speed Two-Parallel Radix-24 FFT/IFFT Processor for MB-OFDM UWB Systems," *IEICE Trans Fundamentals*, vol. E91-A, no. 4, pp. 1206–1211, 2008.

[11] INTEL, "Ultra-wideband (uwb) technology," *http://www.intel.com/technology/comms/uwb/*.

[12] et al. A. Batra, "Multi-band ofdm physical layer proposal for ieee 802.15 task group 3a," *Tech. Rep., IEEE P.802.15-04/0493r0*, 2004.

[13] E. Saberinia, K. C. Chang, G. Sobelman, and A. H. Tewfik, "Implementation of a multi-band pulsed-ofdm transceiver," *J. VLSI Signal Process. Syst.*, vol. 43, no. 1, pp. 73–88, 2006.

[14] Guoping Zhang and F. Chen, "Parallel fft with cordic for ultra wide band," *Personal, Indoor and Mobile Radio Communications, 2004. PIMRC 2004. 15th IEEE International Symposium on*, vol. 2, pp. 1173–1177 Vol.2, Sept. 2004.

[15] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee, "A 1-gs/s fft/ifft processor for uwb applications," *Solid-State Circuits, IEEE Journal of*, vol. 40, no. 8, pp. 1726–1735, Aug. 2005.

[16] Bernard Gold Lawrence R.Rabiner, *Theory And Application of Digital Signal Processing*, Prentice-Hall, 1975.

[17] Pierre Duhamel, "Implementation of "split-radix" fft algorithms for complex, real, and real-symmetric data," *IEEE Transactions on acoustics,speech, and signal processing*, 1986.

[18] E.H. Wold and A.M. Despain, "Pipeline and parallel-pipeline fft processors for vlsi implementations," *Computers, IEEE Transactions on*, vol. C-33, no. 5, pp. 414–426, May 1984.

[19] wikipedia, "Ultra-wideband," *http://en.wikipedia.org/wiki/Ultra-wideband.*

[20] Uma Shanker Jha and Ramjee Prasad, *OFDM towards fixed and mobile broadband wireless access*, ARTECH HOUSE, 2007.

[21] S. Weinstein and P. Ebert, "Data transmission by frequency-division multiplexing using the discrete fourier transform," *Communication Technology, IEEE Transactions on*, vol. 19, no. 5, pp. 628–634, October 1971.

[22] wikipedia, "Wimedia alliance," *http://en.wikipedia.org/wiki/Ultra-wideband.*

[23] R.Simon Sherratt and Sou Makino, "Numerical precision requirements on the multi-band ultra-wideband system for practical consumer electronic devices," *IEEE Transactions on Consumer Electronics*, May 2005.

[24] KESHAB K.PARHI, *VLSI Digital Signal Processing Systems-design and implementation*, JOHN WILEY & SONS,INC., 1999.

[25] Dimitris G.Manolakis John G.Proakis, *Digital Signal Processing principles, algorithms, and applications*, Prentice-Hall,INC., 1996.

[26] A. Cortes, I. Velez, A. Irizar, and J.F. Sevillano, "Area efficient ifft/fft core for mb-ofdm uwb," *Electronics Letters*, vol. 43, no. 11, pp. 649–650, 24 2007.

[27] J.W.Cooley and J.W.Tukey, "An algorithm for machine computation of complex fourier series," *Math Comput*, 1965.

[28] Shousheng He and M. Torkelson, "Designing pipeline fft processor for ofdm (de)modulation," *Signals, Systems, and Electronics, 1998. ISSSE 98. 1998 URSI International Symposium on*, pp. 257–262, Sep-2 Oct 1998.

[29] OH JUNG-YEOL (Chonbuk National Univ.Kor) LIM MYOUNG-SEOB(Chonbuk National Univ.Kor), "New radix-2 to the 4th power pipeline fft processor," *IEICE Trans Electron (Inst Electron Inf Commun Eng)*, 2005.

[30] Jung yeol Oh and Myoung seob Lim, "Fast fourier transform algorithm for low-power and area-efficient implementation," *IEICE TRANS.COMMUN*, APRIL 2006.

[31] S.C. Chan and K.L. Ho, "Prime-factor algorithm and winograd fourier transform algorithm for real symmetric and antisymmetric sequences," *Circuits, Devices and Systems, IEE Proceedings G*, vol. 136, no. 2, pp. 87–94, Apr 1989.

[32] P. Lavoie, "A high-speed cmos implementation of the winograd fourier transform algorithm," *Signal Processing, IEEE Transactions on*, vol. 44, no. 8, pp. 2121–2126, Aug 1996.

[33] A. M. Despain, "Fourier transform computers using cordic iterations," *IEEE Trans. Comput.*, vol. 23, no. 10, pp. 993–1001, 1974.

[34] G. Bi and E.V. Jones, "A pipelined fft processor for word-sequential data," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 12, pp. 1982–1985, Dec 1989.

[35] Jean Armstrong; Himal A. Suraweera; Simon Brewer; and Robert Slaviero, "Effect of rounding and saturation in fixed-point dsp implementation of ifft and fft for ofdm applications," *The Embedded Signal Processing Conference*, 2004.

[36] R. Sarmiento, V. de Armas, J.F. Lopez, J.A. Montiel-Nelson, and A. Nunez, "A cordic processor for fft computation and its implementation using gallium arsenide technology," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 6, no. 1, pp. 18–30, Mar 1998.

[37] John D. Arivoli, Thangadurai; O'sullivan, "Interleaver, deinterleaver, interleaving method, and deinterleaving method for ofdm data," January 2007.

[38] Neil H.E. Weste and David Harris, *CMOS VLSI Design. A Circuits and Systems Perspective*, Addison Wesley, third edition, 2004.

[39] Behrooz Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, pub-OXFORD:adr, 2000.

[40] Aoki Laboratory, "Arithmetic module generator based on arith, arith research group," *Tohoku University http://www.aoki.ecei.tohoku.ac.jp/arith/.*

[41] Digital ASIC Group, *Digital ASIC Design, A Tutorial on the Design Flow*, Lund University Website, 2005.

[42] FARADAY Technology Corporation, *FSD0A_A 90 nm Logic SP-RVT(Low-K) Process*, FARADAY Technology Corporation, 2006.

[43] Jeesung Lee, Hanho Lee, Sang in Cho, and Sang-Sung Choi, "A high-speed, low-complexity radix-24 fft processor for mb-ofdm uwb systems," *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, pp. 4 pp.–, May 2006.