

國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

具能量察覺管線化架構可重組混合基底的
快速傅利葉轉換處理器設計

Energy-Aware Pipeline-based Reconfigurable Mixed-Radix

FFT/IFFT Processor Design

研究生：賴祈成

指導教授：黃 威 教授

中華民國九十五年六月

具能量察覺管線化架構可重組混合基底的
快速傅利葉轉換處理器設計

Energy-Aware Pipeline-based Reconfigurable Mixed-Radix
FFT/IFFT Processor Design

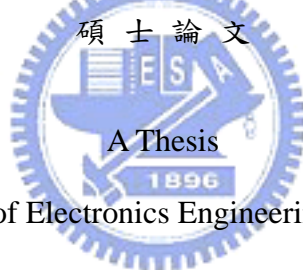
研究生：賴祈成

Student : Chi-Chen Lai

指導教授：黃 威 教授

Advisor : Prof. Wei Hwang

國立交通大學
電子工程學系電子研究所



Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering

June 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

具能量察覺管線化架構可重組混合基底的 快速傅利葉轉換處理器設計

學生：賴祈成

指導教授：黃 威 教授

國立交通大學電子工程學系電子研究所碩士班

摘 要

本論文提出一個先進的可重組混合基底的快速傅利葉轉換處理器。該處理器可動態重組為 16 點至 4096 點之快速傅利葉/反向快速傅利葉轉換運算，並且對於不同長度之模式使用不同的混合基底演算法，所提出的架構同時具有能量察覺的特色。不同於一般管線化架構使用較大的內部字長來提高抗雜訊比，我們的架構使用與輸入資料相同的內部字長，並使用區塊浮點的方法來維持抗雜訊比。並且，使用八個平行資料傳輸路徑的管線化架構有效的降低計算週期。

模擬的結果顯示，所提出的快速傅利葉轉換器在不同的資料長度下，能將抗雜訊比維持在 110dB 以上。所提出的快速傅利葉轉換器以 TSMC 0.13 μm 的技術實現，供應電壓為 1.2V，最高時脈週期為 110MHz，產出率可達四倍時脈週期，亦即 440Msample/s；隨著快速傅利葉轉換運算的長度增加，每筆運算所消耗的能量從 4.34nJ 增加到 5.115 μJ 。

Energy-Aware Pipeline-based Reconfigurable Mixed-Radix FFT/IFFT Processor Design

Student : Chi-Chen Lai

Advisor : Prof. Wei Hwang

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

ABSTRACT

In this thesis, we present a novel FFT/IFFT processor, called reconfigurable mixed-radix (RMR) FFT. It can be easily reconfigured as from 16-point to 4096-point FFT/IFFT with proper mixed-radix algorithm assigned for each mode. The proposed architecture is characterized with scalable energy dissipation for different FFT/IFFT sizes. Unlike general pipeline-based architectures which use a larger internal wordlength to achieve a high signal-to-noise ratio (SNR), our processor keeps the internal wordlength the same as the wordlength of the input data while the block-floating-point (BFP) approach is adopted to maintain the SNR. The pipeline-based architecture with 8-parallel datapath results in low computation cycles.

The simulation result shows that RMR FFT maintain the SNR above 110dB as the FFT size varies. The proposed RMR FFT processor is implemented using TSMC 0.13 μ m technology with a supply voltage of 1.2V. With the maximum clock rate of 110MHz, the throughput rate can reach 440Msample/s, which is 4 times of the input clock rate. The energy dissipation per FFT ranges from 4.34nJ to 5.115 μ J with increasing FFT sizes.

Acknowledgements

I would like to thank my advisor, Prof. Wei Hwang, who has provided me a free research environment for the past two years. He has been supportive all the way, which does help me get rid of the fear of any disturbance in the rear. I was able to think and research independently on interesting topics. I have learned more from these experiences than what books or papers may show.

The fellows of my laboratory also help lot on my study. In addition, they are more helpful on life and many daily events. I have also learned a lot from them and overcome many difficulties with their help.

I would also like to thank my roommates and many schoolmates for the past few years. They have accompanied me a long time and so much has happened. The life in NCTU would be less colorful without them.



Table of Contents

Chapter 1 Introduction	1
1.1 Background.....	1
1.2 Motivation.....	1
1.3 Organization of Thesis.....	2
Chapter 2 Review of FFT Algorithms and Architectures	4
2.1 Introduction.....	4
2.2 Basic Concept of FFT Algorithms	5
2.3 The FFT Algorithms.....	6
2.3.1 Decimation-in-Frequency (DIF) Fixed-Radix Algorithms	6
2.3.2 Decimation-in-Time (DIT) Fixed-Radix Algorithms.....	11
2.3.3 Other FFT Algorithms.....	13
2.4 The FFT Architecture	14
2.4.1 Pipeline-Based Architecture.....	15
2.4.1.1 Single-Path Delay Feedback (SDF) Architecture.....	15
2.4.1.2 Multiple-Path Delay Commutator (MDC) Architecture	16
2.4.2 Memory-Based Architecture.....	17
2.4.3 Reconfigurable Architecture.....	18
2.5 Conclusion	20
Chapter 3 Algorithm of Reconfigurable Mixed-Radix FFT	21
3.1 Introduction.....	21
3.2 Reconfigurable Mixed-Radix Algorithm.....	21
3.3 Data Ordering and Twiddle Factors	25
3.4 Finite Register Length Effect and Block-Floating-Point Method.....	30
3.5 Conclusion	33
Chapter 4 Architecture of Reconfigurable Mixed-Radix FFT	35
4.1 Introduction.....	35
4.2 Overall Architecture	36
4.3 Architecture Design.....	37
4.3.1 Butterfly (BF) Unit.....	37
4.3.1.1 General BF.....	37
4.3.1.2 Reconfigurable BF	38
4.3.2 Multiplier Stage	39
4.3.2.1 Constant Multiplier Approach	39

4.3.2.2	Complex Multiplier Approach	43
4.3.3	Register Banks (RB)	45
4.3.3.1	RB_64	46
4.3.3.2	RB_512 and RB_4096	48
4.3.3.3	Duplicate Module Insertion	51
4.3.4	BFP	52
4.3.5	Input/Output Buffer	53
4.4	Data Flow	55
4.5	Conclusion	58
Chapter 5	Implementation of RMR FFT/IFFT Processor	59
5.1	Introduction	59
5.2	Implementation Issue on Register Banks	59
5.3	Power Control	64
5.4	Simulation Result	66
5.4.1	Performance of the RMR FFT	66
5.4.2	Comparison	69
5.5	Layout Implementation	72
5.6	Conclusion	77
Chapter 6	Conclusions and Future Work	72
6.1	Conclusions	79
6.2	Future Work	80
References	81



List of Tables

TABLE 3.1	Mixed-radix algorithms for different FFT sizes.	23
TABLE 3.2	N-based twiddle factors required for each multiplier stage under different FFT size	29
TABLE 3.3	Storage elements required between each stage.....	33
TABLE 4.1	Truth table of control signals for reconfigurable BF	39
TABLE 4.2	Mapping table of the twiddle factors	40
TABLE 4.3	Implementation table of constants	41
TABLE 4.4	Scheduling of twiddle factors, W_{64}^p	41
TABLE 4.5	Scheduling of twiddle factors, W_{4096}^p	44
TABLE 4.6	Twiddle factors, W_{4096}^p , stored in 4 th ROM	44
TABLE 4.7	Comparison of memory requirement (including the input buffer)	57
TABLE 4.8	Execution cycles required for different FFT length	57
TABLE 5.1	Truth table of the activated modules.....	65
TABLE 5.2	Execution cycles required for various FFT sizes.....	68
TABLE 5.3	Comparison with other reconfigurable architectures.....	70
TABLE 5.4	Execution cycles required per FFT	71

List of Figures

Figure 1.1	Generic OFDM block diagram	2
Figure 2.1	Decomposition of the 8-point DFT step by step in DIF algorithm	8
Figure 2.2	The butterfly unit of radix-2 DIF FFT	8
Figure 2.3	Tree diagrams of (a) normal order and (b) bit-reversed order	9
Figure 2.4	The butterfly unit of radix-4 DIF FFT	10
Figure 2.5	Decomposition of the 8-point DFT step by step in DIT algorithm	12
Figure 2.6	The butterfly unit of radix-2 DIT FFT	13
Figure 2.7	The butterfly unit of split-radix 2/4 algorithm	14
Figure 2.8	Radix-2 DIF SDF architecture for $N = 16$	15
Figure 2.9	Radix-4 DIF SDF architecture for $N = 64$	16
Figure 2.10	Radix-2 MDC architecture for $N = 16$	16
Figure 2.11	Radix-4 DIF MDC architecture for $N = 64$	17
Figure 2.12	Block diagram of the memory-based architecture	18
Figure 2.13	Architecture of 1024-point radix-4 reconfigurable pipelined FFT processor	19
Figure 3.1	SFG of 8-point DIF FFT	24
Figure 3.2	SFG of 128-point FFT in radix-2 DIF algorithm	26
Figure 3.3	SFG of 128-point FFT in mixed-radix algorithm	26
Figure 3.4	SFG of 16-point DFT in (a) radix-2 algorithm, and (b) radix-8/2 algorithm	27
Figure 3.5	Extraction of radix-8 butterfly	28
Figure 3.6	Procedure of combining three radix-2 stages into one radix-8 stage	28
Figure 3.7	The twiddle factors for the m th radix-8 butterfly for N -point decomposition	29
Figure 3.8	Concept of block-floating-point	31
Figure 3.9	Blocks decomposition of 128-point FFT	32
Figure 4.1	FFT environment	35
Figure 4.2	Block diagram of the proposed RMR FFT	36
Figure 4.3	Circuit diagram of multiplication by $1/\sqrt{2}$	38
Figure 4.4	Block diagram of general radix-8 butterfly	38
Figure 4.5	Block diagram of reconfigurable butterfly	39
Figure 4.6	Twiddle factors on the unit circle	40
Figure 4.7	Block diagram of a constant multiplier	42
Figure 4.8	Block diagram of CMULT stage	42
Figure 4.9	Block diagram of MULT stage	43
Figure 4.10	Block diagram of ROM banking	44
Figure 4.11	Block diagram of the two-input register	46

Figure 4.12	Block diagram of RB_64 for three different capacities, (a) 16-word, (b) 32-word, and (c) 64-word	46
Figure 4.13	Data flow in RB for 16-word mode.....	47
Figure 4.14	Block diagram of reconfigurable RB_64	48
Figure 4.15	Data flow in RB for 128-word mode.....	49
Figure 4.16	Control zones for the RB	50
Figure 4.17	Control signals for the 128-word RB	50
Figure 4.18	Block diagram of reconfigurable RB_512	51
Figure 4.19	Block diagram of BFP	52
Figure 4.20	Block diagram of reconfigurable input buffer.....	54
Figure 4.21	Data flow in the input buffer for N = 16	54
Figure 4.22	Flow of data path for 16, 32, 64-pont FFT	55
Figure 4.23	Flow of data path for 128, 256, 512-pont FFT.....	56
Figure 4.24	Control signal, PHASE, for duplicate RB modules	56
Figure 4.25	Flow of data path for 1024, 2048, 4096-pont FFT.....	56
Figure 5.1	Circuit of synthesized scan D flip-flop.....	60
Figure 5.2	Block diagram of the two-input register array	60
Figure 5.3	Various structure of D flip-flops.....	62
Figure 5.4	Average current under low-clock-transition cases	63
Figure 5.5	Average current under high-clock-transition cases.....	63
Figure 5.6	SNR comparison	67
Figure 5.7	Power consumption for various FFT sizes (110MHz, 1.2V).....	67
Figure 5.8	Power distribution characteristics.....	68
Figure 5.9	Energy dissipation per FFT operation, (a) in normal scale, (b) i n log scale	69
Figure 5.10	Comparison of Energy dissipation between RMR FFT and the other reconfigurable pipeline-based architecture.....	71
Figure 5.11	Comparison of Energy dissipation between RMR FFT and the other reconfigurable memory-based architecture	71
Figure 5.12	Layout and schematic view of the 1-bit D flip-flop.....	72
Figure 5.13	Layout and schematic view of a basic block in RB_512.....	73
Figure 5.14	Layout and schematic view of RB_512	74
Figure 5.15	Layout and schematic view of RB_4096	75
Figure 5.16	Layout and schematic view of the 1-bit D flip-flop.....	76
Figure 5.17	Overall Layout view of the proposed RMR FFT	77

Chapter 1

Introduction

1.1 Background

In discrete-time signal processing (DSP), engineers usually study and practice digital signals between time domain and frequency domain [1.1]. A sequence of samples from a measuring device produces a time or spatial domain representation, whereas a discrete Fourier transform (DFT) produces the frequency domain information, that is, the frequency spectrum. As many communications theories are based on frequency domain, the DFT becomes an important component.

However, the direct mapping of DFT equation into a physical implementation results in unacceptable hardware overhead. The fast Fourier transform (FFT) is thus developed to make the implementation possible. FFTs became popular after J. W. Cooley of IBM and John W. Tukey of Princeton published a paper in 1965 [1.2] reinventing the algorithm and describing how to perform it conveniently on a computer. FFTs are of great importance to a wide variety of applications, from digital signal processing to solving partial differential equations to algorithms for quickly multiplying large integers.

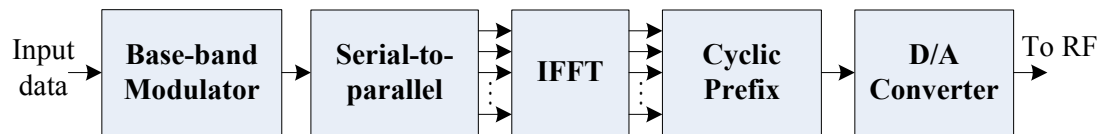
The performance of FFT is often the bottle neck of a DSP system. The design of a high-speed FFT processor has been an important topic for many years. Various architectures have been proposed to serve different applications. Recently, the popularity of portable systems raises the low-power consumption as another serious design issue. The demand for low-power and high-speed FFT processors never stops.

1.2 Motivation

Many recent communication standards propose the orthogonal frequency division multiplexing (OFDM) as the primary modulation method. A general block diagram of an OFDM system is shown in Figure 1.1. The FFT and inverse FFT

(IFFT), which are essential for such modulation, are both computation-intensive and data-exchange-intensive. Many FFT algorithms and architecture have been proposed to drive the performance further in the past decades. However, modern communication standards require even faster FFT processors while the power-consumption is critical. For example, in the popular orthogonal frequency-division multiple (OFDM)-based UWB systems, the execution time of the 128-point FFT/IFFT is only 312.5 ns, or equivalent 409.6Msample/s [1.3].

Transmitter



Receiver

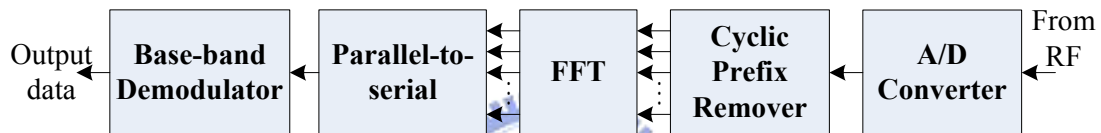


Figure 1.1 Generic OFDM block diagram

On the other side, it is desirable for a processor to perform flexible-size FFTs, thereby facilitating software adaptability when different formats and changing standards must be accommodated. Processors with high re-configurability incur inevitable overhead in all terms. In order to minimize the overhead, the design of such reconfigurable processors must be considered from both algorithm-level and architecture-level.

This thesis aims to design a high performance FFT/IFFT processor that can meet modern high-speed criteria while maintaining low power consumption. The processor can be flexible to perform different lengths of FFTs and thus suitable for various protocols and applications. The FFT length should be easily reconfigured by setting control registers and with minimum hardware overhead possible.

1.3 Organization of Thesis

The rest of this thesis is organized as follow. Chapter 2 is a review of general FFT algorithms and architectures. The basic concept of the FFT algorithm is

explained and various FFT algorithms are introduced here. Also, popular FFT architectures in implementation, memory-based and pipeline-based, are depicted and compared in this chapter. In conclusion, we will give a direction of algorithms and architecture that is most suitable for modern high-speed applications.

In this thesis, we propose an energy-aware reconfigurable mixed-radix FFT/IFFT. The proposed processor can be easily reconfigured as from 16-point to 4096-point FFT/IFFT with proper mixed-radix algorithm assigned for each mode. In chapter 3, we will derive the proposed reconfigurable mixed-radix algorithm. The architecture design and principle of each block will be illustrated in chapter 4.

In chapter 5, the RMR FFT is implemented using TSMC 0.13 μ m technology. As will be shown in the proposed architecture, we find that the internal storage block takes out most of the FFT area and power during the cell-based synthesis flow. The implementation strategy of the internal storage blocks is different from that of the rest RMR FFT. The simulation result will be analyzed and compared with other reconfigurable architectures. Finally, some conclusions and future work will be presented in Chapter 6.



Chapter 2

Review of FFT Algorithms and Architectures

2.1 Introduction

The discrete Fourier transform (DFT) is widely employed in the analysis, design, and implementation of signal processing algorithms and systems. However, the computational complexity of direct evaluation of an N-point DFT is $O(N^2)$, which results in a long computation time and excessive hardware cost. Fortunately, considerable symmetry exists in the operations and coefficients required to compute a DFT. Such symmetry can be exploited to reduce the number of operations required, thus reducing the time required for DFT computation. Collectively, the resulting efficient computation algorithms are called fast Fourier transform (FFT).

Mainly, the FFT is a way of computing the DFT by decomposing the computation into successively smaller DFT computations. In this process, both the symmetry and the periodicity of the complex exponential $W_N^{nk} = e^{-j(2\pi/N)nk}$ are exploited. Algorithms in which the decomposition is based on the input sequence $x[n]$ into successively smaller subsequences are called decimation-in-time (DIT) algorithms. Alternatively, we can consider dividing output sequence $X[k]$ into smaller subsequences and such algorithms are called decimation-in-frequency (DIF) algorithms.

By far the most common FFT is the Cooley-Tukey algorithm [2.1], which is suitable in decomposing DFT that is of size of power of 2. We would like to introduce some variants based on Cooley-Tukey algorithm in this chapter. These variants can be classified as fixed-radix and the others, respectively. Also, we will discuss the architectures for these algorithms in VLSI implementation. Both of the two popular architectures, memory-based and pipeline-based, have their advantages and certain shortcomings.

2.2 Basic Concept of FFT Algorithms

The discrete Fourier transform of a complex data sequence $x[n]$ of length N is defined as:

$$X(k) = \sum_{n=0}^{N-1} x[n]W_N^{nk} \quad k=0,1,\dots,N-1 \quad (2.1)$$

where the coefficient W_N^{nk} is defined as $W_N^{nk} = e^{\frac{-j2\pi nk}{N}}$, which are called twiddle factors. The approach used to improve the efficiency in FFT is to exploit the symmetry and the periodicity properties of W_N^{nk} ;

$$W_N^{(N-n)k} = W_N^{-nk} = (W_N^{nk})^* \quad (\text{Symmetry property}) \quad (2.2)$$

$$W_N^{nk} = W_N^{n(k+N)} = W_N^{(n+N)k} \quad (\text{Periodicity in } n \text{ and } k) \quad (2.3)$$

As an illustration, using the periodicity property, we can group terms in Eq. (2.1) for n and $(n+N)$:

$$x[n]W_N^{nk} + x[n+N]W_N^{(n+N)k} = (x[n] + x[n+N])W_N^{nk} \quad (2.4)$$

Similar groupings can be used for other terms in Eq. (2.1). In this way, the number of complex multiplication can be reduced by approximately a factor of 2. We can also take the advantage of the fact that for certain factors, the real and imaginary parts take on the value 1 or 0, which eliminating the need for multiplication. As a result, applying the above properties achieves significantly reduction in computation.


The Cooley-Tukey algorithm is the most common FFT algorithm. It re-expresses the DFT of an arbitrary composite size $N = N_1N_2$ in terms of smaller DFTs of sizes N_1 and N_2 recursively. FFT algorithms are based on the fundamental principle of decomposing the computation of the DFT of an N -length sequence into successively smaller DFT. The manner of how this principle is implemented leads to a variety of different algorithms. In the following section, various FFT algorithms will be introduced.

2.3 The FFT Algorithms

According to the manner of decomposition, the FFT algorithms can be classified as DIT and DIF algorithms. The difference is the object to be decomposed, input sequence for DIT and output sequence for DIF.

2.3.1 Decimation-in-Frequency (DIF) Fixed-Radix Algorithms

The principle of the decimation-in-frequency algorithm is most conveniently illustrated by considering the N -point DFT where N is an integer power of 2, i.e., $N=2^v$. Since N is an even integer, we can consider computing the even-numbered frequency samples and odd-numbered frequency samples separately. Referring to Eq. (2.1), we can express $X(k)$ as:

$$\begin{aligned}
 X(k) &= \sum_{n=0}^{N-1} x[n]W_N^{nk} \\
 &= \sum_{n=0}^{\frac{N}{2}-1} x[n]W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x[n]W_N^{nk} \\
 &= \sum_{n=0}^{\frac{N}{2}-1} x[n]W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x[n + \frac{N}{2}]W_N^{(n+\frac{N}{2})k} \\
 &= \sum_{n=0}^{\frac{N}{2}-1} x[n]W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x[n + \frac{N}{2}]W_N^{\frac{N}{2}k} W_N^{nk} \\
 &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + x[n + \frac{N}{2}]W_N^{\frac{N}{2}k} \right\} W_N^{nk}
 \end{aligned} \tag{2.5}$$


Based on the above equation, the even-numbered frequency samples are:

$$\begin{aligned}
 X(2r) &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + x[n + \frac{N}{2}]W_N^{\frac{N}{2}2r} \right\} W_N^{n2r} \\
 &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + x[n + \frac{N}{2}] \right\} W_N^{nr}
 \end{aligned} \tag{2.6}$$

The result of Eq. (2.6) can be seen as the $N/2$ -point DFT of the sequence $\{x[n] + x[n + N/2]\}$, which is obtained by adding the first half and the second half of the

input sequence. In the same way, the odd-numbered frequency points are:

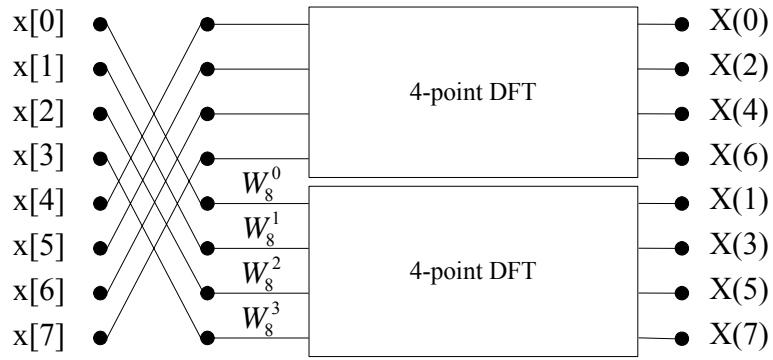
$$\begin{aligned}
 X(2r+1) &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + x\left[n + \frac{N}{2}\right] W_N^{\frac{N}{2}(2r+1)} \right\} W_N^{n(2r+1)} \\
 &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] - x\left[n + \frac{N}{2}\right] \right\} W_N^n W_N^{nr}
 \end{aligned} \tag{2.7}$$

Eq. (2.7) is then the $N/2$ -point DFT of the sequence obtained by subtracting the second half from the first half of the input sequence and multiplying the resulting sequence by W_N^n . Therefore, the problem of computing N -point DFT becomes

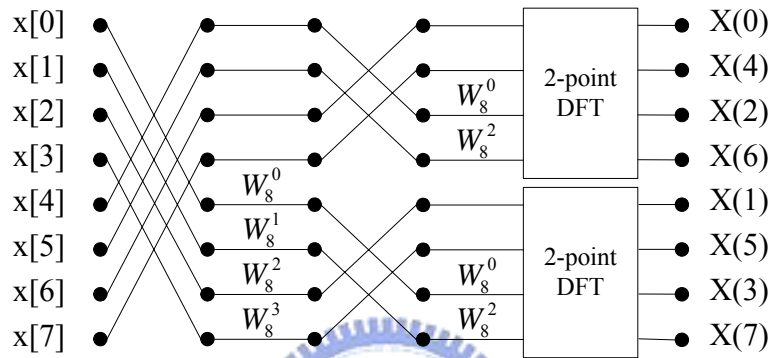
computing $N/2$ -point DFT. Recursively, we can further decompose the $N/2$ -point DFT in Eq. (2.6) and (2.7) into smaller DFT. Proceed with these decomposition until the only DFT required are 2-point DFTs. The 2-point DFT can be derived as the simple form in Eq. (2.6) and (2.7), which are multiplication and addition/subtraction operations. As a result, the computation of N -point DFT requires no real DFT computation but only multiplication and addition/subtraction operations.

Figure 2.1, which is called a signal flow graph (SFG), illustrates the procedure of decomposing the 8-point DFT by the DIF algorithm. First we decompose the 8-point DFT as combinations of two 4-point DFT according to Eq. (2.6) and (2.7), as shown in (a). We can see now the output frequency points have been separated into even-numbered and odd-numbered parts. We then divide the 4-point DFT, respectively, into 2-point DFTs. Again, the output frequency points are separated. For the sequence $\{X(0), X(2), X(4), X(6)\}$, the even-numbered points are $\{X(0), X(4)\}$ and the odd-numbered points are $\{X(2), X(6)\}$. The flow graph then becomes (b). Finally, we decompose the 2-point DFTs further and obtain the flow graph in (c). As we can see, the demand of any DFT block is now eliminated.

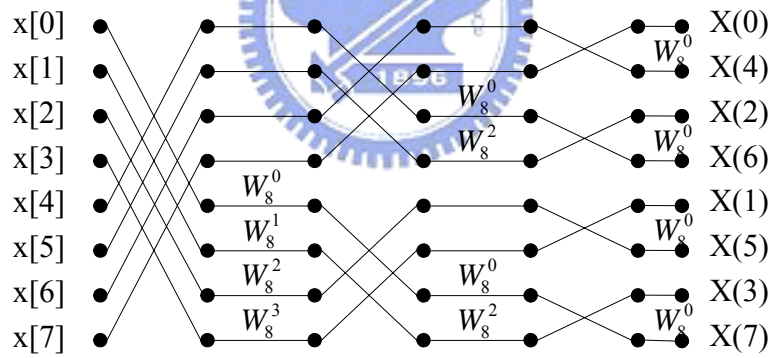
The basic computation unit in the flow graph of Figure 2.1, as brought up in Figure 2.2, is called a butterfly. The butterfly output in DIF algorithms have to multiply certain constants and such constants are called twiddle factors. This basic computation unit is effectively a 2-point DFT unit, as can be seen from (b) and (c) of Figure 2.1. Since the N -point DFT is always divided by 2 recursively, the above algorithm is called the radix-2 DIF algorithm.



(a)



(b)



(c)

Figure 2.1 Decomposition of the 8-point DFT step by step in DIF algorithm

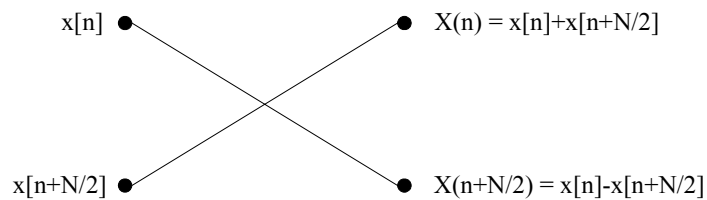


Figure 2.2 The butterfly unit of radix-2 DIF FFT

Further more, the output ordering, as shown in the SFG, is not in normal order as the time-domain input. In fact, the order which the output data present is referred to as bit-reversed order. The idea of the bit-reversed order can be well depicted by tree diagrams. As we take the 8-point DFT as an example, three binary digits are required to index through the data. Figure 2.3 shows the way how normal order and bit-reversed order are derived, respectively. In (a), the normal order is obtained through sorting data sequence by successive examination of the data index bits. In (b), the same procedure takes place to obtain the bit-reversed order except that the data index bits examination is backward.

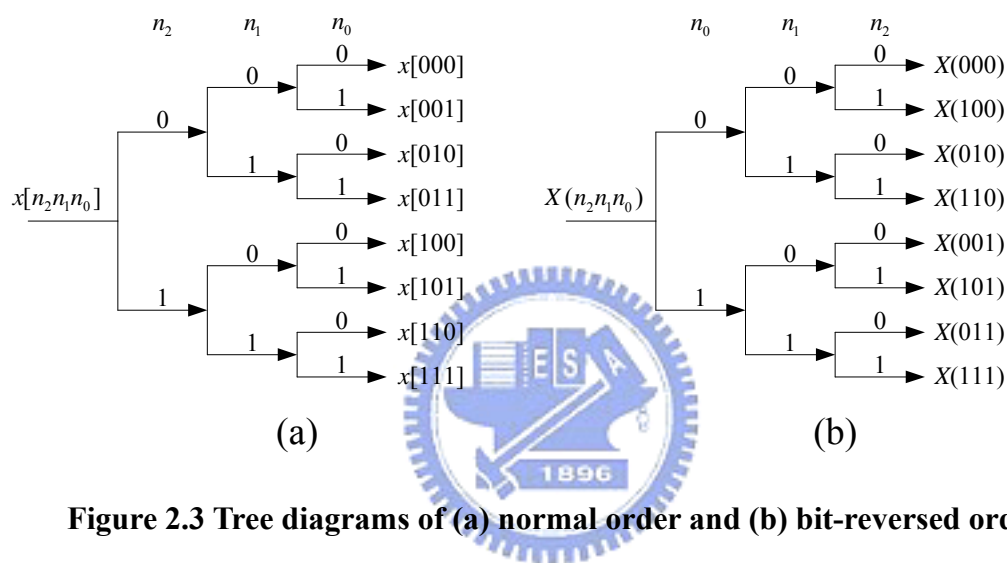


Figure 2.3 Tree diagrams of (a) normal order and (b) bit-reversed order

Similar to the way of decomposing the even integer N , we can decompose N into four parts if N is an integer power of 4, i.e., $N=4^v$. We can divide frequency samples into four parts and consider computing them separately. The equation represents these four frequency parts are thus:

$$X(4r) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] + x\left[n + \frac{N}{4}\right] + x\left[n + \frac{2N}{4}\right] + x\left[n + \frac{3N}{4}\right] \right\} W_{\frac{N}{4}}^n \quad (2.8)$$

$$X(4r+1) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] + x\left[n + \frac{N}{4}\right] W_4^1 + x\left[n + \frac{2N}{4}\right] W_4^2 + x\left[n + \frac{3N}{4}\right] W_4^3 \right\} W_N^n W_{\frac{N}{4}}^n \quad (2.9)$$

$$X(4r+2) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] + x\left[n + \frac{N}{4}\right] W_4^2 + x\left[n + \frac{2N}{4}\right] W_4^4 + x\left[n + \frac{3N}{4}\right] W_4^6 \right\} W_N^{2n} W_{\frac{N}{4}}^n \quad (2.10)$$

$$X(4r+3) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] + x\left[n + \frac{N}{4}\right]W_4^3 + x\left[n + \frac{2N}{4}\right]W_4^6 + x\left[n + \frac{3N}{4}\right]W_4^9 \right\} W_N^{3n} W_{\frac{N}{4}}^n \quad (2.11)$$

A decomposition of a 4^v -point DFT can also be shown through a signal flow graph, similar to the one in Figure 2.1. This time, the basic computation unit is no longer a 2-point DFT butterfly but a 4-point DFT butterfly, as shown in Figure 2.4. The resulting algorithm, therefore, is called a radix-4 DIF algorithm.

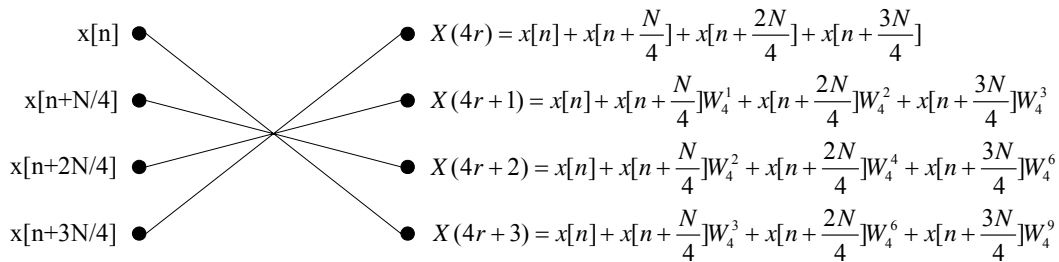


Figure 2.4 The butterfly unit of radix-4 DIF FFT

Practicing the above decomposition procedures, we can further derive even higher radix- r DIF algorithms by restricting N as an integer power of r . The advantage of a higher radix algorithm is that the number of complex multiplications can be effectively lowered. As one radix-4 stage corresponds to two radix-2 stage in the SFG, the twiddle-factor multiplications between the two radix-2 stages are now covered in the radix-4 stage. As shown in Figure 2.4, complex multiplications in the radix-4 butterfly, multiplication by $\{W_4^0, W_4^1, W_4^2, W_4^3\}$, are thought as trivial multiplications. This means that these multiplications can be carried without a true multiplier. Therefore, the effective number of complex multiplication required in radix-4 algorithm is fewer than that in radix-2 algorithm. Accordingly, algorithms with higher radix are more efficient than those with lower radix in arithmetic aspect. On the other hand, the butterfly of a higher radix algorithm is more complicated. The trade-off is between addition/subtraction and multiplications. Since addition/subtractions are of lower computational complexity than multiplications in complex-number computation, the higher radix algorithms are usually preferred. However, the radix- r algorithm is only suitable for r^v -point FFT. For a DFT sequence of length not power of r , lower radix algorithm must be used.

2.3.2 Decimation-in-Time (DIT) Fixed-Radix Algorithms

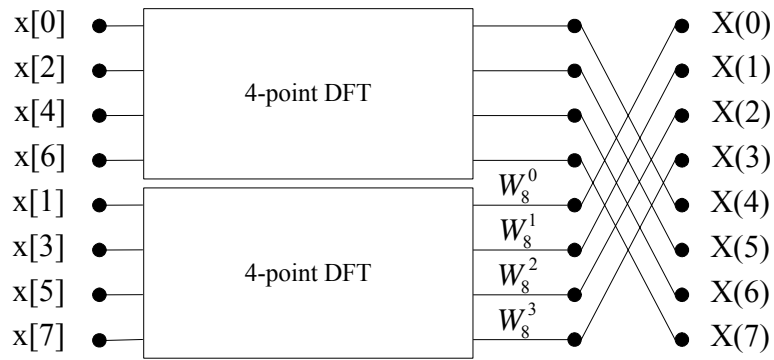
To develop the DIT algorithm, let us again consider the N -point DFT where N is an integer power of 2, i.e., $N=2^v$. Since N is an even integer, we can consider computing $X(k)$ by separating $x[n]$ into the even-numbered points and odd-numbered points. With the $X(k)$ given in Eq. (2.1), we can derive the following equation:

$$\begin{aligned}
 X(k) &= \sum_{n=0}^{N-1} x[n]W_N^{nk} \\
 &= \sum_{r=0}^{\frac{N-1}{2}} x[2r]W_N^{2rk} + \sum_{r=\frac{N-1}{2}}^{N-1} x[2r+1]W_N^{(2r+1)k} \\
 &= \sum_{r=0}^{\frac{N-1}{2}} x[2r]W_N^{2rk} + \sum_{r=0}^{\frac{N-1}{2}} x[2r+1]W_N^k W_N^{2rk} \\
 &= \sum_{r=0}^{\frac{N-1}{2}} x[2r]W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N-1}{2}} x[2r+1]W_{\frac{N}{2}}^{rk}
 \end{aligned} \tag{2.12}$$

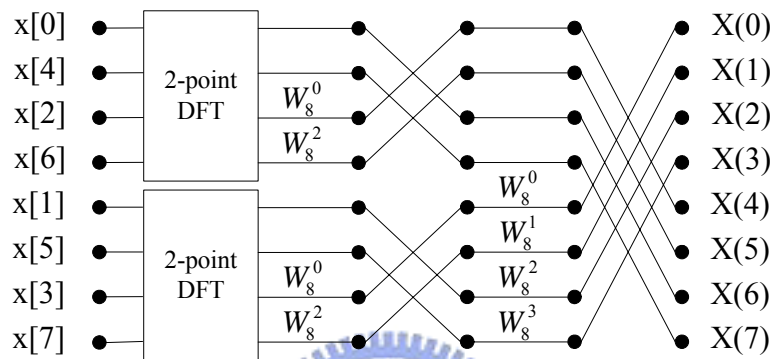
In the above equation, $X(k)$ can be seen as a combination of the DFT of the even-numbered points and odd-numbered points of $x[n]$. Replace them with $G(k)$ and $H(k)$, respectively:

$$\begin{aligned}
 X(k) &= \sum_{r=0}^{\frac{N-1}{2}} x[2r]W_{\frac{N}{2}}^{rk} + W_N^k \sum_{r=0}^{\frac{N-1}{2}} x[2r+1]W_{\frac{N}{2}}^{rk} \\
 &= G(k) + W_N^k H(k)
 \end{aligned} \tag{2.13}$$

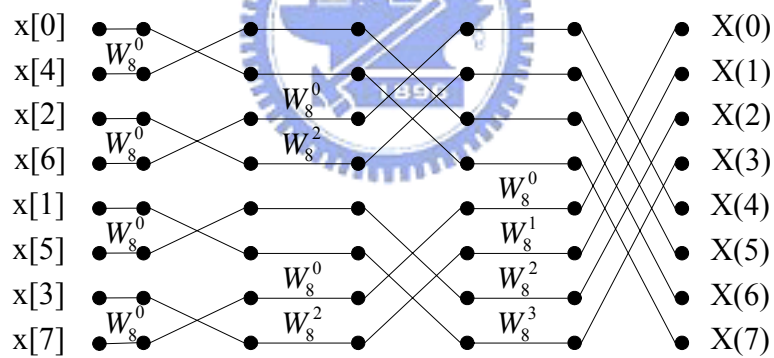
$G(k)$ represents the $N/2$ -point DFT of the even-numbered points in $x[n]$ and $H(k)$ represents the $N/2$ -point DFT of the odd-numbered points in $x[n]$. We can then treat $G(k)$ as an independent DFT and decompose it as the manner in Eq. (2.12). Recursively, $G(k)$ will finally be decomposed into 2-point DFTs, which is multiply-and-add operation of two data. In the same way, $H(k)$ can also be recursively decomposed into combinations of 2-point DFTs. A 2-point DFT, according to Eq. (2.13), is a multiply-and-add operation. Therefore, the N -point DFT can be calculated without any real DFT computations.



(a)



(b)



(c)

Figure 2.5 Decomposition of the 8-point DFT step by step in DIT algorithm

Figure 2.5 shows the procedure of how an 8-point DFT is composed by the DIT algorithms. First we decompose the 8-point DFT as combinations of two 4-point DFT according to Eq. (2.8) and (2.9), as shown in (a). We can see now the time-domain input points have been separated into even-numbered and odd-numbered parts. We then divide the 4-point DFT, respectively, into 2-point DFTs. Again, the input points are separated. For the sequence $\{x[0], x[2], x[4], x[6]\}$, the even-numbered points are

$\{x[0],x[4]\}$ and the odd-numbered points are $\{x[2],x[6]\}$. The flow graph then becomes (b). Finally, we decompose the 2-point DFTs further and obtain the flow graph in (c). At last, the demand of any DFT block is now eliminated.

Similar to the DIF algorithm, the basic butterfly unit of the DIT algorithm is shown in Figure 2.6(a). However, be aware of the fact that:

$$W_N^{r+N/2} = W_N^r W_N^{N/2} = -W_N^r \quad (2.14)$$

The butterfly is modified as in (b), which reduces the number of multiplications to 1. This basic computation unit is also effectively a 2-point DFT unit, as can be seen from (b) and (c) of Figure 2.5. Therefore, the above algorithm is called a radix-2 DIT algorithm.

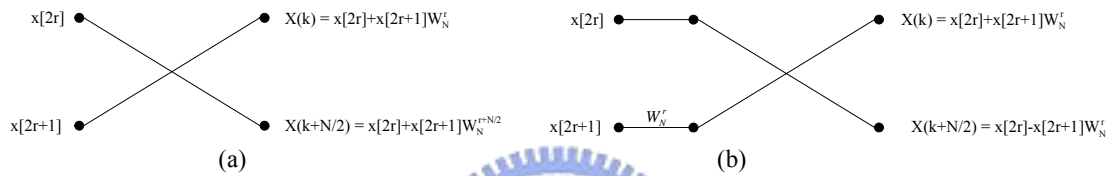


Figure 2.6 The butterfly unit of radix-2 DIT FFT

Observing Figure 2.5, the time-domain input for the DIT decomposition are in bit-reversed order while the frequency-domain output are in normal order. Comprehensively, the SFG of the DIT algorithm is a reverse of the SFG of the DIF algorithm. We can also use the same methods as in previous section to derive a higher radix decomposition of the DIT algorithm.

2.3.3 Other FFT Algorithms

There are many other variations on the Cooley-Tukey algorithm. **Mixed-radix** implementations [2.2-2.5] handle composite sizes with a variety of (typically small) factors in addition to two, usually (but not always) employing the $O(N^2)$ algorithm for the prime base cases of the recursion. The idea of mixed-radix algorithms is straightforward. As the fixed-radix algorithms recursively decompose the N -point DFT into N/r -point DFT, we can also decompose the N -point into N/r_1 -point, N/r_2 -point..., and N/r_m -point DFTs as long as $N = r_1 \times r_2 \dots \times r_m$.

Split radix [2.6-2.8] merges radices 2 and 4, exploiting the fact that the first transform of radix-2 requires no twiddle factor, in order to achieve the lowest known arithmetic operation count for power-of-two sizes. The DIF split-radix 2/4 algorithm decomposes the frequency sample as:

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x[n] + x\left[n + \frac{2N}{4}\right] \right\} W_{\frac{N}{2}}^{nk} \quad (2.15)$$

$$X(4k+1) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] - x\left[n + \frac{2N}{4}\right] - j \left[x\left[n + \frac{N}{4}\right] - x\left[n + \frac{3N}{4}\right] \right] \right\} W_N^n W_N^{4nk} \quad (2.16)$$

$$X(4k+3) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x[n] - x\left[n + \frac{2N}{4}\right] + j \left[x\left[n + \frac{N}{4}\right] - x\left[n + \frac{3N}{4}\right] \right] \right\} W_N^{3n} W_N^{4nk} \quad (2.17)$$

The SFG of the split-radix algorithm can also be drawn as the fixed-radix algorithms. Figure 2.7 shows the basic butterfly unit for split-radix 2/4 algorithm. The split-radix algorithm features low computational complexity and is flexible as radix-2 algorithm.

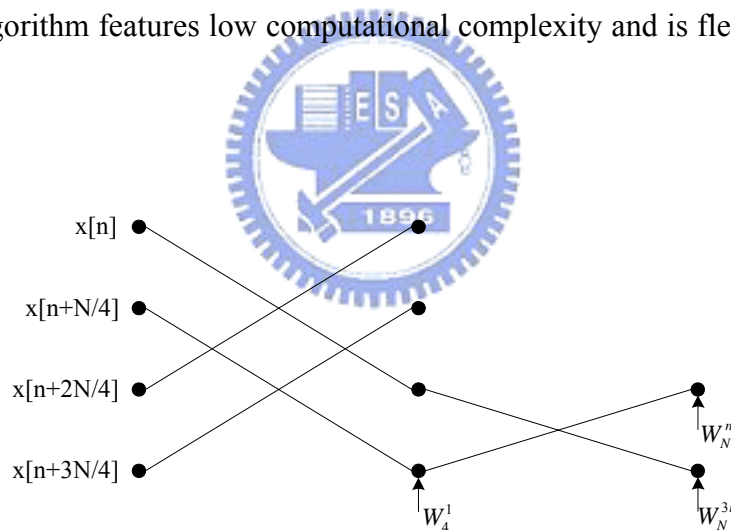


Figure 2.7 The butterfly unit of split-radix 2/4 algorithm

2.4 The FFT Architecture

The FFT architecture is the way to implement the signal flow graph of the FFT algorithms. In this section, we will introduce the FFT architectures which are common for VLSI implementation. There are two popular architectures to implement the FFT algorithms for real time applications. They are pipeline-based architecture and memory-based architecture.

2.4.1 Pipeline-Based Architecture

The pipeline-based architecture is of high regularity and can be easily scaled and parameterized in implementation [2.6, 2.8-2.15]. Compared to the memory-based architecture, it is characterized in high throughput rate while keeping moderate hardware complexity. An efficient method to obtain the pipeline architecture is to project the signal flow graph of the FFT algorithm to the hardware data flow. Two common pipeline-based architectures will be introduced next, the single-path delay feedback (SDF) and the multiple-delay commutator (MDC) architecture.

2.4.1.1 Single-Path Delay Feedback (SDF) Architecture

The block diagram of the SDF architecture in radix-2 DIF algorithm is shown in Figure 2.8. For the FFT length $N = 16$, there will be 4 butterfly stages in the SFG. As we can see from the figure, a butterfly element is dedicated to each stage. The feedback registers are used to store output data of the butterfly outputs. The butterfly element perform the butterfly operation when the required data are ready at the input ports, otherwise it perform the swap operation to store data into the feedback registers. The memory requirement of the SDF architecture is minimal. However, the utilization rate of the butterfly and multiplier units is only 50%.

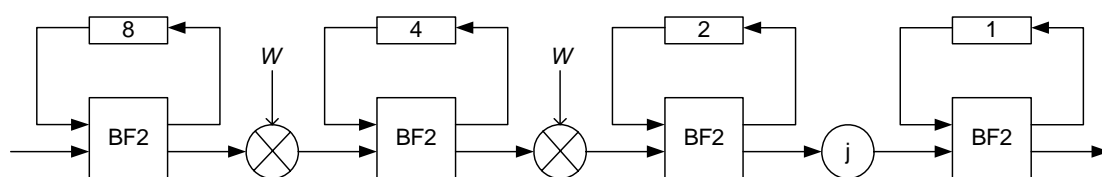


Figure 2.8 Radix-2 DIF SDF architecture for $N = 16$

Similar to the radix-2 SDF architecture, the SDF architecture for the radix-4 algorithm can also be derived from the SFG. Figure 2.9 shows the case when the SDF architecture is applied to the radix-4 algorithm. Compared to the radix-2 architecture, the radix-4 architecture can implement the FFT with fewer computation stages. However, the butterfly unit will be more complicated.

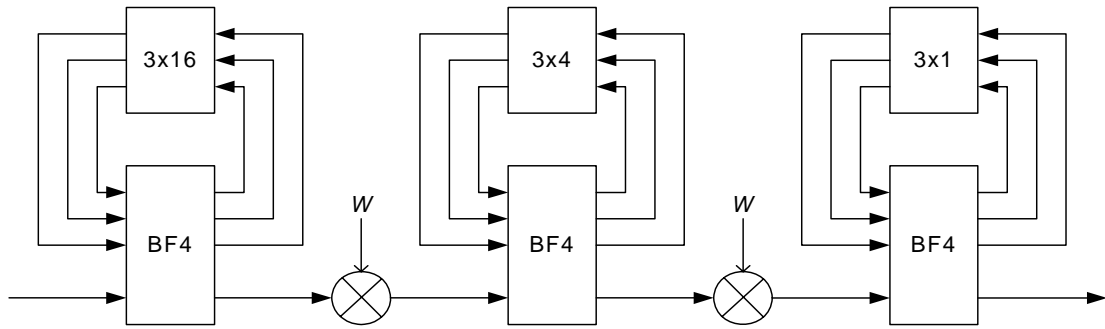


Figure 2.9 Radix-4 DIF SDF architecture for $N = 64$

2.4.1.2 Multiple-Path Delay Commutator (MDC) Architecture

The MDC approach is even more straightforward than the SDF approach. As the butterfly in the SFG, parallel data paths are used in the architecture. Instead of using the delay feedback registers, delay elements are placed on the data paths. Between each computation stages, a commutator is used to switch data to correct positions. Figure 2.10 shows the block diagram of the radix-2 DIF MDC architecture. The throughput rate of the radix-2 MDC architecture is twice that of the radix-2 SDF architecture due to the parallel data paths. However, the memory requirement is larger than that of the SDF architecture and also extra commutators are required.

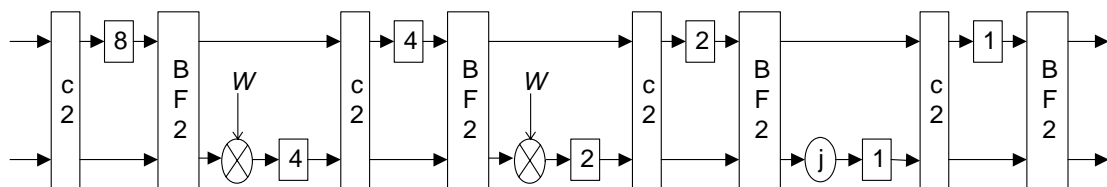


Figure 2.10 Radix-2 MDC architecture for $N = 16$

The radix-4 MDC architecture is of the same principle as the radix-2 one. Figure 2.11 shows the block diagram of the radix-4 MDC architecture for $N = 64$. In the radix-4 MDC architecture, higher throughput rate can be achieved due to the four parallel data paths. However, more memory requirement and higher hardware complexity are the overhead in return.

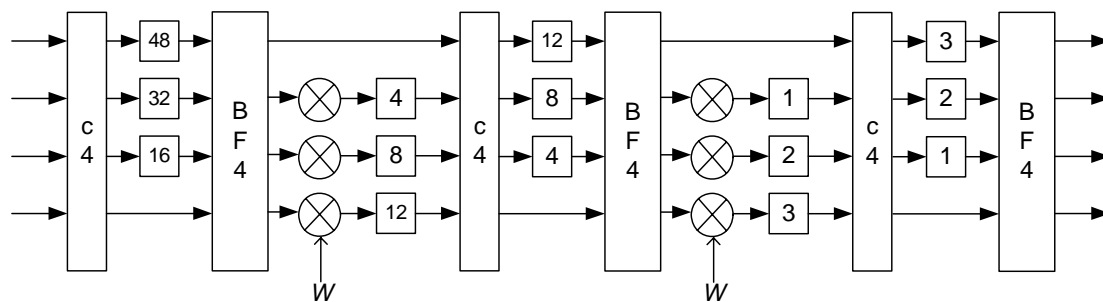


Figure 2.11 Radix-4 DIF MDC architecture for $N = 64$

2.4.2 Memory-Based Architecture

The memory-based architecture is considered the most area efficient way of implementing the FFT [2.2, 2.4-2.5, 2.16-2.19]. It usually consists of one computation block, coefficient memory for twiddle factors, and memory to store IO and internal data. The feature of such architecture is that it usually uses only one or few butterfly elements as the computation block. Since the butterflies and multipliers usually take out most area and consume large power in the pipeline-based architecture, the memory-based architecture reduces such hardware cost and thus lowers the power consumption. Figure 2.12 shows the generic block diagram of the memory-based architecture. The hardware complexity of the memory-based architecture concentrates on the control block. Since there are only one or few butterfly elements available, the execution order is stage by stage as in the SFG. The memory-based architecture usually uses one memory module to store the intermediate data. Since the data ordering is different from stage to stage, the order of data stored in the memory must be taken care after every stage of operation

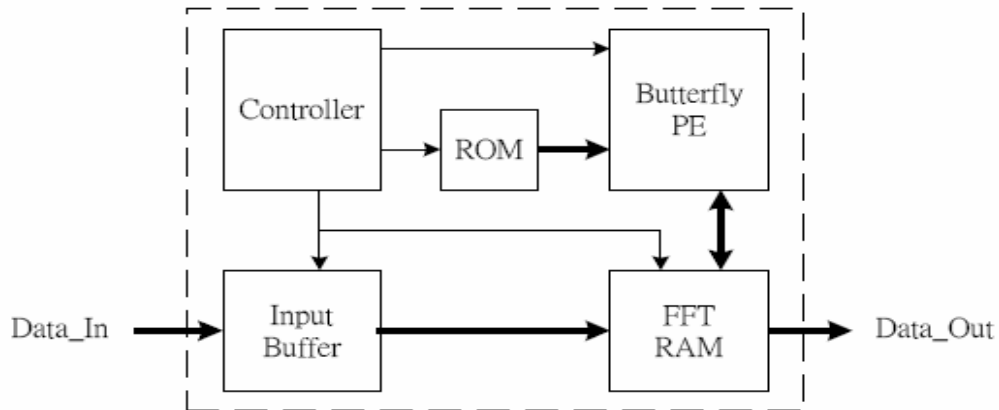


Figure 2.12 Block diagram of the memory-based architecture

As the number of butterfly units available reduces, the number of butterfly on the SFG remains the same. Therefore, the memory-based architecture results in low throughput rate. In a radix- r algorithm, an N -point FFT requires $\frac{N}{r} \times \log_r N$ radix- r butterfly operation. Assume that the memory access bandwidth is K and the time for a butterfly operation is t . Then, the time to compute an N -point FFT can be expressed as:

$$\text{Time for one FFT} = \frac{N}{r} \times \log_r N \times \frac{r}{K} \times t = \frac{N}{K} \times \log_r N \times t \quad (2.18)$$

From the above equation, it can be seen that the time for one FFT can be reduced linearly with K and exponentially with r . Therefore, using high radix algorithms is an efficient way to raise the throughput rate of a memory-based architecture.

2.4.3 Reconfigurable Architecture

A FFT processor that can perform various lengths of FFT is usually preferred. For the pipeline-based architecture, the reconfiguration can be easily achieved. Recall the principle of the FFT algorithms. The idea is to break the N -point DFT into smaller DFTs recursively. Therefore, after a radix- r butterfly stage, the N -point FFT is decomposed into r N/r -point FFTs. This relation can be observed from the SFG as

previously shown in Figure 2.1 or Figure 2.5. Since the pipeline-based architecture is the projection of the SFG, the backend stages actually calculate the FFTs of smaller sizes. Therefore, the pipeline-based architecture can be reconfigured for calculating FFT of smaller size by feeding input data directly into later stages [2.3, 2.20].

However, such reconfiguration does require lots of multiplexers when we demand higher flexibility in the FFT size. The multiplexers added between each stage not only increase the overhead on area and power, but also influence the speed performance of overall architecture. Figure 2.13 shows an example of the reconfigurable pipeline-based architecture. The 1024-point FFT architecture is divided into five stages ($1024=4^5$). The architecture can also be reconfigured as 16, 64, or 256-point FFT. Reconfiguration is achieved by inserting three multiplexers namely MUX I, MUX II, and MUX III. The FFT processor can act as a 256-point processor by feeding the input data directly into stage 2 and clocking down the first stage. In the same way, reconfigurations to 64-point or 16-point FFT can also be achieved by feeding input data directly into stage 3 or stage 4, respectively.

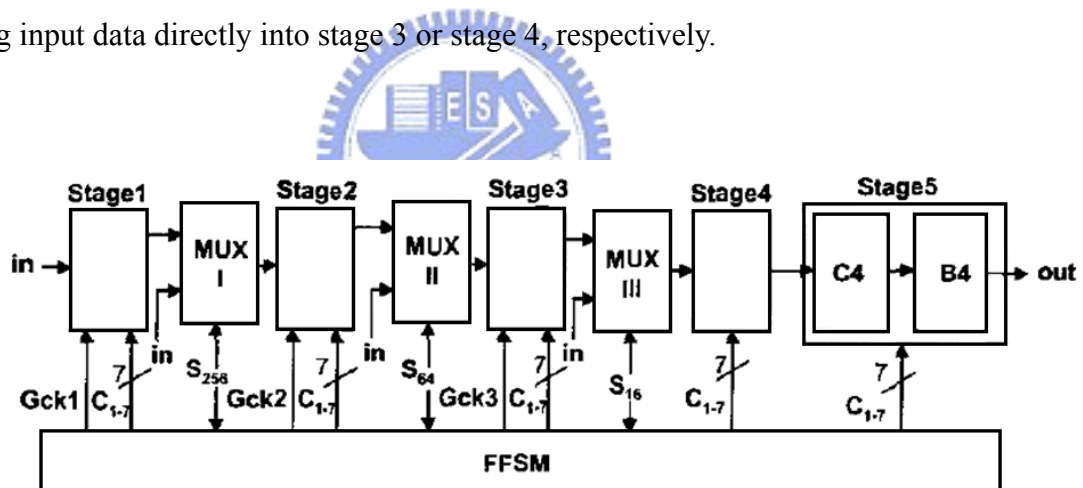


Figure 2.13 Architecture of 1024-point radix-4 reconfigurable pipelined FFT processor

Alternatively, the memory-based architecture can be modified as reconfigurable architecture [2.21-2.22], too. Unlike the pipeline-based architecture, no much hardware needs to be added since there is only one butterfly computation block. Reconfigurability is achieved by adding flexibility to address generation block, coefficient memory block, and data memory block. The difficulty lies on the generation of control signal and the data ordering in the memory.

2.5 Conclusions

In this chapter, we have reviewed the generic FFT algorithms and architectures. The fixed-radix algorithms are popular in VLSI implementation due to the regularity of their SFGs. However, while algorithms with high radix are of lower computational complexity, the flexibility in FFT size is also limited. The mixed-radix algorithms are thus more suitable for decomposing various FFT sizes. The drawback is that their twiddle-factor multiplications are more irregular than fixed-radix algorithms.

In the architecture level, the memory-based architecture which only uses one or few computation blocks, is considered the most area efficient architecture. However, the low throughput rate makes it unsuitable for the high-speed application. The pipeline-based architecture is easy to scale and parameterize in hardware design. Although it is also easy to reconfigure for different FFT size, the data path may grow too long if we want higher flexibility.



Chapter3

Algorithm of

Reconfigurable Mixed-Radix FFT

3.1 Introduction

Our purpose is to design a reconfigurable FFT processor that can be dynamically configured to perform FFT length as from 16-point to 4096-point. In the fixed-radix algorithms, only radix-2 FFT algorithms can cover this range of reconfiguration. However, the radix-2 algorithms result in large calculation cycles and low throughput rate. As the higher radix algorithms are preferred for our high throughput purpose, the flexibility of the FFT size is also limited. Therefore, the mixed-radix algorithm is adopted in our design to keep the architecture flexible while using a high radix algorithm. Also, the algorithm should have certain common properties for decomposing different points of FFT.

In this chapter, we will derive a reconfigurable mixed-radix algorithm. We manage to find regularity for data ordering and twiddle factors for FFTs of different sizes. Such regularities facilitate the construction of the hardware architecture. Also, special block execution order for the RMR FFT will be introduced in order to adopt the block-floating-point method.

3.2 Reconfigurable Mixed-Radix Algorithm

The Discrete Fourier Transform (DFT) of a complex data sequence $x[n]$ of length N is defined as

$$X(k) = \sum_{n=0}^{N-1} x[n]W_N^{nk} \quad k=0,1,\dots,N-1 \quad (3.1)$$

where the coefficient W_N^{nk} is defined as $W_N^{nk} = e^{\frac{-j2\pi nk}{N}}$, which is called twiddle factors.

A direct implementation of this equation requires large hardware and thus is impractical. By using the FFT algorithm, the computational complexity can be reduced. Let

$$\begin{aligned}
N &= 2^v = r_1 \times r_2 \\
n &= n_1 + r_1 n_2, \quad \begin{cases} n_1 = 0, 1, \dots, r_1 - 1 \\ n_2 = 0, 1, \dots, r_2 - 1 \end{cases} \\
k &= r_2 k_1 + k_2, \quad \begin{cases} n_1 = 0, 1, \dots, r_1 - 1 \\ n_2 = 0, 1, \dots, r_2 - 1 \end{cases}
\end{aligned} \tag{3.2}$$

Combining (1) and (2), the N -point FFT can be formulated as

$$\begin{aligned}
X(r_2 k_1 + k_2) &= \sum_{n_1=0}^{r_1-1} \sum_{n_2=0}^{r_2-1} x[n_1 + r_1 n_2] W_N^{(n_1 + r_1 n_2)(r_2 k_1 + k_2)} \\
&= \sum_{n_1=0}^{r_1-1} \left\{ \underbrace{\sum_{n_2=0}^{r_2-1} x[n_1 + r_1 n_2] W_{r_2}^{n_2 k_2}}_{r_2\text{-point DFT}} \times \underbrace{W_N^{n_1 k_2}}_{\text{Twiddle factor}} \right\} W_{r_1}^{n_1 k_1}
\end{aligned} \tag{3.3}$$

From the above equations, we can divide any N -point (power of 2) FFT into a combination of r_1 -point and r_2 -point FFT. The r_2 -point DFT in (3) can be further decomposed in the same manner. Therefore, when N is not prime, such that $N = r_1 \times r_2 \times r_3 \times \dots \times r_m$, it is possible to divide the N -point DFT as combination of $r_1, r_2, r_3, \dots, r_m$ -point DFTs.

The proposed RMR FFT is divided as four-stage pipeline architecture. The idea is that, if each butterfly unit can act as radix-2, radix-4, or radix-8 butterfly, then the processor is capable of performing different points of FFT algorithms ranging from $2 \times 2 \times 2 \times 2 = 16$ points to $8 \times 8 \times 8 \times 8 = 4096$ points. That is, decompose the N -point FFT as combination of r_1, r_2, r_3, r_4 -point DFT, where $N = r_1 \times r_2 \times r_3 \times r_4$. In this way, one FFT may have several combinations of radices. Since the duplications are unnecessary for the hardware design, specific mixed-radix algorithm is assigned for each FFT mode.

The higher radix is chosen first. Based on the radix-8 algorithm, smaller FFT sizes are realized by bypassing preceding stages. For example, the 512-point FFT can be decomposed by the radix-8 algorithm as three stages and the four-stage pipeline thus becomes unnecessary. In such cases, we would like to bypass one of the four stages as the conventional reconfigurable pipeline architecture does, instead of assigning an $8 \times 8 \times 4 \times 2$ algorithm or other four-stage decomposition. Radix smaller than 8 is arranged at last stage. In this way, we only have to consider the last stage as

a reconfigurable butterfly stage while other stages being radix-8 under all modes. The resulting radix arrangement is shown in TABLE 3.1. As the table shown, we only need four-stage butterflies when the FFT is of size {1024, 2048, 4096}. Meanwhile, FFTs of size {128, 256, 512} need three-stage butterflies and FFTs of size {16, 32, 64} need only two.

TABLE 3.1 Mixed-radix algorithms for different FFT sizes

FFT size	Stage 1	Stage 2	Stage 3	Stage 4
16			8	2
32			8	4
64			8	8
128		8	8	2
256		8	8	4
512		8	8	8
1024	8	8	8	2
2048	8	8	8	4
4096	8	8	8	8

The basic butterfly units in our design are thus radix-2, radix-4, and radix-8 butterflies. Based on the decimation in frequency decomposition, the SFG of the 8-point DFT is shown in Fig. 3.1. Notice that there is no explicit multiplication operation in realization of an 8-point DFT. The trivial multiplications of $\pm j$, $(1-j)/\sqrt{2}$, and $-(1+j)/\sqrt{2}$ can be realized by using only shift-and add operation. Another observation through the SFG is that the 8-point DFT is a combination of two parallel 4-point DFTs if we neglect the first stage and a combination of four parallel 2-point DFTs if the first two stages are neglected. Therefore, the radix-8 butterfly can serve as radix-4 and radix-2 butterfly as well. The side advantage is that the width of data path can stay at 8-data when goes from radix-8 to a lower radix stage.

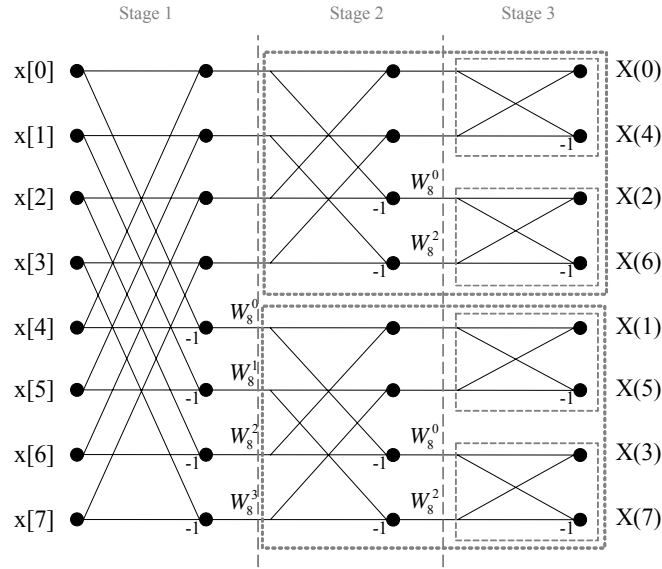


Figure 3.1 SFG of 8-point DIF FFT

Moreover, consider the DFT and IDFT equations:

$$\text{DFT: } X(k) = \sum_{n=0}^{N-1} x[n]W_N^{nk} \quad (3.4)$$

$$\text{IDFT: } x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X(k)W_N^{-nk} \quad (3.5)$$

Let

$$\begin{aligned} W_N^{nk} &= Wr + jWi \\ W_N^{-nk} &= Wr - jWi \end{aligned} \quad (3.6)$$

We can find that the differences between DFT and IDFT are (a) the scaling constant $1/N$ and (b) twiddle factors are conjugate of each other. Now, consider the complex multiplication of multiplying conjugate twiddle factors respectively:

$$(Xr + jXi)(Wr + jWi) = (XrWr - XiWi) + j(XiWr + XrWi) \quad (3.7)$$

$$(Xr + jXi)(Wr - jWi) = (XrWr + XiWi) + j(XiWr - XrWi) \quad (3.8)$$

If we swap the real and imaginary parts of the input variable, that is, changing $(Xr + jXi)$ to $(Xi + jXr)$, Eq. (3.7) becomes:

$$(Xi + jXr)(Wr + jWi) = (XiWr - XrWi) + j(XrWr + XiWi) \quad (3.9)$$

Comparing Eq. (3.8) and Eq. (3.9), the real part of Eq. (3.8) equals to the imaginary part of Eq. (3.9) and the imaginary part of Eq. (3.8) equals to the real part of Eq. (3.9). This means that these two equations are equal if we swap the real and imaginary parts

of one of the equations. Therefore, there is way to transform Eq. (3.7) to Eq. (3.8). Since Eq (3.7) represents the multiplications in DFT and Eq. (3.8) represents multiplications in IDFT, this means that we are able to use the DFT to calculate the IDFT.

In summary, the IDFT can be performed by first swap the real and imaginary parts of input data. Then, after the DFT computation, swap the real and imaginary parts of output data. By scaling the output with the constant $1/N$, the IDFT result is obtained. In the view of hardware implementation, we only have to add the swap unit at the input and output data port of the FFT processor in order to use the same processor to calculate IFFT.

3.3 Data Ordering and Twiddle Factors

As the proposed architecture can be reconfigured from 16 to 4096-point FFT, the data ordering will be different from mode to mode due to the dedicated mixed-radix algorithm. To make the architecture realizable, there must be rules that apply to all modes. The dedicated mixed-radix algorithms for different modes are listed in TABLE 3.1. The approach we use here is first to decompose the N -point FFT by the radix-2 decimation-in-frequency algorithm. As mentioned in the previous section, a radix-8 stage can be decomposed as combination of radix-2 stages or radix-4 stages. In other word, we can combine two radix-2 stages as one radix-4 stage and three radix-2 stages as a radix-8 stage, as shown in Figure 3.1. Based on signal flow graph of the radix-2 DIF decomposition, we recompose N -point FFT to the mixed-radix algorithms listed in TABLE 3.1. Since all the FFTs are decomposed by the radix-2 DIF algorithm, the data ordering follows the same rules. The order of output data for the radix-2 flow graph is referred to as bit-reversed order. Figure 3.2 shows the example of radix-2 decomposition of the 128-point FFT. As Table 3.1 listed, the assigned mixed-radix algorithm for 128-point is radix-8/8/2. Figure 3.3 shows the recomposed SFG, which is the desired SFG for our architecture. Notice that the black nodes in Figure 3.2 correspond to the nodes in Figure 3.3 respectively.

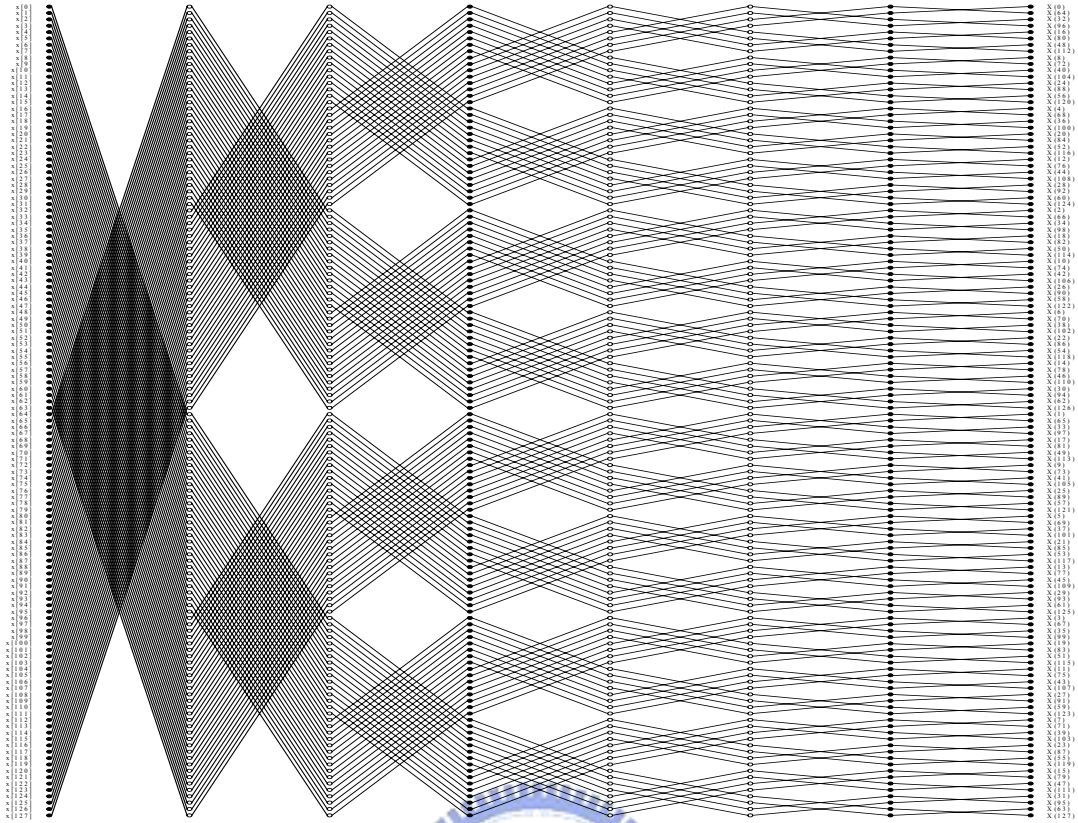


Figure 3.2 SFG of 128-point FFT in radix-2 DIF algorithm

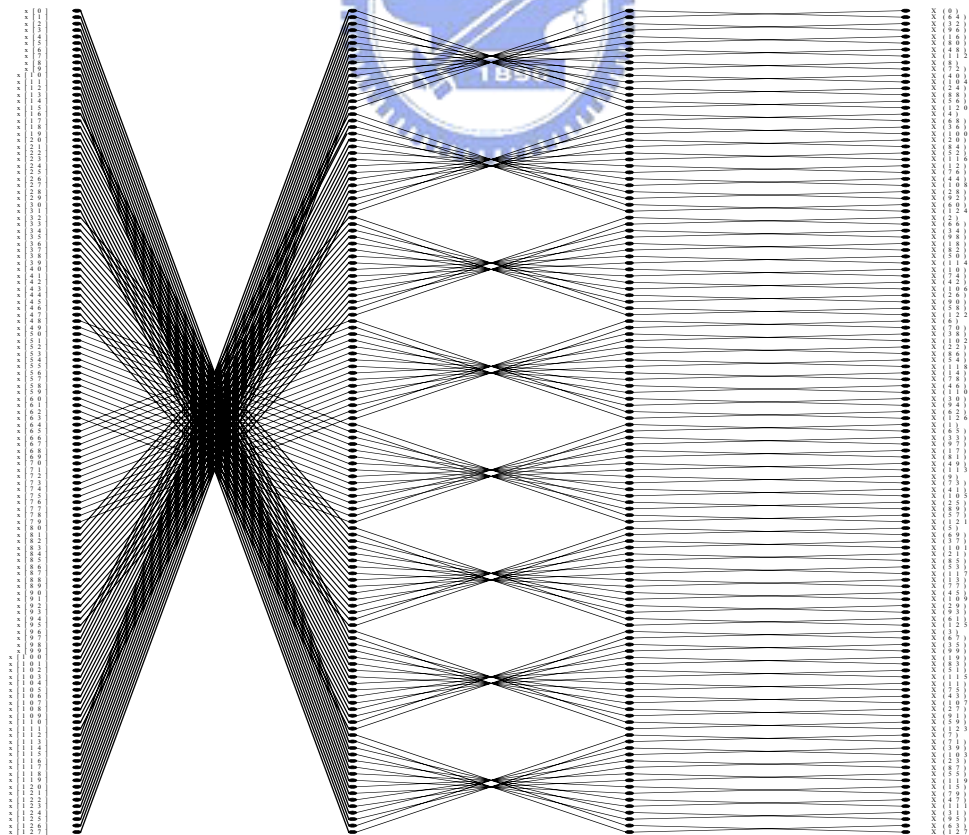
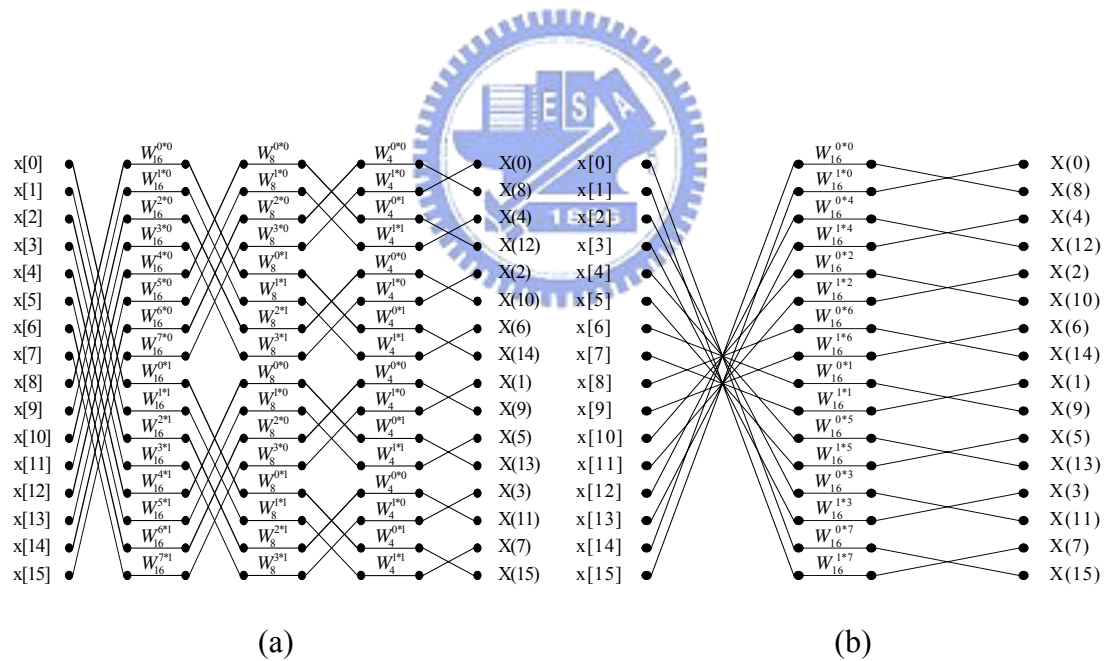


Figure 3.3 SFG of 128-point FFT in mixed-radix algorithm

After determine the data ordering for every butterfly stage, the next question is how the twiddle factors arrange. Clearly, it is not likely that we can directly map the twiddle factors from the radix-2 SFG to our mixed-radix SFG. However, we have found relation between that is easy enough for us to derive a common rule.

Start with the example of the 16-point FFT SFG and as mentioned before, first we draw the SFG using radix-2 algorithm, as shown in Figure 3.4(a). According to TABLE 3.1, the 16-point FFT is supposed to recombine as radix-8/2 butterfly stages and thus we know that the first three radix-2 stages should be combined as one radix-8 stage. Since there are 16 points, there will be two radix-8 butterflies and we extract them as in Figure 3.5. The first butterfly is readily a radix-8 butterfly as shown in Figure 3.1. For the second butterfly, we must transform the internal twiddle factors in order to map to Figure 3.1. The procedure is shown in Figure 3.6.



**Figure 3.4 SFG of 16-point DFT in
(a) radix-2 algorithm, and (b) radix-8/2 algorithm**

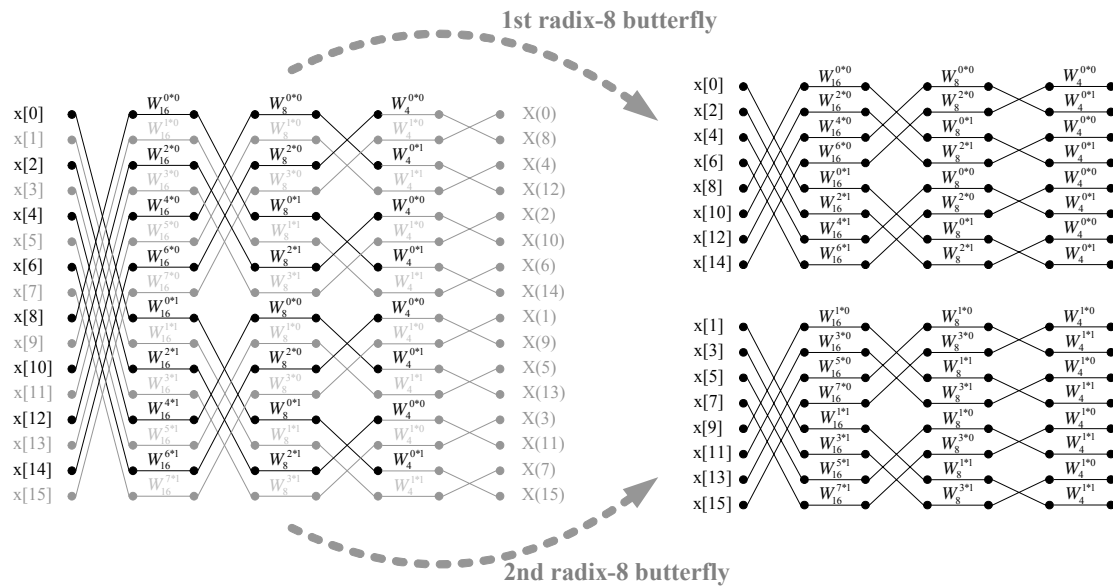


Figure 3.5 Extraction of radix-8 butterfly

The transformation starts with the first stage. In order to map the twiddle factors in the first column to those of the radix-8 butterfly, W_{16}^{-1} is multiplied to the twiddle factors, as shown in (a). Since the radix-2 butterfly performs only addition/subtraction operations, the output must multiply W_{16}^1 in order to compensate the multiplication at input. The procedure goes on through (b) and (c), and we can obtain the resulting SFG as in (d).

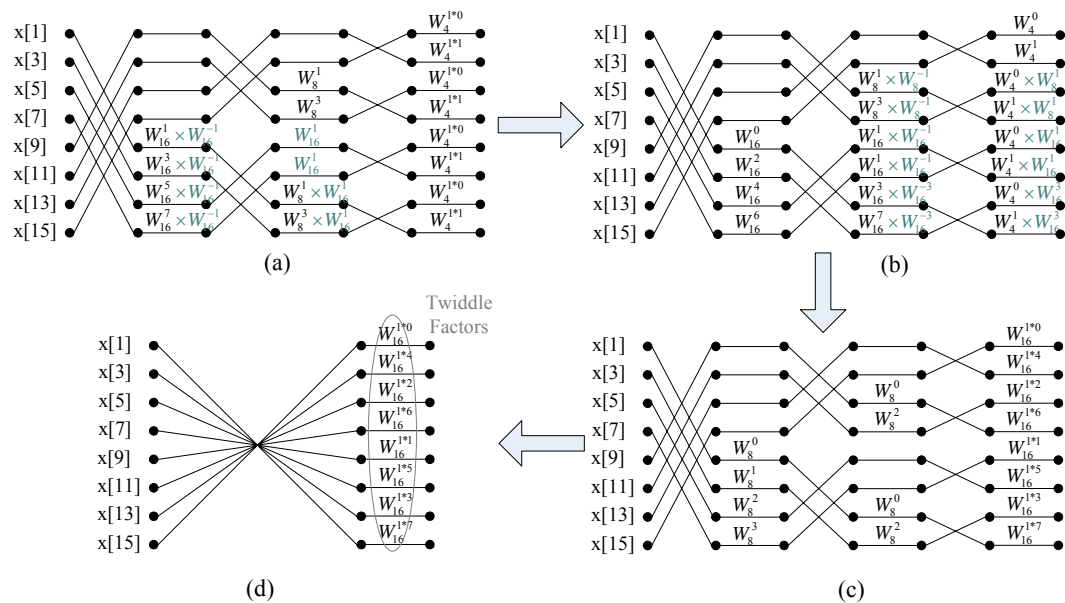


Figure 3.6 Procedure of combining three radix-2 stages into one radix-8 stage

We can then conclude the formula for the twiddle factors scheduling. For the N -point radix-8 decomposition, there will be $N/8$ butterflies. The twiddle factors for the m^{th} radix-8 butterfly are $\{W_N^{m*0}, W_N^{m*4}, W_N^{m*2}, W_N^{m*6}, W_N^{m*1}, W_N^{m*5}, W_N^{m*3}, W_N^{m*7}\}$, where m is a integer from 0 to $(N/8)-1$. The relation is shown in Figure 3.7. We will later find that such relation greatly simplify the control for the multiplier stage. Therefore, we can say that, for the N -point radix-8 decomposition, the complex multiplications required are of N -based twiddle factors. As our reconfigurable FFT may maximally perform 4 BF-stage operations, three multiplier stages are required. For these three multiplier stages, the possible N -based twiddle factors required are shown in TABLE 3.2.

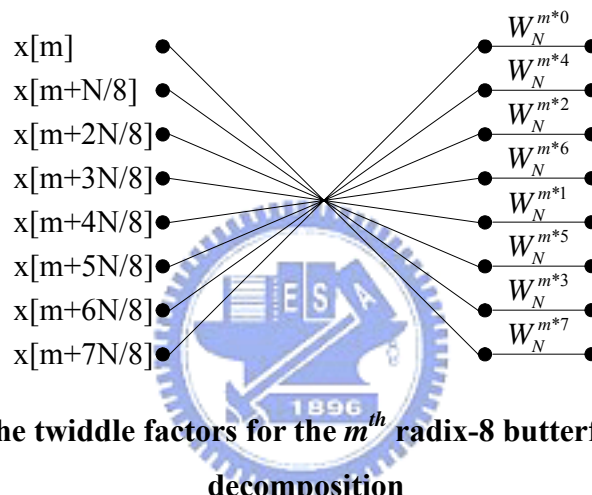


Figure 3.7 The twiddle factors for the m^{th} radix-8 butterfly for N -point decomposition

TABLE 3.2 N -based twiddle factors required for each multiplier stage under different FFT size

FFT size	Stage 1	Stage 2	Stage 3
16			16
32			32
64			64
128		128	16
256		256	32
512		512	64
1024	1024	128	16
2048	2048	256	32
4096	4096	512	64

3.4 Finite Register Length Effect and Block-Floating-Point Method

For physical circuit implementation, an exact precision of computation is usually not possible due to the effects of finite register length. For example, in implementing an FFT algorithm with fixed-point arithmetic we must ensure against overflow. For every fixed-point addition, M bits for example, $(M+1)$ -bit registers are required to store the result. However, the result will eventually be rounded to M bits and therefore generate quantization errors. The analysis of signal quality can be measured by the signal-to-quantization noise ratio (SNR), where each step of rounding reduces the SNR accordingly.

An analysis of the effects is given in [3.1]. The analysis goes under the assumption that the FFT length N is power of 2 and radix-2 decomposition is applied. The simplified result shows that the signal-to-noise ratio decrease as N^2 , or 1 bit per stage. That is, after every radix-2 butterfly stage, 1 bit must be added to the register length in order to maintain the same noise-to-signal ratio.

There several methods to maintain the signal-to-noise in FFT implementations. For general pipelined-based architectures, the common way is to use a larger internal wordlength than the input data wordlength, either increasing the internal wordlength gradually or keeping it fixed. This method is useful when FFT length is small. However, when the FFT length becomes large, such as 1024-point or larger, considerable overhead on circuitry comes out. The increase of wordlength affects not only the number of register bits but also the size of computation circuitry. Another drawback of this approach is that it is not suitable for a reconfigurable architecture. As the general reconfigurable architecture introduced in sec 2.4.3, when the processor is going to perform a shorter length FFT, preceding stages are often bypassed. As a result, circuitry with large wordlength may be used to calculate a small FFT.

In the proposed architecture, the Block Floating Point (BFP) method is used to minimize the quantization error [3.2-3.4]. The concept of BFP is that: The incoming data are partitioned into non-overlapping blocks, and depending upon the data sample with the highest magnitude in each block, a common exponent is assigned to the block. As illustrated in Figure 3.8, the original block is normalized to the word with largest magnitude in the block and a scaling factor k is obtained. Then the fixed-point computation proceeds with the normalized data. When all the data in this block is done computation, the whole block of data are shift back to the original precision point according to the scaling factor previously obtained. Block-floating-point outperforms fixed-point, since its input signals are always block normalized.

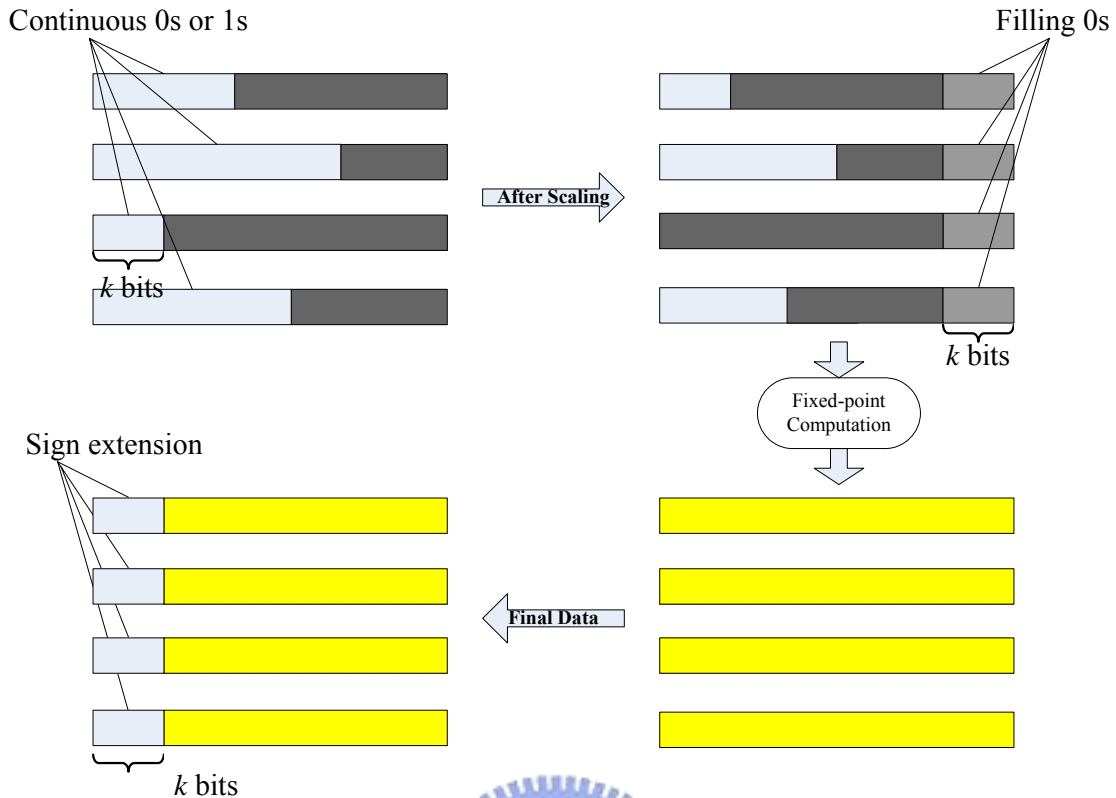


Figure 3.8 Concept of block-floating-point

The difficulty of using the BFP in a pipeline-based architecture is that the execution order is not processed stage by stage as the memory-based architecture. The general pipeline-based architecture starts the next stage calculation as soon as the available data arrive. In order to adopt the BFP, first we have divided the execution into blocks. Figure 3.9 shows an example of how the blocks will be arranged when 128-point FFT is calculated in our approach. The data will be divided into r group after a radix- r butterfly stage. The required data for butterfly stages afterward only come from the previously block. In the example, the data are separated as B-0 ~ B-7 after the first radix-8 butterfly stage. The calculation beginning from B-0 will only involved the data of B-0. That is, the operation of block C-0 ~ C-7 will need no data from B-1 ~ B-7. Here, we call B-0 as the supply block of C-0 ~ C-7 and A-0 as the supply block of B-0 ~ B-7.

To adopt the BFP method, the execution order of blocks thus follows two rules. First, the execution of certain block will not start before their supply block is finished. Secondly, the execution order of each stage is from top to bottom as in the SFG.

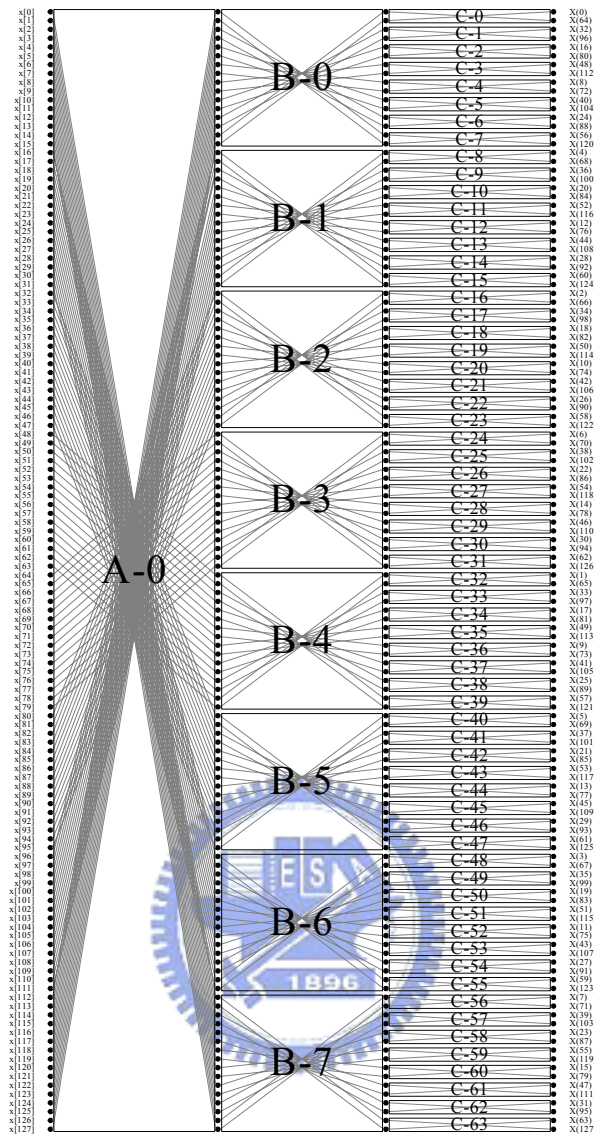


Figure 3.9 Blocks decomposition of 128-point FFT

Now, we can apply the BFP method. After the calculation of a certain block, the data coming out from this block will be evaluated to obtain a scaling factor. According to this factor, the data will be scaled when they go into next stage operation. The scaling factors are stored as a table during execution. The final scaling factor for the output will be the sum of scaling factors of all its supply blocks. The final scaling factors are required to shift back the data to the coordinate precision as the FFT input. Namely, output scaling factor for $X(0)$ is the sum of scaling factors of block A-0 and B-0, and the output scaling factor for $X(4)$ is the sum of scaling factors of block A-0 and B-1, etc..

Accordingly, the number of storage elements required to store the intermediate data between each stage is related to the block size respectively. TABLE 3.3 shows

the required storage elements between stages for each FFT size. Take the 128-point FFT for example, as the SFG in Figure 3.9. There are three computation stages for the 128-point FFT in our mixed-radix algorithm. For the first stage, the 128 data must be computed before any computation of the second stage starts. Therefore, 128 storage elements are required between first two stages. In the second stage, the 128 points are divided into eight 16 points by the radix-8 decomposition, and now the block size becomes 16 data. Again, 16 storage elements are required between second and the last stage.

TABLE 3.3 Storage elements required between each stage

FFT size	Stage 1&2	Stage 2&3	Stage 3&4
16			16
32			32
64			64
128		128	16
256		256	32
512		512	64
1024	1024	128	16
2048	2048	256	32
4096	4096	512	64

The actual number of storage elements required in the implementation of RMR FFT architecture, however, is a little different from TABLE 3.3. This is due to the design style of “Register Banks”, the internal storage modules. The detail will be explained in section 4.3.3.

3.5 Conclusions

In this chapter, we have derived the reconfigurable mixed-radix algorithm for the proposed FFT. For FFTs of length from 16 points to 4096 points, different mixed radix algorithms are assigned. In the proposed algorithm, we will need a reconfigurable butterfly stage that can act as radix-8, radix-4, or radix-2 butterfly accordingly. We also show the approach to calculate the IFFT using the FFT architecture.

By establishing the SFG through the radix-2 decomposition, we manage to find

certain rules in the way of data ordering. Such rules will be helpful for the architecture design. The twiddle factors are also clarified over various FFT sizes. Regularities are found in the twiddle factors for the 8-data-path architecture.

In the issue of signal-to-noise ratio, general pipeline-based architectures use larger internal wordlength to maintain a reasonable SNR. In our RMR FFT, the block-floating point approach is used to maintain the data accuracy. The procedure is first to identify data blocks on the SFG and then execute the butterfly computations according to block ordering. The utilization of BFP greatly changes the execution order in our RMR FFT compared to other pipeline-base architecture.

As our purpose is to construct a reconfigurable processor with high throughput rate, we may already draw out the RMR FFT architecture as in MDC structure. However, there are more to consider, as will be discussed in next chapter



Chapter 4

Architecture of

Reconfigurable Mixed-Radix FFT

4.1 Introduction

The design environment is shown in Figure 4.1. We will assume that existing input buffer has arranged the time domain input as 8 parallel data. The output of the FFT is of 8 parallel data as well. External power management unit are required for applying power gating on the proposed architecture.

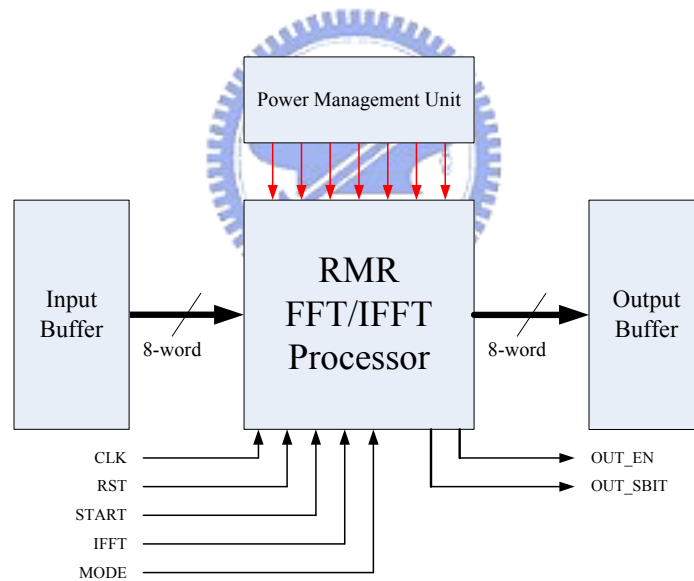


Figure 4.1 FFT environment

In this chapter, we will construct the architecture for the proposed RMR FFT and demonstrate the data flow in the processor. Section 4.2 first depicts the overall architecture. The detail design of each function module is given in section 4.3. Section 4.4 illustrates the data flow of the FFT processor under different mode. The conclusion is stated in section 4.5.

4.2 Overall Architecture

The overall architecture of the proposed RMR FFT is shown in Figure 4.2. The width of the data path is of 8 words. Each word represents a complex number data, which consists of real and imaginary part. The wordlength is 16-bit, and then the data path is of $8 \times 2 \times 16 = 256$ -bits. The proposed reconfigurable architecture is of four butterfly computation stages while the architecture in the figure shows only three. This is because the first two butterfly computation stages are combined as one. Observing TABLE 3.1, the first BF stage is enabled only during 1024, 2048, or 4096-point FFT. As block execution order explained in previous chapter, the first BF stage and the second BF stage in a signal flow graph will not overlap in calculation since the second BF stage will not start until the first BF stage is totally completed. Therefore, these two computation stages can share the same hardware.

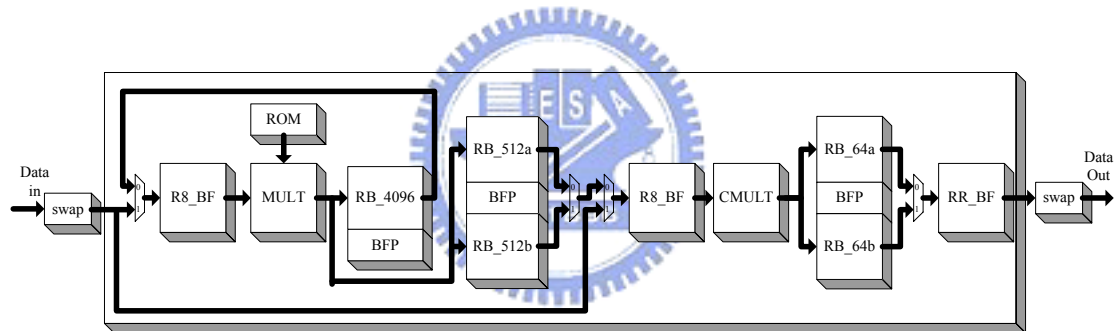


Figure 4.2 Block diagram of the proposed RMR FFT

As Table 3.1 suggests in section 3.2, only the last BF stage needs to be a reconfigurable butterfly (*RR_BF*) while the rest are radix-8 butterflies (*R8_BF*). We also have different strategies for the two multiplier stages, as *MULT* and *CMULT*. Between each computation stage, register banks (*RB*) are used to store and switch internal data. Unlike the traditional commutators in pipeline architecture [4.1-4.2], our internal *RB* is of more regular structure and easier to control. The construction and principle of each module will be introduced in next section.

4.3 Architecture Design

The detail architecture of each module will be explained in the following sub-sections.

4.3.1 Butterfly (BF) Unit

The butterfly blocks are designed based on the signal flow graph in Figure 3.1. In overall architecture, there will be two kinds of butterfly required. One is a general radix-8 butterfly block which performs an 8-point DFT operation. The other is the reconfigurable butterfly that can be reconfigured as radix-2, radix-4, and radix-8 butterfly respectively.

4.3.1.1 General BF

The general radix-8 butterfly is a direct implementation of SFG of the 8-point DFT (Figure 3.1). As explained earlier, the multiplications involved in an 8-point DFT are trivial, which are multiplication by $\pm j$, $(1-j)/\sqrt{2}$, and $-(1+j)/\sqrt{2}$. The multiplication of $\pm j$ is simply sign and real/imaginary part adjustment and multiplication of $1/\sqrt{2}$ can be implemented as in Figure 4.3. Therefore, these multiplications required only some shift-and-add, swap, and sign-changing operation. Without any true multiplier, it is possible to carry out the whole 8-point DFT in one clock cycle. The implementation diagram of a general BF is shown in Figure 4.4. The 8-point DFT is implemented in a fully parallel datapath by exactly following the SFG (Figure 3.1). The internal wordlength of these units is 16-bit, which is the same as input wordlength.

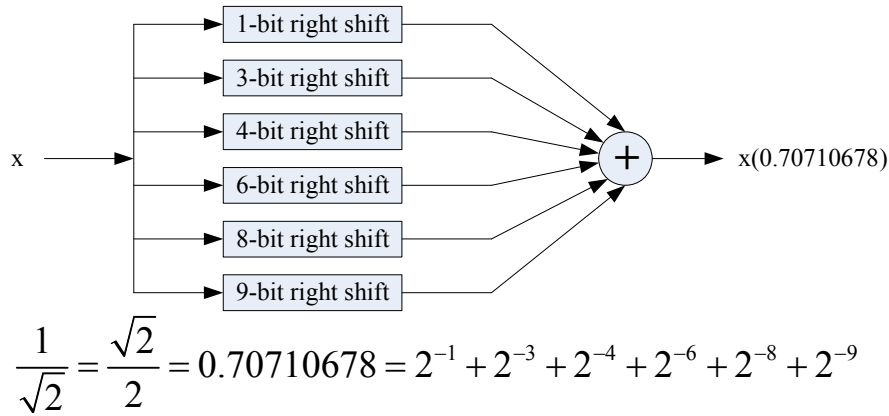


Figure 4.3 Circuit diagram of multiplication by $1/\sqrt{2}$

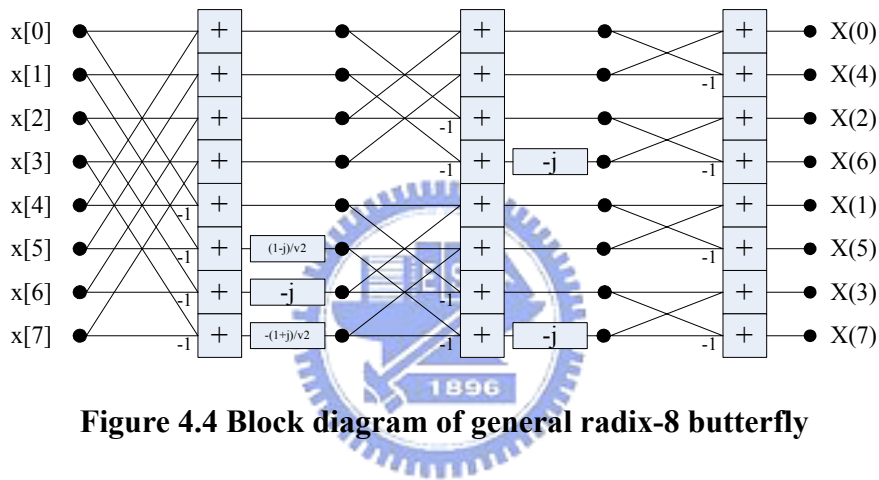


Figure 4.4 Block diagram of general radix-8 butterfly

4.3.1.2 Reconfigurable BF

The SFG of implementing the reconfigurable butterfly also refers to Figure 3.1. The block diagram of the reconfigurable BF is much like the general BF except that multiplexers are inserted every two stages. The block diagram of the reconfigurable is shown in Figure 4.5. For the three-stage partition, there will be two columns of multiplexers present, controlled by *ENA* and *ENB* respectively. The multiplexers select data from previous stage or the butterfly input. When the BF is going to act as a radix-8 butterfly, *ENA* and *ENB* are set to 0. When the two radix-4 BF combination is demanded, the input data should go directly into stage 2, and thus set *ENA* to 1 and *ENB* to 0. When the four radix-2 BF combination is needed, only the last stage is required for calculation and thus set *ENB* to 1. The relation between control signals and operation mode is listed in TABLE 4.1.

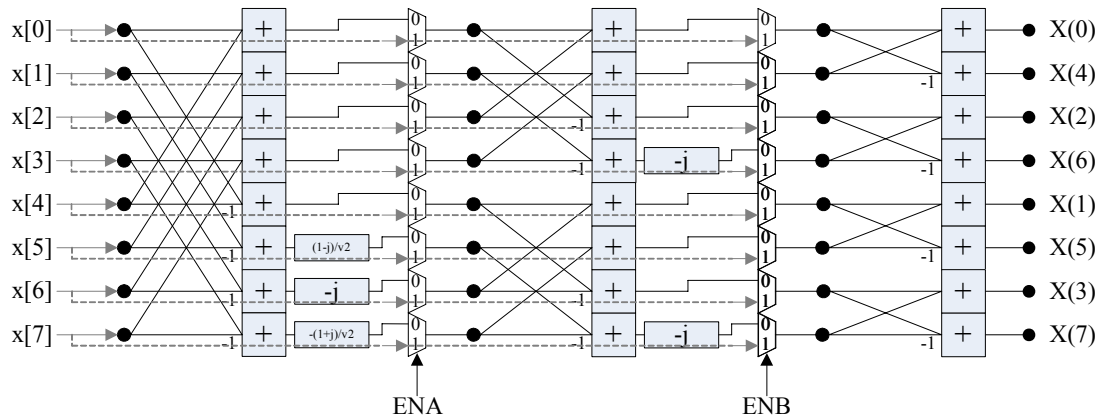


Figure 4.5 Block diagram of reconfigurable butterfly

TABLE 4.1 Truth table of control signals for reconfigurable BF

BF Mode	ENA	ENB
Radix-8 BF	0	0
2 parallel radix-4 BF	1	0
4 parallel radix-2 BF	X	1

4.3.2 Multiplier Stage

As the overall block diagram shows (Figure 4.2), there are two multiplier stages required in the proposed architecture. Here, we use two different strategies for these two multiplier stages respectively. The required twiddle-factor multiplications are verified in the previous chapter. For the multiplier stage between last two BF stages, we will use the strategy proposed in [4.3] to construct this multiplier stage, which is called the “constant multiplier” method. For the other multiplier stage, the traditional multiplier-ROM architecture is applied.

4.3.2.1 Constant Multiplier Approach

According to section 3.3, the twiddle factors for the N -point radix-8 decomposition are $\{W_N^{m*0}, W_N^{m*4}, W_N^{m*2}, W_N^{m*6}, W_N^{m*1}, W_N^{m*5}, W_N^{m*3}, W_N^{m*7}\}$, where m is an integer from 0 to $(N/8)-1$. For the multiplier stage between last two BF

stages, N could be 16, 32, or 64, as shown in TABLE 3.2. Therefore, possible twiddle factors are $W_{16}^{0\sim 15}$, $W_{32}^{0\sim 31}$, or $W_{64}^{0\sim 63}$. Notice that $W_{16}^k = W_{64}^{4k}$ and $W_{32}^k = W_{64}^{2k}$. As a result, the twiddle factors are in the range $W_{64}^{0\sim 63}$.

In the complex-number coordinates, we can map the twiddle factors to the unit circle. As shown in Figure 4.6, we can divide the unit circle into 8 equal regions, each containing $N/8$ twiddle factors ($N = 64$ in our case). Due to the symmetry of the unit circle, we can use only the twiddle factors in one of the regions to obtain the others by proper swapping and sign-changing operation.

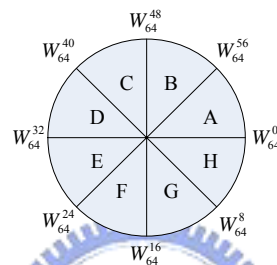


Figure 4.6 Twiddle factors on the unit circle

In the approach proposed in [4.3], it suggests that a twiddle factor multiplication of W_N^p , where $W_N^p = X + jY$, can be carried out by a set of constant multipliers (X, Y) instead of a complex multiplier. The twiddle factors required in this multiplier stage are W_{64}^p , where p ranges from 0 to 63. According to the above property, however, only nine sets of constants need to be implemented. Only twiddle factors in region A need to be implemented and the twiddle factors in other regions can be transformed through TABLE 4.2. In these nine sets of constants, one of them is $(0, 1)$, which is a trivial multiplication. The rest 8 sets of constants are listed in TABLE 4.3.

TABLE 4.2 Mapping table of the twiddle factors

Region	Real	Imaginary
A	X	Y
B	Y	X
C	-Y	X
D	-X	Y
E	-X	-Y
F	-Y	-X
G	Y	-X
H	X	-Y

TABLE 4.3 Implementation table of constants

Constant	Value of real part	Value of img. part	Realization of real part	Realization of img. part
Const1	0.995178	0.097961	$1-2^{-8}-2^{-10}+2^{-14}$	$2^{-4}+2^{-5}+2^{-8}+2^{-12}+2^{-14}$
Const2	0.980773	0.195068	$1-2^{-6}-2^{-8}+2^{-12}+2^{-14}$	$2^{-3}+2^{-4}+2^{-7}-2^{-12}$
Const3	0.956909	0.290283	$1-2^{-5}-2^{-7}-2^{-8}-2^{-13}$	$2^{-2}+2^{-5}+2^{-7}+2^{-10}+2^{-12}$
Const4	0.923828	0.382690	$1-2^{-4}-2^{-7}-2^{-8}-2^{-9}$	$2^{-1}-2^{-3}+2^{-7}-2^{-13}$
Const5	0.881896	0.471374	$1-2^{-3}+2^{-7}-2^{-10}+2^{-14}$	$2^{-1}-2^{-5}+2^{-9}+2^{-11}+2^{-12}-2^{-14}$
Const6	0.831420	0.555541	$1-2^{-3}-2^{-5}-2^{-6}+2^{-8}-2^{-11}-2^{-13}$	$2^{-1}+2^{-4}-2^{-7}+2^{-10}-2^{-13}$
Const7	0.773010	0.634399	$1-2^{-2}+2^{-6}+2^{-7}-2^{-11}+2^{-14}$	$2^{-1}+2^{-3}+2^{-7}+2^{-9}-2^{-11}+2^{-13}$
Const8	0.707092	0.707092	$2^{-1}+2^{-3}+2^{-4}+2^{-6}+2^{-8}+2^{-14}$	$2^{-1}+2^{-3}+2^{-4}+2^{-6}+2^{-8}+2^{-14}$

We can derive the scheduling of twiddle factors. TABLE 4.4 shows the scheduling of the twiddle factors in each data path after mapping to region A. The scheduling is first derived by the 64-base case. For the 64-base case, there will be 8 radix-8 butterflies and thus require 8 cycles. The twiddle factors are $\{W_{64}^{m*0}, W_{64}^{m*4}, W_{64}^{m*2}, W_{64}^{m*6}, W_{64}^{m*1}, W_{64}^{m*5}, W_{64}^{m*3}, W_{64}^{m*7}\}$ at cycle m . For example, twiddle factors at cycle 2 are $\{W_{64}^{2*0}, W_{64}^{2*4}, W_{64}^{2*2}, W_{64}^{2*6}, W_{64}^{2*1}, W_{64}^{2*5}, W_{64}^{2*3}, W_{64}^{2*7}\}$. After mapping through TABLE 4.2, they becomes $\{W_{64}^0, W_{64}^8, W_{64}^4, W_{64}^4, W_{64}^2, W_{64}^6, W_{64}^6, W_{64}^2\}$. For 32-base case, since $W_{32}^m = W_{64}^{2m}$, scheduling of twiddle factors for four cycles corresponding time slot $\{0, 2, 4, 6\}$ of the 64-base case. The same reason applies to the 16-base case.

TABLE 4.4 Scheduling of twiddle factors, W_{64}^p

Time slot \ Data path	0				1				
	0	1	2	3	4	5	6	7	
0 th	0	0	0	0	0	0	0	0	←16-base
1 st	0	4	8	4	0	4	8	4	←32-base
2 nd	0	2	4	6	8	6	4	2	←64-base
3 rd	0	6	4	2	8	2	4	6	
4 th	0	1	2	3	4	5	6	7	
5 th	0	5	6	1	4	7	2	3	
6 th	0	3	6	7	4	1	2	5	
7 th	0	7	2	5	4	3	6	1	

The multiplier stage is based on the constant multipliers shown in TABLE 4.3. Figure 4.7 shows the block diagram of one of such constant multipliers. The hard-wired circuitry is dedicated to perform multiplication by certain constant. The overall block diagram of the *CMULT* stage is shown in Figure 4.8. The input shuffle network routes the incoming data to the appropriate hard-wired constants and the output shuffle network will perform the appropriate sign-changing and swapping operation according to the mapping table in TABLE 4.2. Noticing the scheduling of TABLE 4.4, some constant multipliers may be duplicated in the same time slot. Therefore, there will be duplicate constant multiplier in the central multiplier bank, too. For example, there will be four *Const4* in the multiplier bank since there are maximally 4 *Const4* multipliers required at the same time (time slot 4 in TABLE 4.4).

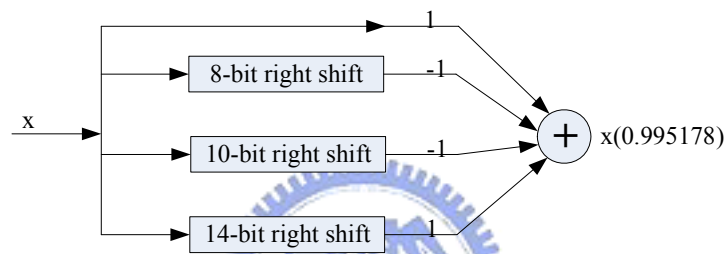


Figure 4.7 Block diagram of a constant multiplier

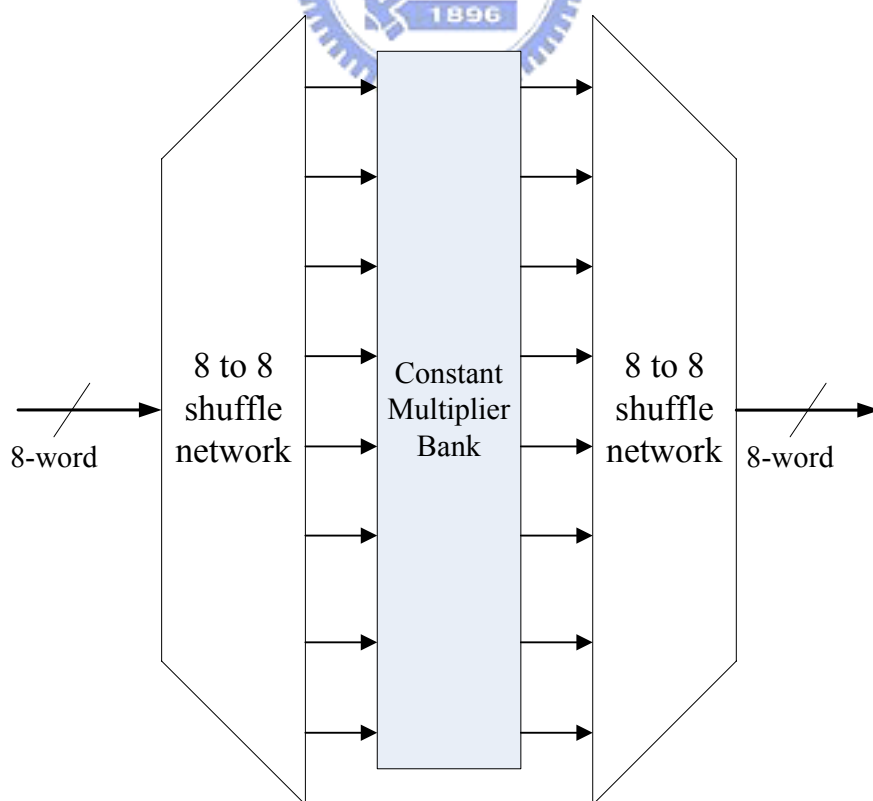


Figure 4.8 Block diagram of *CMULT* stage

4.3.2.2 Complex Multiplier Approach

For first two multiplier stages, possible twiddle factors are $W_{128}^{0\sim127}$, $W_{256}^{0\sim255}$, $W_{512}^{0\sim511}$, $W_{1024}^{0\sim1023}$, $W_{2048}^{0\sim2047}$, or $W_{4096}^{0\sim4095}$. Similar to before, we can say that the possible twiddle factors are in the range $W_{4096}^{0\sim4095}$. For this multiplier stage, we use the traditional multiplier-ROM approach. The block diagram of *MULT* is shown in Figure 4.9. Seven parallel complex multipliers are dedicated to the eight data paths respectively. For the 0th data path, the twiddle factor is always $W_N^0 = 1$ and thus no multiplier required.

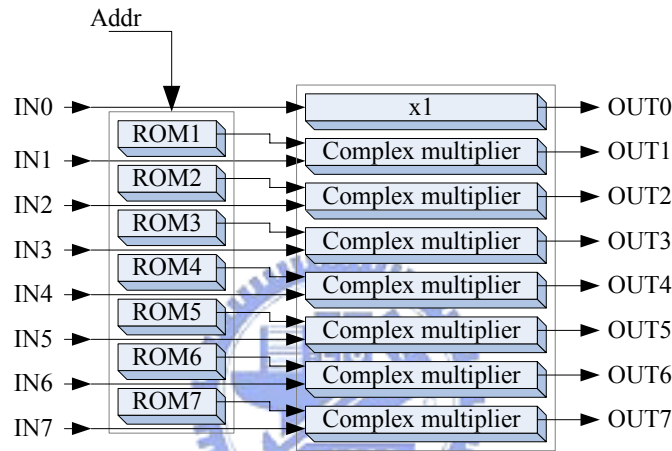


Figure 4.9 Block diagram of *MULT* stage

The read-only memory (ROM) is used to store the twiddle factors and supply them to the complex multipliers. The scheduling of twiddle factors for each data path for 4096-base case is listed in TABLE 4.5. The strategy used here is to map this table to the ROM storage. That is, with a specific ROM dedicated to each data path, each ROM stores 512 twiddle factors. The time slot information is used as the address for the ROMs. This strategy simplifies the control for ROM address generation for the multiplier stage. A counter is only required to generate the ROM address. For twiddle factors of 2048-base case, applying $W_{2048}^m = W_{4096}^{2m}$, the twiddle factors for 256 cycles thus corresponding to time slot $\{0, 2, 4 \dots 510\}$ in TABLE 4.5. Based on the same reason, we can use the same ROMs to supply twiddle factors for different modes. The address generation is simply modified by changing the counter interval.

Moreover, we divide each ROM into six banks. When small-base multiplication

mode is on, not all 512 time slots are required. It means that certain twiddle factors will not be needed and thus we can turn off ROM banks that store these unnecessary twiddle factors. Figure 4.10 shows the block diagram of the ROM banking and TABLE 4.6 shows the example of the twiddle factors stored in each bank for 4th ROM. As we can see, only bank {F} is activated during 128-base multiplications, and bank {E, F} are activated during 256-base multiplications, etc.

TABLE 4.5 Scheduling of twiddle factors, W_{4096}^p

Time slot \ Data path	0	1	2	3	4	5		510	511
0 th	0	0	0	0	0	0		0	0
1 st	0	4	8	12	16	20		2040	2044
2 nd	0	2	4	6	8	10		1020	1022
3 rd	0	6	12	18	24	30		3060	3066
4 th	0	1	2	3	4	5		510	511
5 th	0	5	10	15	20	25		2550	2555
6 th	0	3	6	9	12	15		1530	1533
7 th	0	7	14	21	28	35		3570	3577

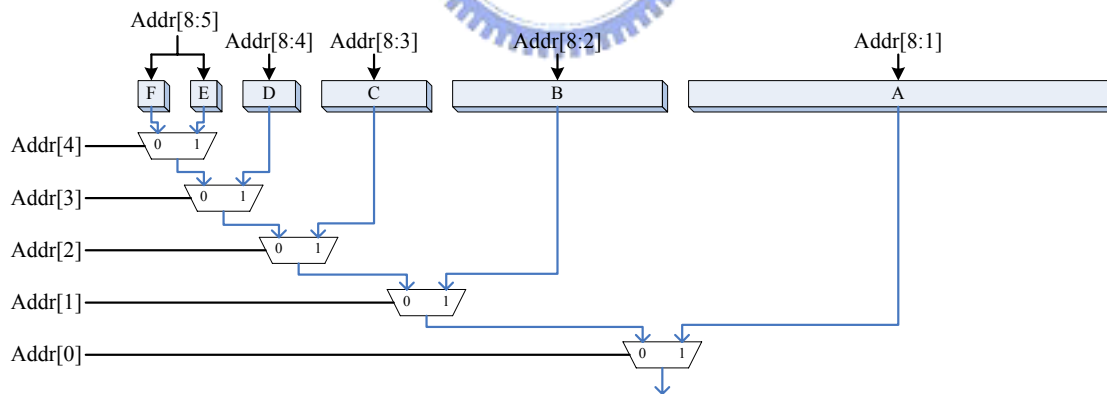


Figure 4.10 Block diagram of ROM banking

TABLE 4.6 Twiddle factors, W_{4096}^p , stored in 4th ROM

ROM	Content (p)	ROM	Content (p)
A	1, 3, 5...511	D	8, 24, 40...504
B	2, 6, 10,...510	E	16, 48, 80...496
C	4, 12, 20...508	F	0, 32, 64...480

4.3.3 Register Banks (RB)

The register banks are used to store intermediate data during calculation. As shown in section 3.4, the number of storage elements required between each stage at each mode is listed in TABLE 3.3. There should be three RB modules between four computation stages. For each of these RB modules, the number of storage elements varies according to different FFT length. Here, we want to design the register banks to be reconfigurable. That is, a RB module should be able to change its storage capacity. More importantly, the extra storage elements should be able to be fully turned off while the RB is in a low capacity mode. Therefore, a good circuitry partition is required in design the RB. Furthermore, the register banks also have the responsibility for reordering the data sequence before output them to next computation stage. Since there are nine different mixed-radix algorithms for our reconfigurable architecture, the control and dataflow of this reordering might be an annoying problem. Fortunately, we have a common rule for data ordering as described in previous section (section 3.2). The design of internal register banks thus become a much smooth job.

Also, in the design of our proposed register banks, a two-input register is used (Figure 4.11(a)). A direct implementation such registers will be using a D flip-flop with a 2-to-1 multiplexer for input selection, as shown in (b). We will discuss the implementation of the required registers in later chapter. The use of two-input register effectively adds another control signal, *CTRL*, for input selection. To control the data flow in the register banks, the clock signal, *CLK*, and *CTRL* must be taken care of.

In the overall architecture, there are three kinds of register banks, RB_4096, RB_512, and RB_64 (as shown in Figure 4.2). The structure of RB_4096 and RB_512 is of the same type while RB_64 is of another. We will depict these two types of RB respectively.

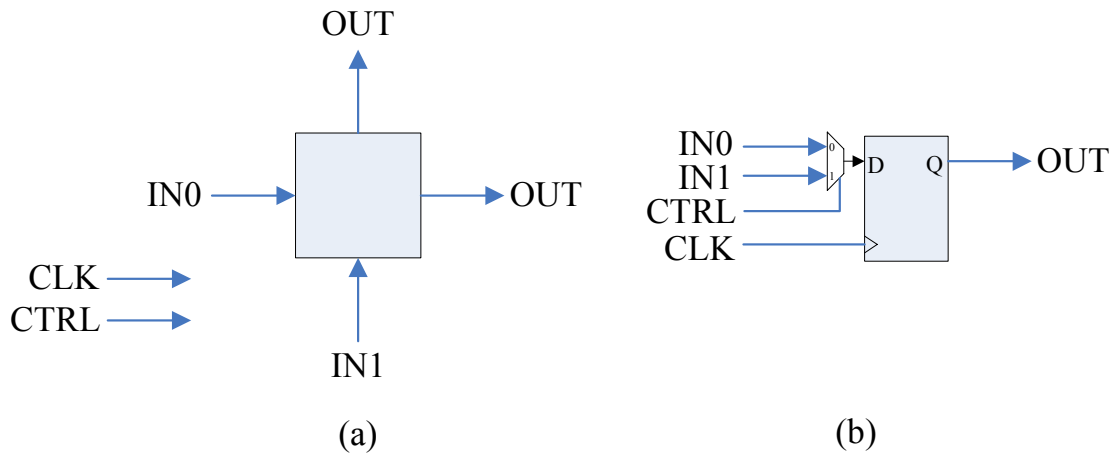


Figure 4.11 Block diagram of the two-input register

4.3.3.1 RB_64

Observing TABLE 3.1, the butterfly stage before RB_64 is radix-8 BF stage while the stage after is reconfigurable BF stage, which can be one radix-2, two radix-4, or one radix-8 butterfly. Referring to TABLE 3.3, the possible capacity of RB_64 may be of 16, 32, 64-word according to different FFT length. Figure 4.12 shows the block diagrams for the three different modes.

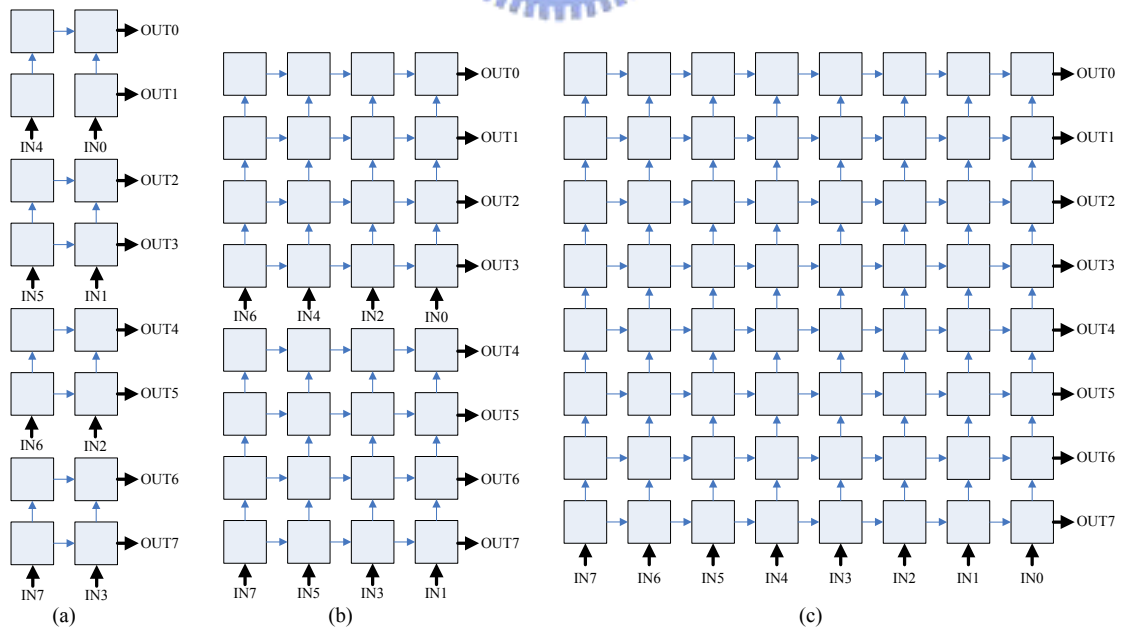


Figure 4.12 Block diagram of RB_64 for three different capacities, (a) 16-word, (b) 32-word, and (c) 64-word

Assuming that M -data capacity is required for the RB under current FFT mode, then it will take the RB $M/8$ to clock cycles to receive data previous stage and 8 parallel data for each clock cycle. During the input phase, the index of incoming data at cycle i is:

$$i+(M/8)*k \quad (4.1)$$

, where $k = 0\sim7$ represents the index of the 8-word datapath. During the output phase, the desired data ordering should be:

$$j*(M/8)+k \quad (4.2)$$

, where j is the output cycle count. Take the 16-word mode for example, as shown in Figure 4.13. Two cycles are required for the RB to receive data. During the input phase ($PHASE=1$), data from previous stage goes into the 8 dedicated input ports. For every cycle, the RB performs a shift-up operation. In other word, the two-input register choose the data from downward. During output phase ($PHASE=0$), the overall RB performs a shift-right operation and the desired data are obtained at the output ports every cycle. Therefore, we can use the $PHASE$ signal to control the data flow in the RB. The $PHASE$ signal is used for all the register in RB as the $CTRL$ signal, which selects the input data for the two-input register.

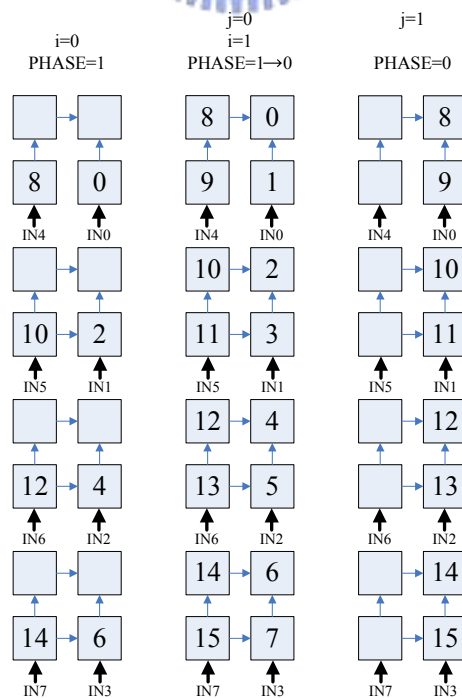


Figure 4.13 Data flow in RB for 16-word mode

The similar operation and data flow stand for the 32-word and 64-word RB. In order to deal with the various FFT modes, three different RBs are thus to be constructed, which is undesired overhead since only one of them is required at the same time. Observing Figure 4.12, it is not hard to find that the 64-word RB can also be used for the other two. Figure 4.14 shows such reconfigurable RB. Using parts of the 64-word RB, it can perform as the 32-word or 16-word RB as long as we redirect input data to the corresponding positions. The advantage of the reconfigurable RB is that, when a smaller capacity RB mode is required, the unnecessary registers can be fully turned off since they have nothing to do with the correct data-flow operation. Also, the control signal is simple since only the *PHASE* signal is needed. The overhead for reconfiguration is the multiplexers at the corresponding input ports.

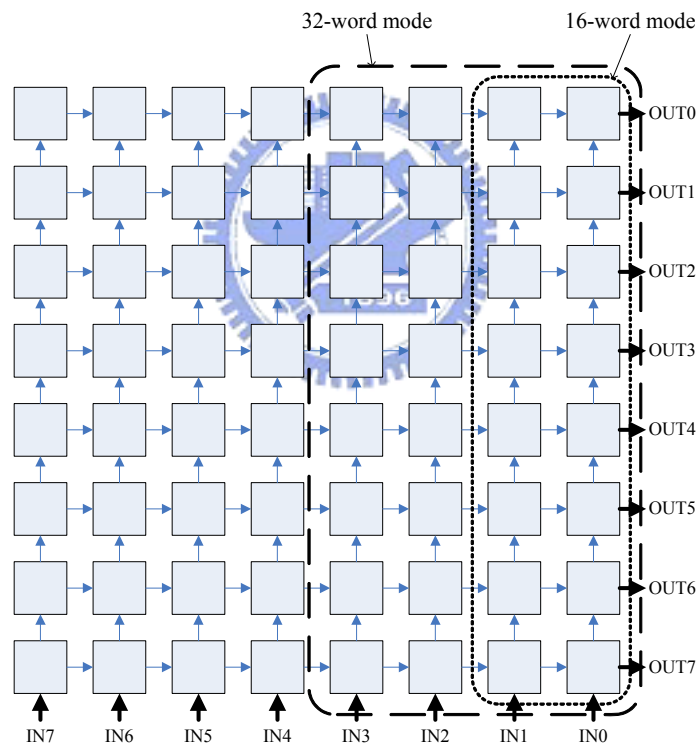


Figure 4.14 Block diagram of reconfigurable RB_64

4.3.3.2 RB_512 and RB_4096

The structure of RB_512 and RB_4096 are of the same type. Also observing TABLE 3.1, the butterfly stage before and after these two RB stages are both radix-8 BF stages. The RB capacity ranges from 128-word to 4096-word.

Assuming that M -data capacity is required for the current RB, then it will take the RB $M/8$ to clock cycles to receive data previous stage and 8 parallel data for each clock cycle. During the input phase, the index of incoming data at cycle i is:

$$i+(M/8)*k \quad (4.3)$$

, where $k = 0\sim 7$ represents the index of the 8-word datapath. During the output phase, the desired data ordering should be:

$$\left\lfloor j/\left(\frac{M}{8^2}\right) \right\rfloor * \frac{M}{8^2} + j\%(\frac{M}{8^2}) + \frac{M}{8^2} * k \quad (4.4)$$

, where j is the output cycle count. Take the 128-word RB mode for example, as shown in Figure 4.15. The overall RB can be seen as a combination of eight blocks and each input data goes into one block respectively. During the input phase, the bottom row of registers performs shift-right operation. For every 2 cycles, the upper rows of registers perform shift-up operation in order to shift-up the bottom row data to let next data coming in. After 16 cycles, the first output data are ready at the output ports. At output phase, the overall RB performs the shift-right operation to deliver output data every cycle.

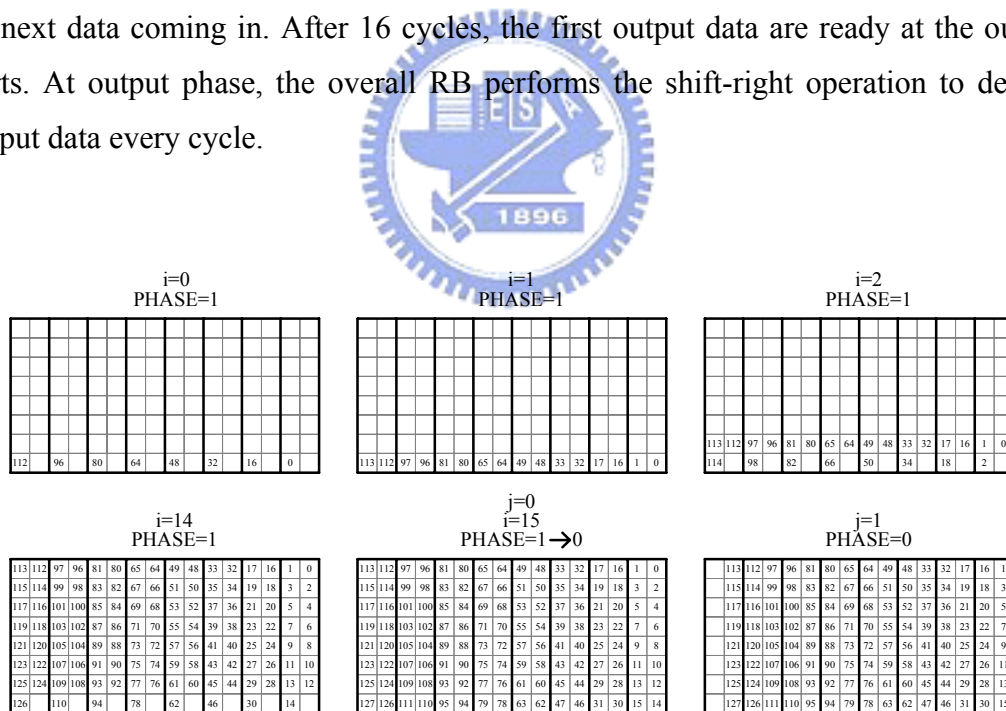


Figure 4.15 Data flow in RB for 128-word mode

The control signals of the above operations are also of simple regularity. Both the $PHASE$ and CLK need to be considered. First, we divide the RB into two control zones, as shown in Figure 4.16. For the two-input register in Zone 1, they share the

same clock and use the *PHASE* signal as the input-select signal. For registers in Zone 2, except for those at input ports, their input-select signals are set to 0 in order to perform the shift-right operation at all time. Each of the 8 blocks has a dedicated input port. For the register at input ports, the *PHASE* signal is used for the input-select signal. The resulting timing relation of the control signals is shown in Figure 4.17. Notice that the clock signals for Zone 1 toggle only every two cycle during the input phase.

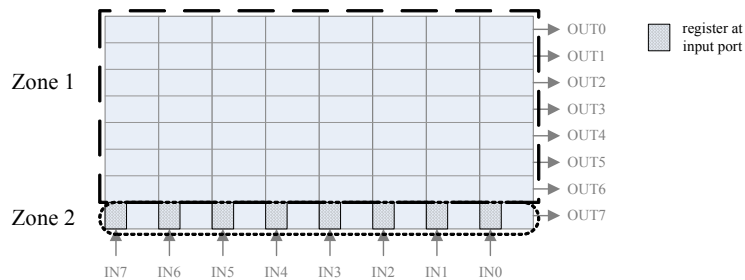


Figure 4.16 Control zones for the RB

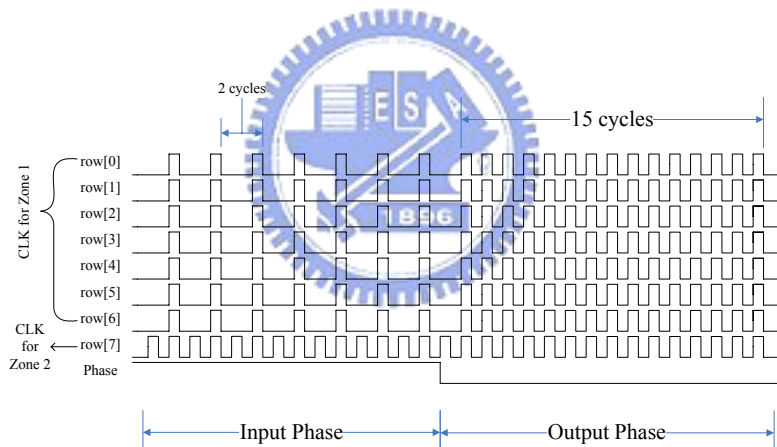


Figure 4.17 Control signals for the 128-word RB

Based on the above scheme, we can derive a rule for the structure of this type of RB. The RB is combined by 8 basic blocks, which are of the same structure and connected one after another, and there are 8 rows of two-input registers in a basic block. For the RB of capacity of M -data, the width of a basic block is $(M/8^2)$ -word. The control signals are the *PHASE* signal and two clock signals for the two control zones. The clock signals for Zone 1 toggle every $(M/8^2)$ cycle during input phase.

According to TABLE 3.3, the RB_512 module may be of the capacity 128, 256, or 512-word and RB_4096 module may be of the capacity 1024, 2048, or 4096-word

for different FFT length. The same as the RB_64 in previous section, we can also use a reconfigurable RB. Figure 4.18 shows the block of RB_512, which can be reconfigured as 128, 256 or 512-word. The basic block is of width 8-word. The dedicated input for a basic block is connected to three different registers. For RB_512 to act as 128-word RB, the input data goes into the right most input register, which effectively set the width of the basic block to 2-word. The column where the input register is at now takes data from the output of previous basic block instead of previous column. For the rest 6 columns at the left of the basic block, they can be fully turned off since they have nothing to do with the correct data-flow operation. With this scheme, we can change the mode of RB just by setting the corresponding input registers and modifying control signals. The overhead for reconfiguration is the multiplexers at the columns of the input registers in every block.

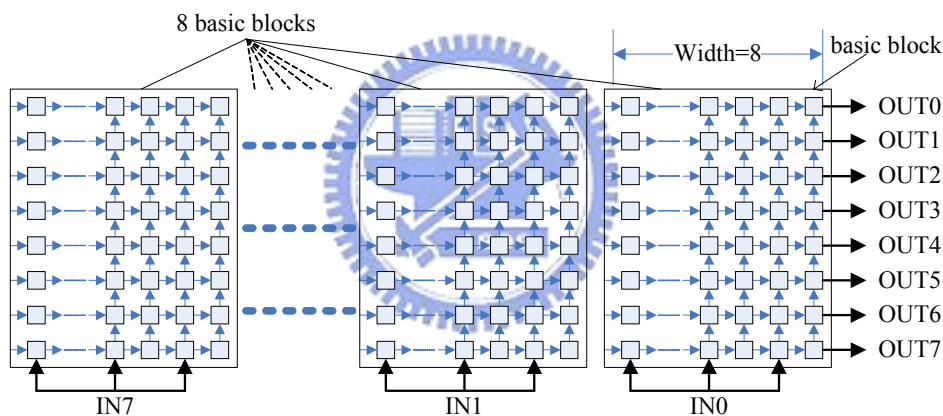


Figure 4.18 Block diagram of reconfigurable RB_512

4.3.3.3 Duplicate Module Insertion

In the above design of register banks, the RB modules are not able to accept input data during output phase, even if there is empty register available. This is because the flow in the RB is two-way direction instead of the traditional one-way. The situation is awkward to the pipeline-based architecture. In the correct pipeline operation, the RB should be able to accept one data input after one data output. Since the designed RB can not do this, the data flow will have to stall to wait for the RB available.

Our method to deal with this problem is to insert duplicate RBs. Two identical

RBs are presented for the stage. When the first RB is at output phase, the input data from previous will go into the second one. Then the second RB will go into output phase and the first RB is back to input phase in turn. Since the input and output phase both take the same cycles, two RBs are enough to make the data flow with stalls.

However, we do not use duplicate RB module for the first RB stage in order to save the number of storage elements required. Since the capacity of the first stage RB is N -word, where N is the FFT length, large number of registers can be saved without the duplicate module in the first stage RB. This approach effectively reduces the throughput rate of our architecture. The original throughput rate is 8 times of the input clock rate since the datapath is of 8-data width. The resulting throughput rate turns out to be 4 times of the input clock rate. This is because the first stage takes $N/8$ clock cycles to take input data and $N/8$ clock cycles to deliver output data afterward. Therefore, the throughput rate at the FFT input is cut half due to this effect.

4.3.4 BFP



As mentioned in Chapter 3, the block-floating-point technique is used in the proposed RMR FFT in order to maintain the SNR. The approach used here is the “input scaling” approach. The data are evaluated before writing to memory (RB) and not shifted until they are read into next computation stage. That is, in our architecture, the incoming data of RB are evaluated during the input phase of RB. While the RB is at output phase, the RB output data will be shifted according to the scaling factor obtained during input phase.

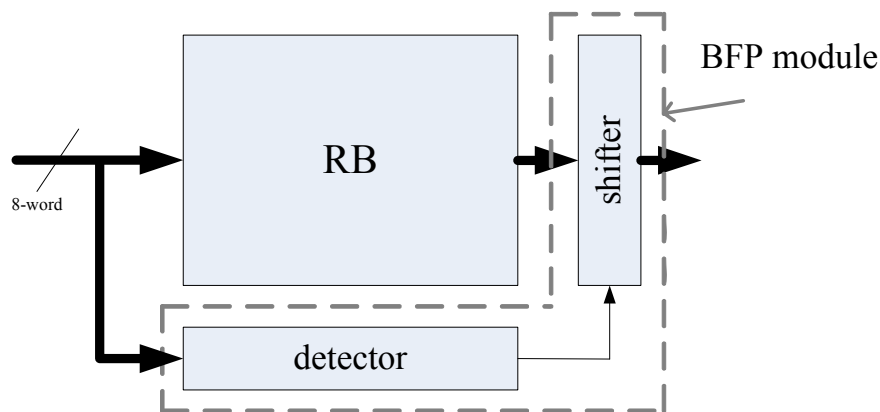


Figure 4.19 Block diagram of BFP

The block diagram of the BFP block is shown in Figure 4.19. The BFP block consists of a detector and a shifter. The detector picks out the largest data in a block, and records its number of MSB zeros or ones as the scaling factor. The shifter shifts data according to the scaling factor of the block. Data will not be shifted until they go into the next computation stage.

As the overall architecture of Figure 4.2, there are three BFP modules in the FFT. For RB stage with duplicate module, only one BFP is needed since the input phase and output phase of these two RBs will not overlap. A scaling table is constructed to store the scaling factors from each BFP modules. The final output scaling factor will be determined according to this scaling table.

4.3.5 INPUT/OUTPUT Buffer

Since the IFFT can be performed by first swap the real and imaginary parts of input data, and then swap the real and imaginary parts of output data [4.4], the FFT processor is capable of performing IFFT operations by adding swap units at input and output respectively. Although we have assumed that external input buffer has arranged the input data in 8-parallel format as the proposed FFT required, we will give a simple demonstration of how such a reconfigurable input buffer can be designed.

Figure 4.20 shows the block diagram of a reconfigurable input buffer. The data flow is similar to that of RB_512 or RB_4096. Taking $N=16$ for example, only two right most columns are activated. For every two cycles, the upper rows perform the shift-up operation. After the buffer is full, shift-right operation is performed to deliver the output data. The data flow example is shown in Figure 4.21. The way of data moving in the buffer is similar to that in a RB (section 4.3.3.2). The number of memory required for the input buffer for N -point FFT is $(7N/8) + 1$.

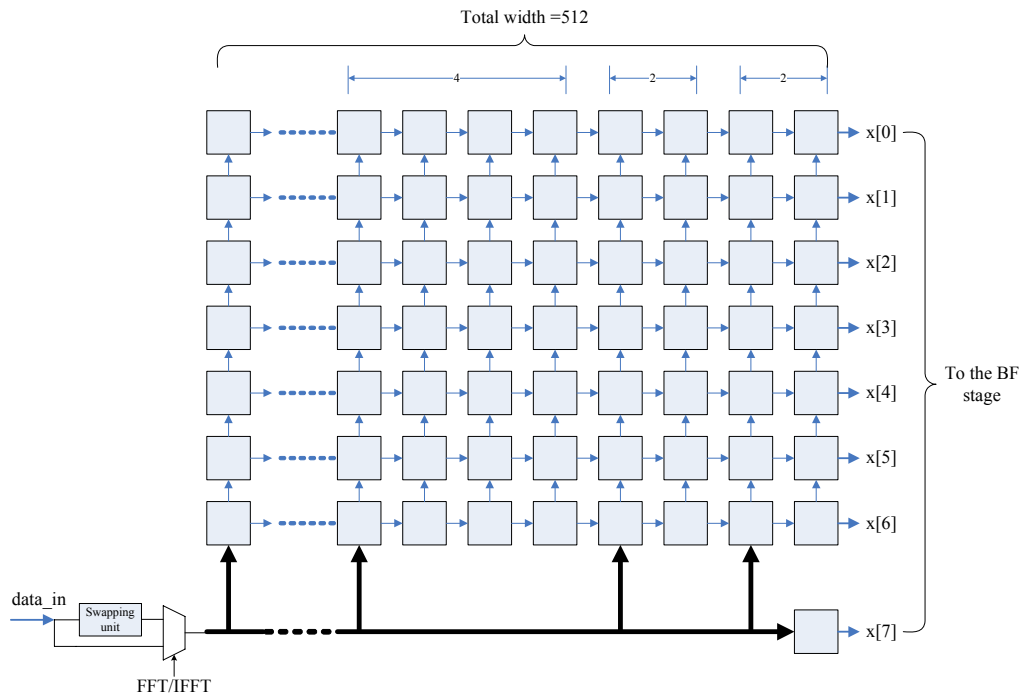


Figure 4.20 Block diagram of reconfigurable input buffer

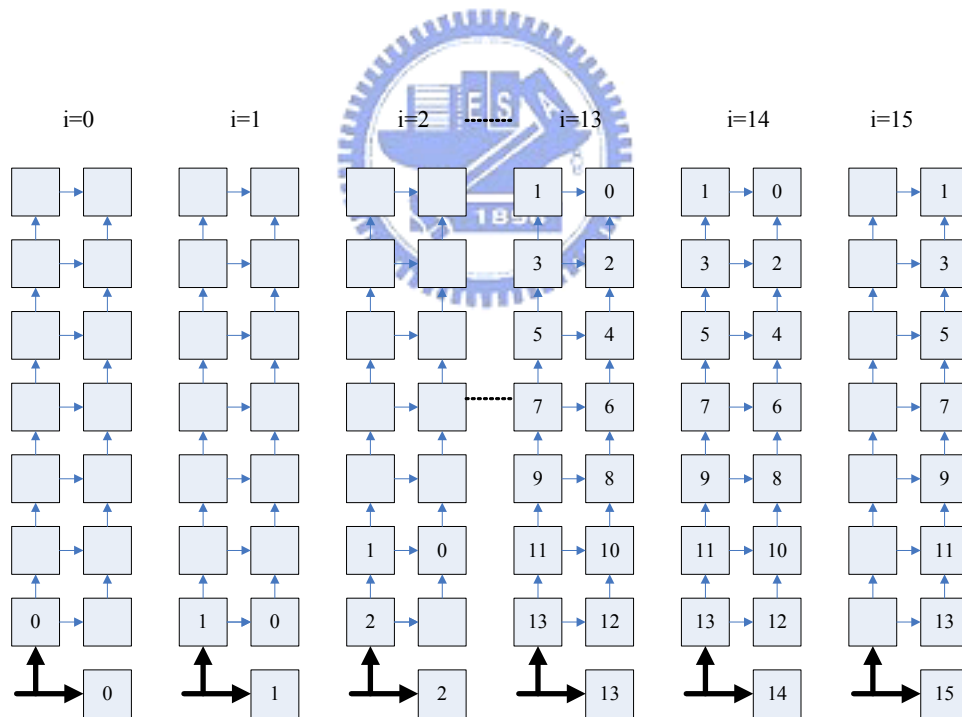


Figure 4.21 Data flow in the input buffer for $N = 16$

As derived in chapter 2, the output order of the FFT is in bit-reversed order for whatever FFT length. Therefore, the output can be indexed without too much trouble. Also, the final data will be output with a scaling factor, OUT_SBIT , as described previously. The correct output precision will have to be adjusted with the output

scaling factor. Alternatively, the block-floating-point representation of output data can directly go into next functional module, which also supports the BFP operation.

4.4 Data Flow

The proposed RMR FFT is capable of performing FFT of length from 16 to 4096 points. According to different FFT length, the flow of data path varies in three ways. For {16, 32, 64}-point FFT, the data flow is shown as Figure 4.22. Since only two butterfly stages are required, the preceding stages are directly bypassed. Only one RB stage, RB_{64} , is needed. As described in section 4.3.3, no duplicate RB is used for the first RB stage. Therefore, only RB_{64a} is activated in the only RB stage.

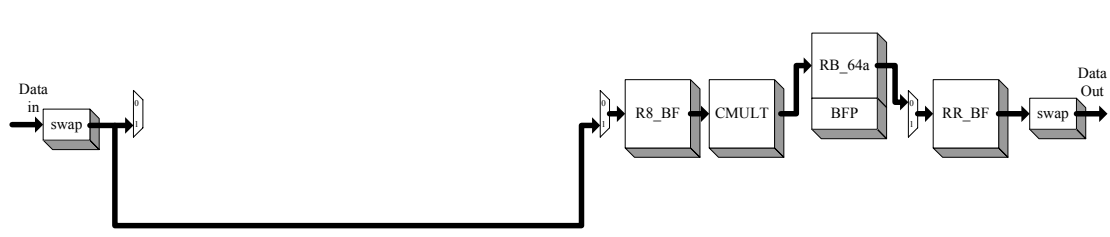


Figure 4.22 Flow of data path for 16, 32, 64-point FFT

Figure 4.23 shows the flow of data path for {128, 256, 512}-point FFT. Three butterfly stages are required for these three FFT lengths and therefore all three BF stages are activated. As the same, the first RB stage, RB_{512} , does not use a duplicate module and thus only RB_{512a} is activated. For the RB_{64} stage, the $PHASE$ signal for the two RB modules is shown in Figure 4.24. The RB_{64a} will first accept input data and while it is in output phase, incoming data from previous stage go into RB_{64b} instead. The width of an output or input phase is $M/8$ cycles, where M is the current RB capacity. Because the capacity of previous RB stage, RB_{512} , must be 8 times of the RB_{64} (referring to TABLE 3.3), 8 input-output phases are required and each RB_{64} takes four. The control signals show that the two RB work in turn without overlap.

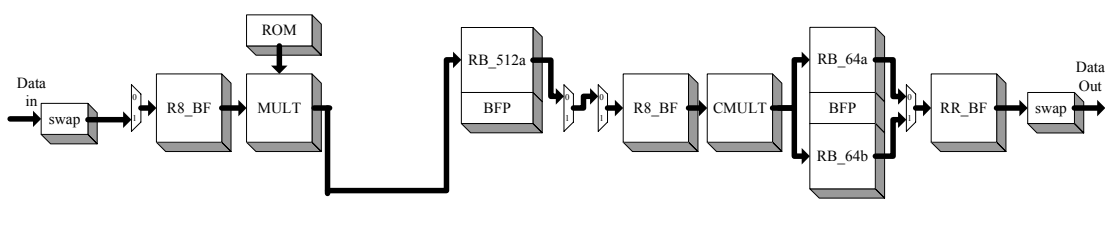


Figure 4.23 Flow of data path for 128, 256, 512-point FFT

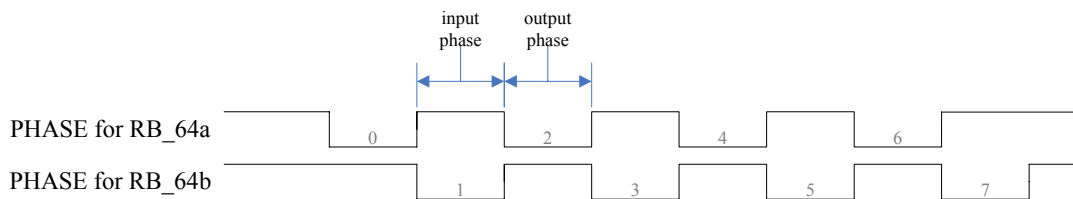


Figure 4.24 Control signal, *PHASE*, for duplicate RB modules

For {1024, 2048, 4096}-point FFT, the data flow is shown as Figure 4.25. The data flow will go through all the modules. The first computation stage, {*RB_BF*, *MULT*, *ROM*}, is used to calculate the first two butterfly stages in the SFG.

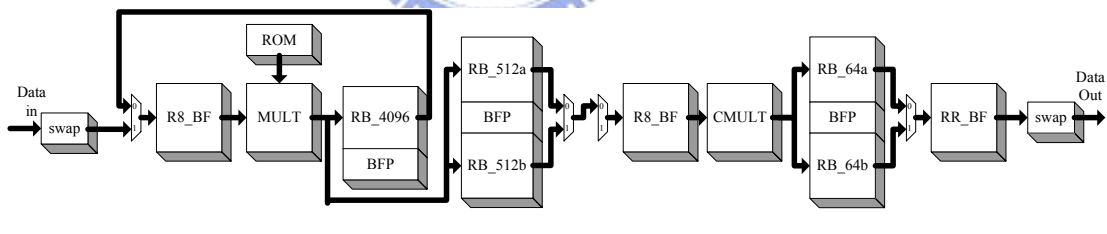


Figure 4.25 Flow of data path for 1024, 2048, 4096-point FFT

TABLE 4.7 shows the memory requirement of our architecture compared to other generic architectures [4.5]. As our architecture can achieve the throughput rate 4 times of the input clock rate, which is close to the radix-4 MDC architecture, the memory required is less than that of the radix-4 MDC architecture.

**TABLE 4.7 Comparison of memory requirement
(including the input buffer)**

Algorithm m N	Proposed RMR FFT		Radix-2		Radix-4		Radix-8
			SDF	MDC	SDF	MDC	MDC
16	31	$(1+7/8)N+1$	N-1	1.5N-1	2N-2	2.5N-1	/
32	61	$(1+7/8)N+1$	N-1	1.5N-1	/	/	/
64	121	$(1+7/8)N+1$	N-1	1.5N-1	2N-2	2.5N-1	4.5N-8
128	273	$(1+2/8+7/8)N+1$	N-1	1.5N-1	/	/	/
256	545	$(1+2/8+7/8)N+1$	N-1	1.5N-1	2N-2	2.5N-1	/
512	1089	$(1+2/8+7/8)N+1$	N-1	1.5N-1	/	/	4.5N-8
1024	2209	$(1+2/8+2/64+7/8)N+1$	N-1	1.5N-1	2N-2	2.5N-1	/
2048	4417	$(1+2/8+2/64+7/8)N+1$	N-1	1.5N-1	/	/	/
4096	8833	$(1+2/8+2/64+7/8)N+1$	N-1	1.5N-1	2N-2	2.5N-1	4.5N-8

In the implementation of the proposed architecture, pipeline registers will be inserted between the BF module and multiplier module. The execution cycles required for an N -point FFT are:

$$\{16, 32, 64\}\text{-point} : \frac{N}{8} + \frac{N}{8} + 2$$

$$\{128, 256, 512\}\text{-point} : \frac{N}{8} + \frac{N}{8} + \frac{N}{8^2} + 3$$

$$\{1024, 2048, 4096\}\text{-point} : \frac{N}{8} + \frac{N}{8} + \frac{N}{8^2} + \frac{N}{8^3} + 4$$

The required cycles are summarized in TABLE 4.8.

TABLE 4.8 Execution cycles required for different FFT length

FFT size	16	32	64	128	256	512	1024	2048	4096
Execution cycles	6	10	18	37	71	139	278	552	1100

4.5 Conclusions

In this chapter, the overall architecture of the proposed RMR FFT has been drawn out. In algorithm level, the RMR FFT is divided as four computation stages. In architecture design, we manage to combine the first two computation stages into one. The intermediate data are stored in the register bank modules. The two-input registers are used for constructing the RB modules, which are organized in chessboard-like structure. Reconfiguration of the RB modules can be easily achieved by modifying few control signals. Two different strategies are used for the multiplier stages. The constant multiplier approach eliminates the need for another ROM. For the multiplier-ROM stage, the coefficient address generation is accomplished through a simple counter. In order to adopt the block-floating-point method, BFP modules are added with the RB modules to scale block data before every computation stage. The approach of designing the input buffer is also described.

Unnecessary circuit can be fully turned off without affecting the correct operation due to good circuitry partition. The number of memory elements required is relative low comparing to generic pipeline-based architecture. Considering the hardware overhead, the overall throughput rate reaches 4 times of the input clock rate.

Chapter 5

Implementation of RMR FFT/IFFT Processor

5.1 Introduction

In this chapter, we will show the implementation strategy and simulation results of the proposed RMR FFT. The overall design is to be implemented through cell-based synthesis design flow, except for the register bank modules. For the RB modules, we will use the full-custom design flow to further exploit the regularity of the proposed structure. The implementation of the register banks is described in section 5.2. As we partition the circuitry carefully to achieve an energy-aware design, section 5.3 shows a simple way to realize the control for external power management unit. As will be seen in section 5.4, the proposed RMR FFT achieves energy-aware design. Compared to other reconfigurable architecture, our FFT outperforms in execution speed with relative low power and energy dissipation.

5.2 Implementation issue on Register Banks

As introduced in section 4.3.3, the structures of our register banks are of high regularity. The two-input registers are put together in a chessboard-like manner. Using Design Compiler for synthesis, the synthesized result for the two-input register is shown in Figure 5.1 [5.1]. The *SI* represents the scan input and *SE* is the scan enable. Such flip-flop is usually used as the scan D flip-flop. Basically, it is D flip-flop with a 2-to-1 multiplexer at the input. However, this implementation can cause unnecessary power consumption. As shown in section 4.3.3, there are situations that the inputs of the registers are changed while the clocks to flip-flops are gated. This means that the multiplexer can consume unnecessary transition power. Also, the regularity of the RB structure can not be fully reflected through the CBD flow. Therefore, we take the

FCD flow to further implement the RB modules.

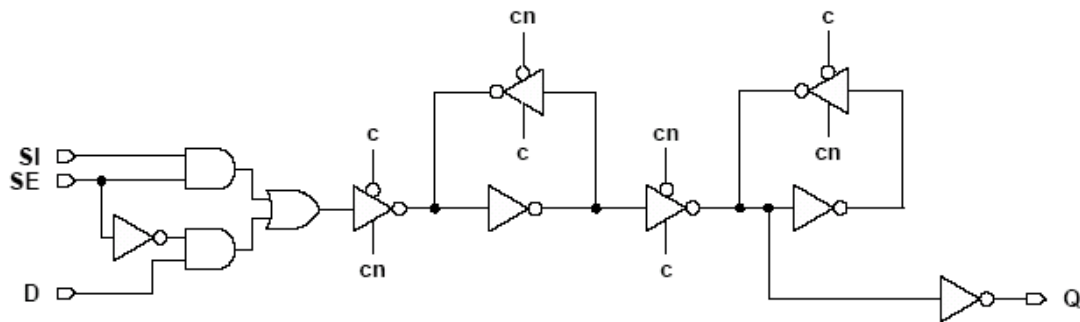


Figure 5.1 Circuit of synthesized scan D flip-flop

Referring to the block diagrams of Figure 4.18, we construct the RB arrays in the way shown in Figure 5.2. Transmission gates are used for the input selection. To further optimize the RB modules, now the problem becomes the choice of the D flip-flop.

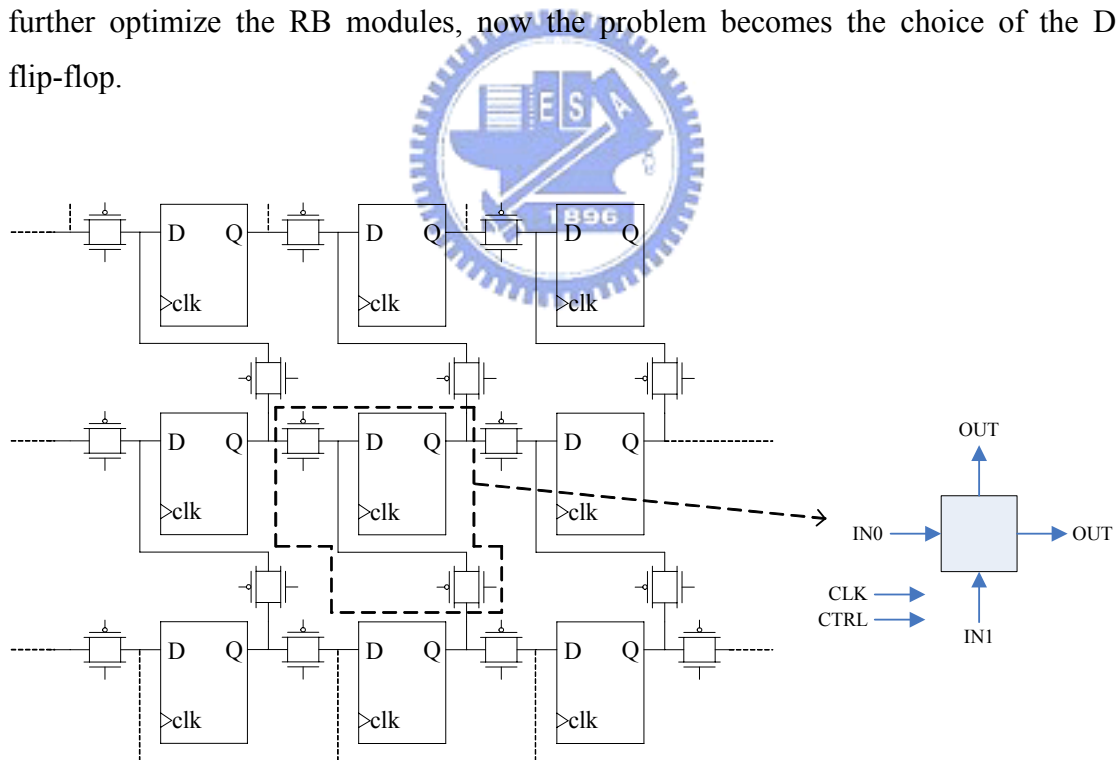


Figure 5.2 Block diagram of the two-input register array

We have searched some popular architecture of D flip-flops [5.2-5.7], as shown in Figure 5.3.

- (a) The positive-edge triggered, static D-type flip-flop according to the data sheet of TSMC 0.13 μ m technology [5.1].
- (b) Clocked CMOS (C^2 MOS) register [5.8], which is insensitive to clock overlaps. Only eight transistors are used. However, with sufficient slow rise and fall times, there will be a time slot both NMOS and PMOS are conducting, which results in extra leakage power.
- (c) Hybrid-latch Flip-Flop (HLFF) [5.9] is one of the fastest structure presented. It also has a small power-delay product. The major advantage of this structure is its robustness to clock skew. However, unnecessary internal transitions increase the total power consumption.
- (d) True single-phase clocked register (TSPCR) [5.10] uses a single phase clock. Similar to C^2 MOS register, slow clocks cause both the NMOS and PMOS clocked transistors to be on simultaneously.
- (e) The modified C^2 MOS Register has low power feedback assuring fully static operation. Compared to the classical C^2 MOS structure, the modified C^2 MOS is robust to clock slope variation.
- (f) PowerPC 603 master-slave latch is one of the fastest classical structures [5.11]. Its main advantages are a short direct path and low-power feedback. However, the large clock load influences the total on-chip power consumption. Such effect can be reduced by the local clock buffering.

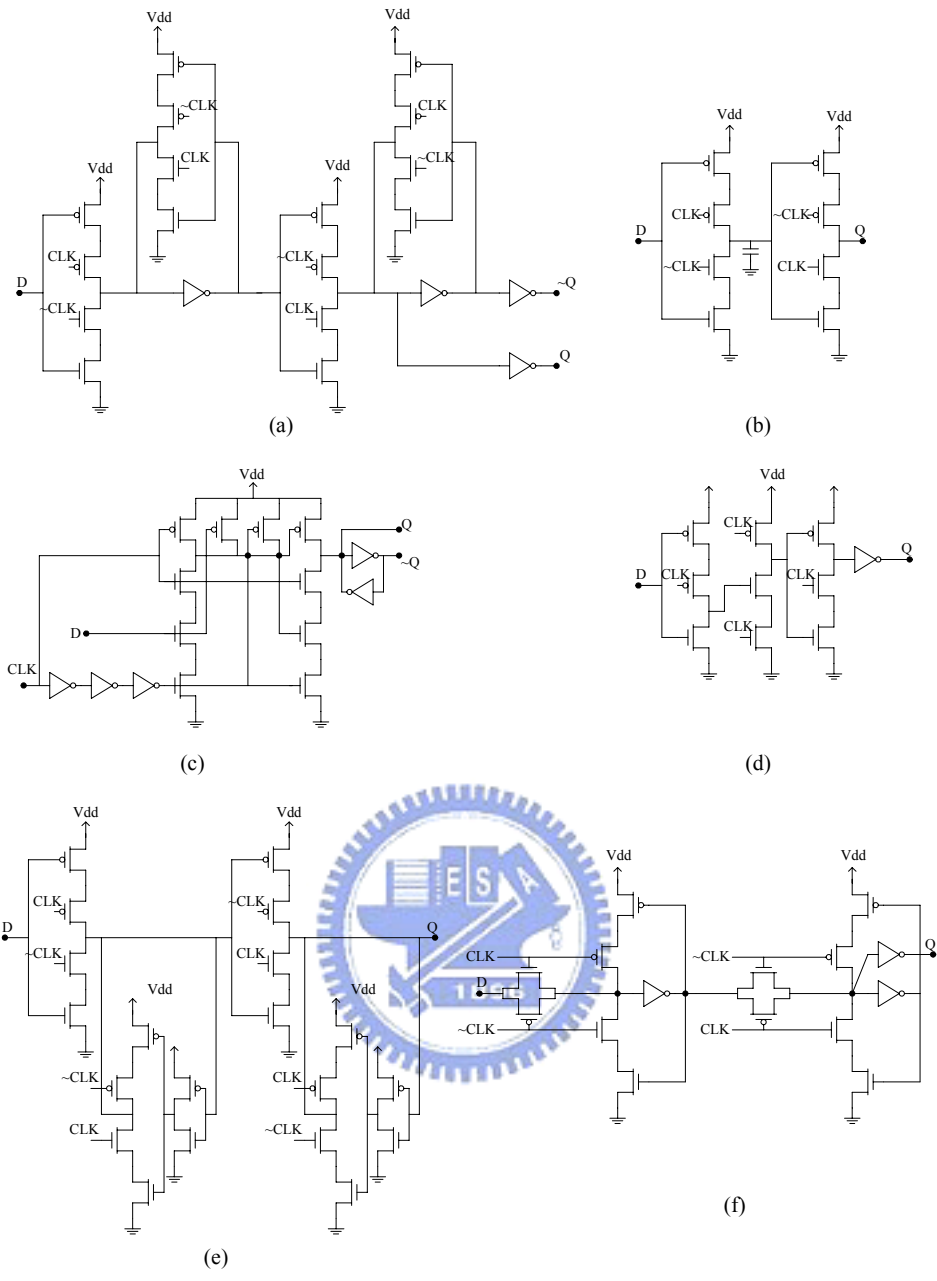


Figure 5.3 Various structure of D flip-flops

We have constructed a basic block of RB_4096 ($64 \times 8 = 512$ bits) using the above flip-flops respectively. Since the D flip-flops perform only shifter operation in our pipeline architecture, speed is not the most concerned criterion. Instead, we are more interested in the performance of power consumption. Figure 5.4 shows the average current with low-clock-transition input patterns and Figure 5.5 is the case with high-clock-transition. As describe in section 4.3.3, most registers in the RB are in low-clock-transition during the input phase and high-clock-transition during the

output phase. Architecture (b) and (f) both consume relative low transition current, as shown in Figure 5.5. However, the data of (b) in Figure 5.4 is high, which means that leakage current is large for this architecture. According to the simulation result, we choose the structure of (f) as the D flip-flop in the RB modules.

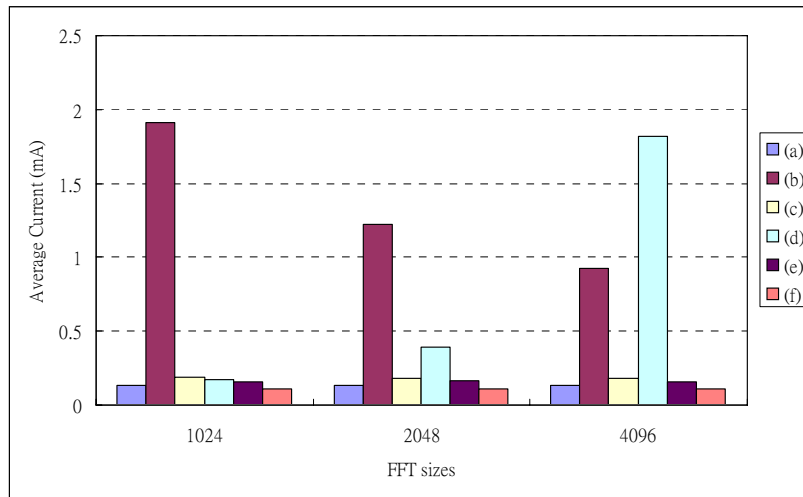


Figure 5.4 Average current under low-clock-transition cases

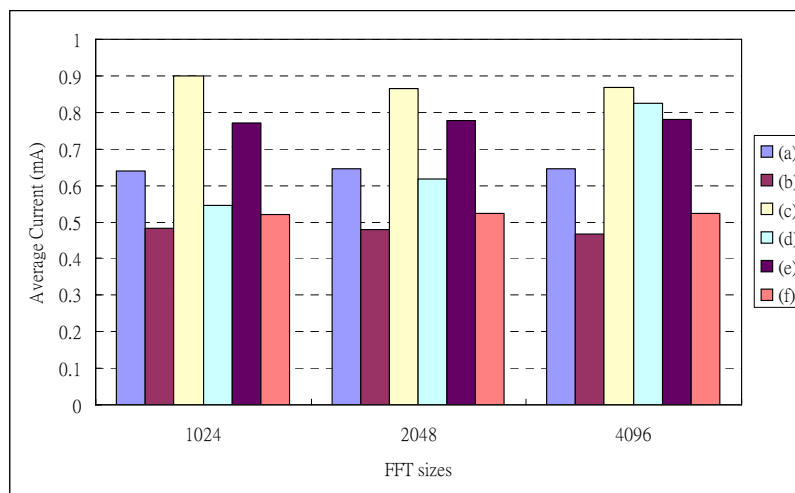


Figure 5.5 Average current under high-clock-transition cases

5.3 Power Control

As described previously, different amount of modules are required to be activated for different FFT length. As shown in section 4.4, three different flow of data path are dedicated for the 9 FFT lengths. Also, the activation of certain sub-modules is different according to different FFT length, such as ROM and register banks. For these unnecessary modules, they can be fully turned off since they have nothing to do with the correct operation. Although the power management unit is not implemented in our RMR FFT, we have derived a control table for the power control, as shown in TABLE 5.1. The number *1* represents a turn-on situation and *0* represents opposite. The *CONTROL* block is turned on under all modes since it generates control signals for all modules. The *ROM* and *MULT* modules are turned on when the FFT size is larger than 128 points. The ROM is divided into six banks as described in 4.3.2.2. Each register bank is divided as three parts. They are of the size 1/4, 1/4, and 1/2 of the RB respectively. As described in section 4.3.3, the RBs are made reconfigurable such that only the required numbers of registers are turned on for different FFT sizes.

As shown in TABLE 5.1, the control of such power management unit is simple enough to realize through a truth table. By turning off the unnecessary blocks during between FFT operations of different sizes, the proposed RMR FFT can achieve energy-aware and power scalability.

TABLE 5.1 Truth table of the activated modules

Mode	16	32	64	128	256	512	1024	2048	4096
CONTROL	1	1	1	1	1	1	1	1	1
R8_BF	0	0	0	1	1	1	1	1	1
MULT	0	0	0	1	1	1	1	1	1
ROM									
rom A	0	0	0	1	1	1	1	1	1
rom B	0	0	0	0	1	1	1	1	1
rom C	0	0	0	0	0	1	1	1	1
rom D	0	0	0	0	0	0	1	1	1
rom E	0	0	0	0	0	0	0	1	1
rom F	0	0	0	0	0	0	0	0	1
RB_4096									
RB_4096_1	0	0	0	0	0	0	1	1	1
RB_4096_2	0	0	0	0	0	0	0	1	1
RB_4096_3	0	0	0	0	0	0	0	0	1
BFP	0	0	0	0	0	0	1	1	1
RB_512a									
RB_512a_1	0	0	0	1	1	1	1	1	1
RB_512a_2	0	0	0	0	1	1	0	1	1
RB_512a_3	0	0	0	0	0	1	0	0	1
RB_512b									
RB_512b_1	0	0	0	0	0	0	1	1	1
RB_512b_2	0	0	0	0	0	0	0	1	1
RB_512b_3	0	0	0	0	0	0	0	0	1
BFP	0	0	0	1	1	1	1	1	1
R8_BF	1	1	1	1	1	1	1	1	1
CMULT	1	1	1	1	1	1	1	1	1
RB_64a									
RB_64a_1	1	1	1	1	1	1	1	1	1
RB_64a_2	0	1	1	0	1	1	0	1	1
RB_64a_3	0	0	1	0	0	1	0	0	1
RB_64b									
RB_64b_1	0	0	0	1	1	1	1	1	1
RB_64b_2	0	0	0	0	1	1	0	1	1
RB_64b_3	0	0	0	0	0	1	0	0	1
BFP	1	1	1	1	1	1	1	1	1
RR_BF	1	1	1	1	1	1	1	1	1

5.4 Simulation Result

With the implementation strategy described above, we have implemented the proposed RMR FFT using **TSMC 0.13 μ m** technology. For the three RB modules, RB_4096, RB_512a, and RB_512b, full-custom design approach is used and the circuitry is simulated using the HSPICE netlist. The rest of the reconfigurable FFT processor is designed with Verilog HDL and synthesized to TSMC 0.13 μ m CMOS standard cell technology library with Synopsys Design Compiler. This is followed by gate level simulations. Synopsys PrimePower is used for power analysis. The wordlength of the FFT is 16 bits. The simulation is run under supply voltage 1.2V. The maximum working frequency is 110MHz. The throughput of the RMR FFT is four times of the input clock rate, which can reach 440Msample/s. The profile of power consumptions are analyzed with the power control table mentioned in previous section.

5.4.1 Performance of the RMR FFT



With random patterns as the input, the SNR of the RMR FFT maintains above 110 dB for various FFT sizes. As described previously, the SNR is maintained due to the utilization of the BFP in RMR FFT. The BFP approach also enables us to keep the internal wordlength the same as the IO wordlength. Without the BFP approach, the SNR degrades as the FFT size grows. Figure 5.6 shows the SNR comparison between the RMR FFT with and without BFP approach applied.

The power consumption of the proposed RMR FFT is shown in Figure 5.7. As can be seen, there are three levels of the power consumption. We can consider FFT of size {16, 32, 64} as one group and {128, 256, 512} and {1024, 2048, 4096} as the other two groups. This is due to that different numbers of computation stage are required for different groups of FFTs. The figure also shows that our FFT has good power scalability.

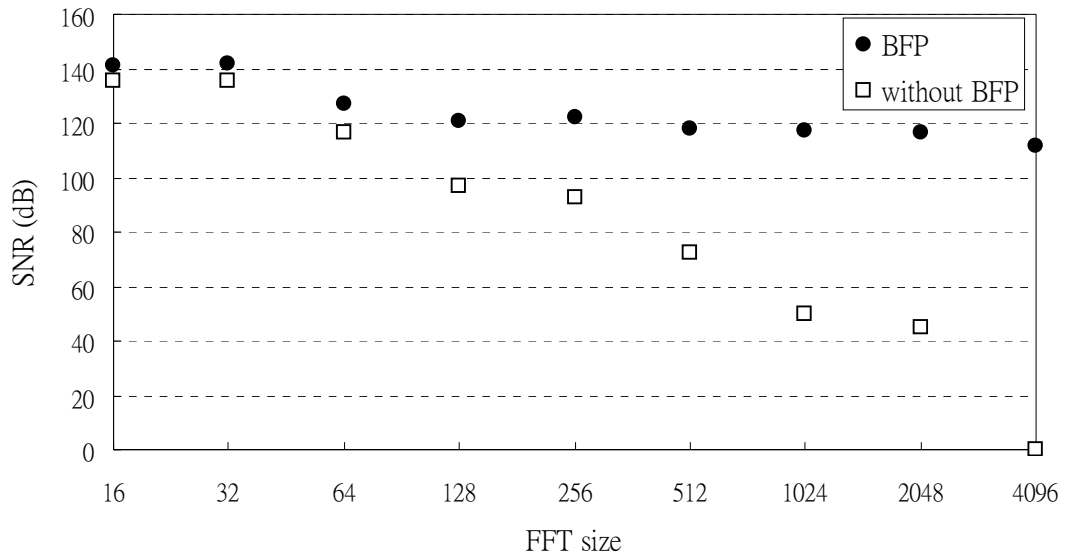


Figure 5.6 SNR comparison

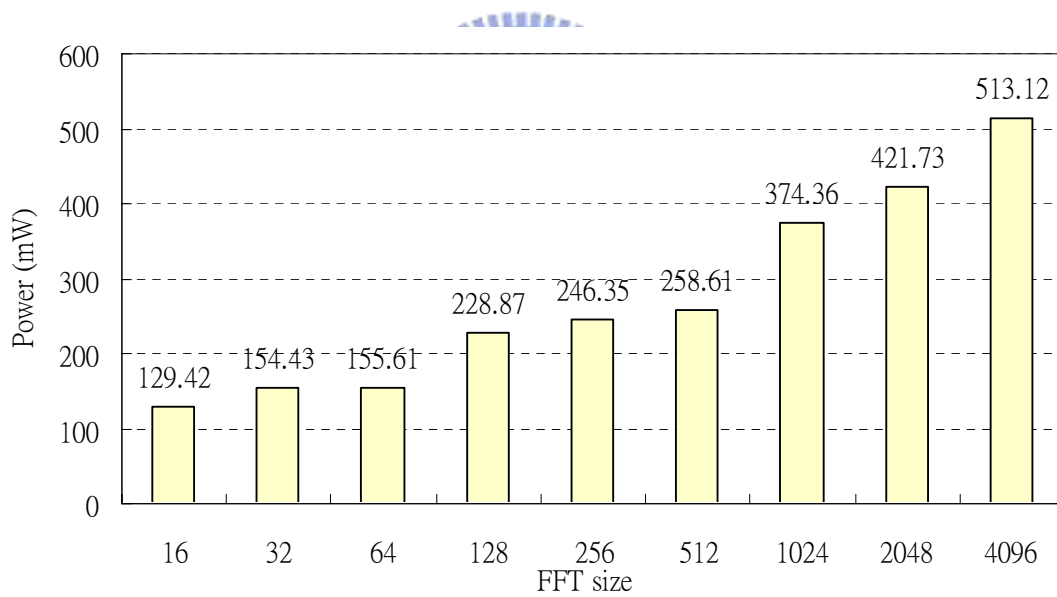


Figure 5.7 Power consumption for various FFT sizes (110MHz, 1.2V)

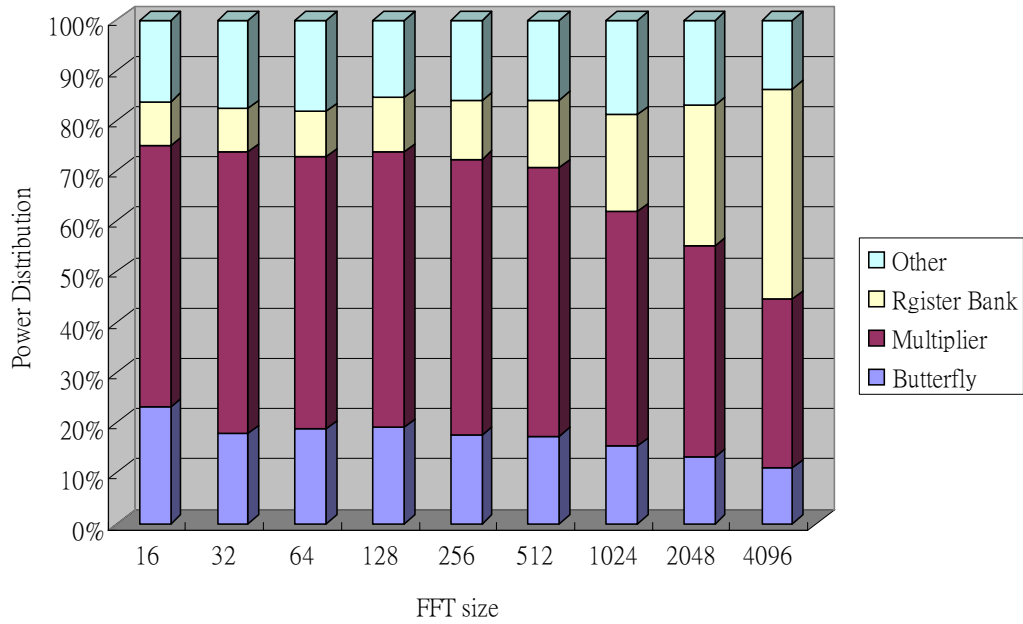


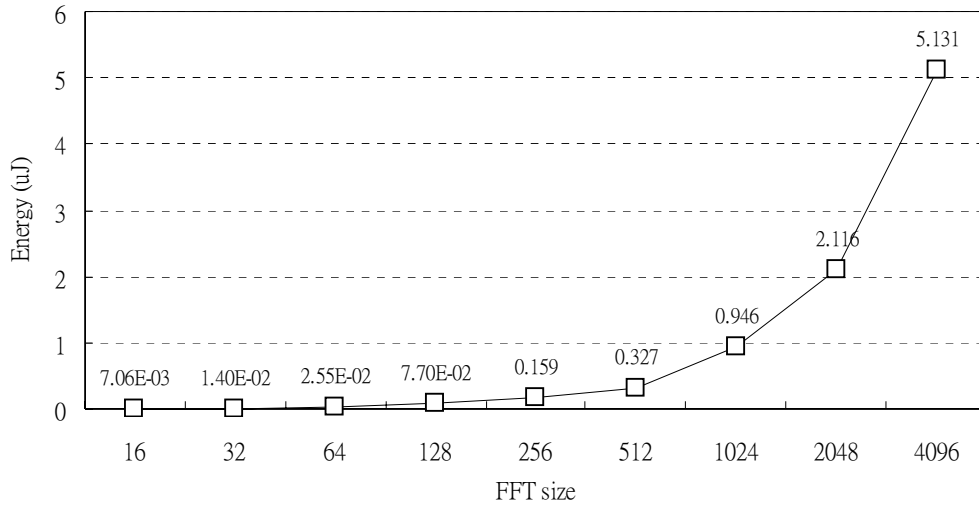
Figure 5.8 Power distribution characteristics

Figure 5.8 shows the distribution of power consumption over the RMR FFT. The power dissipation concentrates on the computation modules when the FFT size is small. In long-size FFT, the power of the internal register banks start to dominate.

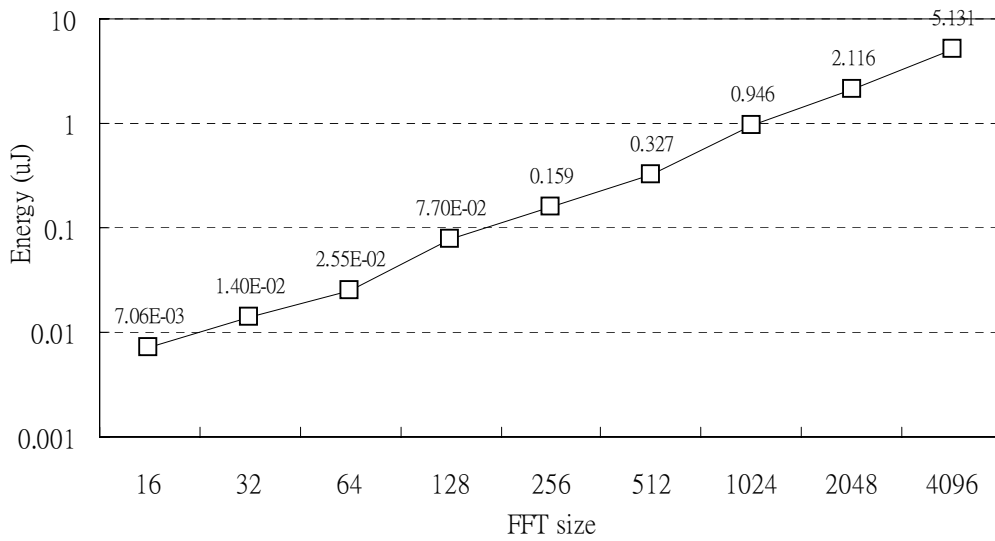
Referring to the execution cycles listed in TABLE 5.2, the energy dissipation for one FFT operation is shown in Figure 5.9. The RMR FFT shows good energy scalability over various FFT sizes.

TABLE 5.2 Execution cycles required for various FFT sizes

FFT size	16	32	64	128	256	512	1024	2048	4096
Execution cycles	6	10	18	37	71	139	278	552	1100



(a)



(b)

**Figure 5.9 Energy dissipation per FFT operation,
(a) in normal scale, (b) in log scale**

5.4.2 Comparison

The performance RMR FFT is compared to two other reconfigurable architectures, as shown in TABLE 5.3. One is the delay spread based architecture proposed in [5.12]. It is based on the radix-4 SDF architecture and can be

reconfigured as 16, 64, 256, or 1024-point FFT. The throughput rate of this pipeline-based architecture is the same as the input clock rate due to the single data path. The other one is the reconfigurable FFT processor proposed in [5.13]. It adopts the memory-based architecture and can be configured as from 16-point to 1024-point FFT.

The execution time for one FFT of the RMR FFT is sufficient fast compared to other architecture, as shown in TABLE 5.4. This is due to the utilization of parallel data paths and high mixed-radix algorithm. The energy of the two architectures can be scaled to the 0.13 μ m technology with the following relationship:

$$Scaled\ Energy = Energy \times \left(\frac{110MHz}{Frequency}\right) \times \left(\frac{0.13\mu m}{Technology}\right) \times \left(\frac{1.2V}{Voltage}\right)^2 \quad (5.1)$$

Since the dynamic energy dissipation is proportional to CV^2 and C scales approximately as linear to the technology. The scaled energy is compared in Figure 5.10 and Figure 5.11. The energy dissipation of the RMR FFT outperforms the other two when the FFT size grows large (>64). The energy saving compared to the reconfigurable radix-4 pipeline-based architecture [5.12] is 51%, 64%, and 80% for FFT of size 64, 256 and 1024 points respectively. On the other hand, the energy compared to the reconfigurable radix-2 memory-based architecture [5.13] is 17%, 62%, 82%, and 85% for FFT of size 128, 256, 512, 1024 points respectively.

TABLE 5.3 Comparison with other reconfigurable architectures

	RMR FFT (This work)	Hasan et al.[5.12]	Yutian Zhao et al.[5.13]
Algorithm	Reconfigurable Mixed-Radix	Radix-4	Radix-2
Architecture	Pipeline-based	Pipeline-based	Memory-based
Word length	16-bit	16-bit	16-bit
Technology	0.13 μ m	0.18 μ m	0.18 μ m
Clock Rate (R)	110MHz	20MHz	20MHz
Supply voltage	1.2V	1.8V	1.8V
Throughput Rate	4R	R	<R

TABLE 5.4 Execution cycles required per FFT

FFT size	16	32	64	128	256	512	1024	2048	4096
This work	6	10	18	37	71	139	278	552	1100
Hasan et al.[5.12]	16	/	64	/	256	/	1024	/	/
Yutian Zhao et al.[5.13]	32	80	192	448	1024	2304	5120	/	/

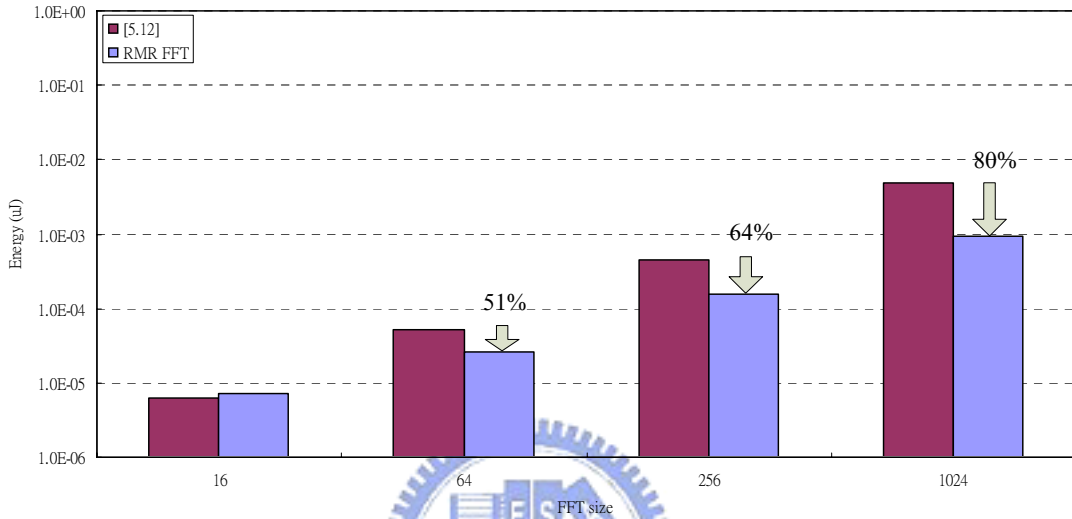


Figure 5.10 Comparison of Energy dissipation between RMR FFT and the other reconfigurable pipeline-based architecture

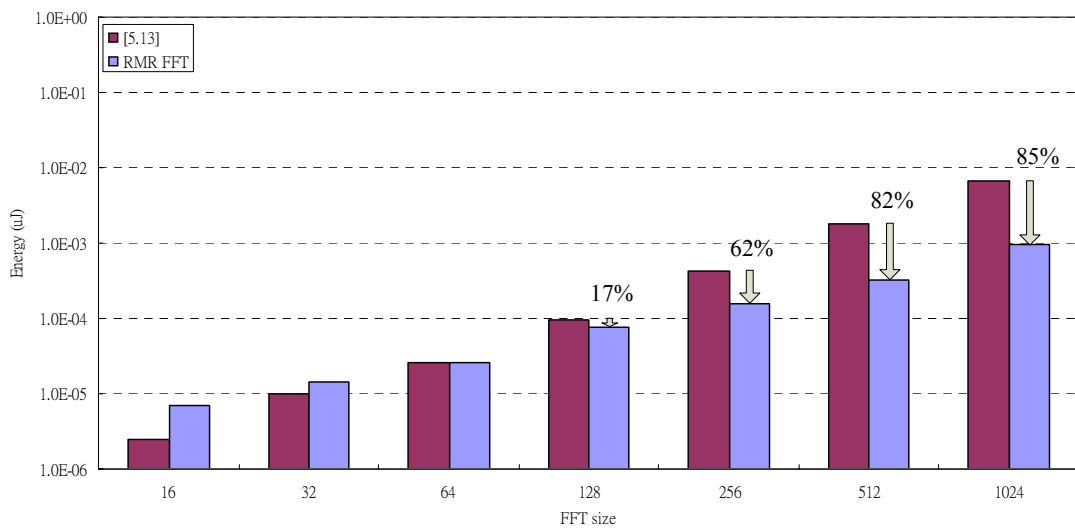


Figure 5.11 Comparison of Energy dissipation between RMR FFT and the other reconfigurable memory-based architecture

5.5 Layout Implementation

The layout view of the RMR FFT will be shown in this section. Figure 5.12 shows the layout view of the D flip-flop in the register banks. The dimension of the 1-bit flip-flop is $9.8\mu\text{m} \times 3.69\mu\text{m}$.

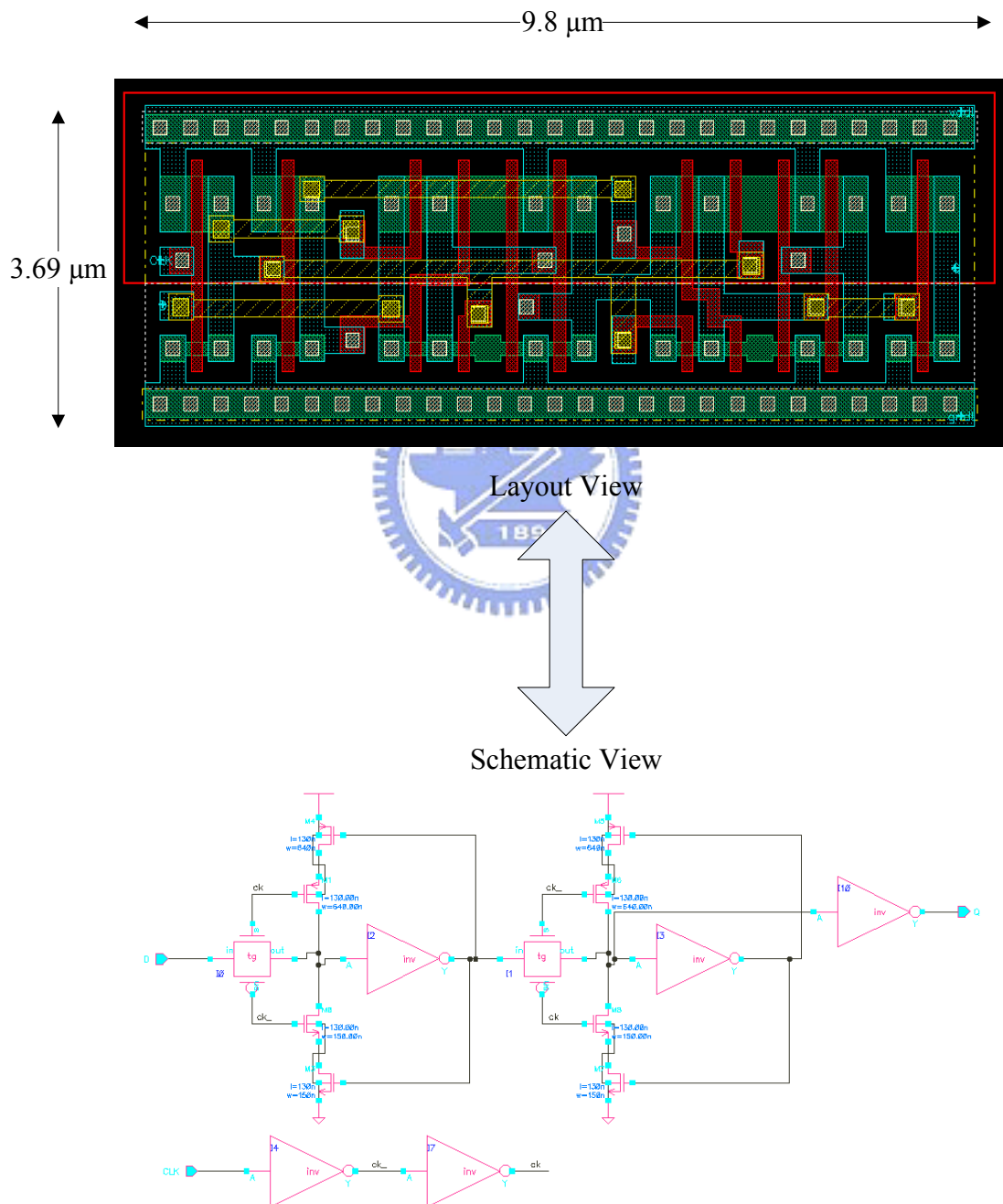


Figure 5.12 Layout and schematic view of the 1-bit D flip-flop

Figure 5.13 shows the layout view of a basic block in RB_512. The basic block consists of 64 D flip-flops. The dimension of the layout is $130\mu\text{m} \times 25\mu\text{m}$.

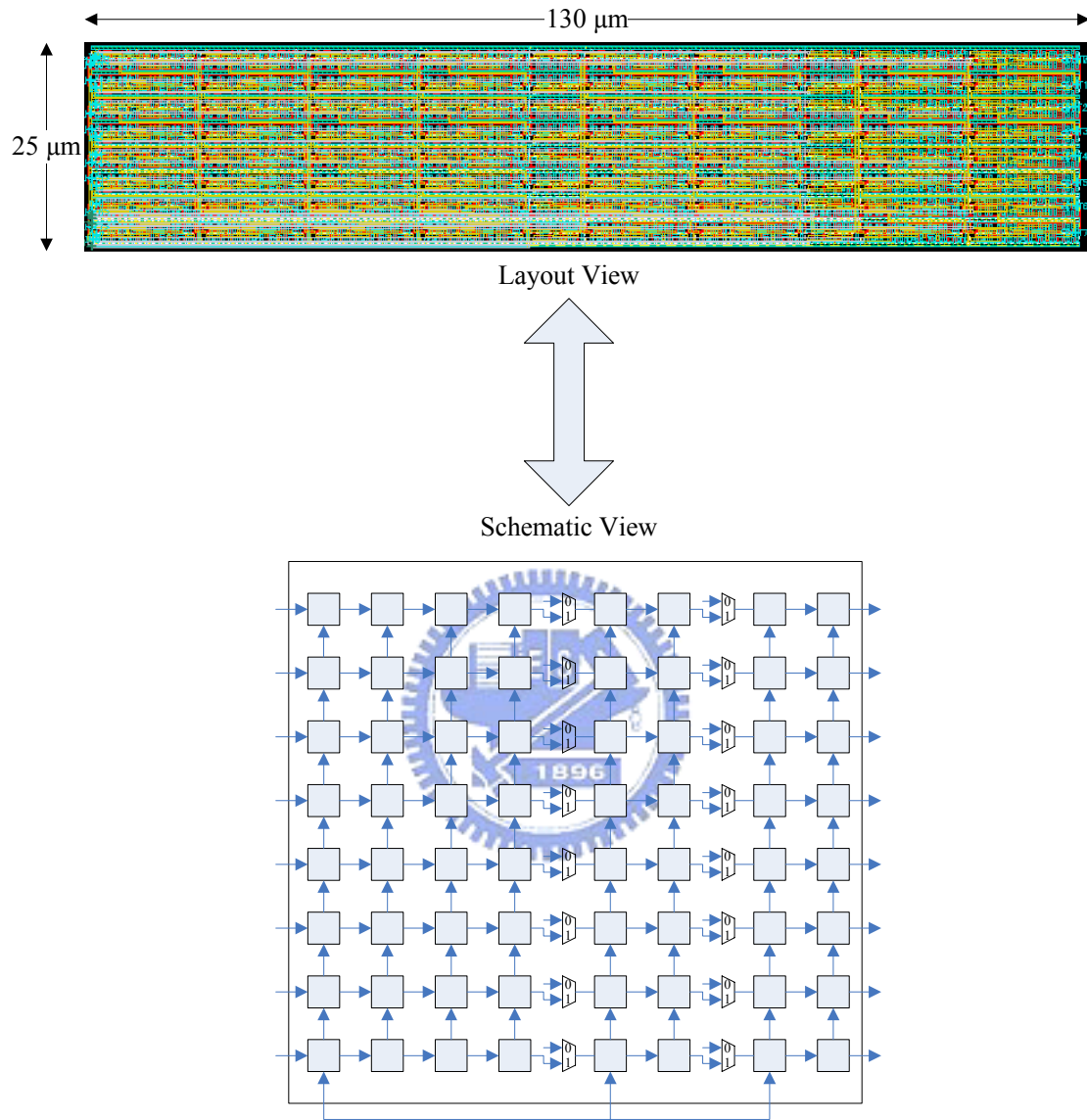


Figure 5.13 Layout and schematic view of a basic block in RB_512

Figure 5.14 shows the layout view of RB_512. As described before, the RB_512 consists of 8 basic blocks. With 16-bit wordlength, the total number of register in one RB_512 module is $16 \times 2 \times 512 = 16K$ bits. The dimension of the layout is $1050\mu\text{m} \times 830\mu\text{m}$.

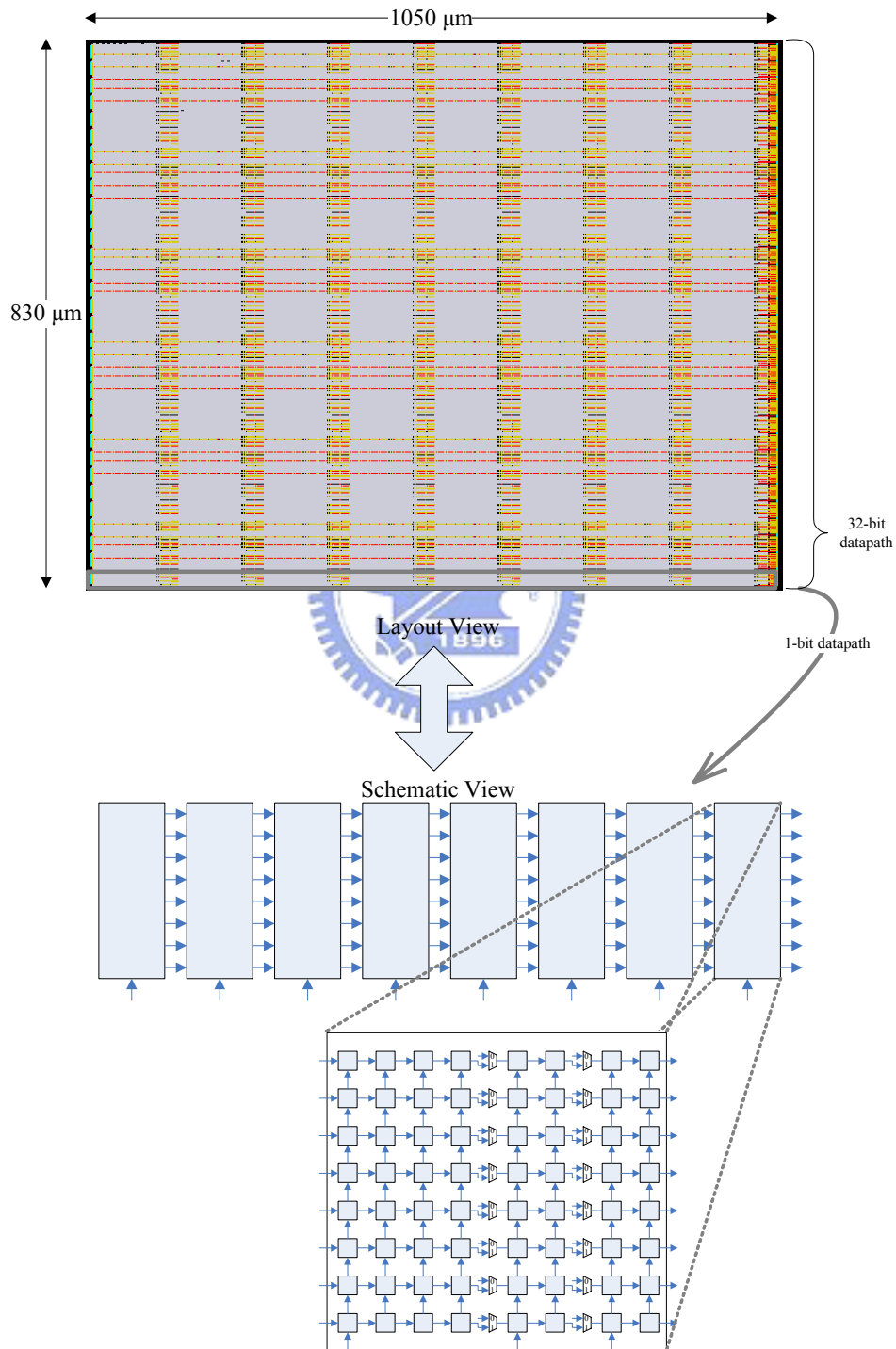


Figure 5.14 Layout and schematic view of RB_512

Figure 5.15 shows the layout view of RB_4096. The RB_4096 also consists of 8 basic blocks. The basic block capacity is 8 times that of the RB_512. With 16-bit wordlength, the total number of register in one RB_4096 module is $16 \times 2 \times 4096 = 128K$ bits. The dimension of the layout is $3750\mu\text{m} \times 1700\mu\text{m}$.

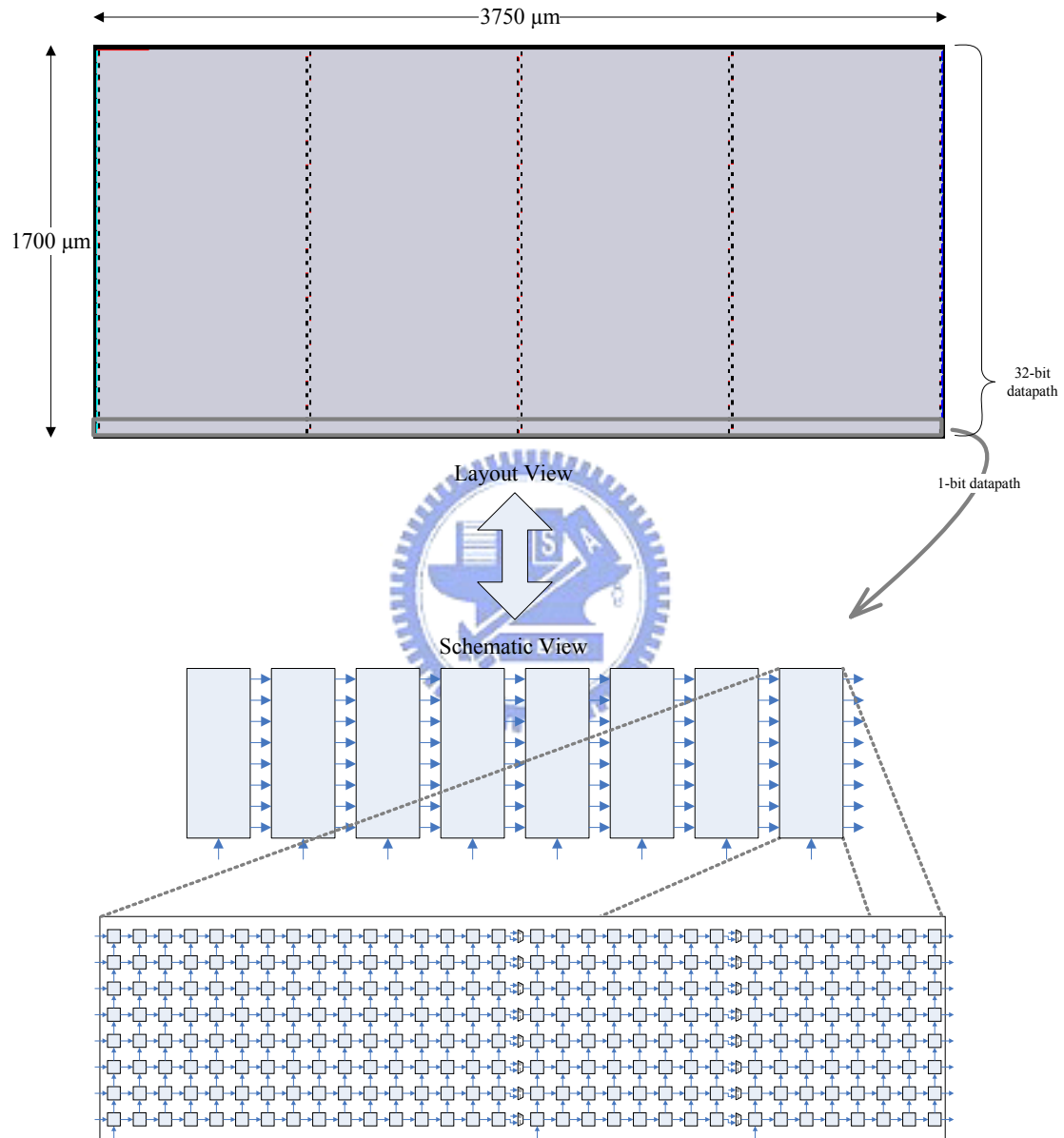


Figure 5.15 Layout and schematic view of RB_4096

Figure 5.16 shows the layout view of RMR FFT, except for RB_4096 and RB_512s. The layout is implemented through TSMC 0.13 μm CMOS technology. The dimension of the layout is 1650 μm \times 830 μm .

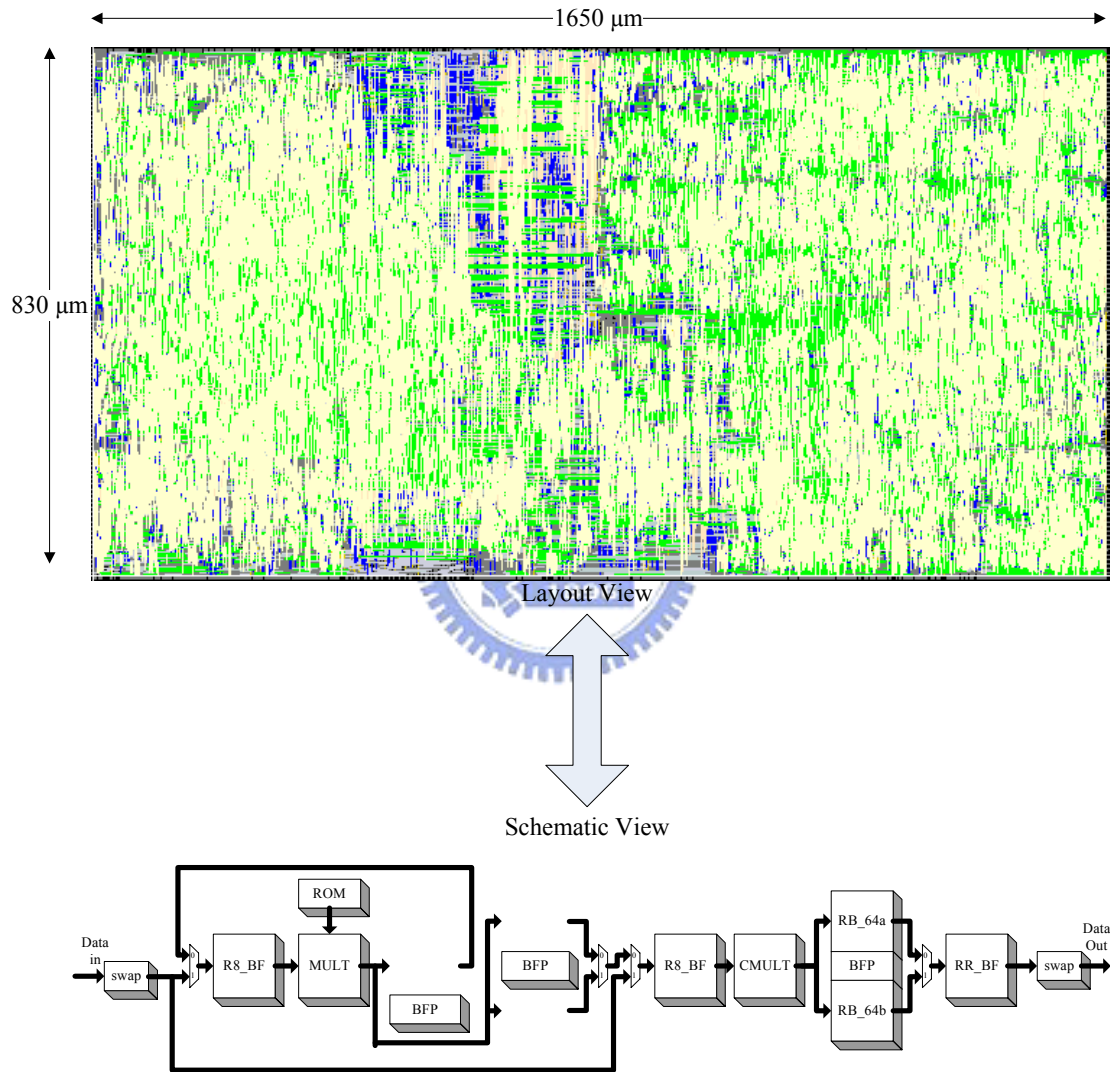


Figure 5.16 Layout and schematic view of the 1-bit D flip-flop

Figure 5.17 shows the layout view of the entire RMR FFT. Three RB modules are implemented through the full-custom design flow. The rest of the RMR FFT is implemented through cell-base synthesis flow using TSMC 0.13 μm CMOS technology. The figure shows that the memory (RB_4096, RB_512a, and RB_512b) takes over 80% of the FFT area. As the simulation result shows before, these register banks consume nearly 50% of the total power when they are all turned on. The dimension of the overall layout is 3750 μm \times 2530 μm .

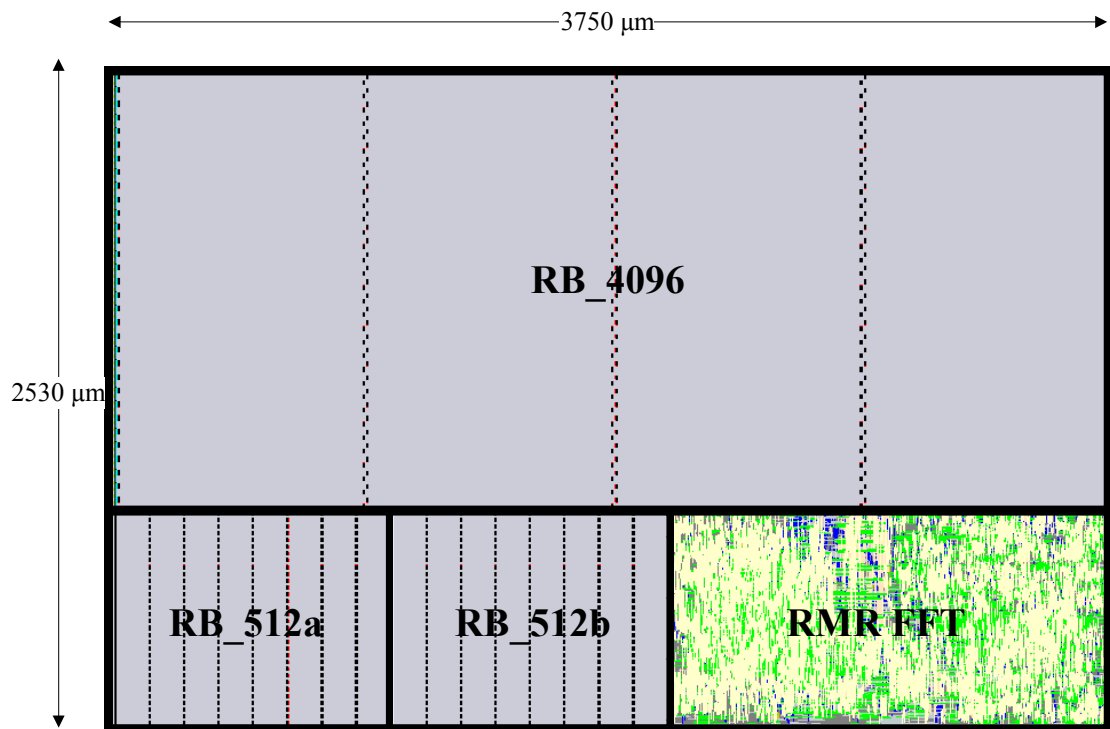


Figure 5.17 Overall Layout view of the proposed RMR FFT

5.6 Conclusions

In this chapter, we have shown the implementation strategy and the simulation results of the RMR FFT. The wordlength of the FFT is set to 16-bit, which means that 32-bit registers are required to store one complex word. Observing that large numbers of registers are used, we implement the RB modules through the full-custom design flow, which can further exploit the regularity of our RB design. External power

management is required for the power gating of each module for different FFT sizes.

The simulation result shows that our approach of adopting BFP in the RMR FFT does help maintain the SNR over 100 dB. Without the BFP approach, the SNR degrades rapidly as the FFT size grows. The simulation result also shows that the RMR FFT has good power scalability. The power distribution shows that the computation blocks consume most portion of power when the FFT size is small. However, the register banks become to dominate when the FFT size grows. This distribution clarifies that our approach to further optimize the RB modules is worthwhile.



Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis, we have proposed a novel reconfigurable FFT architecture, called the reconfigurable mixed-radix (RMR) FFT. The proposed architecture is able to reconfigured dynamically as from 16-point to 4096-point FFT/IFFT. The reconfigurable mixed-radix algorithm is based on the radix-2 DIF algorithm. Different mixed-radix algorithms are assigned for different FFT sizes while keeping the data ordering of each mode in same regularity. Also, unlike other pipeline-base architecture, the block-floating-point approach is adopted here to maintain the SNR. The BFP method help keeping the internal wordlength of data flow fixed which prevent the circuitry growing to big for long size FFTs. The simulation result shows that RMR FFT maintain the SNR above 110dB as the FFT size varies. Without the BFP, the SNR degrades rapidly as the FFT size grows.

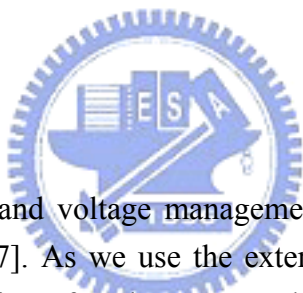
The pipeline-based hardware architecture achieve high throughput rate by using 8 parallel data paths in data flow. Internal register banks, instead of traditional commutators, are used to store the intermediate data between each computation stages. The butterfly (*BF*) and register banks (*RB*) modules are designed to be reconfigurable. Along with the bypassing multiplexers, the architecture can be reconfigured as from 16-point to 4096-point FFT without resulting in a long data path. The data flow goes in a way of block execution order in coordinate to the BFP approach. Duplicate register banks are inserted between middle stages to smoothen the data flow. Considering the hardware overhead, the register bank of the first stage is not duplicated which effectively result in a throughput rate as 4 times of the input clock rate. With good circuit partitioning, unnecessary blocks can be fully turned off when calculate smaller FFT without affecting the correct operation. With external power management unit, the RMR FFT can achieve a power scalable and energy aware design.

In implementation, the RMR FFT is to be synthesized using the TSMC 0.13 μ m

standard cell library. Considering that large bits of register required, the register banks are implemented through the full-custom flow in order to fully exploit its regularity. As the simulation result shows, the 16-point FFT takes only 6 clock cycles and energy dissipation is 4.34nJ/FFT while the 4096-point FFT takes 1100 clock cycles with energy dissipation 5.115 μ J/FFT. Compared to other reconfigurable architecture, the RMR FFT outperforms in execution speed and overall energy dissipation especially for long size FFTs. Comparing to other reconfigurable pipeline-based architecture, the energy saving is 51%, 64%, and 80% for 64, 256, and 1024-point FFT respectively. Comparing to other reconfigurable memory-based architecture, the energy saving is up to 85% for 1024-point FFT.

The RMR FFT meets the UWB standard, which requires throughput rate of 409.6 Msample/s [6.1], at clock rate of 110MHz. The proposed RMR FFT is especially suitable for modern high-speed and long-size FFT applications.

6.2 Future Work



The dynamic frequency and voltage management (DVFM) is popular in SOC design in recent years [6.2-6.7]. As we use the external power management unit to adjust supply voltage, the design of such PMU can be taken into consideration with the overall FFT design. Besides static power gating over different FFT size, more voltage-control techniques can be applied to the FFT processor, such as the multi-V_{dd} approach.

On the other hand, we assume that the IO data of our RMR FFT have been wrapped by external buffers. As shown in Figure 1.1, the OFDM requires a serial-to-parallel or a parallel-to-serial block to wrap the data. As the proposed RMR FFT is a general purpose FFT, it is also interesting to design reconfigurable wrapper for different FFT sizes and specifications that is suitable to various applications.

References

References of Chapter 1

- [1.1] Alan V. Oppenheim, Ronald W. Schfer, and John R. Buck, “**DISCRETE-TIME SIGNAL PROCESSING, Second Edition,**” *Prentice Hall International*, 1999.
- [1.2] J. W. Cooley and J. W. Tukey, “**An algorithm for machine computation of complex fourier series,**” *Math. Computation*, Vol. 19, pp. 297-301, Apr. 1965.
- [1.3] A. Batra et al., “**Multi-Band OFDM Physical Layer Proposal for IEEE 802.15 Task Group 3a,**” *IEEE P802.15-03/268r3*, Mar. 2004.



References of Chapter 2

- [2.1] J. W. Cooley and J. W. Tukey, “**An algorithm for machine computation of complex fourier series**,” *Math. Computation*, Vol. 19, pp. 297-301, Apr. 1965.
- [2.2] B.G. Jo and M.H. Sunwoo, “**New continuous-flow mixed-radix (CFMR) FFT Processor using novel in-place strategy**,” *IEEE Transactions on Circuits and Systems*, Vol. 52, pp. 911-919, May 2005.
- [2.3] L. Fanucci, M. Forliti, and F. Gronchi, “**Single-chip mixed-radix FFT processor for real-time on-board SAR processing**,” *IEEE International Conference on Electronics, Circuits and Systems*, Volume 2, pp. 1135 – 1138, Sept. 1999.
- [2.4] K.L. Heo, J.H. Baek, M.H. Sunwoo, B.G. Jo, and B.S. Sun, “**New in-place strategy for a mixed-radix FFT processor**,” *Proceedings IEEE International SOC Conference*, pp. 81 – 84, Sept. 2003.
- [2.5] J.H. Baek, B.S. Son, B.G. Jo, M.H. Sunwoo, and S.K. Oh, “**A continuous flow mixed-radix FFT architecture with an in-place algorithm**,” *Proceedings of the 2003 International Symposium on Circuits and Systems*, Volume 2, pp. II-133 - II-136, May 2003.
- [2.6] Wen-Chang Yeh and Chein-Wei Jen, “**High-speed and low-power split-radix FFT**,” *IEEE Transactions on Signal Processing*, Vol. 51, pp. 864 – 874, Mar. 2003.
- [2.7] S. Bouguezzel, M.O. Ahmad, and M.N.S. Swamy, “**Arithmetic complexity of the split-radix FFT algorithms**,” *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 5, pp. v/137 - v/140, March 2005.
- [2.8] Lihong Jia, Yonghong Gao, Jouni Isoaho, and Hannu Tenhunen, “**A new VLSI-oriented FFT algorithm and implementation**,” *Eleventh Annual IEEE International ASIC Conference*, pp. 337-341, Sept. 1998.
- [2.9] K. Maharatna, E. Grass, and U. Jagdhold, “**A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM**,” *IEEE Journal of Solid-State Circuits*, Vol. 39, pp. 484 – 493, Mar. 2004.
- [2.10] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee, “**A 1-GS/s FFT/IFFT processor for UWB applications**,” *IEEE Journal of Solid-State Circuits*, Vol. 40, pp. 1726-1735, Aug. 2005.
- [2.11] Shung-Chih Chen, Chao-Tang Yu, Chia-Lian Tsai, and Jing-Jou Tang, “**A new IFFT/FFT hardware implementation structure for OFDM applications**,” *IEEE Asia-Pacific Conference on Circuits and Systems*, Volume 2, pp. 1093-1096, Dec 2004.

- [2.12] Wei Han, A.T. Erdogan, T. Arslan, and M. Hasan, "**The development of high performance FFT IP cores through hybrid low power algorithmic methodology,**" *Proceedings of Asia and South Pacific Design Automation Conference*, Volume 1, pp.549-552, Jan. 2005.
- [2.13] Yun-Nan Chang and K.K Parhi, "**An efficient pipelined FFT architecture,**" *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Volume 50, Issue 6, pp. 322-325, June 2003.
- [2.14] J. Choi and V. Boriakoff, "**A new linear systolic array for FFT computation,**" *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, Issue 4, Volume 39, pp.236-239, April 1992.
- [2.15] Yunho Jung, Hongil Yoon, and Jaeseok Kim, "**New efficient FFT algorithm and pipeline implementation results for OFDM/DMT applications,**" *IEEE Transactions on Consumer Electronics*, Issue 1, Volume 49, pp. 14-20, Feb. 2003.
- [2.16] Yu-Wei Lin and Chen-Yi Lee, "**A new dynamic scaling FFT processor,**" *IEEE Asia-Pacific Conference on Circuits and Systems*, Vol. 1, pp. 449- 452, Dec. 2004.
- [2.17] Yu-Wei Lin, Hsuan-Yu Liu, and Chen-Yi Lee, "**A dynamic scaling FFT processor for DVB-T applications,**" *IEEE Journal of Solid-State Circuits*, Vol. 39, pp. 2005- 2013, Nov. 2004.
- [2.18] Haining Jiang, Hanwen Luo, Jifeng Tian, and Wentao Song, "**Design of an efficient FFT Processor for OFDM systems,**" *IEEE Transactions on Consumer Electronics*, Issue 4, Volume 51, pp. 1099-1103, Nov. 2005.
- [2.19] Shi Xin, Zhang Tiejun, and Hou Chaohuan, "**A high-performance power-efficient structure of FFT (fast Fourier transform) processor,**" *2004 7th International Conference on Signal Processing*, Volume 1, pp. 555-558, Sept. 2004.
- [2.20] Hasan, T. Arslan, and J.S. Thompson, "**A delay spread based low power reconfigurable FFT processor architecture for wireless receiver,**" *Proceedings International Symposium on System-on-Chip*, pp. 135-138, Nov 2003.
- [2.21] Yutian Zhao, A.T. Erdogan, and T. Arslan, "**A novel low-power reconfigurable FFT processor,**" *IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 41-44, May 2005.
- [2.22] Guichang Zhong, Fan Xu, and A.N. Willson, Jr., "**A power-scalable reconfigurable FFT/IFFT IC based on a multi-processor ring,**" *IEEE Journal of Solid-State Circuits*, Vol. 41, pp.483-495, Feb. 2006.

References of Chapter 3

[3.1] Alan V. Oppenheim, Ronald W. Schfer, and John R. Buck, “**DISCRETE-TIME SIGNAL PROCESSING, Second Edition,**” *Prentice Hall International*, 1999.

[3.2] S. Kobayashi and G.P. Fettweis, “**A new approach for block-floating-point arithmetic,**” *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 4, pp. 2009-2012, March 1999.

[3.3] S. Kobayashi, I. Kozuka, W.H. Tang, and D. Landmann, “**A software/hardware codesigned hands free system on a "resizable" block-floating-point DSP,**” *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 5, pp. V - 149-52, May 2004.

[3.4] A. Mitra, M. Chakraborty, and H. Sakai, “**A block floating-point treatment to the LMS algorithm: efficient realization and a roundoff error analysis,**” *IEEE Transactions on Signal Processing*, Issue 12, Volume 53, pp. 4536-4544, Dec. 2005.



References of Chapter 4

- [4.1] Wei Han, T. Arslan, A.T. Erdogan, and M. Hasan, “**Low power commutator for pipelined FFT processors,**” *IEEE International Symposium on Circuits and Systems*, Vol. 5, pp. 5274-5277, May 2005.
- [4.2] Guoan BI and E. V. Jones, “**A Pipelined FFT Processor for Word-Sequential Data,**” *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Vol. 37, pp. 1982-1985, Dec. 1989.
- [4.3] K. Maharatna, E. Grass, and U. Jagdhold, “**A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM,**” *IEEE Journal of Solid-State Circuits*, Vol. 39, pp. 484 – 493, Mar. 2004.
- [4.4] Shung-Chih Chen, Chao-Tang Yu, Chia-Lian Tsai, and Jing-Jou Tang, “**A new IFFT/FFT hardware implementation structure for OFDM applications,**” *IEEE Asia-Pacific Conference on Circuits and Systems*, Volume 2, pp. 1093-1096, Dec 2004.
- [4.5] T. Sansaloni, A. Perez-Pascual, V. Torres, and J. Valls, “**Efficient pipeline FFT processors for WLAN MIMO-OFDM systems,**” *Electronics Letters*, Volume 41, Issue 19, 15 September 2005.



References of Chapter 5

- [5.1] **TSMC 0.13 μ m (CL013G) Process 1.2-Volt SAGE-XTM Standard Cell Library Databook**, Release 2.6, January 2004.
- [5.2] V. Stojanovic and V.G. Oklobdzija, “**Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems**” *IEEE Journal of Solid-State Circuits*, Issue 4, Volume 34, pp. 536-548, April 1999.
- [5.3] A.S. Seyedi, S.H. Rasouli, A. Amirabadi, and A. Afzali-Kusha, “**Clock gated static pulsed flip-flop (CGSPFF) in sub 100 nm technology**,” *IEEE Computer Society Annual Symposium on Emerging VLSI Technologies and Architectures*, Volume 00, pp. 5, March 2006.
- [5.4] V. Stojanovic, V. Oklobdzija, and R. Bajwa, “**Comparative analysis of latches and flip-flops for high-performance systems**,” *International Conference on Computer Design: VLSI in Computers and Processors*, pp. 264 – 269, Oct. 1998.
- [5.5] Shang Xue and B. Oelmann, “**Comparative study of low-voltage performance of standard-cell flip-flops**,” *The 8th IEEE International Conference on Electronics, Circuits and Systems*, Volume 2, pp. 953 – 957, Sept. 2001.
- [5.6] S.H. Rasouli, A. Khademzadeh, A. Afzali-Kusha, and M. Nourani, “**Low-power single- and double-edge-triggered flip-flops for high-speed applications**,” *IEE Proceedings Circuits, Devices and Systems*, Issue 2, Volume 152, pp. 118-122, April 2005.
- [5.7] A. Fish, V. Mosheyev, V. Linkovsky, and O. Yadid-Pecht, “**Ultra low-power DFF based shift registers design for CMOS image sensors applications**,” *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems*, pp. 658-661, Dec. 2004.
- [5.8] Y. Suzuki, K. Odagawa, and T. Abe, “**Clocked CMOS calculator circuitry**,” *IEEE Journal of Solid-State Circuits*, Issue 6, Volume 8, pp. 462-469, Dec 1973.
- [5.9] H. Partovi, R. Burd, U. Salim, F. Weber, L. DiGregorio, and D. Draper, “**Flow-through latch and edge-triggered flip-flop hybrid elements**,” *IEEE International Solid-State Circuits Conference*, pp. 138-139, Feb. 1996.
- [5.10] J. Yuan and C. Svensson, “**High-speed CMOS circuit technique**,” *IEEE Journal of Solid-State Circuits*, Issue 1, Volume 24, pp. 62-70, Feb. 1989.
- [5.11] G. Gerosa, S. Gary, C. Dietz, Dac Pham, K. Hoover, J. Alvarez, H. Sanchez, P. Ippolito, Tai Ngo, S. Litch, J. Eno, J. Golab, N. Vanderschaaf, and J. Kahle, “**A 2.2 W, 80 MHz superscalar RISC microprocessor**,” *IEEE Journal of Solid-State Circuits*, Issue 12, Volume 29, pp. 1440-1454, Dec. 1994.

[5.12] Hasan, T. Arslan, and J.S. Thompson, "**A delay spread based low power reconfigurable FFT processor architecture for wireless receiver,**" *Proceedings International Symposium on System-on-Chip*, pp. 135-138, Nov 2003.

[5.13] Yutian Zhao, A.T. Erdogan, and T. Arslan, "**A novel low-power reconfigurable FFT processor,**" *IEEE International Symposium on Circuits and Systems*, Vol. 1, pp. 41-44, May 2005.



References of Chapter 6

- [6.1] A. Batra et al., “**Multi-Band OFDM Physical Layer Proposal for IEEE 802.15 Task Group 3a**,” *IEEE P802.15-03/268r3*, Mar. 2004.
- [6.2] Seokwoo Lee, S. Das, T. Pham, T. Austin, D. Blaauw, and T. Mudge, “**Reducing pipeline energy demands with local DVS and dynamic retiming**,” *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pp. 319-324, Aug. 2004.
- [6.3] Jian-Hau Wu and Wei Hwang, “**Dynamic Frequency and Voltage Management Design for Energy-Aware FFT Processor Application**,” *NCTU, Master Thesis*, Sep. 2004.
- [6.4] S. Raje and M. Sarrafzadeh, “**Variable Voltage Scheduling**,” *International Symposium on Low Power Design*, pp. 9-13, 1995.
- [6.5] J. Chang and M. Pedram, “**Energy Minimization Using Multiple Supply Voltages**,” *International Symposium on Low Power Electronics and Design*, pp. 157-162, 1996.
- [6.6] M. Igarashi, et. al., “**A Low-Power Design Method Using Multiple Supply Voltage**,” *International Symposium on Low Power Electronics and Design*, pp. 36-41, 1997.
- [6.7] A. Dancy and A. Chandrakasan, “**Techniques for aggressive supply voltage scaling and efficient regulation**,” *IEEE Custom Integrated Circuits Conf.*, pp. 579–586, 1997.