國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

可平行順序輸入及輸出快速傅立葉轉換處理器之設計

Design of FFT Processor with Parallel-In-Parallel-Out in Normal Order

研究生：胡祥蛙

指導教授：周世傑 博士

中華民國九十七年十一月

可平行順序輸入及輸出快速傅立葉轉換處理器之設計

Design of FFT Processor with Parallel-In-Parallel-Out in Normal Order

研究生：胡祥蛀　　Student：Hsiang-Sheng Hu

指導教授：周世傑 博士　　Advisor：Dr. Shyh-Jye Jou

國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

A Thesis

Submitted to Department of Electronics Engineering &

Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

In

Electronics Engineering

July 2006

Hsinchu, Taiwan, Republic of China

中華民國九十七年十一月

可平行順序輸入及輸出快速傅立葉轉換處理器之設計

研究生：胡祥蛙　　指導教授：周世傑 博士

國立交通大學

電子工程學系 電子研究所碩士班

摘要

為了設計一個具有可平行輸入及平行輸出的快傅立葉轉換處理器以適用於高速移動無線都會型區域網路(WMAN)基頻接收器中的通道估測方法，本論文由硬體設計層級研究各種可平行輸入及平行輸出的快速傅立葉轉換硬體架構技術。同時，為了簡化快速傅立葉轉換電路的資料串列輸入及資料串列輸出所需的控制訊號複雜度及暫存器使用，本論文提出一個可依序平行輸入及平行輸出的快速傅立葉轉換電路，以符合系統對快速傅立葉轉換處理器的輸入輸出規格要求。最後，本論文提出一個可適用於 802.16e 通訊系統中離散傅立葉通道估測法(DFT-based channel estimation)的可平行順序輸入及平行順序輸出之快速傅立葉轉換處理器的架構設計。並且根據離散傅立葉通道估測法中對快速傅立葉轉換的特殊需求，提出一個可適用於此通道估測法的部份傅立葉轉換(Partial FFT)架構設計。最後，本論文所提出的快速傅立葉轉換處理器已實現於一個 2x1 STBC/OFDMA 基頻接收器中。此傅立葉轉換處理器可達到最高 1.28 G 樣本/秒的

資料吞吐量；當操作在最大工作頻率 160 MHz 下，其資料延遲時間僅需 7.3 us；當操作在系統給定頻率 78.4 MHz 下，此傅立葉轉換電路消耗功率為 21.7 mW，面積為 155792 邏輯閘數(包含記憶體)，使用 90 奈米 1V CMOS 製程下，其面積為 0.545 mm^2。

Design of FFT Processor with Parallel-In-Parallel-Out in Normal Order

Student：Hsiang-Sheng Hu Advisor：Dr. Shyh-Jye Jou

Department of Electronics Engineering & Institute of Electronics
National Chiao Tung University

ABSTRACT

In order to design a parallel-in-parallel-out Fast Fourier Transform (FFT) processor suitable for channel estimation in a highly mobile wireless metropolitan area network (WMAN) baseband receiver, this thesis studies various parallel-in-parallel-out FFT design techniques from hardware architecture level. Also, in order to reduce the control complexity and buffer overhead for data stream-in and stream-out of the FFT processor, this thesis proposes a FFT processor with parallel-in-parallel-out in normal order to meet the input data and output data requirement for the systems requirement. Finally, this thesis proposes a 1024-point FFT processor architecture with parallel-in-parallel-out in normal order, which can meet the needs of DFT-based channel estimation in 802.16e communication system. Furthermore, according to the special requirement of the DFT-based channel estimation, the thesis proposes the partial FFT processor architecture suitable for the DFT-based channel estimation. The FFT/IFFT processor is designed and is implemented together with a 2×1 STBC/OFDMA baseband receiver. The proposed 1024-point FFT/IFFT processor

can achieve the throughput rate up to 1.28 G samples/sec and the execution time down to 7.3 us when working at 160 MHz. When working at the system required 78.4 MHz, it consumes 21.7 mW with 155792 gates (including memory) that occupy 0.545 mm^2 by using 90 nm, 1V CMOS process.

誌　　謝

首先我要感謝指導教授 周世傑老師兩年多來的提攜，使我在研究迷惘之餘，給予我方向與努力的目標，並且在為人處事及研究態度上，都給予我不少寶貴的建議。

再來，我要感謝 momo 學姐的幫忙，總是不吝其煩的幫我解決研究上得困難，還有陳紹基老師實驗室的學生黃紳叡學長的協助，使得我在理論及實作方面能有顯著的進步，誠文學長更是分享很多他工作上的經驗，使我研究更加順利，庭楨學長也給予我很多理論和實作上的幫助，在此感激這些學長姊使我能順利完成研究。

另外實驗室的好夥伴們，紹維、盈志、小胖、運翔、舒蓉、儷蓉、莊立、…還有許多許多其他的好同學們，感謝你們總在我最失意時，幫我加油打氣，給予我許多鼓勵與包容。

最後，感謝我的父母親、家人以及女朋友長久以來的支持與鼓勵，沒有你們這條研究的路很難堅持下去，謹代表我內心致上最大的感激與敬意。

Content

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

In many digital signal processing applications, especially in communication systems, Fast Fourier Transform (FFT) becomes more important nowadays. Orthogonal frequency division multiplexing (OFDM) technology [1] is used in the most modern wired or wireless communication systems, such as ADSL, VDSL, 802.11a, DVB-T, 802.16-2004 [2], 802.16e [3], which needs a FFT processor to transform the data between time domain and frequency domain; however, the FFT processor is the critical component in many OFDM based communication systems because the FFT processor's hardware complexity is too high. For this reason, many FFT processors are designed for OFDM based communication systems to make the FFT processor become efficiency for system implementation.

As the result of growing VLSI technology, improved modulation and channel estimation can be implemented with reasonable cost. OFDM is an improved modulation technique that can provide high data rate, immunity to delay spread, resistance to frequency selective fading, and efficient bandwidth usage. In wireless communication, OFDM also reduces inter-symbol-interference (ISI) and inter-carrier interference (ICI) caused by multipath effect. Also the Discrete Fourier Transform (DFT)-based channel estimation [4] with space time block code (STBC) [5] is proposed to do channel estimation in OFDM wireless communication system, which is effective in high mobility channel environment. In these applications, FFT plays an

important role to decide the system performance and hardware cost; thus, a high throughput FFT processor with low hardware cost is an important module to make more advanced modulation and channel estimation algorithm to be implemented on chip reasonable.

In order to design a high throughput FFT and also speed up the operations ahead or behind the FFT processor, a parallel-in-parallel-out FFT will be introduced in this thesis; also a 1024-point parallel-in-parallel-out in normal order FFT processor design example used in DFT-based channel estimation in 802.16e will be proposed.

1.2 Thesis Organization

In this thesis, FFT/IFFT designs for robust channel estimation of high-mobility STBC/OFDMA communication system are proposed. System simulation, architecture and circuit design, and implementation of FFT/IFFT processor with baseband of 802.16e are carried output in thesis. IEEE 802.16e, DFT-based channel estimation, and the system block we used, will be introduced in Chapter 2. Since the system block we used including two kinds of FFT/IFFT processor design, we also introduce the system requirement for different kind of FFT/IFFT processor: one for OFDMA demodulation, the other for DFT-based channel estimation. The system requirement of FFT processor for 802.16e OFDMA demodulation has no difference with other OFDM communication, thus, the thesis will introduce the conventional FFT processor we used in Chapter 2. Shared memory concept is used between FFT processor, used for OFDMA demodulation, and channel estimation. The requirement of FFT/IFFT processor used in DFT-based channel estimation is different from the conventional FFT processor by two aspects. One aspect is parallel-in-parallel-out of data and the other aspect is a FFT processor with several zero value input or several valid output, called partial FFT processor. The thesis focus on the FFT/IFFT processor hardware

design for channel estimation with parallel-in-parallel-out in normal order, and then the concept of partial FFT processor design will be demonstrated.

Investigation of the conventional FFT algorithm and various parallel-in-parallel-out FFT architectures is presented in Chapter 3. The conventional high throughput FFT processors usually use a pipeline-based FFT architecture which provide high throughput but also has high hardware cost. Memory-based FFT architecture has the advantage of low hardware cost, and it can also provide high throughput by parallel-in-parallel-out with multi-partitioned memories. The comparisons among the different parallel-in-parallel-out FFT architectures are also carried out in Chapter 3. The comparison results are helpful to FFT processor design in our system. At the end of Chapter 3, concept of partial FFT processor design will be introduced to solve another goal of FFT processor for DFT-based channel estimation.

The architecture design of FFT processor with parallel-in-parallel-out in normal order will be proposed in Chapter 4. A novel memory allocation method for parallel-in-parallel-out in normal are proposed in this chapter. Designs of processing elements, memory allocation, commutator, scale down block, and coefficient ROM table for the proposed FFT processor will be introduced, and considered as the key contribution of this thesis. In the end of Chapter 4, comparisons are carried out for the hardware implement result with other FFT processor with parallel-in-parallel-out in normal order.

Backend design flow for the chip of 802.16e receiver will be introduced in Chapter 5. In order to tape out the chip, two versions of chip implementation results are presented, one for UMC shuttle, the other for CIC. The chip floor plan and design flow will be presented in Chapter 5.

In the end of the thesis, the conclusion future works will be presented in Chapter

6.

Chapter 2

FFT Application in OFDM Communication System

2.1 Concept of OFDM

Orthogonal Frequency Division Multiplexing (OFDM) is based on frequency division multiplexing (FDM). FDM translates several message signals to different spectral locations. An example of bandwidth allocation of FDM is shown in Fig. 2.1.

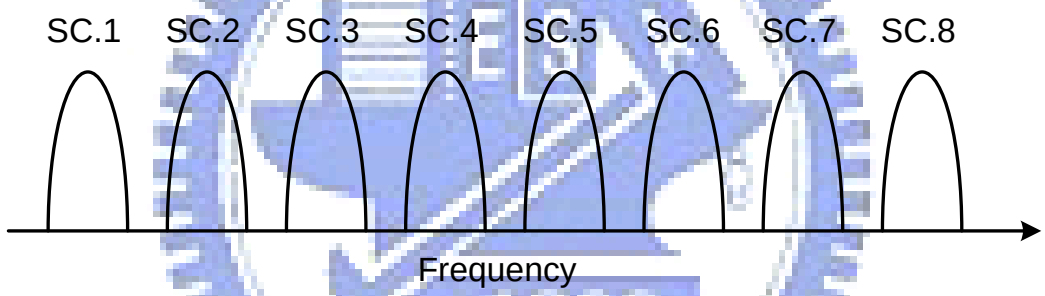


Fig. 2.1 Bandwidth allocation for sub-cannels in FDM system

FDM technique keeps all sub-channels away from overlapping by guard bands to against the adjacent sub-channels producing inter-channel interference (ICI); however, guard bands waste the bandwidth efficiency, which is important in communication system, because it is not used to carry any message signals. OFDM uses orthogonal sub-carriers to overlap the sub-channels to carry more message signals in the same bandwidth than FDM as shown in Fig. 2.2.

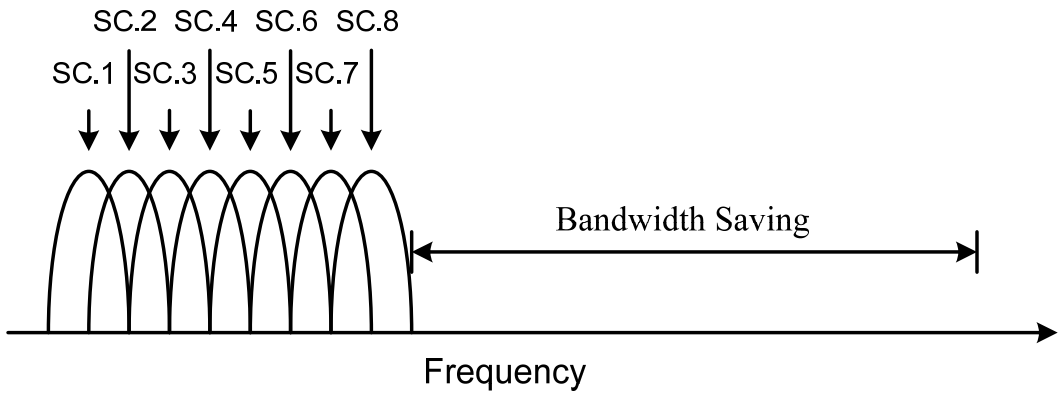


Fig. 2.2 Bandwidth allocation for sub-channels in OFDM system

A basic block diagram of OFDM system is shown in Fig. 2.3. Fig. 2.3 shows the transmitter in OFDM system need IFFT module to modulate the message signal, called OFDM modulation, and the receiver also need FFT module for OFDM de-modulation; thus, FFT processor is a key block in OFDM transceiver system.

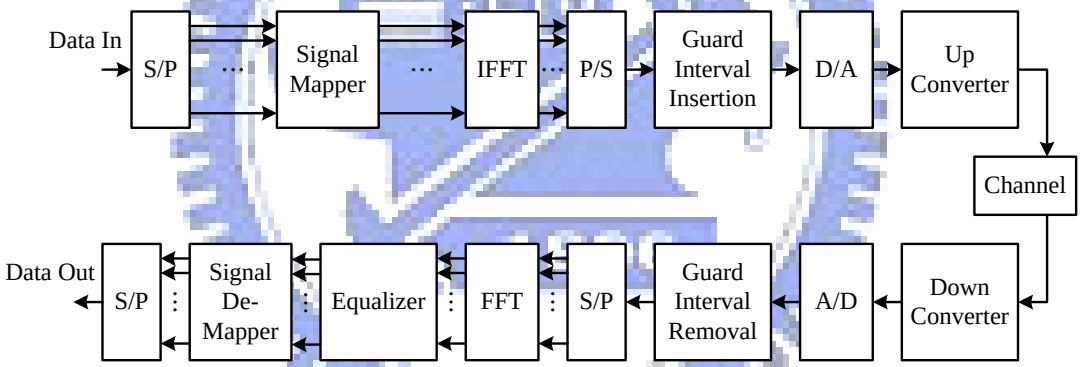


Fig. 2.3 Basic block diagram of an OFDM transceiver system

2.2 Introduction of IEEE 802.16e

IEEE 802.16 is a broadband wireless access (BWA) standard. The first standard of IEEE 802.16 approved in December 2001 called IEEE 802.16-2001 [6]. It delivered a standard, which transmits in 10-66 GHz with only a line-of-sight (LOS) capability, used in Wireless Metropolitan Area Networks (WiMAN). It uses a single carrier (SC) physical (PHY) standard. IEEE 802.16a is a extension of IEEE 802.16-2001. It transmits in 2-11 GHz with both LOS and none-light-of-sight (NLOS),

and less distortion by rain than IEEE 802.16-2001. IEEE 802.16-2004 (also called IEEE 802.16d) is a fixed broadband wireless access (BWA) standard, which combines both of IEEE 802.16-2001 and IEEE 802.16a standards. IEEE 802.16-2004 describes more detail for media access control layer (MAC) and PHY in 2-66 GHz. It supports multiple physical layer (PHY) specifications, such as WiMAN-SC, WiMAN-OFDM, WiMAN-OFDMA, and WiMAN-SCa, operation in different frequency. For operation frequency in 10-66 GHz, the WiMAN-SC PHY, based on single carrier, is specified; for operation frequency below 11 GHz, the IEEE 802.16-2004 transmitting in NLOS provides three alternative PHY specifications: WiMAN-OFDM (based on orthogonal frequency division multiplexing), WiMAN-OFDMA (based on orthogonal frequency division multiple access), WiMAN-SCa (based on single carrier). IEEE 802.16e, which is a fixed and mobile broadband wireless access (BWA) standard, is an enhancement of IEEE 802.16-2004 standard. It fills the gap between very high data rate local area network and very high mobility cellular system. An extension PHY layer specification called scalable-OFDMA (SOFDMA), based on WiMAN-OFDMA, provide different FFT Size for OFDMA, such as 128, 512, 1024, 2048 points. Table 2-1 is the summary of IEEE 802.16-2001, IEEE 802.16a, and IEEE 802.16e.

Table 2-1 Comparisons of IEEE 802.16 standards

	IEEE 802.16-2001	IEEE 802.16a	IEEE 802.16e
Spectrum	10-66 GHz	2-11 GHz	2-6 GHz
Channel Bandwidth	20, 25, 28 MHz	1.5 to 20 MHz	1.5 to 20 MHz
Carrier	Single Carrier	OFDM/OFDMA	OFDM/SOFDMA
FFT Size	N/A	256(OFDM) 2048(OFDMA)	256(OFDM) 128/512/1024/2048 (SOFDMA)
Modulation	QPSK, 16QAM, 64QAM	QPSK, 16QAM, 64QAM	QPSK, 16QAM, 64QAM
Bit Rate	32-134 Mbps (28 MHz)	75 Mbps (20 MHz)	15 Mbps (5 MHz)
Channel Conditions	LOS	Non-LOS	Non-LOS
Typical Cell Radius	2-5 Km	7-10 Km, max 50 Km	2-5 Km
Application	Fixed	Fixed and portable	Fixed and mobile

2.3 DFT-Based Channel Estimation

Channel estimation in conventional OFDM system is a simple one-tap equalizer since the channel gain varies slowly between each adjacent OFDM symbol. However, in the mobile wireless communication environment, such as the channel in IEEE 802.16e, the channel gain varies rapidly between each adjacent OFDM symbol, so a one-tap equalizer seems not suitable for the time-varying channel environment. The one-tap equalizer can be realized as a least square (LS) channel estimator, and it has low hardware complexity but low performance than minimum-mean-square-error (MMSE) estimator. MMSE estimator has better performance but the hardware complexity is too high. DFT-based channel estimation [7-9] is presented to combine the LS and MMSE estimator, and it reduces the hardware complexity of MMSE estimator. A simple block diagram of DFT-based channel estimation is shown in Fig. 2.4. R(k) is the received data in sub-carrier k after OFDM demodulation, X(k) is the

decision data, which is determined by the latest OFDM symbol channel estimator, and H(k) is the channel estimator used in next OFDM symbol.

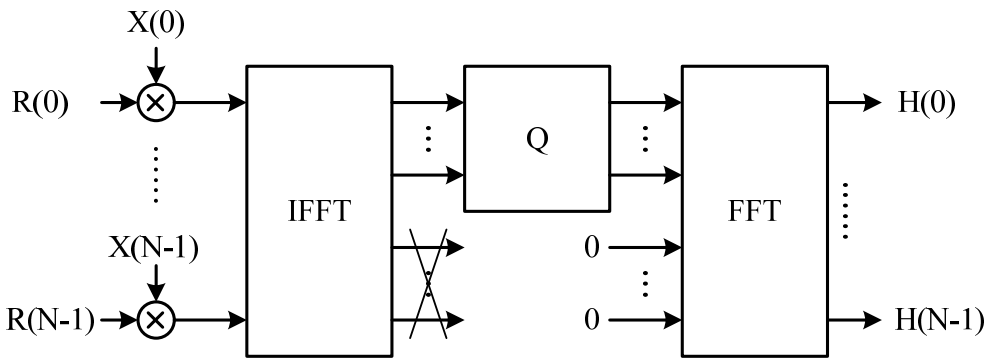


Fig. 2.4 Block diagram of DFT-based channel estimation

DFT-based channel estimation can provide more accurate channel gain with lower hardware complexity than the original MMSE estimator. However, it needs both IFFT block and FFT block to implement the algorithm. Thus, a suitable FFT or IFFT processor design can reduce the hardware cost of DFT-based channel estimation.

2.4 System Specification

For mobile WMAN baseband transceiver using standard IEEE 802.16e, we proposed a baseband transceiver [10]. A simply block diagram of the 2×1 multiple-input-single-output (MISO) IEEE 802.16e OFDM system is shown in Fig. 2.5. For chip implementation, we only implement the receiver part of Fig. 2.5. The key system specifications are listed in Table 2-2.

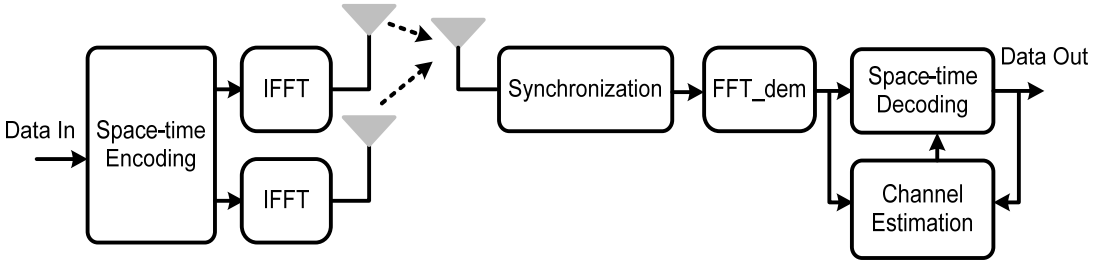


Fig. 2.5 Block diagram of baseband transceiver in IEEE 802.16e

Table 2-2 System specification of IEEE 802.16e transceiver system

Items	Specification
Bandwidth	10 MHz
PHY Layer Specification	WiMAN-SOFDMA
FFT Size	1024
Sample Rate	11.2 MHz
Guard Interval	1/8
Constellation	QPSK, 16QAM
OFDM Symbol Time	102.9 us

The channel estimation block is a decision feedback (DF) DFT-based channel estimation [10], which combines the channel estimation and data detection as shown in Fig. 2.6. The system requirement for channel estimation will be introduced in the following sections.

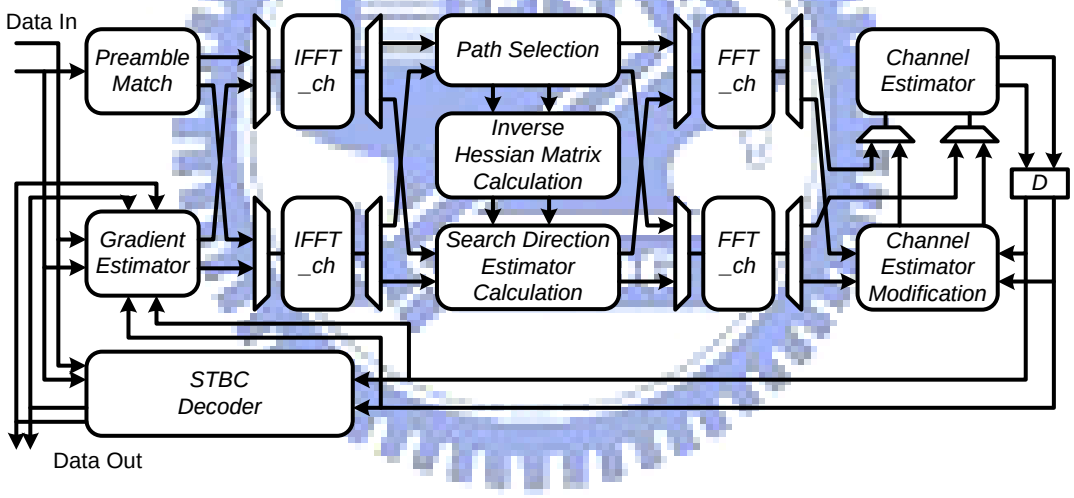


Fig. 2.6 Block diagram of decision feedback DFT-based channel estimation

There are two kinds of FFT processor in the receiver part, FFT_dem located of the synchronization block called OFDM demodulator. FFT_ch and IFFT_ch blocks are required in channel estimation block. The following sections will introduce the system specifications of these two kinds of FFT processor.

2.4.1 Specification of FFT Processor on Demodulation Path

The FFT_dem processor in Fig. 2.5 receives the data from synchronization block, and passes the data to channel estimation and space-time decoding. The input data format of FFT processor is like that in other OFDM communication system. However, the output ports have to buffer 2 OFDM symbol since we use 2×1 MISO system with STBC coding and DF DFT-based channel estimation. For this reason, we design a conventional memory-based FFT processor [11] with 5 memory banks shown in Fig. 2.7.

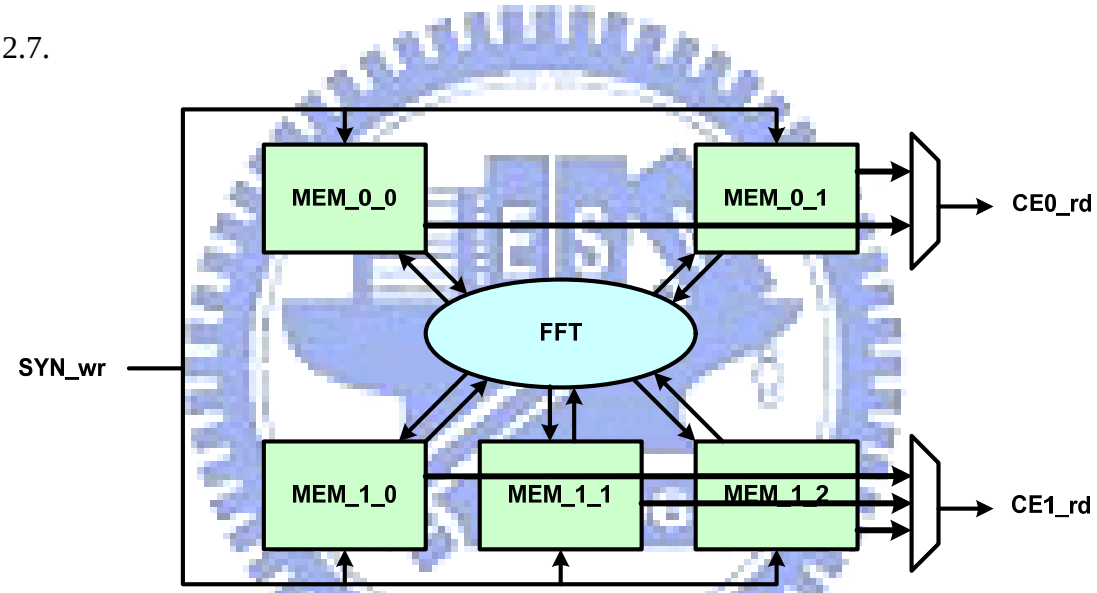


Fig. 2.7 FFT Processor with 5 shared memories

SYN_wr is the data from synchronization block, and only one of the memory banks would be written by synchronization block in an OFDM symbol time. Then, the written memory bank would be used to do FFT by the FFT processor. In the same time, the synchronization block is writing the data to another memory bank. After the two OFDM symbols in a STBC time slot have been calculate by FFT processor, the memories, which stored the FFT calculation result of this two OFDM symbols, would be read from channel estimation, called CE_rd, in two OFDM symbol time.

The time chart of 5 memory banks is shown in Fig. 2.8. At the first preamble

symbol, the data from synchronization block are written to MEM_1_0. At the second and the third symbols, the data from synchronization block are written to MEM_0_0 and MEM_1_1 while the data in MEM_1_0 are calculated by FFT processor and read by channel estimation. Furthermore, the memory operations for OFDM symbol index 12 is the same as index 0, thus the memory operations of 5 memory banks are repeated every 12 OFDM symbols.

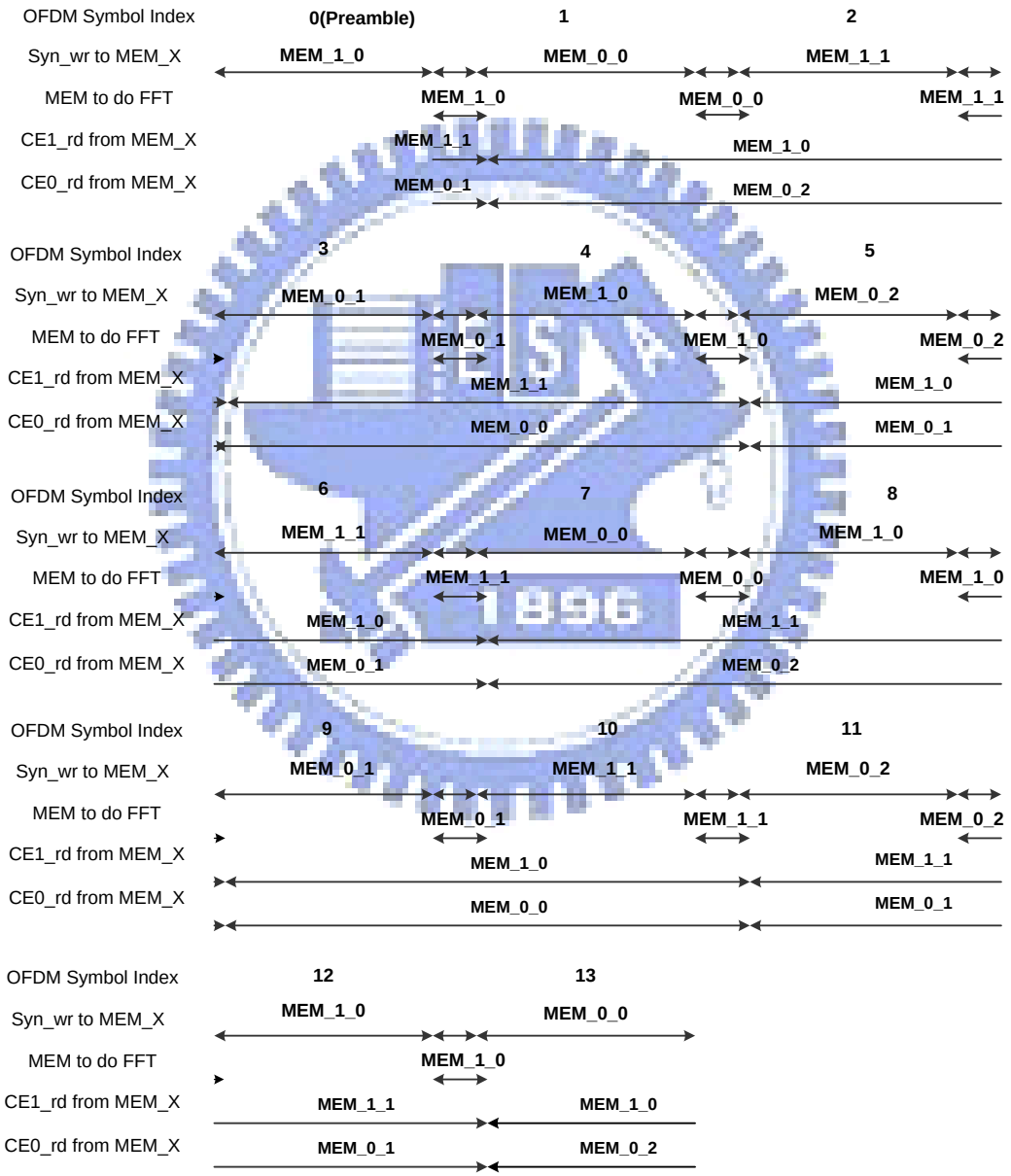


Fig. 2.8 Time chart for the 5 memory banks

2.4.2 Specification of FFT Processor in Channel Estimation

The FFT_ch and IFFT_ch blocks in decision feedback DFT-based channel estimation (DF DFT-based CE) are shown in Fig. 2.9. Before introducing the system requirement, we make a brief description of the DF DFT-based CE. The DF DFT-based CE has two parts. One is initial channel gain calculated by using the preamble signals. The operational blocks are preamble match block, two IFFT_ch blocks, path selection block, inverse hessian matrix calculation, two FFT_ch blocks, and channel estimator block. The channel gain should be calculated within 2 OFDM symbol time. The second part is channel gain tracking loop. The operational block s are gradient estimator, two IFFT_ch blocks, search direction estimator calculation, two FFT_ch blocks, channel estimator modification block, and the channel estimator block. The channel gain is calculated by tracking loop with 2 iterations. At the first iteration, the channel gain variance for the channel estimator modification block is determined by the pilot signals, called global tracking, since the pilot signals have higher SNR than data signals. At the second iteration, variance is determined by the data signals, called local tracking. Both of two parts can use the same IFFT_ch blocks and FFT_ch blocks.

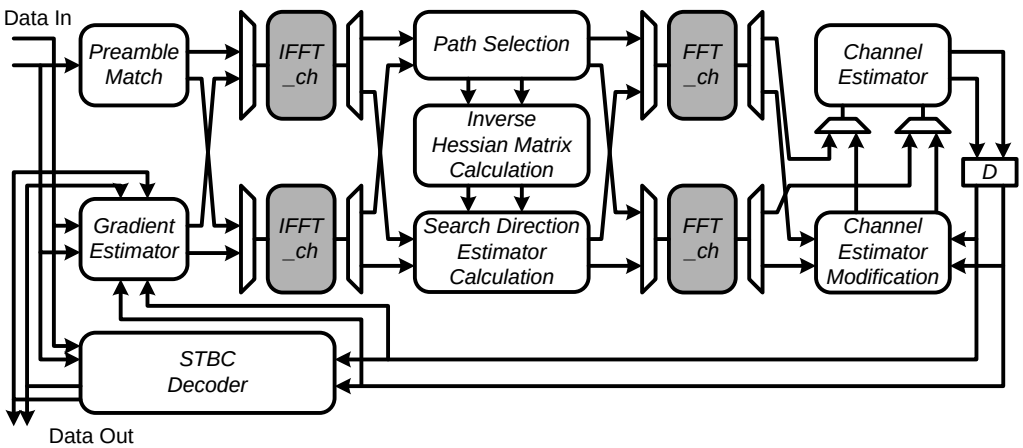


Fig. 2.9 FFT processor in decision feedback DFT-based channel estimation

Since the channel estimation included tracking loop, the channel gain should be calculated within 2 OFDM symbol time before the data buffers for channel estimation in Fig. 2.7 are updated; thus, data latency is an important issue to implement the channel estimation block into hardware. With this purpose, a parallel-in-parallel-out (PIPO) FFT/IFFT processor is necessary for not only increasing the throughput rate of FFT/IFFT processor but also increasing the throughput rate of other blocks in channel estimation block.

The DFT-based channel estimation has a special feature for the FFT_ch and IFFT_ch blocks. Only a subset of output data is required for IFFT_ch output ports. Also, the input data of FFT_ch block may have several zero points, which are no required to be computed with other non-zero points. The FFT processors design for only some subset of input or output points are called partial FFT [23]. The thesis will introduce the idea of partial FFT processor design for DFT-based channel estimation.

Finally, there are two purposes of FFT processor design, one is a FFT processor with parallel-in-parallel-out in normal order, and the other is partial FFT processor design. The thesis will focus on the FFT processor design with parallel-in-parallel-out in normal order. The partial FFT processor design concept will be introduced in the end of next chapter.

Chapter 3

FFT Algorithms and Architectures

3.1 Concept of FFT Algorithms

Discrete Fourier Transform (DFT) is a key block in OFDM communication system, and it is widely used in many applications; however, its computational complexity is so high that implementation of DFT algorithm directly seems not feasible to meet low cost design goal. Fortunately, early contributors, particularly Cooley and Turkey in 1965 [12], employed the redundancy of DFT operations by iteratively decomposing the computation, called radix-2 FFT algorithm, to reduce the computation complexity from $O(N^2)$ to $O(N\log_2 N)$. Based on Cooley and Turkey's FFT algorithm, various FFT algorithms were later developed, which provide flexible choices for implementation.

According to the ways of decomposing DFT, there are two types of FFT algorithms: one is the decimation-in-time (DIT) decomposition, which decomposes the time domain input sequence into successively smaller subsequences; the other is the decimation-in-frequency (DIF) decomposition, which alternately decomposes the frequency domain output sequence into smaller subsequences.

The basic N-point DFT equation is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{k \cdot n} \qquad (3.1)$$

where $W_N^{k \cdot n} = \exp(-j2p\,nk / N)$ is the DFT coefficient. Since a complex number multiplied with a coefficient is equivalent to a vector rotation, the DFT coefficient is also called twiddle factor.

The key feature of the FFT algorithm is to divide a complete DFT operations into several small point DFT operations; moreover, the FFT algorithm also uses the symmetry property of the twiddle factors. First, radix-2 FFT algorithm use the symmetry property of $W_N^{k\cdot n+\frac{N}{2}} = -W_N^{k\cdot n}$; then, we can reduce number of multiplications in Eq. (3.1) by half as shown in Eq. (3.2).

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n)\cdot W_N^{k\cdot n} + \sum_{n=0}^{\frac{N}{2}-1} x(n+\frac{N}{2})\cdot W_N^{k\cdot\left(n+\frac{N}{2}\right)} = \sum_{n=0}^{\frac{N}{2}-1}\left[x(n)-x(n+\frac{N}{2})\right]\cdot W_N^{k\cdot n} \quad (3.2)$$

Another symmetry feature is in its phase difference of ± 90° as $W_N^{k\cdot\left(n+\frac{N}{4}\right)} = -j\cdot W_N^{k\cdot n}$. Multiplying a complex number with – j, we can just exchange the real part and imaginary part, and then negate the imaginary part. Therefore, we can reduce the computational complexity of Eq. (3.1) by using Eq. (3.3).

$$A\times W_N^{k\cdot n} + B\times W_N^{k\cdot\left(n+\frac{N}{4}\right)} = (A-jB)\times W_N^{k\cdot n} \quad (3.3)$$

Finally, symmetry feature of its phase difference of ± 45° is also common in FFT algorithms. Based on the symmetry, the equation can be reduced to

$$\begin{aligned} &A\times W_N^{k\cdot n} + B\times W_N^{k\cdot\left(n+\frac{N}{8}\right)} = (A+\frac{1}{\sqrt{2}}(1-j)\times B)\times W_N^{k\cdot n} \\ &\frac{1}{\sqrt{2}}(1-j)\times B = \frac{1}{\sqrt{2}}(1-j)\times(c+jd) = \frac{1}{\sqrt{2}}((c+d)+(d-c)j) \end{aligned} \quad (3.4)$$

The multiplication of $\frac{1}{\sqrt{2}}$ can be realized by constant multiplications, which may be customized to shifter and adder, and will be demonstrate in Chapter 4.

According to the symmetry of twiddle factors, the computation complexity of DFT operation can be reduced to a fraction of the original operation. We will take a example of radix-2 DIF FFT algorithm in following subsection.

3.1.1 Radix-2 DIF FFT Algorithm

The DIF FFT Algorithm is decomposed the frequency domain output sequence

into small subsequence. Here we take a example of radix-2 DIF FFT algorithm. The radix-2 DIF FFT algorithm divided the frequency domain sequence into even and odd parts and using the symmetry of twiddle factor in Eq. (3.2), as shown below.

$$X(k_1+2k_2)=\sum_{n_2=0}^{\frac{N}{2}-1}\sum_{n_1=0}^{1}x(\frac{N}{2}n_1+n_2)\cdot W_N^{(k_1+2k_2)\cdot(\frac{N}{2}n_1+n_2)}$$

$$=\sum_{n_2=0}^{N/2-1}\underbrace{\left(\sum_{n_1=0}^{1}x(\frac{N}{2}n_1+n_2)\cdot W_2^{k_1\cdot n_1}\right)}_{\text{2-point DFT}}\cdot\underbrace{W_N^{k_1\cdot n_2}}_{\text{twiddle factor}}\cdot W_{N/2}^{k_2\cdot n_2}\quad\text{N/2-point DFT}$$

$$,\quad\begin{cases}n_1=0,1\\ n_2=0,1,\ldots,(N/2)-1\\ k_1=0,1\\ k_2=0,1,\ldots,(N/2)-1\end{cases}\qquad(3.5)$$

The DFT operation can be divided into 2 stages, one is 2-point DFT, and another is N/2-point DFT, which is shown below

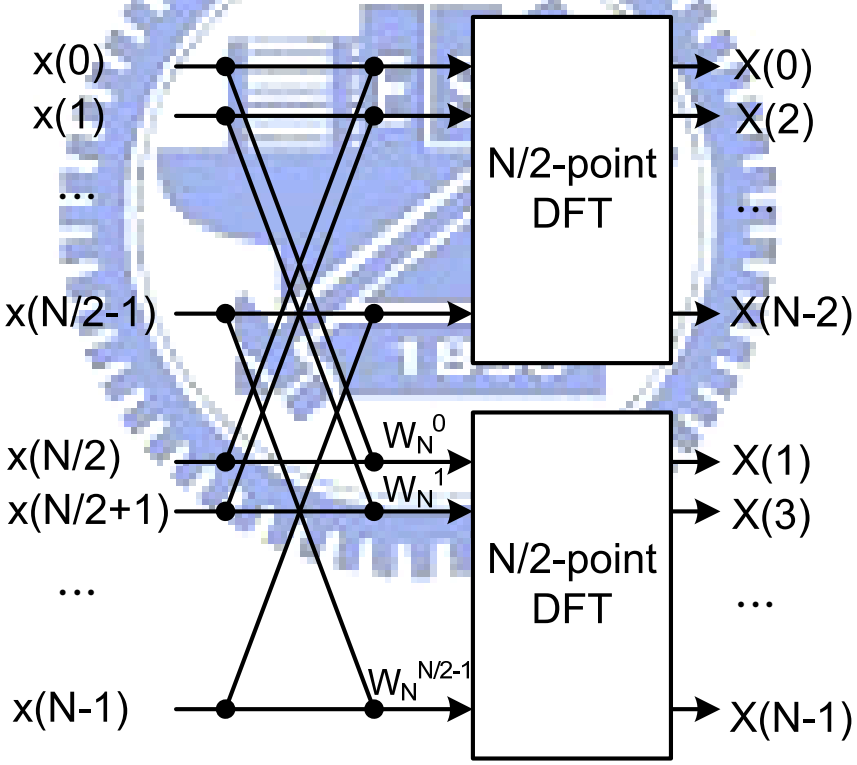


Fig. 3.1 Radix-2 DIF FFT algorithm architecture

After the first decomposition, the N-point DFT operation can be divided into N/2 2-point DFT operation and 2 N/2-point DFT operation, where the 2-point DFT is well known that the operation can be realized as a radix-2 butterfly (BF) module, shown as Fig. 3.2.

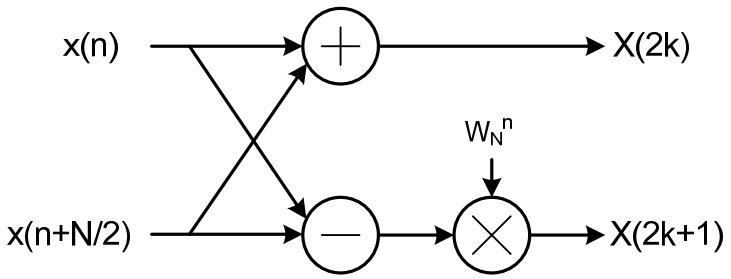


Fig. 3.2 Radix-2 butterfly module

Similar to the first decomposition, we can further decompose the N/2-point DFTs into even smaller DFTs until all DFTs are decomposed into 2-point DFT.

3.2 Concept of FFT Architectures

The FFT processor architecture design can be simply divided into two types, one is pipeline-based FFT architecture [13-15], and the other is memory-based FFT architecture [16-17]. Pipeline-based FFT architecture has the advantage of high throughput rate and low data latency, but it also has the disadvantage of high hardware cost; in contrast, memory-based FFT architecture has low hardware cost but high data latency.

For both of the FFT processor architectures, to increase the FFT processor throughput rate, high working clock rate is the simplest way to meet the throughput constrain; however, it will also increase the FFT processor hardware cost and power consumption. In this chapter, we will discuss different architectures for high throughput FFT processors with multi-input-and-multi-output in normal order.

3.2.1 Pipeline-Based FFT Architecture

The pipeline-based FFT architectures are the most popular architectures in many applications because they are designed for high speed performance and sequence of data input; but, in order to make the output data in normal order, they usually need a

reorder buffer in output stage, which regular a very high hardware cost. The best way to obtain the pipeline-based architecture is through vertical projection of signal flow graph (SFG). Fig. 3.3 shows an example to explain vertical projection mapping of 8-point radix-2 DIF FFT.

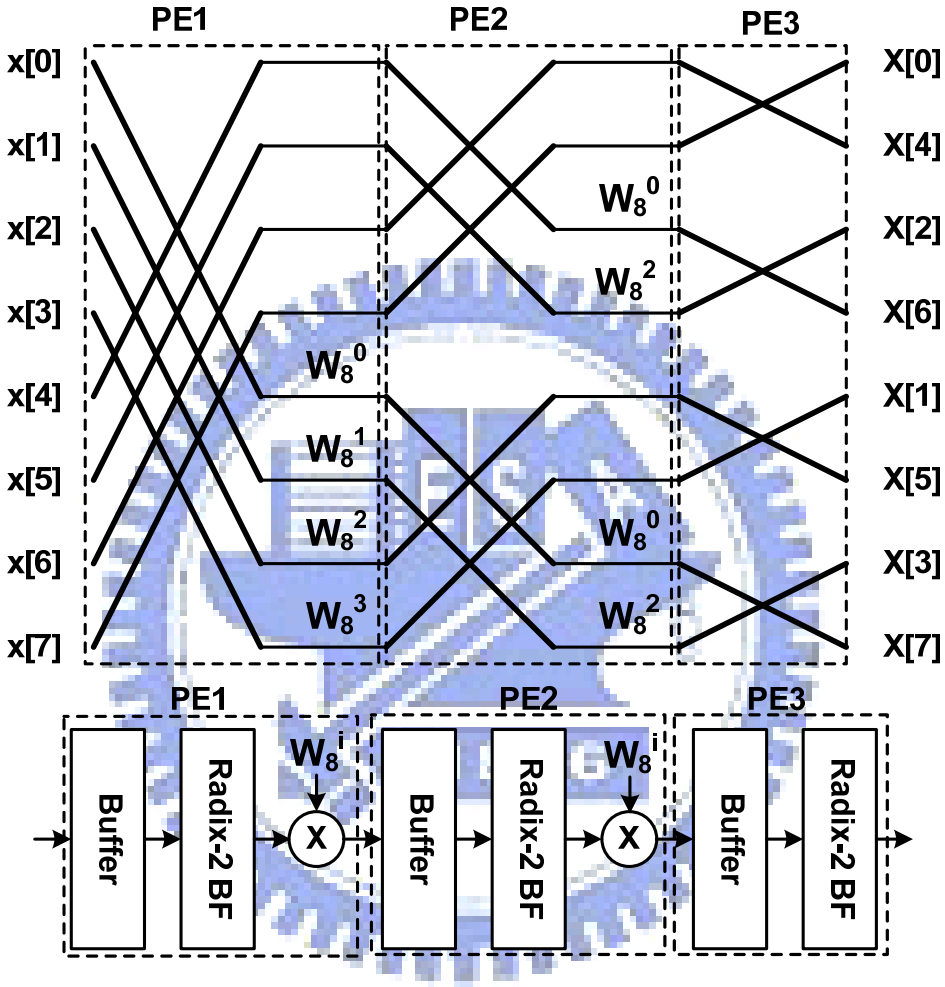


Fig. 3.3 Vertical projection mapping of 8-point radix-2 DIF FFT

Each stage obtained by vertical projection is called a processing element (PE), which contains a delay buffer (Buffer), a radix-2 butterfly unit (Radix-2 BF), and a complex multiplier. The delay buffer is used to reorder the data input for each stage butterfly unit. There are two types of the delay buffer, one is called delay-feedback (DF), and the other is called delay-commutator (DC). According to the structure difference, pipeline-based FFT architecture can be divided into three types: single-path delay feedback (SDF) architecture, single-path delay commutator (SDC)

architecture, and multi-path delay commutator (MDC) architecture. Since the SDC architecture can provide only one-path output data stream, similar to SDF architecture, and hardware cost is between the SDF and MDC architectures, the SDC architecture can not provide parallel data stream with least hardware cost. Here we only focus on SDF and MDC architectures. In the following subsections, we will introduce different radix-r SDF and MDC pipeline-based FFT architectures, where r is the radix number for the decimation-in-time (DIT) or decimation in frequency (DIF) algorithm.

3.2.1.1 Radix-r Multi-Path Delay Commutator Architecture

Radix-r MDC architecture [18-19] uses commutator to break the input data into r parallel data streams flowing forward with correct ordering for the data entering the butterfly unit by proper delays. Here are two examples to introduce MDC architecture in the following discussions.

(1). Radix-4 Multi-Path Delay Commutator (R4MDC) Architecture

Fig. 3.4 shows a 64-point FFT with radix-4 multi-path delay commutator (R4MDC) architecture. In Fig. 3.4, the elements of the R4MDC architecture are commutators, shift registers, and radix-4 butterfly units. The butterfly unit is also called arithmetic element (AE). At the beginning, the first 16 points of input data are delay at the first line of AE1's inputs, the next 16 points are delay at the second line, and the next 16 points are delay at the third line. When the 49^{th} point of input data coming at the forth line, the first butterfly is computing at AE1. With proper delays and commutation between each AE, the input data of each AE has correct ordering to compute a radix-4 butterfly in each AE. Finally, the output data of AE3 are 2bit-reverse order of the input data order.

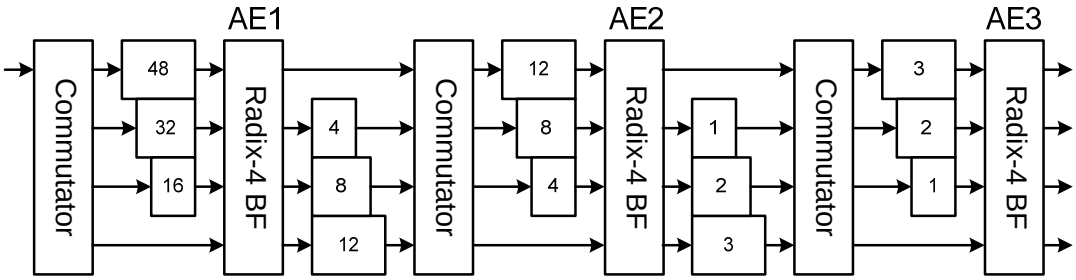


Fig. 3.4 64-point FFT with R4MDC architecture

In order to revise the 64-point FFT of R4MDC architecture with multi-input and multi-output in normal order, we have to replace the first stage and add a reorder stage at the output stage, which is shown in Fig. 3.5.

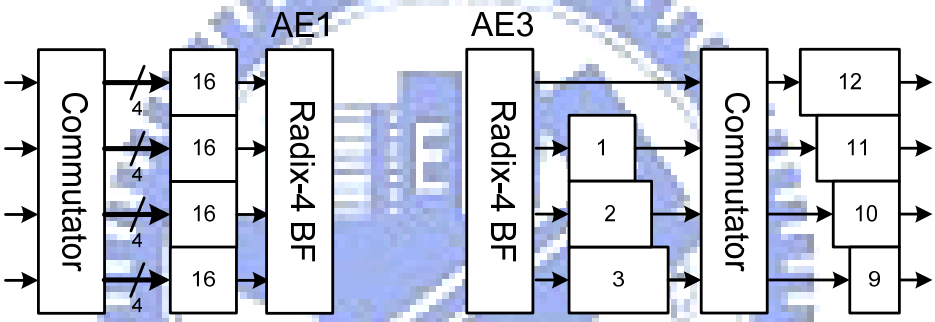


Fig. 3.5 Modified input stage and output stage of 64-point R4MDC architecture

For multi-input in normal order, the first stage has to change the commutator from one input to four inputs. And, in order to write four data in one cycle to one of the input shift registers of AE1, the shift registers have to be changed into random access registers. Thus, the first to the third line of AE1's inputs shifter registers are changed into $\frac{N}{4}$ random access registers. Furthermore, the fourth line have to add a 4×4 random access registers to buffer the input data because the fourth line has four input data and one output data.

For multi-output in normal order, the output stage has to add a reorder stage to reorder the output data from bit-reverse order to normal order. By using the similar way of delay commutator, the output data order will be changed into normal order.

For N-point FFT computation, R4MDC needs $\frac{11}{4}N-4$ registers, $3\cdot(\log_4 N-1)$ complex multipliers, and $8\cdot\log_4 N$ complex adders. The latency is $\frac{11}{16}N-5$ cycles.

(2). Radix-8 Multi-Path Delay Commutator (R8MDC) Architecture

Radix-8 multi-path delay commutator (R8MDC) is similar to R4MDC architecture, it use radix-8 algorithm with MDC architecture, and it can provide higher throughput rate than R4MDC architecture with 8 parallel data streams. But, it also has more delay buffers and other arithmetic elements. The 512-point FFT with R8MDC architecture is shown in Fig. 3.6.

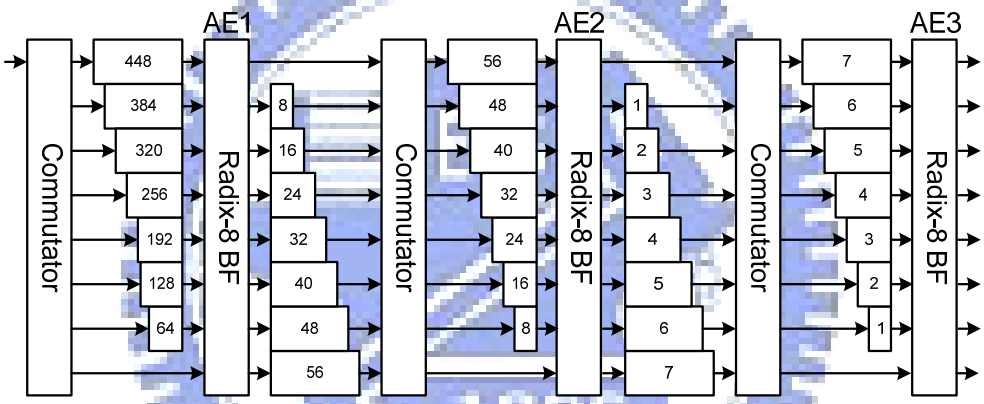


Fig. 3.6 512-point FFT with R8MDC architecture

Also, for multi-output in normal order, the architecture has to revise the first stage and add a reorder stage at output stage, which is shown in Fig. 3.7. The input stage and output stage are similar to R4MDC architecture with multi-input and multi-output in normal order. As a result, reorder stage has more delay buffer when higher radix is used in MDC architecture.

For N-point FFT computation, R8MDC needs $\frac{23}{8}N-8$ registers, $7\times(\log_8 N-1)$ complex multipliers, and $(24+2T)\times\log_8 N$ complex adders, where the parameter T indicates the number of adders required in the implementation of multiplications by constant values. The latency is $\frac{23}{64}N-9$ cycles.

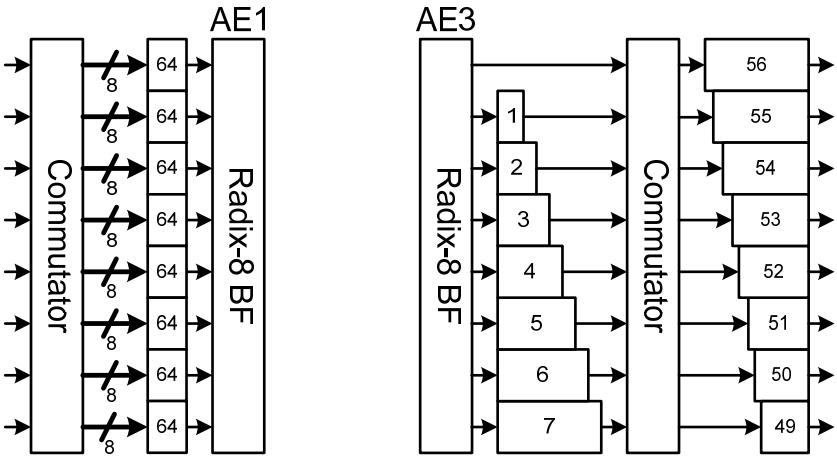


Fig. 3.7 Modified input stage and output stage of 512-point R8MDC architecture

3.2.1.2 Radix-r Single-Path Delay Feedback Architecture

Unlike multi-path delay commutator (MDC) architecture, single-path delay feedback (SDF) architecture combines the commutator and the radix-r butterfly unit, and uses delay feedback method to reuse the delay buffer of each stage to reorder the data input of butterfly unit. The SDF architecture's hardware is less than the MDC architecture's, but the data latency is more than the MDC architecture's. Moreover, the SDF has only one path between butterfly units, the throughput rate can't be higher even it uses higher radix FFT algorithm. For input and output data in normal order, it needs a reorder buffer at output stage, and, the buffer size is about N/2 for N-point DFT with SDF architecture. Also, we take two cases of SDF architecture in the following discussions.

(1). Radix-2 Single-Path Delay Feedback (R2SDF) Architecture

The radix-2 single-path delay feedback (R2SDF) architecture combines radix-2 MDC architecture's commutator and radix-2 butterfly unit in R2SDF's radix-2 butterfly unit shown in Fig. 3.8. Without 2 parallel data streams from the radix-2 butterfly unit output to the next stage, R2SDF only has one output to the next stage,

and the other output is feedback to store in delay buffer; therefore, it is called single-path delay feedback architecture.

For N-point FFT computation, R2SDF needs N–1 registers, ($\log_2$N–1) complex multipliers, and 2 ×$\log_2$N complex adders. The latency is N–1 cycles without reorder buffer.

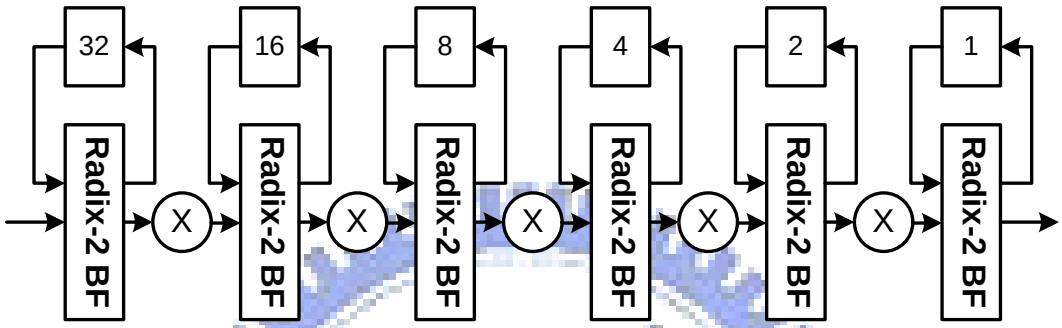


Fig. 3.8 64-point FFT with radix-2 SDF architecture

(2). Radix-8 Single-Path Delay Feedback (R8SDF) Architecture

The block diagram of 64-point radix-8 single-path delay feedback (R8SDF) architecture is shown in Fig. 3.9. It has less multiplier than the R2SDF architecture, for 64-point FFT architecture, R8SDF can save 80% of complex multipliers; but it also has more register banks to store the data for BF unit, which may have more power consumption.

For N-point FFT computation, R8SDF needs N–1 registers, ($\log_8$N–1) complex multipliers, and (24+2T) ×$\log_8$N complex adders. The latency is N–1 cycles without reorder buffer.

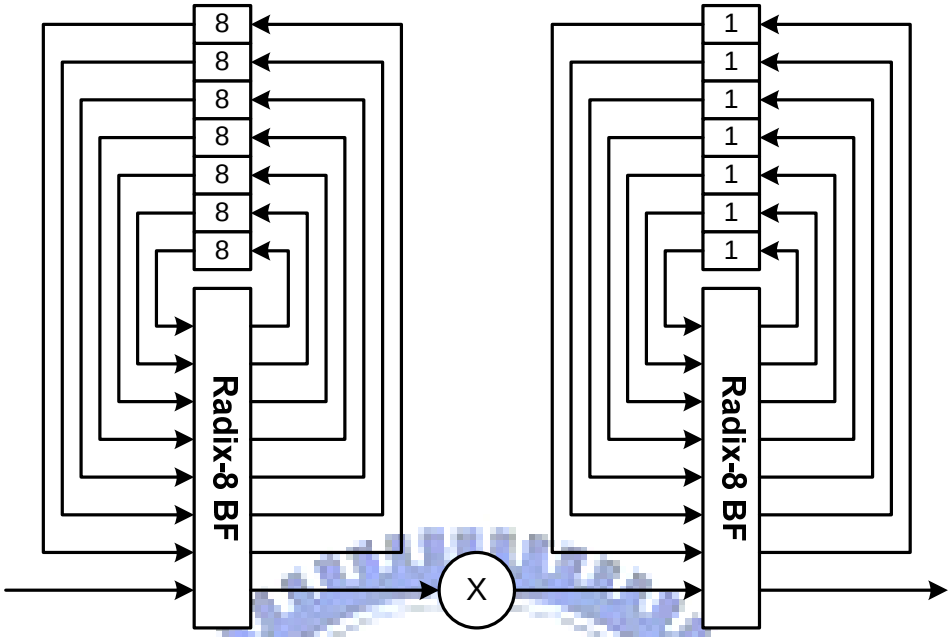


Fig. 3.9 64-point FFT with R8SDF architecture

(3). Radix-2/4/8 Single-Path Delay Feedback ($R2^3SDF$) Architecture

Radix-2/4/8 single-path delay feedback ($R2^3SDF$) architecture is based on R2SDF architecture with radix-8 FFT algorithm shown in Fig. 3.10, and, it replaces the radix-2 butterfly unit with the radix-8 FFT processing element, which is shown in Fig. 3.11. The numbers of required complex multiplier are the same as R8SDF architecture, and the numbers of required complex adders are less than R8SDF architecture; moreover, the partitions of registers are less than R8SDF architecture, which may have less power consumption.

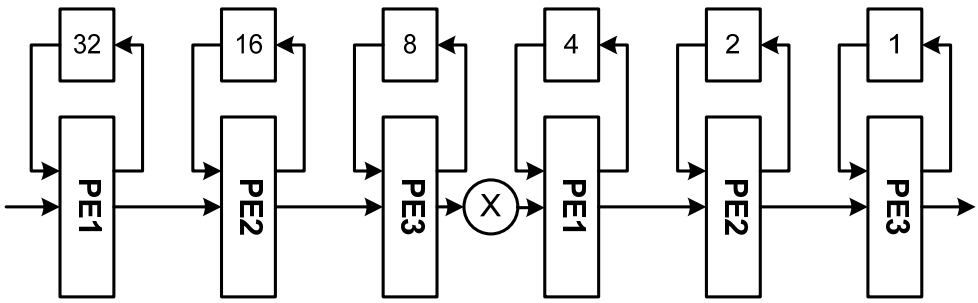


Fig. 3.10 64-point FFT with $R2^3SDF$ architecture

For N-point FFT computation, R2^{3}SDF needs N–1 registers, ($\log_8$N–1) complex multipliers, and (6+2T) · $\log_8$N complex adders. The latency is N–1 cycles without reorder buffer.

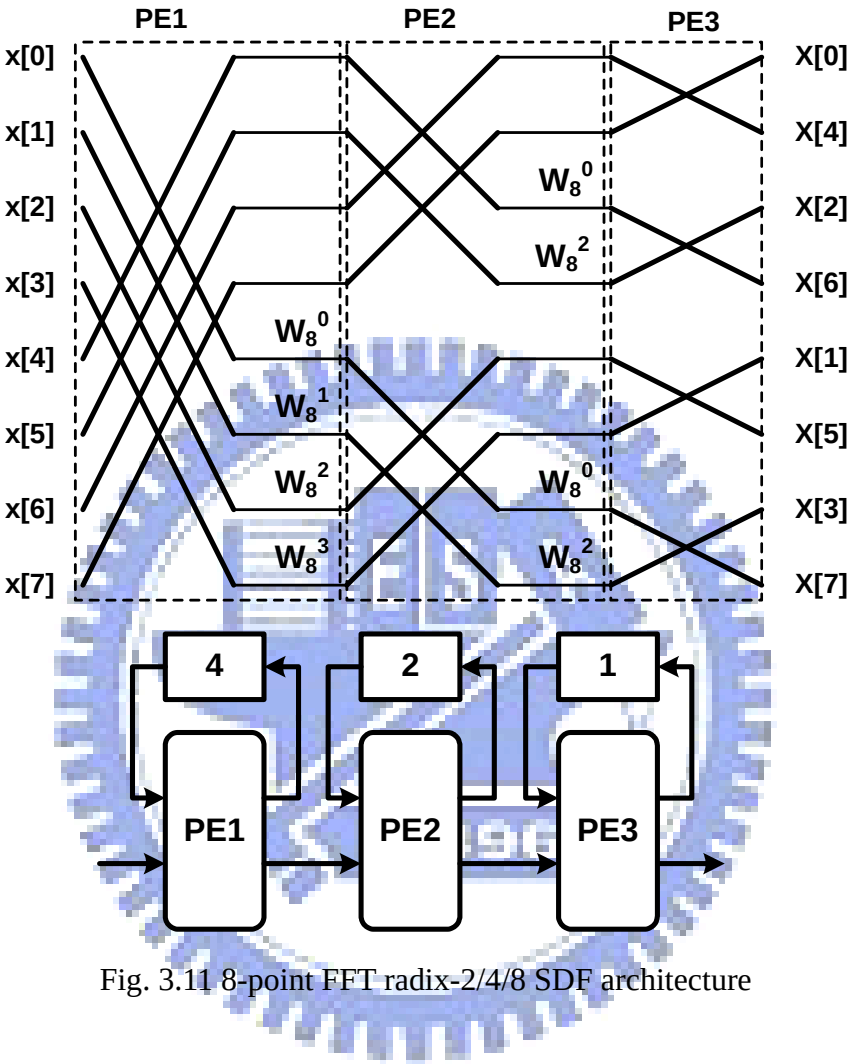


Fig. 3.11 8-point FFT radix-2/4/8 SDF architecture

3.2.2 Memory-Based FFT Architecture

Memory-based FFT architecture, unlike pipeline-based FFT architecture, only has a few arithmetic elements (AE), which also called processing element (PE). There are two advantage of using memory-based FFT architecture: One is that the hardware area of the processing elements for N-point DFT computation is the same even N is very large; the other is that the total number of memory banks are less than pipeline-based FFT architecture because it used a few PE and need less read or write

operations in the same time. Fig. 3.12 shows a radix-8 memory-based FFT architecture, it only has one radix-8 butterfly unit and 8 memory banks.

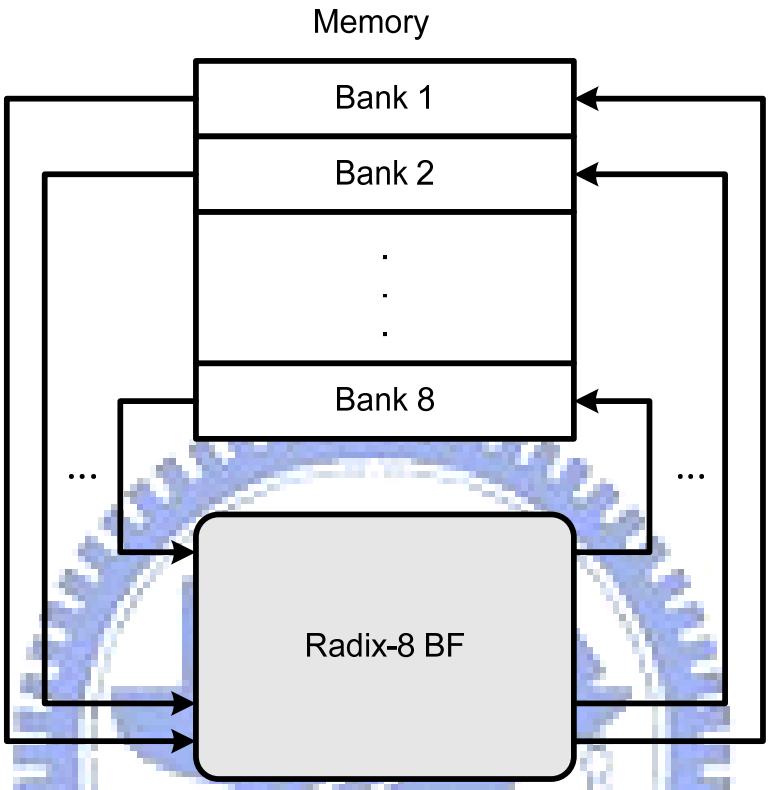


Fig. 3.12 Radix-8 memory-based (R8M) FFT architecture

For multi-input in normal order, different input data in one cycle should write to different memory banks, but, this requirement is conflict with radix-r FFT algorithm for memory-based architecture. Similar to the MDC architecture, radix-r memory-based FFT architecture can add reorder stage at the input stage for parallel data to be written to different memory banks. Also, for multi-output in normal order, it needs a reorder stage at the output stage.

Another choice for memory-based FFT architecture with multi-input and multi-output in normal order is rearrangement of data in memory with higher control complexity. Next chapter will show the proposed FFT processor architecture based on this concept.

For N-point FFT computation, R8M needs $\frac{7}{8}$N+56 registers and N words memory with 8 memory banks, 7 complex multipliers, and (24+2T) complex adders. The latency is $\frac{15}{64}N-8+\frac{N}{8}\log_8 N$ cycles.

3.3 Comparison of Different FFT Architecture

Table 3-1 Comparison of different FFT architecture

	R2SDF	**R8SDF**	**R2^{3}SDF**	**R4MDC**	**R8MDC**	**R8M**
Complex Multipliers	$\log_2$N−1	$\log_8$N−1	$\log_8$N−1	3· $\log_4$N−1)	7· $\log_8$N−1)	7
Complex Adders	2· $\log_2$N	(24+2T) · $\log_8$N	(6+2T) · $\log_8$N	8· $\log_4$N	(24+2T) · $\log_8$N	24+2T
Memory Size	N−1	N−1	N−1	7N/4+12	15N/8+56	N
Reorder Buffer Size	N/2	7N/8	N/2	N−16	N−64	N−64
Data Latency	3N/2−1	15N/8−1	3N/2−1	11N/16−5	23N/64−9	15N/64−8+ (N/8)$\log_8$N
Throughput Rate	R	R	R	4R	8R	8R

The comparison of different FFT architecture with multi-input and multi-output in normal order is shown in Table 3-1, where the N is the FFT size and R is the internal clock rate of the FFT processor. Due to the FFT algorithms, all architecture need reorder buffer at input stage or output stage, and the hardware cost of reorder buffer is so high that the conventional FFT architecture can't provide an efficient way to make the output sequence in normal order. For this reason, we have to develop a FFT processor providing high throughput rate with multi-input and multi-output in normal order in an efficient way for low hardware cost. As the goal of low hardware cost, radix-8 memory-based FFT architecture has the least hardware cost for high throughput rate with the same clock working frequency. However, it also needs a very

large reorder buffer. Therefore, the main issue of the FFT architecture with multi-input and multi-output in normal order is to reduce the reorder buffer. The proposed FFT architecture can provide high throughput rate with multi-input and multi-output in normal order, and does not need any reorder buffer. It will be introduced in next chapter.

3.4 Partial FFT Design

3.4.1 Concept of Partial FFT

Partial FFT design is a study of redundancies of the standard FFT algorithm due to a reduction in either the number of input or output points. For most applications, the input and the output sequence of the DFT operation are equal, but, there are still some applications where the numbers of input and output points are different, such as DFT-based channel estimation. Hence, many researches of partial FFT design are presented to reduce the redundant operations of FFT algorithm. The thesis will introduce the partial FFT design in two points of view in the following subsections, one is concerned that only a subset of input or output points of DFT operation are computed, another point is concerned that multiple subsets of input or output points of DFT operation are computed. Finally, we propose a partial FFT design, combining the reducing methods with only a subset and multiple subsets of input or output points of DFT operation, suitable for DFT-based channel estimation.

3.4.2 DFT with only a Subset of Input or Output Points

There are two conditions we have to design a partial FFT with only a subset of input or output points, one is that only a narrow spectrum is interested but the resolution within the band has to be very high; the other is that a very high resolution

spectrum is to pad the input sequence with a large number of zeros. It usually use a regular FFT to compute the results, but if the number of nonzero input or the number of output concerned is small compared with the DFT length, it is very inefficient. The pruning algorithm [25][26] and transform decomposition [27] is presented for efficient DFT computation with only a subset of input or output points. Because the transform decomposition method is not suitable in our application, we only introduce the pruning algorithm in the following.

The pruning algorithm is first developed by Markel [25] for computing only a subset of input or output points. An example of Markel's pruned 16-point FFT with a subset of nonzero input is shown in Fig. 3.13, where the Markel's pruning algorithm is based on radix-2 DIF FFT algorithm. We focus on the case that the nonzero input points are from the first L points of input sequence because this case is similar to the case of FFT processor in DFT-based channel estimation. As the result from Fig. 3.13, it reduces $N \times \log_2(N/L)$ complex additions and $(N/2)\log_2(N/L) - N + L$ complex multiplications than the original FFT algorithm, where L is a power of 2.

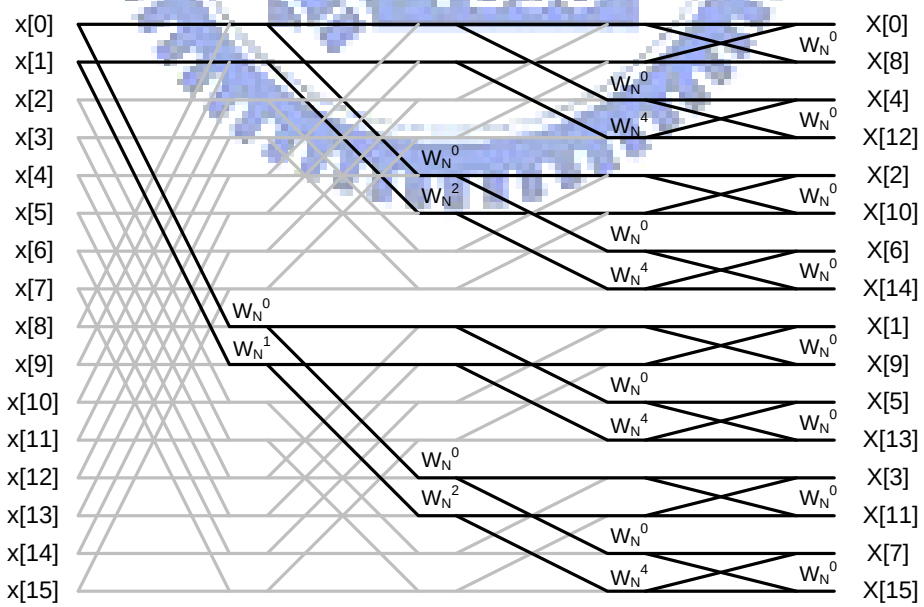


Fig. 3.13 Markel's pruned 16-point FFT with a subset of nonzero input (L=2)

The Skinner develops more efficient pruning algorithm [26] than that of Markel

as shown in Fig. 3.14. However, Skinner's algorithm is only for L is a power of 2. It is achieved by pruning a decimation-in-time algorithm instead of the decimation-in-frequency that Markel's algorithm is based on. In Skinner's pruning algorithm, the first $\log_2(N/L)$ stages contain no complex additions and no complex multiplications, and it means that it reduces $N \times \log_2(N/L)$ complex additions and $(N/2)\log_2(N/L)$ complex multiplications. Therefore, the Skinner's algorithm with a subset of nonzero input saves N-L of complex multiplications as compared to Markel's algorithm when L is a power of 2.

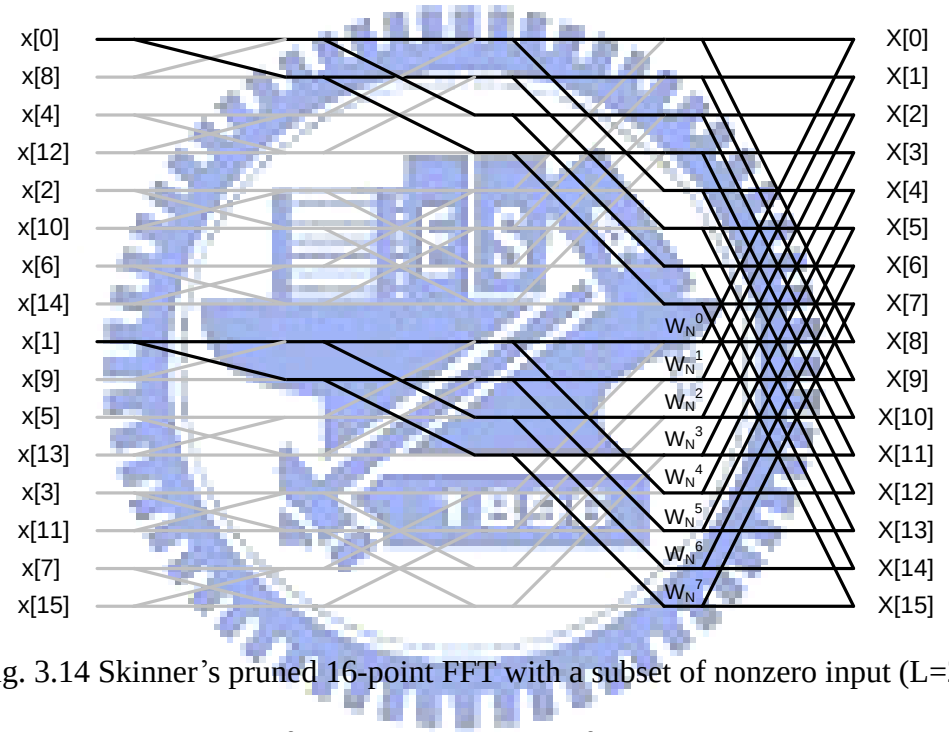


Fig. 3.14 Skinner's pruned 16-point FFT with a subset of nonzero input (L=2)

The pruning algorithm for FFT with a subset of output points is also presented by Markel and Skinner as shown in Fig. 3.15 and Fig. 3.16. The Markel pruning algorithm is based on decimation-in-time algorithm while that of the Skinner's is based on decimation-in-frequency algorithm. The Markel's algorithm can reduce $N\log_2(N/L)-N+L$ of complex additions and $(N/2)\log_2(N/L)-N+L$ of complex multiplications, and the Skinner's algorithm can reduce $N \times \log_2(N/L)$ of complex additions and $(N/2)\log_2(N/L)$ of complex multiplications.

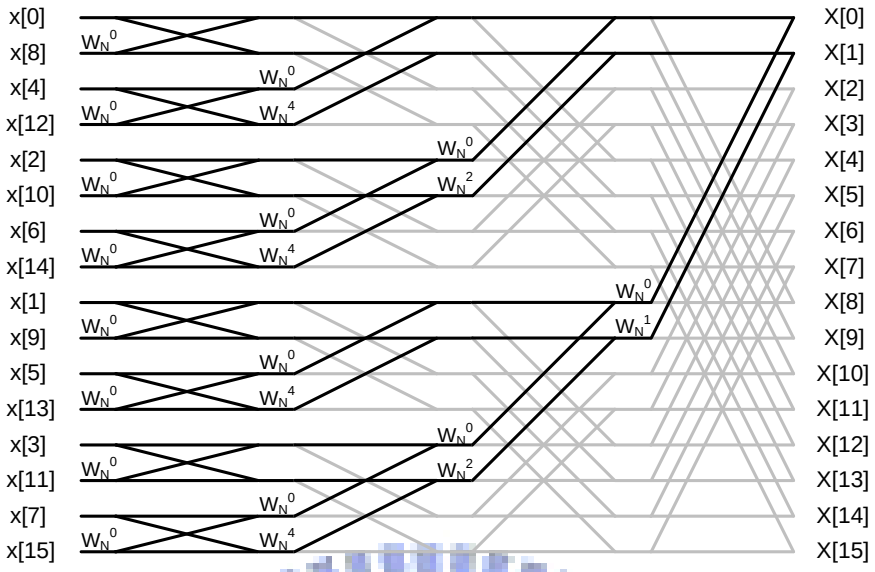


Fig. 3.15 Markel's pruned 16-point FFT with a subset of output points (L=2)

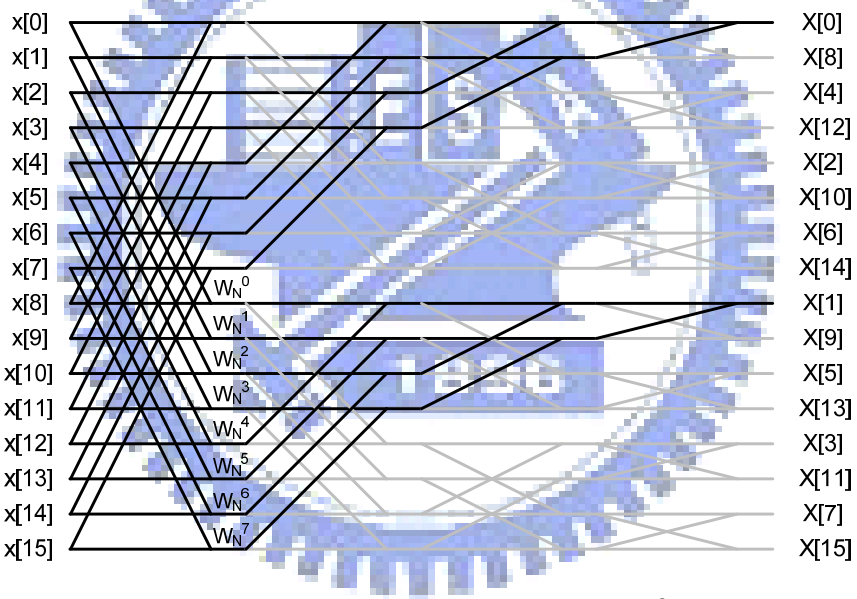
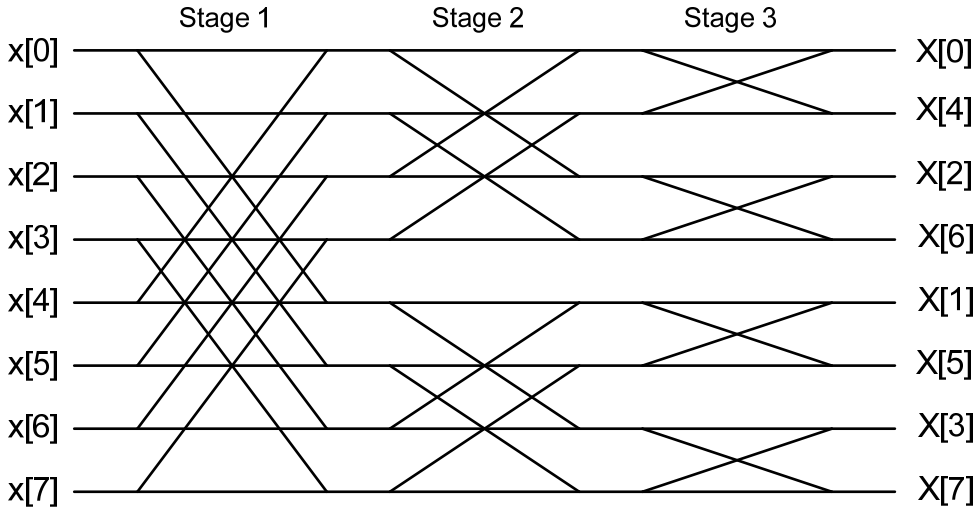


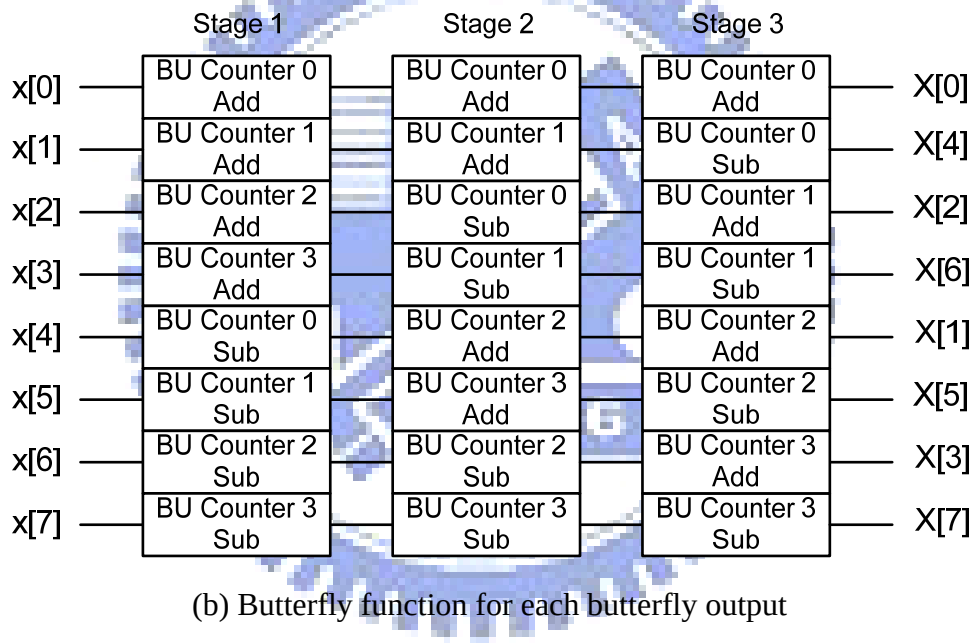
Fig. 3.16 Skinner's pruned 16-point FFT with a subset of output points (L=2)

3.4.3 DFT with Multiple Subsets of Output Points

Conventional partial FFT algorithm can only extract one subset of spectrum. An efficient partial FFT algorithm for DFT with multiple subsets of output points has been presented [28], which focus on the control of DFT with multiple subsets of output points, and an example of 8-point DFT based on the concept [28] is shown in Fig. 3.17.



(a) Signal flow graph of 8-point DFT



(b) Butterfly function for each butterfly output

Fig. 3.17 8-point DFT with butterfly function of each butterfly unit output point

The signal flow graph of 8-point DFT is shown in Fig. 3.17(a), and the butterfly function for each butterfly output is shown in Fig. 3.17(b). In order to reduce the redundant operations of butterfly unit, we have to decide the butterflies need to be computed and operations for needed butterflies, and an example of 8-point DFT with multiple subsets of output points is shown in Table 3-2.

Table 3-2 Control counter and function of FFT with partial output points

	Butterfly Counter	Butterfly Function
Stage 1	b_1b_0	$Q_0 = \{0,1\} \rightarrow$ Normal $Q_0 = 0 \rightarrow$ Addition $Q_0 = 1 \rightarrow$ Subtraction
Stage 2	Q_0b_0	$Q_1 = \{0,1\} \rightarrow$ Normal $Q_1 = 0 \rightarrow$ Addition $Q_1 = 1 \rightarrow$ Subtraction
Stage 3	Q_0Q_1	$Q_2 = \{0,1\} \rightarrow$ Normal $Q_2 = 0 \rightarrow$ Addition $Q_2 = 1 \rightarrow$ Subtraction

If the multiple output subcarriers, whose indices are [G_2 G_1 G_0], [H_2 H_1 H_0],[I_2 I_1 I_0],…, are interested in the system, the Q_n in the Table 3-2 can be defined as $Q_n = \{G_n \cup H_n \cup I_n \cup \ldots\}$; then, the possible results for Q_n are {0,1},{0},{1}. The needed butterflies and operations of the butterflies can be defined as butterfly counter and butterfly function in Table 3-2. In addition, b_1b_0 is the original butterfly counter counting from 0 to 3. It is clearly that all the butterflies should be computed in stage 1, the stage is defined in Fig. 3.17, for all possible result of output points, but if the Q_0 equals to 0 or equals to 1, all the butterflies only compute the addition or subtraction butterfly function. In stage 2, the butterflies should be computed only if butterfly counter of the butterflies equals to Q_0b_0, and the operations is decided by Q_1. Similar to stage 2, in stage 3, the butterflies should be computed only if butterfly counter of the butterflies equals to Q_0Q_1, and operations of the butterflies are decided by Q_2.

An example of DFT with multiple output points is shown in Fig. 3.18. The expected signal flow graph is shown in upper side, and the active operations and butterflies are shown in lower side. In stage 1 and stage 2, the active operations and butterflies meet the expected signal flow graph, but, in stage 3, the addition operation of butterfly counter 3 is a redundant operation due to the butterfly function control is shared with all butterflies. Although there are still redundant operations in this

algorithm, it provides an efficient way to simplify the control of partial FFT.

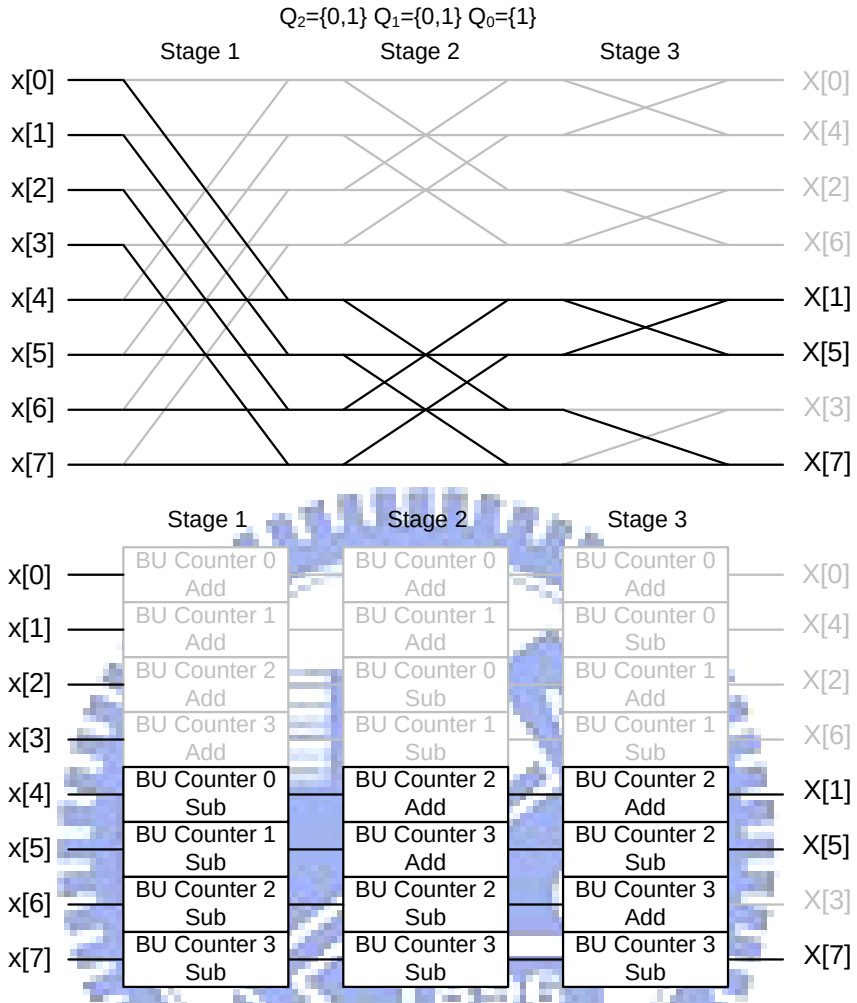


Fig. 3.18 Example of 8-point DFT with multiple subsets of output points

3.4.4 DFT with Multiple Subsets of Input and Output Points

Based on the algorithm presented in Section 3.4.3, we enhance the algorithm from only suitable for multiple subsets of output points to both multiple subsets of input and output points, and the modified butterfly counter and operations is shown in Table 3-3. Q_n represents the multiple output subcarriers' indices as mentioned in Section 3.4.3, and P_n represents the multiple nonzero input points' indices. The new operation of butterfly function, bypassing input values, is added due to that there are several zero input points of DFT operation. An example of DFT operation with multiple nonzero input and output points is shown in Fig. 3.19.

Table 3-3 Control counter and function of FFT with partial input and output points

	Butterfly Counter	Butterfly Function	
Stage 1	P_1P_0	$P_2 = \{0,1\}$	$Q_0 = \{0,1\} \rightarrow$ Normal $Q_0 = 0 \rightarrow$ Addition $Q_0 = 1 \rightarrow$ Subtraction
		$P_2 = 0$	$Q_0 = \{0,1\} \rightarrow$ Bypass upper input to both upper and lower output $Q_0 = 0 \rightarrow$Bypass upper input to upper output $Q_0 = 1 \rightarrow$Bypass upper input to lower output
		$P_2 = 1$	$Q_0 = \{0,1\} \rightarrow$ Bypass lower input to both upper and lower output $Q_0 = 0 \rightarrow$Bypass lower input to upper output $Q_0 = 1 \rightarrow$Bypass lower input to lower output
Stage 2	Q_0P_0	$P_1 = \{0,1\}$	$Q_1 = \{0,1\} \rightarrow$ Normal $Q_1 = 0 \rightarrow$ Addition $Q_1 = 1 \rightarrow$ Subtraction
		$P_1 = 0$	$Q_1 = \{0,1\} \rightarrow$ Bypass upper input to both upper and lower output $Q_1 = 0 \rightarrow$Bypass upper input to upper output $Q_1 = 1 \rightarrow$Bypass upper input to lower output
		$P_1 = 1$	$Q_1 = \{0,1\} \rightarrow$ Bypass lower input to both upper and lower output $Q_1 = 0 \rightarrow$Bypass lower input to upper output $Q_1 = 1 \rightarrow$Bypass lower input to lower output
Stage 3	Q_0Q_1	$P_0 = \{0,1\}$	$Q_2 = \{0,1\} \rightarrow$ Normal $Q_2 = 0 \rightarrow$ Addition $Q_2 = 1 \rightarrow$ Subtraction
		$P_0 = 0$	$Q_2 = \{0,1\} \rightarrow$ Bypass upper input to both upper and lower output $Q_2 = 0 \rightarrow$Bypass upper input to upper output $Q_2 = 1 \rightarrow$Bypass upper input to lower output
		$P_0 = 1$	$Q_2 = \{0,1\} \rightarrow$ Bypass lower input to both upper and lower output $Q_2 = 0 \rightarrow$Bypass lower input to upper output $Q_2 = 1 \rightarrow$Bypass lower input to lower output

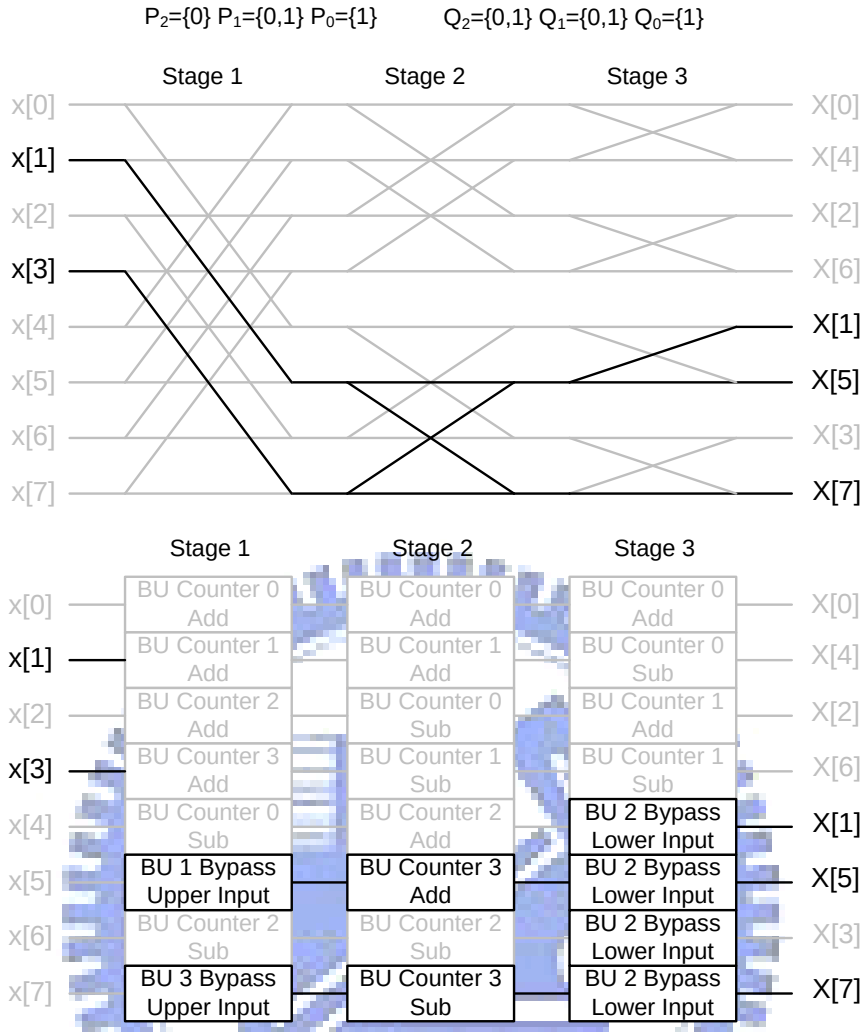


Fig. 3.19 Example of 8-point DFT with multiple subsets of input and output points

Table 3-3 also presents the dependency between the butterfly counter and the parameter Q_n and P_n in each stage. In the first stage, the butterflies needing to be computed is only dependent on the set of valid input points indices, which is the parameter P_n. In the second stage, the butterfly counter is dependent on half of the set of valid input point's indices and half of the set of valid output point's indices. In the final stage the butterfly counter is only dependent on the set of valid output points indices, Q_n. As the result, this algorithm reduces more redundant operations than Fig. 3.18 due to multiple zero input points, and it is helpful to design the partial FFT in DFT-based channel estimation, and it will be explained later.

3.4.5 Partial FFT Processor Design in DFT-Based Channel Estimation

There are two purposes for designing partial FFT in DFT-based channel estimation. The first one is that the partial FFT processor should compute the IDFT operations with N points of input and N×GI points of output and DFT operations with N×GI points of input and N points of output as discussed in Section 2.3. The second one is that the partial FFT should reduce the redundant operations due to the non regular input or output points. For example, the IDFT operations with zero input points of guard band and redundant output points of non-usable of multi-path response shall be avoided. Hence, we design the partial FFT for different purposes by the algorithm mentioned in Section 3.4.3 and Section 3.4.4.

The partial FFT processor specification of the proposed DF DFT-based CE in 802.16e baseband receiver is that the FFT size is 1024 points and the guard interval is 1/8. Thus, we have to design a partial FFT/IFFT processor for IDFT operation with 1024 points input transform to 128 points output and DFT operation with 128 points nonzero input transform to 1024 points output as shown in Fig. 3.20.

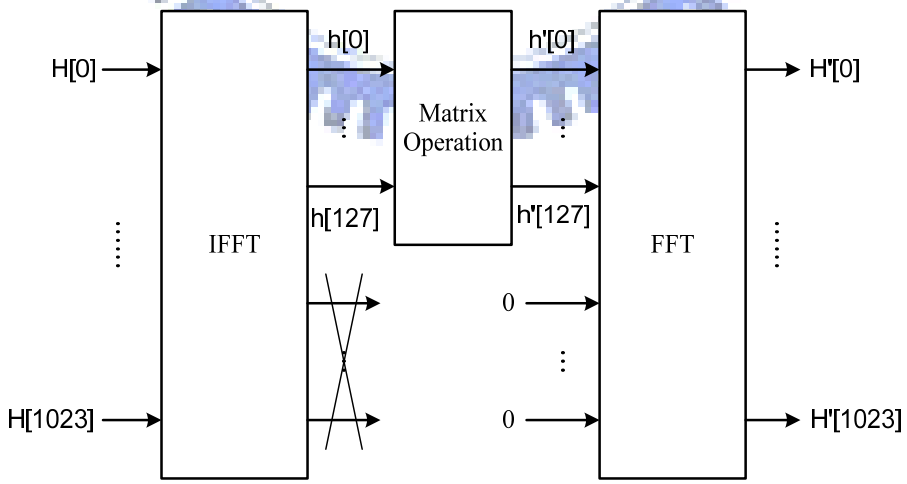


Fig. 3.20 System specification for the partial FFT/IFFT processor

For this purpose, the FFT and IFFT blocks in DF DFT-based CE is well suitable for partial FFT/IFFT design with only a subset of input / only a subset of output

points. A pipeline-based architecture for the partial FFT is presented in Fig. 3.21, which used the concept of Section 3.4.3 and combined the IFFT and FFT in the same hardware.

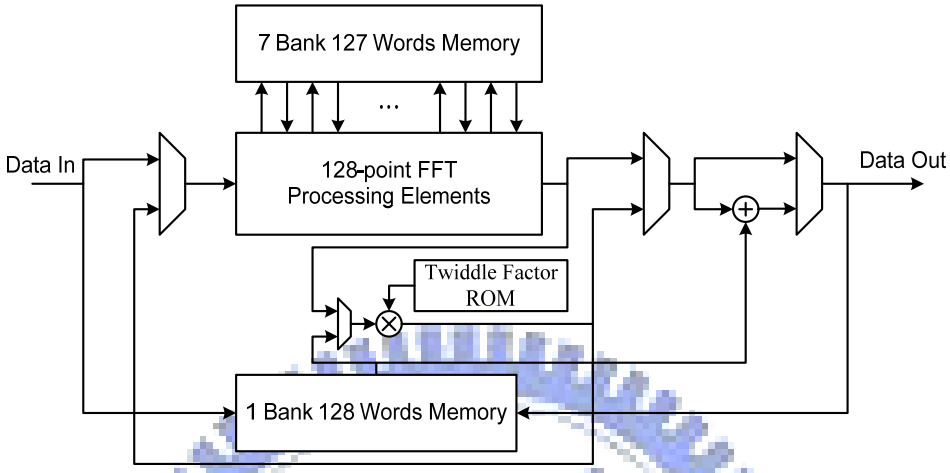


Fig. 3.21 Pipeline-based partial FFT/IFFT processor

The active block of partial FFT/IFFT processor in IFFT mode is shown in Fig. 3.22. Due to the DIF algorithm, the 1024-point IFFT operation can be partitioned into 8 128-point IDFT operations and combining the output data of 128-point IDFT operations with a radix-8 butterfly. Since we only need to compute the first subset of output points, we use only a complex adder to replace the radix-8 butterfly unit. As the result, we used only a 128-point FFT/IFFT processor to compute the 128-point IDFT operation and a 128 words memory to buffer the combining output data. Finally, we sent the data out from the 128 words buffer memory.

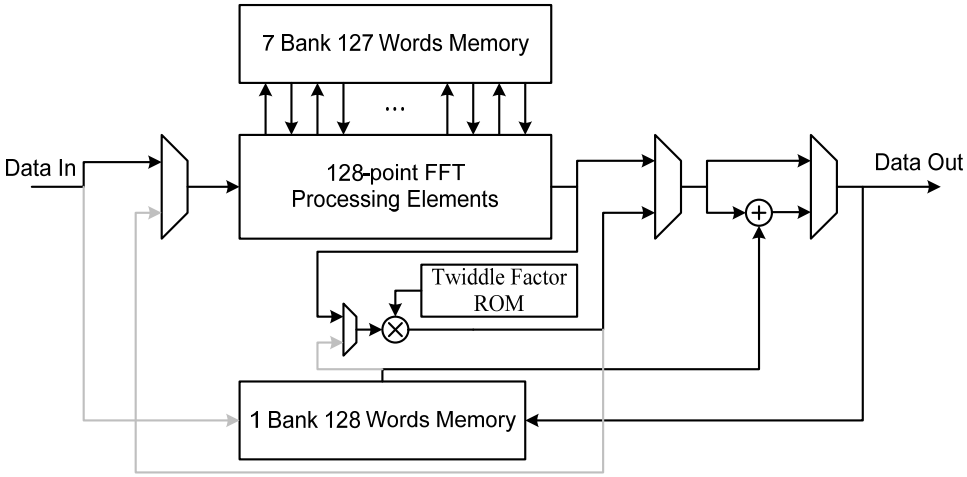


Fig. 3.22 Partial FFT/IFFT processor in IFFT mode

The active block of partial FFT/IFFT processor in FFT mode is shown in Fig. 3.23. Due to the DIT algorithm, the 1024-point FFT operation can be partitioned into 8 128-point DFT operations with a modified input by radix-8 butterfly unit. Since the non-zero input points of DFT operations are only in the first subset of input points, we use only a complex multiplier to replace the radix-8 butterfly unit. Therefore, the partial FFT/IFFT processor in FFT mode will first buffer the input data in 128 words memory, and then read the data from memories by multiplying with suitable twiddle factors to send as the input of 128-point FFT/IFFT processor. Finally, the output data order is a bit-reversal order of input order.

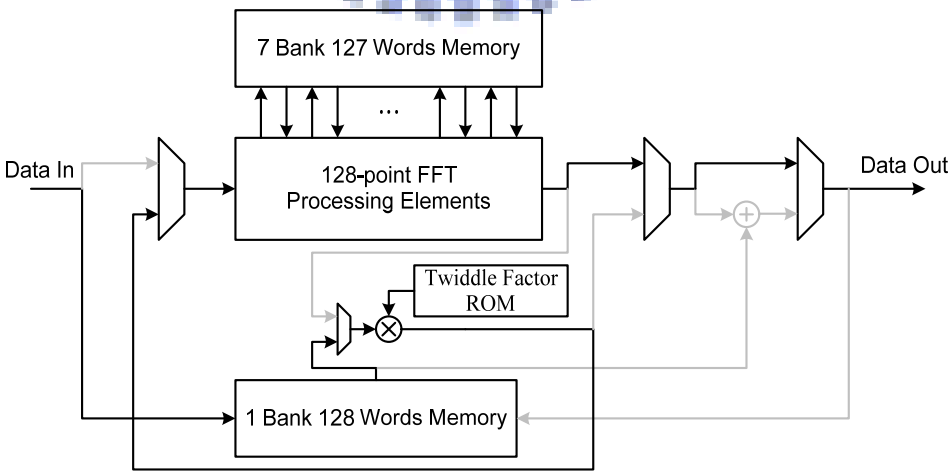


Fig. 3.23 Fig. 3.24 Partial FFT/IFFT processor in FFT mode

Moreover, we can use the concept of DFT with multiple subsets of input and output points in our 128-point FFT processing element with suitable control. It is useful to reduce the redundant operations due to zero input points of guard band or none usable multi-path response. In our proposed DFT-based channel estimation, the path selector will only choose 8 path impulses of the 128 output points for IDFT operation by system simulation. Hence, we can increase the partial FFT control in 128-point FFT processing elements to reduce more redundant operations.

The comparison of hardware complexity is shown in Table 3-4, the proposed partial FFT can reduce 75.1% of the memory size, 22.3% of the complex multipliers, and 30% of the complex adders as compared with traditional radix-2 SDF FFT architecture. Furthermore, with increasing the partial FFT control for the 128-point FFT processor shown in Table 3-5, the proposed partial FFT can reduce maximum 65.3% of multiplication operations and 49.5% of addition operations, which may save more power if the 8 valid output point's indices have common bits.

Table 3-4 Comparison with Partial FFT and Conventional FFT

	Conventional Radix-2 SDF	Partial FFT with Radix-2 SDF
Memory Size (words)	1023(100%)	255(24.9%)
Complex Multiplier	9(100%)	7(77.7%)
Complex Adder	20(100%)	14(70.0%)
Data Latency	1023(100%)	1023(100%)

Table 3-5 Reduced operations of partial FFT with radix-2 SDF architecture

	Original FFT	Modified Architecture Partial FFT	Modified Control Partial FFT	Reduced
Operations of Complex Multiplications	4608 (100%)	3584 (77.7%)	Max 3584 (77.7%) Min 1600 (34.7%)	Max 65.3%
Operations of Complex Additions	10240 (100%)	8064 (78.8%)	Max 8064 (78.8%) Min 5176 (50.5%)	Max 49.5%

3.5 Summary

This chapter introduces the method of designing a parallel-in-parallel-out FFT processor and partial FFT/IFFT processor. In order to tape out the chip of 802.16e baseband receiver, a parallel-in-parallel-out FFT processor is more urgent to make the DFT-based channel estimation to be achievement. Hence, this thesis only focus on the hardware implementation of a parallel-in-parallel-out FFT processor, and the partial FFT processor design can be a future work to improve our system. Next chapter will introduce the design of FFT/IFFT processor with parallel-in-parallel-out in normal order.

Chapter 4

Parallel-In-Parallel-Out FFT/IFFT Processor Architecture Design

4.1 System Requirement of the FFT/IFFT Processor

The decision feedback DFT-based channel estimation (DF DFT-based CE) block diagram is shown in Fig. 4.1, it needs FFT_ch and IFFT_ch blocks with parallel-in-parallel-out to speed up the circuits blocks before or after the FFT_ch and IFFT_ch blocks with parallel computation.

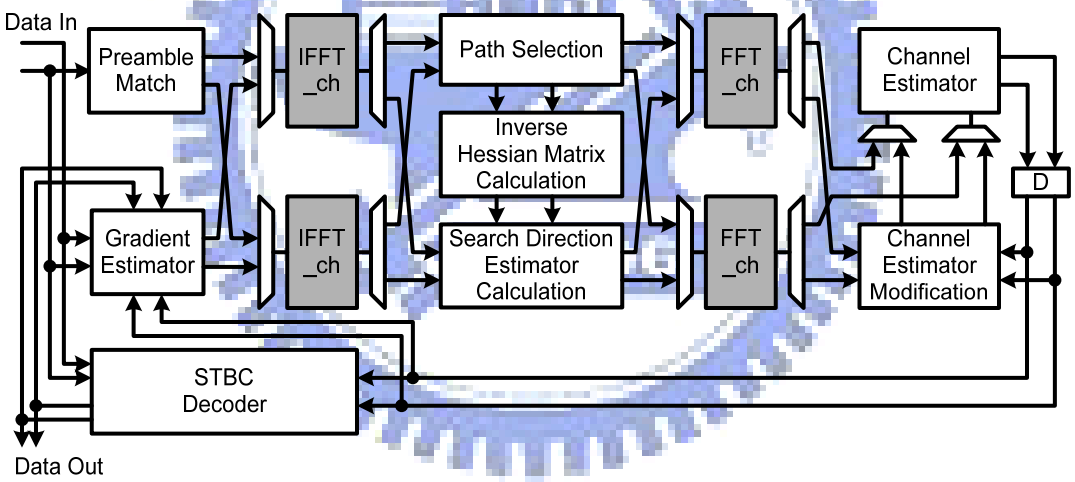


Fig. 4.1 Decision feedback DFT-based channel estimation block diagram

From the analysis of high throughput FFT/IFFT processor architecture with multi-input and multi-output in Chapter 3, memory-based architecture is the best choice for the lowest hardware cost without data latency concerned. In order to speed up the memory-based FFT/IFFT architecture to meet the data latency of system requirement, parallel memory-based architecture is used in our FFT/IFFT processor design. Furthermore, to reduce the hardware cost and control complexity of the

processing elements, we use pipeline-based SDF processing elements to replace the radix-r butterfly units of the memory-based architecture. As a result, the proposed FFT/IFFT processor is based on parallel memory-based FFT architecture with pipeline-based SDF processing elements. The system requirement of the FFT/IFFT processor is shown in Table 4-1.

Table 4-1 FFT/IFFT system requirement

Items	Specification
System Clock Rate	78.4 MHz
FFT Size	1024 points
No. of Inputs or Outputs of FFT processor	8
Data Latency	25 us

The FFT_ch/IFFT_ch blocks have to be designed as the 1024-point FFT/IFFT processor with 8 inputs and 8 outputs working at the system clock rate of 78.4 MHz and the data latency of the FFT/IFFT processor must less than 1/4 OFDM symbol time which is about 25 us.

4.2 Architecture of the FFT/IFFT Processor

According to Chapter 3, we focus on the memory-based FFT processor design with parallel-in-parallel-out in normal order. The conventional memory-based FFT processor with 1 PE and 1 dual-port memory can not achieve the goal of 8 parallel-in-parallel-out data streams. Thus, first, we change the memory from 1 dual-port memory to 8 dual-port memories to achieve the goal of 8 parallel-in-parallel-out data streams. However, the data latency is too long for the memory-based FFT processor with only 1 PE. A FFT/IFFT processor with 4 PE and 8 memory banks is designed to reduce the data latency. In the later discussion, we will show that the

FFT/IFFT processor with 4 PE and 8 memory banks can achieve the best hardware efficiency. The 1024-point FFT/IFFT processor architecture is shown in Fig. 4.2, and it combines parallel radix-8 memory-based architecture and radix-2/4/8 pipeline-based SDF processing elements. The FFT/IFFT Processor has 8 banks of single port memory, 4 radix-2/4/8 SDF processing elements (PEs), 2 radix-2 butterfly units, and 2 commutators between memories and PEs.

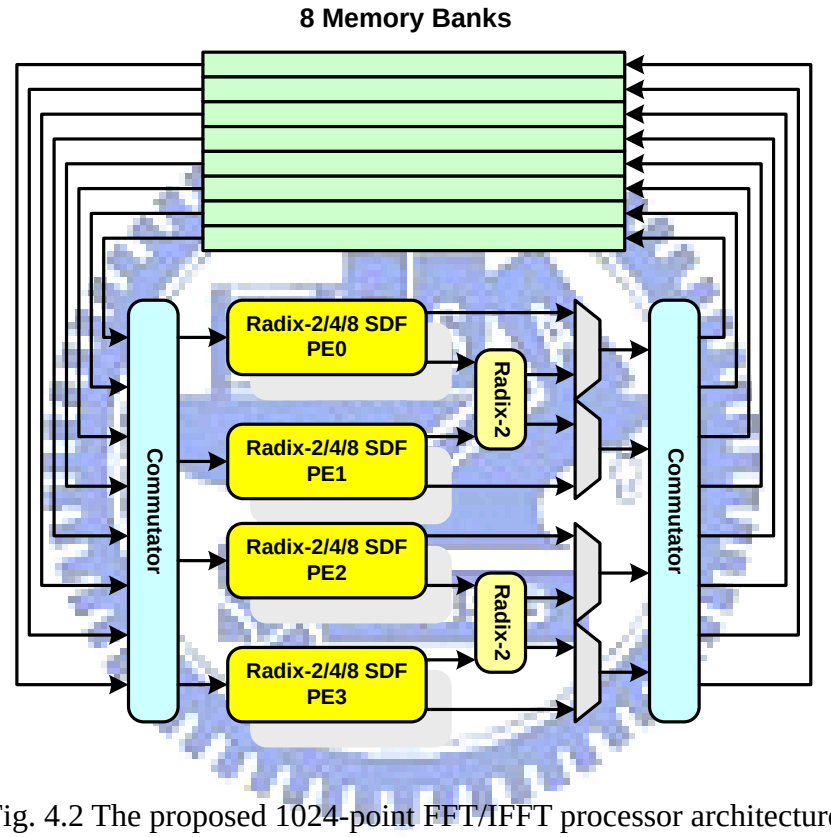


Fig. 4.2 The proposed 1024-point FFT/IFFT processor architecture

According to radix-8 FFT algorithm, 1024-point DFT must use 4 stage of computation with the last stage is a radix-2 computation. If the last stage still uses radix-2/4/8 SDF PE, it wastes 1024 cycles to compute a radix-2 computation because the data of 3rd stage had already written to memory. The last stage can save 25% data latency of the FFT/IFFT computation time by adding a radix-2 butterfly unit at the output of radix-2/4/8 SDF PE, which is shown in Fig. 4.3.

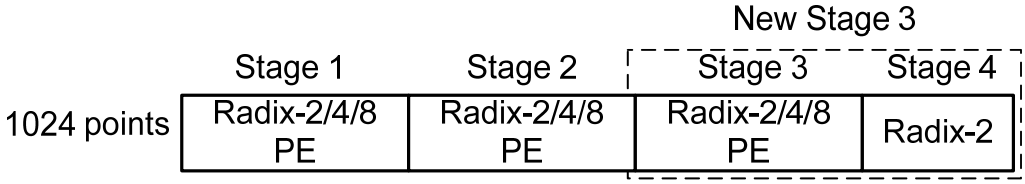


Fig. 4.3 FFT/IFFT processing structure

The proposed architecture may submit 8 parallel-in-parallel-out data streams, and the execution cycles are $(1024/4) \times (\log_8(1024) - 1) = 768$ cycles, which means the data latency is 768/78.4 = 9.79 us (satisfy the system requirement). The detail sub_ module design is described in the next section.

4.3 FFT Sub_Module Design

4.3.1 Radix-2/4/8 SDF Processing Element

The radix-2/4/8 SDF processing element architecture [29] is shown in Fig. 4.4, which contains 3 processing elements of radix-2/4/8 SDF architecture, a reorder buffer, a twiddle factor ROM, a complex multiplier, and a fixed-point block.

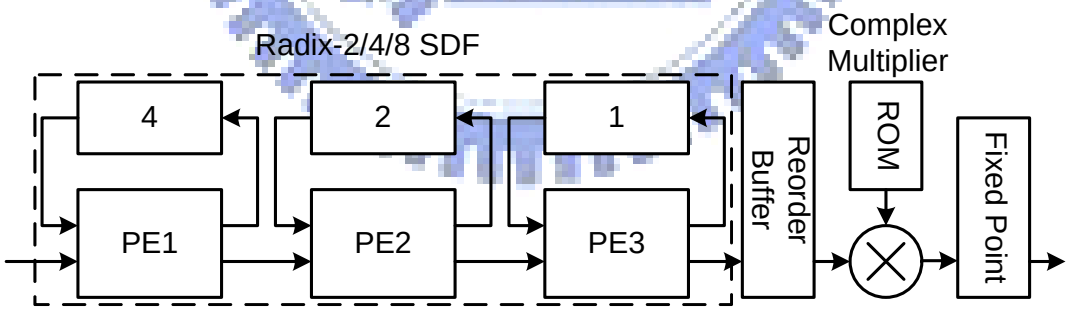


Fig. 4.4 Radix-2/4/8 SDF processing element

We use radix-2/4/8 SDF architecture with decimation-in-time (DIT) algorithm to implement because the radix-8 DIT algorithm has less quantization error than DIF algorithm [12]. This is because the quantization error is caused by constant multiplier. In DIF algorithm, the constant multiplier is in the PE1 while the constant multiplier is in the PE3 in DIT algorithm as shown in Fig. 4.6(c). The path of the quantization error

passing in DIF algorithm is longer than that in DIT algorithm; hence, we choose DIT algorithm to implement the radix-2/4/8 SDF PE. Fig. 4.5 shows the radix-2/4/8 SDF architecture with DIT algorithm. The detail of the processing elements is shown in Fig. 4.6.

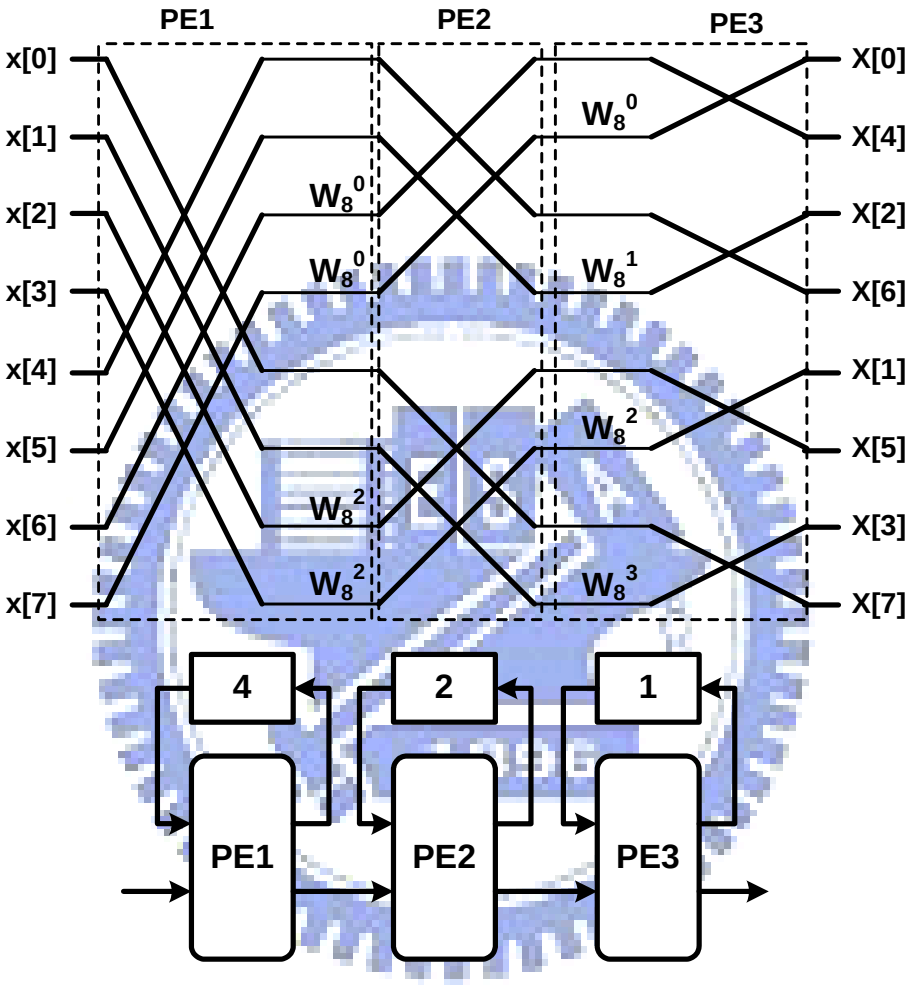
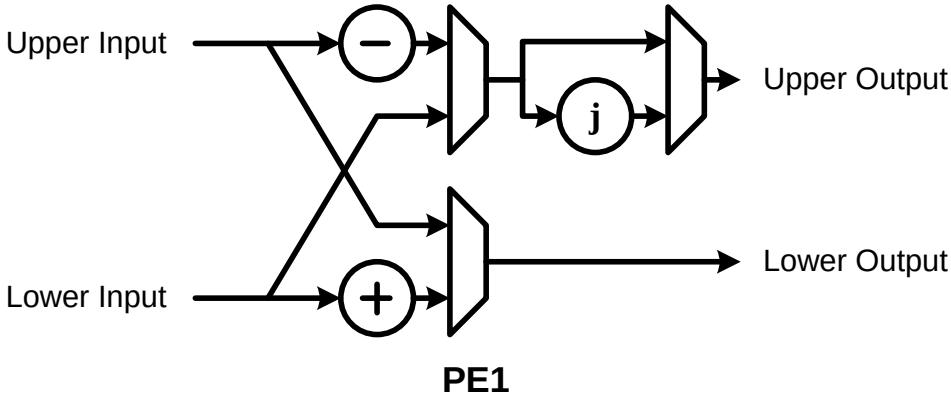
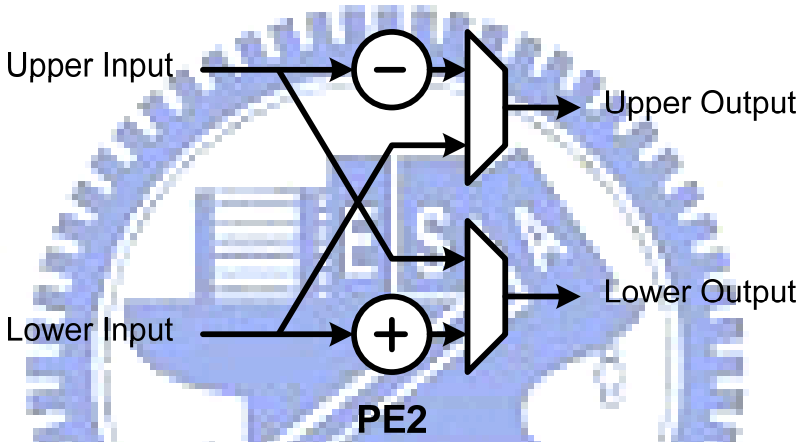


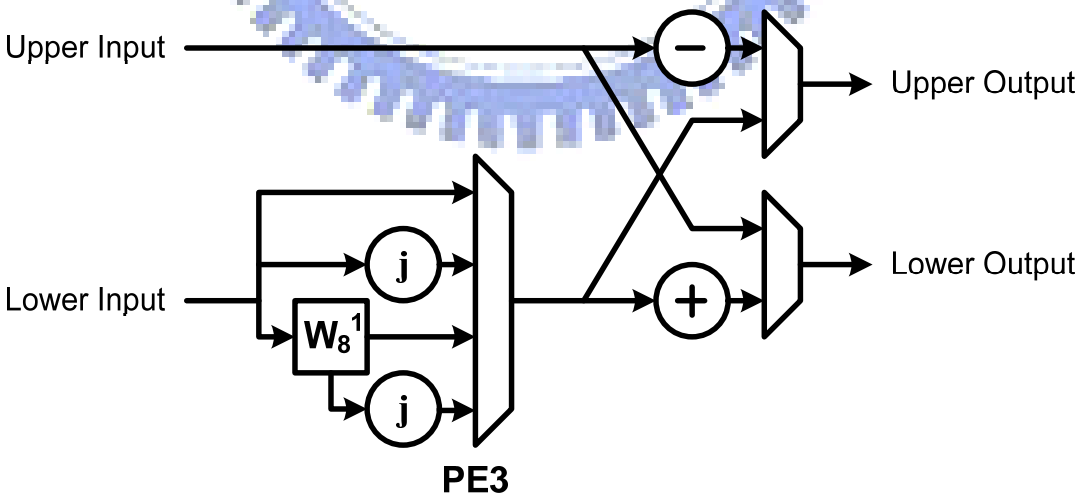
Fig. 4.5 Radix-2/4/8 SDF with DIT algorithm



(a)



(b)



(c)

Fig. 4.6 Processing elements of radix-2/4/8 SDF with DIT algorithm

The reorder buffer contains 3 registers, and the timing flow of the reorder buffer is shown in Fig. 4.7. It requires 3 unit time delays to make the output data in normal order. The red line is the writing cycle for input data to be written to reorder buffer, and the green line is the reading cycle for output data to be read from reorder buffer or input. In addition, there are 3 modes for the read or write order which will repeat every 24 cycles.

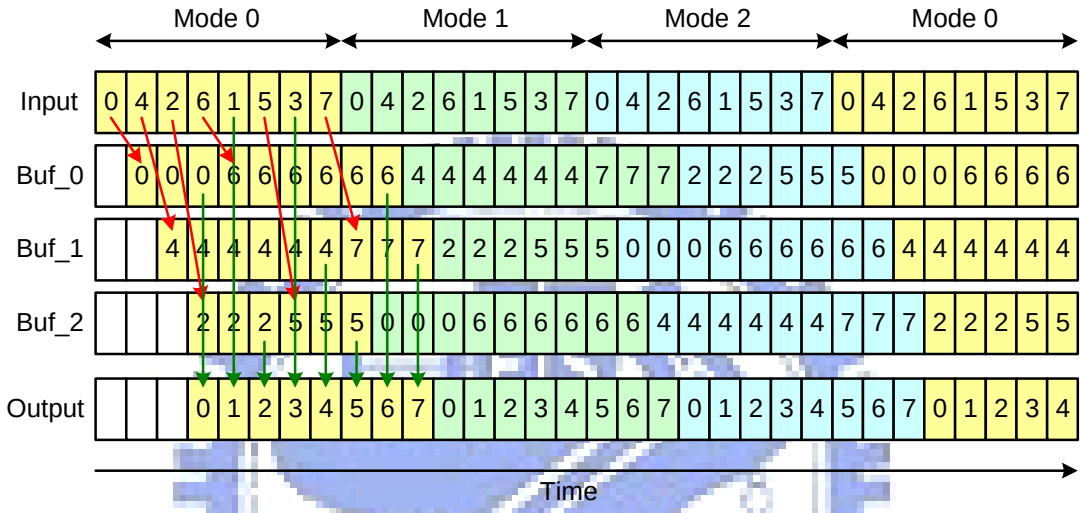


Fig. 4.7 Reorder buffer input and output timing flow graph

4.3.2 Complex Multiplier

There are three types of complex multiplication in our design, the first one is multiplication of – j, the second one is multiplication of a constant twiddle factor W_8^1, W_8^3, and the least one is multiplication of a complex twiddle factor.

(1). Multiplication of – j:

The multiplication by –j can be realized by interchanging the real part and imaginary part of input, then negate the imaginary part of output, which is shown in Fig. 4.8. The only hardware is multiply by -1 which can be realized with an inverter and an adder.

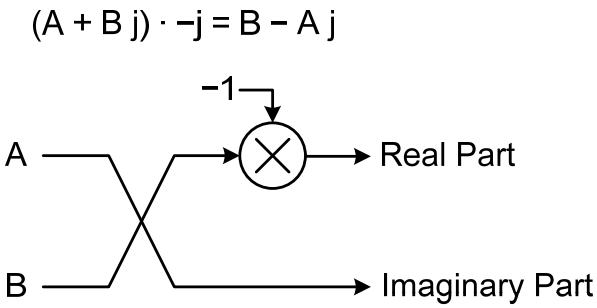


Fig. 4.8 Architecture of multiplication of -j

(2). Multiplication of constant twiddle factor W_8^1, W_8^3:

Multiplication of constant twiddle factor W_8^1, W_8^3 can be defined as the following equation.

$$(A+Bj)\times W_8^1 = (A+Bj)\times\left(1/\sqrt{2}\right)\times(1-j) = \left(1/\sqrt{2}\right)\times\left((A+B)+(B-A)j\right) \quad (4.1)$$

$$(A+Bj)\times W_8^3 = (A+Bj)\times W_8^1\times(-j) \quad (4.2)$$

From Eq. (4.1), the constant complex multiplier can be implemented by one real adder, one real subtraction, and two real constant multipliers as shown in Fig. 4.9. Also, from Eq. (4.2), the constant complex multiplication of W_8^3 can use the same hardware of multiplication by W_8^1 and multiplication by –j. In addition, the multiplication by real constant value can be realized as shifter and adders [20], where the constant value $(1/\sqrt{2})$'s binary representation is 0.10110101, 9 bits fixed-point of $1/\sqrt{2}$; thus, the multiplication by $(1/\sqrt{2})$ can be realized with shifter and carry-save adder (CSA) tree, also called Wallace tree, shown in Fig. 4.10.

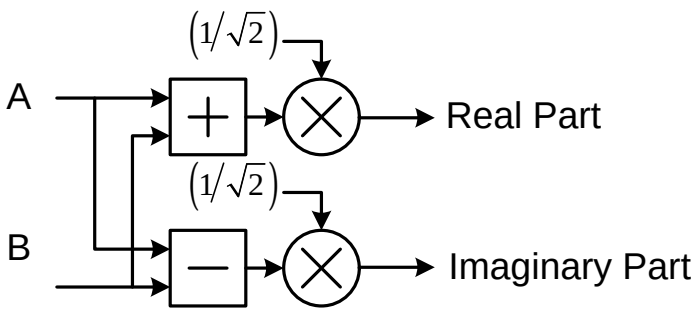


Fig. 4.9 Architecture of multiplication of W_8^1

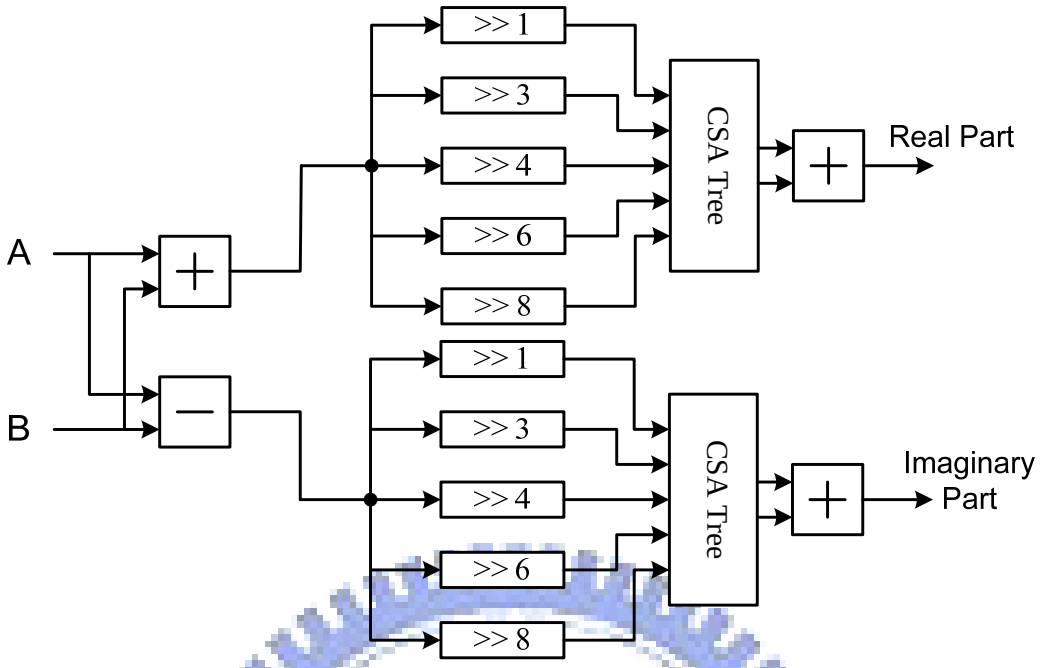


Fig. 4.10 Architecture of multiplication by W_8^1 with CSA tree

From Fig. 4.10, the architecture replace the constant multiplier with shifters, CSA tree, and a carry-propagation adder which reduce the hardware cost. However, this architecture needs to pass 2 carry-propagation adders which are not efficient for delay optimization. For the reason of delay optimization, we modify Eq. (4.1) as follow:

$$\begin{aligned}&\left(1/\sqrt{2}\right)\times\left((A+B)+(B-A)\,j\right)\\&=\left(\left(\left(1/\sqrt{2}\right)A+\left(1/\sqrt{2}\right)B\right)+\left(\left(1/\sqrt{2}\right)B-\left(1/\sqrt{2}\right)A\right)j\right)\end{aligned}\tag{4.3}$$

The architecture can be realized without adder or subtraction before the CSA tree as shown in Fig. 4.11, and the total delay is reduced to one inverter, one CSA tree, and one carry-propagation adder. In addition, the compensation bits for negative number of A are combined and realized as a input of CSA Tree.

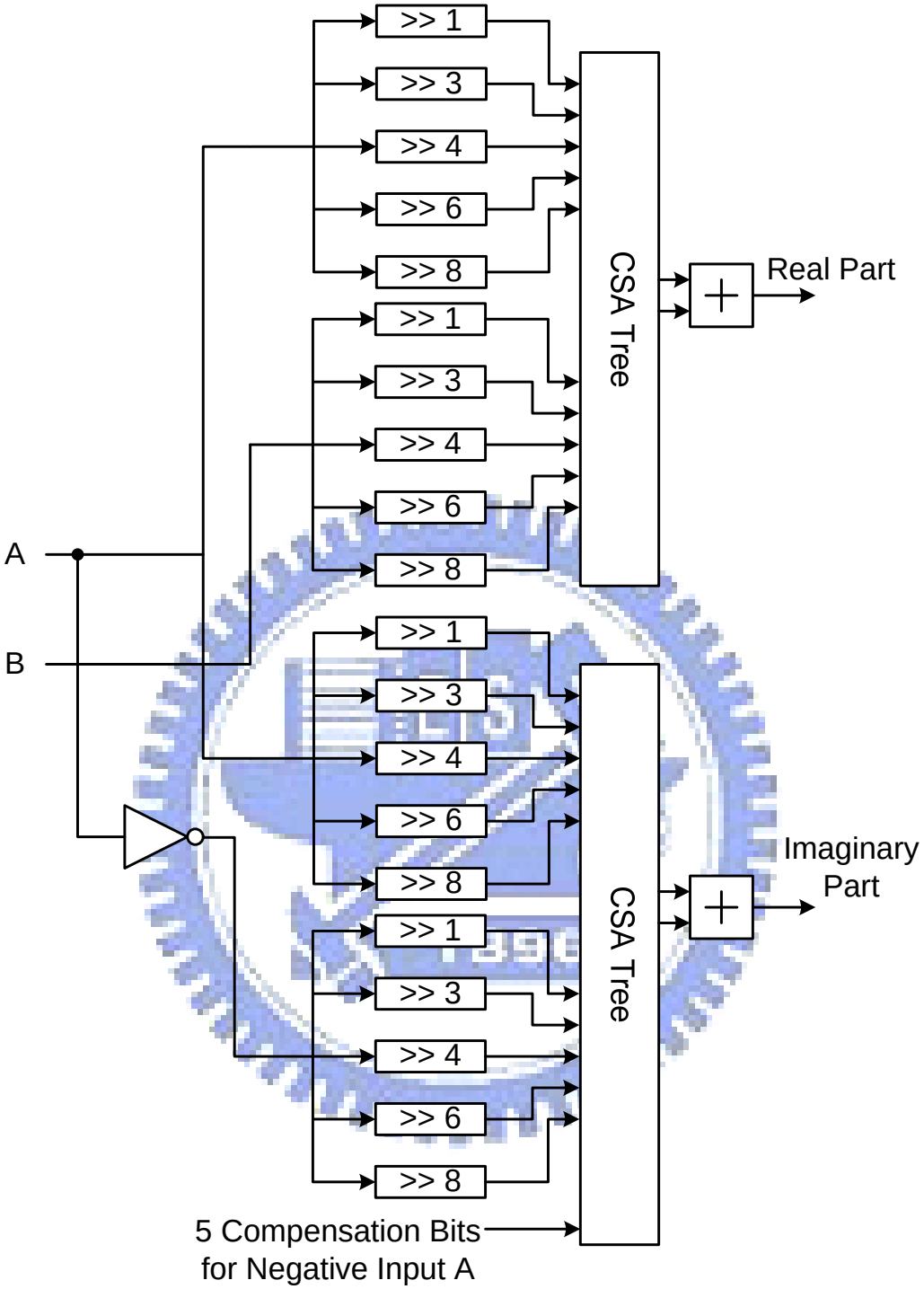


Fig. 4.11 Delay optimized architecture of multiplication by W_8^1 with CSA tree

(3). Multiplication of complex twiddle factor:

A complex multiplier needs four real multipliers, one real addition, and one real subtraction, which can express as:

$$(A+Bj)\times(C+Dj)=(AC-BD)+(AD+BC)j \qquad (4.4)$$

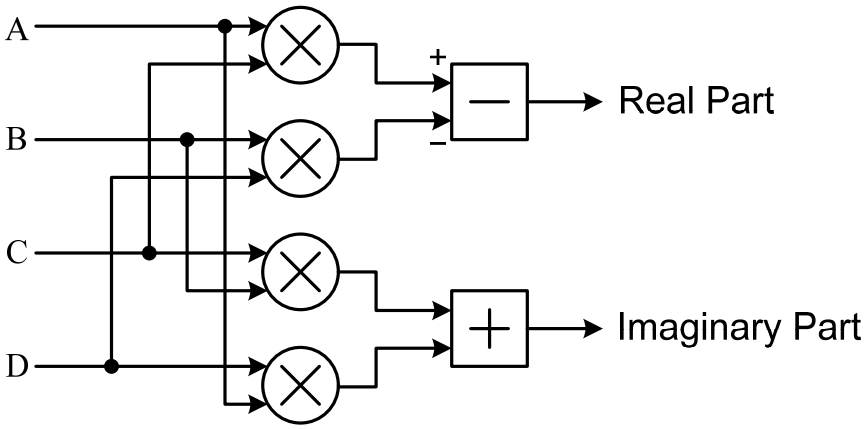


Fig. 4.12 Architecture of complex multiplication

The complex multiplier is about 4 times area of real multiplier shown as Fig. 4.12, which consumes much hardware. Fortunately, the complex multiplier can be realized by three real multipliers, three real additions, and two real subtractions [30], which can reduce large area for hardware implementation of complex multiplier. The equation can be express as:

$$(A+Bj)\times(C+Dj)=(A(C+D)-D(A+B))+(A(C+D)+C(B-A))\,j \qquad (4.5)$$

The hardware architecture of modified complex multiplication is shown in Fig. 4.13.

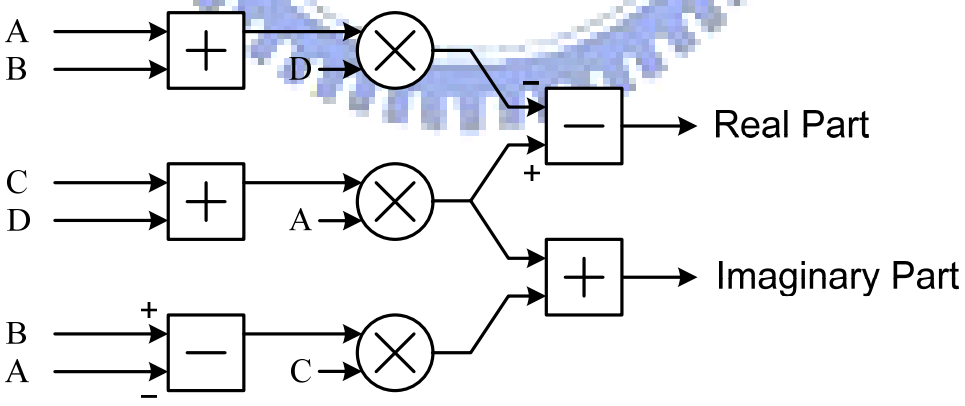


Fig. 4.13 Modified architecture of complex multiplication

4.3.3 ROM Table

The conventional FFT processors usually stored the required coefficient called twiddle factor in a look-up table which generally implemented by ROM. Symmetry property of twiddle factor is used to reduce the hardware cost of twiddle factor ROM. For instance, the twiddle ROM only stored 0 ~ π/4 of twiddle factor value, and the other value is based on the symmetry property of these values. However, they also need a generator to generate the required value. According to our FFT processor architecture, four processing elements need different twiddle factors in different stages as shown in Table 4-2.

Table 4-2 Twiddle factors value for different PE in different stages

	PE0	PE1	PE2	PE3
Stage 1	$W_{1024}^{k\cdot(4n1+0)}$	$W_{1024}^{k\cdot(4n1+1)}$	$W_{1024}^{k\cdot(4n1+2)}$	$W_{1024}^{k\cdot(4n1+3)}$
Stage 2	$W_{1024}^{8\cdot k\cdot(4n2+0)}$	$W_{1024}^{8\cdot k\cdot(4n2+1)}$	$W_{1024}^{8\cdot k\cdot(4n2+2)}$	$W_{1024}^{8\cdot k\cdot(4n2+3)}$
Stage 3	$W_{1024}^{64\cdot k\cdot(0)}$	$W_{1024}^{64\cdot k\cdot(1)}$	$W_{1024}^{64\cdot k\cdot(0)}$	$W_{1024}^{64\cdot k\cdot(1)}$
k=0…7, n1=0…31, n2=0…3				

There are 3 stages with different twiddle factors for each processing element. In the first stage, each processing element has 256 different twiddle factors to be stored in each twiddle factor ROM. In the second stage, 32 different values are stored in each ROM. In the final stage, only 8 different values are stored in each ROM. As a result, each twiddle factor ROM has 296 different values to be stored in each twiddle factor ROM, called PE-based twiddle factor ROM (PE-based TW ROM).

An 8 bits counter is used to generate the address of PE-based TW ROM, and the binary representation is $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$. The address of PE-based TW ROM in each stage is shown in Table 4-3.

Table 4-3 Address of PE-based TW ROM in each stage

	Twiddle factor ROM Address
Stage 1	$0\ b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$
Stage 2	$1\ 0\ 0\ 0\ b_4\ b_3\ b_2\ b_1\ b_0$
Stage 3	$1\ 0\ 0\ 1\ 0\ 0\ b_2\ b_1\ b_0$

The PE-based TW ROM for each processing element is 15% larger than the conventional ROM table that only stores 0 ~ π/4 of twiddle factor value. However, no generator for PE-based TW ROM is needed for the symmetry twiddle factors. Moreover, the PE-based TW ROM is implemented with combination circuits, and the area is different for each PE's ROM due to different number of the same twiddle factor values in its twiddle factor ROM. For example, PE-based TW ROM for PE1 has many stored twiddle factor equals to one for k, n1, n2, or n3 equals to zero. The area difference will be shown in next chapter.

4.3.4 Memory Allocation

The system requirement of 1024-point FFT/IFFT processor for multi-input and multi-output in normal order is shown in Fig. 4.14. The memory must write 8 input data in normal order in one cycle before FFT/IFFT computation, and read 8 output data in normal order after FFT/IFFT computation, too.

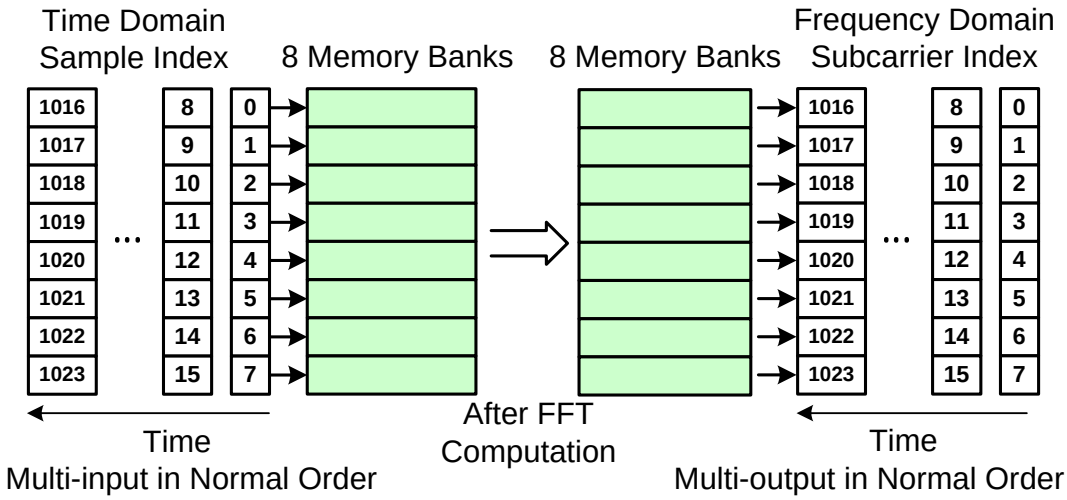


Fig. 4.14 System requirement for multi-input and multi-output in normal order

The memory allocation is an important issue because the memory allocation of the input data will affect the memory allocation of the output data by 3-digit reversal for radix-8. For instance, assume the input data index $x(n)$'s binary index is $x(n_9n_8n_7n_6n_5n_4n_3n_2n_1n_0)$. If we partitioned the input data by $\{n_2n_1n_0\}$ into the 8 memory banks, the input data order is able to write 8 input data in normal order to different memory banks, but, the output data location is unable to read 8 output data in normal order. Assume the binary index of output data $X(k)$ is $X(k_9k_8k_7k_6k_5k_4k_3k_2k_1k_0)$, the memory location is at input data location $x(k_2k_1k_0k_5k_4k_3k_8k_7k_6k_9)$, and $\{k_2k_1k_0\}$ is in the same memory bank. Since $\{k_2k_1k_0\}$ is related to $\{n_9n_8n_7\}$, an easy way to partition the input data to 8 memory banks is that each bank select the input data by $\{n_2n_1n_0\}+\{n_9n_8n_7\}$, and the data location in 8 partitions memory is shown in Fig. 4.15. By this memory allocation of input data, $\{k_2k_1k_0\}$ is in the different memory banks. Thus, the output data location is able to read 8 output data in normal order.

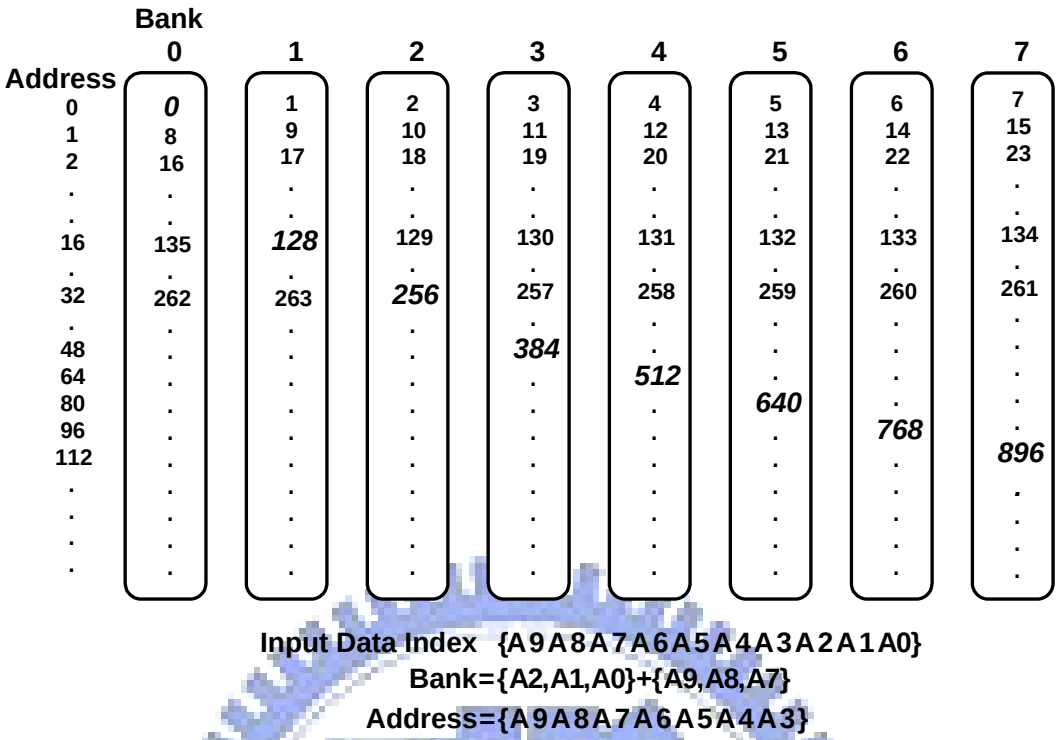


Fig. 4.15 Memory allocation of the FFT/IFFT input data

Without reorder buffer, the proposed FFT/IFFT processor can save 46.8 % memory area than the conventional radix-8 memory-based FFT architecture with reorder buffer.

4.3.5 Commutator Design

The commutator has an important issue to make the read write operations of different memory banks to be conflict free. Besides, in memory design, a single port memory's area is about half of a dual port memory's area. For example, a 128 words × 38 bits dual port memory size is 0.054 mm^2, but a 128 words × 38 bits single port memory size is 0.023 mm^2 (The memory is generated by memory compiler using 90nm process technology). The single port memory size is 42.6% of the dual port memory size. With the commutator, we can change the 4 dual port memories into the 8 single port memories. The read or write address for the 4 PE in each stage is shown in Table 4-4. A counter is used to read or write the data from the memory for 4 radix-8

PEs, which's binary index is $b_7b_6b_5b_4b_3b_2b_1b_0$, and, p_1p_0 is the ID of PE, {00,01,10,11} means {PE0,PE1,PE2,PE3}. The following will show how to design the single port memory with read write operations conflict free [21][22].

Table 4-4 Read or write address for the processing elements in each stage

	Read or Write Address	Address in Memory Bank
Stage1	$b_2b_1b_0b_7b_6b_5b_4b_3p_1p_0$	$\{b_2b_1b_0\}+\{b_3p_1p_0\}$
Stage2	$b_7b_6b_5b_2b_1b_0b_4b_3p_1p_0$	$\{b_7b_6b_5\}+\{b_3p_1p_0\}$
Stage3	$b_7b_6b_5b_4b_3p_1b_2b_1b_0p_0$	$\{b_7b_6b_5\}+\{b_1b_0p_0\}$

The read write operations for stage 1 are shown in Fig. 4.16. According to Fig. 4.16, we need at least 8 pipeline stages for each PE in stage 1; however, 8 pipeline stages for each PE can't meet the timing constrain of the processing elements. For the system timing constrain, we choose 24 pipeline stages in stage 1 to make memory read write operations conflict free, and also make the timing constrain of the processing elements meet the system requirement.

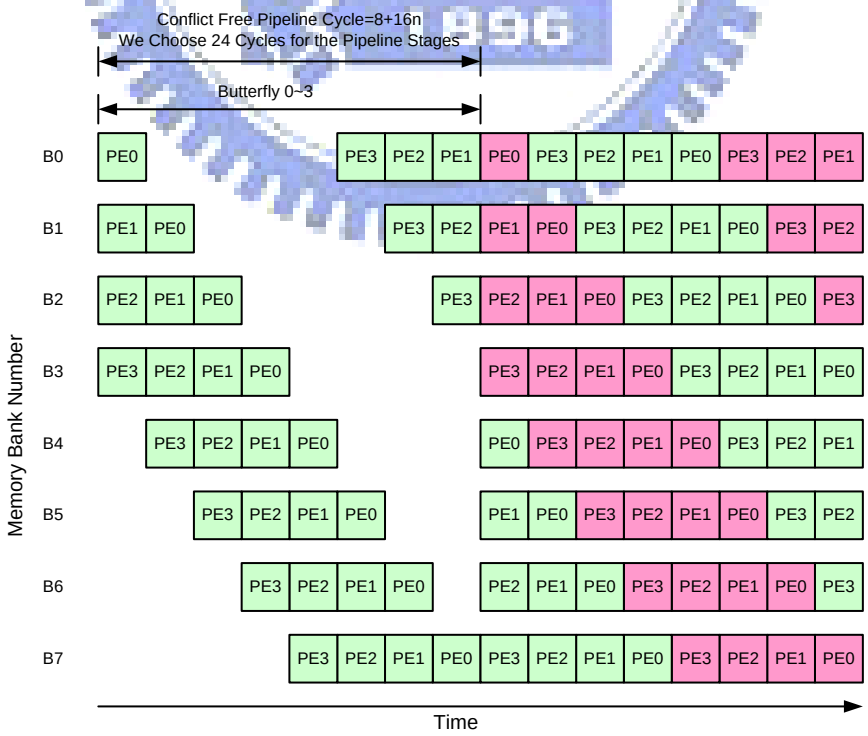


Fig. 4.16 Memories read write operations for different PE in stage 1

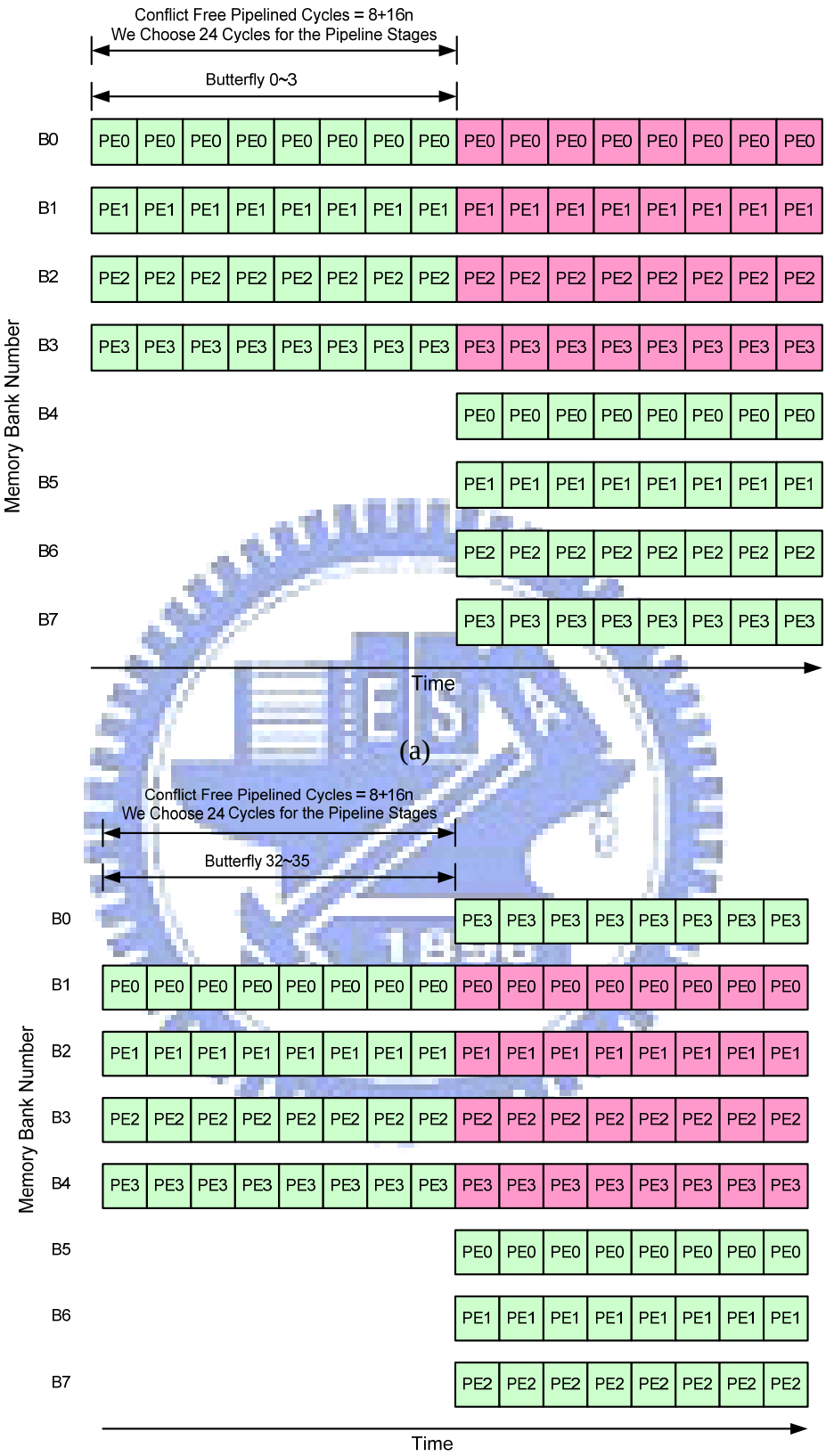


Fig. 4.17 Memories read write operations for different PE in stage 2

(a) butterfly 0~7 (b) butterfly 32~39

The read write operations for stage 2 are shown in Fig. 4.17. The read write operations in stage 2 which is not similar to stage 1, change the operations order every 32 butterflies. In addition, the 32 butterflies with the same read write operations are called inner stage, which is differ to stage defined by FFT algorithm called outer stage. For this reason, we have to stall cycles every 32 butterflies to wait the data already written to the memories. Then, after that, we start to read data for next 32 butterflies. Here we also choose 24 pipeline stages in stage 2 for the system timing constrain.

In stage 3, the read and write operations for PE0 and PE1 is the same as that for PE2 and PE3. Thus, we have to delay one cycle for PE2 and PE3 reading or writing the data. The operations are shown in Fig.4.18. In addition, the read write operations in stage 3, similar to stage 2, have to stall cycles every 32 butterflies, too. Here we choose 22 pipeline stages in stage 3 for the system timing constrain.

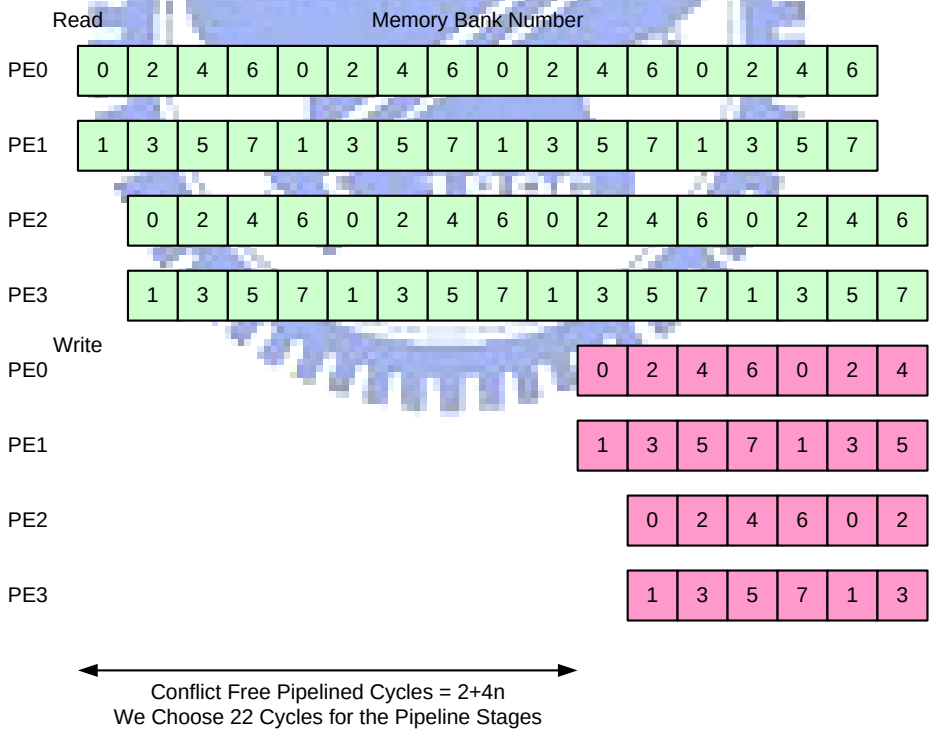


Fig. 4.18 Memories read write operations for different PE in stage 3

According to the analysis of the commutator operations, the state diagram of the proposed FFT/IFFT processor is shown in Fig. 4.19. The state diagram begins with

the IDLE state waiting for the fft_start signal to start the FFT/IFFT computation. Each stage has 5 states: Rd, Tw, Wr, Wait_Tw, and Wait_Wr. The Rd state is for PE to read the input data from memories and also triggers the counter of memory reading address. The Tw state is for PE's data multiplying by twiddle factor and also triggers the counter of twiddle factor ROM reading address. The Wr state is for PE to write the output data to memories and also triggers the counter of memory writing address. The last 2 states, Wait_Tw/Wait_Wr, are waiting for the reading data in PEs already multiply with twiddle factor/writing to the memory.

The state is beginning with stage 1 Rd. Then, after suitable pipeline stages, the current state is changed to the next states, which are stage 1 Rd, stage 1 Tw, stage 1 Wr, stage 1 Wait_Tw, and stage 1 Wait_Wr. After the stage 1 Wait_Wr state has already done, the current state is changed to 5 states of next stage.

From analysis of read write operations in each stage discussed in Fig. 4.16, Fig. 4.17, and Fig. 4.18, stage 2 and stage 3 has more stall cycles than stage 1 due to the operations order changed every 32 butterflies. Therefore, there are two signals to change the current state of the last 3 states in stage 2 and stage 3. One is outer stage signal triggered every 255 butterflies called outer_stage_inc. The other is inner stage signal triggered every 32 butterflies called inner_stage_inc. The outer stage and inner stage is defined as the discussion mentioned before. Thus, there is a loop in stage 2 and stage 3 due to the operations of current state in inner stage or outer stage.

Finally, the state diagram for commutator will make the read write operations of different memory banks to be conflict free by stall the cycles between inner stages or outer stages.

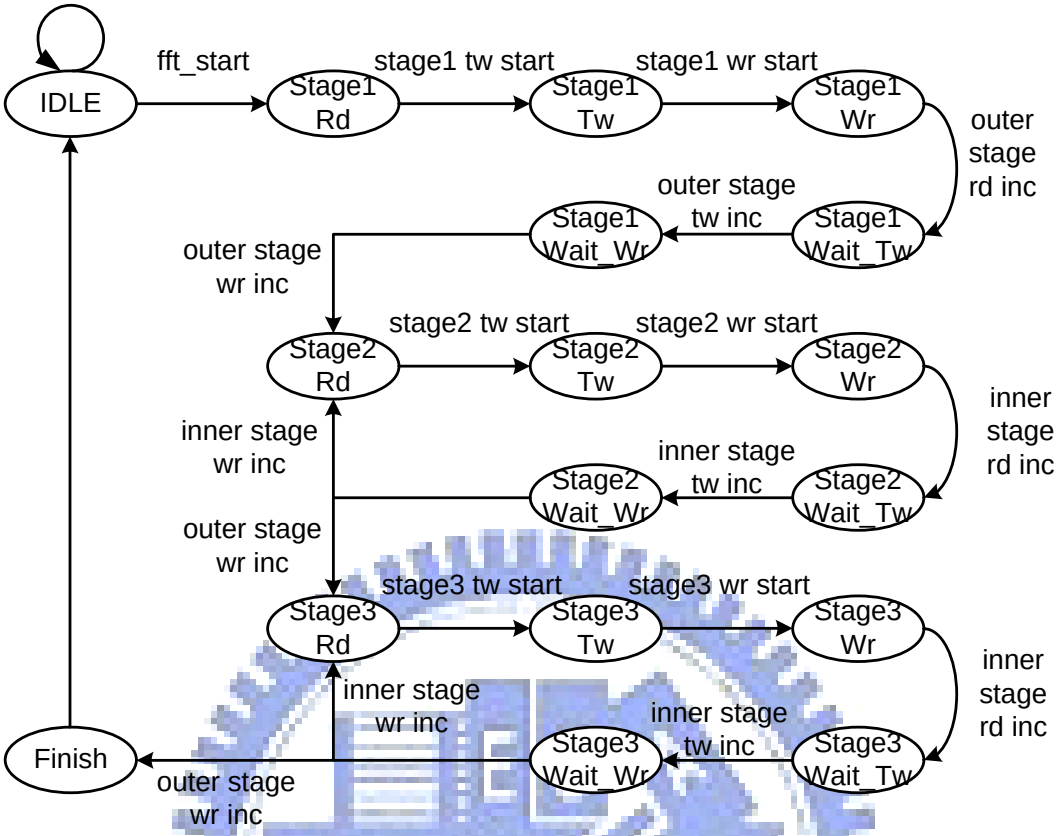


Fig. 4.19 State diagram of FFT/IFFT processor

4.3.6 Mixed FFT/IFFT Processor

For hardware efficiency, we want to use the same FFT/IFFT processor to compute the FFT and IFFT block, which is shown in Fig. 4.20.

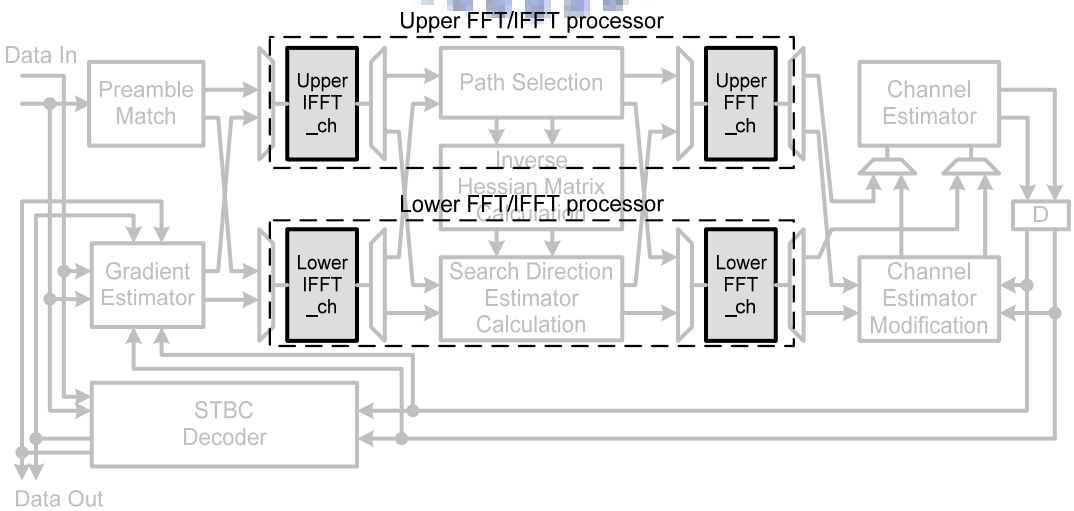


Fig. 4.20 The FFT/IFFT processor in the DF DFT-based CE block diagram

Assume the fast Fourier transform (FFT) equation is defined as

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{k \cdot n} \tag{4.6}$$

The inverse fast Fourier transform (IFFT) equation is defined as

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

If we take the conjugate of right side in Eq. (4.7), we find

$$x(n) = \frac{1}{N} \left[\sum_{k=0}^{N-1} \left(X(k) \right)^* \cdot W_N^{k \cdot n} \right]^* \tag{4.8}$$

Therefore, the IFFT function can be performed with FFT and conjugate operation [11]. The modified processing elements for the proposed FFT/IFFT processor are shown in Fig. 4.21. With the conjugate operation, we can use only two FFT/IFFT processors in our system; one computes the upper FFT and IFFT, the other computes the lower FFT and IFFT.

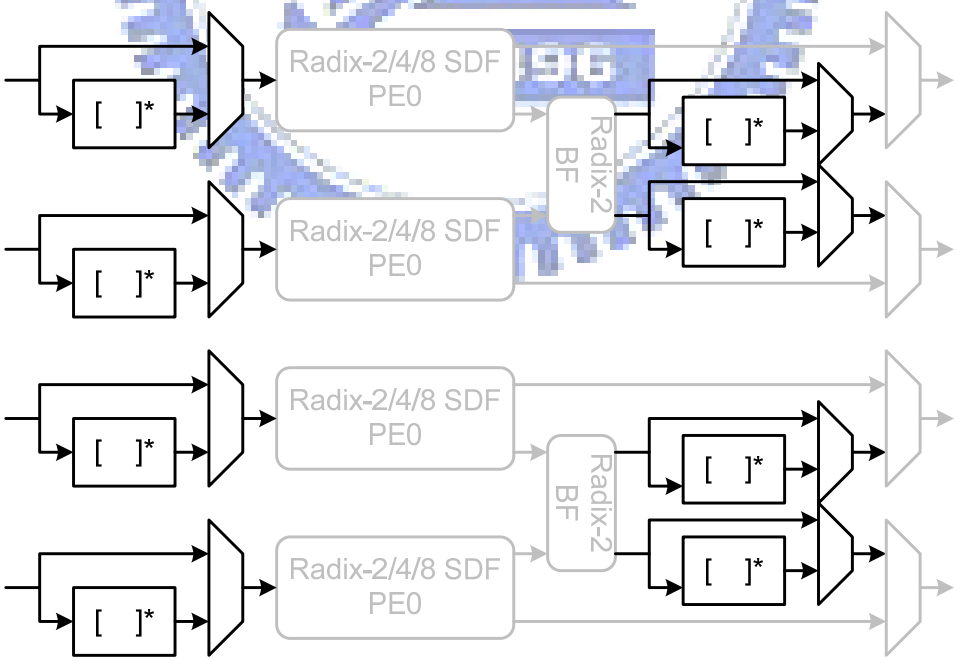


Fig. 4.21 Modified processing elements with conjugate operation

4.3.7 Fixed-Point Block Design with Dynamic Scaling

The fixed-point block is a truncating block which truncates the output data of the processing elements before writing to the memories. Due to the system requirement of FFT and IFFT are different, we have to optimize the fixed-point block for different condition of FFT or IFFT computation [24].

For IFFT computation, there are 1024 input data. According to 1024-point radix-8 DIF algorithm, the stage 1 is 8-point DFT multiply with the twiddle factor, which defined as

$$\begin{aligned} & X_{stage1}(n_1+2n_2+16n_3+128k_4) \\ & = BU_{8\ stage1}(n_1,n_2,n_3,k_4) \\ & = \left[\sum_{n_4=0}^{7} x(n_1+2n_2+16n_3+128n_4)\cdot W_8^{k_4\cdot n_4}\right]\times W_{1024}^{k_4\cdot(n_1+2n_2+16n_3)} \end{aligned} \tag{4.9}$$

Take the absolute value of both sides in Eq. (4.9), we find

$$\left|X_{stage1}\right| \le \sum_{n_4=0}^{7}\left|x(n_1+2n_2+16n_3+128n_4)\right| \le 8\times\left|x\right|_{\max} \tag{4.10}$$

Since the range of IFFT input data *x* for both real part and imaginary part is -2 ~ +2, which is defined by the system simulation, we find $|X_{stage1}| \le 22.6274$. If the X_{stage1} is defined as A + j·B, $\left|X_{stage1}\right| = \sqrt{A^2+B^2} \le 22.6274$. Thus, both |A| and |B| are smaller than 22.6274. As the result, we take 6 bits of integer for both real part and imaginary part of the X_{stage1}, and the fraction bits of X_{stage1} is decided by overall FFT/IFFT fixed-point simulation.

The stage 2, stage 3 and stage 4 are calculated as

$$\begin{aligned} & X_{stage2}(n_1+2n_2+16k_3+128k_4) \\ & = BU_{8\ stage2}(n_1,n_2,k_4,k_3) \\ & = \left[\sum_{n_3=0}^{7} X_{stage1}(n_1+2n_2+16n_3+128k_4)\cdot W_8^{k_3\cdot n_3}\right]\times W_{1024}^{8k_3\cdot(n_1+2n_2)} \end{aligned} \tag{4.11}$$

$$X_{stage3}(n_1 + 2k_2 + 16k_3 + 128k_4)$$
$$= BU_{8\ stage3}(n_1, k_3, k_4, k_2) \tag{4.12}$$
$$= \left[\sum_{n_2=0}^{7} X_{stage2} \cdot W_8^{k_2 \cdot n_2} \right] \times W_{1024}^{64k_2 \cdot n_1}$$

$$X_{stage4}(512k_1 + 64k_2 + 8k_3 + k_4)$$
$$= BU_{2\ stage4}(k_2, k_3, k_4, k_1) \tag{4.13}$$
$$= \sum_{n_1=0}^{1} X_{stage3} \cdot W_2^{k_1 \cdot n_1}$$

Take the absolute value of both sides in Eq. (4.11), Eq. (4.12), and Eq. (4.13), we find

$$\left|X_{stage2}\right| \le \sum_{n_3=0}^{7} \left|X_{stage1}\right| \le 64 \times |x|_{\max} \tag{4.14}$$

$$\left|X_{stage3}\right| \le \sum_{n_2=0}^{7} \left|X_{stage2}\right| \le 512 \times |x|_{\max} \tag{4.15}$$

$$\left|X_{stage4}\right| \le \sum_{n_1=0}^{1} \left|X_{stage3}\right| \le 1024 \times |x|_{\max} \tag{4.16}$$

Since the range of IFFT input data x for both real part and imaginary part is -2 ~ +2, we find $|X_{stage2}| \le 181.0193$, $|X_{stage3}| \le 1448.2$, and $|X_{stage4}| \le 2896.3$; thus, we take 9, 12, 13 bits of integer for both real part and imaginary part of the X_{stage2}, X_{stage3}, X_{stage4}.

For FFT computation, there are 8 random nonzero input data of the first 128 input data, defined by system simulation for DF DFT-based CE, and the other input data are zeros. The range of the FFT input data is the same as the range of the IFFT input data, which is -2 ~ +2 for both real part and imaginary part.

According to Eq. (4.10), $|X_{stage1}| \le 1 \times |x|_{\max} \le 2.8284$, since x is a nonzero point only if n_4 is equal to zero. As the result, we take 3 bits of integer in stage 1. In stage 2, the maximum of $|X_{stage2}|$ occurred when input data are nonzero points with n3 equaling to 0 to 7; therefore, $|X_{stage2}| \le 8 \times |x|_{\max} \le 22.6274$ and we take 6 bits of integer in stage 2. In stage 3, the maximum of $|X_{stage3}|$ occurred when input data are nonzero points with n2 equaling to 0 to 7; therefore, $|X_{stage3}| \le 8 \times |x|_{\max} \le 22.6274$ and we take 6 bits of integer in stage 3. The X_{stage4} is the final stage output data, $|X_{stage4}| \le 8 \times |x|_{\max} \le$

22.6274 because only 8 input data are nonzero data, and we take 6 bits of integer in stage 4. In addition, the fraction bits of X_{stage2}, X_{stage3} and X_{stage4}, are also defined by overall FFT/IFFT fixed-point simulation.

The fixed-point block parameter for FFT or IFFT mode is shown in Table 4-5. Parameter WL in Table 4-5 is the internal word length of FFT/IFFT processor.

Table 4-5 Scale down block parameter for FFT/IFFT mode

	IFFT Mode		FFT Mode	
	Integer bits (bits)	Fraction bits (bits)	Integer bits (bits)	Fraction bits (bits)
Stage 1	6	WL－6	3	WL－3
Stage 2	9	WL－9	6	WL－6
Stage 3	12	WL－12	6	WL－6
Stage 4	13	WL－13	6	WL－6

4.4 The FFT/IFFT Processor Fixed Point Simulation

The proposed FFT/IFFT processor has been modeled in Matlab and C language. The FFT/IFFT processor performance is evaluated by SQNR (Signal-to-Quantization Noise Ratio) for the system requirement. The simulation model is shown in Fig. 4.22. First, we simulate the least truncate bits for input and output data of the FFT_ch/IFFT_ch block which can meet the system required BER as shown in upper of Fig. 4.22. Then, we use random pattern to obtain the system required SQNR for the FFT/IFFT processor for general case of input pattern as shown in lower of Fig. 4.22, where the truncate bits of input or output are decided by upper of Fig. 4.22.

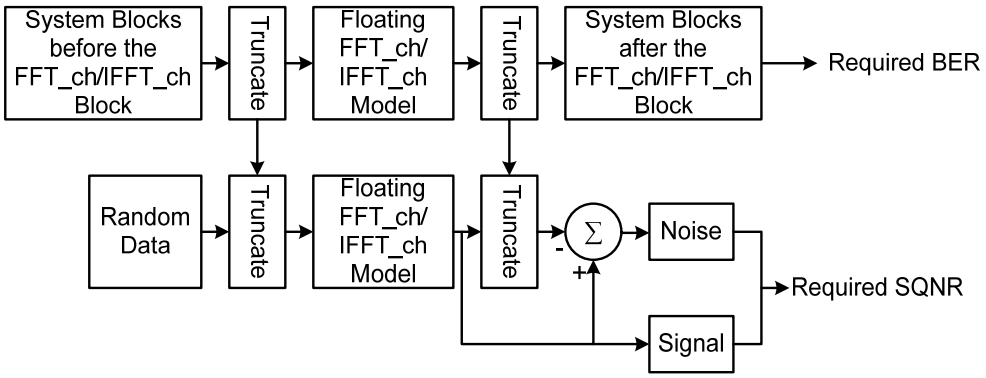


Fig. 4.22 System required SQNR simulation model

Since we combine the FFT and IFFT block to use the same hardware of FFT/IFFT processor, we must meet the system requirement for both FFT and IFFT blocks. The system required SQNR of the FFT_ch/IFFT_ch block decided by Fig. 4.22 is shown in Table 4-6.

Table 4-6 System required SQNR for FFT/IFFT processor

	Required SQNR
IFFT Mode	60.1 dB @ 1024 point input
FFT Mode	81.5 dB @ 8 point input

4.4.1 Fixed Point Simulation for Constant Multiplier in Radix-2/4/8 PE

A constant multiplier can always be composed of several shifters and adders. The quantization bits of the constant multiplier will affect the FFT/IFFT processor performance and hardware area.

By Eq. (4.3), the constant value of the multiplier is 1 divided by square root of 2, the output SQNR versus quantization bits of 1 divided by square root of 2 is shown in Fig. 4.23; in addition, the binary representation of 1 divided by square root of 2 is 0.10110101000001001111001 1. According to the system requirement, 8 fraction bits are the least required truncation bits to meet the system required SQNR 81.5 dB as shown in Table 4-6.

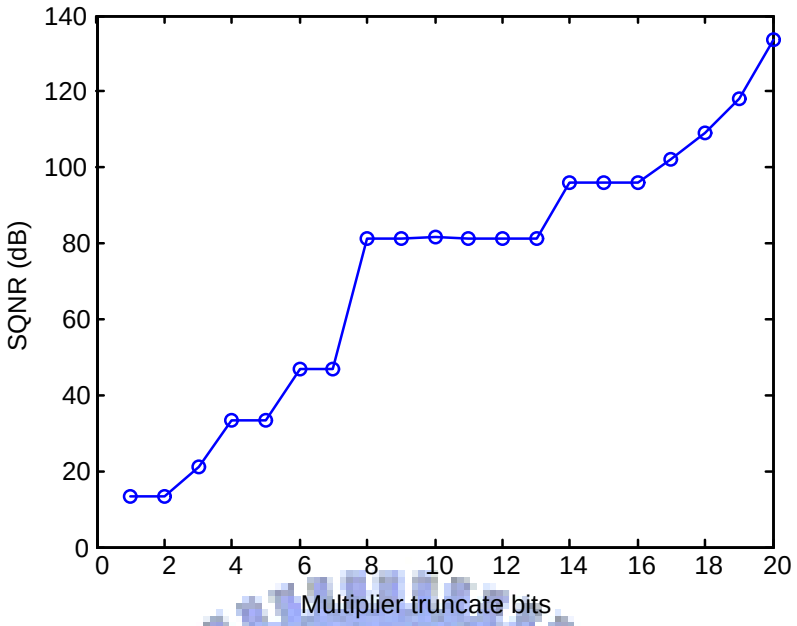


Fig. 4.23 SQNR versus constant multiplier truncate bits

4.4.2 Fixed Point Simulation for Twiddle Factor

We decide 8 fraction bits for constant multiplier to evaluate the performance with different word length of twiddle factor. The output SQNR versus the word length of twiddle factor is shown in Fig. 4.24. According to the system requirement, 17 bits of the twiddle factor are the least bits to meet the required performance.

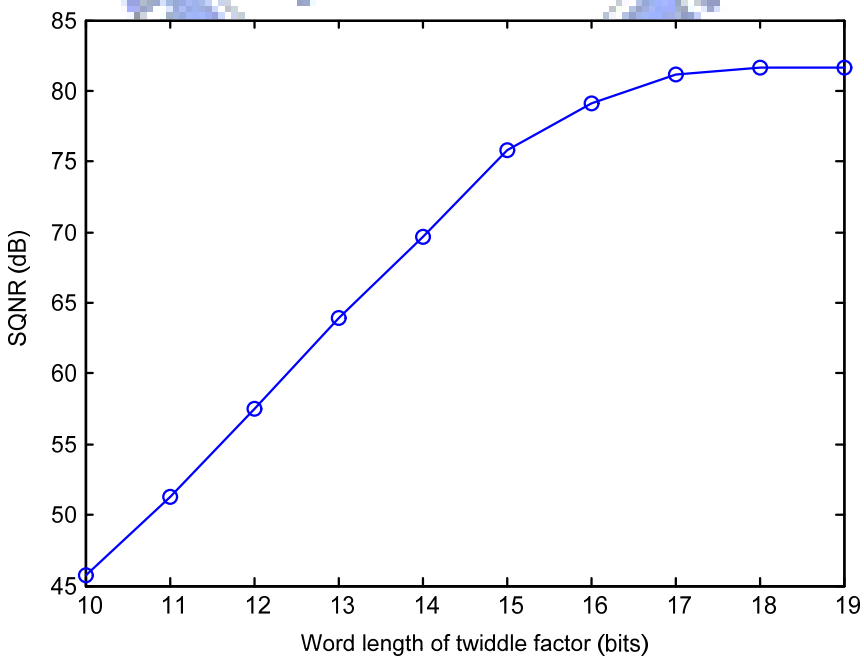


Fig. 4.24 SQNR versus word length of twiddle factor

4.4.3 Fixed Point Simulation for FFT/IFFT Processor

The dynamic scaling [24] is effective to reduce the internal word length without reduce the performance, especially in multiple stage of FFT/IFFT computation. In our application, the IFFT mode is assume 1024 point valid input data as usual. Without dynamic scaling, the hardware has to keep 13 bits of integer in each stage from the analysis in Section 4.3.7, which is inefficient. In this situation, dynamic scaling is so effective that the processor need less internal word length to meet the system requirement, which is shown in Fig. 4.25. From Fig. 4.25, the processor reduces 30% of the internal word length, which can reduce much of the hardware area. Another mode is FFT mode. The FFT mode has 8 point valid input data, so the output has to keep only 6 bits without dynamic scaling. From the analysis of Section 4.3.7, the scaling factor can be fixed in stage 1 but can not be fixed in other stage, where the stages are defined in Fig. 4.3. Fig. 4.26 shows the difference for with and without dynamic scaling. As the result, it also reduces 15% of the internal word length.

In order to meet the system requirement for both FFT and IFFT mode, we decide 20 bits for internal word length, 8 bits for constant multiplier, and 18 bits for twiddle factor.

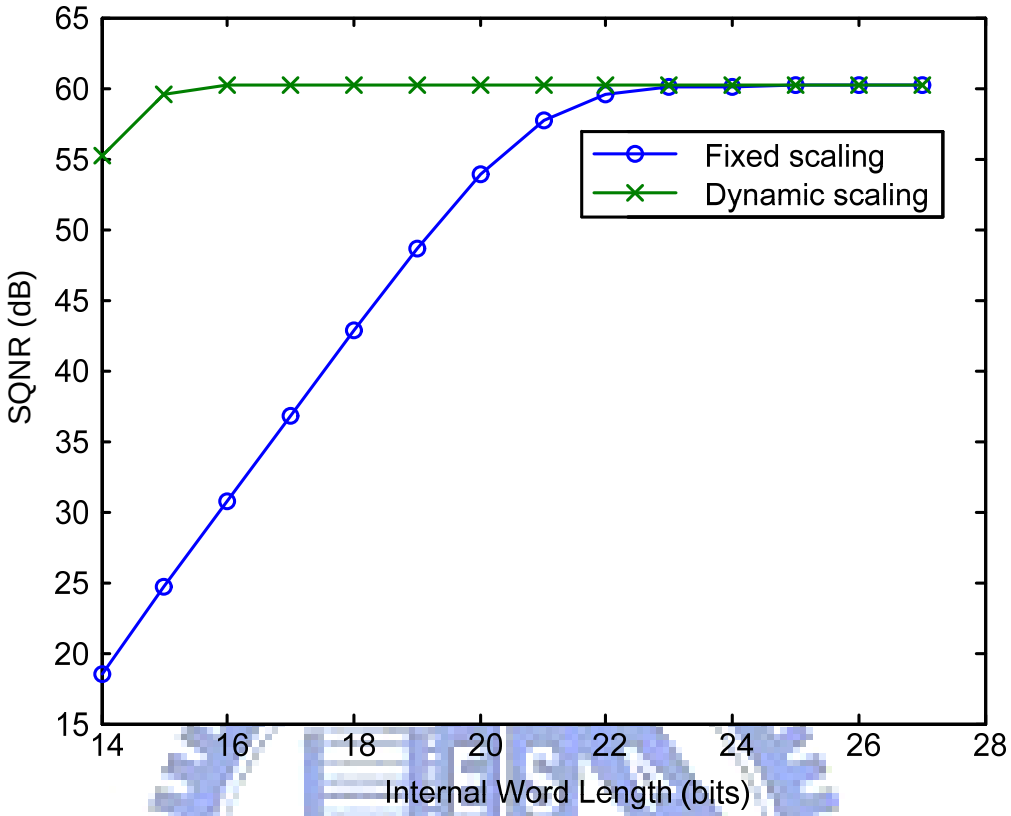


Fig. 4.25 SQNR versus internal word length in IFFT mode

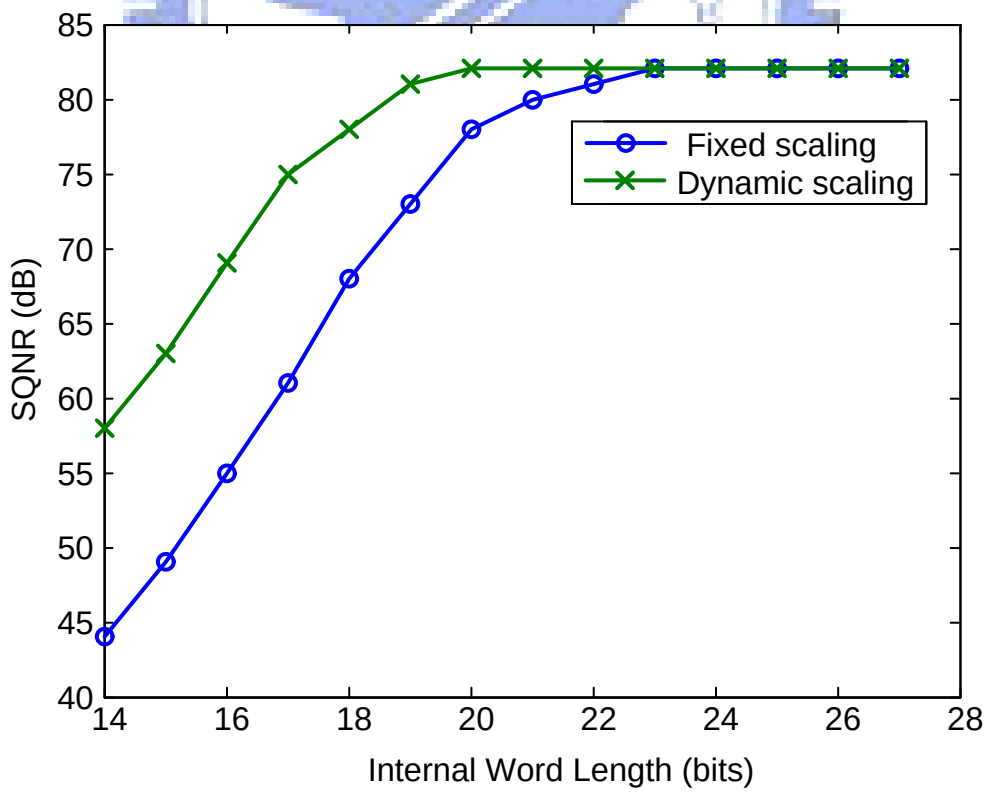


Fig. 4.26 SQNR versus internal word length in FFT mode

4.5 Hardware Implementation Result

4.5.1 Comparison for the FFT Processor Design Flow

The hardware cost comparisons of 5 versions of FFT processor are shown in Fig. 4.27. The FFT processor contains a 1024x46 bits dual port memory, a processing element, and a twiddle ROM, called version 1. The version which partition the memory of version 1 into 8 128x46 bits memory banks are called version 2. The parallel version of version2 containing 4 processing elements are called version 3. The version 3 also partition the twiddle factor ROM into 4 smaller twiddle factor ROMs. Version 4 changes the dual-port memories in version 3 into single-port memories. The final version is a dynamic scaling version of version 4, which reduces the area but has the same performance of version 4.

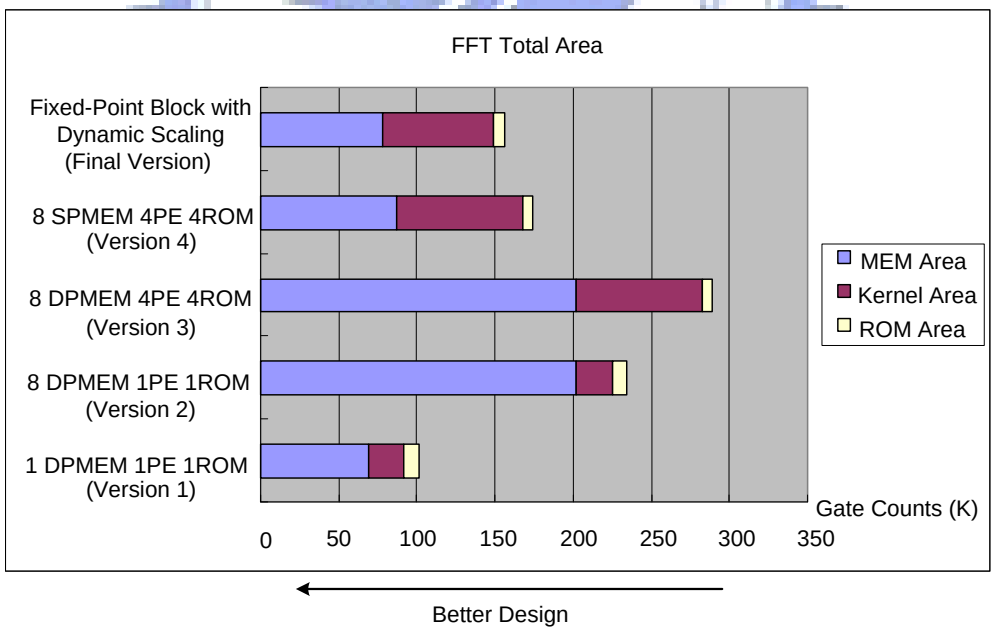


Fig. 4.27 Area comparisons for different versions of FFT processor

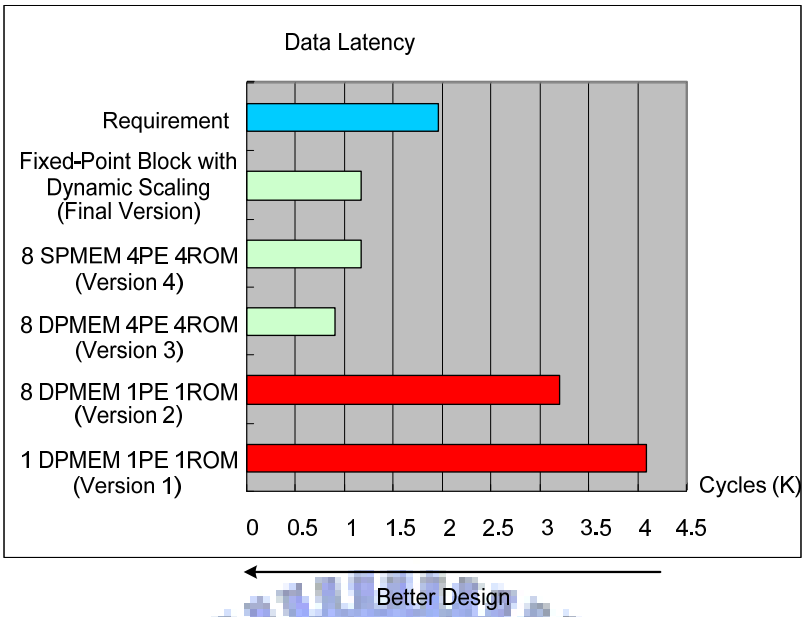


Fig. 4.28 Data latency comparisons for different versions of FFT processor

Table 4-7 Comparison of different version FFT processor

	Memory Area (gates)	Kernel Area (gates)	TW ROM Area (gates)	Total Area (gates)	Data Latency (cycles)
1024x46 DPMEM x 1 PE x 1 1024x36 ROM x 1 (Version 1)	68694 (34.0%)	22733 (28.2%)	9849 (100%)	101276 (35.1%)	4096 (100%)
128x46 DPMEM x 8 PE x 1 1024x36 ROM x 1 (Version 2)	202088 (100%)	22733 (28.2%)	9849 (100%)	234670 (81.2%)	3200 (78.1%)
128x46 DPMEM x 8 PE x 4 TW ROM x 4 (Version 3)	202088 (100%)	80418 (100%)	6381 (64.8%)	288887 (100%)	896 (21.9%)
128x46 SPMEM x 8 PE x 4 TW ROM x 4 (Version 4)	87272 (43.2%)	80418 (100%)	6381 (64.8%)	174071 (60.3%)	1169 (28.5%)
Scale down block fixed (Final version)	77816 (38.5%)	71566 (89.0%)	6410 (65.1%)	155792 (53.9%)	1169 (28.5%)

The comparisons of data latency for 5 versions of FFT processor are shown in Fig. 4.28 and Table 4-7. The final version reduces 63.5% of data latency as compared with version 2. Although final version's data latency is longer than version 3, but the final version saves 46.1% area as compared with version 3. Besides, the final version's data latency is shorter than the system requirement where the data latency of system requirement is 25us × 78.4 MHz = 1960 cycles. The comparisons of area for 5 versions of FFT processor are also shown in Table 4-7. Version 1 can not achieve the system requirement for parallel-in-parallel-out, so we ignore the area of version 1 in comparison. From Fig. 4.27, the final version cans save 61.5%, 10.8% of memory area as compared with version 2 and version 4. It also saves 11.0% of FFT kernel area as compared with version 4. Furthermore, the final version saves 34.9% of twiddle factor ROM area as compared with version 2. Finally, the final version saves 33.6%, 46.1%, 10.5% of total area as compared with version 2, version 3, and version 4.

4.5.2 Comparison of Separated Twiddle Factor ROM

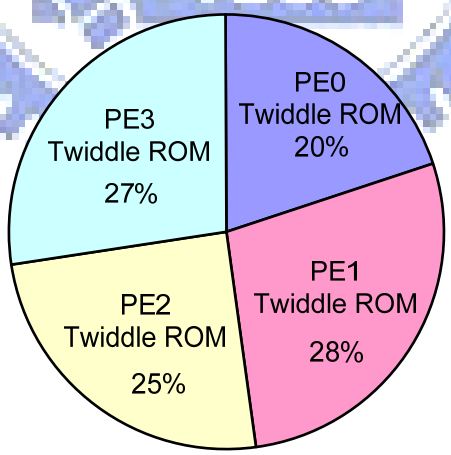


Fig. 4.29 Area comparison of separated twiddle factor ROM

As the discussion in Section 4.3.3, the conventional twiddle factor ROM is partitioned for different PE called PE-based TW ROM. The PE-based TW ROM is implemented by combinational circuits using synthesis tool to optimize the area of

each ROM. For instance, PE0 has many twiddle factors with the same number as discussed in Section 4.3.3. Thus, as the result in Fig. 4.29, the area of PE-based TW ROM for PE0 is lower than other twiddle factor ROM's area.

4.5.2 Hardware Implementation Result

As the hardware implementation results shown in Table 4-8, the proposed 1024-point FFT/IFFT processor can achieve the throughput rate up to 1.28 G samples/sec and the execution time down to 7.3 us when working at 160 MHz. When working at the system required 78.4 MHz, the execution time is 14.9 us which meets the system requirement of 25 us, and the power consumption is 21.7 mW with 155792 gates (including memory) that occupy 0.545 mm^2 by using 90 nm CMOS 1P9M 1V process.

Table 4-8 Hardware Implementation of the Proposed FFT/IFFT Processor

Items	Specification
FFT Size	1024 points
Process Technology	90 nm CMOS 1P9M 1V
Max Working Frequency	160 MHz
System Working Frequency	78.4 MHz
Throughput Rate	1.28G samples/sec @ 160MHz
Power Consumption	45.1 mW @ 160 MHz 21.7 mW @ 78.4 MHz (estimate by Design Compiler)
Gate count/Area	155792 gates @ 78.4 MHz (including memory)/ 0.545mm^2
Memory Size	8 x 128 words x 40 bits
Memory Area	77816 gates (8 bank memories)
Execution Time	7.3 us @ 160 MHz 14.9 us @ 78.4MHz

4.6 Summary

In order to evaluate the proposed FFT/IFFT processor, we compare the computation complexity and memory requirement in Table 4-8. It is apparent that compared with R8MDC and radix-8 memory-based, the proposed FFT processor requires less complex multipliers and no reorder buffer. As the result, the proposed FFT/IFFT architecture can meet the system requirement with the least hardware complexity.

Table 4-9 Comparison of several high throughput FFT architectures

	$R2^3SDF$	**R8MDC**	**Radix-8 Memory-Based**	**Proposed**
Complex Multipliers	3	21	7	4
Complex Adders	20+6T	88+6T	24+2T	28+8T
Memory Banks	13 (dual port)	47 (dual port)	8 (dual port)	8 (single port)
Memory Size (words)	1023	1976	1024	1024
Reorder Buffer Size	512	960	960	0
Data Latency (cycles)	1535	360	872	1160
Throughput Rate (clock rate is R)	R	8R	8R	8R

Also, we use the proposed FFT processor hardware cost to evaluate the hardware cost of different architectures, and the results are shown in Table 4-9. It is apparent that the proposed FFT processor can save about 81.0% and 43.9% complex multipliers as compared with R8MDC and radix-8 memory-based. The memory bank composed of single port memory reduces about 64.8% and 32.0% area of R8MDC and radix-8 memory-based. Moreover, it frees the requirement of reorder buffer which

needs area of 69490 gates. The data latency of the proposed FFT processor is 14.9 us at 78.4 MHz and meets the system requirement of 25 us. Meeting the system requirement, the proposed FFT processor has the least hardware cost for low complexity design.

Table 4-10 Comparison of hardware cost for different architectures

	R8MDC	**R8M**	**Proposed**
Complex Multipliers	180516 (100%)	60172 (33.3%)	**34384 (19%)**
Complex Adders	22243 (100%)	6289 (28.2%)	10808 (47.6%)
Memory Banks	47 (dual port)	8 (dual port)	**8 (single port)**
Memory Size (40bits)	138326 (100%)	71672 (51.8%)	**48744 (35.2%)**
Reorder Buffer Size	69490 (100%)	69490 (100%)	**0 (0%)**
Total	410575 (100%)	207623 (50.5%)	**93936 (22.9%)**
Data Latency (cycles)	360	872	1169
Throughput Rate (clock rate is R)	8R	8R	8R

Chapter 5

Chip Implementation of IEEE 802.16e Receiver

This chapter will introduce the chip design flow for IEEE 802.16e baseband receiver. The 802.16e baseband receiver is including a frequency divider, a synchronization block, a FFT processor for FFT_dem block with 5 memory banks, and a channel estimation block with FFT/IFFT processors for FFT_ch/IFFT_ch blocks as shown in Fig. 2.5 and Fig. 2.6.

5.1 Design Flow

The 802.16e baseband receiver system is modeled in C language. For hardware implementation, each component uses fixed-point simulation to have the least performance decreasing as compared with the floating system model. After the word length of each component is decided, the hardware implementation of each component is modeled in Verilog language, called RTL design. Besides the performance analysis, the RTL design of each component in Verilog is verified in Verilog XL with the result generated by the model in C language, called RTL verification. If the RTL verification is done, the RTL code is synthesized in synthesis tool, Design Compiler, with suitable constrain, and we usually have timing overdesign in this stage because the synthesis tool don't have the real line delay. The gate-level netlist generated by synthesis tool is verified with the result generated by RTL verification in Verilog XL, called gate-level verification. APR (Automatic Place and Route) tool, such as SOC Encounter, help us to implement the chip from gate-level netlist and also help us to make sure the chip meet the layout design rule. The layout

file (GDS) generated by APR tool is verified in Calibre by DRC (Design Rule Check) and LVS (Layout versus Schematic). The post-layout simulation using gate-level netlist generated by APR tool is used to verify the result as compared with gate-level simulation. Usually, the chip has post transistor-level simulation before the chip is taped out. However, the simulation time is too long if the chip has too many transistors. The 802.16e baseband receiver has over 1 million gates and is too large to simulate by post transistor-level simulation. Thus, we skip the simulation in this stage. Finally, the chip is taped out.

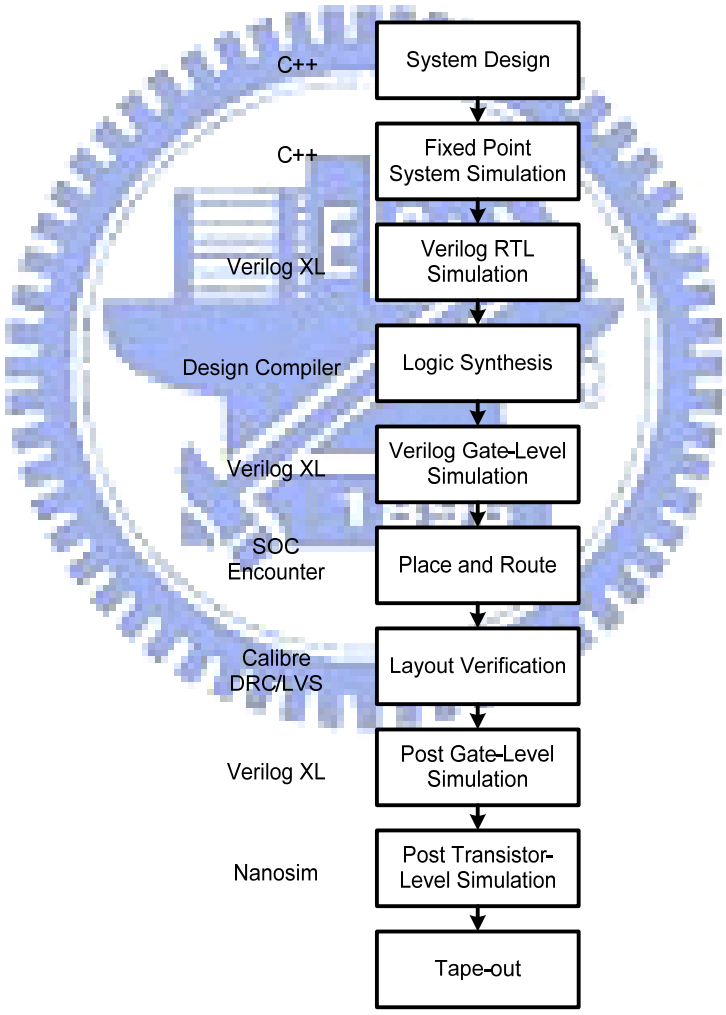


Fig. 5.1 Cell based chip design flow

5.2 Multi-Frequency Design

The 802.16e baseband receiver has two clock domains. One is 11.2 MHz for achieving the required data rate to IEEE 802.16e. The other is 7 times of 11.2 MHz which equals to 78.4 MHz. Since there are several combinational circuits between the registers in different clock domain, we have to set the different timing constrain in those path for the respected timing check. An example of two clock domains is shown in Fig. 5.2, and the default timing check is shown in Fig. 5.3.

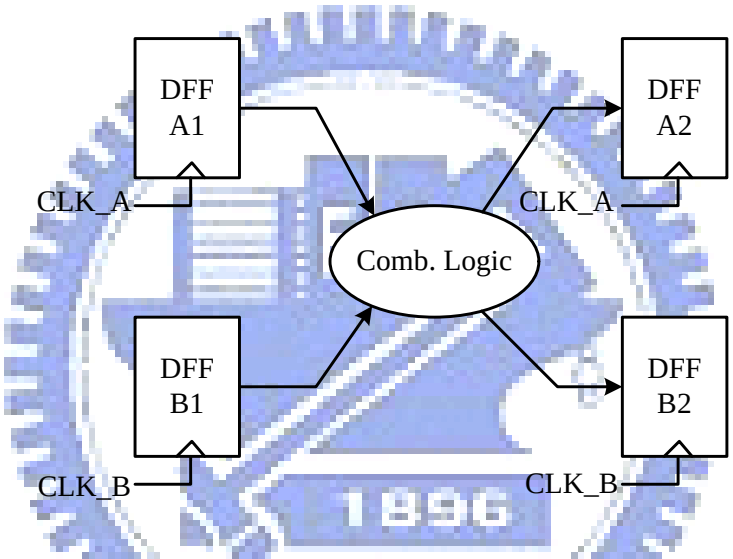


Fig. 5.2 Combination logic circuits between 2 clock domains

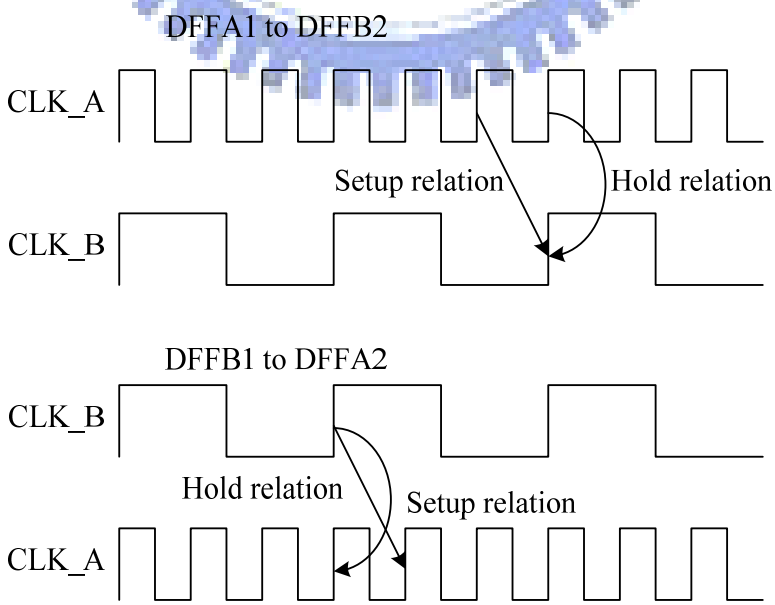


Fig. 5.3 Default timing check in 2 clock domains

The frequency of CLK_A is 3 times of CLK_B, and the 2 different conditions of timing check in different clock domains are shown in Fig. 5.3. For the upper case of Fig. 5.3 (DFFA1 to DFFB2), the default timing check leads the timing constrain of the combination circuits is limited in cycle of CLK_A. And, for the lower case of Fig. 5.3 (DFFB1 to DFFA2), the default timing check also leads the timing constrain of the combination circuits is limited in cycle of CLK_A. However, in the lower case, the expected timing constrain of the combination circuits usually is cycle of CLK_B shown in Fig. 5.4. Therefore, we have to correct the default timing check by setting the SDC (Synopsys Design Constrain) constrain in synthesis tool. The commands of SDC constrain for changing the timing constrain from Fig. 5.3 to Fig. 5.4 are "set_multicycle_path 3 –end –setup –from CLK_B –to CLK_A" and "set_multicycle_path 2 –end –hold –from CLK_B –to CLK_A".

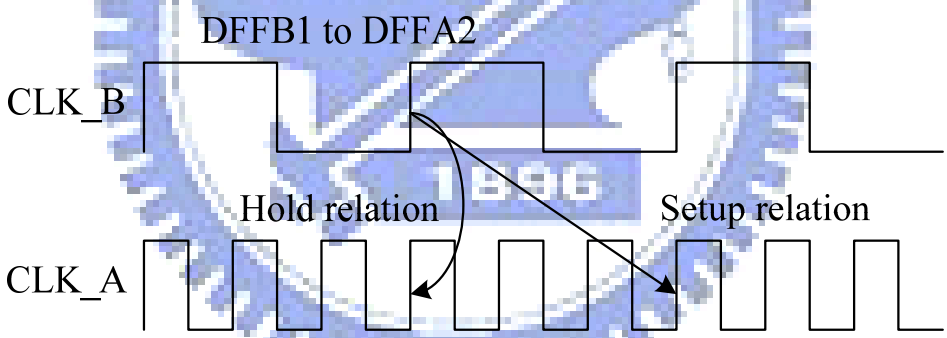


Fig. 5.4 Expected timing constrain for DFFB1 to DFFA2

In our case, FFT_dem block gets the signal from synchronization block, and the synchronization is working at the low clock frequency 11.2 MHz while the FFT_dem block is working at the high clock frequency 78.4 MHz. Thus, we have to set the commands for SDC constrain, too, and the commands are similar to the commands mentioned before.

Since we have two clock domains, frequency divider is used in our baseband receiver design. In synthesis stage, the frequency divider will introduce the clock

skew, which should be fixed by APR tool if we synthesize the receiver with frequency divider. Therefore, we separate the receiver into 2 parts, one is frequency divider, and the other is circuits with 2 ideal clock input. The two parts of receiver are synthesized individually, and combined in APR tool shown in Fig. 5.5. In addition, the gate-level verification is verified the gate-level netlist of circuits without frequency divider, and is simulated with 2 ideal clocks.

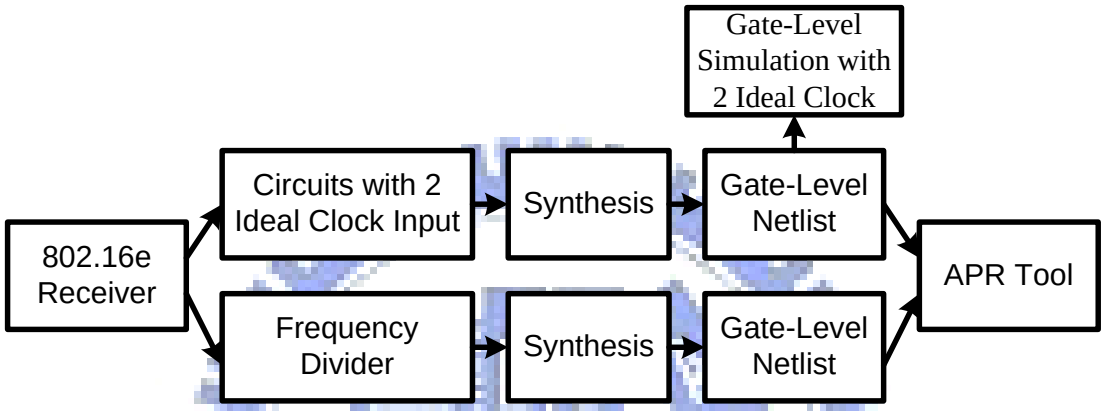


Fig. 5.5 Synthesis flow of chip with frequency divider

5.3 Chip Floor Plan

Since the 802.16e baseband receiver is a sequential system, the floor plan of the baseband receiver is based on the sequential order of the receiver shown in Fig. 5.6. From Fig. 5.6, the components of the receiver based on the sequential order are planed from north to south of the whole chip.

As the result of APR, the chip size of the 802.16e baseband receiver is 3211 × 3211 um^2; however, the size of the chip is too large to piece together with other chips in a shuttle since the shuttle size is 4000 × 4000 um^2. In order to tape out with other chips, a rectangular version of the receiver chip is used to replace the square version shown in Fig. 5.7. The chip size of rectangular version is 3955 × 2755 um^2 which is large than that of square version but is more flexible to piece together with other chips in a shuttle.

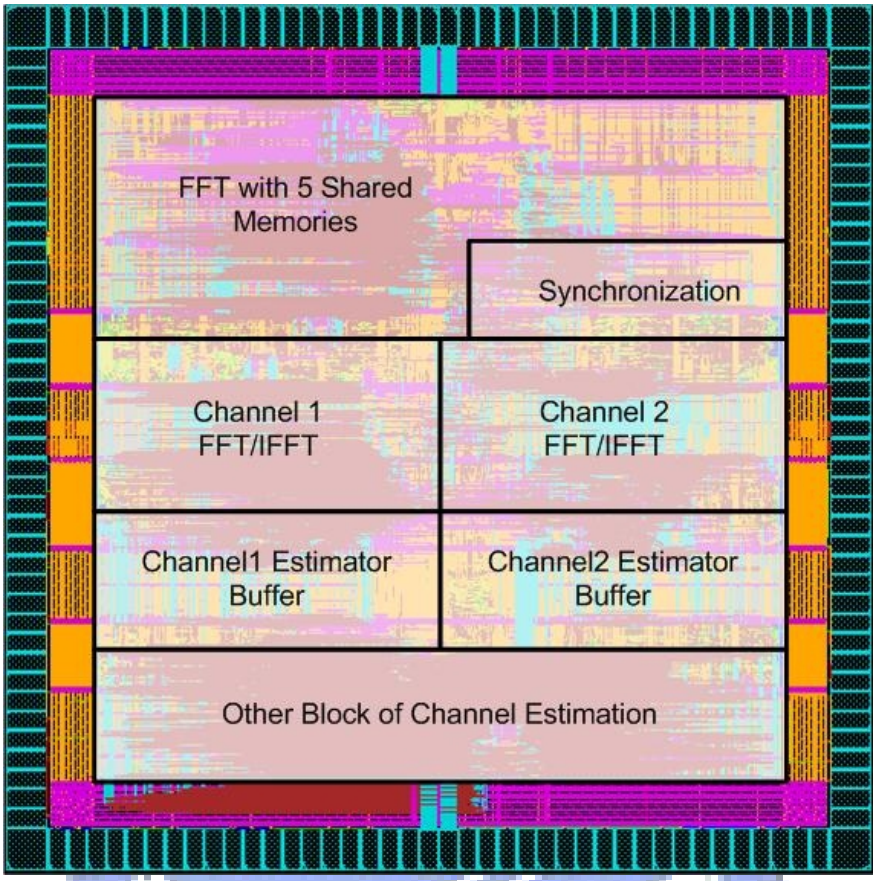


Fig. 5.6 Floor plan of the 802.16e baseband receiver

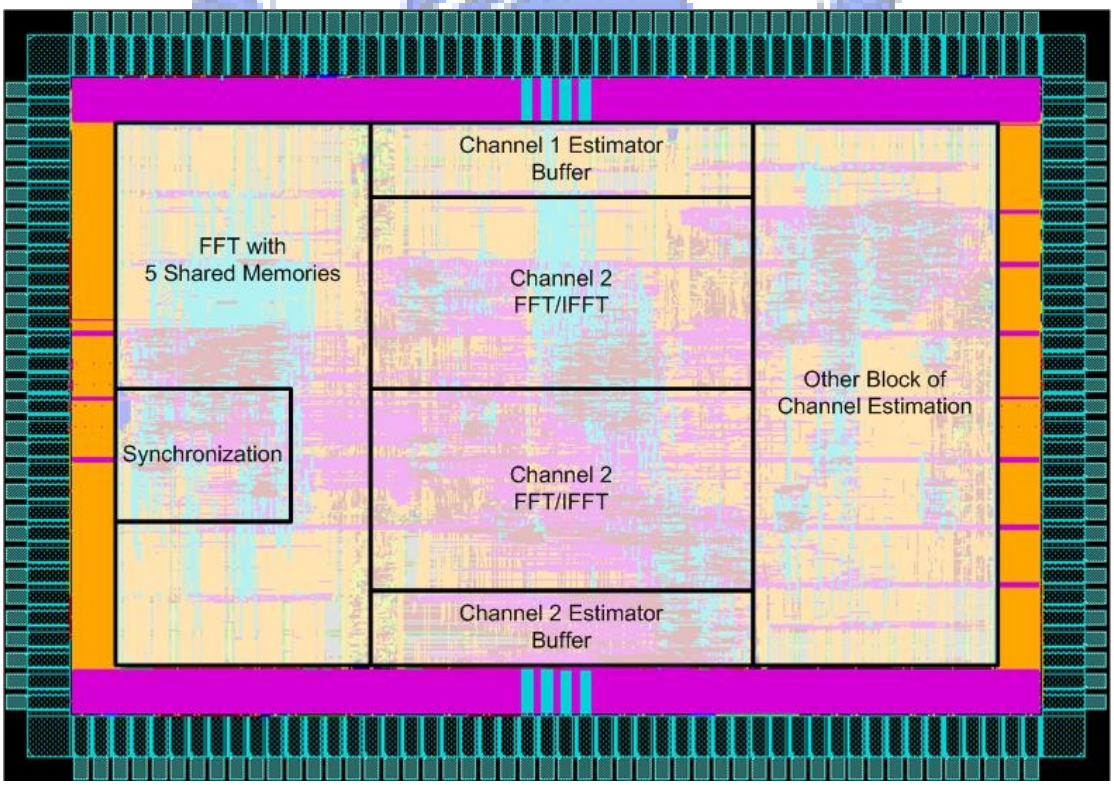


Fig. 5.7 Rectangular version floor plan of the 802.16e baseband receiver

5.4 Chip Summary

The chip summary is shown in Table 5-1. The square version is prepared to tape out from CIC, and the rectangular version is directly taped out from UMC. The cell library and PAD library is different between the two versions: square version's library is from Faraday, and rectangular version's is from UMC. As the results shown in Table 5-1, the square version's working frequency can meet the system specification while the rectangular version's can not. In summary, the taped out version chip size is 3955×2755 um^2, power consumption is 47.1 mW at 8.2/57.1 MHz, and is using UMC 90nm 1V CMOS process. Moreover, the area of two FFT/IFFT processors for FFT_ch/IFFT_ch blocks in DF DF-based CE in the taped out chip is 1.711 mm^2, and the power consumption of that is 20.2 mW working at 57.1 MHz.

Table 5-1 Chip summary

Item		Specification		
		Square Version	Rectangular Version (taped out)	FFT_ch/IFFT_ch Processors (taped out)
Technology		UMC 90nm CMOS 1P9M 1V (Cell/PAD from Faraday)	UMC 90nm CMOS 1P9M 1V (Cell/PAD from UMC)	UMC 90nm CMOS 1P9M 1V (Cell/PAD from UMC)
Area(um^2)	Core	2411×2411	3144×1944	1.711 mm^2
	PAD Core	3057×3057	3799×2599	
	Chip	3211×3211	3955×2755	
Working Frequency		11.2/78.4 MHz	8.2/57.1 MHz	57.1 MHz
Power Consumption (Simulation)		68.5 mW	47.1 mW	20.2 mW
PAD	Input	26	26	N/A
	Output	66	66	
	Power/GND	66	68	
	Bias for I/O PAD (only for PAD from Faraday)	2	N/A	
	Total	160	160	

Chapter 6

Conclusion and Future Work

In this thesis, a FFT/IFFT processor with parallel-in-parallel-out in normal order which is used in a DF DFT-based channel estimation block is proposed. A 802.16e baseband receiver including this DF DFT-based channel estimation is taped out.

To design a FFT/IFFT processor with parallel-in-parallel-out in normal order, we analyze different parallel-in-parallel-out FFT architecture, and try to design the FFT/IFFT processor based on memory-based architecture. Memory allocation helps us to design a FFT/IFFT processor with parallel-in-parallel-out in normal order, and commutator design helps us to use single port memories to reduce the area of memories. These two methods can also be applied to different specification of parallel-in-parallel-out FFT processor. As the synthesis results, the proposed 1024-point FFT/IFFT processor can achieve the throughput rate up to 1.28 G samples/sec and the execution time down to 7.3 us when working at 160 MHz. When working at the system required 78.4 MHz, it consumes 21.7 mW with 155792 gates (including memory) that occupy 0.545 mm^2 by using 90 nm, 1V CMOS process.

A study of partial FFT for DF DFT-based channel estimation is also presented in this thesis. The pruning algorithm with only a subset of input or output points can help us to decrease the FFT processor hardware cost, and the multiple subsets of input or output points help us to save more power in FFT computation. As the analysis, the proposed partial FFT processor can reduce 75.1% of the memory size, 22.3% of the complex multipliers, and 30% of the complex adders as compared with traditional radix-2 SDF FFT architecture. Furthermore, with increasing the partial FFT control

for the proposed partial FFT processor, the proposed partial FFT can reduce maximum 65.3% of multiplication operations and 49.5% of addition operations, which may save more power if the 8 valid output point's indices have common bits.

In the future, since we only implement the FFT/IFFT processor with parallel-in-parallel-out in normal order, a suitable FFT/IFFT processor for DF DFT-based channel estimation have to keep on study, such as the FFT/IFFT processor combining partial FFT algorithm and MIMO FFT concept.

Reference

[1] R.W. Chang, ”Synthesis of Band-Limited Orthogonal Signals for Multichannel Data Transmission”, Bell Syst. Tech. J., Vol.45, pp. 1775-1796, Dec. 1966.

[2] IEEE, Std. 802.16-2004: Air Interface for Fixed Broadband Wireless Access Systems, 2004.

[3] IEEE, Std. 802.16e: Air Interface for Fixed and Mobile Broadband Wireless Access Systems, 2005.

[4] M. Julia., F. G. Garcia, M. Jose, P. B., S. Zazo, **“**DFT-based channel estimation in 2D-pilot-symbol-aided OFDM wireless systems” IEEE Vehicular Technology Conference, Vol. 2, pp. 810-814, May 2001.

[5] V. Tarokh, N. Seshadri, and A. R. Calderbank, “Space-time codes for high data rate wireless communication: Performance analysis and code construction,” IEEE Trans. Inform. Theory, Vol. 44, No. 2, pp. 744–765, Mar 1998.

[6] IEEE Std. 802.16-2001 IEEE Standard for Local and Metropolitan area networks Part 16: Air Interface for Fixed Broadband Wireless Access Systems.

[7] Y. Li, “Channel Estimation for OFDM Systems with Transmitter Diversity in Mobile Wireless Channels,” IEEE J. Selected Areas in Commun., Vol. 17, pp. 461-471, Mar. 1999.

[8] Y. Li, “Simplified Channel Estimation for OFDM Systems With Multiple Transmit Antennas,” IEEE Trans. Wireless Commun., Vol. 1, pp. 67-75, Jan. 2002.

[9] J-J V. D. Beek, O. Edfors, M. Sandell, S. K. Wilson and P. O. Brjesson, “On channel estimation in OFDM systems,” Vehicular Technology Conf., pp. 815-819, 1995.

[10] M. L. Ku and C. C. Huang, “A Derivation on the Equivalence between Newton’s Method and DF DFT-Based Method for Channel Estimation in OFDM Systems,” submitted to IEEE Trans.Wireless Commun.

[11] Rabiner, L.R., and Gold, B. “Theory and application of digital signal processing” (Prentice Hall, 1975).

[12] J. W. Cooley and J. W. Tukey, “An Algorithm for Machine Computation of Complex Fourier Series,” Math. Computation, Vol. 19, pp. 297-301, April 1965.

[13] S. He and M. Torkelson, “A New Approach to Pipeline FFT Processor,” Parallel Processing Symposium, pp. 766-770, 1996.

[14] S. He and M. Torkelson, “Designing Pipeline FFT Processor for OFDM (de) Modulation,” URSI International Symposium on Signals, Systems and Electronics, pp. 257-262, 1998.

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

[16] S. Magar, S. Shen, G. Luikuo, M. Fleming, and R. Aguilar, “An application specific DSP chip set for 100 MHz data rates,” in Proc. Int. Conf. Acoustics, Speech, and Signal Processing, Vol. 4, pp. 1989-1992, Apr. 1988.

[17] B. M. Bass, “A low-power, high-performance, 1024-point FFT processor,” IEEE J. Solid-State Circuits, Vol. 34, No. 3, pp. 380–387, Mar. 1999.

[18] Y. W. Lin, H. Y. Liu, and C. Y. Lee, “A 1-GS/s FFT/IFFT Processor for UWB Applications,” IEEE journal of solid-state circuits, Vol. 40, No. 8, pp. 1726-1735, Aug 2005.

[19] T. Sansaloni, A. Pe´rez-Pascual, V. Torres and J. Valls, “Efficient pipeline FFT processors for WLAN MIMO-OFDM systems,” Electronics letters 15th, Vol. 41, No. 19, Sep 2005.

[20] L. Jia, Y. Gao, J. Isoaho, and H. Tenhunen, “A new VLSI-oriented FFT algorithm and implement,” in Proc. 11th Annu. IEEE Int. ASIC Conf., pp. 337–341, Sep 1998.

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

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

[23] H. V. Sorensen, “Efficient Computation of the DFT with Only a Subset of Input or Output Points,” IEEE Transactions on signal processing, Vol. 41, No. 3, pp. 1184-1200, March 1993.

[24] Y. W. Lin, H. Y. Liu, and C. Y. Lee, “A Dynamic Scaling FFT Processor for DVB-T Applications,” IEEE Journal of solid-state circuits, Vol. 39, No. 11, pp. 2005-2013, November 2004.

[25] J. D. Markel, “FFT pruning,” IEEE Trans. Audio Electroacoust., Vol. 19, No. 4, pp. 305-311, Dec. 1971.

[26] D. P. Skinner, “Pruning the decimation in-time FFT algorithm,” IEEE Trans. Acoust., Speech, Signal Processing, Vol. 24, No. 2, pp. 193-194, Apr. 1976.

[27] H. V. Sorensen, “Efficient Computation of the DFT with Only a Subset,” IEEE Transaction on signal processing, Vol. 41, No. 3, March 1993.

[28] C. M. Chen, Y. H. Huang, “Partial Cached-FFT Algorithm for OFDMA Communications,” IEEE TENCON, Oct 2007.

[29] L. Jia, Y. Gao, J. Isoaho and H. Tenhunen, “A New VLSI-Oriented FFT Algorithm and Implementation”, IEEE International ASIC Conference, pp.

337-341, Sep 1998.
[30] Xilinx Corporation, “Fast Fourier Transform,” LogiCore v3.1, Nov 2004.