應用多重區域條件式成組縮放法於快速傅利葉轉換處理器之面積最小化技術

Area Minimization for FFT Processor Using Multi-Region Conditional Block Scaling

研 究 生：陳柏霖　　Student：Po-Lin Chen

指導教授：周景揚　　Advisor：Jing-Yang Jou

國 立 交 通 大 學

電子工程學系　電子研究所

碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering and 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

September 2012

Hsinchu, Taiwan, Republic of China

中華民國一〇一年九月

應用多重區域條件式成組縮放法於快速傅利葉轉換處理器之面積最小化技術

學生：陳柏霖　　指導教授：周景揚 博士

國立交通大學
電子工程學系　電子研究所碩士班

摘　要

快速傅利葉轉換處理器是正交分頻多工系統的計算核心，並且在過去幾十年間可以找到許多研究資料。為了提升定字元長度傅利葉轉換處理器的訊號對量化雜訊比，動態縮放法在執行運算時適應性地決定其縮放行為以避免不必要的精確度流失。當訊號對量化雜訊比的要求提高時，傳統上是去增加字元長度以得到更高的精確度。然而增加字元長度在面積上會付出許多代價，再者，有時候其實並不需要將精確度提升這麼多去滿足要求。在這篇論文裡，我們提出了一個利用了條件式縮放法及成組縮放法的優點的動態縮放法。此方法擁有非常經濟地使用面積去提升訊號對量化雜訊比的能力而不只是單純地去增加字元長度。因此，此方法的目標是在滿足訊號對量化雜訊比的要求下得到最小化的快速傅利葉轉換處理器的面積。實驗結果顯示在最佳的情形下，我們的方法相較於原本的條件式縮放法可以對 8192 點的快速傅利葉轉換處理器省下約 13% 的面積。

Area Minimization for FFT Processor Using Multi-Region Conditional Block Scaling

Student：Po-Lin Chen Advisor：Dr. Jing-Yang Jou

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

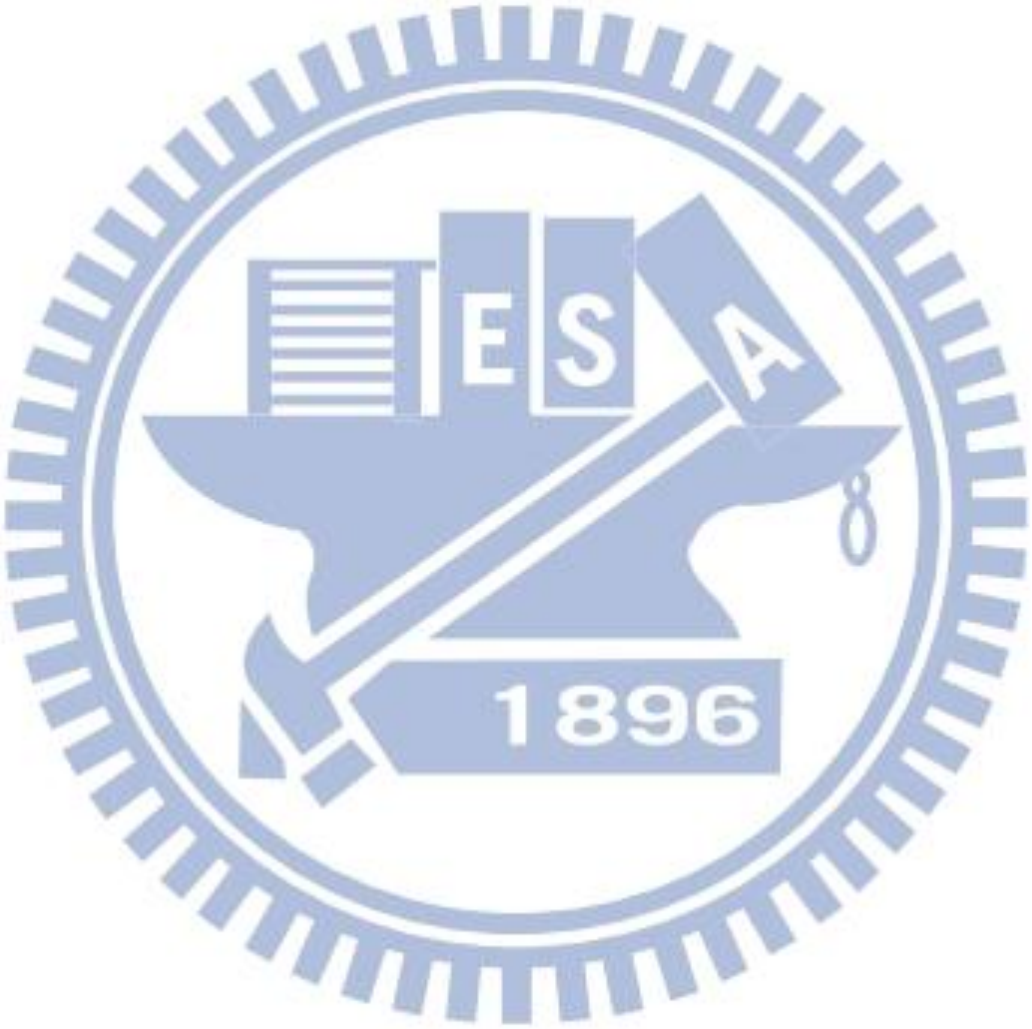
The Fast Fourier Transform (FFT) processor is the key component of OFDM-base systems, and many literatures of FFT can be found in the past decades. To improve the SQNR in fixed-wordlength FFT, dynamic scaling methods adaptively determine the scaling behavior in run time to avoid the unnecessary loss of data information. Traditionally, increasing the wordlength is the way to acquire higher precision if the SQNR constraint is tighter. However, the increased wordlength results in a large amount of area cost. Moreover, sometimes we do not have to increase SQNR so much to meet the constraint. In this thesis, we proposed a dynamic scaling scheme which utilizes the profits of conditional scaling method and block scaling method. Our approach has the ability to economize the usage of area rather than increase the wordlength for SQNR improvement and the target is to minimize the area of FFT under the SQNR constraint. Experimental results show that our approach can reduce the area cost by about 13% in the best case for 8192-point FFT as compared to the existing conditional scaling method.

Acknowledgements

I would like to express my sincere gratitude to my advisors, Dr. Jing-Yang Jou for his patient guidance and valuable suggestion during these years. I am also grateful to Dr. Juinn-Dar Huang for his help on my research. Besides, I have many thanks to my seniors Bu-Ching Lin, Yung-Chun Lei and my lab-mate Chuang-Ren Yang for their gentle teaching, directing, and beneficial discussion. Moreover, I greatly appreciate all members of NCTU EDA Lab. All of you are so nice that I will remember the happy time during the past two years with you. Finally, I wish to give my deepest acknowledgements to my family and my girlfriend, Pei-Chen Wu for their love, support, and encouragement.

Contents

List of Figures



List of Tables

Chapter 1
Introduction

In recent years, research and development on high-data-rate wireless communications have attracted great attention. Orthogonal frequency-division-multiplexing (OFDM) is the modulation technique which is a favorable choice for many new and emerging broadband communication applications, such as local area networks (WLAN) [1], high definition television (HDTV) [2], digital video broadcasting-terrestrial (DVB-T) [3], and digital audio broadcasting (DAB). The simplified architecture of OFDM system is shown in Fig.1.

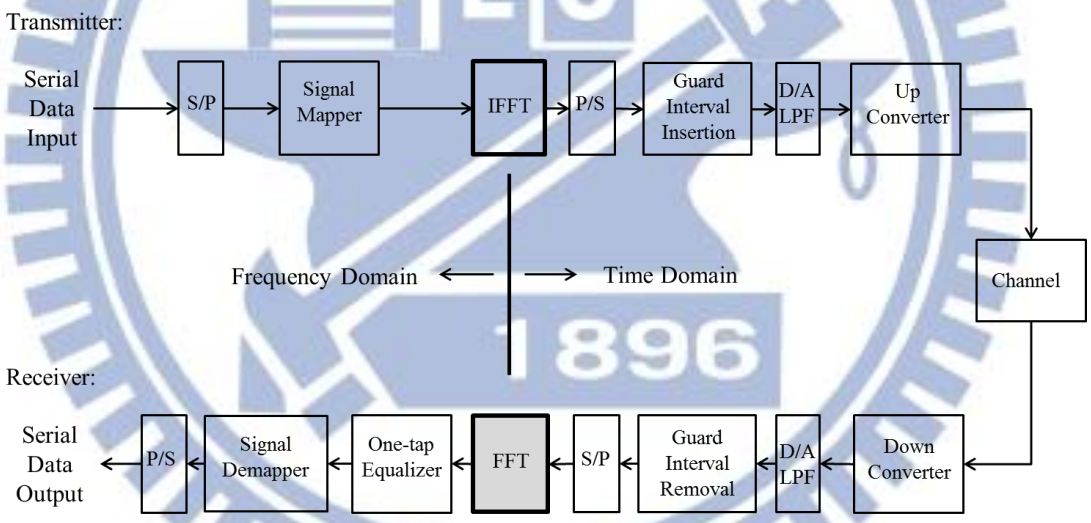


Fig. 1 Simplified architecture of OFDM system

In those applications, Fast Fourier Transform (FFT) is the most widely used algorithms for calculating the Discrete Fourier Transform (DFT) because of its efficiency in reducing computation time. Therefore, it is one of the most important processing blocks to meet the design constraints. In [4], the authors showed that in such high-data-rate systems, the most computationally intensive part is the FFT core. Therefore, there have been many literatures reported on the design of FFT processors nowadays.

Based on the hardware cost and the required throughput, there are two main categories of FFT architectures. One is called memory-based architectures, which consist of a butterfly unit and certain number of memory blocks. The other is called pipeline-based architecture, which consists of multiple stages to provide higher throughput. In general, memory-based architectures are suitable for long-size and low hardware design [5]. And pipeline-based architectures are feasible for short-size and high throughput design. In this thesis, we focus on the long-size FFT design with memory-based architecture where the wordlength *WL* of the output in every stage is the same as that of the input.

Taking the practical design into consideration, the precision of FFT module in terms of Signal to Quantization Noise Ratio (SQNR) is a significant design factor of system performance. In reality, an FFT cannot be implemented exactly since the algorithm is implemented by fixed-point arithmetic. All signals and coefficients have to be represented with finite number of bits in binary format. Therefore, conducting the addition and subtraction operations in butterfly unit may cause overflow during the FFT computations. For this reason, the wordlength should be increased stage by stage to avoid possible overflow after butterfly computations. The increasing wordlength can be used to avoid accuracy loss and increase the precision [6], but the hardware cost and the critical-path delay are increased accordingly and it is unsuitable for memory-based architecture. Consequently, rounding or truncation operations introduce noise which is referred as quantization noise and result in accuracy loss. Furthermore, the wordlength may also affect the accuracy. Longer wordlength may be used to achieve better SQNR with larger area, and shorter wordlength may be chosen to maintain a lower hardware cost at the sacrifice of the precision. Therefore, many scaling methods have been proposed to meet SQNR requirement with the fixed-wordlength constraint [7-12].

Oppenheim *et al.* [7] proposed a basic scaling method which scales the results by a factor of 1/2 for each stage. That is, the results are divided by two after each butterfly calculation. Since it is trivial to implement, the approach is the simplest but the least accurate scaling

method called the forced scaling method. Another is called the Block Floating Point (BFP) scaling [11] which employs intermediate buffers to store the output data, and detects the largest value to decide the output format appropriately which gets better SQNR. However, this kind of method results in a large amount of area overhead and power consumption. Besides, there is another approach which is called conditional scaling [13, 14]. The idea of the method is to predetermine whether to scale in next stage according to the magnitude of the results in current stage. That is to say, after each butterfly computation, the magnitudes of the results are compared to a threshold and the results can be written back to the memory instantly after the comparisons. After all comparisons are finished, it will be judged that overflow will occur in next stage if one or more values exceed 0.5. Thus, after each computation of the butterfly in next stage, the results will be scaled to avoid overflow. Although the conditional scaling method acquires less SQNR than BFP, it saves much area since it does not need the buffers to store internal results.

In this thesis, we propose a dynamic scaling method combining the concepts of BFP scaling and conditional scaling with memory-based architecture to acquire higher SQNR in an area-efficient way. Our method not only improves the previous conditional scaling method to predict overflow more precise but also modifies BFP scaling to divide data into blocks appropriately. Moreover, our method can minimize the area with given FFT size and SQNR constraint.

The remainder of this thesis is organized as follows. In Chapter 2, we briefly review the fundamentals of FFT algorithms, architectures and previous scaling methods. Chapter 3 explains the motivation of this work. The proposed method is demonstrated in Chapter 4. Our experimental results are shown in Chapter 5. Finally, Chapter 6 gives the concluding remarks of this thesis.

Chapter 2
Preliminaries

In this chapter, we will review basic FFT algorithms, FFT architectures, the scaling considerations and previous scaling methods in FFT hardware design.

2.1 The FFT Algorithms

The Discrete Fourier Transform (DFT) plays an important role in the region of digital signal processing (DSP) and communications. However, the computation complexity of directly evaluating an N-point DFT is $O(N^2)$, which costs a lot amount of computation time and power consumption. Therefore, a fast algorithm to evaluate DFT is required.

FFT algorithm is a decomposition of an N-point DFT into successively smaller DFT transform which was proposed by Cooley and Turkey [15] in 1965. It is very popular because it reduces the complexity of DFT from $O(N^2)$ to $O(N \log_2 N)$, and makes it suitable for VLSI implementation due to the regularity of the algorithm. Besides, many similar algorithms have been developed to further reduce the computational complexity of FFT [16-18]. Owing to these algorithms, FFT computes the DFT efficiently and produces exactly the same result as evaluating the DFT equation.

2.1.1 Basic Concepts of FFT Algorithms

FFT algorithms are approaches to evaluate DFT. The formulation of N-point DFT is defined as

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

Where $X(k)$, $x(n)$ and W_N^{nk} are complex numbers. $X(k)$ is in frequency domain, and $x(n)$ is in time domain. The coefficient W_N^{nk} is defined as (2.2) and is called the twiddle factor which the symmetric property is shown in Fig. 2.

$$W_N^{nk} = e^{\frac{-j2\pi nk}{N}} = \cos(\frac{2\pi nk}{N}) - j\sin(\frac{2\pi nk}{N}) \tag{2.2}$$

Decimation-in-time (DIT) FFT algorithm is to decompose the input sequence $x(n)$ into smaller and smaller sequence. Alternatively, decimation-in-frequency (DIF) FFT algorithm is to decompose the output sequence $X(k)$ in the same way. Both of these two algorithms are similar in nature, the DIT FFT algorithm is chosen to illustrate in this thesis.

Fixed-radix algorithms include the radix-2, radix-4, radix-8, *etc.* We will review the radix-2 and radix-4 DIT FFT algorithms and the general form, that is, radix-r DIT FFT algorithm in the following subsections. Among them, the radix-2 algorithm has the simplest form and is popular in FFT processor design. In this thesis, we implement radix-2 DIT FFT algorithms to explain our thought.

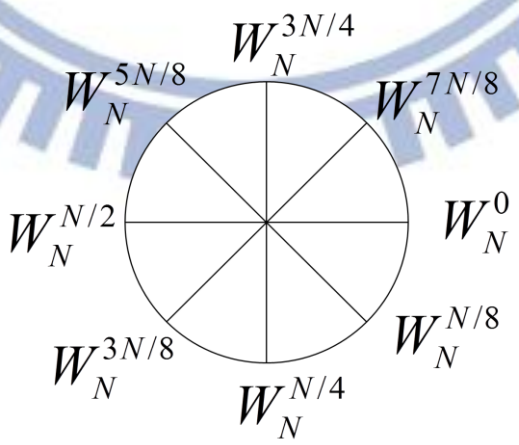


Fig. 2 Symmetric property of twiddle factor

2.1.2 Radix-2 DIT FFT Algorithm

The raidx-2 algorithm is using the divide-and-conquer approach which divides the problem of *N*-point FFT by a factor of 2, where *N* is power-of-2. Radix-2 DIT FFT Algorithm divides $x(n)$ into its even-numbered points and odd-numbered points, and uses $2r$ to substitute n for n is even, $2r+1$ to substitute n for n is odd.

$$\begin{aligned} X(k) &= \sum_{r=0}^{\frac{N}{2}-1} x(2r)W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)W_N^{(2r+1)k} \\ &= \sum_{r=0}^{\frac{N}{2}-1} x(2r)W_N^{2rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)W_N^{2rk} \end{aligned} \tag{2.3}$$

Since $W_N^2 = W_{N/2}$, (2.3) can be rewritten to (2.4).

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)W_{N/2}^{rk} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)W_{N/2}^{rk} \tag{2.4}$$

Each of the sums in (2.4) is recognized as an *N*/2-point DFT. As shown in Fig. 3, the first term is the *N*/2-point DFT of the even-numbered points of the original sequence, and the second term is the odd-numbered point of the original sequence.

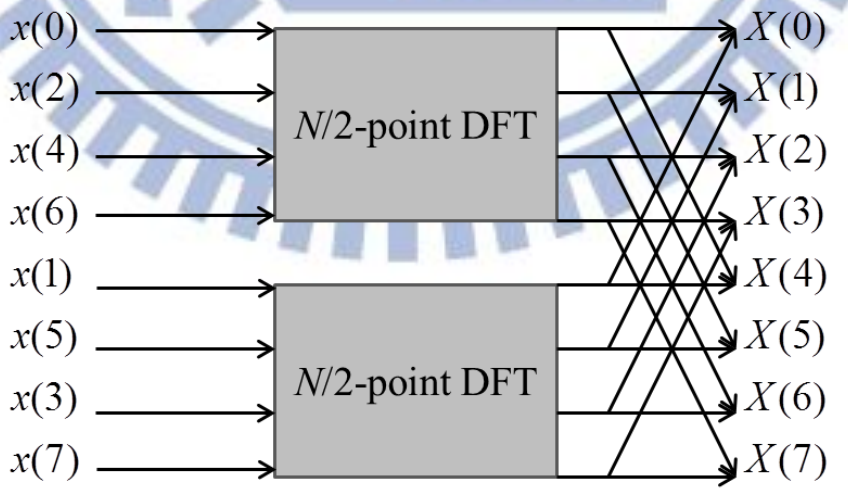


Fig. 3 First stage of the DIT FFT algorithm for 8-point FFT

Then we divide the original k into two new parts k and $k+\frac{N}{2}$, for k = 0, 1, …, $\frac{N}{2}-1$. Since $W_N^{N/2}=e^{-j\left(\frac{2\pi}{N}\right)\frac{N}{2}}=e^{-j\pi}=-1$ and $W_N^{r+N/2}=W_N^r\times W_N^{N/2}=-W_N^r$, (2.4) can be rewritten to (2.5). Through $\log_2 N$-time recursive decompositions, the complete radix-2 DIT FFT algorithm can be obtained.

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

$$X\left(k+\frac{N}{2}\right)=\sum_{r=0}^{\frac{N}{2}-1}\left[x(2r)-W_N^k x(2r+1)\right]\times W_{N/2}^{rk} \tag{2.5}$$

The decomposition can be mapped to a butterfly unit which is an essential arithmetic component in an FFT processor shown in Fig. 4. The butterfly unit operates complex multiplications, additions and subtractions. Fig. 5 illustrates the reorganized signal flow graph of 8-point radix-2 DIT algorithm. From (2.1), we know that the computational complexity of DFT is N^2. However, after the decompositions of FFT, the computational complexity of multiplications is $\frac{N}{2}\times(\log_2 N-1)$ and the complexity of additions and subtractions is $N\log_2 N$, which is much less than the original DFT equation.

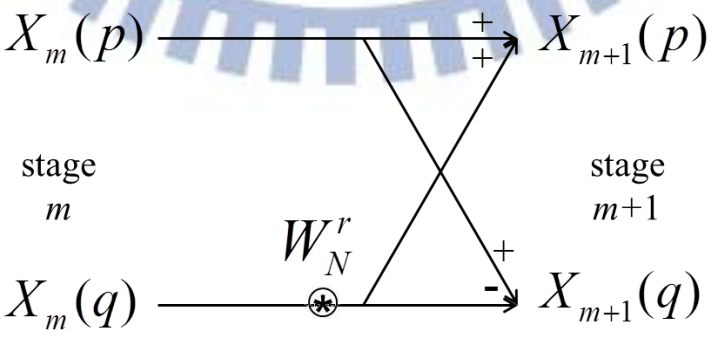


Fig. 4 The butterfly unit of a radix-2 DIT FFT algorithm

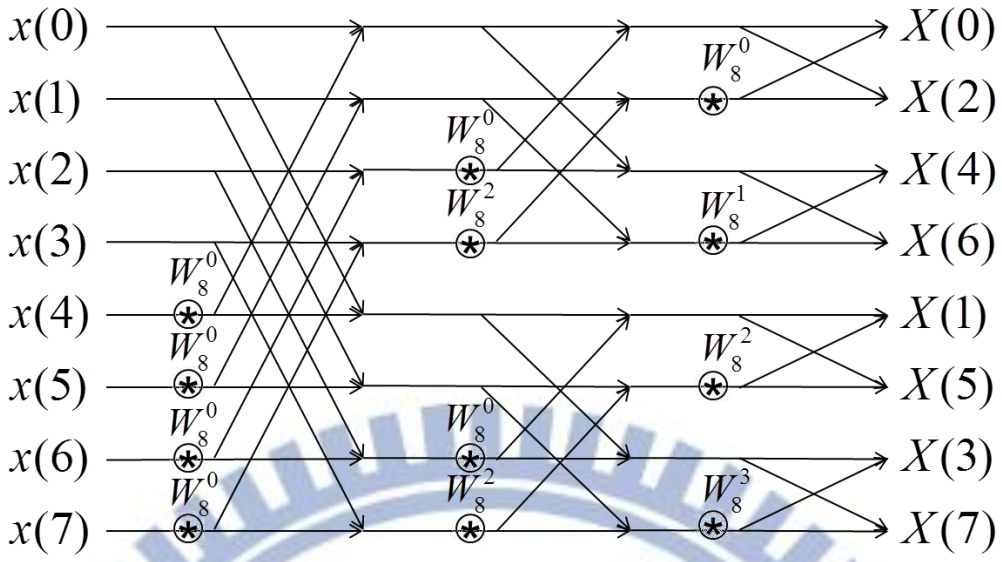


Fig. 5 The reorganized signal flow graph of the 8-point DIT FFT algorithm

2.1.3 Radix-4 DIT FFT Algorithm

Radix-4 FFT algorithm uses 4-point DFT to decompose *N*-point FFT. The original input sequence of radix-4 FFT algorithm is divided into four parts, $x(4r)$, $x(4r+1)$, $x(4r+2)$, and $x(4r+3)$, where $r = 0, 1, 2, \ldots, \frac{N}{4}-1$. Substituting these four subsequence for $x(n)$ into (2.1) and dividing $X(k)$ into four parts, we can get (2.6).

$$\begin{aligned}
X(k) &= \sum_{r=0}^{\frac{N}{4}-1}\left[x(4r)+W_N^k x(4r+1)+W_N^{2k}x(4r+2)+W_N^{3k}x(4r+3)\right]\times W_{N/4}^{rk}\\
X(k+\tfrac{N}{4}) &= \sum_{r=0}^{\frac{N}{4}-1}\left[x(4r)-jW_N^k x(4r+1)-W_N^{2k}x(4r+2)+jW_N^{3k}x(4r+3)\right]\times W_{N/4}^{rk}\\
X(k+\tfrac{N}{2}) &= \sum_{r=0}^{\frac{N}{4}-1}\left[x(4r)-W_N^k x(4r+1)+W_N^{2k}x(4r+2)-W_N^{3k}x(4r+3)\right]\times W_{N/4}^{rk}\\
X(k+\tfrac{3N}{4}) &= \sum_{r=0}^{\frac{N}{4}-1}\left[x(4r)+jW_N^k x(4r+1)-W_N^{2k}x(4r+2)-jW_N^{3k}x(4r+3)\right]\times W_{N/4}^{rk}
\end{aligned} \tag{2.6}$$

Although the complexity of multiplications in the radix-4 FFT algorithm is equal to $\frac{3}{4}N\times(\log_4 N-1)$, which is lower than radix-2, the complexity of additions and subtractions is still $N\log_2 N$, which is equal to that in the radix-2 FFT algorithm.

2.1.4 Radix-*r* DIT FFT Algorithm

Larger *r* can much further reduce the complexity of multiplications. For general cases, we derive the radix-*r* DIT FFT algorithm, where *r* is 2^S, and *S* is any positive integer. For *N*-point FFT, the general form is

$$X(k+\frac{qN}{r}) = \sum_{p=0}^{r-1} W_r^{pq} W_N^{pk} \sum_{n=0}^{N/r-1} X(rn+p) W_{N/r}^{nk} \tag{2.7}$$

where $q = 0, 1, \ldots, r\text{-}1$. And the complexity of multiplications is $\frac{(S-1)N}{S} \times (\log_S N - 1)$.

2.2 The FFT Architectures

The FFT is one of the most widely used DSP algorithms. Generally speaking, there are two kinds of popular FFT architectures to implement FFT algorithms. One is memory-based architectures and the other is pipeline-based architectures. Memory-based architectures are suitable for low throughput, low hardware cost, and long-size FFT designs whose size is not smaller than 512 [5]. On the other hand, pipeline-based architectures are suitable for high throughput, high hardware cost and short-size FFT designs. In this thesis, we focus on long-size FFT design with memory-based architecture. The details of those two architectures are introduced in the following subsections.

2.2.1 Memory-Based Architectures

The memory-based architecture is the simplest FFT architecture. It consists of one memory device and one radix-*r* processing element (PE) which contains one or few butterfly units to operate all computations in the signal flow graph. The basic components of memory-based architecture are shown in Fig. 6. Data are read from the memory and computed in the PE. After the computations, the output data are written back to the memory

and occupy the same storage locations as input data. For radix-2 FFT algorithm, there are $\log_2 N$ stages for the N-point FFT computations, and the number of butterflies in the PE can be chosen freely to meet the throughput rate requirement. The generalized conflict-free addressing schemes for memory-based FFT architectures presented in [19, 20] solve the problem of the memory bandwidth. In this work, we implement the memory-based FFT architecture with one butterfly unit in the PE block for simplification.

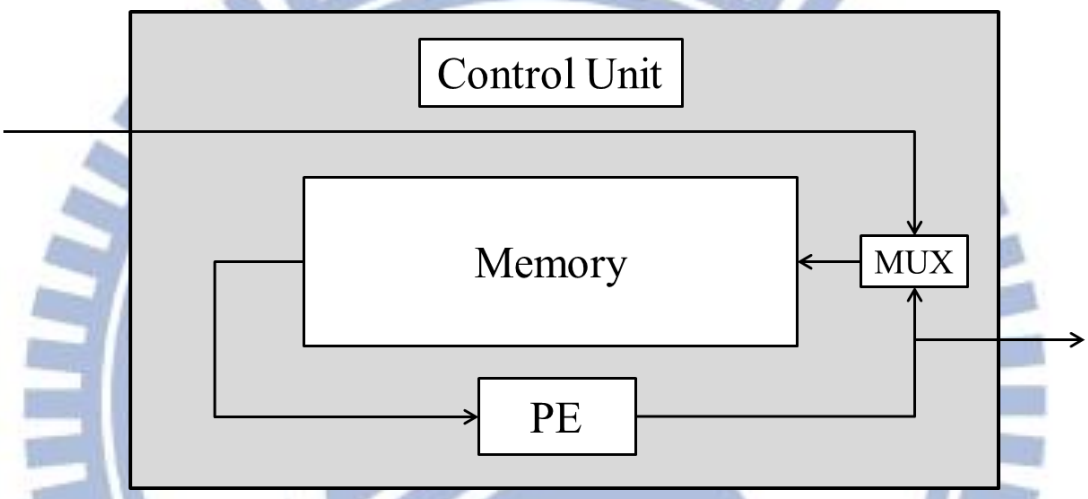


Fig. 6 A simple architecture of memory-based FFT

2.2.2 Pipeline-Based Architectures

The pipeline-based FFT architecture is regular, modular, local connection, and often adopted for high-throughput-rate applications with high hardware complexity [21]. It can be generally divided into two kinds of architectures depending on the design of register. One is the Single-path Delay Feedback (SDF) architecture [22, 23] and the other is the Multi-path Delay Commutator (MDC) architecture [24]. SDF architecture has higher hardware usage and lower hardware cost. On the other hand, MDC architecture has higher throughput than SDF architecture. Here we only introduce the radix-2 SDF architecture as below since we do not take the pipeline-based architecture into account in this work.

Take the radix-2 SDF (R2SDF) architecture as an example and the architecture is shown in Fig. 7. The R2SDF uses the registers efficiently by storing the output of the butterfly into the shift registers. When doing addition operation, the butterfly unit passes the output to the next stage. On the contrary, the butterfly unit stores the output into the shift registers when doing subtraction operation. Thus, there is only one output passes to the next stage in each cycle, and the utilization of the memory is 100%.

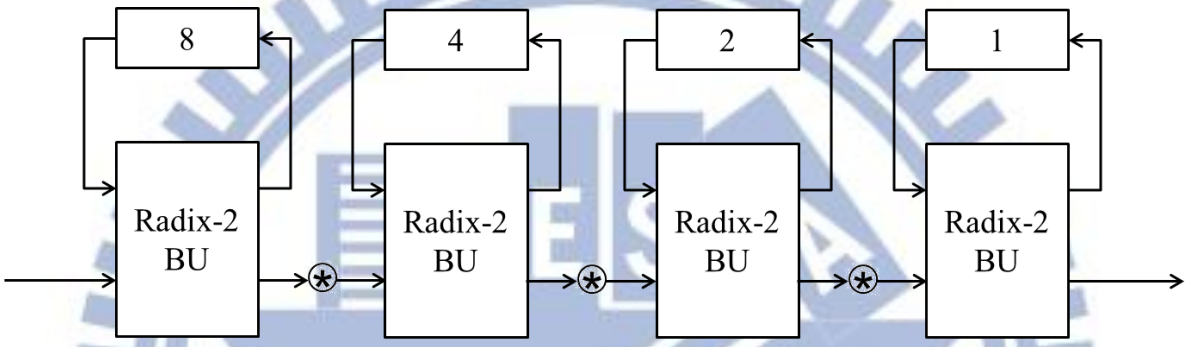


Fig. 7 The R2SDF architecture for 16-point pipeline-based FFT

2.3 Scaling Operation

Butterfly unit in PE shown in Fig. 4 operates on two complex numbers and produces two new complex numbers which replace the original ones in the sequence. Let $X_m(p)$ and $X_m(q)$ be the original complex numbers in tage m, the new pair $X_{m+1}(p)$ and $X_{m+1}(q)$ in stage m+1 are given as (2.8).

$$\begin{aligned} X_{m+1}(p) &= X_m(p) + X_m(q) \times W_N^{nk} \\ X_{m+1}(q) &= X_m(p) - X_m(q) \times W_N^{nk} \end{aligned} \tag{2.8}$$

The coefficient W_N^{nk} is the twiddle factor and substantially is the complex root of unity. For this reason, (2.9) shows that the multiplication with twiddle factor in the butterfly does not change the magnitude of the result. Hence, the magnitude of outputs can never be larger than twice the maximum magnitude of inputs as (2.10) explains.

$$\left|X_m(q)\times W_N^{nk}\right| = \left|X_m(q)\right| \times \left|W_N^{nk}\right| = \left|X_m(q)\right| \tag{2.9}$$

$$\left|X_{m+1}(p)\right| = \left|X_m(p) + X_m(q)\times W_N^{nk}\right| \le \left|X_m(p)\right| + \left|X_m(q)\right| \le 2\cdot\max\left\{\left|X_m(p)\right|, \left|X_m(q)\right|\right\} \tag{2.10}$$

From (2.10), we know that the range of data is increased from stage to stage. Thus, there is possibility of overflow during computation of butterflies if the data wordlength does not increase. Naturally, increasing the wordlength is a solution to avoid possible overflow [6]. However, the increased wordlength requires a larger storage to store the data which increases both area and power. Moreover, increasing wordlength is unacceptable for memory-based FFT architecture because the wordlength is fixed and cannot allow different wordlength from stage to stage. Therefore, it needs to scale the data for overflow prevention with the fixed-wordlength constraint.

Basically, the principle of scaling operation is dividing the data value by a factor of 2 before written back to the storage. That is, the data is shifted right by one bit after butterfly computations to avoid overflow as shown in Fig. 8.

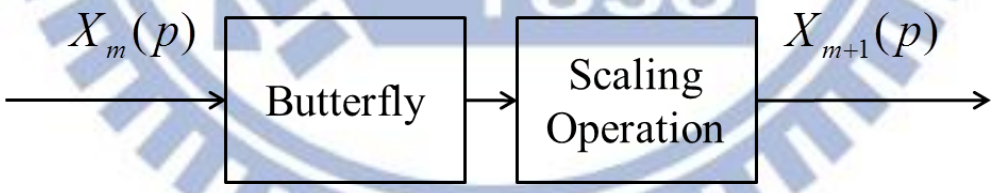


Fig. 8 An example of scaling operation

However, the truncation in scaling operation introduces noise and influences the accuracy. Longer wordlength will be needed to meet the required performance. As a result, the decision of whether to scale affects the accuracy very much. As shown in Fig. 9, a one-bit scaling flag is being defined to decide whether to truncate the least significant bit of the result or not. If the scaling flag is set to one, the result of the butterfly will be scaled. Otherwise, the result

keeps its original value without scaling. When the scaling operation is finished, results are written back to the memory device.

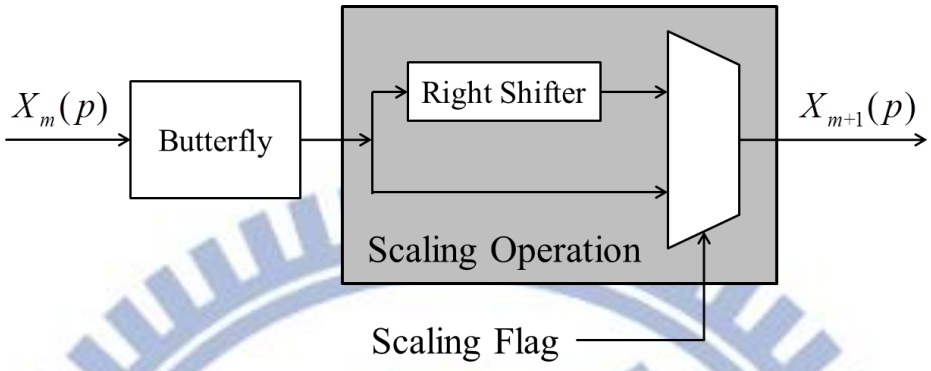


Fig. 9 An example of scaling operation with scaling flag

As we know, the wordlength affects the area and precision. Longer wordlength achieves higher precision but costs more chip area. The memory storage dominates the area of the FFT core as Fig. 10 shows. Besides, if long-size FFT is chosen, longer wordlength is needed to maintain the same required precision [8]. Therefore, it is demanded to design an efficient scaling algorithm in FFT which satisfy the SQNR requirement with shorter wordlength.

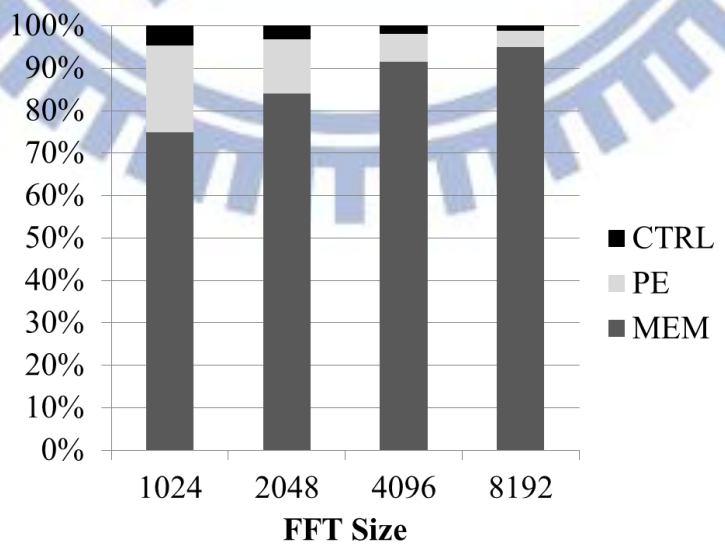


Fig. 10 The area occupancy of each component in memory-based 16-bit FFT

2.4 Scaling Method

The scaling method with a constant scaling flag of each stage is called static scaling method such as forced scaling [7] where the scaling flag is always set to one. Conversely, the scaling method determining the scaling factor of each stage at run-time is called dynamic scaling method, like BFP scaling [12] and conditional scaling [13, 14]. Each kind of scaling method has its advantages and disadvantages respectively. Static scaling is the simplest in hardware implementation as well as dynamic scaling greatly improves accuracy by ultimately avoiding unnecessary truncations. We will introduce these scaling schemes in fixed-wordlength FFT design as follows.

2.4.1 Forced Scaling

Oppenheim [7] proposed a scaling method which is the easiest to understand and the simplest to implement. The idea of this algorithm of preventing overflow is to scale the results after each butterfly for each stage, as shown in Fig. 11. That is, the scaling flag in each stage is set to one. This kind of scaling strategy is called forced scaling. In such case, the hardware is very simple to implement but the SQNR may not be very good. Actually, most data values need not to be scaled in every stage to avoid overflow. As a result, precision is unnecessarily lost by forced scaling method.

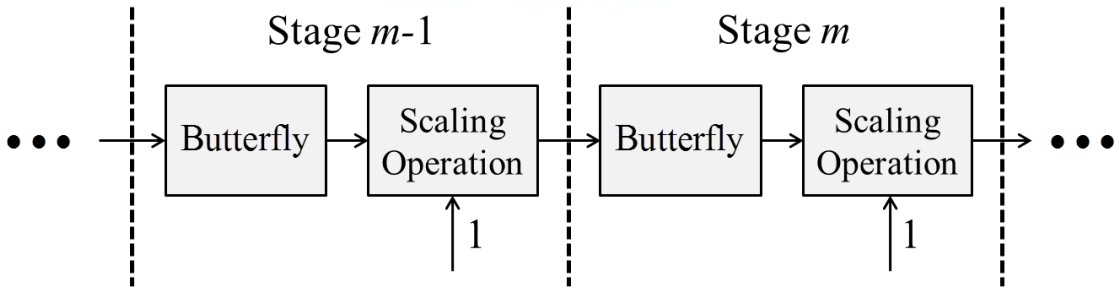


Fig. 11 The architecture of the forced scaling method

2.4.2 Block Floating Point Scaling

The floating point representation is shown in Fig. 12. The bit width of the floating point contains exponent bits, mantissa bits, and one sign bit. The magnitude of the value which is stored in the mantissa part is always smaller than one, and the value in the exponent part is an integer which is larger or equal to zero. BFP scaling method [12] uses a shared-exponent concept which groups the floating-point data into blocks with a common exponent to reduce the wordlength. An example of BFP with four blocks is shown in Fig. 13 where the capital letter "*E*" stands for the shared exponent of each block. The data only keep their own sign bit and mantissa bits, and each block has its own shared exponent as shown in Fig. 14. And the block size and the number of blocks are fixed through all stages.

Fig. 12 Floating point representation

Compared to the regular scaling such as forced scaling, BFP acquires higher performance of SQNR by avoiding unnecessary loss of data information. The idea is to scale only if we found the necessity of that. Thus, BFP employs intermediate buffers to store the output data of a certain block, as shown in Fig. 15. After all data in this block are computed and stored into the buffer, a detector will detect them to find out the largest value. If the magnitude of the largest value is larger than one, the scaling flag of this block is set to one. Then all the data are scaled and written back to the memory as the shared exponent is increased by one. On the other hand, if the largest value does not cause overflow, scaling will not be performed. Therefore, the least significant bit for each data value is preserved.

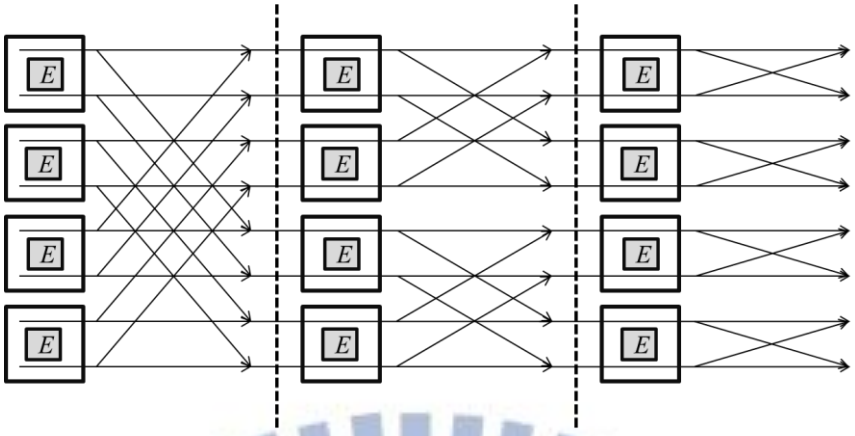


Fig. 13 An example of 8-point FFT with 4 blocks BFP scaling

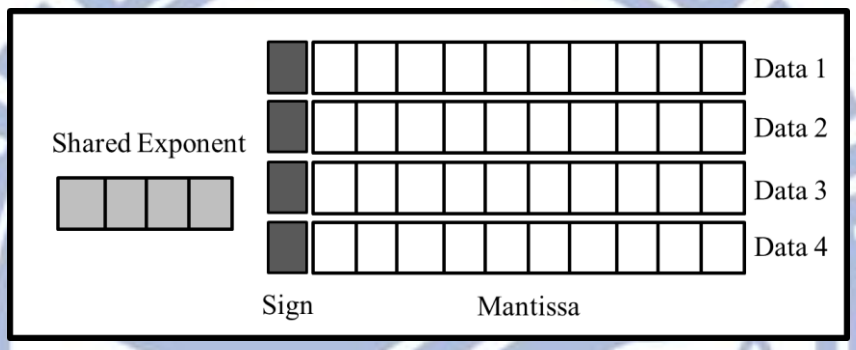


Fig. 14 A data block example with block size = 4

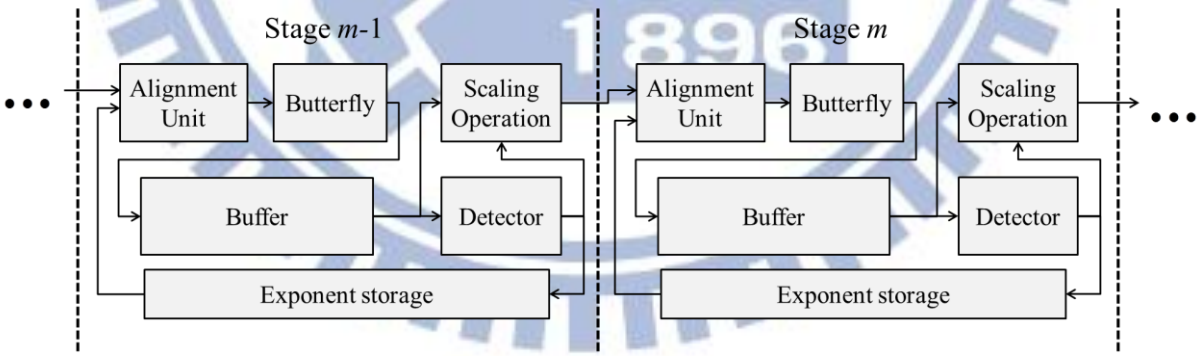


Fig. 15 The architecture of the BFP scaling method

Unlike the forced scaling, BFP scales data only when it is necessary. By adaptively determining the scaling flag to avoid accuracy loss, BFP acquires better precision than forced scaling method. Hence, BFP uses shorter wordlength to achieve the required SQNR and saves the area of memory device. Unfortunately, the inputs of the butterfly in BFP scaling method

will come from different data blocks, that is, their exponent may not be the same. Therefore, the floating point arithmetic of addition and subtraction operation needs an alignment unit to align two input data. It needs to shift the mantissa bits to represent them with the same exponent. Since the exponent bits have to be checked and mantissa bits have to be shifted, it will be more complicated than fixed point arithmetic in hardware implementation. Moreover, the buffer accesses and data detections introduce the additional processing latency and power consumption. Also, the intermediate buffers, detectors and the storage of shared exponents cause a large amount of additional area overhead.

2.4.3 Conditional Scaling

Instead of storing results in the buffers and detecting the largest value to decide the scaling flag, the conditional scaling method using the concept of prediction is another way to avoid overflow but saves the area overhead of buffers. Fig. 16 shows the architecture of conditional scaling method. In details, conditional scaling predetermines the scaling flag of current stage by the detections in previous stage. Then the detections in current stage will predetermine the scaling flag of next stage. Therefore, conditional scaling does not need buffers to store intermediate results. Moreover, there is only one shared-exponent in conditional scaling because it uses the fixed point representation. As a result, the alignment unit is unnecessary. We can directly operate additions and subtractions on the two inputs of butterfly.

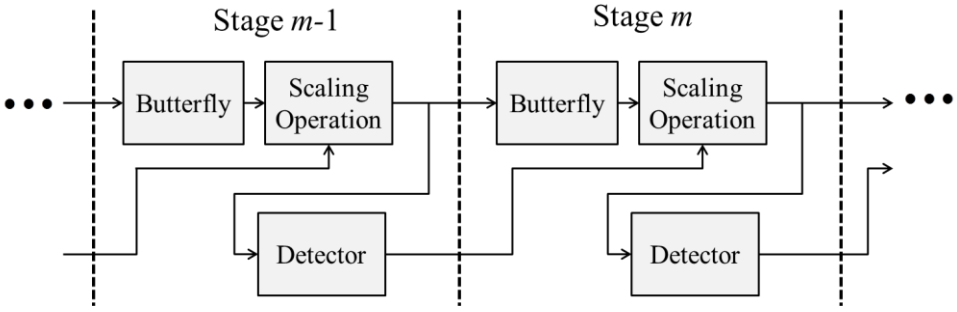


Fig. 16 The architecture of the conditional scaling method

The criterion for deciding the scaling flag of next stage is based on the observation of whether any value in current stage is outside a particular region on the complex plane [13]. Besides, (2.10) tells us that $|X_{m+1}(p)|$, $|X_{m+1}(q)| \le |X_m(p)| + |X_m(q)|$. If $|X_m(p)|$ and $|X_m(q)|$ are both smaller than 0.5, $|X_{m+1}(p)|$ and $|X_{m+1}(q)|$ will be smaller than one. Therefore, the region with which to compare the data in current stage is the circle of radius 0.5. As shown in Fig. 17, the circle of radius 0.5 is the idealized threshold for deciding the scaling flag of next stage.

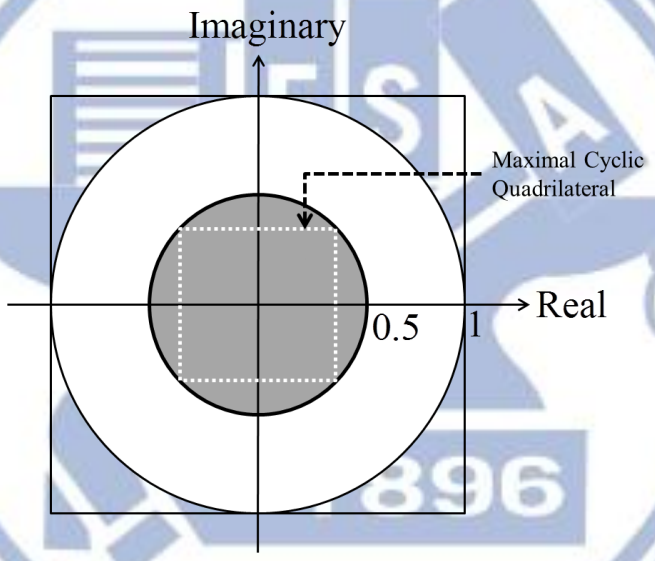


Fig. 17 The particular region of the complex plane in conditional scaling method

The magnitudes of the results are checked during the butterfly computations. If all output data in current stage are inside the region with radius 0.5, it guarantees that there will not be any data with magnitude larger than one to cause overflow in next stage. Thus, the scaling flag will not be set and the exponent will be kept. On the contrary, if there is at least one data outside the circular region with radius 0.5, it may cause overflow through butterfly computation in next stage. As a result, when computing the butterflies of next stage, the results should be scaled and the exponent should be increased by one.

Because conditional scaling predicts the necessity of scaling, the SQNR performance of conditional scaling is much higher than that of forced scaling but a little lower than that of BFP scaling. However, calculating the magnitude of a complex number needs to compute the square of real part and imaginary part. The required multipliers and adders will cost area. Alternatively, for hardware concern, the maximal cyclic quadrilateral which is the square with dotted line in Fig. 17 is chosen to define the particular region [14]. As a result, only comparators are needed to detect the region information of data.

In this thesis, we will combine the concepts of BFP scaling method and the conditional scaling method and utilize the profits of them.

Chapter 3
Motivation

To satisfy the required SQNR performance, we will choose a scaling approach which produces SQNR higher enough with less area. However, once the constraint is tighter and the original design does not satisfy the requirement, using longer wordlength is the only way to further increase the accuracy. Based on the experience of simulations, increasing wordlength by one will acquire about 6 dB improvement for SQNR but about 6% area penalty in addition. However, sometimes we do not have to increase SQNR so much to meet the constraint. Thus, by the improvements of conditional scaling and modifications of block floating point scaling, we will acquire SQNR improvement in demand with the corresponding area overhead.

3.1 Multi-Region Detection

With the approaches of [13, 14], the complex plane has been divided into two regions to detect the region information of the outputs of the butterfly. Traditional conditional scaling method avoids overflow in current stage by ensuring the data in previous stage are all in the internal region with radius 0.5. However, overflow comes from the addition and subtraction operations in butterfly which result in the growth of data magnitude. And the computation only has relations with the two input data. That is to say, restricting all data in the same region to avoid overflow is excessively severe. In order to avoid overflow, we only need to ensure that $|X_m(p)|+|X_m(q)|$ is smaller than one as (2.10) says.

We assume that we are now computing butterflies in stage m-1, and the complex plane is divided into two regions as Fig. 18(a) shows. The $X_m(q)$ is inside R_0 and $X_m(p)$ is outside R_0 and they are both the inputs of the same butterfly in stage m. In previous conditional scaling

method, it is judged that overflow will be produced in stage *m* and the scaling flag of stage *m* will be set. However, overflow can also be avoided as long as $|X_m(q)|$ is small enough. Therefore, we try to divide the complex plane into more regions. Fig. 18(b) shows the idea of which four regions are divided. In such case, $X_m(p)$ is inside R_{+1} and $X_m(q)$ is inside R_{-1} where the radius of R_{+1} is 0.7 and the radius of R_{-1} is 0.2. By ensuring the summation of the radii of R_{+1} and R_{-1} is less than unity, we can judge that the butterfly computation in stage *m* which operates on these two data will not cause overflow. That is to say, dividing the complex plane into more regions will further prevent the unnecessary scaling operations and produce better SQNR performance. And we can expect that the more regions the complex plane is divided, the higher precision can be obtained.

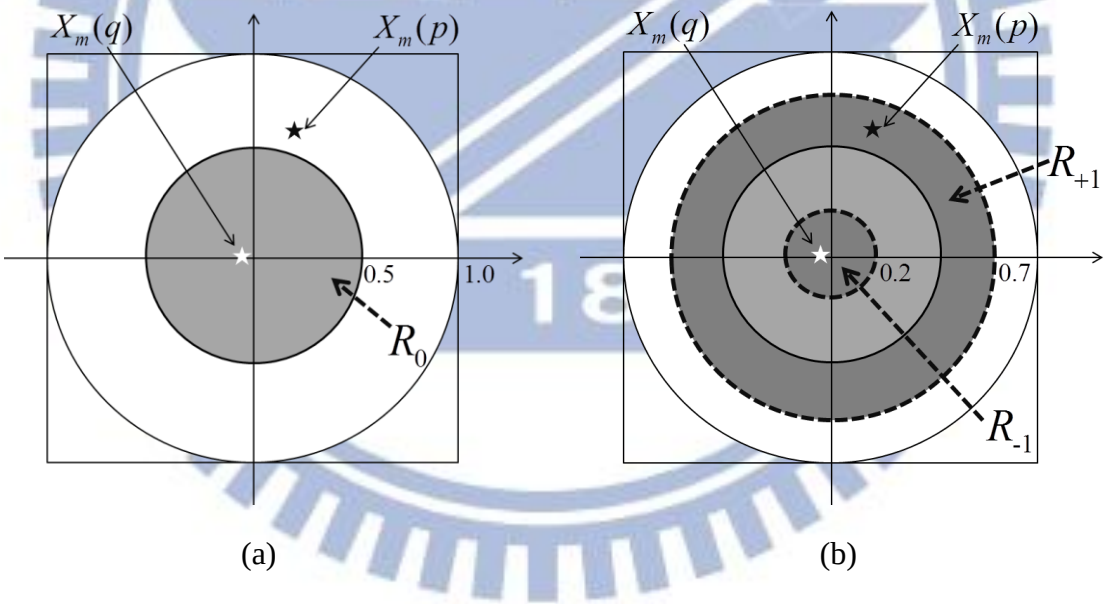


Fig. 18 The complex plane with (a) two regions (b) four regions are divided

3.2 Convergent Block Scaling

The hardware of floating point arithmetic is more complicated than that of fixed point arithmetic. As mentioned in 2.4.2, one part of the area overhead of BFP scaling method is the alignment unit because the floating point representation is used. Since the inputs of butterfly

may come from different blocks and their exponent may be different, we cannot operate these two mantissas directly without alignment. However, figuring out the larger exponent and shifting the smaller mantissa introduces area and processing latency. If we want to save the hardware of alignment unit, we must ensure that the two inputs of butterfly are come from the same block which means their exponent is always the same one.

It can be observed in Fig. 5 that during the decomposition of FFT algorithms, a *k*-point DFT in stage *m* will be separated to two *k*/2-point DFT in stage *m*+1. And the computation of the first *k*/2 data in stage *m*+1 only depends on the first *k*/2 data in stage *m*. Thus we group the data into blocks in a convergent way mentioned in [25] and the idea is shown in Fig. 19. In first stage, all data are grouped into one block. That is, the number of blocks and shared exponents in first stage are both equal to one. Afterwards, the number of blocks and shared exponents are doubled as the size of the block is one half from stage to stage. In this way, inputs of butterfly for each stage are surely come from the same block with the same exponent.

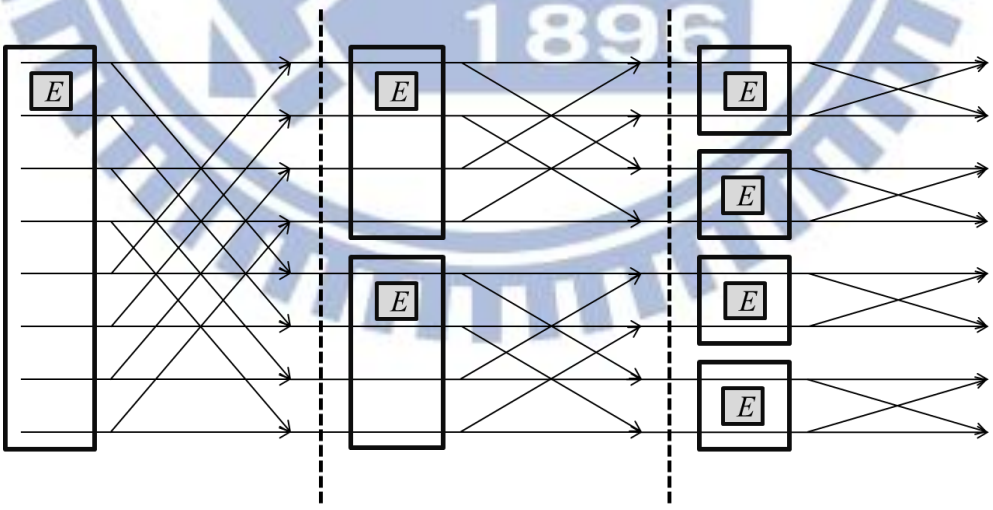


Fig. 19 An example of 8-point FFT with convergent block scaling

Through the convergent block scheme, the data will be represented in different dynamic ranges with different exponents so the SQNR is higher than forced scaling scheme where the data are all in the same dynamic range. Besides, the number of blocks is a key factor for SQNR improvement. Larger number of blocks results in better SQNR performance. However, such kinds of block scaling methods require the additional area of storage to store the shared exponents. By the way, since the conditional scaling is assumed that the data are all in the same dynamic range with the same exponent, the convergent block scheme is naturally suitable for implementation with conditional scaling in fixed point representation.

3.3 Our Strategy

We are informed that the SQNR performance can be improved by two ways. One is the multi-region conditional scaling and the other is the convergent block scaling. Therefore, we propose the *multi-region conditional block scaling* (*MRCBS*) method which combines these two methods mentioned above to obtain many solutions of hardware architecture for SQNR improvement. As a result, by searching those solutions, we can figure out the solution which has the minimum area cost with the required SQNR performance.

3.4 Problem Formulation

Given FFT size and required SQNR, our goal is to minimize the area of memory-based radix-2 FFT under the given SQNR constraint by applying our MRCBS method.

Chapter 4
The Proposed Approach

In this chapter, we present the proposed MRCBS method for memory-based FFT which utilizes the profits of conditional scaling and the convergent block scaling to improve SQNR performance. The first section describes the scheduling of the butterfly computation in order to predict overflow precisely and save the additional storage. The second section illustrates the MRCBS and its architecture. Finally, in the third section, we will discuss the MRCBS with different number of blocks and the relationship between the number of blocks and the performance of area and SQNR. MRCBS generates many solutions for improving SQNR, and the purpose of this thesis is to find out the architecture of scaling method for FFT which meets the SQNR requirement and has the smallest area.

4.1 Scheduling of Butterfly Computation

In order to precisely predict the overflow and prevent the unnecessary scaling, we should detect the magnitude of the two data which are the inputs of the same butterfly in next stage. As Fig. 20 shows where BU is abbreviated from butterfly unit, BU_1 and BU_2 are in current stage and other two butterflies BU_3 and BU_4 are in next stage. In such case, X_0 and X_1 should be detected overflow together because they are both the inputs of BU_3. However, X_0 is computed by BU_1 and X_1 is computed by BU_2. We cannot get them at the same time. Therefore, when we get the results of BU_1, we have to store them and wait for the results of BU_2. For this reason, we will schedule the computational order of butterfly computations to save the required storage.

The concept of the scheduling is that when the computation of BU_1 is finished, the computation of BU_2 is followed. After BU_1 and BU_2 are finished, X_0 and X_1 are both available,

we can predict overflow and determine the scaling flag for BU_3. Fortunately, X_2 and X_3 are both available as well. We can predict overflow for BU_3 and BU_4 simultaneously. That is, while two butterflies are finished in current stage, we can predict two butterflies in next stage smoothly. As a result, only four registers are required to store the results of BU_1 and BU_2 for overflow predictions. When the predictions of BU_3 and BU_4 are finished and the scaling flags are determined, those four registers can be reset for storing the results of other butterflies. Furthermore, compared to the original order, just small extra control circuits are required to schedule the computation of butterflies as we wish.

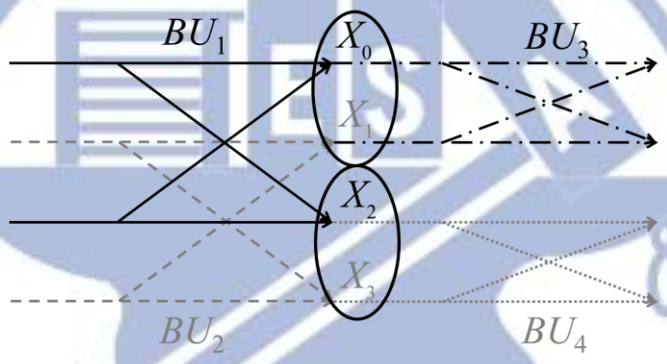


Fig. 20 Detection of the two data in the same butterfly of next stage

4.2 Multi-Region Conditional Block Scaling

Since the thought of conditional scaling is to predict the overflow and predetermine the scaling flag for next stage, it does not need intermediate buffers to store the output data to determine the scaling flag of current stage. Therefore, we develop the architecture for our scaling method which is shown in Fig. 21. The memory block is the original part of the traditional memory-based FFT architecture shown in Fig. 6 and there is one butterfly unit in the PE block in our work. The detector is to detect the region information and the predictor is to predict possible overflow. The shared exponents and the scaling flags of each block are stored in the exponent array.

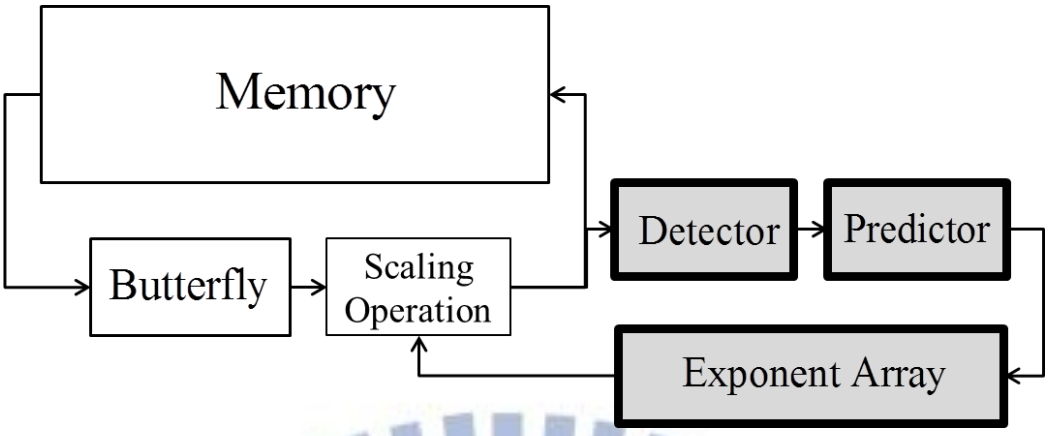


Fig. 21 The architecture of the proposed MRCBS

When evaluating the FFT, two data are read from the memory for each cycle and computed in BU. The scaling flag predetermined in previous stage will be read from exponent array to scale the results of butterfly in current stage. After computation of the butterfly is finished, the results will be straightly written back to the memory. In the meanwhile, the results are passed to the detector to define their region information by detecting their magnitudes. Then the predictor receives the region information of the results from the detector to judge whether overflow will occur in next stage or not. After the prediction is finished, the predetermined scaling flags and the shared exponents of next stage will be stored into the exponent array. Moreover, the detector and predictor are worked in parallel with the computation of butterfly unit since the results of butterfly can be written back to the memory without waiting for the results of them. Thus, such kind of architecture will not produce large amount of processing latency. The details of detector, predictor, and exponent array will be described in the following subsections.

4.2.1 Region Detector

Because we divide the complex plane into many regions, the overflow detector consists of comparators in order to determine the region information of the data by comparing the outputs of butterfly with several thresholds. The detector dividing the complex plane into many circular regions with different radii is called circular-type detector. On the other hand, dividing the complex plane into many square regions with different side lengths is called square-type detector. Because the square region is the maximal cyclic quadrilateral of each circular region, the area is smaller and the prediction is severer. As a result, the square-type detector improves less SQNR than the circular-type one but increases less area.

The purpose of the multi-region detection is to handle the situation shown in Fig. 18 where $X_m(p)$ is outside the internal region but $X_m(q)$ is deeply inside and they are actually overflow-free in next stage. Therefore, we should define an additional pair of regions that one region is larger and the other is smaller. As a result, the case with larger $X_m(p)$ and smaller $X_m(q)$ or vice versa will possibly be judged to be overflow-free. And that is why we divide the complex plane into even number of regions. In our work, we divide the complex plane into two regions, four regions, and six regions and implement circular-type and square-type detectors respectively. That is, there are six different detectors in total with different area overhead and different SQNR performance.

After the detection of the detector is finished, the region information which indicates the region where the data is located will be output.

4.2.1.1 Circular-Type Detector

First we discuss the region detector which divides the complex plane into two regions as Fig. 22(a) shows. This type of detector is named "*C2*". The internal region R_0 is defined as the circle of radius 0.5 and the threshold t_0 representing the radius of R_0 is equal to 0.5.

Next we divide the complex plane into four regions. This type of detector is named "*C4*". Additional regions R_{-1} and R_{+1} are defined as shown in Fig. 22(b). The R_{-1} is the circle of radius t_{-1} and R_{+1} is the annulus with inner radius t_0 and outer radius t_{+1}, and we have to ensure that t_{-1} plus t_{+1} is less than one. Because the threshold t_{-1} is absolutely larger than the magnitude of $X_m(q)$ and t_{+1} is larger than that of $X_m(p)$, the addition and subtraction operations of those two complex data will not be larger than one to cause overflow. Therefore, once $X_m(p)$ is outside R_0 but is inside R_{+1} while $X_m(q)$ is inside R_{-1}, it will be judged that the butterfly computing X_m(p) and $X_m(q)$ in next stage is overflow-free.

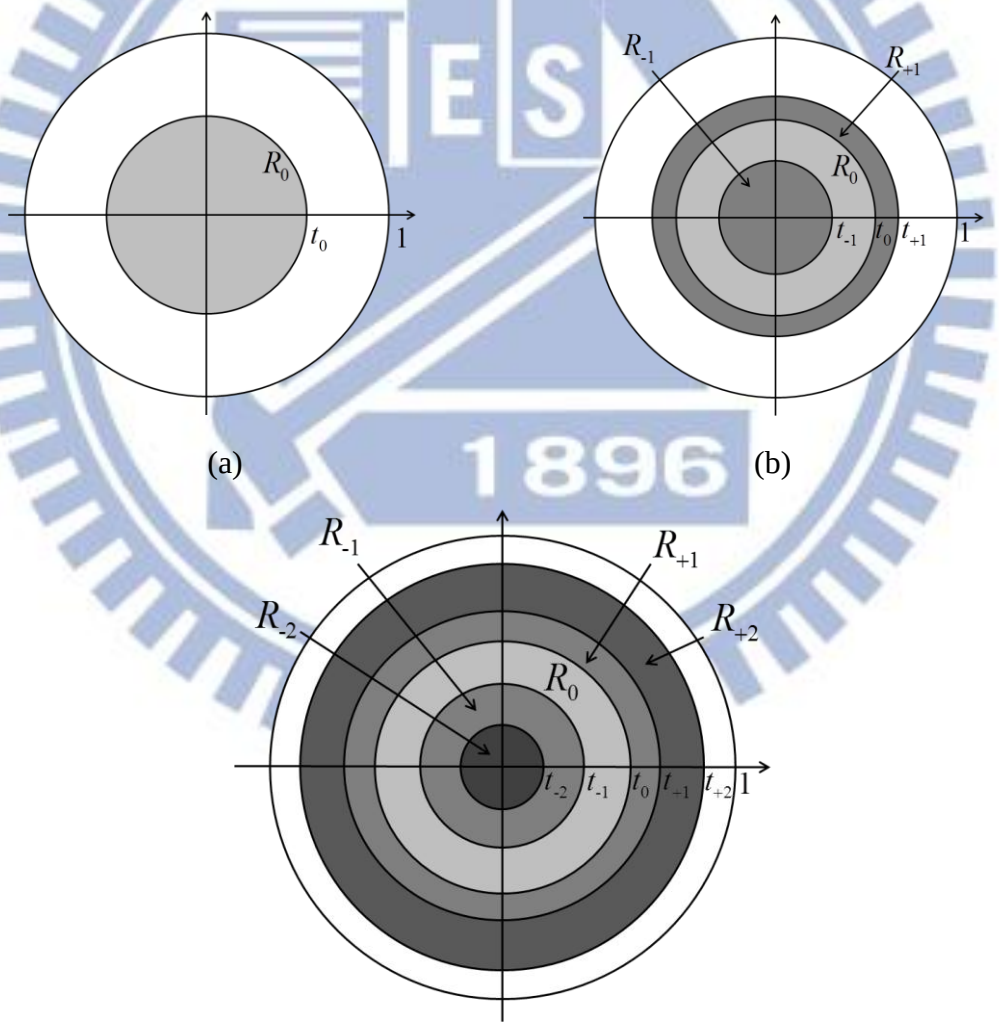


Fig. 22 The regions of the circular-type detectors (a) *C2* (b) *C4* (c) *C6*

Finally we further divide the complex plane into six regions as shown in Fig. 22(c) and this type of detector is named "*C6*". With the existed four regions on the complex plane in the *C*4 detector, the regions R_{-2} and R_{+2} are defined additionally. In *C*6 detector, R_{-2} is the circle of radius t_{-2}, R_{+2} is the annulus with inner radius t_{+1} and outer radius t_{+2}, and R_{-1} becomes the annulus with inner radius t_{-2} and outer radius t_{-1}. For the same idea in *C*4, the summation of t_{-2} and t_{+2} should also be less than one.

As mentioned above, we have known that t_{-k} plus t_k where k = 1 or 2 should be less than one to avoid overflow. And if t_{-k} is larger, t_k will become smaller. In the meanwhile, the area of R_{-k} becomes larger as the area of R_k becomes smaller. Since our purpose is to avoid the unnecessary scaling as accurate as possible, the area of the two regions should be larger and the possibility of data in R_{-k} should be equal to the possibility of data in R_k. As a result, we have two conditions as (4.1) and (4.2) to determine the thresholds t_k in detectors. And the results of thresholds are shown in Table 1.

$$t_{-k} + t_k = 1 \tag{4.1}$$

$$Area(R_{-k}) \approx Area(R_k) \tag{4.2}$$

Type	t_{-2}	t_{-1}	t_0	t_1	t_2
C2			0.5		
C4		0.375	0.5	0.625	
C6	0.305	0.375	0.5	0.625	0.695

Table 1 The value of the thresholds in circular-type detectors

Here we sweep t_{-1} from 0 to 0.5 to simulate the SQNR performance and the result is shown in Fig. 23. As we can see, SQNR is almost the highest when t_{-1} is equal to 0.375 and t_1 is equal to 0.625 as we expect.

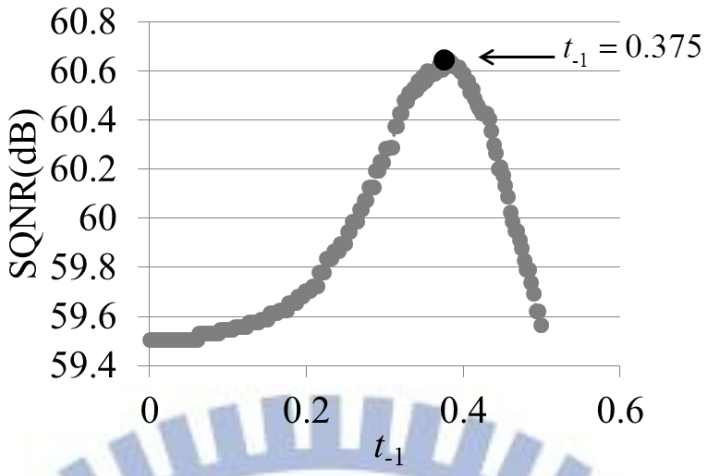


Fig. 23 The simulation result of SQNR with different t_{-1}

Because the regions are all circles in the complex plane, we are required to calculate the magnitude of the complex data by computing its summation of the square of the real part and the imaginary part. As a result, multipliers, adders, and comparators are introduced which are required to compare the thresholds as shown in Fig. 24. It is intuitive that *C6* has the best performance and the largest area of comparators since there are six thresholds to be compared while *C2* has the smallest area of comparators and the performance is relatively worst.

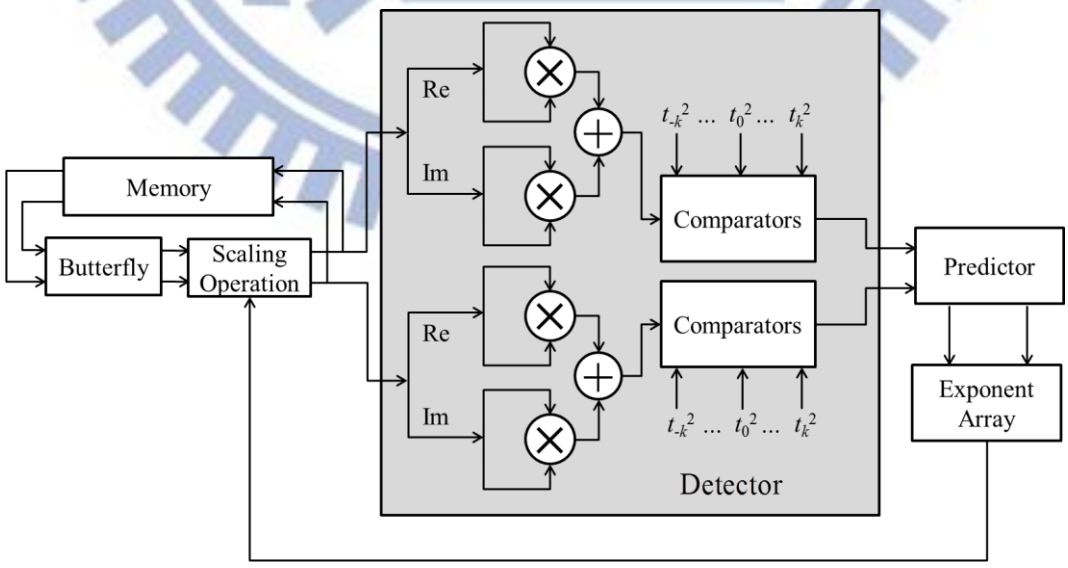


Fig. 24 The block diagram of the circular-type detector

However, the bit width *BW* of multipliers, adders and comparators influences the area and the accuracy as well. That is to say, the arithmetic unit with longer bit width will produces better accuracy and cost more area. In our work, we implement 10-bit comparators, *BW*-bit multipliers, and 2**BW*-bit adders where *BW* is an integer and can be chosen from 5 to 10.

4.2.1.2 Square-Type Detector

Although the circular-type region detectors make precise predictions, they cost a lot of area for introducing the multipliers and adders. For hardware concern, there are alternative ways which are the square-type region detectors [14]. That is, we can simplify those circular regions to their maximal cyclic quadrilaterals. The square regions are described in Fig. 25. As the circular-type detectors, "*S2*" is the square-type detector which divides the complex plane into two square regions and "*S4*" is the detector dividing the complex plane into four square regions. The detector dividing the complex plane into six squares is therefore named "*S6*".

Because each square region shown in Fig. 25 is the maximal cyclic quadrilateral of the circular region shown in Fig. 22, the thresholds in square-type detectors will be defined as (4.3) where k = -2, -1, 0, 1, and 2. And the thresholds h_k of the square-type detectors are listed in Table 2.

$$h_k = t_k \times \sqrt{2}/2 \tag{4.3}$$

Type	h_{-2}	h_{-1}	h_0	h_1	h_2
S2			0.354		
S4		0.265	0.354	0.442	
S6	0.215	0.265	0.354	0.442	0.492

Table 2 The value of the thresholds in square-type detectors

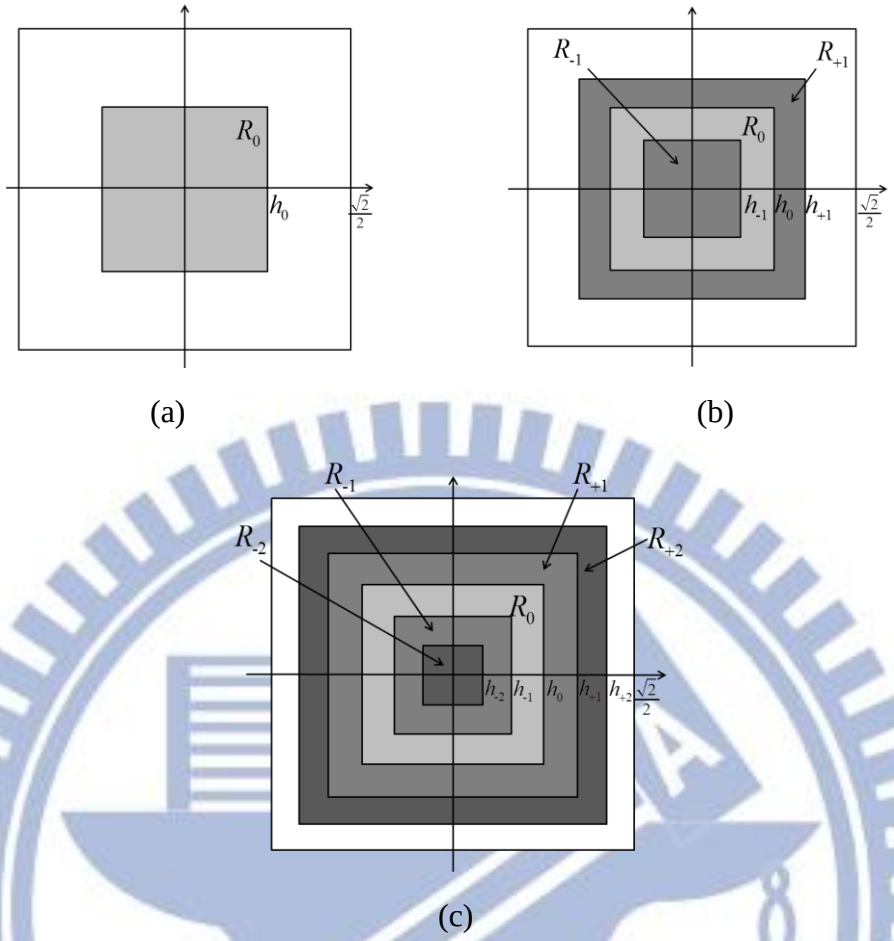


Fig. 25 The regions of the square-type detectors (a) *S2* (b) *S4* (c) *S6*

Also we sweep h_{-1} from 0 to 0.354 to simulate the SQNR performance of *S4*-type detector. And the result is shown in Fig. 26. The SQNR is almost the highest when h_{-1} is equal to 0.265 and h_1 is equal to 0.442 as we expect.

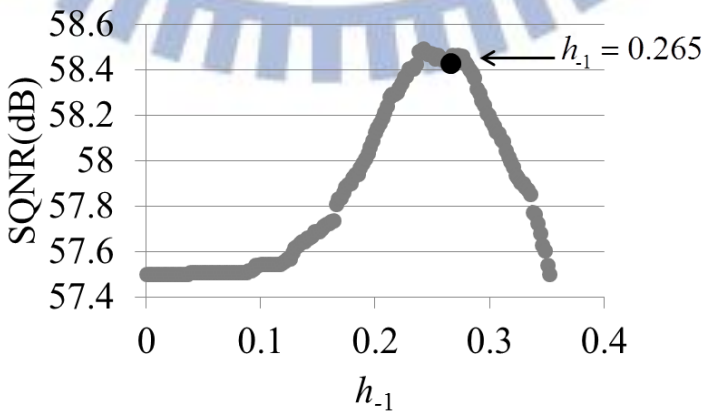


Fig. 26 The simulation result of SQNR with different h_{-1}

To detect the region information for the data in square-type detectors, we only need to compare the absolute value of the real part and imaginary part with the half of the side lengths of those squares. The block diagram of square-type detector is shown in Fig. 27. The only difference between circular type and square type is that square type does not need the multipliers and adders to calculate the magnitude. As a result, the circuits of the square-type detectors are much simpler than the circuits of the circular-type detectors. However, the bit width of the comparators influences the accuracy as we have mentioned. Thus, in our work we implement *BW*-bit comparators where *BW* is an integer and can be chosen from 5 to 10.

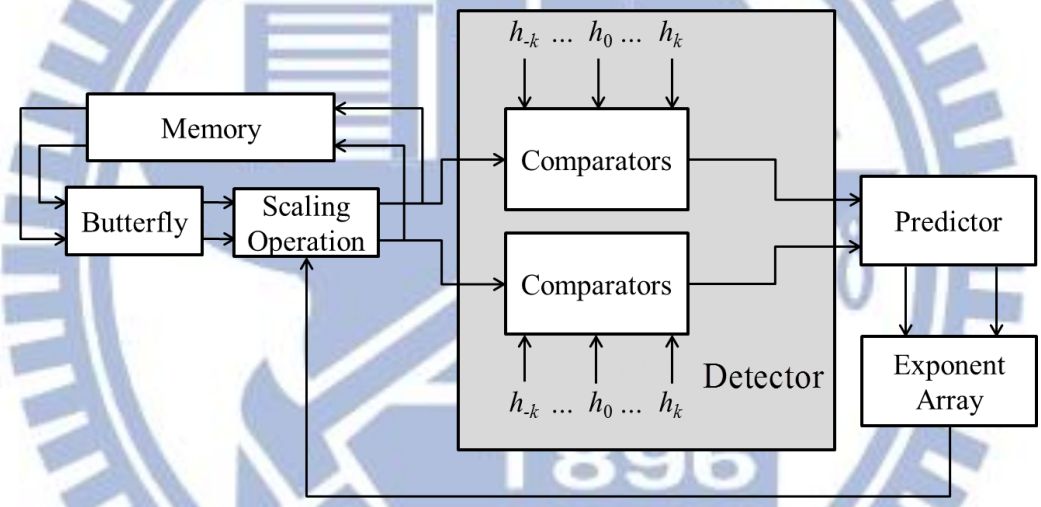


Fig. 27 The block diagram of the square-type detector

4.2.2 Overflow Predictor

With the region information of the data come from the region detector, we will predict overflow of the butterflies of next stage. As shown in Fig. 28, X_0 and X_2 are computed by BU_1 as X_1 and X_3 are computed by BU_2. After the computations of BU_1 and BU_2 are finished, we will get the four results from X_0 to X_3. Then we will predict whether X_4 to X_7 may cause overflow or not. Here we define two variables P and Q to represent the region information. For the prediction of BU_3, P_{BU3} is the region information of X_0 and Q_{BU3} is the region

information of X_1. And for the prediction of BU_4, P_{BU4} is the region information of X_2 and Q_{BU4} is the region information of X_3. The values of P and Q are decided according to the data locations of X_0 to X_3. The region information is equal to k as the data is inside the region R_k where k = -2, -1, 0, 1, and 2 as Table 3 shows.

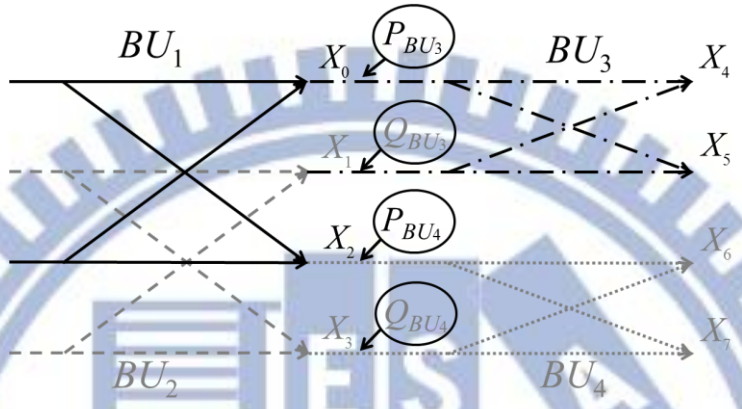


Fig. 28 Overflow Prediction based on the region information of the inputs

	R_{-2}	R_{-1}	R_0	R_1	R_2	
P, Q	-2	-1	0	1	2	3

Table 3 The value of region information P and Q according to the data locations

Taking the prediction of BU_3 with the *C6*-type detector as an example, P_{BU3} is set to -2 while X_0 is inside the region R_{-2} and Q_{BU3} is set to 2 while X_1 is inside the region R_{+2}. As we know, X_4 and X_5 will cause overflow if the summation of the magnitude of X_0 and X_1 is larger than one. As a result, we will sum up the variable P_{BU3} and Q_{BU3} and compare to a constant zero. If the result of P_{BU3} plus Q_{BU3} is larger than 0, it implies that the magnitude of X_0 plus the magnitude of X_1 is larger than one and the outputs of the BU_3 should be scaled to avoid overflow. Table 4 shows the decisions of scaling which are based on the result of the summation of P and Q.

Q \ P	R_{-2}	R_{-1}	R_0	R_1	R_2	
R_{-2}	-4	-3	-2	-1	0	1
R_{-1}	-3	-2	-1	0	1	2
R_0	-2	-1	0	1	2	3
R_1	-1	0	1	2	3	4
R_2	0	1	2	3	4	5
	1	2	3	4	5	6

No Scaling

Scaling

Table 4 Scaling decision according to the summation of P and Q

The block diagram of the predictor is shown in Fig. 29. There are four registers to temporarily store the region information. After the computation of BU_1 in Fig. 28 is finished, we store P_{BU3} and P_{BU4} and wait for the results of BU_2. After BU_2 is finished, we will get Q_{BU3} and Q_{BU4} and store them into the registers. While the four variables are getting ready, we will calculate P_{BU3} plus Q_{BU3} and P_{BU4} plus Q_{BU4} and then compare the results to zero.

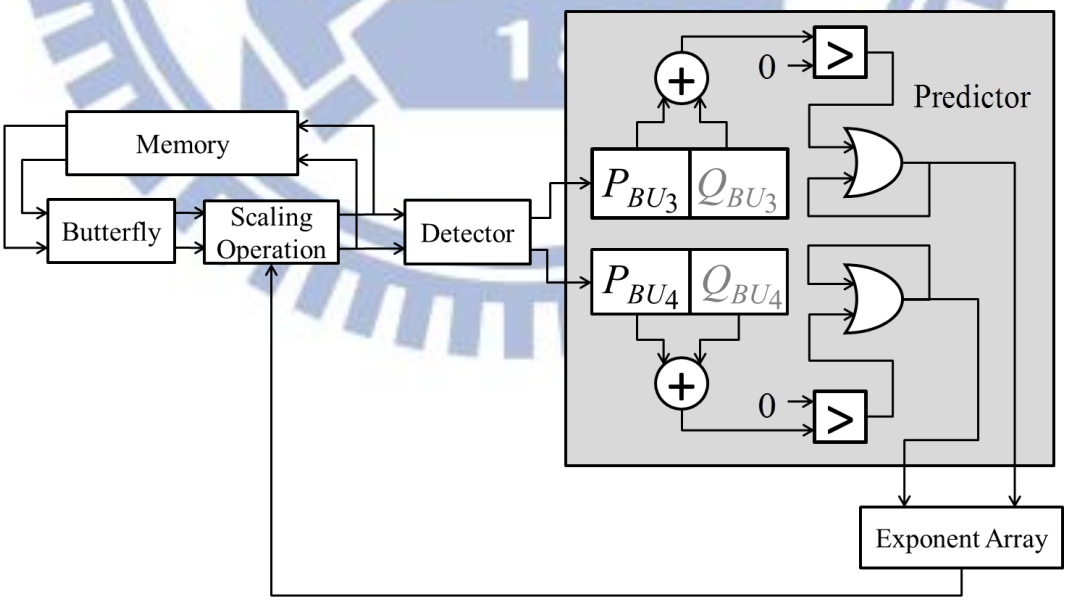


Fig. 29 The block diagram of the overflow predictor

Besides, there are two special flags in the predictor which memorize the scaling flags in next stage. It is because that the convergent block scaling method will separate the data block in current stage to two smaller blocks in next stage. Once the result of P plus Q in the new smaller blocks is larger than zero, the special flag will be set and held. After the computations of the data in a certain block are all finished, the two flags will determine the scaling flags of the new two blocks and will be stored in the exponent array.

4.2.3 Exponent Unit

The block scaling method needs exponent units to store the shared exponents and the scaling flags of the blocks. As shown in Fig. 30, the exponent units are stored in the exponent array. Each exponent unit consists of a k-bit shared exponent and a one-bit scaling flag where k is depending on the FFT size N and is equal to $\lceil \log_2 \log_2 N \rceil$.

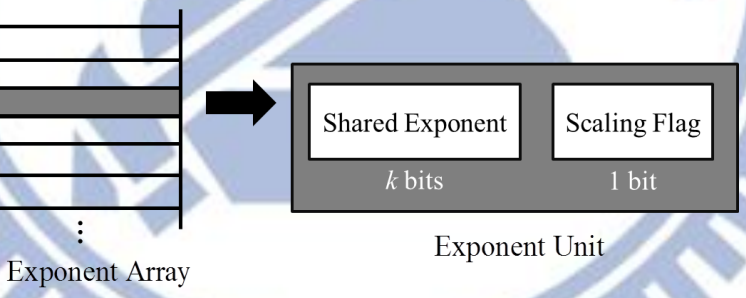


Fig. 30 The exponent array with the exponent unit

The shared exponent is shared for all data of a certain block, and the scaling flag is to decide whether to scale the results of the butterfly when the data in this block are being computed. After all computations in one block are finished, the two new scaling flags and shared exponents will be stored in the corresponding exponent units as shown in Fig. 31. If the flag is set, the shared exponent will be increased by one. Otherwise, if the flag is unset, the shared exponent will keep its original value.

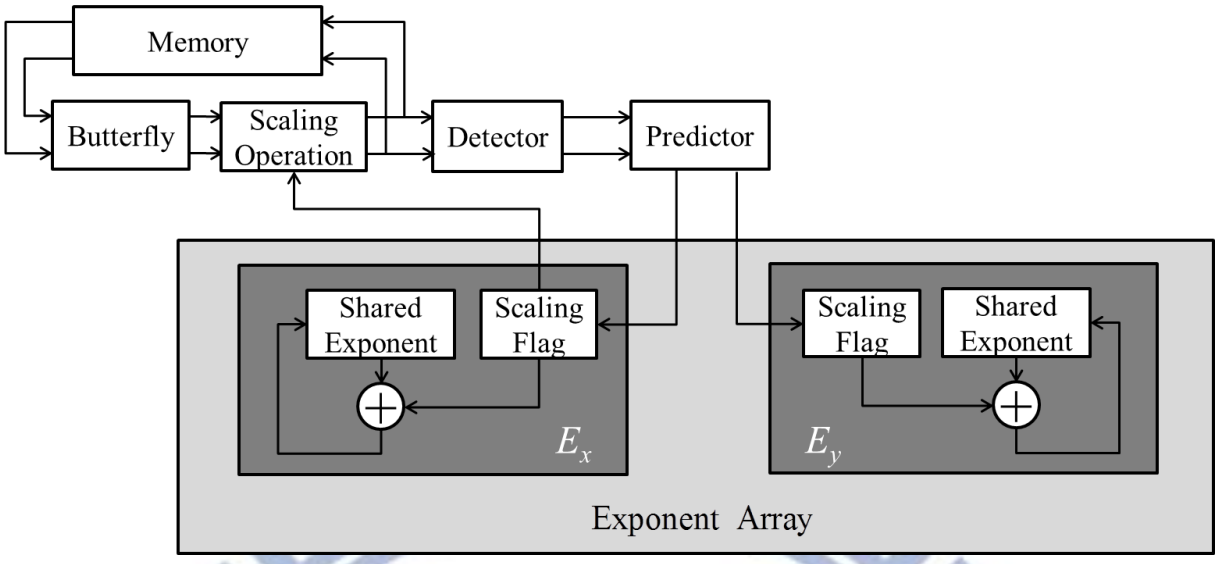


Fig. 31 The block diagram of the exponent unit

As we know, each block has its own exponent unit. Here we define B_n as the tag of the block and E_n as the tag of the corresponding exponent unit where n is an integer. If there are m blocks, n is from 0 to m-1. Besides, during the computations of the convergent block scaling, the block in current stage will be divided into two blocks in next stage. Therefore, after the computations of the block B_x in stage s are finished, the new two shared exponents and scaling flags will be stored in the exponent units E_x and E_y where $y = x + m / 2^s$. The usage of the exponent array for each stage is shown in Fig. 32.

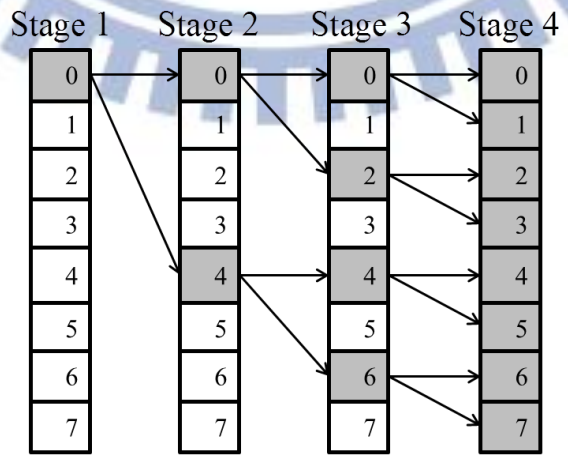


Fig. 32 The usage of the exponent array for convergent block scaling with 8 blocks

4.3 Restricted Number of Blocks

As the convergent block scaling method we have mentioned, the dynamic scaling method only scales when it is necessary to avoid the loss of accuracy. And the concept of grouping data into several blocks improves the SQNR since there are lots of exponents to represent the data with different dynamic range. Therefore, it is easy to expect that the larger number of blocks will acquire higher precision. However, the convergent block scaling method will divide one block into two blocks from the first stage to the last stage. That is, the number of blocks and the area of the exponent storage will be doubled through one stage. For an *N*-point FFT, there will be *N*/2 blocks in the last stage and *N*/2 exponent units are required. As a result, it will cost a lot amount of storage. Therefore, we define $B_{max} = 2^{s-1}$ which is the total number of blocks in convergent block scaling and the number of blocks is doubled until the stage *s*. Fig. 33 shows the convergent block scaling with different B_{max}.

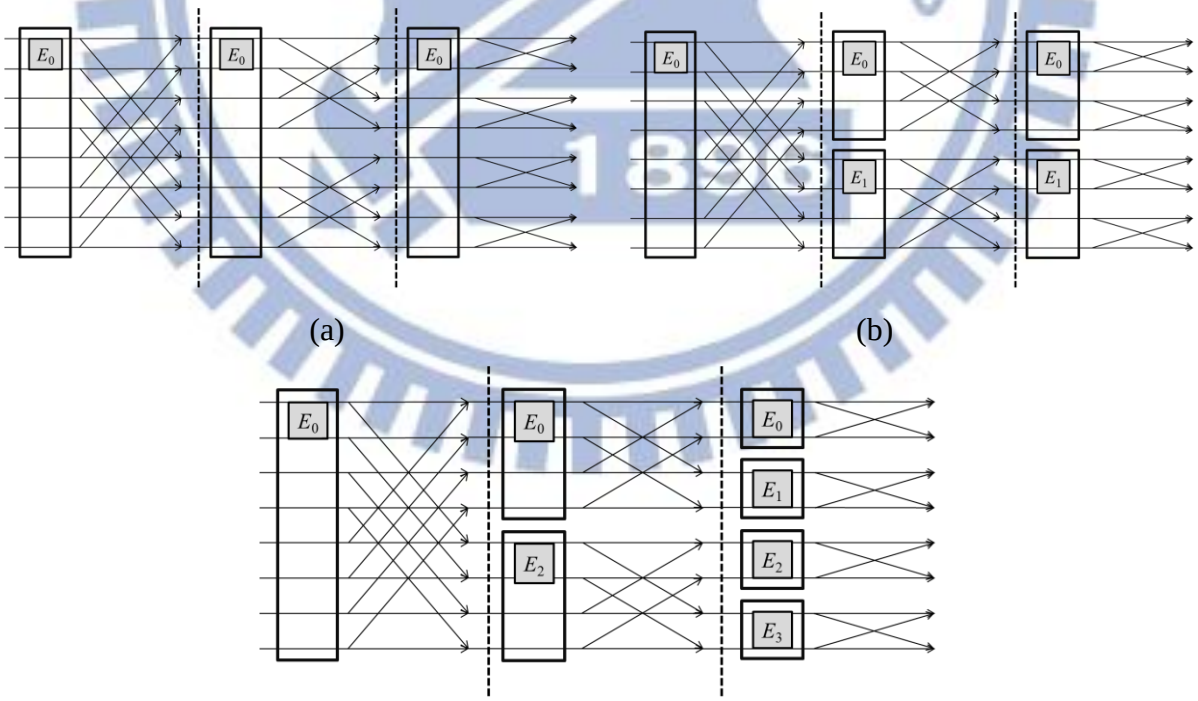


Fig. 33 The convergent block scaling with (a) $B_{max} = 1$ (b) $B_{max} = 2$ (c) $B_{max} = 4$

Taking the 8192-point 16-bit wordlength FFT with MRCBS as an example which uses the *S2*-type detector with 10-bit comparators, the performance of SQNR and area are shown in Fig. 34. It can be observed that the area of the storage is getting increased yet SQNR is getting saturated while the B_{max} is getting larger. It implies that in deeper stages, we are failed to get the SQNR we expect even if we double the area of the exponent storage. As we can see, if we divide the blocks until stage 11 which requires only 1024 exponent units, the area overhead of exponent storage is only 1/4 of that we divide until stage 13. However, the SQNR is just 0.13 dB lower than before. Thus, through doubling the number of blocks until a certain stage rather than doubling the number of blocks incessantly until the last stage, we can economize the usage of exponent storage to acquire the SQNR improvement we want. Although the SQNR performance is not the ultimately highest if we restrict the number of blocks, we can still get the acceptable SQNR and reduce area cost consequently.

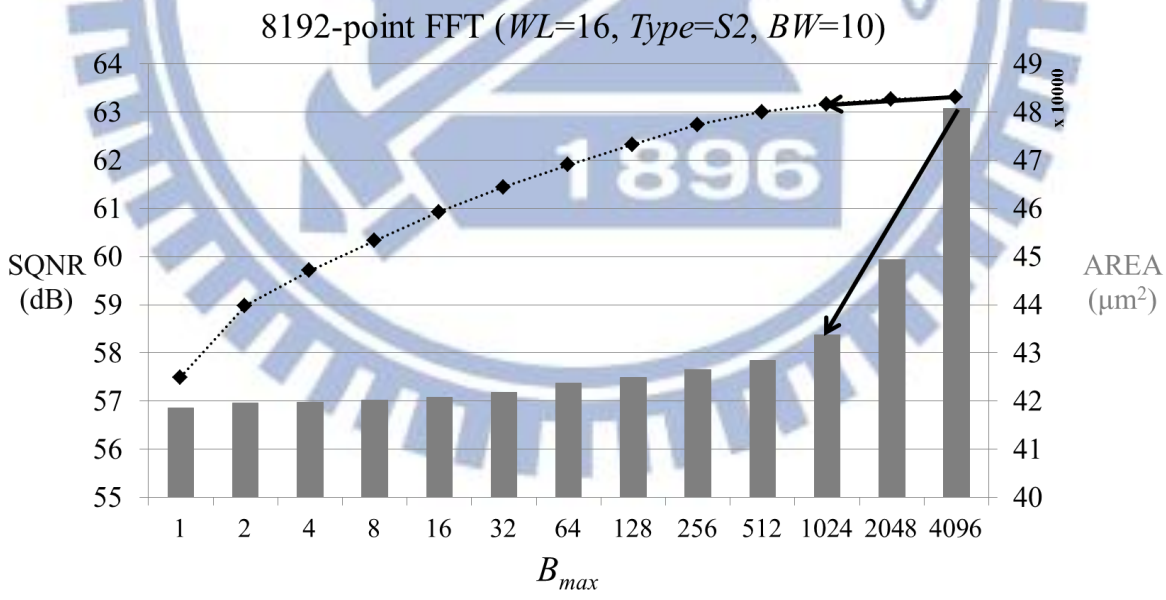


Fig. 34 The SQNR and area cost with different B_{max}

Chapter 5
Experimental Results

The proposed MRCBS method is to generate many hardware solutions for SQNR improvement and find out the one which meets the SQNR constraint with minimum area cost. Here we define the performance pair (PP): (*SQNR*, *AREA*) which indicates the SQNR performance with the corresponding area cost. Thus, each solution obtained by MRCBS has its own PP defined as PP^T: ($SQNR^T$, $AREA^T$) where the $SQNR^T$ represents the total SQNR performance and the $AREA^T$ represents the minimized total area cost.

The PP^T is determined by the quintuple (*N*, *WL*, *Type*, *BW*, B_{max}) where *N* is the given FFT size and *WL* is the wordlength of storage from 14 bits to 18 bits. The *Type* indicates different type of the detectors. *Type* = *Cj* implies the circular-type detectors and *Type* = *Sj* implies the square-type ones where *j* = 2, 4, and 6. The *Cj* detector includes four multipliers with bit width = *BW*, two adders with bit width = 2**BW* and 2*j* comparators with fixed bit width = 10 while the *Sj* detector includes 2*j* comparators with bit width = *BW*. The *BW* can be chosen from 5 to 10. And the total number of blocks B_{max} can be 2^{s-1} where *s* is from 1 to $\log_2 N$.

In this work, we choose radix-2 FFT for implementation, and the FFT size and SQNR constraint are user defined. We present the FFT size *N* = 1024, 2048, 4096, and 8192 in our experimental results as the SQNR constraint is in the range from 50 dB to 70 dB. Given the FFT size, we apply MRCBS method and build some tables for PPs by simulations and syntheses. And we will obtain many solutions by combining those tables. Consequently, for the given FFT size, we can find out the solution among them which meets the SQNR constraint and has the minimum area overhead. In addition to our MRCBS scheme, the traditional forced scaling method [7] and the conditional scaling method [14] are implemented as well and will be compared to our approach.

The fixed-point FFT model is built by C++, and the SQNR performance is obtained by simulations with random input signals. And the circuit area is implemented with TSMC 90 nm cell library and using Synopsys DesignWare to synthesize under 100MHz clock rate. Finally, the platform for both C++ and Synopsys DesignWare are built in Intel dual Pentium Xeon at 2.53GHz with 50GB of main memory.

5.1 The Solution Generated by MRCBS

The MRCBS scheme improves the SQNR by two ways. One is dividing data into blocks with additional exponent storage, and the other is adding the multi-region detector to the basic memory-based FFT design proposed in [7] which is implemented with forced scaling. Therefore, the total performance is the combinations of PP^{+} and the PP^{Base} as shown in (5.1). And the operation of combining two PPs is shown in (5.2).

The PP^{Base}: ($SQNR^{Base}$, $AREA^{Base}$) is the basic SQNR performance and original area cost obtained by the traditional memory-based FFT. On the other hand, the PP^{+} is the SQNR improvement and the additional area overhead obtained from the multi-region detection and convergent block scaling. PP_x^{+}: ($SQNR_x^{+}$, $AREA_x^{+}$) is the additional SQNR performance obtained by the multi-region detection with the extra area cost of the detector and predictor. And the PP_y^{+}: ($SQNR_y^{+}$, $AREA_y^{+}$) indicates the additional SQNR performance obtained by the block scaling with the extra area cost of the exponent array. Therefore, we can obtain those three performance pairs respectively and combine them to acquire the PP^{T}s. We will present the simulation results of these PPs in the following subsections.

$$PP^{T} = PP_x^{+} + PP_y^{+} + PP^{Base} \tag{5.1}$$

$$PP_1 + PP_2 = (SQNR_1 + SQNR_2,\ AREA_1 + AREA_2) \tag{5.2}$$

5.1.1 Performance Pair of the Forced Scaling FFT

We define the PP^{Base}: ($SQNR^{Base}$, $AREA^{Base}$) which is the performance pair of the traditional FFT design [7]. By SQNR simulation and hardware synthesis, the PP^{Base}s are shown in Table 5 which are determined by (*N*, *WL*).

WL \ N	1024	2048	4096	8192
14	47.97	46.75	44.85	44.17
15	53.96	52.75	50.85	50.18
16	60.00	58.78	56.87	56.20
17	66.01	64.78	62.89	62.23
18	72.01	70.81	68.92	68.25

(a)

WL \ N	1024	2048	4096	8192
14	67961	108890	208288	370549
15	72892	116783	222544	394919
16	77131	123983	236109	418597
17	80143	129931	249202	450454
18	84500	137225	263639	483656

(b)

Table 5 The PP^{Base} determined by (*N*, *WL*) (a)$SQNR^{Base}$ (dB) (b)$AREA^{Base}$ (μm^2)

5.1.2 Improvement from Multi-Region Detection

To know the effects on the performance of SQNR and area by the detector and predictor, we fix the numbers of blocks B_{max} = 1 and wordlength *WL* = 16 to get PPs. That is, those PPs are determined by (*N*, *WL* = 16, *Type*, *BW*, B_{max} = 1) by simulations and syntheses. Since we want to realize the improvement of SQNR and area called PP_x^+s produced by multi-region detection compared to the traditional FFT, those PPs will be offset by PP^{Base}s (*N*, *WL* = 16) which can be obtained by Table 5. We present the $SQNR_x^+$s for *N* = 1024, 2048, 4096 and 8192 in Table 6(a), (b), (c), and (d) respectively. Since the area of the detector and predictor are all the same with different *N*, we only show the $AREA_x^+$s of those PP_x^+s once in Table 7. By simulations, the $SQNR_x^+$ is getting saturated while *BW* is larger than 10, so we have *BW* only from 5 to 10 to choose for six types of detectors.

Type \ BW	5	6	7	8	9	10
S2	8.49	9.12	9.12	9.29	9.29	9.34
S4	10.02	10.37	10.50	10.57	10.59	10.61
S6	10.15	10.54	10.72	10.80	10.80	10.83
C2	10.59	10.99	11.19	11.27	11.30	11.31
C4	11.69	12.20	12.45	12.57	12.61	12.64
C6	11.73	12.30	12.58	12.68	12.72	12.76

(a)

Type \ BW	5	6	7	8	9	10
S2	10.20	10.95	10.95	11.13	11.13	11.19
S4	11.87	12.11	12.22	12.30	12.34	12.34
S6	12.05	12.44	12.62	12.67	12.69	12.70
C2	12.44	12.81	13.11	13.16	13.21	13.23
C4	13.53	13.98	14.34	14.42	14.50	14.52
C6	13.65	14.12	14.47	14.54	14.62	14.63

(b)

Type \ BW	5	6	7	8	9	10
S2	11.29	12.20	12.20	12.32	12.32	12.41
S4	12.75	13.24	13.38	13.45	13.47	13.53
S6	12.83	13.37	13.61	13.68	13.69	13.74
C2	13.63	14.02	14.24	14.35	14.40	14.42
C4	14.68	15.15	15.35	15.44	15.50	15.54
C6	14.72	15.23	15.45	15.55	15.61	15.64

(c)

Type \ BW	5	6	7	8	9	10
S2	13.59	14.66	14.66	14.83	14.83	14.88
S4	15.22	15.65	15.72	15.78	15.80	15.81
S6	15.31	15.99	16.15	16.18	16.19	16.24
C2	15.91	16.41	16.66	16.79	16.84	16.88
C4	17.32	17.68	17.85	17.91	17.96	17.98
C6	17.37	17.72	17.91	17.99	18.05	18.07

(d)

Table 6 The $SQNR_x^+$ (dB) of the PP_x^+ for (a)1024 (b)2048 (c)4096 (d)8192 -point FFT

Type \ BW	5	6	7	8	9	10
S2	92	107	122	138	159	176
S4	288	295	353	402	493	507
S6	461	502	620	732	763	804
C2	875	1320	2002	2669	3465	4270
C4	1057	1509	2187	2817	3633	4402
C6	1226	1705	2404	3023	3834	4660

Table 7 The $AREA_x^+$ (μm^2) of the PP_x^+

5.1.3 Improvement from Convergent Block Scaling

To realize the relationship between total number of blocks B_{max} and the performance of area and SQNR, we fix BW = 10 and WL = 16 to get PPs by simulations and synthesis. Those PPs will be offset by B_{max} = 1 to obtain the additional SQNR and area cost produced by the

block scaling scheme with shared exponents which are defined as PP_y^+s . That is, the PP_y^+ is obtained by (N, WL = 16, $Type$, BW = 10, B_{max}). Table 8 (a), (b), (c), and (d) shows the $SQNR_y^+$ of PP_y^+ for N = 1024, 2048, 4096 and 8192 respectively. Because the $AREA_y^+$ consists of the exponent storage and the control circuits of exponent accesses, it only depends on the B_{max} and N. Therefore, we only show the $AREA_y^+$ once in the second row of each table. The larger B_{max} implies the more storage of the exponents so the area is larger. And the control circuit accessing the exponent units is more complicated while N is larger, so $AREA_y^+$ of 8192-point FFT is larger than that of 1024-point with the same B_{max}.

B_{max}		1	2	4	8	16	32	64	128	256	512
$AREA_y^+$		0	782	1033	1359	1947	3015	4967	6165	7690	9588
Type	*S2*	0	1.26	1.92	2.51	3.06	3.54	4.09	4.40	4.59	4.64
	S4	0	1.24	1.88	2.51	3.04	3.49	3.97	4.23	4.39	4.45
	S6	0	1.33	2.06	2.73	3.28	3.71	4.15	4.38	4.53	4.59
	C2	0	1.31	1.98	2.60	3.10	3.53	4.05	4.32	4.48	4.53
	C4	0	1.30	1.94	2.48	2.96	3.35	3.80	4.06	4.21	4.26
	C6	0	1.32	2.00	2.55	3.05	3.44	3.87	4.12	4.27	4.32

(a)

B_{max}		1	2	4	8	16	32	64	128	256	512	1024
$AREA_y^+$		0	845	1090	1426	2003	3074	5035	6231	7746	9636	15093
Type	*S2*	0	1.40	2.11	2.71	3.28	3.78	4.22	4.64	4.89	5.02	5.07
	S4	0	1.40	2.16	2.79	3.33	3.80	4.19	4.59	4.84	4.97	5.02
	S6	0	1.39	2.21	2.85	3.41	3.87	4.28	4.65	4.89	5.01	5.05
	C2	0	1.36	2.13	2.74	3.26	3.70	4.12	4.52	4.77	4.90	4.94
	C4	0	1.39	2.08	2.66	3.19	3.63	4.01	4.37	4.58	4.71	4.75
	C6	0	1.40	2.12	2.74	3.28	3.74	4.10	4.45	4.65	4.77	4.81

(b)

B_{max}		1	2	4	8	16	32	64	128	256	512	1024	2048
$AREA_y^+$		0	894	1120	1461	2049	3122	5086	6282	7795	9711	15132	30712
Type	*S2*	0	1.57	2.34	2.97	3.56	4.07	4.54	5.18	5.49	5.70	5.81	5.85
	S4	0	1.62	2.42	3.08	3.66	4.20	4.66	5.21	5.49	5.67	5.77	5.81
	S6	0	1.65	2.51	3.24	3.89	4.43	4.88	5.37	5.63	5.80	5.90	5.93
	C2	0	1.55	2.36	3.03	3.63	4.15	4.59	5.19	5.48	5.67	5.78	5.82
	C4	0	1.78	2.59	3.23	3.76	4.26	4.65	5.16	5.41	5.59	5.69	5.73
	C6	0	1.77	2.61	3.28	3.85	4.35	4.74	5.23	5.47	5.65	5.74	5.78

(c)

B_{max}		1	2	4	8	16	32	64	128	256	512	1024	2048	4096
$AREA_y^+$		0	953	1177	1537	2130	3206	5156	6357	7874	9776	15220	30787	62096
Type	*S2*	0	1.48	2.21	2.84	3.43	3.95	4.41	4.83	5.24	5.50	5.67	5.77	5.81
	S4	0	1.73	2.56	3.20	3.75	4.26	4.70	5.08	5.46	5.70	5.86	5.95	5.99
	S6	0	1.66	2.52	3.17	3.75	4.27	4.71	5.08	5.44	5.67	5.83	5.91	5.94
	C2	0	1.55	2.30	2.96	3.50	4.02	4.45	4.83	5.24	5.49	5.66	5.75	5.78
	C4	0	1.72	2.56	3.19	3.71	4.17	4.59	4.95	5.34	5.57	5.71	5.80	5.83
	C6	0	1.75	2.62	3.25	3.79	4.28	4.71	5.08	5.45	5.67	5.81	5.89	5.92

(d)

Table 8 The PP_y^+ (dB, μm^2) for (a) 1024 (b)2048 (c)4096 (d) 8192 -point FFT

5.1.4 Performance Pair Combination

To get the result of total area and total SQNR performance PP^T, we have to combine PP_x^+, PP_y^+, and PP^{Base} as (5.1) shows. The PP^{Base} can be figure out in Table 5. And the PP_x^+ can be obtained in Table 6 and Table 7 as PP_y^+ can be obtained in Table 8. Although *WL* in PP_x^+ and PP_y^+ is fixed to 16, we found that the *WL* does not affect the results so much and assume different *WL* will have the same results. As a result, given FFT size *N*, we will combine PP_x^+, PP_y^+, and PP^{Base} with *WL* from 14 to 18 to get 5(*WL*) * 6(*Type*) * 6(*BW*) * $\log_2 N$(B_{max}) PP^Ts. In these PP^Ts, there may be some ones producing the same $SQNR^T$ but the $AREA^T$s are different. Therefore, we will delete the PP^T which has the larger $AREA^T$ but lower $SQNR^T$ to

reserve the irreplaceable PP^{T}s. Consequently, in each 6 dB range, we have 40 PP^{T}s to be chosen to satisfy the SQNR constraint.

Besides, our PP^{T}s include the solutions obtained by conditional scaling scheme in [14]. Those solutions are the special cases determined by (N, WL, $Type$ = $S2$, BW = 10, B_{max} = 1). As shown in Fig. 35, Fig. 36, Fig. 37, and Fig.38, the black dots are the PP^{T}s obtained by the proposed MRCBS method, the gray diamonds are the solutions obtained by the scheme in [14], and the triangles are the solutions obtained by the scheme in [7] which are the PP^{Base}s for N = 1024, 2048, 4096, and 8192.

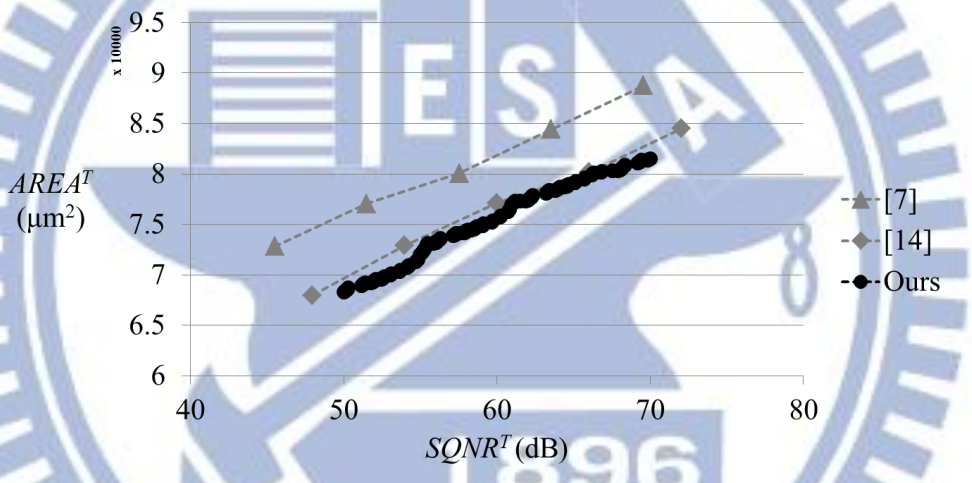


Fig. 35 The PP^{T}s for 1024-point FFT generated by MRCBS

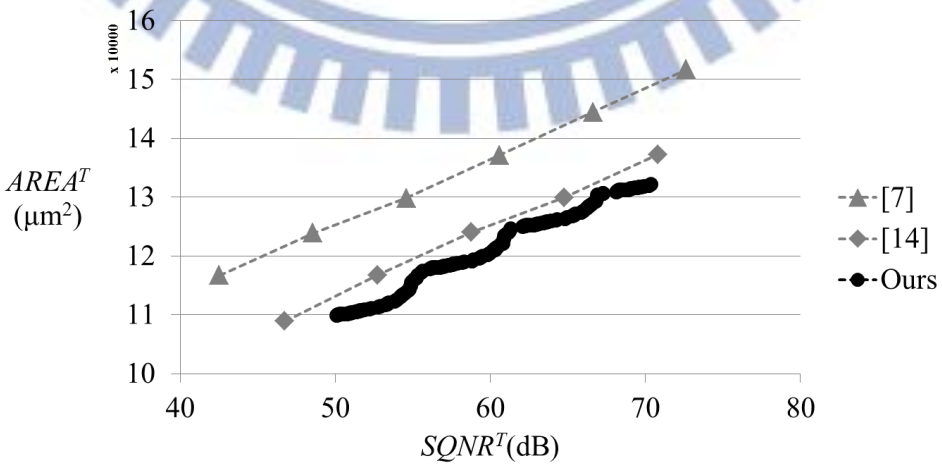


Fig. 36 The PP^{T}s for 2048-point FFT generated by MRCBS

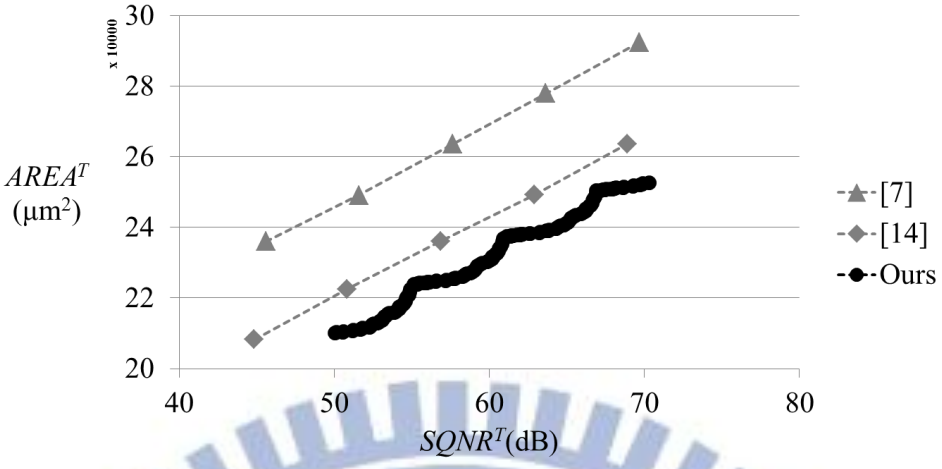


Fig. 37 The PP^{T}s for 4096-point FFT generated by MRCBS

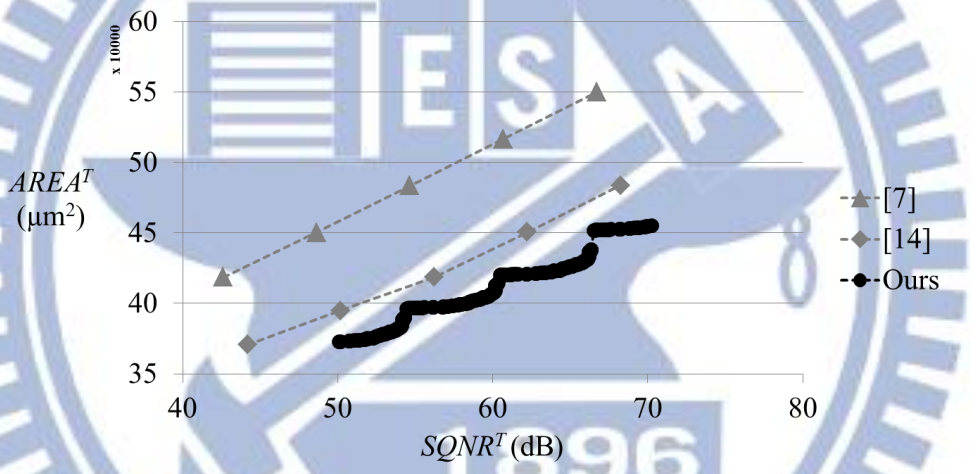


Fig. 38 The PP^{T}s for 8192-point FFT generated by MRCBS

5.2 Area Minimization under SQNR Constraint

In those irreplaceable PP^{T}s for certain FFT size, the $AREA^T$ is definitely larger while the $SQNR^T$ is higher. Therefore, we sort the PP^{T}s by $SQNR^T$ from small to large, and then search the $SQNR^T$ which is just satisfying the requirement. As a result, the PP^T we find out will be the solutions which has the smallest $AREA^T$.

Table 9, Table 10, Table 11, Table 12 show 8 different SQNR requirements with FFT size N = 1024, 2048, 4096, and 8192, respectively. Under different constraints, the solutions will tell us the required wordlength, the type of the detector, the bit width in the detector, and the

total number of blocks. The exact SQNR is obtained by simulations and is almost equal to the the $SQNR^T$ estimated by MRCBS method. And if previous work has area cost K, the area reduction is derived by (K - $AREA^T$) / K. Compared to the traditional FFT implemented with forced scaling, our method can reduce the area cost by 12.61% for N = 1024 and 23.57% for N = 8192 in the best case.

Besides, we know that conditional scaling has better performance compared to the forced scaling. However, if the conditional scaling scheme just meets the constraint in some cases, our method can reduce one bit of wordlength to save the area of memory storage. And if the constraint becomes tighter so that the previous conditional scaling scheme has to increase one bit to meet the constraint, our method will uses more blocks or more precise detector to meet the requirement and still maintain the wordlength. Therefore, we will reduce 2 bits of wordlength. That is, with larger-size FFT, the area occupancy of 2-bit memory wordlength will become larger. As we can see, we can reduce the area cost by 6.34% for N = 1024 but reduce 12.84% for larger N = 8192.

(dB) Constraint	*WL*	*Type*	*BW*	B_{max}	$SQNR^T$ (dB)	$AREA^T$ (μm²)	Exact SQNR (dB)	SQNR [7] (dB)	Area [7] (μm²)	Area Reduction [7] (%)	SQNR [14] (dB)	Area [14] (μm²)	Area Reduction [14] (%)
50	14	*S4*	8	1	50.04	68271	50.00	51.51	77039 (*WL* = 16)	**11.38**	53.96	72892 (*WL* = 15)	**6.34**
53	14	*C4*	5	4	53.07	69959	53.25	57.55	80050 (*WL* = 17)	**12.61**	53.96	72892 (*WL* = 15)	**4.02**
55	14	*C6*	7	16	55.11	72219	55.11	57.55	80050 (*WL* = 17)	**9.78**	60.00	77131 (*WL* = 16)	**6.37**
58	15	*S6*	7	4	58.10	74452	57.94	63.54	84407 (*WL* = 18)	**11.79**	60.00	77131 (*WL* = 16)	**3.47**
61	15	*C6*	7	16	61.10	77150	61.15	63.54	84407 (*WL* = 18)	**8.60**	66.01	80143 (*WL* = 17)	**3.73**
63	16	*C4*	5	1	63.19	78096	63.19	63.54	84407 (*WL* = 18)	**7.48**	66.01	80143 (*WL* = 17)	**2.55**
66	16	*C4*	6	8	66.21	79906	66.26	69.57	88764 (*WL* = 19)	**9.98**	66.01	80143 (*WL* = 17)	**0.30**
68	17	*S4*	7	1	68.02	80403	67.91	69.57	88764 (*WL* = 19)	**9.42**	72.01	84450 (*WL* = 18)	**4.85**

Table 9 The solutions under the SQNR constraints for 1024-point FFT

(dB) Constraint	*WL*	*Type*	*BW*	B_{max}	$SQNR^T$ (dB)	$AREA^T$ (μm^2)	Exact SQNR (dB)	SQNR [7] (dB)	Area [7] (μm^2)	Area Reduction [7] (%)	SQNR [14] (dB)	Area [14] (μm^2)	Area Reduction [14] (%)
50	14	*C*4	5	1	50.09	109855	50.09	54.58	129839 (*WL* = 17)	**15.39**	52.75	116783 (*WL* = 15)	**5.93**
53	14	*C*4	6	8	53.25	111733	53.37	54.58	129839 (*WL* = 17)	**13.94**	58.78	123983 (*WL* = 16)	**9.88**
55	14	*C*4	7	64	55.11	116020	54.94	60.61	137132 (*WL* = 18)	**15.40**	58.78	123983 (*WL* = 16)	**6.42**
58	15	*C*4	5	4	58.20	118837	58.41	60.61	137132 (*WL* = 18)	**13.34**	58.78	123983 (*WL* = 16)	**4.15**
61	15	*C*4	7	64	61.12	123912	60.95	66.63	144426 (*WL* = 19)	**14.20**	64.78	129931 (*WL* = 17)	**4.63**
63	16	*S*4	9	4	63.02	125473	63.05	66.63	144426 (*WL* = 19)	**13.12**	64.78	129931 (*WL* = 17)	**3.43**
66	16	*C*4	7	16	66.21	128080	66.11	66.63	144426 (*WL* = 19)	**11.32**	70.81	137225 (*WL* = 18)	**6.66**
68	17	*C*4	5	1	68.12	130896	68.13	72.64	151719 (*WL* = 20)	**13.72**	70.81	137225 (*WL* = 18)	**4.61**

Table 10 The solutions under the SQNR constraints for 2048-point FFT

(dB) Constraint	*WL*	*Type*	*BW*	B_{max}	$SQNR^T$ (dB)	$AREA^T$ (μm^2)	Exact SQNR (dB)	SQNR [7] (dB)	Area [7] (μm^2)	Area Reduction [7] (%)	SQNR [14] (dB)	Area [14] (μm^2)	Area Reduction [14] (%)
50	14	*S*4	10	8	50.05	210163	50.15	51.60	249109 (*WL* = 17)	**15.63**	50.85	222545 (*WL* = 15)	**5.56**
53	14	*C*6	7	32	53.08	213721	53.36	57.63	263547 (*WL* = 18)	**18.91**	56.87	236109 (*WL* = 16)	**9.48**
55	15	*S*4	6	4	55.13	223867	55.31	57.63	263547 (*WL* = 18)	**15.06**	56.87	236109 (*WL* = 16)	**5.18**
58	15	*C*4	6	16	58.27	226009	58.25	63.65	277984 (*WL* = 19)	**18.7**	62.89	249202 (*WL* = 17)	**9.31**
61	16	*S*4	6	4	61.16	237431	61.33	63.65	277984 (*WL* = 19)	**14.59**	62.89	249202 (*WL* = 17)	**4.72**
63	16	*C*4	5	8	63.23	238534	63.55	63.65	277984 (*WL* = 19)	**14.19**	68.92	263639 (*WL* = 18)	**9.52**
66	16	*C*4	7	128	66.12	244485	66.13	69.69	292421 (*WL* = 20)	**16.39**	68.92	263639 (*WL* = 18)	**7.27**
68	17	*S*4	8	8	68.03	250972	68.12	69.69	292421 (*WL* = 20)	**14.17**	68.92	263639 (*WL* = 18)	**4.80**

Table 11 The solutions under the SQNR constraints for 4096-point FFT

(dB) Constraint	*WL*	*Type*	*BW*	B_{max}	$SQNR^T$ (dB)	$AREA^T$ (μm^2)	Exact SQNR (dB)	SQNR [7] (dB)	Area [7] (μm^2)	Area Reduction [7] (%)	SQNR [14] (dB)	Area [14] (μm^2)	Area Reduction [14] (%)
50	14	*C*4	5	4	50.12	372690	50.32	54.65	483564 (*WL* = 18)	**22.93**	50.18	394919 (*WL* = 15)	**5.63**
53	14	*C*4	6	128	53.09	378322	53.15	54.65	483564 (*WL* = 18)	**21.76**	56.20	418597 (*WL* = 16)	**9.62**
55	15	*S*4	6	8	55.08	396658	55.43	60.67	516760 (*WL* = 19)	**23.24**	56.20	418597 (*WL* = 16)	**5.24**
58	15	*C*4	6	32	58.22	399541	58.40	60.67	516760 (*WL* = 19)	**22.68**	62.23	450454 (*WL* = 17)	**11.30**
61	16	*S*4	6	8	61.10	420336	61.45	66.70	549968 (*WL* = 20)	**23.57**	62.23	450454 (*WL* = 17)	**6.69**
63	16	*C*4	6	8	63.13	421550	63.44	66.70	549968 (*WL* = 20)	**23.35**	68.25	483656 (*WL* = 18)	**12.84**
66	16	*C*6	7	512	66.03	430685	66.19	66.70	549968 (*WL* = 20)	**21.69**	68.25	483656 (*WL* = 18)	**10.95**
68	17	*C*4	5	4	68.18	452596	68.38	72.73	583170 (*WL* = 21)	**22.39**	68.25	483656 (*WL* = 18)	**6.42**

Table 12 The solutions under the SQNR constraints for 8192-point FFT

Chapter 6
Conclusions and Future Works

In this thesis, a scaling scheme for the memory-based FFT design is proposed which improves SQNR in an area-efficient way. This method takes advantage of both conditional scaling and convergent block scaling. By implementing with different detectors and using different number of the shared exponents, it will generate many solutions with different SQNR and area performance. Moreover, we can satisfy the SQNR requirement by increasing the area economically by applying this method.

The experimental results show that it will save at least one bit of wordlength to reduce about 5.6% area from previous conditional scaling method. And if the constraint is just a little tighter, our method can satisfy the required SQNR by increasing small area rather than increasing one bit of wordlength in previous approaches. As a result, the proposed scheme will save 2 bits of wordlength to bring about 13% area reduction from the conditional scaling scheme for 8192-point FFT in the best case.

In the future, the multi-region detection and the convergent block scaling method can be improved to optimize the SQNR and the area of the FFT core for different architectures and different algorithms.

References

[1] C. T. Lin, Y. C. Yu, L. D. Van, "A low-power 64-point FFT/IFFT design for IEEE 802.11a WLAN application," *IEEE International Symposium on Circuits and Systems*, pp. 4 pp. -4526, 2006.

[2] R. V. Nee, R. Prasad, *OFDM for Multimedia Communications*, Artech House, 2000.

[3] ETSI, "Digital Video Broadcasting (DVB); Framing Structure, Channel Coding and Modulation for Digital Terrestrial Television," ETSI EN 300 744 v1.4.1, 2001.

[4] E. Grass, K. Tittelbach, U. Jagdhold, A. Troya, G. Lippert, O. Kruger, J. Lehmann, K. Maharatna, K. Dombrowski, N. Fiebig, R. Kraemer, P. Mahonen, "On the single-chip implementation of a Hiperlan/2 and IEEE 802.11a capable modem," *IEEE Personal Communications*, vol. 8, no. 6, pp. 48-57, 2001.

[5] S. Li, H. Xu, W. Fan, Y. Chen, X. Zeng, "A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e," *IEEE International Symposium on Circuits and Systems*, pp. 1488-1491, 2010.

[6] C. Y. Wang, C. B. Kuo, J. Y. Jou, "Hybrid Wordlength Optimization Methods of Pipelined FFT Processors," *IEEE Transactions on Computers*, vol.56, no.8, pp. 1105-1118, 2007.

[7] A. V. Oppenheim, C. J. Weinstein, "Effects of finite register length in digital filtering and the fast Fourier transform," *Proceedings of the IEEE* , vol. 60, no. 8, pp. 957-976, 1972.

[8] Y. W. Lin, H. Y. Liu, 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, 2004.

[9] S. N. Tang, J. W. Tsai, T. Y. Chang, "A 2.4-GS/s FFT Processor for OFDM-Based WPAN Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 6, pp. 451-455, 2010.

[10] Y. Chen, Y. C. Tsao, Y. W. Lin, C. H. Lin, C. Y. Lee, "An Indexed-Scaling Pipelined FFT Processor for OFDM-Based WPAN Applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 55, no. 2, pp. 146-150, 2008.

[11] S. Ramakrishnan, J. Balakrishnan, K. Ramasubramanian, "Exploiting signal and noise statistics for fixed point FFT design optimization in OFDM systems," *National Conference on Communications (NCC)*, pp. 1-5, 2010.

[12] E. Bidet, D. Castelain, C. Joanblanq, P. Senn, "A fast single-chip implementation of 8192 complex point FFT," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 300-305, 1995.

[13] R. R. Shively, “A Digital Processor to Generate Spectra in Real Time,” *IEEE Transactions on Computers*, vol. C-17, no. 5, pp. 485-491, 1968.

[14] R. Koutsoyannis, P. Milder, C. R. Berger; M. Glick, J. C. Hoe; M. Puschel, “Improving Fixed-point Accuracy of FFT Cores in O-OFDM Systems,” *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012.

[15] J. W. Cooley, J. W. Turkey, “An algorithm for machine computation of complex Fourier series,” *Math. Computation*, vol. 19, pp. 291-301, 1965.

[16] W. C. Yeh; C. W. Jen, “High-speed and low-power split-radix FFT,” *IEEE Transactions on Signal Processing*, vol. 51, no. 3, pp. 864-874, 2003.

[17] Y. W. Lin, H. Y. Liu, 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, 2005.

[18] R. C. Agarwal, J. W. Cooley, “Vectorized mixed radix discrete Fourier transform algorithms,” *Proceedings of the IEEE* , vol. 75, no. 9, pp. 1283-1292, 1987.

[19] P. Y. Tsai, C. Y. Lin, “A Generalized Conflict-Free Memory Addressing Scheme for Continuous-Flow Parallel-Processing FFT Processors With Rescheduling,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 12, pp. 2290-2302, 2011.

[20] D. Reisis, N. Vlassopoulos, “Conflict-Free Parallel Memory Accessing Techniques for FFT Architectures,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 55, no. 11, pp. 3438-3447, 2008.

[21] Y. W. Lin, “The study of FFT processors for OFDM systems,” Ph. D. thesis, Dept. of Electronic Engineering, National Chiao Tung University, Hsinchu, R.O.C., 2004.

[22] S. Lee, S. C. Park, “Modified SDF Architecture for Mixed DIF/DIT FFT,” *IEEE International Symposium on Circuits and Syatems*, pp. 2590-2593, 2007.

[23] A. Cortes, I. Velez, J. F. Sevillano, “Radix r^k FFTs: Matricial Representation and SDC/SDF Pipeline Implementation,” *IEEE Transactions on Signal Processing*, vol. 57, no. 7, pp. 2824-2839, 2009.

[24] B. C. Lin, Y. H. Wang, J. D. Huang, J. Y. Jou, “Expandable MDC-based FFT architecture and its generator for high-performance applications,” *IEEE International SOC Conference (SOCC)* , pp. 188-192, 2010.

[25] E. Bidet, D. Castelain, C. Joanblanq, P. Senn, “A fast single-chip implementation of 8192 complex point FFT,” *IEEE Journal of Solid-State Circuits*, vol. 30, no. 3, pp. 300-305, 1995.