國立交通大學

電機學院 IC 設計產業研發碩士班

碩士論文

Wimax 應用之快速傅立葉轉換軟硬體共同設計

Trade-offs on Harware-Software Co-design of FFT for Wimax applications

研究生：張登琦

指導教授：董蘭榮 教授

中華民國 九十九 年 一 月

Wimax 應用之快速傅立葉轉換軟硬體共同設計

Trade-offs on Hardware-Software Co-design of FFT for Wimax applications

研 究 生：張 登 琦　　Student：Teng-Chi Chang

指導教授：董 蘭 榮　　Advisor：Lan-Rong Dung

國立交通大學

電機學院 IC 設計產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of Master

in

Industrial Technology R & D Master Program on

IC Design

January 2010

Hsinchu, Taiwan, Republic of China

中華民國 九十九 年 一 月

Wimax 應用之快速傅立葉轉換軟硬體共同設計

研究生：張 登 琦　　指導教授：董 蘭 榮 博士

國立交通大學電機學院產業研發碩士班

中文摘要

簡單的說，如果一個處理器有處理 1000 MIPS 的能力，則它願意提供多少 MIPS 給快速傅立葉轉換運算，然後根據此 MIPS 去決定最適合此運算的 N 點分支傅立葉轉換硬體對於無線和行動通訊系統，傅立葉轉換模組是不可或缺的部分，特別是當寬頻無線系統需要一個高速且低功率硬體於高速封包式資料傳輸，這使得傅立葉轉換成為下一代無線系統必要的要求。在經過處理器運算量分析後，N 點分支傅立葉轉換以硬體系統需要一個高速且低功率硬體於高速封包式資料傳輸，這使得傅立葉轉換成為下一代無線系統必要的需求。一般而言，傅立葉轉換模組的設計會針對特定的系統，因此，希望能去設計一個可以適合不同標準規格的傅立葉轉換模組。在此論文中採用處理器彈性的特色和硬體具有加速的機制去建立一個傅立葉轉換模組，並且可以符合 IEEE 802.11n /16e 的規格要求。除此之外，我們提出對於單輸入輸出/多輸實現於系統中，並且它已 16 位元及 85MHz 產出率(Throughput rate)為規格。之後，我們有針對是用於系統的傅立葉轉換架構做分析比較。最後，不只有對 8 點分支傅立葉轉換於 FPGA 上做驗證，並且有針對提出的排程做驗證是可以滿足 IEEE 802.11n/16e 的規格。

Trade-offs on Hardware-Software Co-design of FFT for Wimax applications

Student: Teng-Chi Chang Advisor: Dr. Lan-Rong Dung

Industrial Technology R & D Master Program of
Electrical and Computer Engineering College
National Chiao-Tung University

Abstract

Briefly speaking, if a processor can process 1000 MIPS, it will provide which MIPS for us to operate Fast Fourier Transform (FFT). According to the MIPS it provides us, we can decide which N-points branch FFT of ASIC is suitable for us. FFT module is an indispensable part for wireless and mobile communication, especially when broadband wireless systems require a high speed and low power hardware module for its packet-based high-speed data transfer. This has made the design of FFT processor a critical requirement for the next generation wireless systems. In general, FFT module is designed for specific system. Therefore, it is desirable to design adaptive FFT module for different standards. This thesis adopts processor flexible characteristic and ASIC accelerated mechanism to set up a flexible FFT module which can meet IEEE 802.11n/16e standards. Besides, we propose optimized timing schedule for SISO/MIMO systems. After processor computational analysis, 64-points branch FFT of ASIC can be applied in proposed system and it computes 16-bits input data at a throughput rate of 85MHz. After that, we compare various pipeline-based FFT architectures suited to our system. Finally, it not only verifies the 8-points branch FFT on FPGA, but also checks proposed timing schedule which can satisfy IEEE 802.11n/16e specification.

誌　　謝

首先感謝我的指導教授董蘭榮老師在我碩士班生涯的悉心指導，在於研究學問的過程上讓我學會了更多以前沒碰過的觀念跟方向，也讓我學會解決問題的能力與思考，處事態度上有所進步，在這段時間讓我受益良多。

同時也要感謝學長俊衛，再研究的時候給予的幫忙以及實驗室曾經一起奮鬥相處的同學們志恆、建勛、宇佑、文俊、展嘉、嘉洋以及學弟建樺，謝謝你們熱心的協助與指導。因為有你們的陪伴，使我的研究生活增添了許多歡樂愉悅，過的非常充實。

最後要感謝我親愛的家人、及女友筱婷全家人，感謝他們的鼓勵、支持和愛心以及一起經歷我求學生涯所有經過的大大小小事情，使我得以在精神與生活上非常充實，順利完成學業。

僅以本論文獻給摯愛的大家、最深的謝意。

登琦 于新竹交大工程五館

2010年1月

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In digital signal processing and communications, FFT is one of the most utilized operations. The FFT plays an important role in modern communication systems, so its inverse transform-IFFT does. It is desired that FFT module can flexibly adjust FFT size to meet various standards. Generally speaking, FFT is designed for specific standards such as Ultra-Wide Band (UWB) system needs high throughput FFT module and Very High Data Rate DSL (VDSL) system required long length FFT computation. Therefore, it is difficult to design a FFT module which is suitable for any system specification.

For custom hardware that is often less cost-effective and flexible than general processors. Therefore, the approaches of ASIC have been added to achieve the high performance on software or processors. In the thesis, we discuss the ASIC and processor characteristic to design variable-length FFT modules. ASIC plays an accelerated role in the proposed system and it executes partial FFT algorithm. Processor can flexibly execute remaining FFT computation and it takes the performance of processor into consideration. Therefore, the proposed system can meet in different communication systems by reconfiguring processor computation.

1.2 Organization of this Thesis

In this thesis, the proposed FFT system can process on IEEE 802.11n/16e standards and it not only proposes optimized timing schedule, but also provides ASIC and processor analysis which are shown in the following chapters. The list of each chapter we write five chapters. Chapter 1 is our motivation. Chapter 2 reviews the background that we introduce the MIMO OFDM system standards, FFT algorithm and comparison of different radix algorithm. Then, variable-length FFT architectures are described. Chapter 3 presents the Co-design analysis. First, we analyze FFT computational complexity which can calculate processor performance. After that, the proposed timing schedule and architectures are combined for SISO/MIMO systems. Finally, we can analyze the relationship between ASIC and processor according to MOPS (Million Operations per Second). Chapter 4 shows the implementation of the structure with MicroBlaze Processor on FPGA tools. In the implementation domain of FFT processor, we chose the “MicroBlaze embedded system” which is implemented by FPGA tools. The MicroBlaze embedded soft core is a Reduced Instruction set Computer (RISC). In the environment, we will show the MIPS (Million Instructions per Second) of variable-length FFT based on IEEE 802.11n/16e standards. Chapter 5 is conclusion of this thesis.

Chapter 2

Backgroud

2.1 OFDM Backgrounds

In modern communication systems, the OFDM (Orthogonal Frequency Division Multiplexing) algorithm is effective in combating the problem of frequency-selective fading, inter-symbol interference (ISI), and inter-carrier interference (ICI). It is also efficient for wideband data transmission.

Bandwidth allocation of traditional frequency multiplexing is shown in Fig. 2.1. The conventional systems not only keep all sub-channels away from overlapping each other, but also allow some guard band bandwidth such that adjacent sub-channels will not introduce inter-channel interference (ICI). This allocation method is inefficient in bandwidth utilization. However, in Fig. 2.2, OFDM uses orthogonal carriers to modulate signal of sub-channels and eliminate ICI that allow sub-bands overlap.

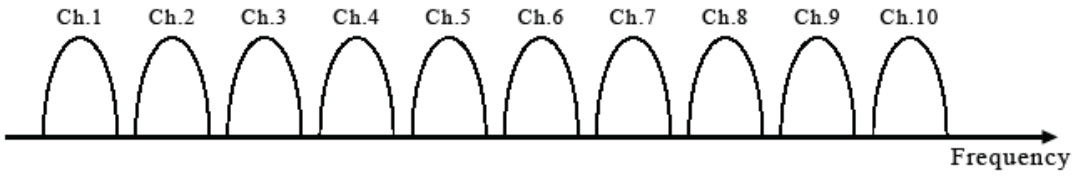


Fig. 2.1 Traditional bandwidth allocation of a frequency multiplexing system

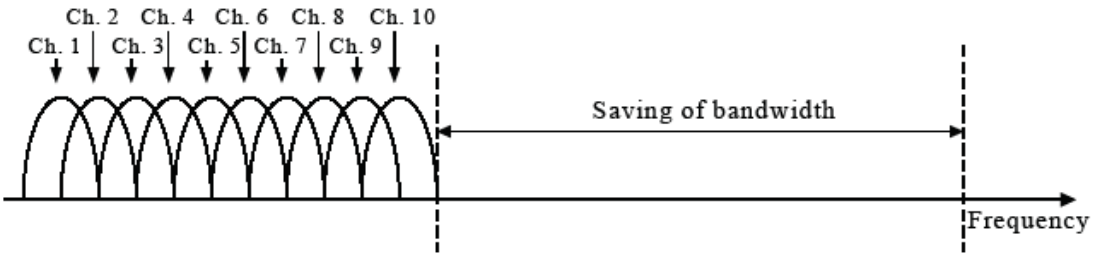


Fig. 2.2 Bandwidth allocation of OFDM

2.2 WLAN MIMO-OFDM System

Orthogonal Frequency Division Multiplexing (OFDM) is widely applied in high-speed Wireless Local Area Network (WLAN) such as IEEE 802.11a/g/n, Hiperlan/2, Wireless Personal Area Network (WPAN) and Ultra-Wide Band (UWB) system. OFDM is a special case of multi-carrier transmission, where a single data stream is transmitted over a number of lower rate sub-carriers. OFDM can be seen as either a modulation technique or a multiplexing technique. One of the main reasons to use OFDM is to increase the robustness against frequency selective fading or narrowband interference. To eliminate the banks of sub-carriers oscillators and coherent demodulators required by frequency division multiplex, Discrete Fourier Transform (DFT) processor is essential to be implemented.

Multiple-Input Multiple-Output (MIMO) system was instituted by Marconi in 1908. Channel fading can be suppressed by multiple antennas in both transmitter and receiver, called MIMO system, have received significant attention in recent years owing to their potential to increase system capacity.

The High Throughput Task Group which establishes IEEE 802.11n standard is going to draw up the next-generation WLAN proposal based on the 802.11a/g which is the current OFDM-based WLAN standards [1]. The IEEE 802.11n standard based on the MIMO OFDM system provides very high data throughput rate from the original data rate 54 Mb/s to the data rate in excess of 600 Mb/s because the technique of the MIMO can increase the data rate by extending an OFDM-based system. A block diagram of the 2x2 transceiver and receiver of IEEE 802.11n is shown in Fig. 2.3 and Fig. 2.4. Depending on the desired data rate, the modulation scheme can be Binary Phase Shift Keying (BPSK), Quaternary Phase Shift Keying

(QPSK), or Quadrature Amplitude Modulation (QAM) with 1-6 bits. The encoding rates in this specification are 1/2, 2/3, 3/4, or 5/6. The number of spatial sequence is supported by 1, 2, 3, or 4. The guard interval period is 400 ns or 800 ns. The bandwidth of the transmitted signal is 20 or 40 MHz. The FFT (Fast Fourier Transform) size is 64 points or 128 points based on IEEE 802.11n standard.

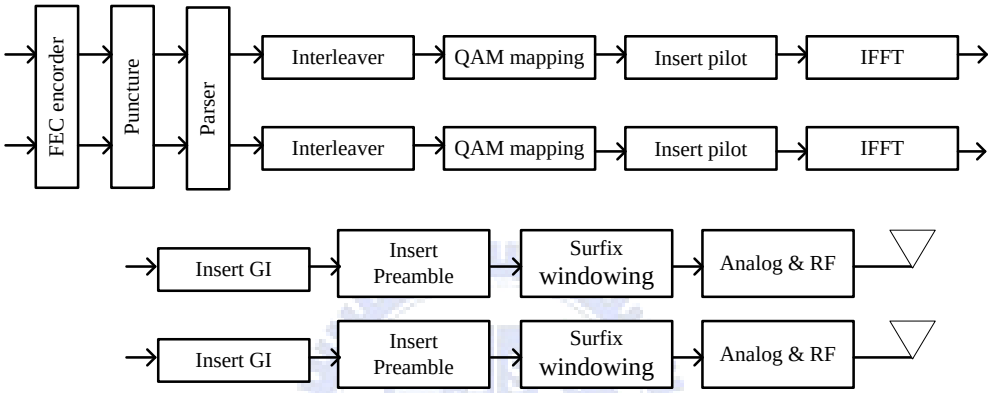


Fig. 2.3 Block diagram of IEEE 802.11n WLAN 2x2 transmitter system

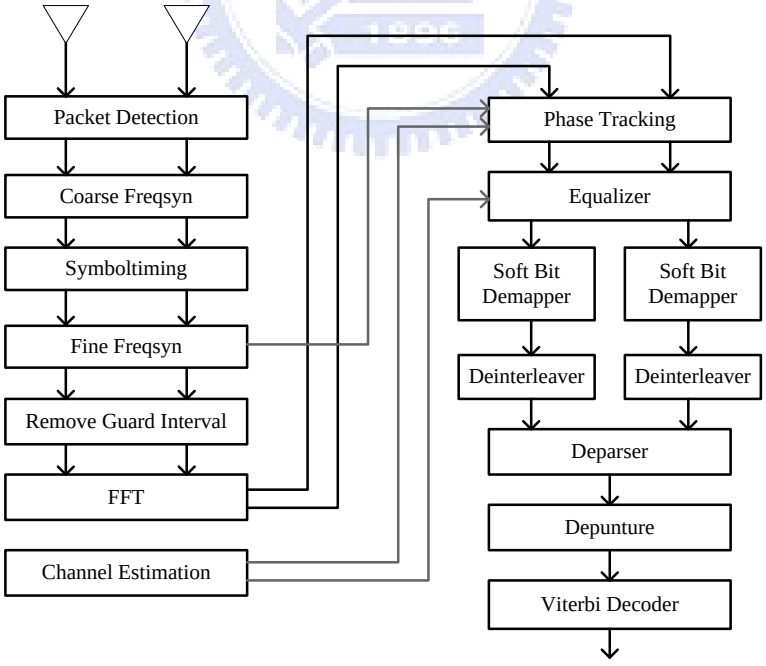


Fig. 2.4 Block diagram of IEEE 802.11n WLAN 2x2 receiver system

However, the IEEE 802.11n standard also increases the computational and hardware

complexities greatly compared with the current WLAN standards. The FFT/IFFT processor is one of the highest computational complexity modules in the physical layer of the IEEE 802.11n standard, as shown in Table 2.1 [2]. Multiple FFT processors are added to deal with multiple data sequences in a MIMO OFDM system. Therefore, FFT causes a large increase in the hardware complexity and power consumption.

Table 2.1 Comparison of the hardware complexity of the receiver

	Multiplier	Adder	Register	Gate Count (K)
Packet Detection	4	4	50	50
AGC	1	1	1	30
Frequency Offset	4	18	96	80
Frame Detection	8	8	8	50
FFT	1	12	68	160
Channel Estimation	0	0	128	60

2.3 Flexible FFT Processor

OFDM technique plays an important role in wireless and modern communication systems. The FFT processor is one of the highest computational complexity modules and FFT sizes, sampling rates are different in various standard requirements that Table 2.2 shows. It is desired to design a single FFT processor which adapts to various FFT sizes for different communication standards.

Table 2.2 FFT sizes and sampling rates needed in various communication systems

Communication System	FFT Size (Sampling Rate)
802.11a	64 (20MHz)
802.11n	64 (20MHz)、128 (40MHz)
802.16e	2048 (20MHz)、1024 (10MHz)、512 (5MHz)、128 (1.25MHz)
DAB	2048、1024、512、256 (2MHz)
DVB-T	8192、2048 (8MHz)
DVB-H	4096 (8MHz)
ADSL	512 (2.2MHz)
VDSL	8192 (34.5MHz)、4096 (17.3MHz)、2048 (8.6MHz)、1024 (4.3MHz)、512 (2.2MHz)
UWB	128 (528MHz)

2.4 Discrete Fourier Transform

The basic N-point DFT (Discrete Fourier Transform) X(k) of a complex data sequence x(n) is defined as:

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

Where the twiddle factor is

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

Most approaches to improve the efficiency of the computation of the DFT exploit the symmetry and periodicity properties of the twiddle factor. First, the complex conjunction

symmetry is

$$W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^* \tag{3}$$

Second, the periodicity in n and k is

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{(k+N)n} \tag{4}$$

According to equation (1), the computational complexity is O (N^2) through directly performing the required computation. It needs N^2 complex multiplications and N (N-1) complex additions. To use the FFT algorithm, the computational complexity can be reduced to O ($Nlog_rN$), where r means the radix-r FFT. The radix-r FFT can be derived from DFT by decomposing the N-point DFT into a set of recursively related r-point transform. There are two types of FFT algorithm are Decimation-in-Time (DIT) and Decimation-in-Frequency (DIF) FFTs. The computational complexity of these two types is the same.

2.4.1 Decimation-In-Time FFT Algorithm

The DIT algorithm is to decompose $x(n)$ into radix-r module sequence (It is the same as DIT FFT Radix-2 algorithm).

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} \\ &= \sum_{n:\,even} x(n) W_N^{kn} + \sum_{n:\,odd} x(n) W_N^{kn} \\ &= \sum_{r=0}^{N/2-1} x(2r) W_N^{2rk} + \sum_{r=0}^{N/2-1} x(2r+1) W_N^{(2r+1)k} \\ &= \sum_{r=0}^{N/2-1} x(2r) W_{N/2}^{rk} + W_N^k \sum_{r=0}^{N/2-1} x(2r+1) W_{N/2}^{rk} \end{aligned} \tag{5}$$

Fig. 2.5 shows an example of the 8-points DIT FFT radix-2 algorithm according to equation (5). We can find that order of the input time coefficients is must bit-reversed first in Fig. 2.5.

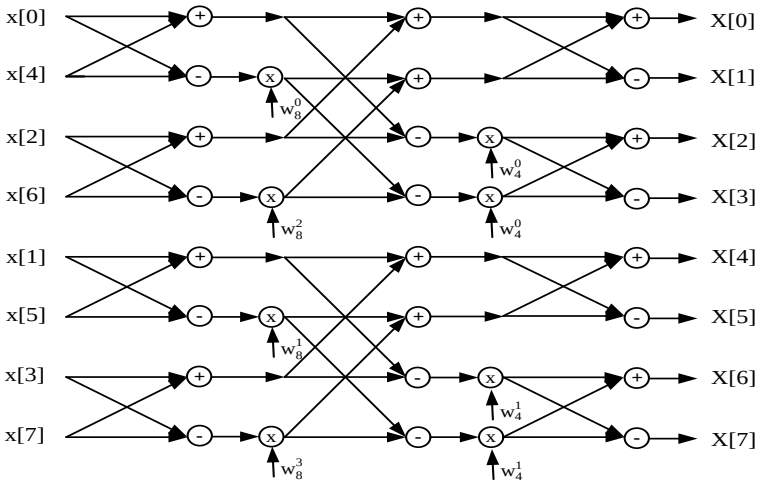


Fig. 2.5 8-points radix-2 DIT FFT signal flow graph

2.4.2 Decimation-In-Frequency FFT Algorithm

The DIF algorithm is to decompose $X(k)$ in the same way [3] (It is the same as DIF FFT Radix-2 algorithm).

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

$$\textit{even term}:X(2r)=\sum_{n=0}^{N-1}x(n)W_N^{2nr},\quad r\in\{0,1,...,N-1\}$$

$$=\sum_{n=0}^{N/2-1}x(n)W_N^{2nr}+\sum_{n=N/2}^{N-1}x(n)W_N^{2nr}$$

$$=\sum_{n=0}^{N/2-1}x(n)W_N^{2nr}+\sum_{n=0}^{N/2-1}x(n+\frac{N}{2})W_N^{2(n+\frac{N}{2})r}$$

$$=\sum_{n=0}^{N/2-1}\left\{x(n)+x(n+\frac{N}{2})\right\}W_{N/2}^{nr}$$

$$\textit{odd term}:X(2r+1)=\sum_{n=0}^{N/2-1}x(n)W_N^{n(2r+1)}+\sum_{n=N/2}^{N-1}x(n)W_N^{n(2r+1)}$$

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

$$=\sum_{n=0}^{N/2-1}\left\{x(n)-x(n+\frac{N}{2})\right\}W_{N/2}^{nr}W_N^{n}\qquad(6)$$

As equation (6) shown, two (N/2)-points DFTs are composed of X (2r) and X (2r+1).

It is well known that can combine these two equations as one basic butterfly (BF) module as shown in Fig. 2.6, where x(n) and x(n+N/2) are the input data.

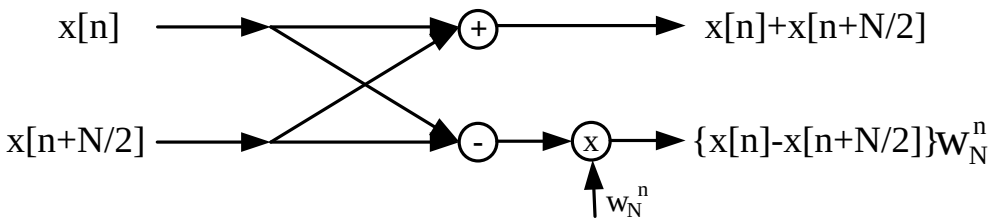


Fig. 2.6 The butterfly signal flow graph of radix-2 DIF FFT

By recursive decompositions, we can further partition these two small DFTs into even smaller DFTs, and so on. Finally, the completed N-points radix-2 DIF FFT algorithm can be obtained. The example of an 8-points radix-2 DIF FFT, in signal flow graph, is shown in Fig. 2.7.

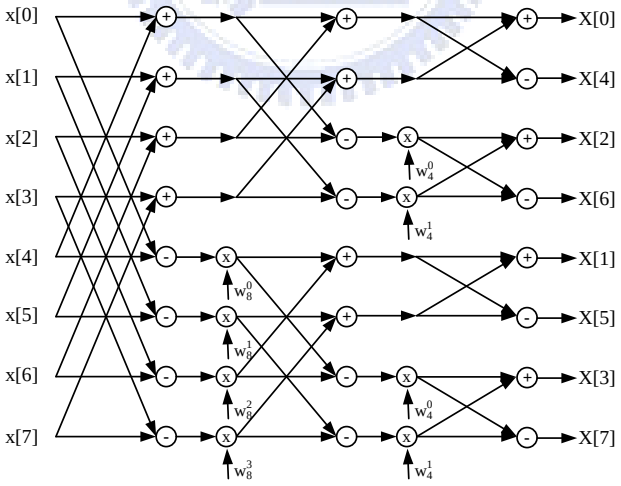


Fig. 2.7 8-points radix-2 DIF FFT signal flow graph

We can find that order of the output frequency coefficients is bit-reversed in Fig. 2.7.

2.5 Variable Length of FFT Architectures

FFT algorithms decompose the fundamental calculation of the DFT with a sequence of length N into continuously smaller subsequences. In section 2.4, the FFT algorithm is applied not only in DSP, image processing and digital data transmission systems, but also in biomedical electronic engineering and home networking. Therefore, FFT processor has variable transform length in different systems. To be able to compute variable FFT length, designer must to implement FFT processor with variable length.

Generally speaking, FFT processor architectures can be divided into two types. One is pipeline-based architecture [4], [5], [6], [7], [8], [9], [10], and the other is memory-based architecture [11], [12], [13], [14], [15], [16], [17]. Different architectures for FFT processors have different advantages and disadvantages, as listed in Table 2.3. There are advantages and disadvantages between these two architectures.

Table 2.3 Comparisons of FFT architectures.

Architectures	Advantages	Disadvantages
Pipeline-based architectures	High throughput rate	High hardware cost
	Regularity	
Memory-based architectures	Low hardware cost	A loss of the throughput rate

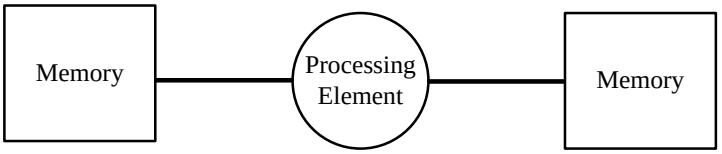
2.5.1 Memory-Based FFT Architectures

A general memory-based FFT processor structure mainly consists of a butterfly processing element (PE), a main memory, ROM for twiddle factor storage, and a controller.

The butterfly PE is responsible for the butterfly operations required by FFT operations. Moreover, the architecture design of PE is dependent on the use of FFT algorithm and generally dominates the performance of whole processor. The main memory stores processed data. The controller contains three functional units: data memory address generator, coefficient index generator, and operation state controller. The data memory address generator follows a regular pattern to generate several addresses, and then the main memory provides input data for butterfly PE and stores output data from butterfly PE according to these addresses. The coefficient index generator provides indices to select coefficients form coefficient ROM or maps to coefficients through twiddle factor generator [18], [19].

Memory-based FFT architectures are designed to increase the utilization rate of butterfly PE's. Different from the pipeline-based architectures, memory-based FFT processor often has one or two large memory block(s) that is accessed by all other PE components, instead of being distributed to many pipelined local arithmetic units.

(a) In-place type architecture



(b) Out-of-place type architecture

Fig. 2.8 Memory-based architecture block diagram

Main memory allocation and access strategy of a memory-based FFT processor can be classified as two types: in-place type and out-of place type [20], [21], [22]. In Fig. 2.8(a), in-place architecture, output data of butterfly PE are written back to the original memory bank with the same addresses as the previously loaded of input data [23]. Alternatively, if output data are written to another memory block without overwriting input data, this design will be generally called out-of-place Fig. 2.8(b). Therefore, memory size of the out-of-place design generally will be twice that of the in-place design.

2.5.2 Pipeline-Based FFT Architectures

The pipeline-based FFT architectures are the most popular FFT processor because they are designed by emphasizing speed performance and the regularity of data path. The best way to obtain the pipeline-based FFT architectures is through vertical projection of signal flow graph (SFG). In Fig. 2.9, we take an example to explain a projection mapping for 8-points radix-2 DIF FFT.

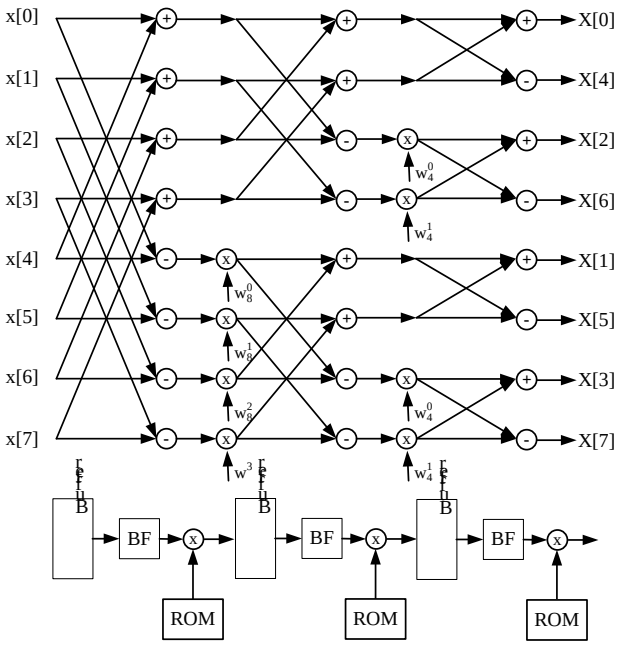


Fig. 2.9 Projection mapping of radix-2 DIF FFT signal flow graph.

In Fig. 2.9, the structure of each stage obtained from the projection mapping is called the processing element (PE). A processing element contains a basic butterfly (BF) unit for addition and subtraction between two input data of each stage, a complex multiplier and a block of buffer are used to store and reorder data for the butterfly unit of next stage.

As the following paragraph that complexity comparison, FFT computational complexity analysis and time schedules will be discussed. SISO Ⅰ, SISO Ⅱ and MIMO schedules are the three time schedules we will show in Chapter 3.

Chapter 3

Co-Design Analysis on ASIC and Processor of FFT

3.1 Introduction

In recent years, a lot of products with some digital signal processing (DSP) techniques have become very popular. They are often more cost-effective and less risky than custom hardware, particularly for low-volume applications, where the development cost of custom ICs may be prohibitive.

In a MIMO OFDM system [24], multiple antennas need multiple FFT and inverse transform (IFFT) processors in transmitter and receiver shown in Fig. 2.3 and Fig. 2.4. Thus, it causes a large increase in the hardware complexity and power consumption. Besides, based on various standards, designers need to re-design different length and throughput of FFT processors that shown in Table 2.2. In recent years, applications in processor become very popular. We use the advantages of processor to propose a new method that the processor and ASIC co-design can enhance flexibility and utilize time schedule efficiently to reduce ASIC cost. We provide designers two crucial messages. How many processor's performance needed in various environments? How many branch FFT need to be implemented by hardware in various processors?

3.2 Complexity Comparison

From Table 3.1 [25] and Table 3.2 [26] show the multiplication and additions comparison, the multiplication and addition of radix-8 have the lowest complexity compared with radix-2 and radix-4. In Table 3.1, the constant multiplication can be implemented by shifters and adders, which the hardware cost is smaller than a real multiplication. Table 3.3 [27] is the complexity equation of multiplications and additions that the radix-8 type-1 algorithm is the original radix-8 FFT algorithm. In radix-8 type-2 algorithm, we replace multiplication of W_8^1 into p additions that the W_8^1 will be implemented in the next section 4.4.2: "Hardware Design on ASIC FFT".

Table 3.1 Multiplication comparison [25]

N-point	Radix-2	Radix-4	Radix-8	
Multiplier	Multiplier	Multiplier	Multiplier	Constant Multiplier
8	2	3	0	2
16	10	8	6	4
32	34	31	20	8
64	98	76	48	32
128	258	215	152	64
256	642	492	376	128
512	1538	1239	824	384
1024	3586	2732	2104	768
2048	8194	6487	4792	1536
4096	18434	13996	10168	4096
8192	40962	32087	23992	8192

Table 3.2 Multiplications and additions comparison [26]

	Real Multiplications			Real Additions		
N-point	All Used by Radix-2	All Used by Radix-4	All Used by Radix-8	All Used by Radix-2	All Used by Radix-4	All Used by Radix-8
16	24	20		152	148	
32	88			408		
64	264	208	204	1032	976	972
128	720			2054		
256	1800	1392		5896	5488	
512	4360		3204	13566		12420
1024	10248	7856		30728	28336	

Table 3.3 Equation of multiplications and additions comparison[27]

Algorithm	Real Multiplication	Real Addition
Radix-2	$\frac{3N}{2}\log_2 N-\frac{7}{2}N+8$	$\frac{5N}{2}\log_2 N-\frac{7}{2}N+8$
Radix-4	$\frac{9N}{8}\log_2 N-3N+3$	$\frac{25N}{8}\log_2 N-3N+3$
Radix-8 Type-1	$\frac{25N}{24}(\log_2 N-3)+4$	$\frac{73N}{24}\log_2 N-\frac{25}{8}N+4$
Radix-8 Type-2	$\frac{21N}{24}\log_2 N-\frac{25}{8}N+4$	$\frac{8p+73N}{24}\log_2 N-\frac{25}{8}N+4$

According to the hardware area and power consumption of complex number multiplier, we only focus on the number of real number multiplications. In Fig. 3.1, radix-8 type-2 has the lowest computational complexity, so we choose radix-8 type-2 as the building block to implement FFT algorithm.

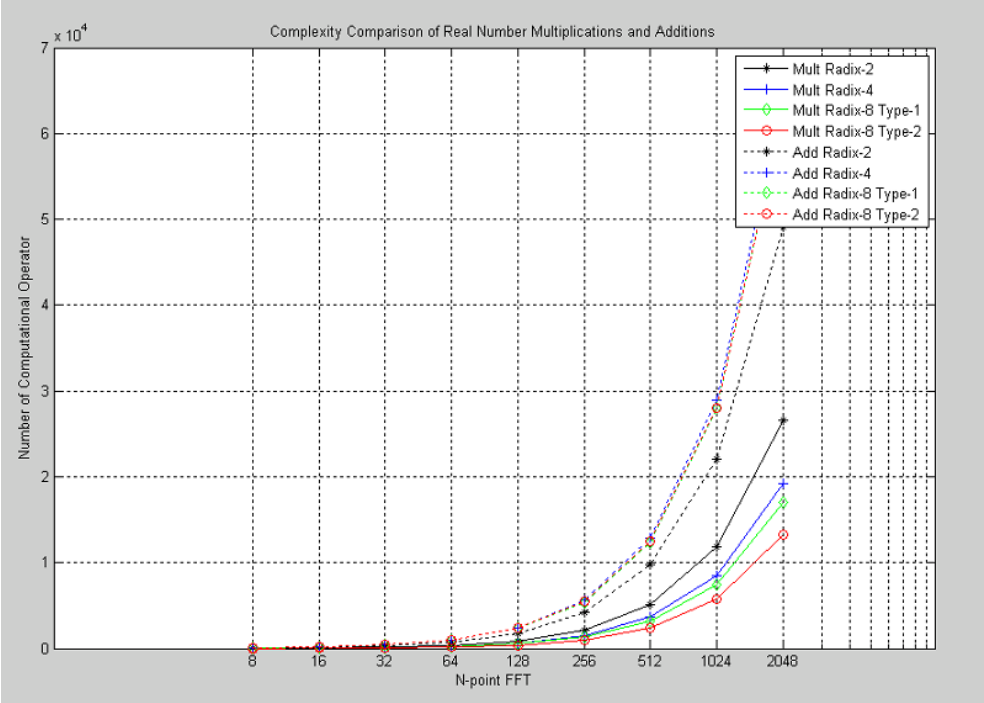


Fig. 3.1 Complexity comparison of Table 3.3

3.3 FFT Computational Complexity Analysis

As the equation (6) shown in section 2.4.2: “Decimation-In-Frequency FFT Algorithm” is composed by even term X(2r) and odd term X(2r + 1) of two (N/2)-point DFTs. It is well known that one can combine these two equations as one basic butterfly (BF) module as shown in Fig. 3.2, where x(n) and x(n+N/2) are the input data.

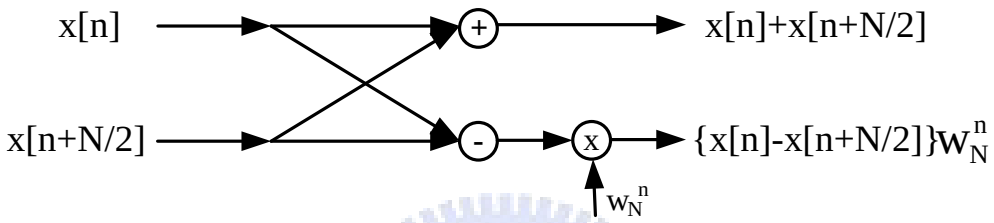


Fig. 3.2 The butterfly signal flow graph of radix-2 DIF FFT

By recursive decompositions, we can further partition small DFTs into even smaller DFTs, and so on. For example, an 8-points radix-2 DIF FFT, in signal flow graph, is shown in Fig. 3.3.

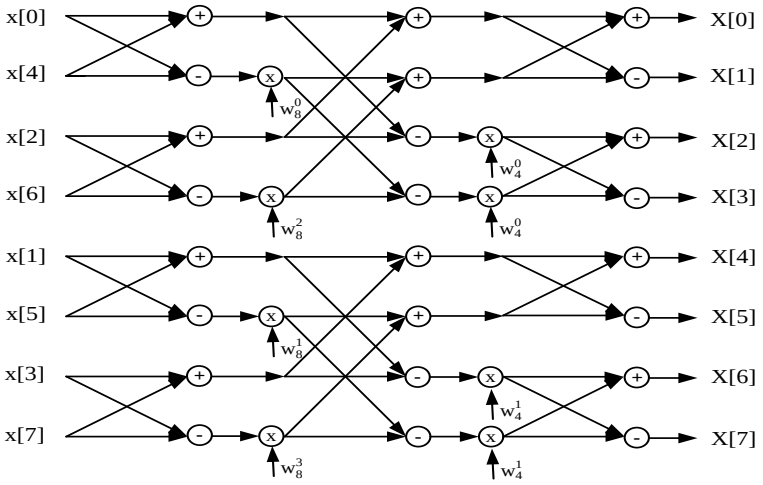


Fig. 3.3 8-points radix-2 DIT FFT signal flow graph

Long-length FFT can be decomposed into several branch FFT by different radix algorithm. Take section 3.2: **"**Complexity Comparison" as a conclusion that radix-8 FFT reduces the complexity more than other radix. But FFT length is restricted to power of eight only. In any event, FFT architecture is composed of many butterfly units, and additions and multiplications form butterfly units. Thus, we can analyze FFT computation by calculating number of additions and multiplications.

Complex addition can be decomposed two real additions, and complex multiplication can be decomposed two real additions and four real multiplications as shown equation (1).

$$(\mathrm{Re}+\mathrm{Im}*\mathrm{j})\times(\cos(\frac{2\pi nk}{N})+\sin(\frac{2\pi nk}{N})*\mathrm{j})$$
$$=(\mathrm{Re}\times\cos(\frac{2\pi nk}{N})-\mathrm{Im}\times\sin(\frac{2\pi nk}{N}))+(\mathrm{Re}\times\sin(\frac{2\pi nk}{N})+\mathrm{Im}\times\cos(\frac{2\pi nk}{N}))*\mathrm{j} \quad (1)$$

Therefore, we try to evaluate different length of FFT computation complexity which is a little different from section 3.2: **"**Complexity Comparison". Because we calculate any computation in terms of processor operations, it doesn't include any hardware reduce computation, just like W_8^1 can be implemented by shifters and adders. In this article, we take IEEE 802.11n/16e standards into consideration as shown in Table 2.2 of section 2.3: "Flexible FFT Processor". FFT length covers from 64-points to 2048-points. We regard real addition or real multiplication as an operation in the analysis. In IEEE 802.11n/16e standards, 64-points/ 2048-points is the critical case separately, because of long-length FFT increase operations dramatically and symbol durations are the same shown in Table 3.4. Therefore, we analyze these two cases and assume partial branch FFT which is implemented by hardware as shown in Table 3.5 and Table 3.6. This analysis can be applied to others standards.

In Table 3.4, the processor operations are added by real additions and real multiplications. According to equation (2), the processor operations are divided into three parts: Addi-

tion operations, multiplications operations and operations for radix-8 only. Addition operations present the all used additions numbers of remaining FFT stages. Multipliers operations are the all used multiplications numbers of remaining FFT stages. Operations for radix-8 only mean that because of the radix-8 FFT algorithm just only uses the W_8^1 and W_8^3 constants than other radix. Therefore, we must take the operations for radix-8 only when we use the radix-8 FFT algorithm, the other radix will not be used. This analysis can be applied to others FFT sizes.

Table 3.4 Comparison operations of FFT size in IEEE 802.11n/16e standards

802.11n FFT Size(Sampling Rate)	Processor Operations = Real additions +Real multiplications
128 (40 MHz)	3142
64 (20 MHz)	1254
802.16e FFT Size(Sampling Rate)	Operations = Real additions +Real multiplications
2048 (20 MHz)	83462
1024 (10 MHz)	38150
512 (5 MHz)	16518
128 (1.25 MHz)	3142

Processor Operations =

$$\underbrace{2\times N\times S}_{\text{addition operations}}+\underbrace{(\sum_{i\text{-}0}^{stage}(\prod_{j=0}R_j)\times(\frac{N}{\prod_{i}R_{j+1}}-1)\times(R_{j+1}-1))}_{\text{multipliers operations}}+\underbrace{\frac{N}{8}\times K\times 2\times(2+4)}_{\text{for radix-8 only}}$$

$$\text{where}: R_0=1, S=\text{Number of remaining stages},$$
$$K=\text{Number of radix-8 groups}, Rn=\text{radix-}Rn, n=\{1,2,3,K\} \quad (2)$$

Table 3.5 is an example that shows all the remaining stages of the 64-points FFT processor operations according to equation (2). If we want to do a radix-8 of 64-points FFT, we can only choose one stage to perform. Therefore, we choose stage is 1 because we only do one time radix-8 through the remaining stages; S is 3 because radix-8 reduces 3 stages; N is 64 that is because we choose 64-points FFT to process; K is 1 because we choose radix-8 that we must consider the constants of W_8^1 and W_8^3. Finally, we can take 2*64*3+7*7*(2+4)+ 1*8*2*(2+4) = 774 which 774 is our desired processor operations. By the same way, Table 3.6 is the processor operations of 2048-points FFT according to equation (2).

Table 3.5 Comparison of different length ASIC operations of a 64-points FFT

ASIC length of 64-points FFT	Operations = Real additions + Real multiplications
64	0
32	2*64+31*1*(2+4) = 314
16	2*64*2+15*3*(2+4) = 526
8	2*64*3+7*7*(2+4)+1*8*2*(2+4) = 774
4	2*64*4+31*1*(2+4)+2*3*7*(2+4)+ 1*8*2*(2+4) = 1046
2	2*64*5+15*3*(2+4)+4*1*7*(2+4)+ 1*8*2*(2+4) = 1174
0	2*64*6+7*7*(2+4)+8*0*7*(2+4)+2*8*2*(2+4) =1254

Table 3.6 Comparison of different length ASIC operations of a 2048-points FFT

ASIC length of 2048-points FFT	Operations = Real additions + Real multiplications
2048	0
1024	2*2048+1023*(2+4) = 10234
512	2*2048*2+511*3*(2+4) = 17390
256	2*2048*3+255*7*(2+4)+256*2*(2+4) = 26070
128	2*2048*4+1023*(2+4)+2*127*7*(2+4)+256*2*(2+4) = 36262
64	74246-2*2048*6-32*7*7*(2+4)+1*256*2*(2+4) = 43334
32	2*2048*6+255*7*(2+4)+8*31*7*(2+4)+2*256*2*(2+4) = 51846
16	74246-2*2048*4+32*15*3*(2+4)-32*7*7*(2+4)+1*256*2*(2+4) = 60166
8	74246-2*2048*3+2*256*2*(2+4) = 68102
4	2*2048*9+255*7*(2+4)+8*31*7*(2+4)+8*8*3*7*(2+4)+ 3*256*2*(2+4) = 75270
2	74246-2*2048+256*1*3*(2+4)+2*256*2*(2+4) = 80902
0	2*2048*2+2*511*3+4*511*3+4*(10882+3332)+3*256*2*(2+4) = 83462

Not only 64-points/2048-points is the critical case in IEEE 802.11n/16e separately but also we will show the other cases, such as 128-points, 512-points, 1024-points FFT, as shown in Fig. 3.4.

In Fig. 3.4, the x-axis means which FFT length of ASIC we can choose; the y-axis is the processor operations we calculate from equation (2). Take an example of IEEE 802.16e 2048-points FFT from Fig. 3.4, if the processor only provides 30000 operations for us to do FFT, we will choose 256-points ASIC FFT for our branch FFT. It means the processor just

takes 26070 operations to do software FFT and then the other remaining stages will be processed by 256-points ASIC FFT. As the following paragraph, users can decide how much operations they want to provide for software to calculate FFT and then the others can be done for ASIC FFT by hardware. That is why we conclude the Fig. 3.4 of all these cases according to equation (2).

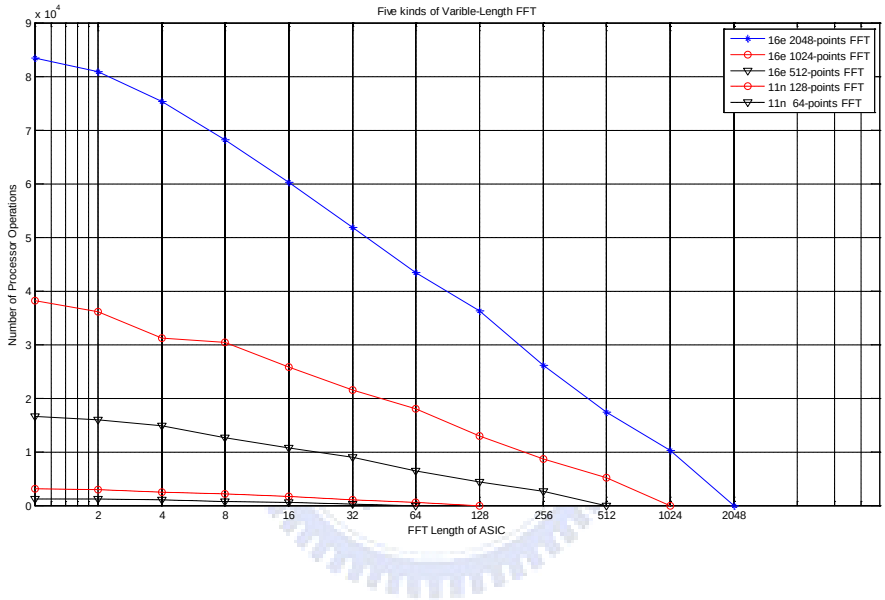


Fig. 3.4 Different cases of length FFT according to processor operations

3.4 ASIC and Processor Timing Schedule Analysis

In this article, because we need to design a variable-length FFT module in our system, timing schedules need to be executed independently. The goal is that we try to lower length of branch FFT and enhance processor and ASIC utilization.

3.4.1 SISO System Timing Schedule Ⅰ

In SISO system we proposed two schedules. First in schedule Ⅰ, we input sequences and write it into memory which can receive continuous data and reorder data. After that process data sequences have been ordered within symbol duration, therefore processor and ASIC utilization are not 100% as shown in Fig. 3.5. In other words, processor has less time to operate. Because of ASIC occupies part of symbol duration; therefore the processor needs better operations to performance.

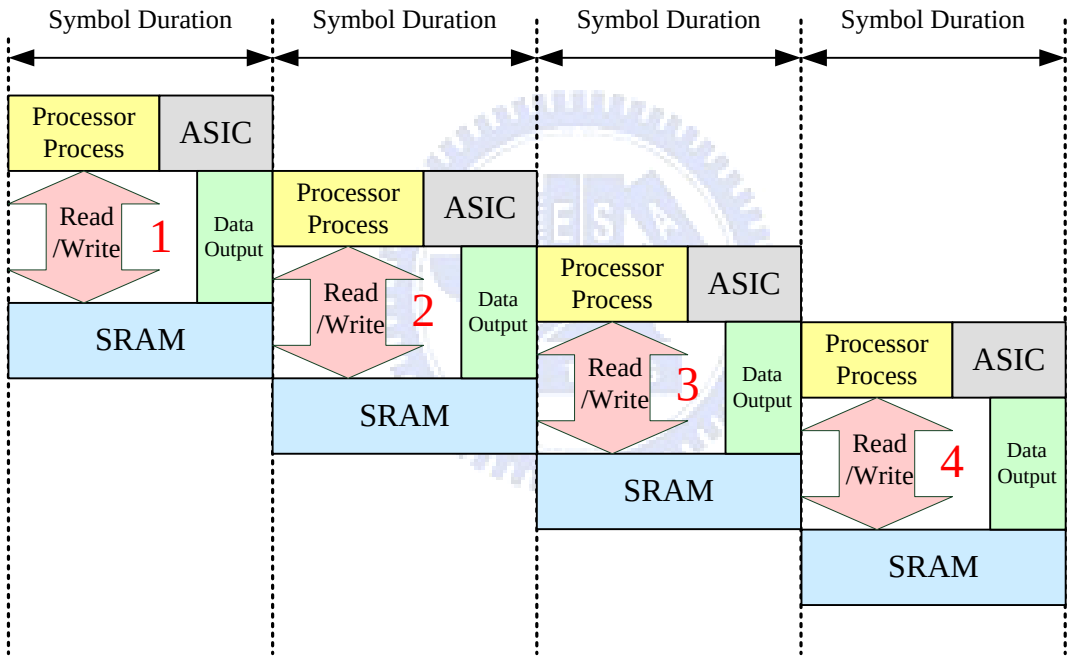


Fig. 3.5 Time schedule Ⅰ of SISO system

In Fig. 3.6, it shows system block diagram based on time schedule Ⅰ. This module is used to communicate On-Chip Peripheral Bus (OPB) handshake signals [28] between software and hardware. Software is used to operate FFT software parts with C-language. ASIC FFT was responsible for branch FFT algorithm if the software parts have been prepared. Control register and state machine modules are stored control signals which govern entire data flow.

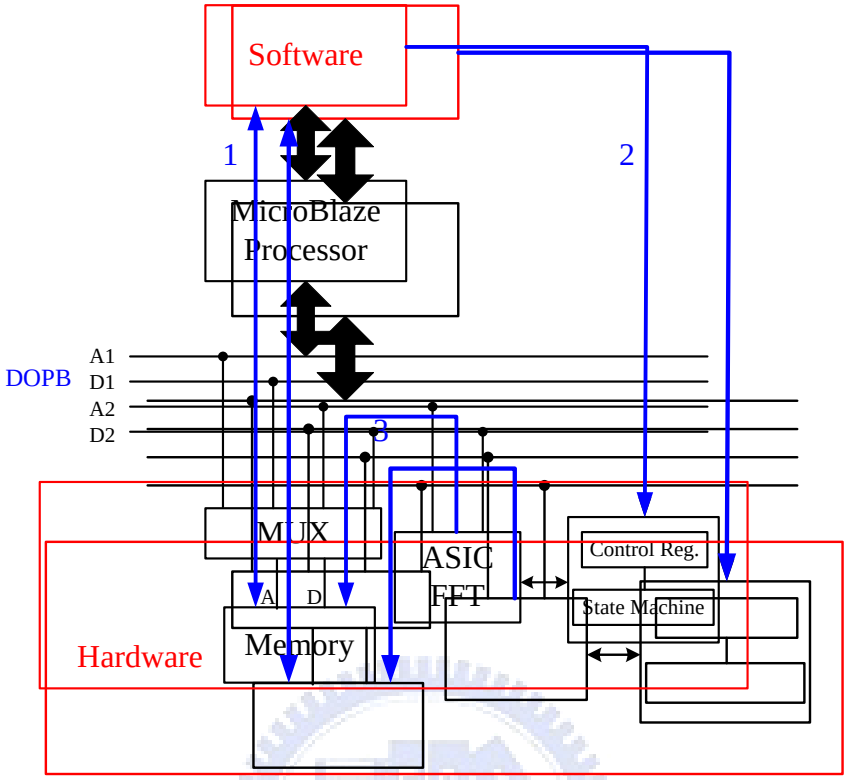


Fig. 3.6 SISO system block diagram of time schedule Ⅰ

In Fig. 3.6, it is the block diagram of SISO Ⅰ, which is designed according to Fig. 3.5 time schedule. In a symbol duration time, there are three steps must be process. First, step 1 is the software part, by reading/writing data between memory and software interface we can accomplish the software part. Second, in order to do the ASIC N-points branch FFT, step 2 is an active signal to execute ASIC FFT that the signal is composed by control register and state machine. Third, step 3 is to do hardware N-points branch FFT. According to these three steps works in one symbol duration the time schedule Ⅰ of Fig. 3.5 will be presented.

3.4.2 SISO System Timing Schedule Ⅱ

Second, in schedule Ⅱ, it makes efforts to raise processor skill and ASIC utilization shown in Fig. 3.7. It not only decreases processor operations per second, but also can cut down the power consumption because of decreasing clock frequency. Additional buffer is

used to increase processor and ASIC processing time up to one symbol duration, but it causes more hardware cost shown in Fig. 3.8.

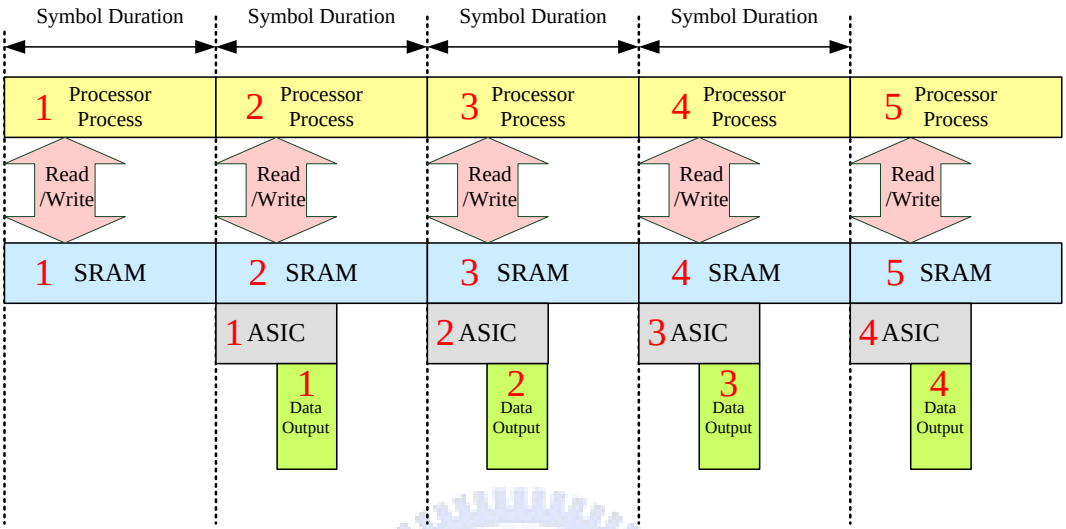


Fig. 3.7 Time schedule Ⅱ of SISO system

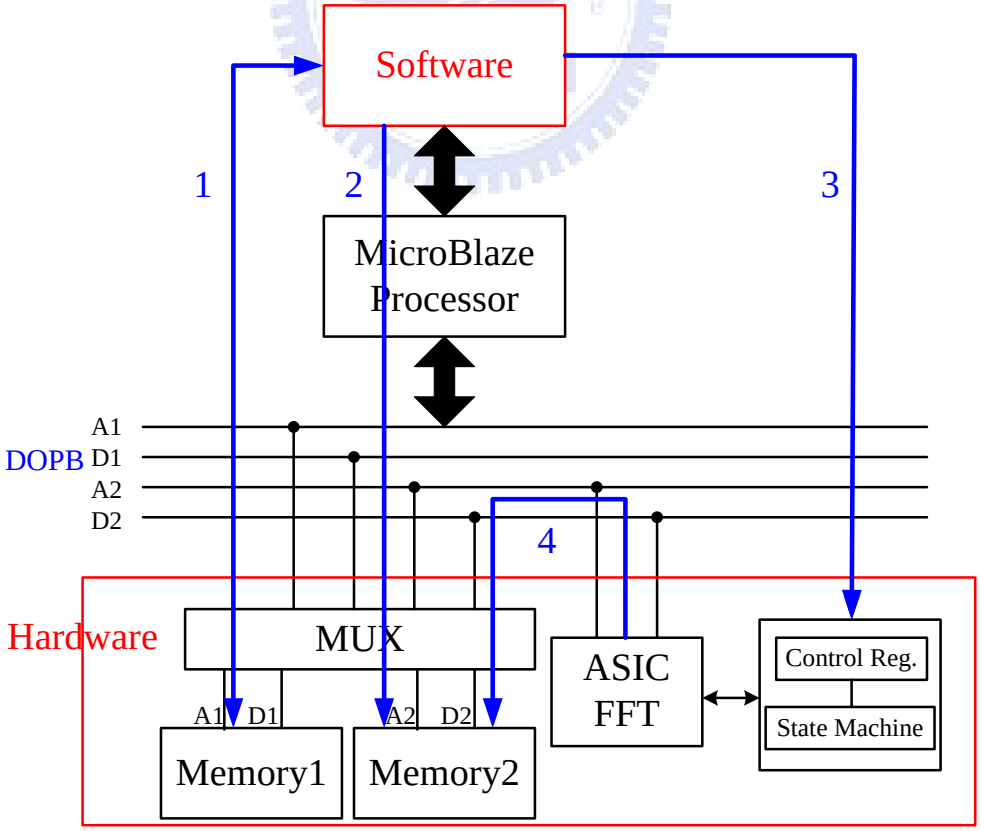


Fig. 3.8 SISO system block diagram of time schedule Ⅱ

In Fig. 3.8, it is the block diagram of SISO Ⅱ, which is designed according to Fig. 3.7 time schedule. In two symbol duration time, there are four steps must be process. First, step 1 is the software part, by reading/writing data between memory 1 and software interface we can accomplish the software part. Step 2, we write the final data of software part to memory 2 because the memory can be used by hardware independently. Third, in order to do the ASIC N-points branch FFT, step 3 is an active signal to execute ASIC FFT that the signal is composed by control register and state machine. Fourth, step 4 is to do hardware N-points branch FFT with memory 2 and when the hardware has work with memory 2, at the same time the processor can return back to memory 1 execute the next event of the next symbol. According to these fourth steps work, step 1 to step 3 works in first symbol duration and step 4 works in the next symbol duration. By several of continuous symbol durations, the time schedule Ⅱ of Fig. 3.5 will be presented. In conclusion, the hardware N-points branch FFT is executed in the next symbol duration. Therefore, SISO Ⅱ not only decreases processor operations per second, but also can cut down the power consumption because of decreasing clock frequency than SISO Ⅰ.

3.4.3 MIMO System Timing Schedule

In general, channel fading can be suppressed by multiple antennas in both transmitter and receiver in MIMO system, but it also increases hardware area dramatically. Therefore, time schedule in MIMO system, it tries to minimize hardware area and enhance processor and ASIC utilization simultaneously. We find that time schedule Ⅱ in SISO system which have many bubbles can be utilized to process others computation. Based on this concept, we proposed a suitable for MIMO system which can eliminate bubbles by processing another antenna's sequences which exchange processor and ASIC processing order as shown in Fig. 3.9.

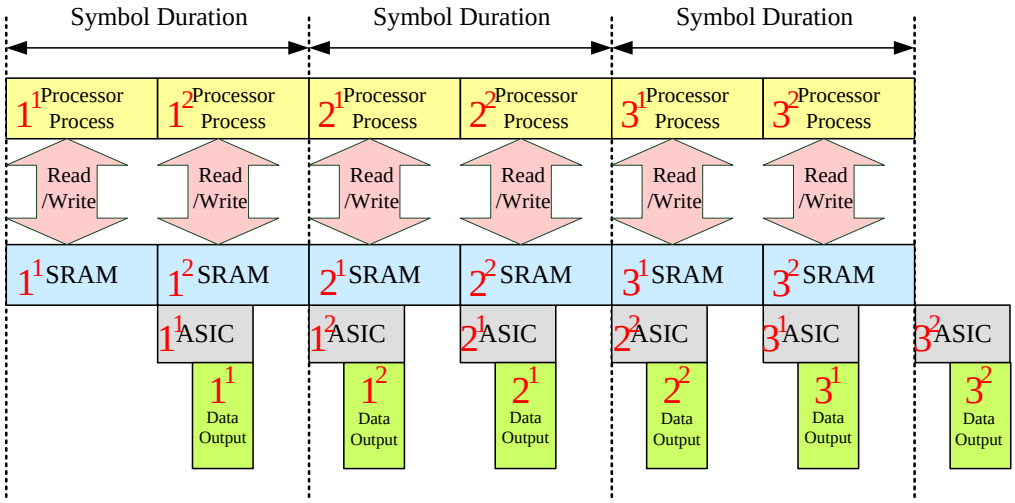


Fig. 3.9 Time schedule of MIMO system

ASIC and processor compute different antenna's sequences by turns within half symbol duration. Therefore, comparison with SISO system, processor need two times operation performance per second in MIMO system. It can process two antenna's sequences simultaneously, and doesn't need additional hardware of branch FFT shown in Fig. 3.10.

In Fig. 3.10, we present the MIMO system block diagram. There four memories for us to execute 2x2 antennas. We use eight steps to perform the MIMO system. First, in the first-half symbol, step 1 is used to operate software FFT of the first antenna and reading/writing data between memory 1 and software. Step 2, we write the final data of software part to memory 2 because the memory can be used by hardware independently. Third, in order to do the ASIC N-points branch FFT of the first antenna, Step 3 is an active signal to execute ASIC FFT that the signal is composed by control register and state machine. Fourth, step 4 is to do hardware N-points branch FFT with memory 2 and when the hardware has work with memory 2, at the same time the processor can change to memory 3 execute the second antenna FFT of the second-half symbol. Fifth, in the second-half symbol, step 5 is used to operate software FFT of the second antenna and reading/writing data between memory 3 and software.

Step 6, we write the final data of software part to memory 4 because the memory can be used by hardware independently. Seventh, in order to do the ASIC N-points branch FFT of the second antenna, Step 7 is an active signal to execute ASIC FFT that the signal is composed by control register and state machine. Eighth, step 8 is to do hardware N-points branch FFT with memory 4 and when the hardware has work with memory 4, at the same time the processor can return back to memory 1 execute the first antenna FFT of the next symbol. In conclusion we perform the Fig. 3.10, step 1 to step 7 works in first symbol duration, which is divided two-half. Step 8 works in the next half symbol.

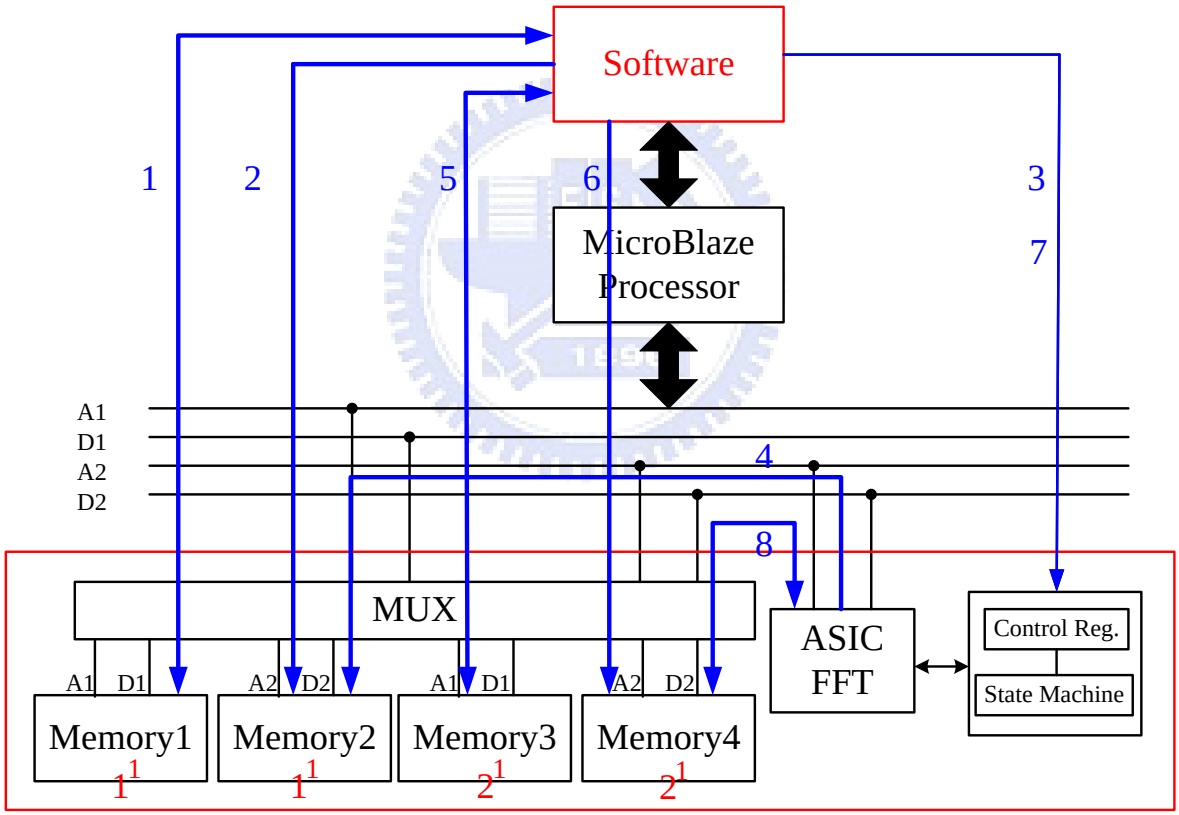


Fig. 3.10 MIMO system block diagram

3.5 ASIC and Processor Performance Estimation

Since SISO schedules are proposed, an evaluation model is developed to verify specification requirements. Bases on IEEE 802.11n/16e standards, we can introduce symbol period to calculate the performance of processor when different length of FFT is implemented by hardware. ASIC plays an accelerative role in the system. Increasing ASIC length of branch FFT can release load of processor. Not only 64-points/2048-points are the critical case in IEEE 802.11n/16e separately but also we will show the other cases, such as 128-points, 512-points, 1024-points FFT.

Schedule I in SISO system, ASIC occupies some symbol duration shown in Fig. 3.5. Therefore, we need to calculate ASIC latency cycles approximately shown in Table 3.7 [29] and assume clock frequency is 50MHz for simulation, according to Fig. 3.4: “Number of Processor Operations”, we will show some cases of different length FFT shown in Fig. 3.11 ~ 3.15. In Table 3.7 we can make sure that the ASIC latency time can be included in a symbol duration time unit. When FFT length of ASIC is too short, it cannot gain any benefit to the processor. FFT length of ASIC affects operations of processor directly. More length branch FFT implemented by hardware will lower processor’s operations, but it increases cost.

Table 3.7 Approximately calculation of latency cycles

FFT Length	Latency	FFT Length	Latency
0	0	64	103
2	2	128	208
4	4	256	336
8	8	512	592
16	26	1024	1616
32	44	2048	2640

Based on IEEE 802.11n/16e standards, the symbol duration of 64-points is 3.2μs and 2048-points is 102.5μs. In Table 3.7, the latency cycles of ASIC will be enough to be included in a symbol duration time unit, if the we choose 85 Mhz for out throughput rate at least. The throughput rate is shown in Fig. 3.11. It shows all the time schedules of N-points FFT based on IEEE standards, the throughput rate of 128-points is up to 85 Mhz at least. If we chose 85 Mhz for our throughput rate, we will know that this throughput rate it can be included in a symbol duration for all kinds of N-points branch FFT according to our time schedules, SISO Ⅰ, SISO Ⅱ and MIMO.

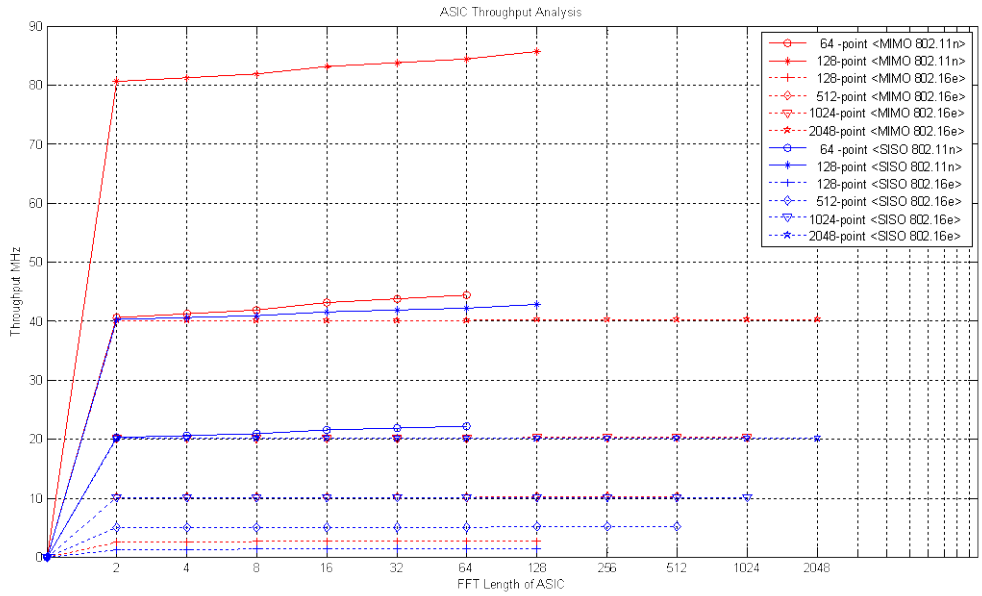


Fig. 3.11 ASIC throughput analysis

3.5.1 SISO Ⅰ System Operation Comparison

From Fig. 3.12 ~ 3.16, we can analyze the relationship between processor operations and branch FFT of ASIC. MOPS (Million Operations per Second) imply that processor operations divided by not needed symbol duration. When our system processes FFT algorithm only by processor, it shows that IEEE 802.11n need more operations per second. Therefore, we can calculate processor's performance probably by MOPS. The time schedule diagram is based on Fig. 3.5.

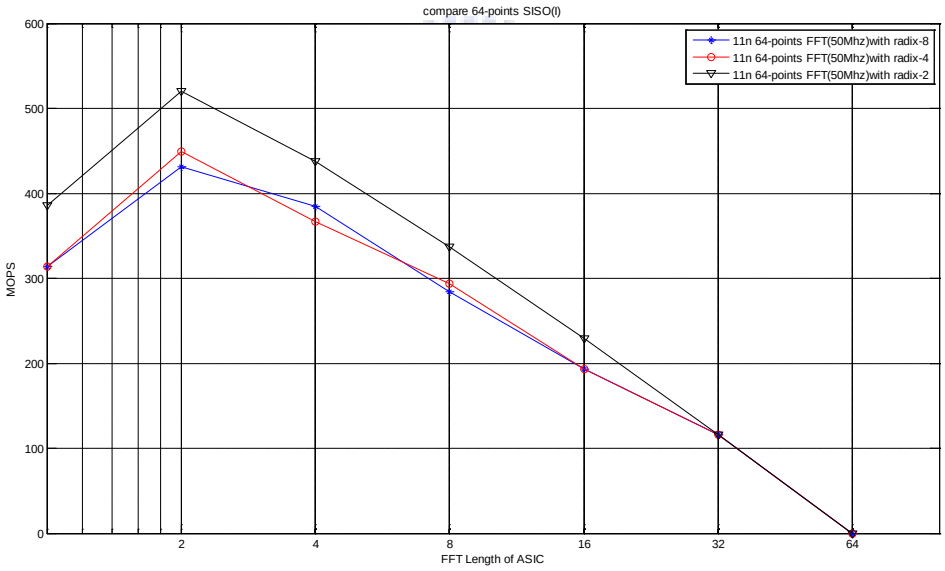


Fig. 3.12 64-points FFT operation comparison of time schedule Ⅰ

In Fig. 3.12, it shows the 64-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.11n standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. Take an example, if the processor just only can provide us 300 MOPS, we will choose 8-points branch FFT as our ASIC because the two types of radix-4 and radix-8 algorithms are satisfy with the required MOPS (under 300 MOPS).In the other word, the radix-2 algorithm will be not satisfy the required 300 MOPS (over 300 MOPS), if we choose 8-points branch FFT as our ASIC.

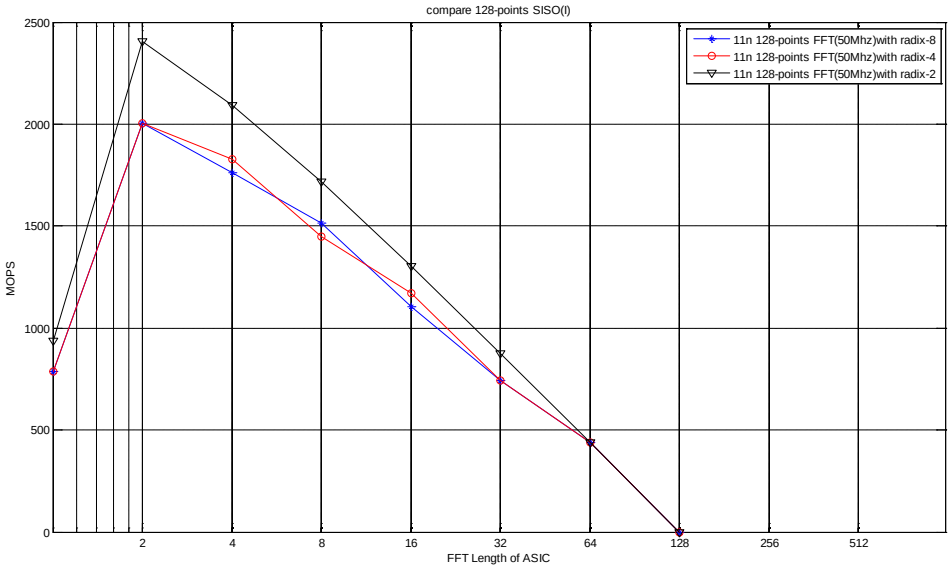


Fig. 3.13 128-points FFT operation comparison of time schedule Ⅰ

In Fig. 3.13, it shows the 128-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.11n standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. According to this figure, we can know that the radix-4 and radix-8 algorithms are more suitable than radix-2 algorithm because radix-2 algorithm will cost more MOPS then the other two types.

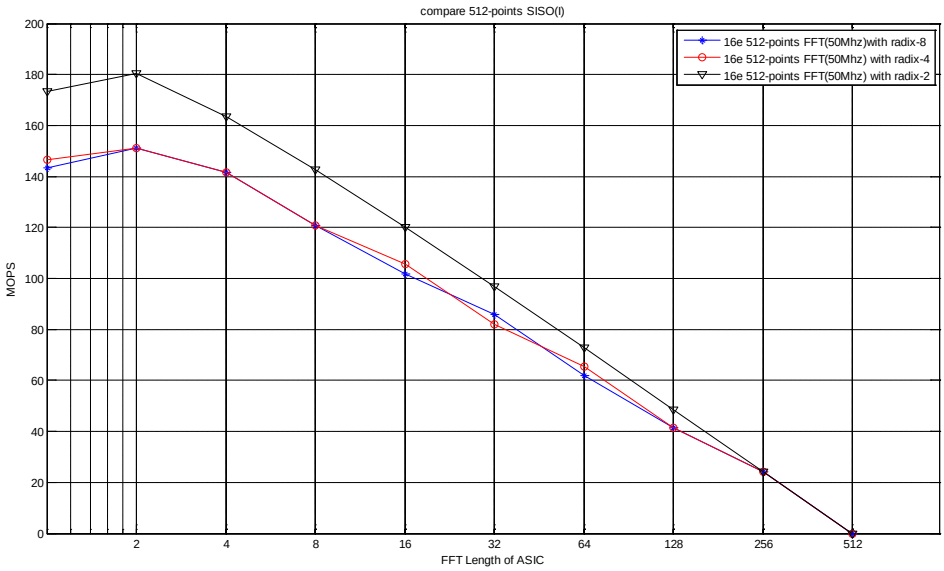


Fig. 3.14 512-points FFT operation comparison of time schedule Ⅰ

In Fig. 3.14, it shows the 512-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. According to this figure, we can know that the radix-4 and radix-8 algorithms are more suitable than radix-2 algorithm because radix-2 algorithm will cost more MOPS then the other two types.

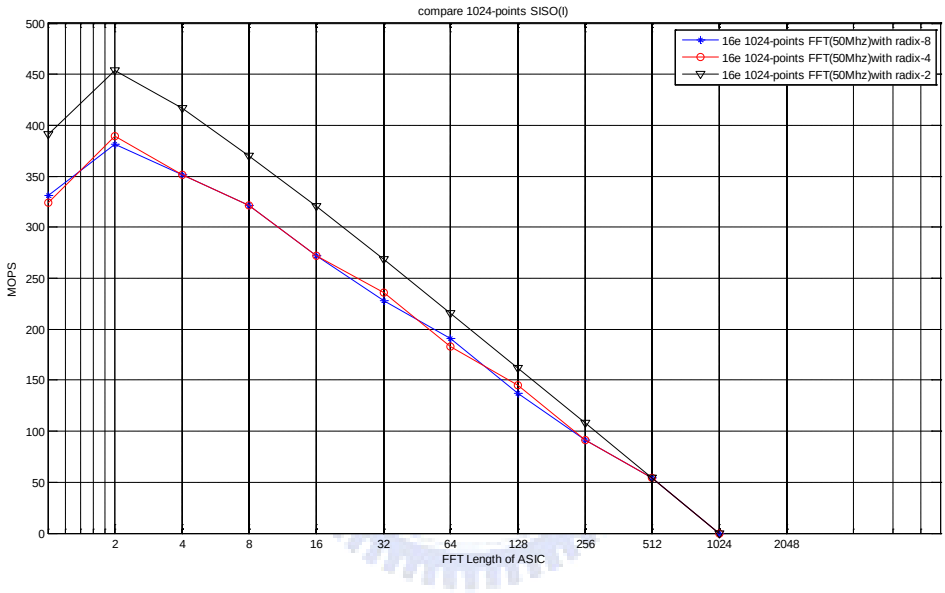


Fig. 3.15 1024-points FFT operation comparison of time schedule I

In Fig. 3.15, it shows the 1024-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. According to this figure, we can know that the radix-4 and radix-8 algorithms are more suitable than radix-2 algorithm because radix-2 algorithm will cost more MOPS than the other two types.

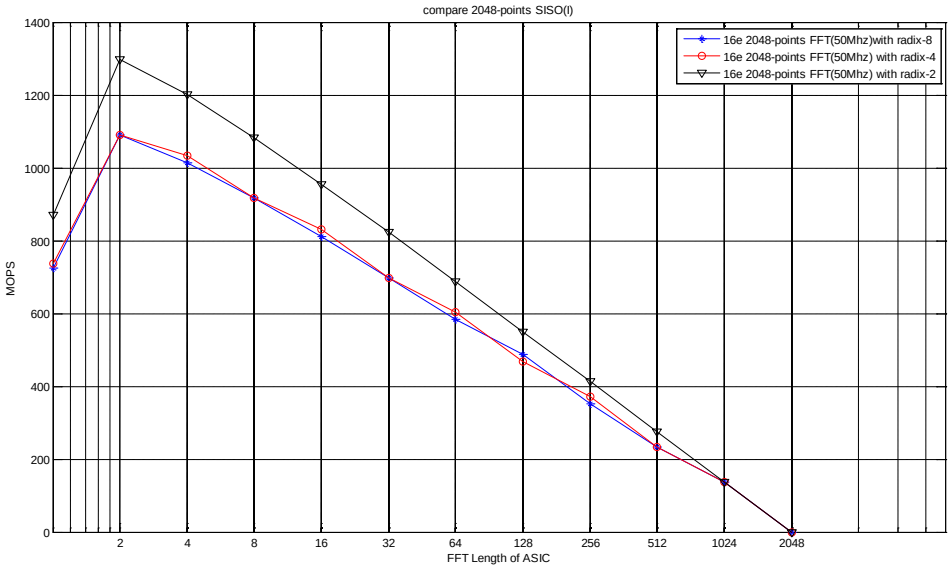


Fig. 3.16 2048-points FFT operation comparison of time schedule Ⅰ

In Fig. 3.16, it shows the 2048-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. According to this figure, we can know that the radix-4 and radix-8 algorithms are more suitable than radix-2 algorithm because radix-2 algorithm will cost more MOPS then the other two types. In the other word, for N-points FFT we decide to implement, we can consider radix-8 and radix-2 algorithms first that the performance is better than only radix-2 algorithm.

In schedule Ⅰ of Fig. 3.12 ~ 3.16, user can design the system which we want. For example in IEEE 802.16e standard , if we want the processor used only for 700 MOPS, we will chose the “2048-points FFT operation comparison of time schedule Ⅰ” method of Fig. 3.16, which we just use the 64-points ASIC FFT to design the system of this standard.

3.5.2 SISO Ⅱ System Operation Comparison

Schedule Ⅱ in SISO system shown in Fig. 3.17 ~ 3.21, it not only decreases processor operations per second than but also can cut down the power consumption. Therefore, the cost of MOPS in Schedule Ⅱ is lese than schedule Ⅰ. The time schedule diagram is based on Fig. 3.7.

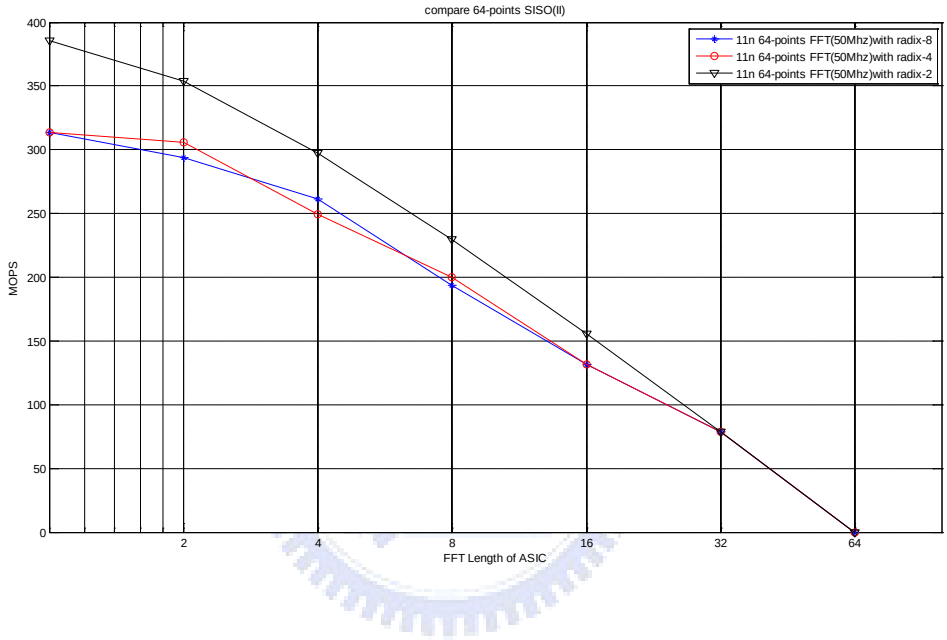


Fig. 3.17 64-points FFT operation comparison of time schedule Ⅱ

In Fig. 3.17, it shows the 64-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.11n standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule Ⅱ of SISO Ⅱ system. The performance is better than SISO Ⅰ system of Fig. 3.12. Take an example, if the processor provides us for 300 MOPS, we can choose the 4-points branch FFT of ASIC in SISO Ⅱ not the 8-points branch FFT of ASIC in SISO Ⅰ. Therefore, the cost of ASIC in SISO Ⅱ will be changed smaller than SISO Ⅰ, if the processor only provides 300 MOPS.

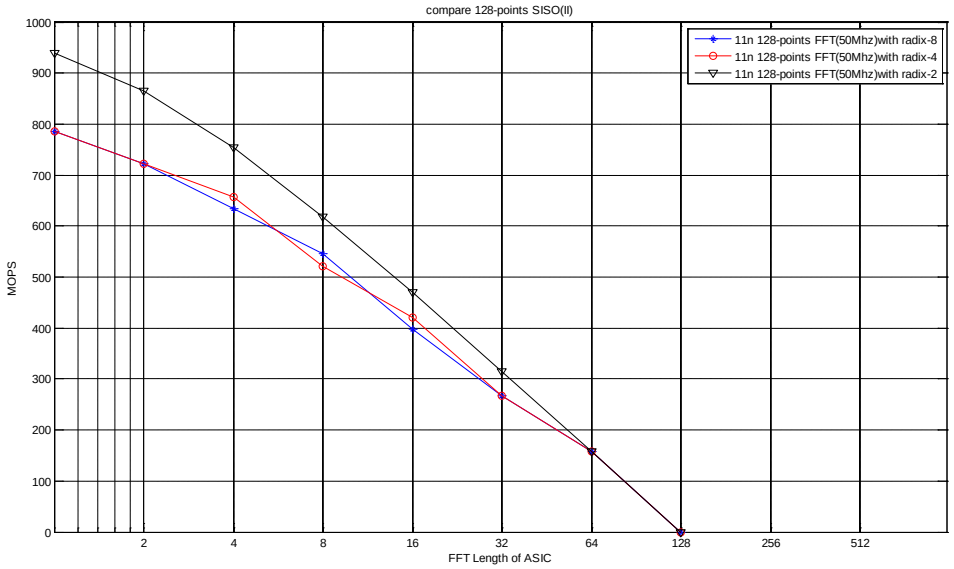


Fig. 3.18 128-points FFT operation comparison of time schedule Ⅱ

In Fig. 3.18, it shows the 128-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.11n standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule Ⅱ of SISO Ⅱ system. The performance is better than SISO Ⅰ system of Fig. 3.13.

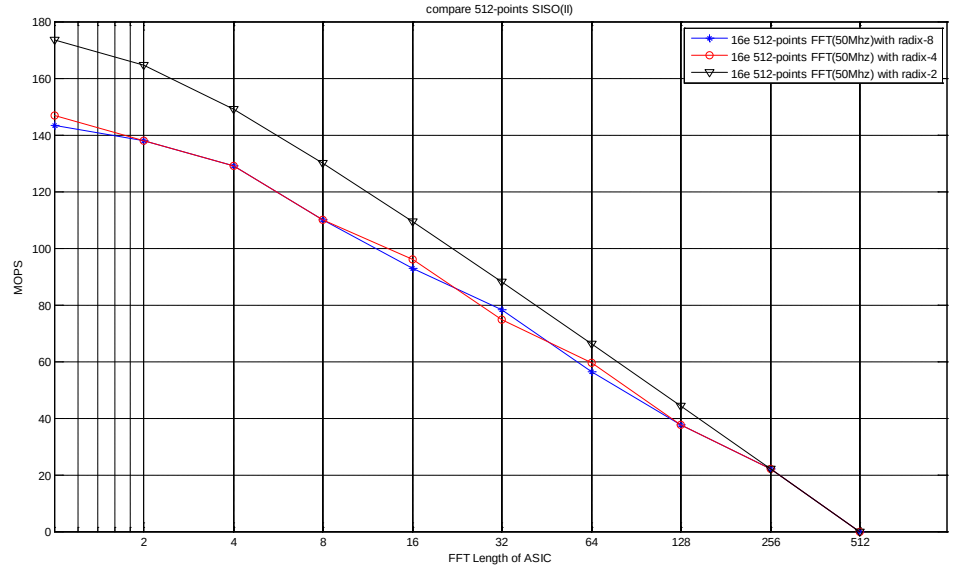


Fig. 3.19 512-points FFT operation comparison of time schedule Ⅱ

In Fig. 3.19, it shows the 512-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule Ⅱ of SISO Ⅱ system. The performance is better than SISO Ⅰ system of Fig. 3.14.

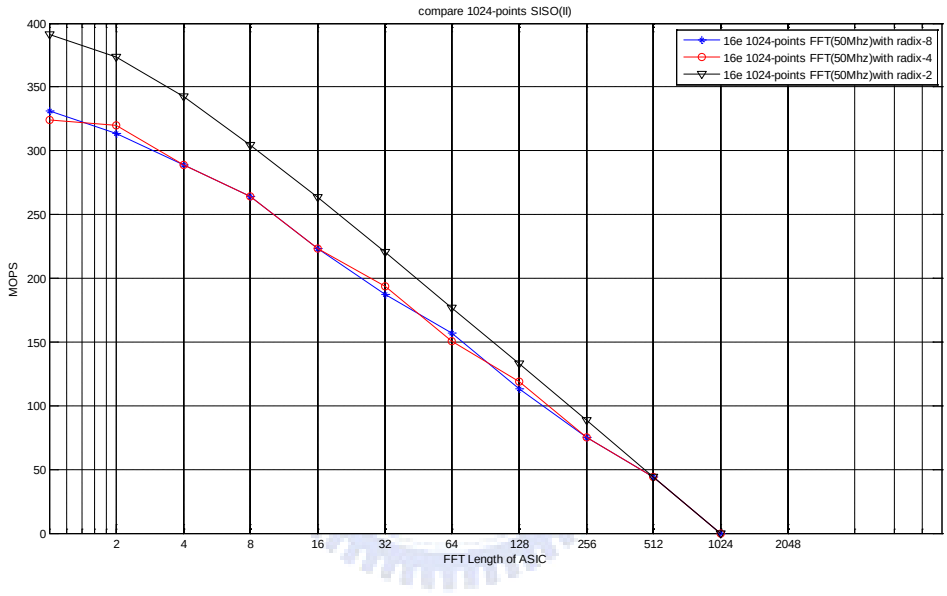


Fig. 3.20 1024-points FFT operation comparison of time schedule Ⅱ

In Fig. 3.20, it shows the 1024-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule Ⅱ of SISO Ⅱ system. The performance is better than SISO Ⅰ system of Fig. 3.15.

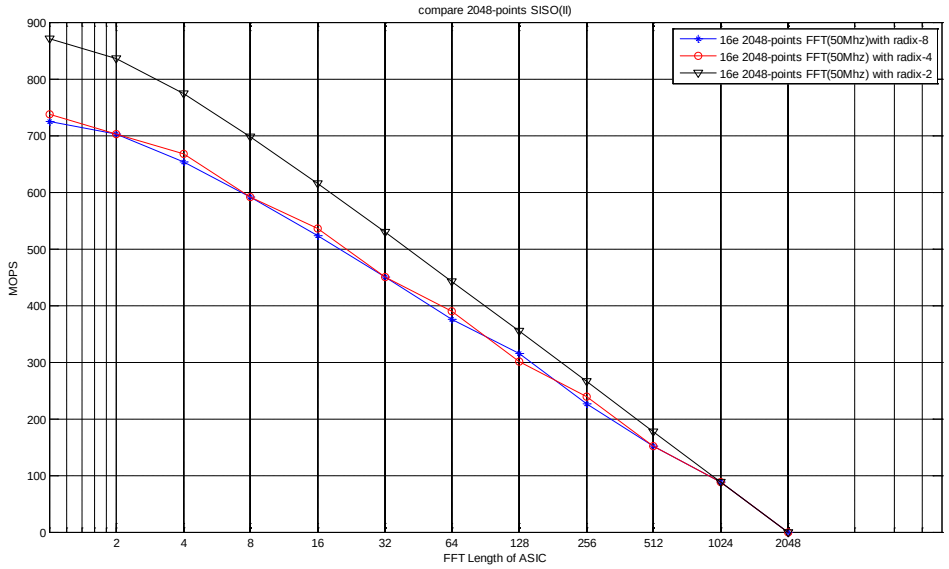


Fig. 3.21 2048-points FFT operation comparison of time schedule Ⅱ

In Fig. 3.21, it shows the 2048-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule Ⅱ of SISO Ⅱ system. The performance is better than SISO Ⅰ system of Fig. 3.16.

In "2048-points FFT operation comparison of time schedule Ⅱ" method of Fig. 3.21 based on IEEE 802.16e standard, if we chose 64-points ASIC FFT, the processor operations will be used just only about 450 MOPS.

3.5.3 MIMO System Operation Comparison

Therefore, in MIMO system, processor and ASIC own half a symbol duration to complete operations. It can be expected that processor's operations per second will be doubled of schedule Ⅱ in SISO system as shown in Fig. 3.22 ~ 3.26 based on time schedule diagram

shown in Fig. 3.9.

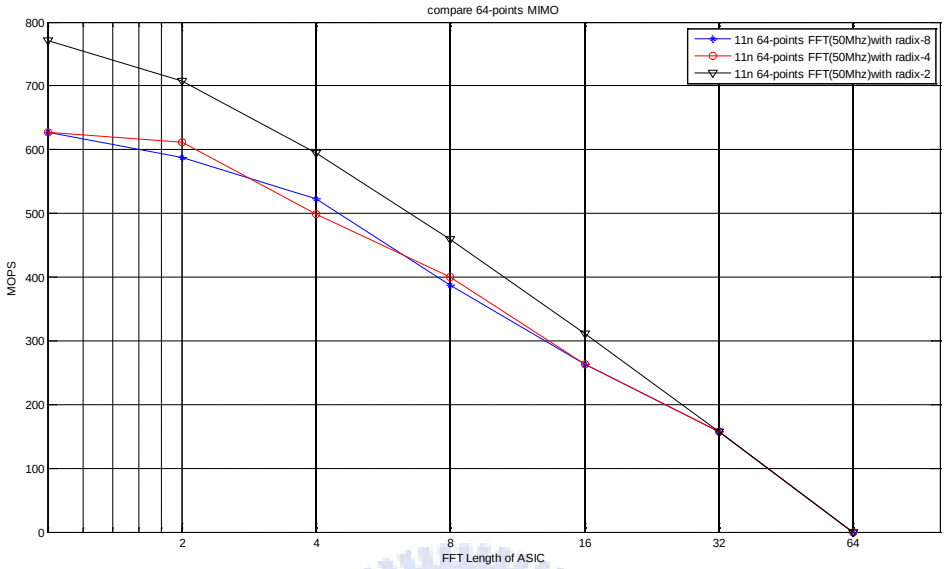


Fig. 3.22 64-points FFT operation comparison of MIMO time schedule

In Fig. 3.22, it shows the 64-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.11n standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule of MIMO system. The performance is twice than SISO Ⅱ system of Fig. 3.17, but the usage of time schedule in a symbol is improved. In a symbol duration, we can execute FFT two times and just only use one N-points branch FFT of ASIC well.

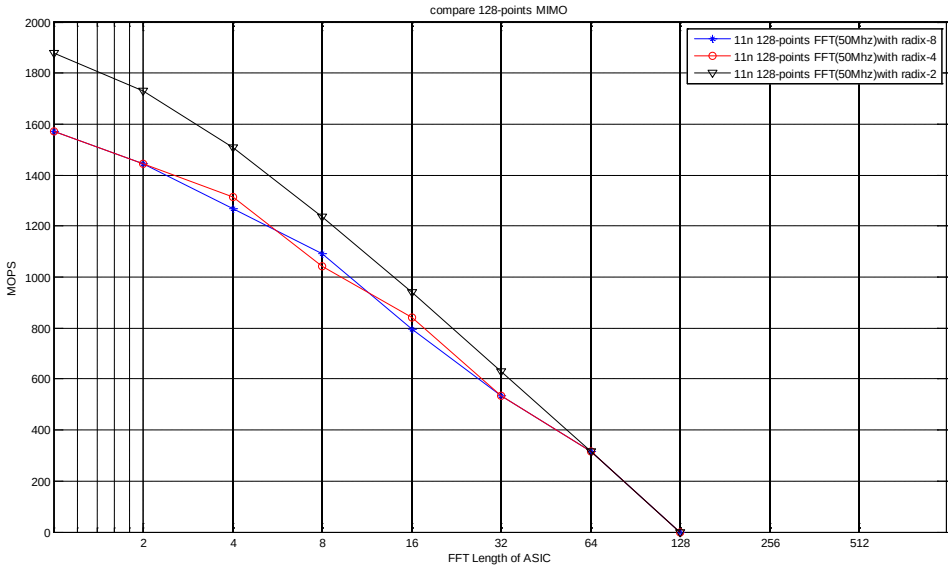


Fig. 3.23 128-points FFT operation comparison of MIMO time schedule

In Fig. 3.23, it shows the 128-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.11n standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule of MIMO system. The performance is twice than SISO Ⅱ system of Fig. 3.18, but the usage of time schedule in a symbol is improved. In a symbol duration time unit, we can execute FFT two times and just only use one N-points branch FFT of ASIC well.

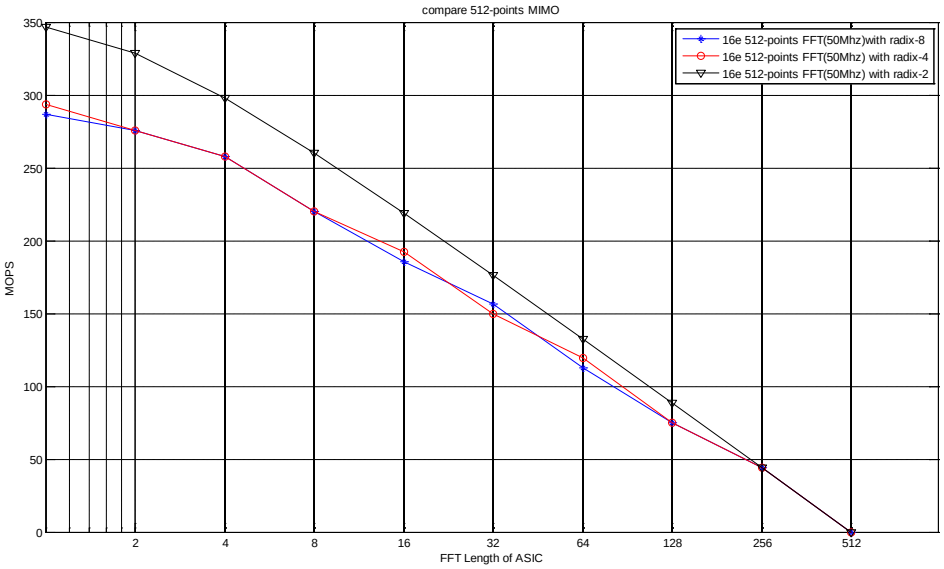


Fig. 3.24 512-points FFT operation comparison of MIMO time schedule

In Fig. 3.24, it shows the 128-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule of MIMO system. The performance is twice than SISO Ⅱ system of Fig. 3.19, but the usage of time schedule in a symbol is improved. In a symbol duration time unit, we can execute FFT two times and just only use one N-points branch FFT of ASIC well.

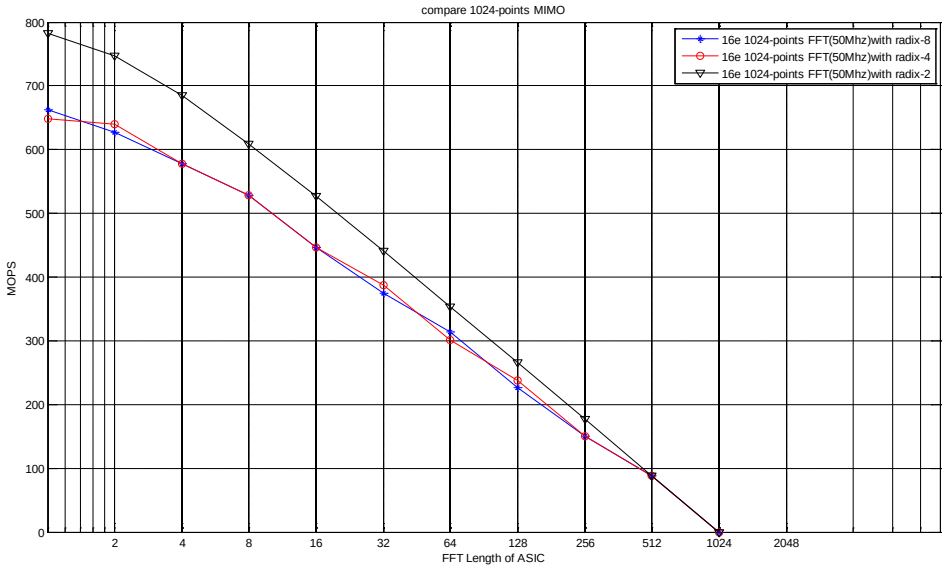


Fig. 3.25 1024-points FFT operation comparison of MIMO time schedule

In Fig. 3.25, it shows the 1024-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule of MIMO system. The performance is twice than SISO Ⅱ system of Fig. 3.20, but the usage of time schedule in a symbol is improved. In a symbol duration time unit of 2x2 antennas, we can execute FFT two times and just only use one N-points branch FFT of ASIC well.

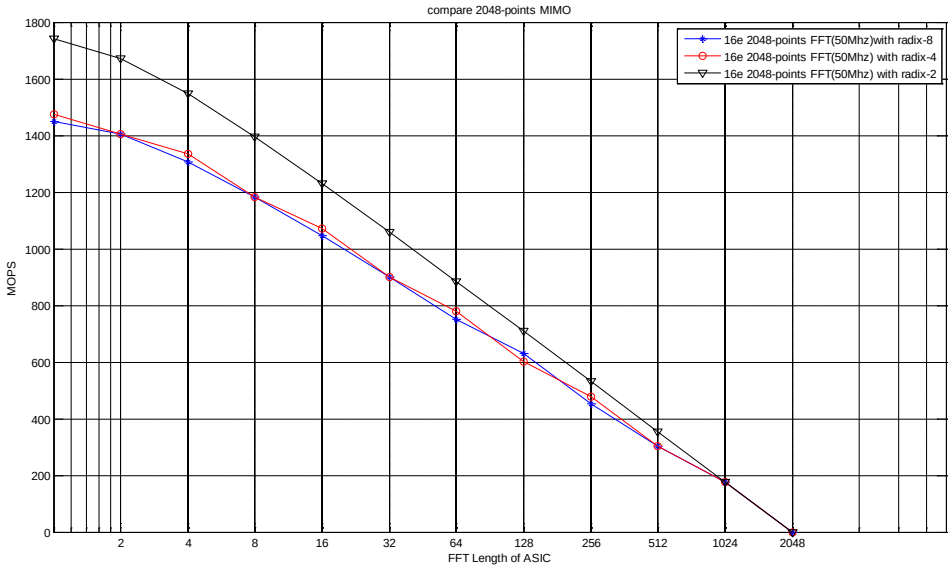


Fig. 3.26 2048-points FFT operation comparison of MIMO time schedule

In Fig. 3.26, it shows the 2048-points FFT MOPS of radix-2, radix-4 and radix-8 based on IEEE 802.16e standard, x-axis is what kind of branch ASIC FFT we will decide, if the processor provide us some restricted MOPS to use shown on y-axis. This figure is designed according to time schedule of MIMO system. The performance is twice than SISO Ⅱ system of Fig. 3.21, but the usage of time schedule in a symbol is improved.

In the method of Fig. 3.26: 2048-points FFT operation comparison of MIMO time schedule. IEEE 802.16e standard, if we chose 64-points ASIC FFT, the processor operations will be used about 900 MOPS. In the other word, the MOPS of MIMO schedule will be doubled than the time schedule Ⅱ of SISO Ⅱ.

In this section, we introduce schedule Ⅰ、Ⅱ in SISO system, and a schedule in MIMO system. According to time schedule system based on time schedule Ⅰ of SISO Ⅰ and time schedule Ⅱ of SISO Ⅱ, the cost of hardware in schedule Ⅰ is less than schedule Ⅱ, but the utilization of schedule Ⅰ is less than schedule Ⅱ. In MIMO system, bad utilization can be improved by changing ASIC and processor order. In 2x2 MIMO systems, it only needs a pro-

cessor and a branch FFT of ASIC. This schedule not only lower hardware cost, but also increase the utilization of module.

Therefore, we use the equation (2) to calculate the “Processor Operations” of 64-points, 128-points, 512-points,1024-points and 2048-points FFT and according to IEEE 802.11n/16e standards, we can predict the time of these three schedules (SISO**I**,SISO**II** and MIMO) in a symbol duration. Finally, MOPS (Million Operations per Second) has been evaluated by our estimation. In next chapter, we want to implement the processor’s architecture of SISO Ⅰ in Fig. 3.6, SISO Ⅱ in Fig. 3.8 and MIMO in Fig. 3.10 with “Micro Blaze Processor”, which is a embedded system implemented by FPGA tools

Chapter 4
Implementation of the Structure with MicroBlaze Processor

4.1 Introduction of the MicroBlaze Processor

In the implementation domain of FFT processor, we chose the “MicroBlaze embedded system” which is implemented by FPGA tools. The MicroBlaze embedded soft core [30] is a reduced instruction set computer (RISC) optimized for implementation in Xilinx field programmable gates arrays (FPGAs) [31]. Fig.4.1 is a block diagram depicting the MicroBlaze core.

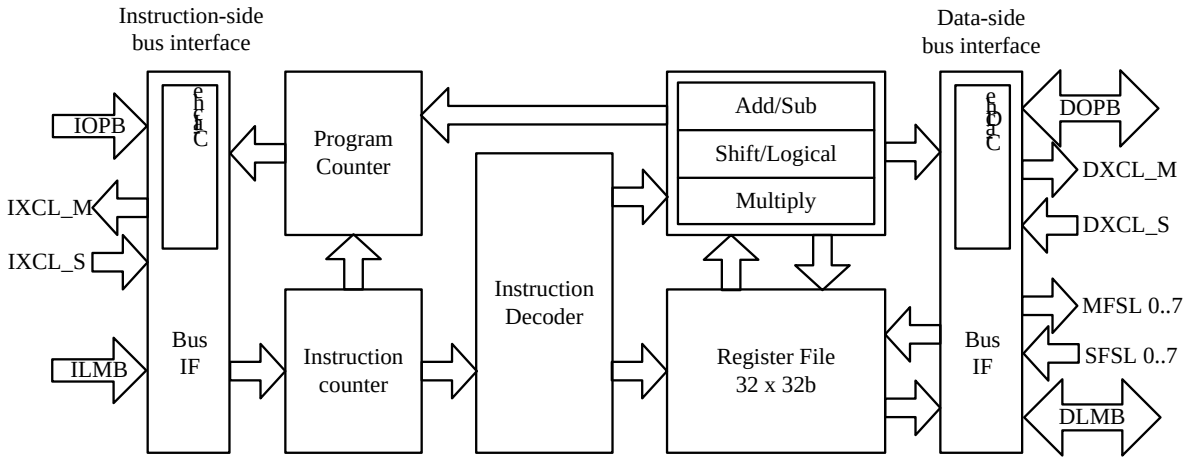


Fig. 4.1 MicroBlaze core block diagram

From Fig. 4.1 that the MicroBlaze embedded soft core is highly configurable, allowing users to select a specific set of features required by their design. The processors features set includes the following. There are twenty-two 32-bits general purpose registers, 32-bits in-

struction word with three operands and two addressing modes, separate 32-bits instruction and data buses that conform to IBM's OPB (On-chip Peripheral Bus) specification [28], separate 32-bits instruction and data buses with direct connection to on-chip block RAM through a LMB (Local Memory Bus), 32-bits address bus and instruction/data cache, single issue pipeline, hardware debug logic, Fast Simplex Link (FSL) support, hardware multiplier, hardware exception handling, Dedicated Cache Link interface for enhanced cache performance.

4.2 Implementation of the Variable-Length FFT

In the environment of the MicroBlaze processor, we desire to complete the three time schedules we have evaluated in section 3.4: "ASIC and Processor Timing Schedule Analysis". The three time schedules are SISO Ⅰ, SISO Ⅱ and MIMO time schedules we discussed in Chapter 3, which we emphasize to design with the processor's tool of MicroBlaze in this section. By using the tool of MicroBlaze processor, we can real know the working situations between hardware and software based on FFT algorithm.

Therefore, we separate two parts of variable-length FFT, one is work on processor and the other is executed by ASIC FFT. According to IEEE 802.11n/16e standards, we take the implementation of 64-points FFT for an example. First, we do software part on processor shown in Fig. 4.2 by radix-2^3 FFT, then complete the remaining part on 8-points ASIC FFT for eight times. Finally, we integrate these two parts: software and hardware, which we can

succeed the embedded system core of hardware and software FFT on MicroBlaze processor.

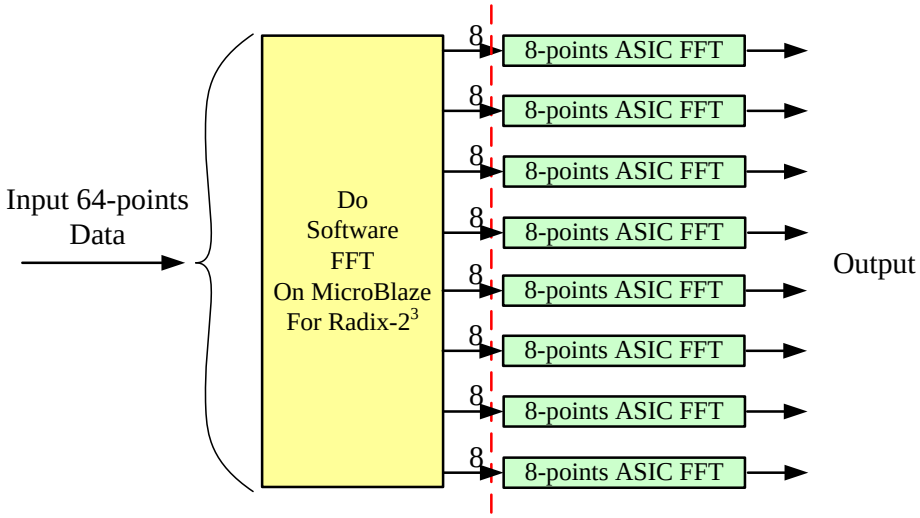


Fig. 4.2 Separate the implementation of 64-points FFT for two parts

By the flow we integrate the software and hardware implementation on the example of 64-poins FFT according to IEEE 802.11n/16e standards. That is just one situation of our desired cases such as 128-points, 512-points, 1024-points and 2048-points FFT that we have evaluated the all time schedules in section 3.4 are our desired cases too.

Therefore, in the next 4.2.1, 4.2.2 and 4.2.3 sections we can show how we implement the example of 64-points FFT, we can learn the concepts and information in Co-Design on ASIC and Processor of FFT.

4.2.1 Software Design on MicroBlaze Processor

As the following paragraph, we want to design the software parts of variable-length N-points FFT. From the input data we change the real floating number into fixed 16-bits data. Thus, we can store all the 16-bits of N-points FFT input data in user's memory block on MicroBlaze processor. On programming interface in Fig. 4.3 we use the data to read/write in order to perform FFT butterfly. First, we read the initial input data(x[n], x[n+1],…, x[n+7N/8])

of FFT from memory that we use the data to do butterfly operation shown in Fig. 4.4. Second, we write the calculated data into memory. According to these two steps, the programming part of variable-length FFT will be implemented easily.

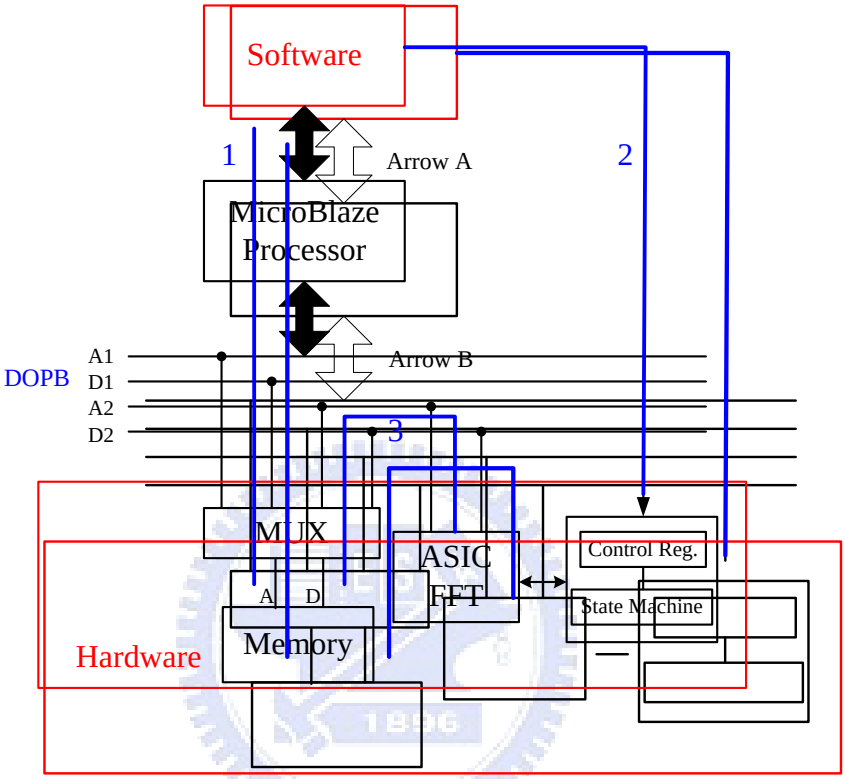


Fig. 4.3 Structure of MicroBlaze Processor's programming diagram

In Fig. 4.3, the big Arrow A and B are the IOPB [28] programming interface, which is used to communicate the processor with Master signals and Slave signals. Master signals and Slave signals are the data buses of DOPB which can process data through SRAM (Memory). Blue Line 1, 2 and 3 are the programming order steps which in step 1 we do programming part of software FFT, in step 2 the MicroBlaze processor will send an active message to the Control Register which the message is used to tell the hardware part start to work. Finally, in step 3 the ASIC FFT of hardware part will execute and catch data from SRAM (Memory). According to these three steps we can perform the programming implementation of software part FFT environment.

From the programming environment, we use the radix-2^3 DIF FFT architecture shown

in Fig. 4.4 to implement the software part of 64-points FFT of Fig. 4.2. Finally, we write the first stage data in memory block after performing the software programming architecture. In the next section 4.2.2, we will use the data of memory block to process the remaining ASIC FFT part.

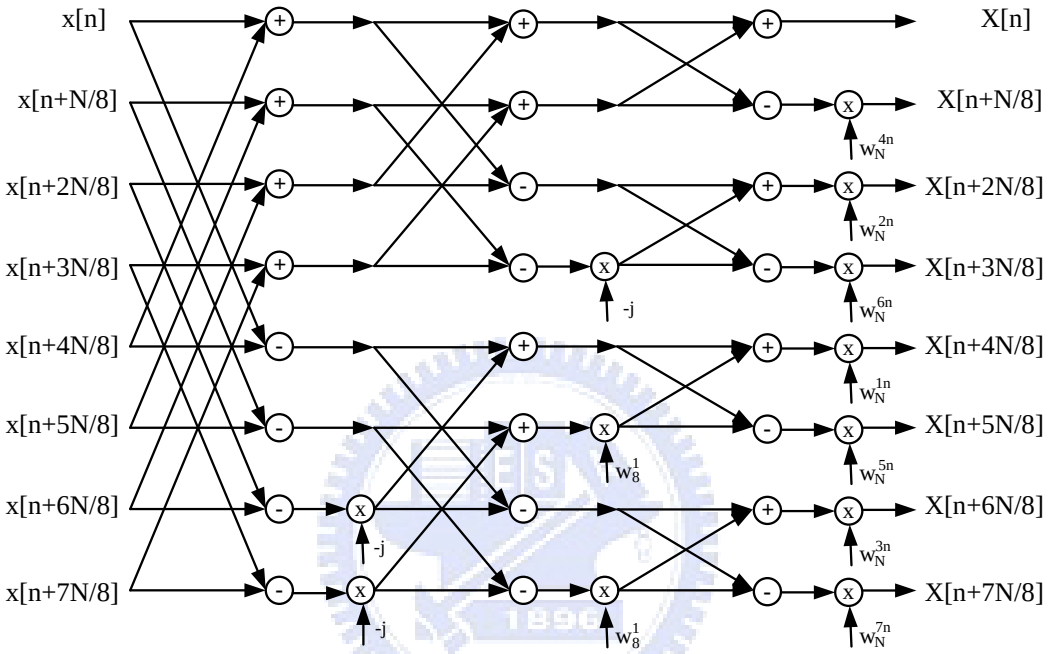


Fig. 4.4 The butterfly signal flow graph of the radix-2^3 DIT FFT algorithm

4.2.2 Hardware Design on ASIC FFT

We want to implement the hardware part immediately in order to connect the remaining part of software FFT from the previous section 4.2.1. In this section, we implement a simple ASIC for 8-points FFT by radix-2^3 DIT FFT algorithm. Thus, we can integrate the whole process of the 64-points FFT architecture that the example we take as shown in Fig. 4.2.

Because of the FFT ASIC is just simple 8-points FFT, we process it with pipeline flow that we transmit 8-points input data at the same time and we receive all 8-points output at the next same time. The 8-points FFT is based on one stage radix-2^3 butterfly and it needs 3 times complex multiplications exclude from W_8^1, W_8^3 and W_8^0. According to Fig. 4.2 we must do

8-points FFT ASIC for eight times that N is 8. We use the radix-2 index map to divide the 8-points DFT into three steps. From Fig.4.4 it shows the butterfly of the three-step DIF radix-8 FFT. The twiddle factors, W_8^1 and W_8^3 at the second step are trivial complex multiplication, because they can be written as $\sqrt{2}/2(1-j)$ and $\sqrt{2}/2(-1-j)$. Thus, a complex multiplication with one of the two coefficients and a real multiplication, whose hardware can be realized by shifters and adders shown in Fig. 4.5.

$$\sqrt{2}/2 = 0.70710678 = 2^{-1} + 2^{-3} + 2^{-4} + 2^{-6} + 2^{-8} + 2^{-9}$$

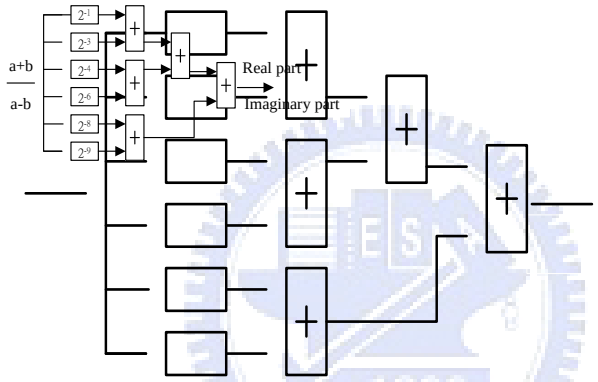


Fig. 4.5 Implementation hardware of multiplication with $\sqrt{2}/2$

The multiplication by –j can be realized with no extra hardware cost by simply interchanging the real and imaginary part of the product as shown in equation (1).

$$(a+bj)\times(-j) = b - aj \quad (1)$$

One complex multiplier can be realized by four real multiplications and two real additions as shown in Fig. 4.6. Its mathematical form can be expressed as equation (2).

$$(a+bj)\times(c+dj) = (ac-bd) + j(bc+ad) \quad (2)$$

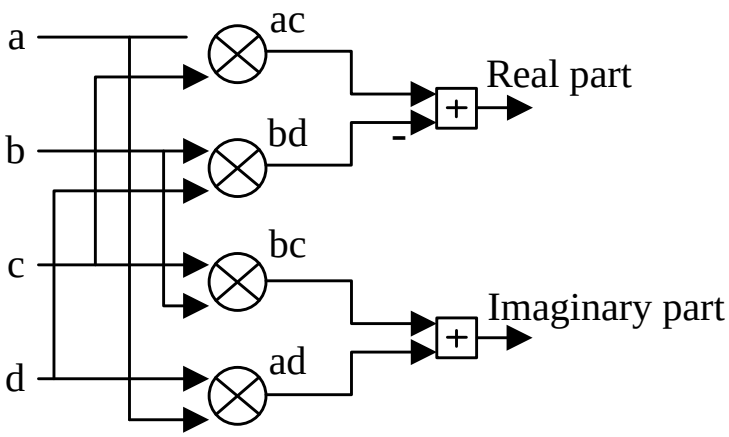


Fig. 4.6 Complex multiplier with four real multiplications and two real additions

Therefore, the hardware block of ASIC FFT architecture based on radix-2^3 DIT FFT algorithm of Fig. 4.4 will be implemented which the real part and image part operations are design according to Fig. 4.5 and Fig. 4.6.

In the next section 4.2.3, we will integrate the software and hardware part clearly, which we will tell the readers how we integrate the FFT algorithm with software and harware FFT block that we would add control register, state machine, memory block (SRAM), memory mapped addressing mode and the simulation figure of result by using the MicroBlaze processor core.

4.2.3 Integrate the Embedded System

From the introduction of software design and ASIC FFT according to section 4.2.1 and 4.2.2, we desire to integrate the embedded system based on MicroBlaze Processor shown in Fig. 4.7. In this embedded environment we run programming on the processor and then execute the ASIC FFT to process data in memory block. Thus, we can accomplish the FFT implementation of hardware and software embedded system design. In the following paragraph we will detailed describe each block of Fig. 4.7 such as control register, state machine, and memory block (SRAM).

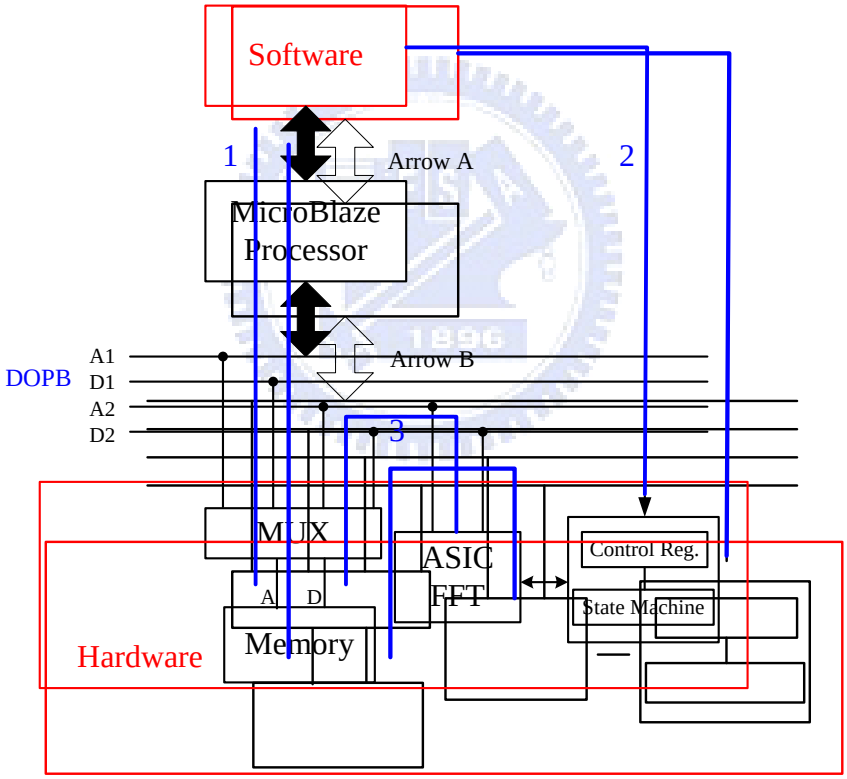


Fig. 4.7 Integrate the embedded system design diagram

In Fig. 4.7, the MUX circuit is used to switch memory block sharing between processor and ASIC, in software environment it can be changed to A1 and D1.If we switch the MUX to A2 and D2, hardware environment will be executed. Fig. 4.8 shows the memory block (SRAM) structure of Fig. 4.7 which is a dual-port memory library of Xilinx field programma-

ble gates arrays (FPGAs) named RAMB16_S36_36 whose size are 512x32 bits. Part A is used to connect with processor and port B is hardware part connecting. If we chose software part programming environment, the MUX will switch to A1 and D1 connecting. In the other word, if the MUX switched to A2 and D2, the hardware part connecting has been executed. The signal ENA/B, SSRA/B and WEA/B are control signals that we can decide READ/WRITE command of the memory block [32], [33], [34], [35].

Because of the size memory block are total 16384 bits, the memory mapped addressing range we can add the hardware on MicroBlaze embedded system is from 0x01800000 to 0x018003FF (0x000~0x3FF are total 16384 bits) and the hardware offset is 0x0180000. Since the mapped addressing of MicroBlaze processor is free defined by users, we can not only put the memory block on range 0x01800000 but also can put it in other offsets as long as two hardware devices are not in conflict with the same addressing range .

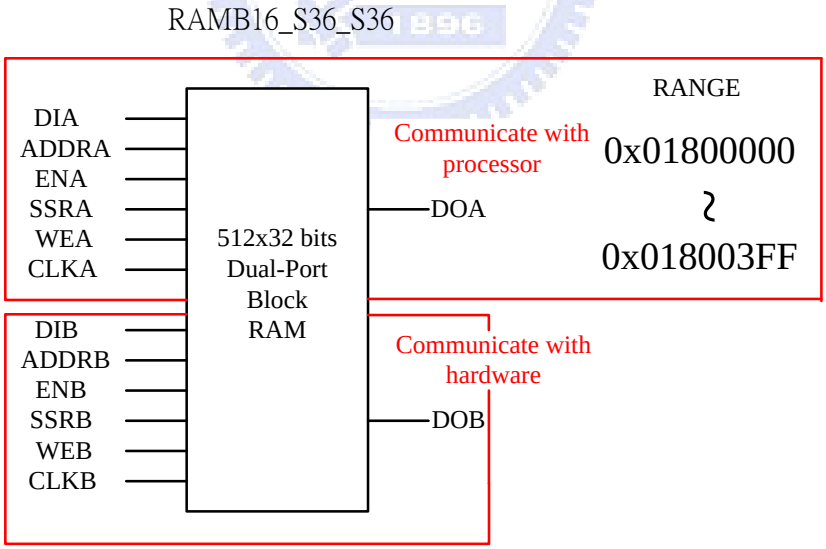


Fig. 4.8 The block of Dual-Port RAM (SRAM)

In Fig. 4.9 is the Control Register of Fig. 4.7, which we can define what address of memory block in Fig. 4.8 we start to work. If we write address on Write register, the hardware device will go to the memory mapped address and start to catch memory data which at the same time the hardware device is preparing to work. In the step we write address on Write

register that the hardware device will send an input enable signal to tell us that the hardware starts on working. If the hardware device finishes work, it will send an output enable signal which we can use Read register to read the output enable signal to make sure how the hardware device works successful. These two enable signals input enable and output enable, which can drive the state machine in Fig. 4.10. The state machine will generate a start signal to tell hardware device on working and if it works successful, it will send output enable signal.

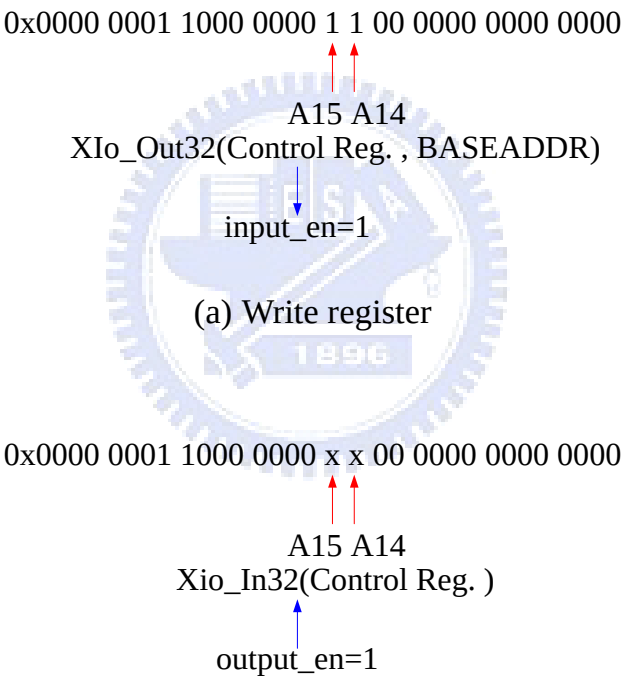


Fig. 4.9 Control Register

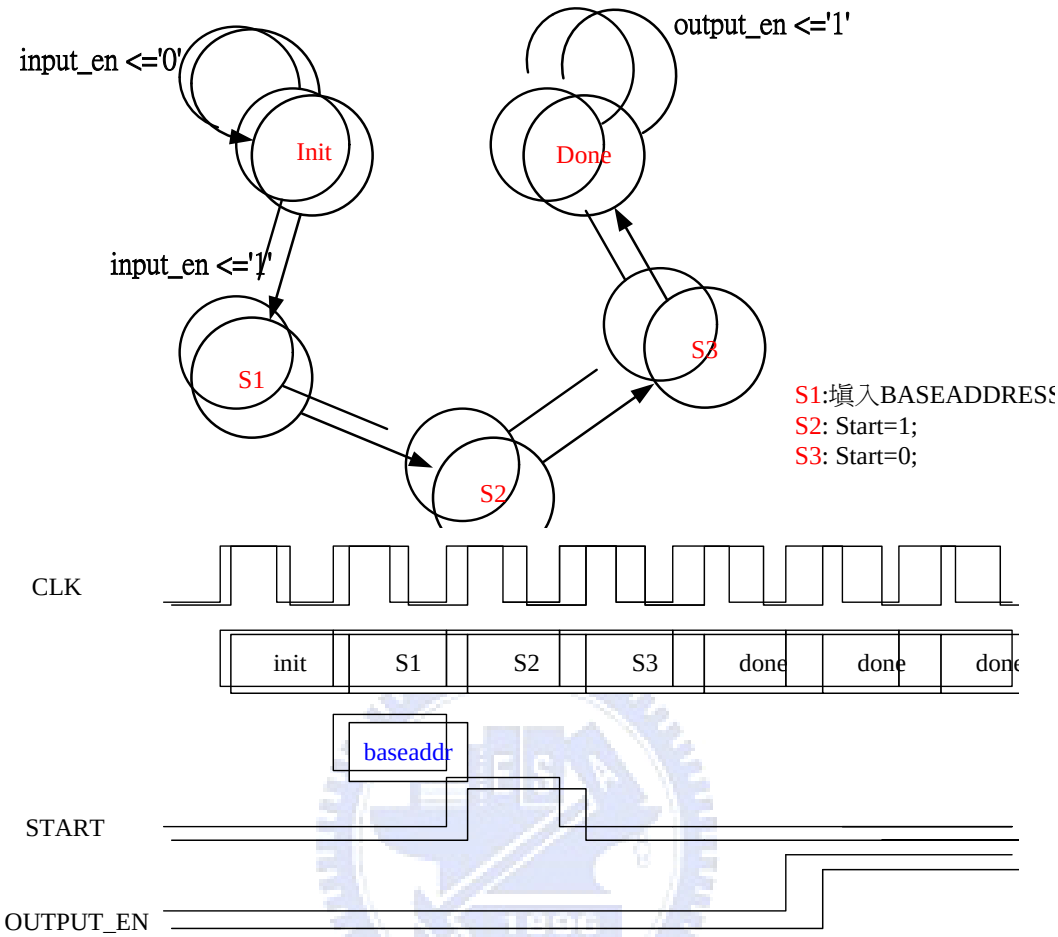


Fig. 4.10 State Machine

In conclusion, the state machine of Fig. 4.10, we purpose to send a start signal on the memory mapped address which the MicroBlaze embedded system will drive ASIC FFT start to work. Therefore, we can complete the software and hardware integrated system based on MicroBlaze embedded system according to control register and state machine control flow.

In the Fig. 4.11 shows the simulation wave of hardware device integration which when the software FFT was done, the remaining part of FFT will be executed by hardware device including control register, state machine, memory block and ASIC FFT. Therefore, we can get the whole data of FFT algorithm and combine from software part to hardware part such as Fig. 4.2: “Separate the implementation of 64-points FFT for two parts” shows. First in square 1, hardware device starts to read the memory mapped addressing data. Second in square 2, prepare to send the read data of memory from square 1 into FFT input port. Third in square 3,

input the FFT 16-bits of 8-points FFT input data and active with a start signal. Fourth in square 4, it shows the output data after doing FFT according to ASIC FFT hardware. Finally in square 5, to store the result data back to memory mapped addressing block.

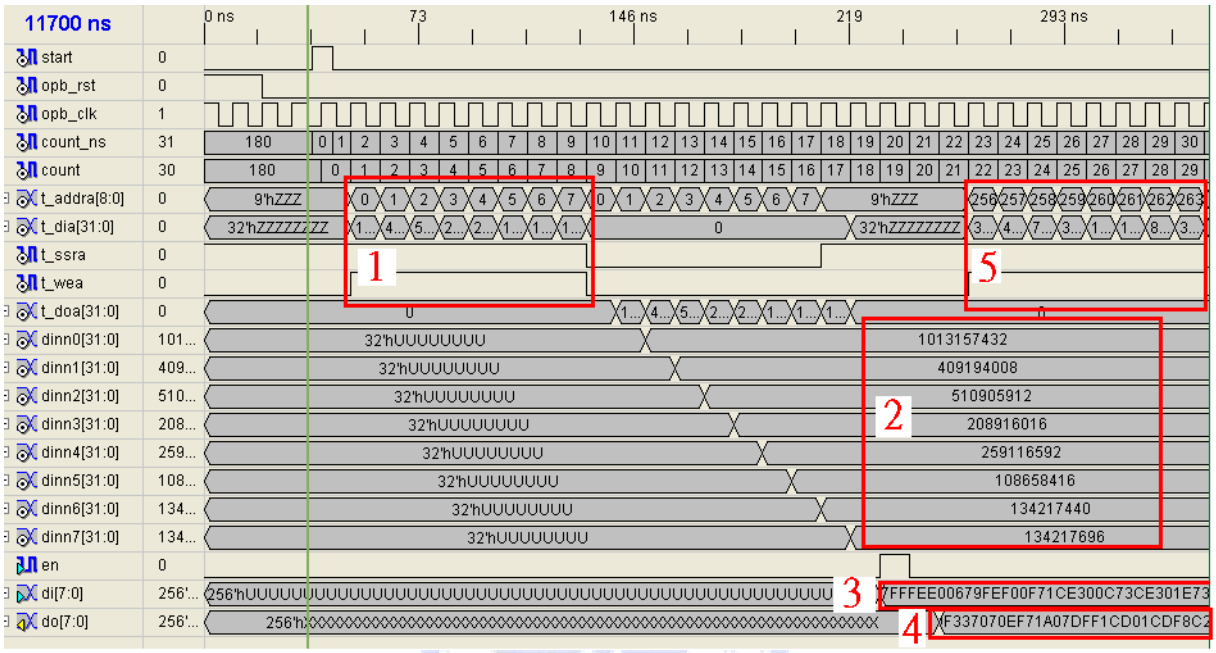


Fig. 4.11 Simulation of hardware operation

4.3 MIPS of the Variable-Length FFT Implemented by MicroBlaze Processor

According to time schedules we discussed in section 3.4:“ASIC and Processor Timing Schedule Analysis”. Our implementation of 64-points FFT can be design for SISO Ⅰ in Fig. 3.5~3.6 ,SISO Ⅱ in Fig. 3.7~3.8 and MIMO in Fig. 3.9~3.10 schedules. Therefore, we conclude these three summary for MIPS (Million Instructions per Second) shown in Table 4.1 based on IEEE 802.11n standards, which 64-points FFT is analyzed by 20Mhz of a symbol duration and FFT length ASIC is 8-points. In the Table 4.1 the MIPS of MIMO is twice as SISO Ⅱ.

Table 4.1 MIPS of 64-points FFT based on IEEE 802.11n standards

64-points FFT	SISO Ⅰ	SISO Ⅱ	MIMO
MIPS	1632	1110	2220

In chapter 3, we discussed the MOPS of variable-length FFT according to IEEE 802.11n standards such as 64-points, 128-points, 512-points, 1024-points and 2048-points FFT that MOPS is implemented by DSP processor. In the other word, we want to implement the variable-length FFT on MicroBlaze processor environment which is based on RISC, it provides processor instructions but no processor operations for users. Therefore, MIPS on MicroBlaze processor is our purposed of variable-length FFT implementation.

From section 4.2: “Implementation of the Variable-Length FFT”, we use the same way to perform 2048-points FFT. Because of the IEEE 802.11n/16e standards, 64-points/ 2048-points FFT are our desired cases which we have to evaluate all the MISP conditions of variable-length FFT for three time schedules.

Fig. 4.12 is the SISO Ⅰ time schedule we implemented by MIPS, Fig. 4.13 is the SISO Ⅱ time schedule and Fig. 4.14 is MIMO time schedule. In conclusion, 64-points FFT is based on IEEE 802.11n standard, 2048-points FFT is based on IEEE 802.16e standard that we can use to implement wimax applications.

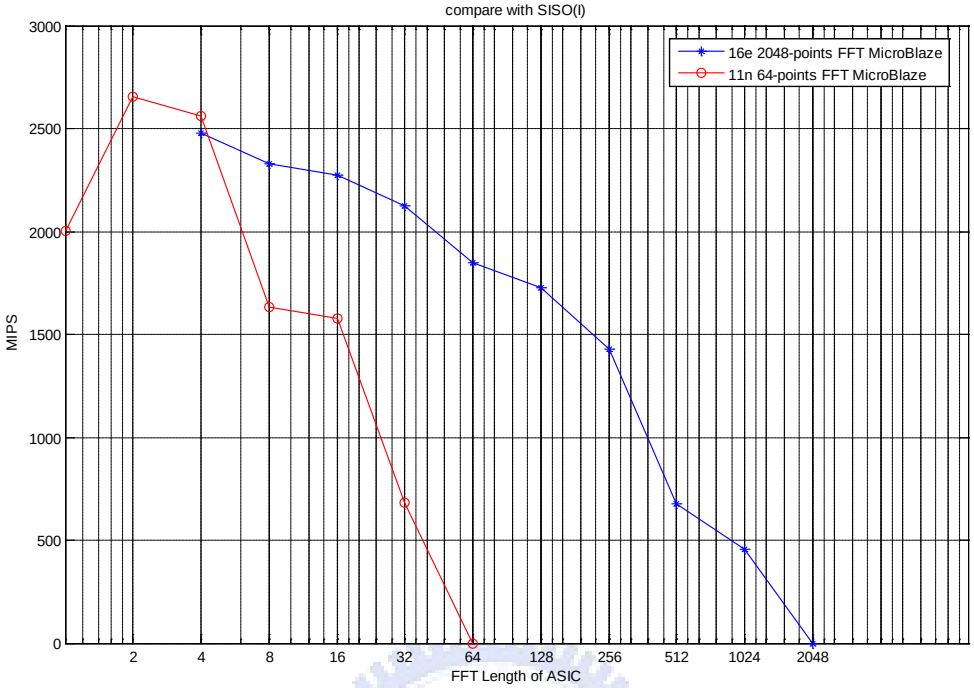


Fig. 4.12 Processor's instructions analysis of schedule Ⅰ in SISO system

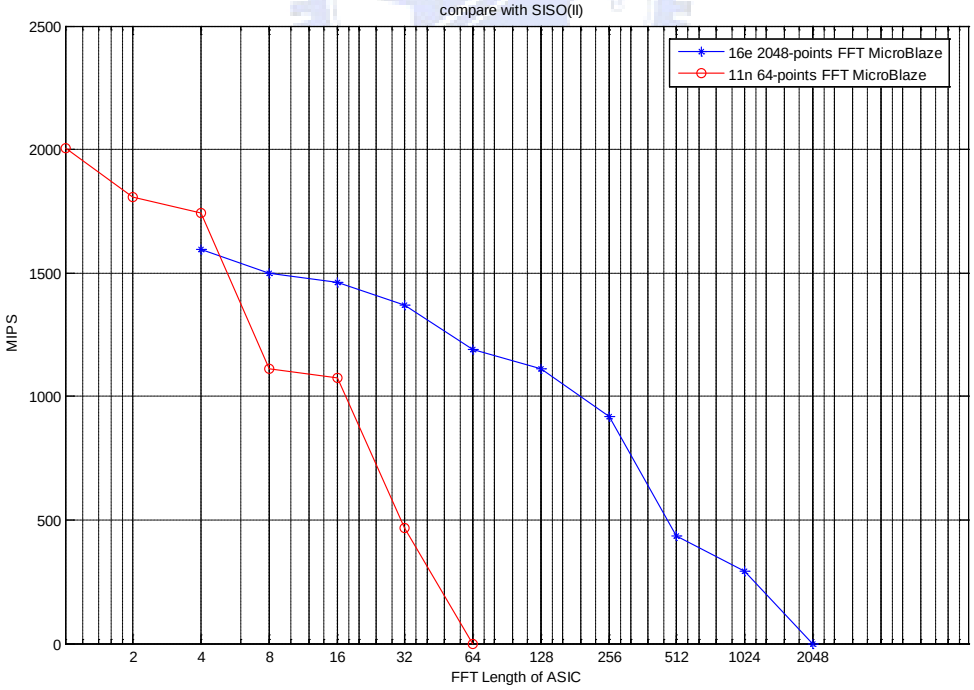


Fig. 4.13 Processor's instructions analysis of schedule Ⅱ in SISO system

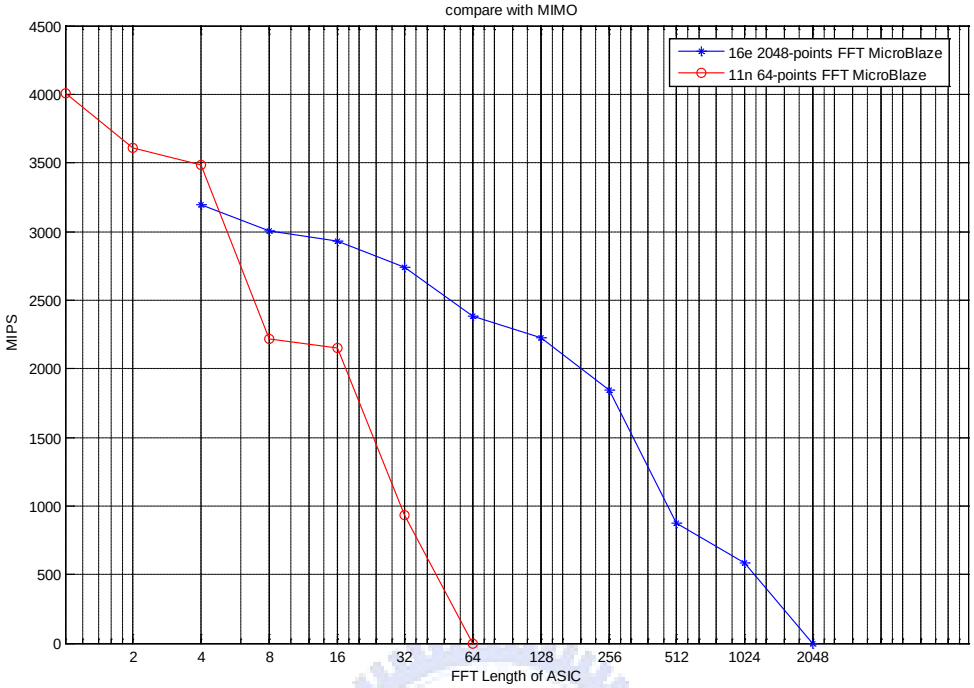


Fig. 4.14 Processor's instructions analysis in MIMO system

4.4 Error Analysis

In the case of FFT hardware implementation, the finite bit-width must be considered because of the fixed-point computation. Many statistical error analysis papers on FFT implementations are proposed [36], [37], [38]. Assume the input sequence of FFT x(n) is a sequence of finite-valued and white complex numbers. The variance of x(n) can be expressed as

$$\sigma_x^2 = \frac{1}{N}\sum_{n=0}^{N-1}(x(n)-\mu_x)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(x(n))^2 \qquad (3)$$

where μ_x is the mean of x(n) andμ_x=0. The SQNR (Signal to Quantization Noise Ratio) is defined as

$$SQNR = \frac{\sigma_x^2}{\sigma_q^2} \qquad (4)$$

Where σ_x^2 is the variance of output andσ_q^2 is the variance of the quantization error. For an N-point FFT module with input of which real and imaginary parts are uniformly dis-

tributed in $(-\frac{1}{\sqrt{2}}N, \frac{1}{\sqrt{2}}N)$, the variance [37] of the output is

$$\sigma_x^2 = \frac{1}{3N} \tag{5}$$

From (4) and (5), the SQNR [38] of the conventional FFT implementation can be carried out：

$$SQNR_{FFT} = \frac{2^{2B}}{5N - 4m - 3} \tag{6}$$

Where B is the bit-width of the input sequence and m=$\log_2$N.

In Fig. 4.11, it shows equation (6) with IEEE 802.11n/16e standards which include five FFT sizes. The more rounding stages, the more noise will be produced. Because long-length FFT will decrease SQNR, it needs to increase bit-width. It will cause more power consumption and area cost.

In this chapter, we introduce various pipeline-based FFT architectures and then compare their characteristic to evaluate our proposed system based on throughput rate and hardware cost analysis. After that, it shows detailed sub-module architectures and analyzes noise issue finally.

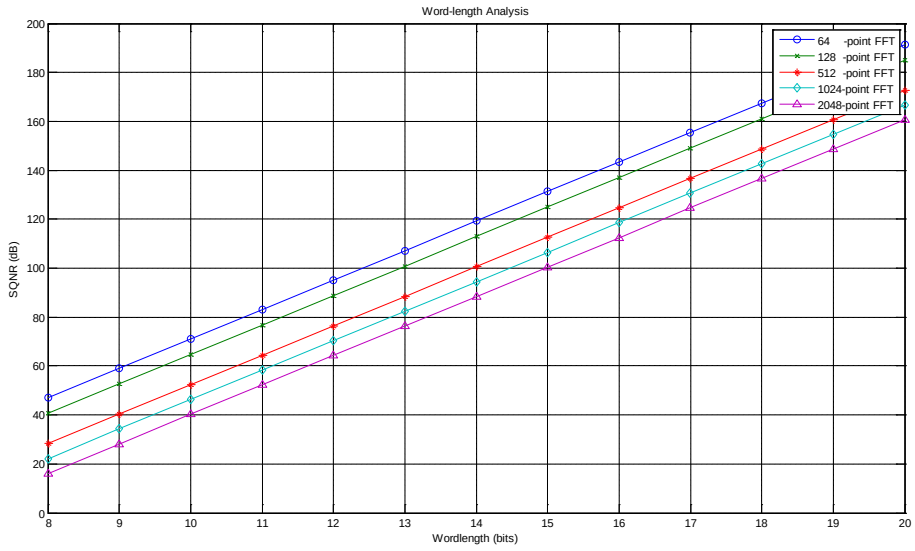


Fig. 4.15 Noise analysis with different FFT length

From the example of 64-points FFT in section 4.2 that we take the output of 16-bits fixed data to compare with the real floating data. We take the SNR is about 144.06 dB shown in function (7) which 144.06 dB is nearly when compared with 64-points FFT and length 16-bits of Fig. 4.15. The sources of $X(f)$ and $\hat{X}(f)$ shown in Fig. 4.16.

$$FFT(\text{floating point of } 64\text{-points FFT}) : X(f)$$

$$FFT(16\text{-bits fixed point of } 64\text{-points FFT}) : \hat{X}(f)$$

$$\sum X^2(f) = 2048$$

$$\sum (X(f) - \hat{X}(f))^2 = 0.00012836$$

$$SNR = 20\log\frac{\sum X^2(f)}{\sum (X(f) - \hat{X}(f))^2} = 144.06 \quad dB \qquad (7)$$

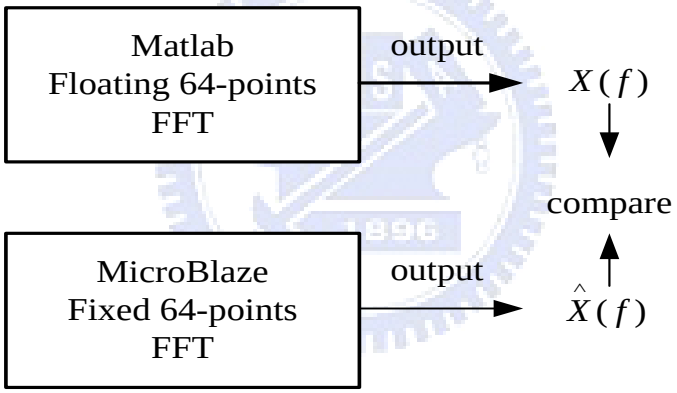


Fig. 4.16 The compare with output data

Chapter 5

Conclusion

5.1 Conclusion

Since processor is popular in recent years, we intend that the FFT module can combine processor with ASIC to form the flexible system. ASIC plays an accelerator role in the proposed system. Based on FFT computational complexity analysis, it shows different length branch FFT of ASIC which affects processor performance. Therefore, it can provide user two anticipation as below. First of all that processor needs to spare how much computational performance at least for proposed system. Second, because of our processor computational performance, we can decide the branch FFT length of ASIC.

In our implementation environment that the processor can contribute the range of MIPS in this thesis based on IEEE 802.11n/16e standards. Users can decide which condition of MIPS they want on processor. After that they accomplish the branch FFT according to the restriction of MIPS. The variable-length FFT implemented by MicroBlaze Processor has 16-bits word-length and its SNR is near 140 dB shown in Fig. 4.15.

Finally, we not only verify the 64-point branch FFT on FPFA of our example in section 4.2: Implementation of the Variable-Length FFT, but also check proposed timing schedule which covers 64-points, 128-points, 512-points, 1024-points and 2048-points FFT algorithm in SISO/MIMO systems based on IEEE 802.11n/16e standards.

5.2 Work of Implementation Environment

The processor is implemented by embedded system which we provide one method to setup up proposed system. In Xilinx Spartn-3 FPGA [30], [31], [32], [33], [34], [35], it has an embedded processor which is MicroBlaze processor of IBM [28]. Therefore, the processor can be entirely built by writing C-language and the N-points branch FFT can be loaded to FPGA as an accelerator.

In this thesis, the processor performance analysis is based on radix-2/4/8 algorithms. Because of the processor performance is based on instructions, we can try to use higher radix algorithm but it requires more high frequency clock cycles. Hence, the resource cost will be reduced while keeping specification requirements and the shift registers is another issue. For a bigger N, the shift registers will cause more power consumption and area cost than using memory access. Therefore, how to improve the efficiency and simplify the memory access scheme in the long length branch FFT module is left for our future work.

Bibliography

[1] Mujtaba et al., TGn Sync Proposal Tech. Specification for IEEE 802.11 Task Group 2005, IEEE 802.11-04/0889r3.

[2] Bing-Juo Chuang, “Design and Implementation of IEEE 802.11n Based Receiver,” Department of Communication Engineering, National Chiao Tung University, July 2005.

[3] A.V. Oppenheim R.W. Schafer, and John R. Buck, Discrete Time Signal Processing, Second Edition, Prentice Hall Inc, 1999.

[4] Shousheng He and Mats Torkelson, “A New Approach to Pipeline FFT Processor,” Parallel Processing Symposium, The 10th International, pp. 766-770, 1996.

[5] Shousheng He and Mats Torkelson, “Designing Pipeline FFT Processor for OFDM (De)Modulation,” in Proc. ISSSE, pp. 257-262, 1998.

[6] Shousheng He and Mats Torkelson, “Design and Implementation of A 1024-point FFT Processor,” in Proc. IEEE Custom Integrated Circuit Conference, pp. 131-134, 1998.

[7] E. H. World and A. M. Design, “Pipeline and Parallel Pipeline FFT Processors for VLSI Implementation,” IEEE Transactions on Computers, Vol. 33 No. 5, pp. 414-426, May 1984.

[8] A. M. Despain, “Fast Fourier Transform using CORDIC iterations,” IEEE Trans. Comput., Vol. C-23 No. 10 pp. 933-1001, Oct. 1974.

[9] G. Bi and E. V. Jones, “A Pipelined FFT Processor for Word Sequential Data,” IEEE Trans. Acoust., Speech, Signal Processing, Vol. 37 No. 12, pp. 1982-1985, Dec. 1989.

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

[11] Chung-Ping Hung, Sau-Gee Chen and Kun-Lung Chen, “Design of An Efficient Variable-Length FFT Processor,” IEEE ISCA, Vol. 2, pp. 833-836, May 2004.

[12] L. G. Johnson, “Conflict Free Memory Addressing for Delicate FFT Hardware,” IEEE Transactions on Circuit and System-II: Analog and Digital Signal Processing, Vol. 39 No. 5, pp. 312-316, May 1992.

[13] Hsin-Fu Lo, Ming-Der Shieh, and Chien-Ming Wu, “Design of An Efficient FFT Processor for DAB System,” IEEE International Symposium on Circuit and Systems, Vol. 4, pp. 645-657, 2001.

[14] Yutai Ma, “An Effective Memory Addressing Scheme for FFT Processors,” IEEE Transactions on Signal Processing, Vol. 47 Issue: 3, pp. 907-911, Mar. 1999.

[15] Yutai Ma and Lars Wanhammar, “A Hardware Efficient Control of Memory Addressing for High-Performance FFT Processors,” IEEE Transactions on Signal Processing, Vol. 48 Issue: 3, pp. 917-921, Mar. 2000.

[16] C. H. Chang, C. L. Wang and Y. T. Chang, “Efficient VLSI Architectures for Fast Computation of the Discrete Fourier Transform and its Inverse,” IEEE Transactions on Signal Processing, Vol. 48 Issue: 11, pp. 3206-3216, Nov. 2000.

[17] C. L. Wang and C. H. Chang, “A New Memory-Based FFT Processor for VSDL Transceivers,” IEEE International Symposium on Circuit and Systems, Vol. 4, pp.670-673, 2001.

[18] M. Hasan, T. Aralan, “FFT Coefficient Memory Reduction Technique for OFDM Applications,” IEEE International Conference on Acoustics, Speed, and Signal Processing, Vol. 1, pp. I1085-I1088, May 2002.

[19] Li-Yun Lin, “Implementation of A Variable Length FFT Processor for VDSL system,” Department of Communication Engineering, National Chiao Tung University, June 2004.

[20] B. S. Kim and L. S. Kim, “Low Power Pipelined FFT Architecture for Synthetic Aperture Radar Signal Processing,” in Proc. IEEE Midwest Symposium on Circuits and Systems, Vol. 3, pp. 1367-1370, 1996.

[21] M. M. Jamali, S. C. Kwatra and D. H. Shetty, “Module Generation Based VLSI Implementation of A Demultiplexer for Satellite Communications,” in Proc. IEEE International Symposium on Circuits and Systems, Vol.4, pp 364-367, 1996.

[22] A. Delaruelle, j. huisken, J. van Loon, F. Welten, “A Channel Demodulator IC for Digital Audio Broadcasting” in Proc. IEEE Custom Integrated Circuits Conference, pp. 47-50, 1994.

[23] “Supplement to IEEE Standard for Information Technology Telecommunications and Information Exchange between Systems-Specific Requirements. Part 11: Wireless LAN Medium Access Control and Physical Layer,” IEEE 802.11a. 1999.

[24] T. Sansaloni, “Efficient Pipeline FFT Processors for WLAN MIMO-OFDM Systems,” IEE electronics letters 15th, Vol. 41 No. 19, September 2005.

[25] Chih-Wei Liu, “Introduce to FFT Processors,” VLSI Signal Processing Lab Department of Electronics Engineering, National Chiao Tung University.

[26] Shousheng He and Mats Torkelson, “Designing Pipeline FFT Processor for OFDM (De)modulation”, in Proc. URSI Int. Symposium on Signals, Systems, and Electronics, Vol. 29, pp. 257-262, Oct. 1998.

[27] Ray Andraka, Andraka Consulting Group, Inc., 16 Arcadia Drive, North Kingstown, RI “A Survey of CORDIC Algorithm for FPGA Based Computers” ACM Press, 1998, New York, USA.

[28] “On-Chip Peripheral Bus Architectures Specifications,” a complete description of the bus by IBM, its inventor.

[29] S. Sukhsawas and K. Benkird, “A High-Level Implementation of A High Performance Pipeline FFT on Virtex-E FPGAs,” School of Computer Science, Queen’s University Belfast, United Kingdom, 2004.

[30] “MicroBlaze Processor Reference Guide: Embedded Development Kit EDK 6.3i,” UG081 (v4.0) August 24, 2004.

[31] “Xilinx ISE 6 Software Manuals,” UG000 (v3.4.1) February 25, 2003.

[32] “Xilinx Embedded Development Kit (EDK documents): Getting Started with the EDK 6.3i,” August 24, 2004.

[33] “Xilinx Embedded Development Kit (EDK documents): Xilinx Platform Studio User Guide,” UG113 (v3.0), August 20, 2004.

[34] “Xilinx Embedded Development Kit (EDK documents): Tools and IP Reference Guides: Embedded System Tools Reference Manual,” UG111 (v3.0), August 20, 2004.

[35] “Xilinx Embedded Development Kit (EDK documents): MicroBlaze Processor Reference Guide,” UG081 (v4.0), August 24, 2004.

[36] P.D. Welch, “A Fixed-Point Fast Fourier Transform Error Analysis,” IEEE trans. on audio and electroacoustics, Vol. AU-17, No. 2, pp. 151-157, Jun. 1969.

[37] A.V. Oppenheim and C. J. Weinstein, “Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform,” pro. of the IEEE, Vol. 60, No.8, pp. 957-976, Aug. 1972.

[38] M. Sundaramurthy and V. U. Reddy, “Some Results in Fixed-Point Fast Fourier Transform Error Analysis,” IEEE trans. on computers, pp. 305-308, Mar. 1997.