國 立 交 通 大 學

電 機 與 控 制 工 程 研 究 所

碩 士 論 文

適用於遞迴架構高點數快速傅利葉轉換之低點數快速傅利葉轉換設計之研究

Study on Short-Length FFT Design for Recursive Long-Length FFT Architecture

指導教授：董蘭榮　博士

研究生：吳智偉

中華民國九十五年七月

Study on Short-Length FFT Design for Recursive Long-Length FFT Architecture

Advisor: Dr. Lan-Rong Dung

Graduate Student: Chi-Wei Wu

July 2006

Graduate Institute of Electrical and Control Engineering

National Chiao Tung University

Hsinchu, Taiwan, ROC

Study on Short-Length FFT Design for Recursive Long-Length FFT Architecture

Graduate Student: Chi-Wei Wu Advisor: Dr. Lan-Rong Dung

Department of Electrical and Control Engineering

National Chiao Tung University

Abstract

With the growing trend on many specific applications adopting long-length FFT, the performance of FFT processors is more and more important. Generally, the long-length FFT is implemented with recursive short-length FFT in order to save the cost and complexity. This thesis presents a study on different short-length FFT designs for recursive architecture long-length FFT. Under recursive architecture, latency of each short-length FFT iteration will be the bottleneck of overall performance. A new structure with CORDIC (COordinate Rotation DIgital Computer) and DA (Distributed Arithmetic) technique is proposed in order to achieve low latency. A case study on realizing an 802.11a 64-point FFT processor is also presented. The specified FFT processor computes 16-bit input data at a throughput rate of 20MHz. The two chips, the proposed 64-point CORDIC-DA FFT processor and the other one with fully parallel 8-point FFT structure, were fabricated using TSMC 0.18-um single-poly six-metal CMOS process. After simulation, analysis, and refinement, we make a conclusion that the parallel 8-point FFT structure is most suitable for recursive architecture FFT.

適用於遞迴架構高點數快速傅利葉轉換之
低點數快速傅利葉轉換設計之研究

學生：吳智偉　　　　　　指導教授：董蘭榮　博士

國立交通大學
電機與控制工程學系研究所

摘要

隨著利用高點數快速傅利葉轉換的應用日益增多，快速傅利葉轉換處理器的效能越來越受到重視。一般而言，為了要節省成本與降低複雜度，高點數快速傅利葉轉換會以低點數快速傅立葉轉換的遞迴架構實現的。在此論文中提出了一份適用於遞迴架構高點數快速傅利葉轉換的低點數快速傅立葉轉換設計的研究。在遞迴架構下，每次低點數快速傅立葉轉換迴圈的延遲將會是整體效能的瓶頸所在。為了要達到更低的延遲，此論文中提出了一個新的使用座標旋轉與分散式運算的快速傅立葉轉換架構。此論文還以 801.11a 無線網路中 64 點快速傅立葉轉換處理器為例，提出了硬體實現的研究。此快速傅立葉轉換處理器的規格是 16 位元及 20MHz 的產出率。我們以 TSMC 0.18-um 製程製作出 2 顆的 64 點快速傅立葉轉換處理器，其中一顆是以我們所提出的架構為基礎，另一顆則是以傳統的平行 8 點快速傅立葉轉換為基礎。經過不斷的模擬、分析及改善後，我們下了一個結論：對於遞迴架構的高點數快速傅利葉轉換處理器，最適合以平行 8 點快速傅立葉轉換實現。

誌謝

本篇論文得以順利完成，首先要感謝的是我的指導教授——董蘭榮教授，在碩士班的兩年間，董教授不厭其煩地指導我，當我陷入瓶頸時，董教授亦適時地指點我正確的方向，做出修正，並且提供非常豐富的資源，讓我能好好潛心於學習研究，讓我在這兩年間獲益良多。

同時，也感謝實驗室的學長——學之、介皇、盟淳、顯文，在我的求學過程中給予指點與幫助，以及同學們——文豪、岳璋、泰佑、耕興，在課業與生活上的互相扶持、分擔紓解彼此的壓力，給了我一段美好的研究所時光。

最後要感謝我的家人的支持，有了你們的鼓勵，使我無後顧之憂，才能夠安心地完成碩士班學業。

謹將此論文獻給所有關心我的人，在此致上最深的謝意。

Contents

List of Tables

List of Figures

Chapter 1   Introduction

1.1   Introduction for Long-Length FFT

The fast Fourier transform (FFT) and inverse fast Fourier transform are key operations in modern communication systems, and the long-length FFT is commonly adopted in order to increase transmission bandwidth or efficiency, such as wireless LAN, ADSL, VDSL, and digital audio/video broadcasting systems, as shown in Table 1.1. By the growing trend towards longer length, FFT processors need more and more computing power, memory spaces, and hardware costs. There are many research works on short-length FFT processors have been done for several decades, but few on long-length FFT processors. There is still much space left for optimizing the implementations of long-length FFT.

Application	FFT/IFFT Size	Frequency spacing	T_{FFT}
WLAN	64	0.3125 MHz	3.2 μs
ADSL	2x256	4.3125 KHz	231 μs
VDSL	$2\times256\times2^n$,n=0~4	4.3125 KHz	231 μs
DAB	256×2^n,n=0~3	4.065×2^n KHz	31×2^n μs
DVB-T	8912 / 2048	1.116 / 4.464 KHz	896 / 224 μs

Table 1.1 Various FFT applications

1.2 Recursive Architecture Overview

In order to decrease the implementation complexity, recursive architectures are commonly used while structuring long-length FFT, as shown in Fig. 1.1

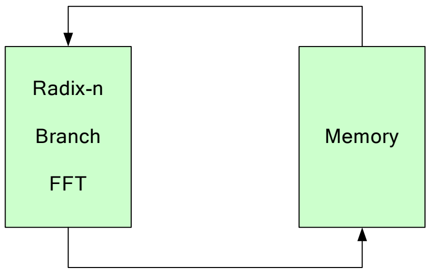


Fig. 1.1 Recursive architecture of radix-n FFT

In such a recursive architecture, the first data of the current iteration can not be loaded into the "branch" FFT until the last data of the previous iteration is done. Thus, the latency of the branch FFT will be a performance bottleneck. An approach to lower the latency between the stages in recursive architectures is required for better performance. Furthermore, the faster calculations are finished up, the lower power is consumed. It is also a way to increase energy efficiency in the meanwhile.

Numerous methods to optimize recursive architecture of long-length FFT were reported. In [11], a cache-memory architecture is adopted to enhance the performance of the memory system. A matrix prefetch buffer scheme is proposed in [12] that reorders the access of the memories between the stages in order to make sure the fluency of the dataflow because of the transposed order of the branch FFT. A COBRA FFT processor [10] uses an array architecture and is composed of multiple chips utilizing bit-serial arithmetic and dynamic reconfiguration. In this thesis, a study on algorithms and low-latency structures of short-length FFT for recursive long-length FFT is presented.

1.3 Design for Optimized Structure

To solve the issues on latencies of branch FFT, a new structure featured in CORDIC (COordinate Rotation DIgital Computer) and DA (Distributed Arithmetic) techniques is proposed. With the property of bit-serial computations, this structure can run at a high frequency (max. 427MHz) with low latency (4 clock cycles). The absolute latency is 9.36 ns.

However, for a 16 bit-parallel 8-point FFT computation, the latency increases to 20 clock cycles, and the absolute latency reaches 46.8 ns. Compared to the parallel 8-point FFT structure, the latter can run at max. 154MHz within 2 clock cycles of latency. The absolute latency is only 12.94 ns. In the view of power dissipation, the proposed structure consumes about 41.6~59.1 mW (depends on the scalability of datapaths, will be described in later chapters) at 20MHz in average, but the straight structure consumes only 12mW. We expected a parallel 8-point FFT to be inferior to the proposed structure because of its long latency at the inter-stage complex multipliers. Apparently, the proposed structure gains no benefits in non-bit-serial computations. A study on this issue is presented in this thesis.

All simulations are done with TSMC 0.18μm single-poly six-metal CMOS process.

1.4 Organization of This Thesis

The following is the summary of each chapter.

Chapter 2 Backgrounds

The main idea of the thesis, including FFT algorithm, CORDIC, and DA, is discussed in this chapter. Then, various implementations of the short-length FFT are described. Some of them are chosen as implementing candidates.

Chapter 3 Short-length FFT

Because of the long latency inside the branch FFT, including 8-point FFT and the twiddle factor rotation, the performance of recursive architectures is limited. The characteristics of several conventional implementations are shown in this chapter.

Chapter 4 Long-length FFT

In order to solve the performance issues while adopting a short-length FFT in recursive long-length FFT architectures, we looked for several methods to realize the branch FFT. A new implementation of the branch FFT with CORDIC and DA techniques is proposed.

Chapter 5 A Case Study: 802.11a wireless LAN 64-point FFT Processor

In order to verify the performance of the proposed architecture, several implementations of the 8-point branch FFT were developed. Then, two test chips are realized following the specification of the 802.11a wireless LAN 64-point FFT processor. The simulation results show that the parallel 8-point FFT is the best choice for implementing short-length FFT of recursive long-length FFT.

Chapter 6 Conclusion

Finally, we make a conclusion on our research works.

Chapter 2 Backgrounds

2.1 Discrete Fourier Transform

The N-point discrete Fourier transform (DFT) X(k) of a complex data sequence x(n) is defined as

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

where the twiddle factor is

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

If N is large, the number of MAC (Multiply and ACcumulate) operations described in (1) will be relatively large. Also, the multiplication required in (1) is a complex multiplication that consists of 4 real multiplications and 2 real additions. Decomposing (1) will help saving the computational costs. Let

$$\begin{aligned} &N = MT \\ &n = n_1 + Tn_2, \quad \begin{cases} n_1 \in \{0, 1, \ldots, T-1\} \\ n_2 \in \{0, 1, \ldots, M-1\} \end{cases} \\ &k = Mk_1 + k_2, \quad \begin{cases} k_1 \in \{0, 1, \ldots, T-1\} \\ k_2 \in \{0, 1, \ldots, M-1\} \end{cases} \end{aligned} \tag{3}$$

Applying the values in (3), (1) can be reformed as

$$\begin{aligned} & X(Mk_1+k_2) \\ &= \sum_{n_1=0}^{T-1}\sum_{n_2=0}^{M-1} x(n_1+Tn_2)W_N^{(n_1+Tn_2)(Mk_1+k_2)} \\ &= \underbrace{\sum_{n_1=0}^{T-1}\left\{\underbrace{\sum_{n_2=0}^{M-1} x(n_1+Tn_2)W_M^{n_2k_2}}_{\text{M-point DFT}}\underbrace{W_N^{n_1k_2}}_{\text{twiddle factor}}\right\}W_T^{n_1k_1}}_{\text{T-point DFT}} \end{aligned} \qquad (4)$$

By the derivation described in (4), one long-length DFT operation can be decomposed into multiple short-length DFT operations with additional twiddle factor rotations. Considering N to be power of *r*, the N-point DFT can be decomposed into $\log_r N$ stages of *r*-point DFT. This special case enables the availability of structuring recursive short-length DFT for long-length DFT.

The computational complexity of (1) is $O(N^2)$. With the FFT algorithm, the computational complexity can be reduced to $O(N\log_r N)$ where *r* means that the radix-*r* FFT operations are utilized. Table 2.1 [17] shows the number of real additions and multiplications required for an N-point FFT.

	Real Additions			Real Multiplications		
N	Radix-2	Radix-4	Radix-8	Radix-2	Radix-4	Radix-8
8	40		52	16		4
16	112	155		48	27	
32	296			136		
64	744	1011	972	360	243	204
128	1800			904		
256	4232	5635		2184	1539	
512	9736		12420	5128		3204
1024	22024	28931		11784	8451	
2048	49160			26632		

Table 2.1 Number of real additions and multiplications required for N-point FFT

For a 2-point and a 4-point FFT operation, the hardware can be implemented with only adders. An 8-point FFT operation can be implemented with adders and $1/\sqrt{2}$ constant scalars which causes hardware complexity a little higher. However, it decreases the overall complexity more while adopted in long-length FFT operations. Since the radix-*r* FFT with an *r* higher than 8, such as radix-16, decreases the overall complexity even more, the complexity of the branch FFT is much higher because of the need of complex multipliers.

2.2 Memory System Architectures

For a radix-*r* FFT algorithm, the hardware architecture of N-point FFT is decomposed into $\log_r N$ stages with *r*-point branch FFT. Each stage requires reading and writing to N data words, and memory access is considered to be one of the bottlenecks under the recursive structure of long-length FFT. The followings are memory system architectures previously proposed.

2.2.1 Single Memory

This is the simplest architecture that only one memory bank is connected to the branch FFT, as shown is Fig. 2.1. Additional input and output buffers are required while adopted in a real-time FFT processing.

2.2.2 Dual Memory

In this architecture, shown in Fig. 2.2, two memory banks are functioned as a set of ping-pong buffer so that it is capable to real-time FFT processing. $\log_r N$ times of iterations are required to complete an N-point FFT. Meanwhile, a clock rate higher than $\log_r N$ times of the sampling rate is also required.

2.2.3 Pipeline

In this architecture, the recursions are flattened. The computational resource costs are increased because of the requirements of $\log_r N$ branch FFT and $\log_r N+1$ buffer memory, as shown in Fig. 2.3. On contrast, the clock rate is comparatively low as the same frequency of the sampling rate to meet real-time FFT processing.

2.2.4 Array

Processors using an array architecture consist of a series of independent processing elements, buffers, and a communication networks, as shown in Fig. 2.4. For example, the COBRA processor [10] contains an array of radix-4 butterfly processors, an 128-element I/O memory, an 128-element data-exchange block, and an 128×128 crossbar matrix.

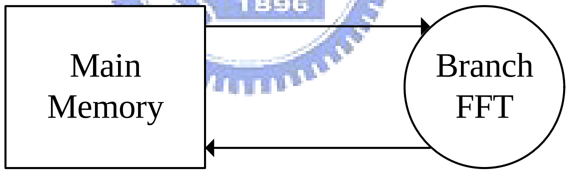


Fig. 2.1 Single-memory architecture block diagram

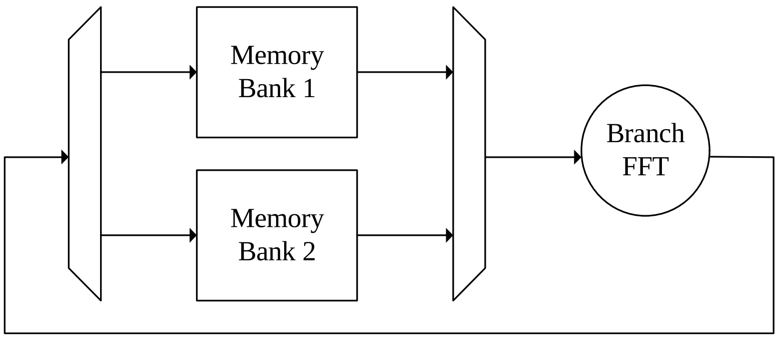


Fig. 2.2 Dual-memory architecture block diagram

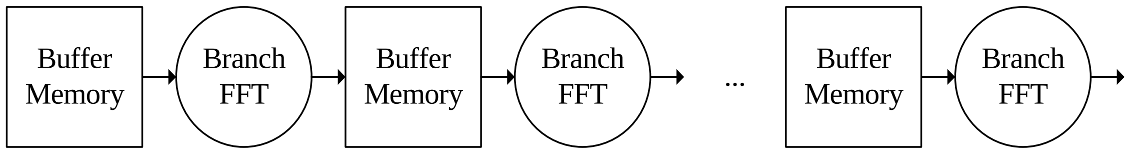


Fig. 2.3 Pipeline architecture block diagram

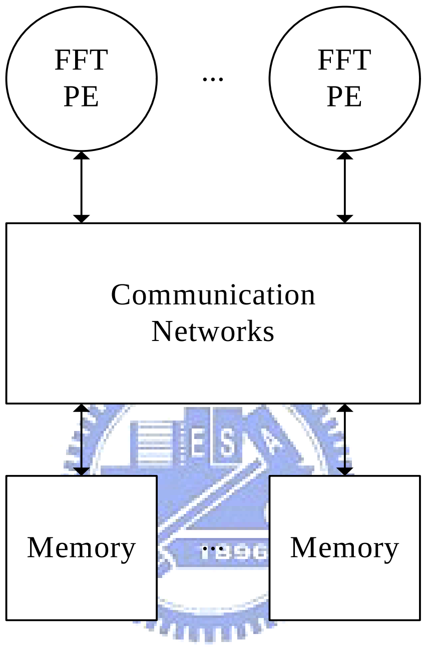


Fig. 2.4 Array architecture block diagram

2.3 8-point FFT Hardware Implementation

As shown in Table 2.1, a radix-8 FFT reduce the complexity more than other radix. Thus, it is chosen as the branch FFT for implementing long-length FFT in this thesis. There are many implementations have been proposed, such as radix-2 multi-path delay commutator (R2MDC) [30] and radix-2 single-path delay feedback (R2SDF) [21]. The FFT algorithm can be expressed in two forms, decimation in time (DIT) and decimation in frequency (DIF). Fig. 2.5 shows the signal flow graph of an 8-point DIT FFT. In this section, several 8-point FFT implementations in DIT form will be introduced.

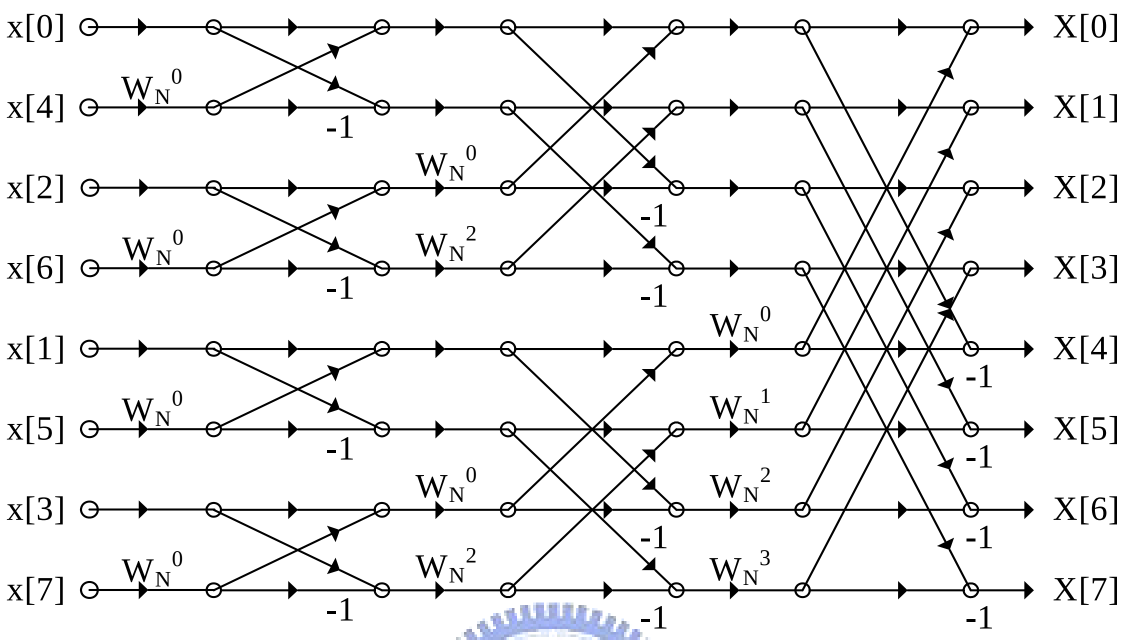


Fig. 2.5 Signal flow graph of 8-point DIT FFT

2.3.1 Parallel DIT Structure

In this structure, the hardware is as Fig. 2.5 shown. Every component is directly mapped and realized. It is the straightest structure but costs the most area resource. However, by great parallelism, it calculates all 8 outputs at the same time within the least latency.

2.3.2 Radix-2 Multi-path Delay Commutator (R2MDC)

This is a direct implementation of radix-2 FFT algorithm using pipeline structure. Fig. 2.6 outlines the block diagram of R2MDC. At first, the MUX switches up, and the first data are loaded into the buffer. Then, the MUX switches down, the second data passes through the path, the both operands of the first butterfly are ready, and the PE calculates the butterfly. The switches between stages operate in two modes, as shown in Fig. 2.7. In mode 1, the two inputs passes through; in mode 2, the inputs are swapped. With proper control, the switch can feed the right operands to next butterfly operation. In R2MDC, the utilization of the butterflies and delay buffers are 50% for each.

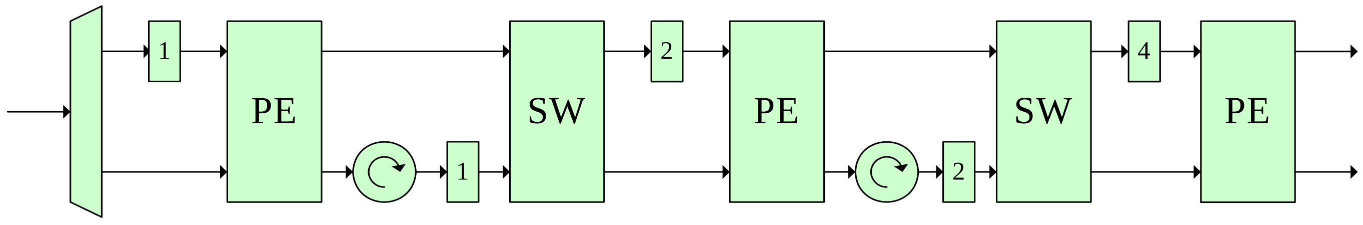


Fig. 2.6 R2MDC block diagram

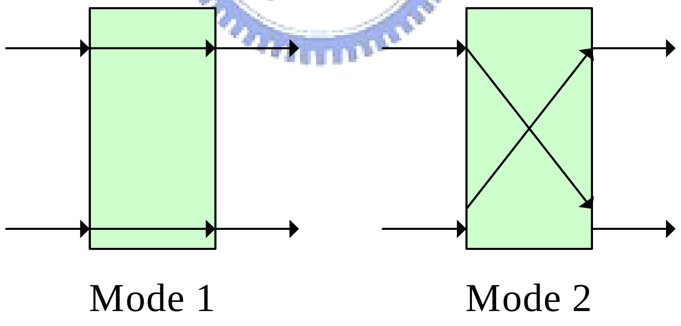


Fig. 2.7 R2MDC switch modes

2.3.2 Radix-2 Single-path Delay Feedback (R2SDF)

Since the utilization of buffers in R2MDC is only 50%, the R2SDF structure reduces its inter-stage buffer size by half. With feedback delay buffers, the utilization can reach 100%. Fig. 2.8 outlines the block diagram of R2SDF. In this structure, the PE holds two jobs: switching data and butterfly. Fig.2.9 shows the two operating modes of PE. In mode 1, the input data is passed to the buffer queue, and the data from the buffer is passed to the next stage with no modification; in mode 2, the PE calculates the butterfly from the input and the buffer, and the results are propagated to the next stage.

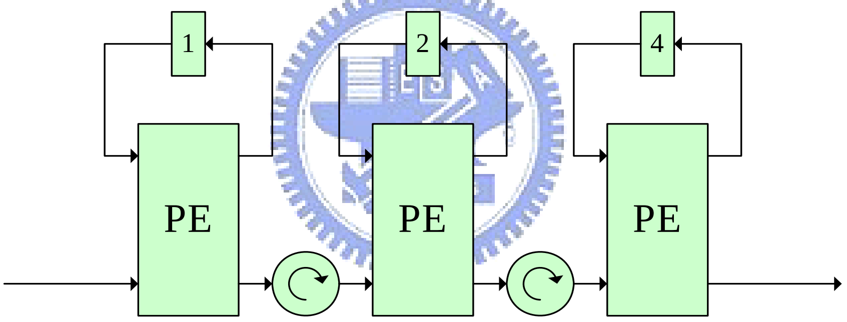


Fig. 2.8 R2SDF block diagram

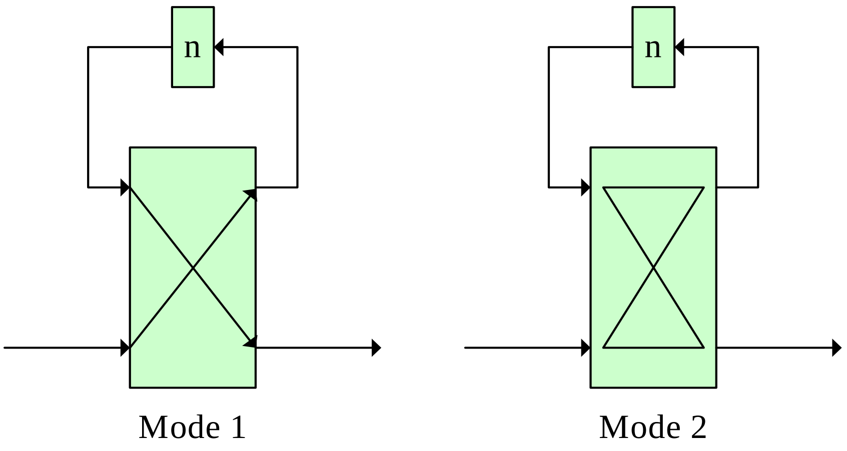


Fig. 2.9 R2SDF operation modes

2.4 CORDIC Overview

The CORDIC (COordinate Rotation DIgital Computer) which was developed by Volder [26] in 1959 is an iterative arithmetic algorithm for phase rotation using a unified shift-add approach [2]. The concept of the CORDIC algorithm is to decompose the desired angle into weighted sum of a set of predefined elementary rotation angles such that the angles can be accomplished with simple shift-add operations. Fig. 2.10 is an example of the CORDIC algorithm with linear coordinate systems. Let the desired angle θ be represented as

$$\theta = \sum_{i=0}^{n-1} \mu_i a(i) \tag{5}$$

where μ_i represents the rotation signs conventionally with set {-1,1} and the i^{-th} elementary rotation angle *a* (*i*) is defined as

$$a(i) = \tan^{-1} 2^{-i} \tag{6}$$

With the above definitions, the CORDIC algorithm can be described as an iterative equation as follows

$$\begin{bmatrix} x(i+1) \\ y(i+1) \end{bmatrix} = \begin{bmatrix} 1 & -\mu_i 2^{-i} \\ \mu_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \end{bmatrix} \tag{7}$$

(7) shows a simple implementation of a CORDIC iteration with a shift-add structure, illustrated in Fig. 2.11. The elementary rotation angles described in (6) are not normalized while *i* >1. To make sure the final coordinate $[x_f \; y_f]^T$ is normalized, the scale factor K is defined as

$$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = K \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} = \frac{1}{\prod_{i=0}^{n-1} \sqrt{1+\mu_i^2 2^{-2i}}} \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} \tag{8}$$

where $x(n)$ and $y(n)$ are the output of the last iteration.

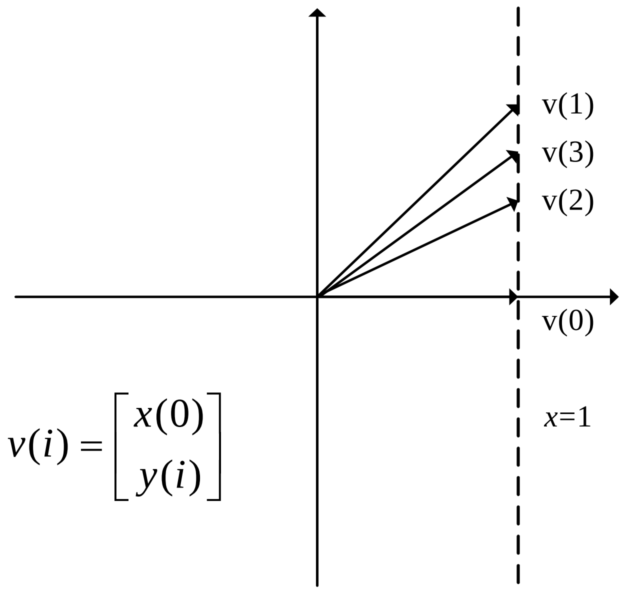


Fig. 2.10 Linear CORDIC rotations

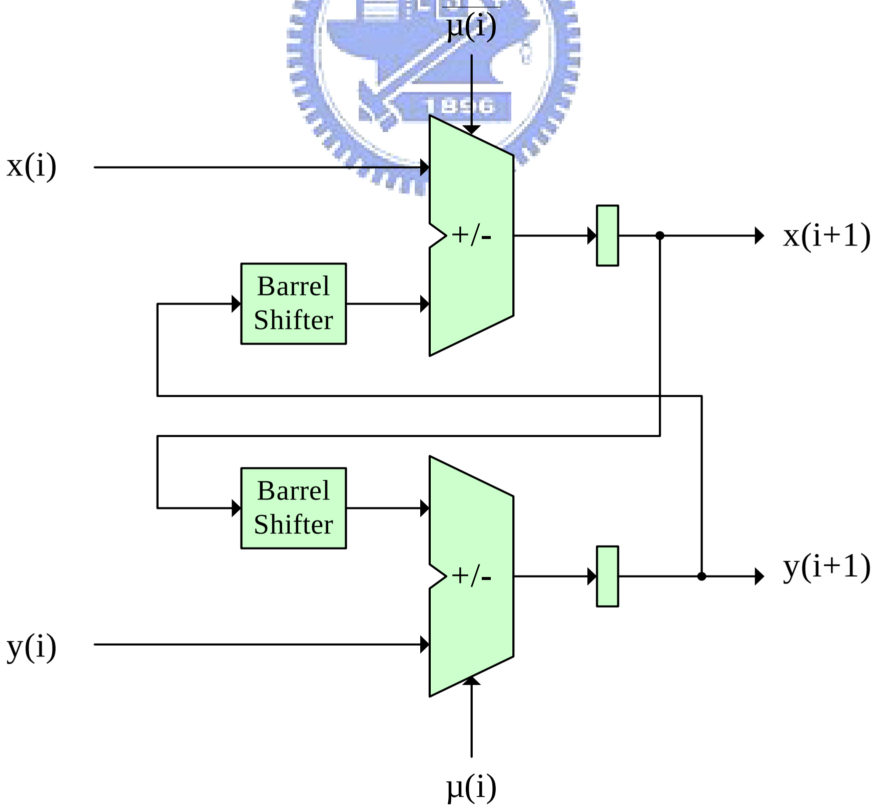


Fig. 2.11 Block diagram of basic single CORDIC iteration

The basic CORDIC algorithm [2] can be described as follows:

```
Initiation: Given x(0), y(0), z(0).
For i=0 to n-1, Do
    /* CORDIC iteration equation */
    $\begin{bmatrix} x(i+1) \\ y(i+1) \end{bmatrix} = \begin{bmatrix} 1 & -\mu_i 2^{-i} \\ \mu_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x(i) \\ y(i) \end{bmatrix}$
    /* Angle updating equation */
    $z(i+1)=z(i)-\mu_i a_m(i)$
End i-loop
/* Scaling Operation */
$\begin{bmatrix} x_f \\ y_f \end{bmatrix} = K \begin{bmatrix} x(n) \\ y(n) \end{bmatrix} = \frac{1}{\prod_{i=0}^{n-1} \sqrt{1+\mu_i^2 2^{-2i}}} \begin{bmatrix} x(n) \\ y(n) \end{bmatrix}$
```

Generally, the rotation angle θ is known in many DSP applications. Thus, the rotation signs μ_i can be precomputed rather than online computation. The calculation is done by (7) and the value of is obtained as

$$\mu_i = \begin{cases} 1 & \text{if} \quad x(i) \geq 0 \\ -1 & \text{if} \quad x(i) < 0 \end{cases} \tag{9}$$

For a fixed number of CORDIC iterations *i*, the scale factor *K* will also be a constant.

The redundant CORDIC proposed by Erocegovac and Lanf [22] is a modified version of the CORDIC method. In this algorithm, the rotation signs μ_i can be taken from the set {-1, 0, 1} instead of the set {-1, 1} defined as

$$\mu_i = \begin{cases} 1 & \text{if} \quad 2^i x(i) \geq \frac{1}{2} \\ 0 & \text{if} \quad \frac{1}{2} > 2^i x(i) > -\frac{1}{2} \\ -1 & \text{if} \quad 2^i x(i) \leq -\frac{1}{2} \end{cases} \tag{10}$$

if μ_i = 0 during this iteration, no computation is required. In this method, the computational costs of certain CORDIC iterations may be saved. However, these zero-rotation angles need no normalization because no rotations are completed as a matter of fact. Thus, it also causes variable scale factors. Many schemes are proposed to compensation the variable scale factors. A double rotation method [23] can reduce number of iterations while keeping constant scale factors. A differential CORDIC method [24] is proposed to implement redundant constant scale factor without correcting iterations. A prediction rotation method with the concept of Wallace tree, Booth encoding, and termination algorithm is proposed to speedup redundant addition while keeping a constant scaling factor [25].

Compared to complex-multiplier-based phase rotators, the standard CORDIC method possesses the advantage of smaller area, higher speed, and better capability for pipelined architecture, but pays longer latency because of its iterative mechanization. The redundant CORDIC method can avoid unnecessary computation while rotating the vectors so that latency will be reduced. But it also introduces the issues on variable scale factors and required additional devices to handle.

2.5 DA Overview

DA (Distributed Arithmetic) is a technique to calculate a sum of product more efficiently. Basically it is a bit-serial arithmetic algorithm. Considering an example of a sum of products:

$$y = \sum_{k=1}^{K} A_k x_k \tag{11}$$

where A_k are fixed coefficients, and x_k are the input data words. If x_k is a 2'complement binary number, then it can be expressed as

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \tag{12}$$

where b_{kn} is the bits, b_{k0} is the sign bit, and b_{kN-1} is the LSB. Replacing (11) with (12), one will get

$$y = -\sum_{k=1}^{K} A_k b_{k0} + \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k b_{kn} \right] 2^{-n} \tag{13}$$

Notice that whether b_{kn} or b_{k0} only takes on values of 0 and 1, the bracketed term in (13):

$$\sum_{k=1}^{K} A_k b_{kn} \tag{14}$$

has only 2^k possible values. These values can be precomputed and stored in ROM rather than run-time computed. Thus, a ROM of lookup tables and a shift-add accumulator form the basic DA mechanization. The input data are fed from LSB to MSB in a bit-serial style, and the accumulator adds the value from LUT (LookUp Table) to the 1-bit right-shifted value of the previous accumulation. The accumulator changes its operating mode from an adder to a subtractor while the incoming bits are the sign bits. Here is an example of a sum of product with 4-word input data, as shown in Fig. 2.12. The T_s control signal is asserted while the current bit-serial input is MSB, as sign bit, and the adder switches to the subtractor.

DA is a very efficient means to computations that are dominates by inner products [4], especially for constant coefficients. Whenever the performance / cost ratio is critical, DA should be taken into consideration.

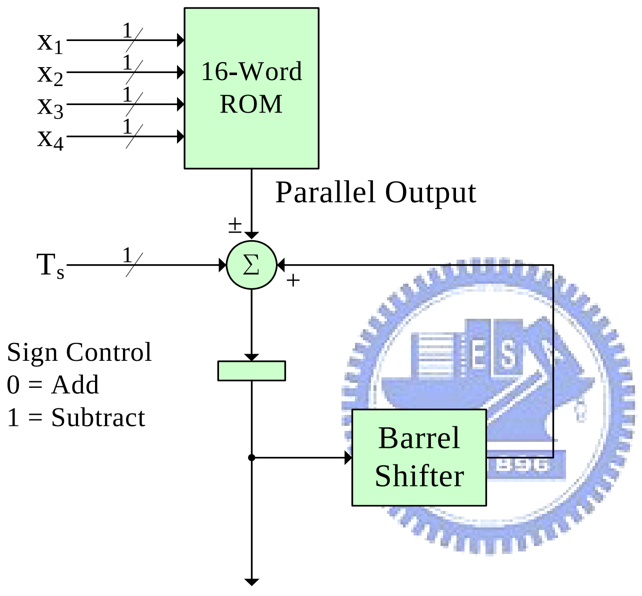


Fig. 2.12 DA mechanization of 4-word input data

Input Code b_{1n}	b_{2n}	b_{3n}	b_{4n}	16-word Memory Contents
0	0	0	0	0
0	0	0	1	A_4
0	0	1	0	A_3
0	0	1	1	A_3+A_4
0	1	0	0	A_2
0	1	0	1	A_2+A_4
0	1	1	0	A_2+A_3
0	1	1	1	$A_2+A_3+A_4$
1	0	0	0	A_1
1	0	0	1	A_1+A_4
1	0	1	0	A_1+A_3
1	0	1	1	$A_1+A_3+A_4$
1	1	0	0	A_1+A_2
1	1	0	1	$A_1+A_2+A_4$
1	1	1	0	$A_1+A_2+A_3$
1	1	1	1	$A_1+A_2+ A_3+A_4$

Chapter 3　Short-Length FFT

3.1　Introduction

In the previous chapter, the radix-8 FFT algorithm and recursive architecture are described. In the recursive architecture, the first data of the current iteration can not be loaded into the branch FFT until the last data of the previous iteration is done. Thus, the timing delay between stages under such a recursive architecture is limited by the latency of branch FFT processors. As the number of iterations grows, a branch FFT with lower latency will result in a faster completion. Besides, under the same specification, an architecture, which completes calculations sooner, will stay in sleep longer and save more power. Therefore, a low latency branch FFT is required for a recursive architecture long-length FFT.

The latency of branch FFT is mainly composed of two parts: 8-point FFT and inter-stage twiddle factor rotations. In this chapter, the objective is to resolve issues about latency in 8-point FFT, and the proposed 8-point FFT structure with CORDIC and DA technique is described. Issues about twiddle factor rotation are left for later chapters. All simulations are done with TSMC 0.18μm single-poly six-metal CMOS process.

3.2 Latency in 8-point FFT

Consider a straight parallel 8-point FFT implementation. Each radix-2 butterfly operation produces 1 adder delay time. Table 3.1 shows the rotation matrix of W_8^n twiddle factors. Only W_8^1, W_8^2, and W_8^3 are performed in an 8-point FFT. W_8^0 does no computations and can be ignored. Fig. 3.1 shows a W_8^2 rotation following a butterfly operation. The lower output signal B’ can be reformulated as:

$$\begin{aligned} B' &= [(A_R - B_R) + j(B_I - A_I)] \times (-j) \\ &= (A_I - B_I) + j(B_R - A_R) \end{aligned} \qquad (15)$$

This modification can be done by simply swap the operands of subtractions. Thus, the rotation can be integrated into the butterflies with no extra computational cost. One $1/\sqrt{2}$ constant scalar and 1 adder delay time are required in W_8^1 and W_8^3. The total computational delay of the critical path in a parallel 8-point FFT is 4 adders and 1 constant scalar, as shown in Fig. 3.2.

n	Rotation Matrix	n	Rotation Matrix
0	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	4	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$
1	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$	5	$\frac{1}{\sqrt{2}}\begin{bmatrix} -1 & -1 \\ 1 & -1 \end{bmatrix}$
2	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	6	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$
3	$\frac{1}{\sqrt{2}}\begin{bmatrix} -1 & 1 \\ -1 & -1 \end{bmatrix}$	7	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix}$

Table 3.1 Rotation matrix of W_8^n

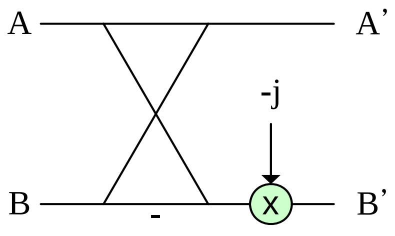


Fig. 3.1 W_8^2 rotation following a butterfly

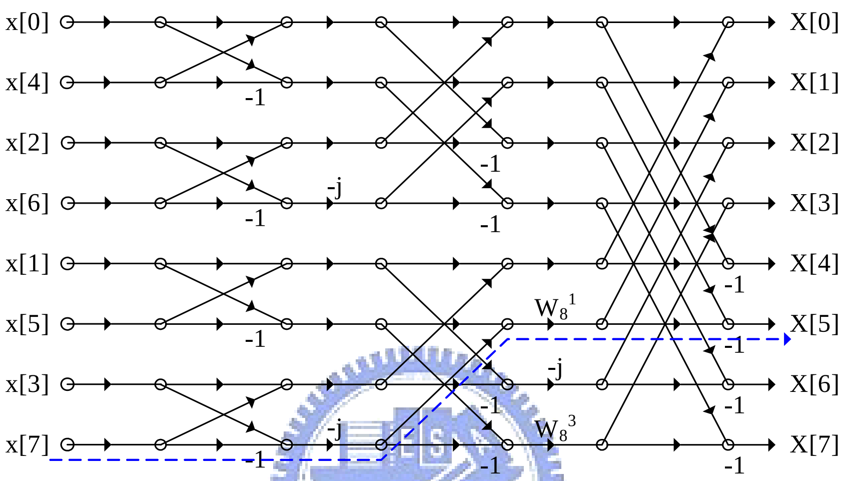


Fig. 3.2 Critical path in an 8-point DIT FFT

3.3 CORDIC-DA Structure

The FFT algorithm saves the computational costs because it makes use of dependencies between stages of radix-2 FFT operations. However, the dependencies become a source of latency while being implemented into hardware. Considering the original DFT definition shown in (1), a DFT operation is intrinsically a sum-of-product operation. For an 8-point DFT, the latency is the delay time of one 2-operand adder, one 8-operand adder, and one constant scalar. If there is an implementation with a better approach to summations and constant scaling, the latency can be reduced. The proposed CORDIC-DA structure may possibly reach this goal.

An 8-point DFT is defined as

$$X[k]=\sum_{n=0}^{7}W_8^{nk}x[n] \tag{16}$$

where W_8^{nk} is the twiddle factor. Implementing the twiddle factor with CORDIC method, (16) can be expressed as

$$\begin{bmatrix}\mathrm{Re}\{X[k]\}\\ \mathrm{Im}\{X[k]\}\end{bmatrix}=\sum_{n=0}^{7}K(n,k)M_{2\times 2}(n,k)\begin{bmatrix}\mathrm{Re}\{x[n]\}\\ \mathrm{Im}\{x[n]\}\end{bmatrix}$$

$$K(n,k)=\begin{cases}\dfrac{1}{\sqrt{2}} & \text{if n and k are both even}\\ 1 & \text{otherwise}\end{cases} \tag{17}$$

where *K(n,k)M$_{2x2}$(n,k)* is the rotation matrix shown in Table 3.1. In the view of the redundant CORDIC method, *K(n,k)* is the scale factor of CORDIC, and *M$_{2x2}$(n,k)* is the pure additions of CORDIC. There are 2 CORDIC iterations: $\pi/2$ rotation and $\tan^{-1}1$ rotation. Notice that certain twiddle factors in () require no $\tan^{-1}1$ rotation, thus the scale factor *K(n,k)* has two possible values: 1 and $1/\sqrt{2}$, as described in (17), and is a variable scale factor. The concept of the CORDIC-DA structure is that the CORDIC stage processes rotations without scale factor; the DA stage handles the variable scale factor and summation.

The block diagram of the proposed structure is illustrated in Fig. 3.3. It is a pipelined 8-point FFT processor. There are 3 stages, the P/S converter (Parallel-to-Serial converter), the CORDIC stage, and the DA stage. Between the CORDIC stage and the DA stage, there exists a series of pipeline registers that separate the two stages. The P/S converter contains total 16 sets of N-bit shift registers, 2 sets for real and imaginary part each operand. They load 8 data words from the memory buffer, outputs the LSB to the DA stage, and then right shift 1-bit in next cycle. The shift operation is a circular shift that the MSB in next cycle will be the content of the LSB in previous cycle. Because shift operations consume heavy power, a 'shift' control signal is designed to control the right shift operation. When the pipeline is inactive, the right-shift operation will be stopped.

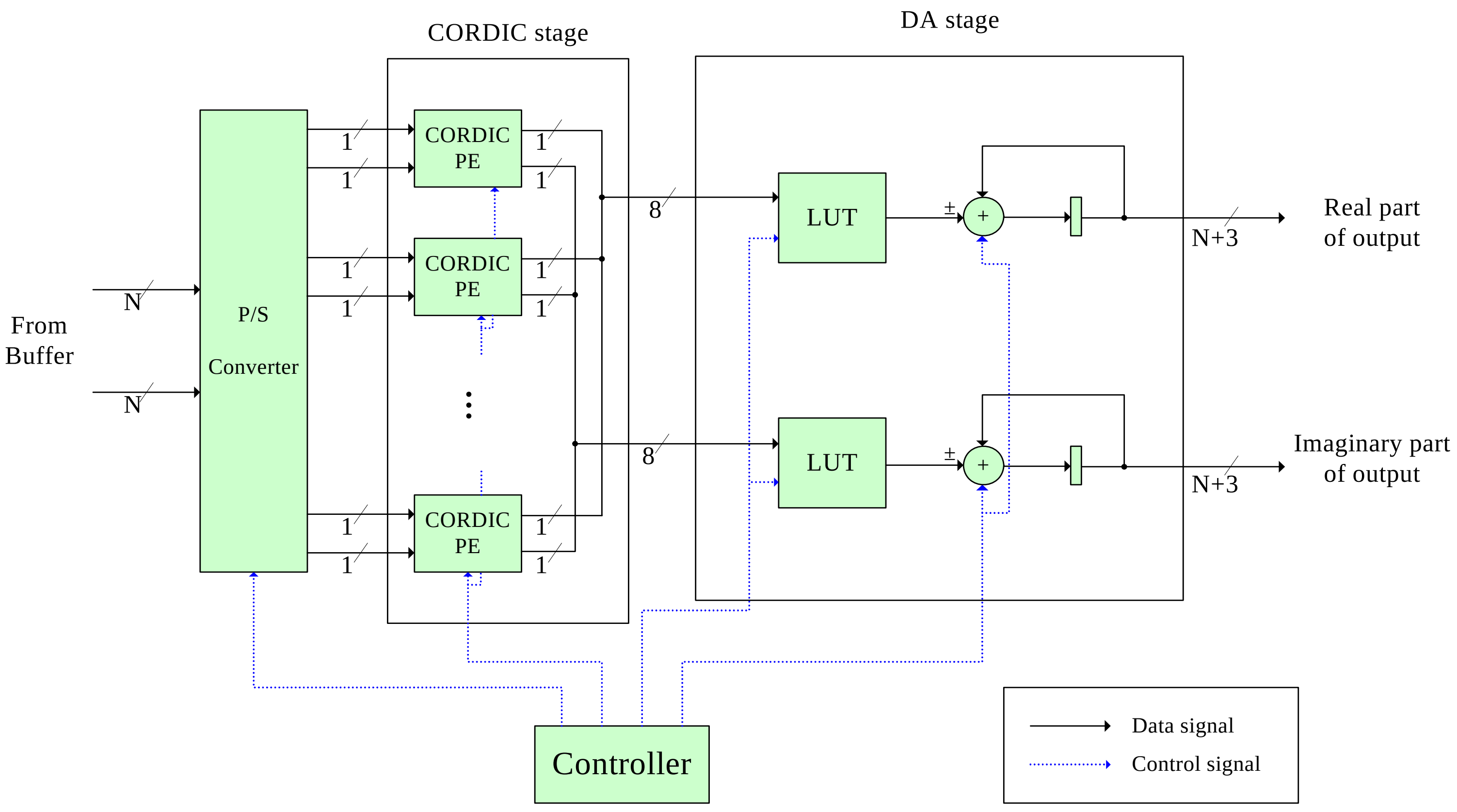


Fig. 3.3 Block diagram of 8-point CORDIC-DA FFT

Originally, the P/S converter is placed in front of the DA stage. However, if the P/S converter is moved behind the CORDIC stage, the CORDIC PE can be utilized with bit-serial arithmetic that extremely saves the cell area. For example, if a 16-bit adder is transformed into a 1-BAAT (Bit At A Time) bit-serial adder, only 1-bit adder and few flip-flops are required, as shown in Fig. 3.4. When 'LSB' signal is asserted, an external carry-in signal C_0 is transmitted to the carry-in of the full adder, otherwise the carry-out of the previous cycle stored in the register is transmitted. It computes sum like a RCA (Ripple-Carry Adder) does, but in a bit-serial style. Theoretically, N cycles are required to complete a bit-serial CORDIC computation for N-bit input. An extra 1-bit guard bit is added in order to prevent from overflow thus total N+1 cycles are required.

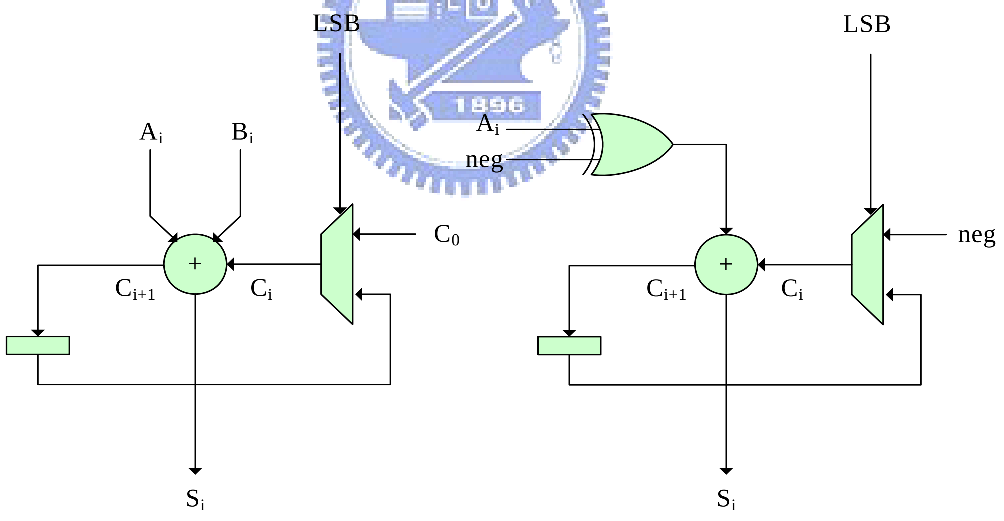


Fig. 3.4 (a) 1-BAAT bit-serial adder (b) 1-BAAT 2's complementor

The CORDIC stage is composed of a CORDIC PE array, as shown in Fig. 3.5. Total 8 CORDIC PEs are required for an 8-point FFT. Fig. 3.5 shows the block diagram of the CORDIC PE. Each PE consists of 4 OR gates, 4 bit-serial 2's complementors, and 2 bit-serial adders. There is a controller that centralizes the control signals which are independent of each other. With proper control signals shown in Table 3.2, the PEs can implement all W_8^n rotations shown in Table 2.1, except the $1/\sqrt{2}$ scaling.

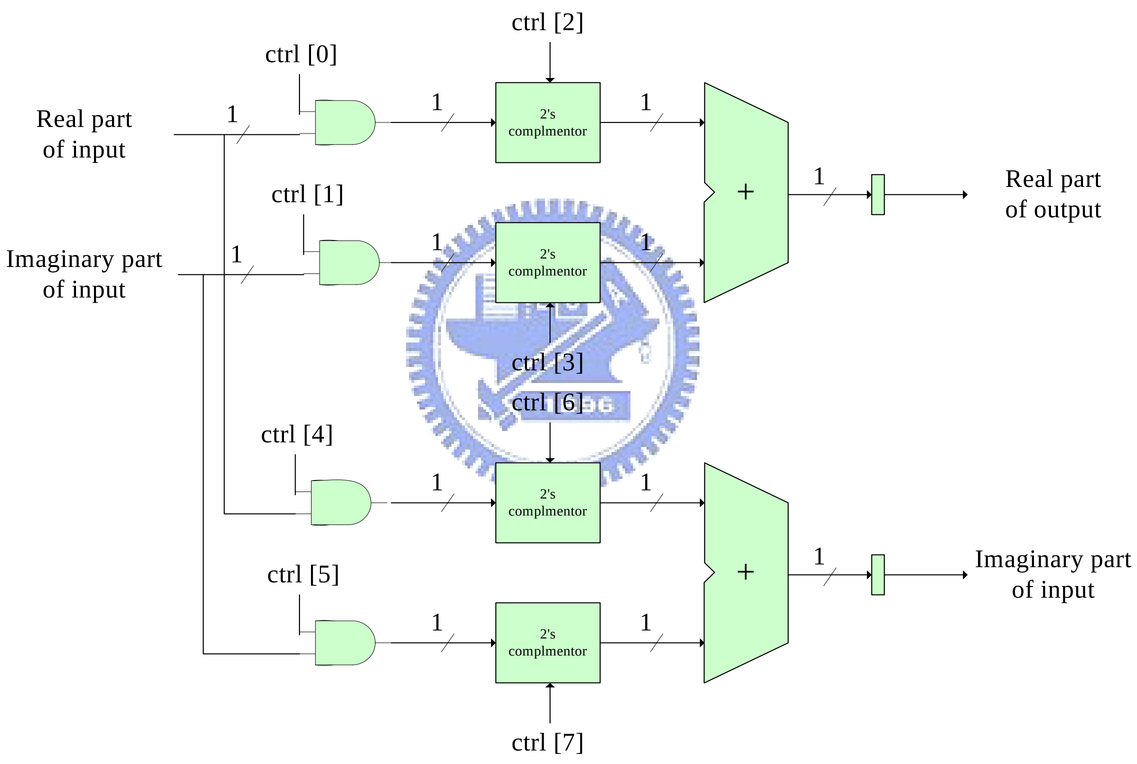


Fig. 3.5 CORDIC PE block diagram

Twiddle factor	Control signal ctrl[0..7]
W_8^0	0 0 1 0 0 0 0 1
W_8^1	0 1 1 1 0 0 1 1
W_8^2	0 1 0 1 0 0 1 0
W_8^3	1 1 1 1 0 1 1 1
W_8^4	1 0 1 0 0 1 0 1
W_8^5	1 0 1 1 1 1 1 1
W_8^6	0 0 0 1 1 0 1 0
W_8^7	0 0 1 1 1 0 1 1

Table 3.2 CORDIC PE control signal mapping

After rotation, the outputs from the CORDIC stage, which are bit-serial styled, are transmitted to the address line of the LUT (LookUp Table) in the DA stage. There are two sets of LUT and accumulator that handle the real part and imaginary part of the processing data individually. Basically, the lookup table sums all inputs up, and scales $1/\sqrt{2}$ if this input channel needs to:

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 & W_8^0 \\ W_8^0 & W_8^1 & W_8^2 & W_8^3 & W_8^4 & W_8^5 & W_8^6 & W_8^7 \\ W_8^0 & W_8^2 & W_8^4 & W_8^6 & W_8^0 & W_8^2 & W_8^4 & W_8^6 \\ W_8^0 & W_8^3 & W_8^6 & W_8^1 & W_8^4 & W_8^7 & W_8^2 & W_8^5 \\ W_8^0 & W_8^4 & W_8^0 & W_8^4 & W_8^0 & W_8^4 & W_8^0 & W_8^4 \\ W_8^0 & W_8^5 & W_8^2 & W_8^7 & W_8^4 & W_8^1 & W_8^6 & W_8^3 \\ W_8^0 & W_8^6 & W_8^4 & W_8^2 & W_8^0 & W_8^6 & W_8^4 & W_8^2 \\ W_8^0 & W_8^7 & W_8^6 & W_8^5 & W_8^4 & W_8^3 & W_8^2 & W_8^1 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \tag{18}$$

where the shaded coefficients contain a 45-degree rotation and need to be scaled.

The address bus width of the LUT is 11 bits, including 8-bit data input and 3-bit iteration index term that loops from 0 to 7 while computing 8-point DFT, and a

2K-word ROM is required. It is large number that is hard to implement and need to be modified. Actually, the sum operation can be calculated with an ones counter which simply counts the number of the logic ‘1’ from the 8-bit input so that the address line can be decreased to 4 bits. Besides, as (14) shown, only the odd channel is possibly to be scaled by $1/\sqrt{2}$. As shown in Table 3.3, according to the combination of each CORDIC channel output, there are 25 possible values. This is a small-size ROM, thus it is realized with synthesized logic rather than real ROM module. The modified LUT architecture is shown in Fig. 3.6, if the current iteration contains 45-degree rotations, the data passes 1’s counter, remapping table, and LUT, otherwise it passes 1’s counter only.

# of $1/\sqrt{2}$ / # of 1	0	1	2	3	4
0	0	0.707107	1.414214	2.121320	2.828427
1	1	1.707107	2.414214	3.121320	3.828427
2	2	2.707107	3.414214	4.121320	4.828427
3	3	3.707107	4.414214	5.121320	5.828427
4	4	4.707107	5.414214	6.121320	6.828427

Table 3.3 The 25 words used in LUT

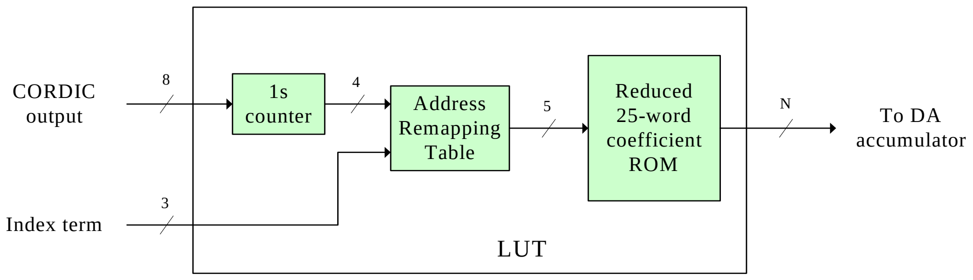


Fig. 3.6 Block diagram inside the modified LUT

Assume the input data width is N bits. Because an 8-point FFT is an 8-operand addition, total N+3 bits are required at the output. In the accumulator, a register of at least N+4 bits is required. The N+3 bits in MSB are necessary to hold the result, and the 1 bit in LSB is left for the accuracy while doing right-shift operations. One can increase the number of bit is LSB to gain more accuracy. In the test chip described in Chapter 5, a total 23 bits is utilized for 16-bit inputs. The latency of the proposed 8-point FFT structure depends on its data width because of the shift-accumulate operation in the DA stage. Total N+4 clock cycles are required to complete an 8-point FFT calculation. For a 16-bit FFT, the latency is 20 cycles, and it is a large number.

3.3 Error Analysis

In the case of FFT hardware implementation, the finite bitwdith must be considered because of the fixed-point computation. Many statistical error analysis papers on FFT implementations are proposed [27-29]. However, the CORDIC-DA structure proposed in this thesis is not a traditional FFT implementation, and an extended error analysis has to be done to choose a suitable bitwdith for the datapath. Assume the input sequence of FFT x(n) is a sequence of finite-valued and white complex numbers. The variance of x(n) can be expressed as

$$\sigma_x^2 = \frac{1}{N}\sum_{n=0}^{N-1}(x[n]-\mu_x)^2 = \frac{1}{N}\sum_{n=0}^{N-1}(x[n])^2 \qquad (19)$$

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

$$SQNR = \frac{\sigma_x^2}{\sigma_q^2} \qquad (20)$$

where σ_x^2 is the variance of output and σ_q^2 is the variance of the quantization error.

For an N-point FFT processor with input of which real and imaginary parts are uniformly distributed in $(-\frac{1}{\sqrt{2}}N, \frac{1}{\sqrt{2}}N)$, the variance [28] of the output is

$$\sigma_X^2 = \frac{1}{3N} \tag{21}$$

From (20) and (21), the SQNR [29] of the conventional FFT implementation can be carried out:

$$SQNR_{FFT} = \frac{2^{2B}}{5N - 4m - 3} \tag{22}$$

where B is the bitwidth of the input sequence and m=log$_2$N.

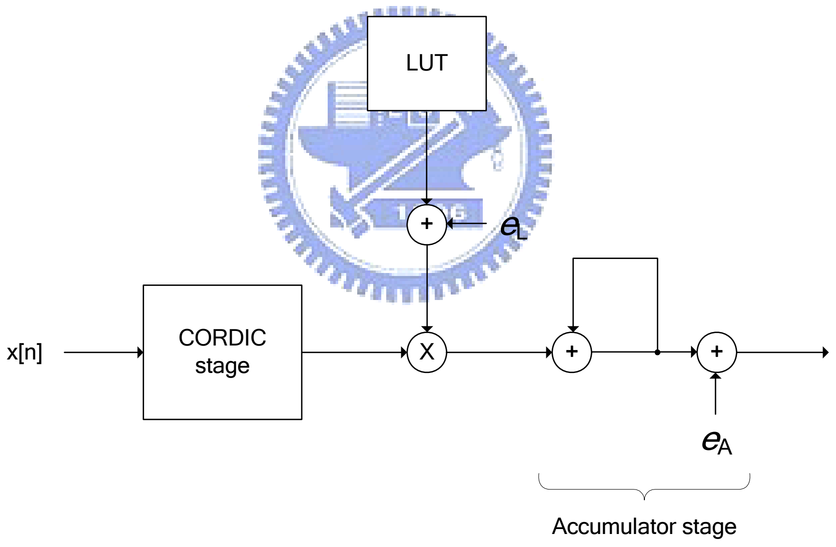


Fig. 3.7 Error model of CORDIC-DA

Fig. 3.7 shows the error model of CORDIC-DA. In the CORDIC stage, one more guard bit is added and can proof noise-free. The coefficients stored in the LUT are precomputed and rounded into M-bit that a roundoff noise e_L is added. Because of the mechanism of DA, total B times shift-add operation will be done in the accumulator for a B-bit input sequence. The variance of the roundoff error until the accumulator is

$$\begin{aligned}\sigma_{LUT}^2 &= \frac{2^{-2M}}{3} \\ \sigma_1^2 &= (\tfrac{1}{2})^{B-1}\sigma_{LUT}^2 + (\tfrac{1}{2})^{B-2}\sigma_{LUT}^2 + \ldots + (\tfrac{1}{2})^0\sigma_{LUT}^2 \\ &= \sigma_{LUT}^2 \sum_{k=0}^{B-1} \frac{1}{2^k} \\ &= 2(1-2^{-B})\sigma_{LUT}^2 \\ &\approx 2\sigma_{LUT}^2 = \tfrac{2}{3}2^{-2M}\end{aligned} \tag{23}$$

At the end of the CORDIC-DA, the output will be rounded into B-bit. The variance of the rounding error is

$$\sigma_2^2 = \frac{2^{-2B}}{3} \tag{24}$$

From (23) and (24), the SQNR of the CORDIC-DA can be expressed as

$$\begin{aligned}SQNR_{CORDIC-DA} &= \frac{\sigma_x^2}{\sigma_1^2 + \sigma_2^2} \\ &= \frac{1}{N} \cdot \frac{1}{2 \cdot 2^{-2M} + 2^{-2B}}\end{aligned} \tag{25}$$

To increase SQNR, we would like to make M and B as large as possible but are limited by the implementation resource. A better decision on M and B is to make sure

$$\begin{aligned}2 \cdot 2^{-2M} &= 2^{-2B} \\ M &= B + 0.5\end{aligned} \tag{26}$$

From (26), M=B+1 is a better solution for trade-off between resource usage and SQNR performance. Replace M=B+1 and N=8 in (22) and (26), the SQNR expressions can be rewritten as

$$SQNR_{FFT} = \frac{2^{2B}}{25} \tag{27}$$

$$SQNR_{CORDIC-DA} = \frac{2^{2B}}{12} \tag{28}$$

From (27) and (28), one can discover that the SQNR performance of the CORDIC-DA structure is better than the conventional implementation in the same bitwidth. To match the SQNR of the two structure, we can lower the B and M value of the CORDIC-DA structure that will reduce the hardware cost. Concretely, to lower the M value is to decrease the bitwidth of the LUT, and to lower the B value is to decrease the registers of the accumulator and the remaining bits after roundoff. Table 3.4 lists the statistical SQNR analysis of the two structures. Although the CORDIC-DA structure performs better than the conventional FFT, there is no just match value on SQNR versus bitwidth. In the test chip, B=16 and M=16 is chosen.

B	8	9	10	11	12	13	14	15	16
FFT	68.37	80.41	92.45	104.49	116.54	128.58	140.62	152.66	164.70
CORDIC-DA	74.75	86.79	98.83	110.87	122.91	134.95	146.99	159.03	171.08

Table 3.4 Statistical SQNR analysis of conventional FFT and CORDIC-DA

3.4 Implementation and Simulation Results

Since the 8-point CORDIC-DA FFT algorithm is proposed, an evaluation model is developed to verify the algorithm. The three FFT models: CORDIC-DA FFT, R2SDF FFT, and fully parallel FFT, are structured with 16-bit data width, and the clock constraints are set to very high frequency so that the limit of these implementations will be carried out. Table 3.5 lists the simulation results of these FFT implementations.

The parallel FFT is a straight implement of 8-point DIT FFT, as shown in Fig. 2.5. There is no pipeline register used thus it requires only 1 clock cycle to complete an 8-point FFT operation. The R2SDF is implemented with Fig. 2.6. Because of the feedback registers, it requires 7 clock cycles to carry out the first result. The CORDIC-DA structure can run at a very fast speed because of the bit-serial arithmetic. However, the latency is affected by the data width, and it requires more clock cycles to carry out the first result.

According to the simulation results, the parallel FFT requires most area costs but shortest latency. On contrast, the CORDIC-DA structure utilized least area costs, but the latency is the longest of the three FFT implementations. It may possibly because of the long data width that the DA stage requires more clock cycles to accumulate. Although it can achieve very high frequency, it is still not fast enough. The critical path of CORDIC-DA is found on the LUT which is hard to be pipelined. The power consumption of the proposed CORDIC-DA structure is higher than the other two implementations. In order to achieve very high clock rate, the logic synthesizer inserted as much clock buffers as possible so that more power is consumed with the high toggle rate of the clock propagation.

The result does not go as we expected so that an idea of merging the twiddle factor rotators into the branch FFT comes out. In this way, the latency produced by the twiddle factor rotator may be eliminated with little delay in our CORDIC-DA structure.

However, if a serial I/O interface is adopted, the CORDIC-DA will possess the shortest latency. The shift-out bit in the accumulator can be passed to the serial output so that the first bit can be carried out in 2 clock cycle. The shortest clock period of CORDIC-DA structure is 1.65 ns, and the shortest latency is about 3.30 ns.

Implementation		Parallel FFT	R2SDF	CORDIC-DA
Max. clock rate		6.47 ns 154.6 MHz	5.47 ns 182.8 MHz	1.65 ns 606.0 MHz
Gate count		23946	10740	9982
Latency	Clock cycle	1	7	20
	Timing	6.47 ns	38.3 ns	33 ns
Throughput Rate (clock cycle)		0.125	1	17
Static power analysis		66 mW	42 mW	24 mW
Simulation-based power analysis		3.77 mW @ 20 MHz	6.65 mW @ 20 MHz	35.6 mW @ 400 MHz

Table 3.5 Simulation results of short-length FFT

Chapter 4　Long-Length FFT

4.1　Introduction

There are two key components structuring recursive architecture long-length FFT: the short-length FFT (also known as the branch FFT) and the twiddle factor rotator. In the previous chapter, the objective of the proposed CORDIC-DA structure is to decrease the latency in the branch FFT. A modification to integrate the twiddle factor into the DA LUT will be described in this chapter.

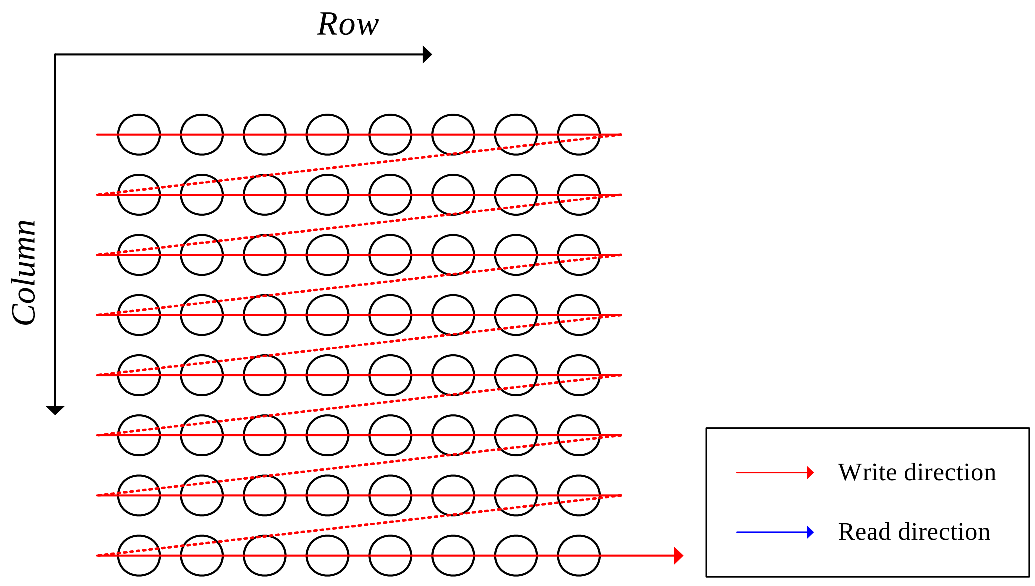
The memory architecture of the recursive FFT is another issue. In a recursive long-length FFT architecture, the frequent access to intermediate memory requires a high-speed memory device and efficient access scheme. A matrix buffer memory access scheme and how it is adopted in the memory devices provided by TSMC 0.18 μm CMOS process will be described in this chapter, too.

4.2 Memory Access

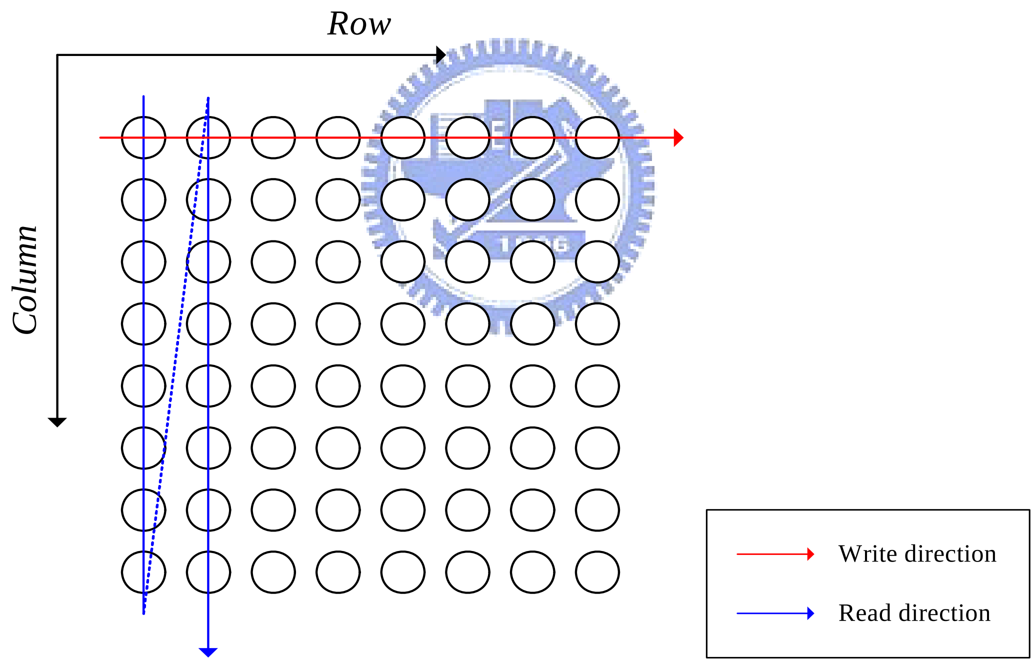
Consider a radix-8 FFT. One may replace M=8 in (4) and get

$$\begin{aligned} &X(8k_1+k_2) \\ &= \sum_{n_1=0}^{\frac{N}{8}-1}\sum_{n_2=0}^{7} x(n_1+\tfrac{N}{8}n_2)W_N^{(n_1+\frac{N}{8}n_2)(8k_1+k_2)} \\ &= \underbrace{\sum_{n_1=0}^{\frac{N}{8}-1}\left\{\underbrace{\sum_{n_2=0}^{7} x(n_1+\tfrac{N}{8}n_2)W_8^{n_2k_2}}_{8-\text{point DFT}}\underbrace{W_N^{n_1k_2}}_{\text{twiddle factor}}\right\}W_{\frac{N}{8}}^{n_1k_1}}_{\frac{N}{8}-\text{point DFT}} \end{aligned} \tag{29}$$

Notice that the access order of the input and output between two iterations are the transpose of each other. For example, a 64-point FFT requires two iterations, and equipped with an 8 × 8 memory. At the first iteration, the output is stored to memory row by row, as shown in Fig. 4.1(a). At the second iteration, the input is loaded from memory column by column, as shown in Fig. 4.1(b). However, if the computation of the first column is completed, and the result is written back to buffer row by row, the operands of later 8-point FFT will be overwritten. To solve this problem, an addition buffer is required. But it increases area cost very much, and is not realizable. A matrix buffer scheme is proposed in [12] and solves this problem. The memory device is specialized that the access to columns and rows will be swapped for each iteration so that no operands will be overwritten.



(a)



(b)

Fig. 4.1 Memory access collision between two stages

The matrix buffer is synthesized with cell-based process. In general, a synthesized memory device usually costs more area and power than a full-custom one. The TSMC 0.18μm CMOS process provides several memory generators, such as single-port SRAM, dual-port SRAM, single-port register file, dual-port register file, etc. These hard-macro modules are good at area, timing, and power consumption compared to a synthesized one. We found that dual-port register file module is suitable for this access scheme, and a N-word register buffer is adopted in the recursive architecture.

4.3 Latency in Twiddle Factor Rotator

There are many methods to implement twiddle factor rotators, such as complex multipliers, CORDIC, etc. A complex multiplier produces delay time of 1 real multiplier and 1 real adder. Delay time produced by a CORDIC rotator depends on the resolution of the elementary rotation angle derived from (6). One single CORDIC iteration stage produces delay of 1 real adder. For a N-stage CORDIC, total N real adder and one constant scalar delay are produced.

The idea of the proposed new structure is to decrease the latency of the branch FFT iteration as possible. In our view, merging the twiddle factor rotation into the branch FFT is an alternative approach to lower the latency. Because of the LUT in DA, the twiddle factor rotation can also be precomputed and stored in the ROM with reasonable increase on ROM size and little additional logic.

4.3.1 Complex Multiplier Phase Rotator

A straight implementation of a phase rotator is complex multipliers as shown in Fig. 4.2. The critical path passed through a real multiplier and a real adder. The real part and imaginary part of the twiddle factors are constants, and can be stored in a ROM, as shown in Fig. 4.3. In practice, only a range of $\pi/4$ will be realized, as shown in Fig. 4.4. The twiddle factors located at area II, III, IV, V, VI, VII, and VIII can be obtained from the ones located at area I with simply exchanging signs or/and exchanging real/imaginary part of operands. Table 4.1 shows the transform function. For example, for a 64-point FFT, total 9 coefficients of W_{64}^{0}~W_{64}^{8} need to be stored in ROM. Other angles ranging from W_{64}^{9}~W_{64}^{63} can be obtained from the 9 angles.

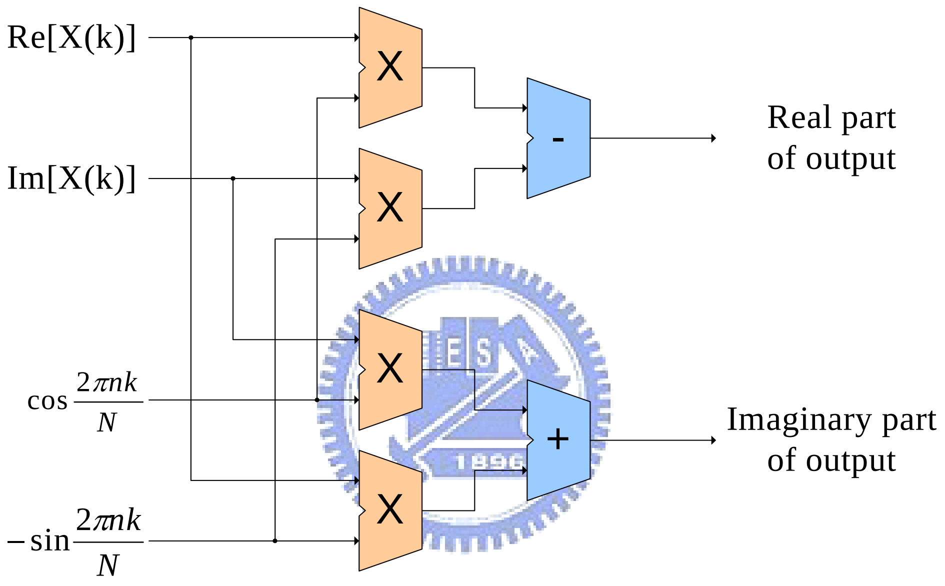


Fig. 4.2 Signal flow chart of a complex multiplier

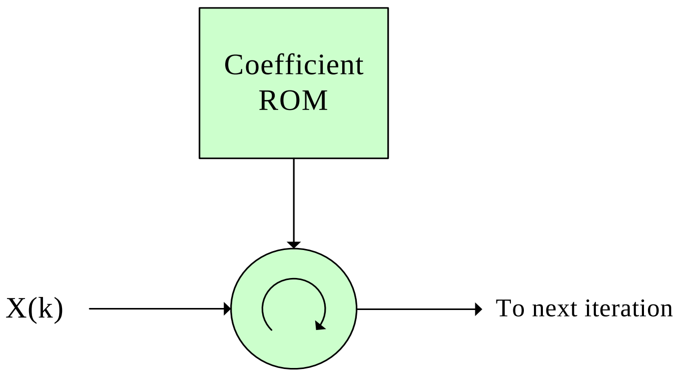


Fig. 4.3 Configuration of coefficient ROM and phase rotator

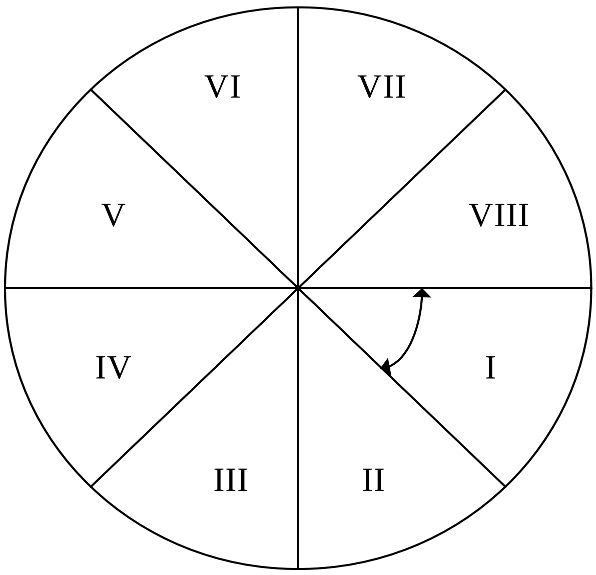


Fig. 4.4　The range of stored angles in practice

Area	Transform Matrix	Index complement	Area	Transform Matrix	Index complement
I	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	No	V	$\begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$	No
II	$\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$	Yes	VI	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	Yes
III	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$	No	VII	$\begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$	No
IV	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	Yes	VIII	$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	Yes

Table 4.1　The Transform Matrix within range of $\pi/8$

4.3.2 ROM Multiplier Phase Rotator

A Basic ROM multiplier is utilized with a ROM which simply stores the products of the multiplication. For example, a ROM multiplier for N-bit multiplicand and M-bit multiplier requires a ROM with size of $2^{(N+M)} \times (N+M)$. However, a ROM multiplier realization becomes impractical while N or M is large. An alternative approach is to partition the data width into lower number of bits, and uses a shift-accumulator to accumulate the products. As shown in Fig. 4.5, the N-bit multiplicand is partitioned by every 4 bits, and right-shifts one block per cycle. The ROM stores the (4+M)-bit product of the 4-bit multiplicand and the M-bit multiplier. Then, the previous result stored in the accumulator is right-shifted by 4-bit, and is summed up with the product provided by ROM. The multiplication requires N/4 cycles to complete. As the lower bits the data is going to be partitioned, the more clock cycles to complete a multiplication are required.

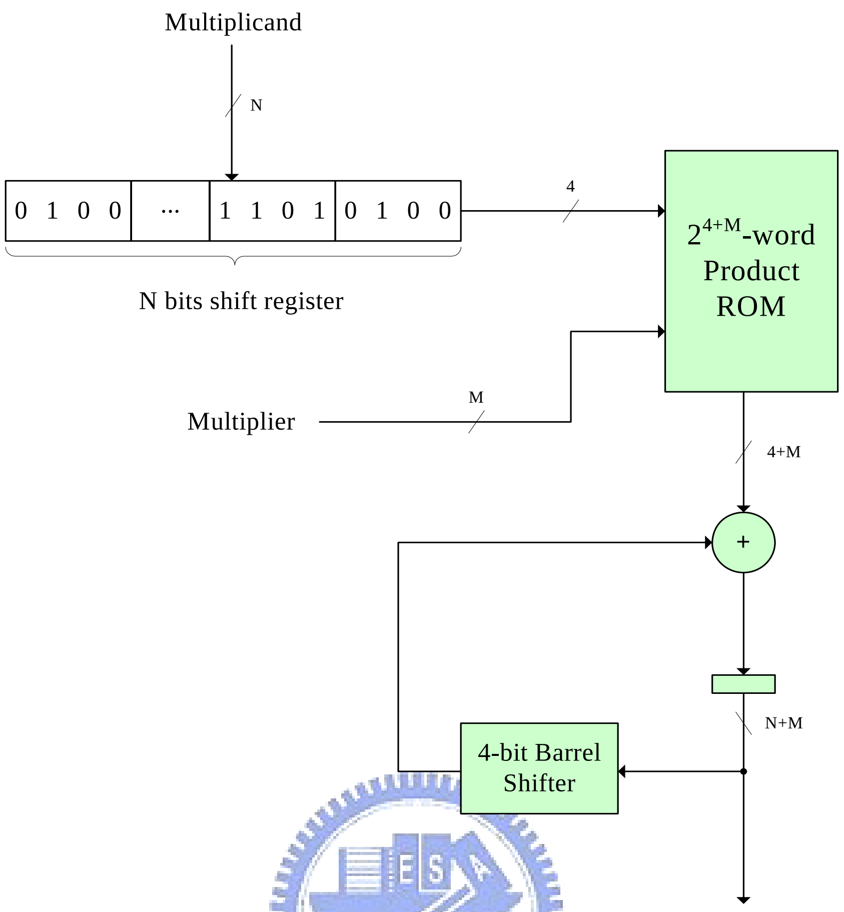


Fig. 4.5 4BAAT ROM multiplier mechanism

In a basic ROM multiplier phase rotator, multipliers shown in Fig. 4.2 are simply replaced with ROM multipliers. Because the twiddle factor ROM stores constant coefficients, it can be merged into these ROM multipliers. For example, a ROM multiplier that multiplies a K-bit input with L possible fixed-point coefficients of twiddle factors $W_N^{1\sim L}$ is shown in Fig. 4.6. The input passes through a series of shift registers which provides 4 bit per cycle to the ROMs. ROM A stores the product related to the real parts of $W_N^{1\sim L}$, and ROM B stores the product related to the real parts of $W_N^{1\sim L}$. The rotation will be complete after K/4 cycles.

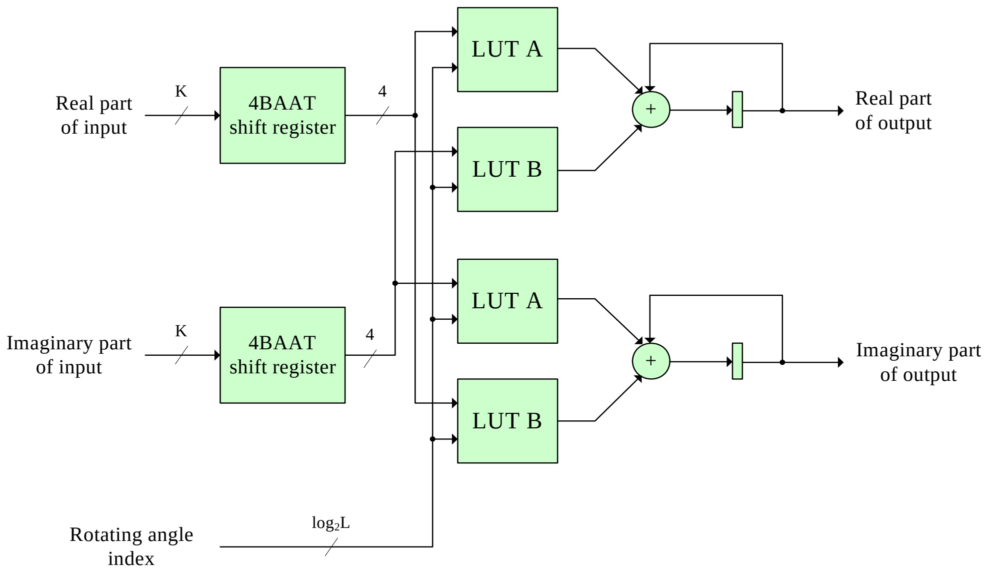


Fig. 4.6 4BAAT phase rotator with constant rotating angle

4.3.3 CORDIC-DA Structure with Phase Rotator

As the configuration described in Chapter 3, the ROM stores only few patterns because there are only two possible coefficients: 1 and $1/\sqrt{2}$. DA is intrinsically a ROM-based multiplier. Considering a CORDIC-DA 8-point FFT processor following a twiddle factor rotator implemented with complex multipliers, the LUT can take advantages of the property of ROM-multiplier that combines the original function with the twiddle factor rotations. Moreover, the transform described in Fig. 4.4 can also be merged with the CORDIC with no extra computation. For example, a radix-8 branch FFT for 64-point FFT derived from (1) can be expressed as

$$T(n_1,k_2)=\sum_{n_2=0}^{7}x(n_1+8n_2)W_8^{n_2k_2} \tag{30a}$$

$$\begin{aligned}&X(8k_1+k_2)\\&=\sum_{n_1=0}^{7}\left\{T(n_1,k_2)W_{64}^{n_1k_2}\right\}W_8^{n_1k_1}\end{aligned} \tag{30b}$$

where $T(n_1,k_2)$ is the intermediate data between stages. Let

$$\hat{n} = n_1 k_2 \bmod 8$$
$$\hat{k} = \left\lfloor \frac{n_1 k_2}{8} \right\rfloor \tag{30c}$$

(16b) can be rewritten as

$$\begin{aligned} & X(8k_1 + k_2) \\ & = \sum_{n_1=0}^{7} \left\{ T(n_1, k_2)\, W_{64}^{\hat{n}} \right\} W_8^{n_1 k_1 + \hat{k}} \end{aligned} \tag{30d}$$

which extracts the W_8 elements from $W_{64}^{n_1 k_2}$. Thus, the W_8 rotation, which has been integrated with the CORDIC stage, can be skipped in LUT. Fig. 4.7 shows the block diagram of the refined CORDIC-DA FFT structure.

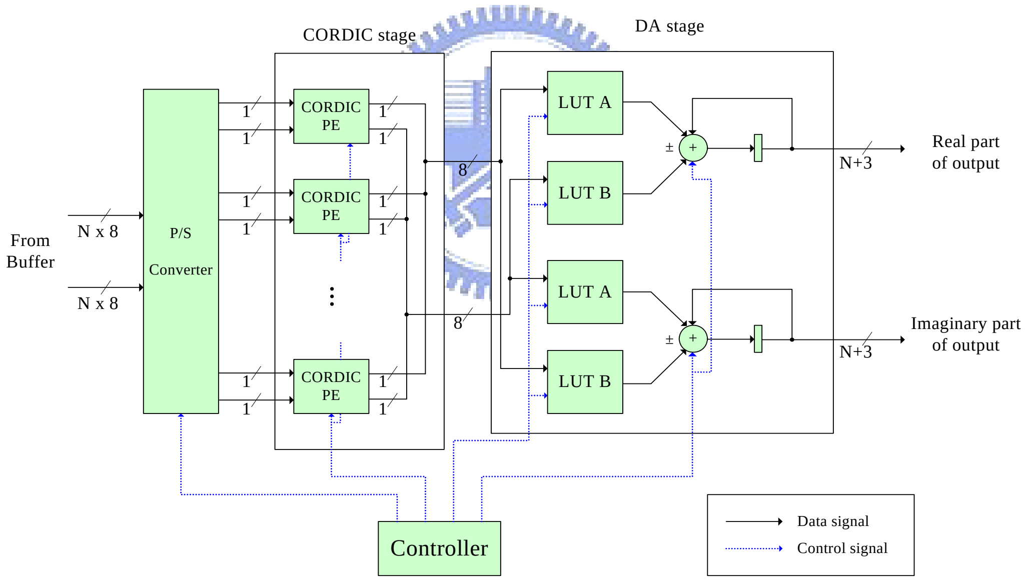


Fig. 4.7 Block diagram of the refined CORDIC-DA FFT

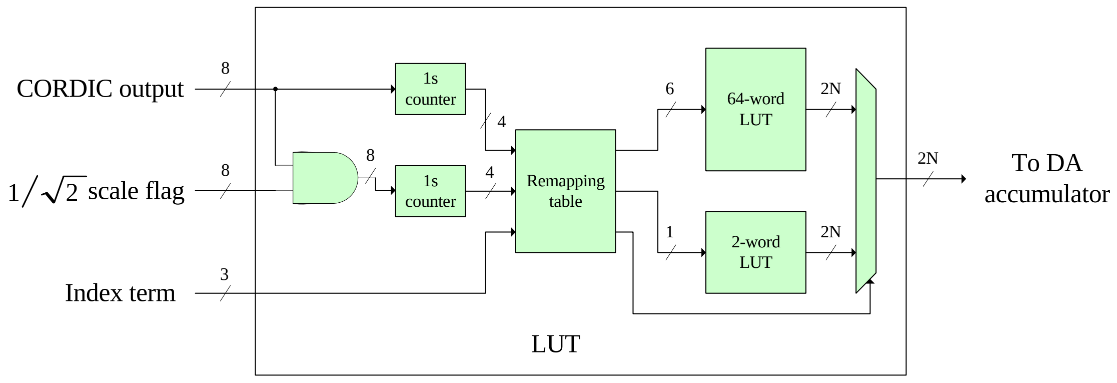


Fig. 4.8 Block diagram of integrated LUT for 64-point FFT

The complex multipliers of the twiddle factor rotation are replaced with four LUTs. Each LUT maps to the real multiplier shown in Fig. 4.2. The LUT A stores the product of input and $\cos\frac{2\pi nk}{N}$, and The LUT B stores the product of input and $\sin\frac{2\pi nk}{N}$. Although in Fig.4.7 LUT A and B are separated, the content of the two LUT are actually integrated in a single ROM in realization so that the bitwidth of the output will be 2N bits. Fig. 4.8 illustrates the block diagram of the integrated LUT for 64-point FFT. There are two control signals: $1/\sqrt{2}$ scaling flags and new ‘index term’ signal. Because of the W_8 extraction described in (16d), every CORDIC PE has had the ability to handle a 45-degree-based rotation, thus a new $1/\sqrt{2}$ scale flag is designed to instruct the LUT to scale the designated input channel individually. The ‘index term’ control signal is different from the old one. In the refined structure, the ‘index term’ signal is directed from the $\hat{n}$ described in (30c) which is calculated in the controller. Considering the output of the two ones counter and index term, the content of the LUT can be expressed as

$$x = \begin{cases} (a-b) + bW_N^{\hat{n}} & \text{if } a \ge b \\ 0 & \text{otherwise} \end{cases} \tag{31}$$

where x is the output of LUT, a is the result from the ones counter of the CORDIC output, and b is the result from the ones counter of logic AND operation between the CORDIC output and $1/\sqrt{2}$ scale flag. *a* and *b* range from 0 to 8, and *a*<*b* will never occur so that a value of zero is filled. In the case of 64-point FFT, $\hat{n}$ ranges from 0 to 7, and total 36×8 words are required to store all LUT contents. However, there are two problems to implement the LUT: this number is not power of 2, and the remapping logic shown in Fig. 4.8 will be complex that produces longer latency. During the synthesis process, the most critical path is found at the LUT stage, thus the latency in this stage needs to be short as possible. A trade-off solution is adopted that splits the LUT into 2 sub-LUTs, a 64-word one and a 2-word one. The 64-word LUT stores the content of *a*=1~7 and *b*=1~7, as expressed in (31). The address is simply concatenated from the 3-bit LSB from the result of ones counters. *a*=0,*b*=0, and *a*=8, *b*=8, are seen as special cases, and are handled by the 2-word LUT. In these special cases, the final output of the LUT stage is switched to the 2-word LUT, or switched to the 64-word LUT otherwise.

The accumulator in the DA stage is modified into a 3-operand adder. It is implemented with a CSA (Carry-Save Adder) and adds delay of a 3-input XOR gate to the critical path. With the integration, little delay on the critical path is attached but the twiddle factor rotators are all saved. This refined CORDIC-DA branch FFT may possibly gains more benefits while utilized in a recursive architecture long-length FFT.

4.4 Implementation and Simulation Results

Although the synthesized 8-point CORDIC-DA FFT structure can run at a very high frequency, the clock is hard to propagate. An in-chip PLL (Phase-Locked Loop) circuit may solve this problem. A PLL circuit requires a low frequency external clock source and can generate a high frequency internal clock source. However, we have no such PLL models. Another approach that increases the number of datapath in CORDIC-DA is adopted. The CORDIC-DA FFT computes one result of 8-point FFT at a time, thus increasing the datapath will help produce more results at a time. As shown in Fig. 4.9, a complete datapath, which is called a ‘channel’, includes a CORDIC stage and a DA stage. The P/S converter can be shared by channels so that it is not duplicated.

The implementations follow the specification described in the previous chapter. A set of CORDIC-DA structures with 1, 2, 4, 8 pipes are developed, and they are already capable of twiddle factor rotation. The other two implementations, R2SDF and parallel FFT, are developed with a complex multiplier twiddle factor rotator shown in Fig. 4.2. According to the synthesis report listed in Table 4.2, we discovered that CORDIC-DA FFT could achieve better power efficiency while more pipes are utilized. With higher working frequency, more power consumes on the clock buffers, and driving ability of logic cells is strengthened so that more power is required. Compared to the basic 8-point CORDIC-DA FFT without merging twiddle factor rotators, the critical path increases to about 2.4 ns because of the growing LUT size. However, they still spent more latency than the other two conventional FFT implementations. Unexpectedly, the optimization effort on the complex multiplier done by the logic synthesizer is very high. As the report shown, the 16×16 multiplier produced very low delay that is less than 5.4 ns.

Notice the two measurements of power density listed in Table 4.2. The “power density” is how much power consumed per K gate counts at the same throughput rate. A higher value somehow means a better area efficiency. As the result shown, the 1-channel version of CORDIC-DA possesses the highest area efficiency, and verifies the previous inference. The “normalized power” is how much power consumed per MHz working frequency, not the same throughput rate. A higher value somehow means a higher toggle rate, or utilization on physical circuits. According to the reported values, this value is proportion to the number of channels. This value of the 1-channel CORDIC-DA structure is low, but the total power consumption is higher than the CORDIC-DA configured with more channels. The high power consumption is possibly is possibly because of the high clock rate that more power is consumed by the clock propagation.

If a full serial environment is considered that the I/O interfaces and memory buffers are configured in bit-serial arithmetic, the latency of the CORDIC-DA will still be the shortest of these FFT implementations. As described in the previous chapter, only 2 clock cycles are required to carry out the first bit.

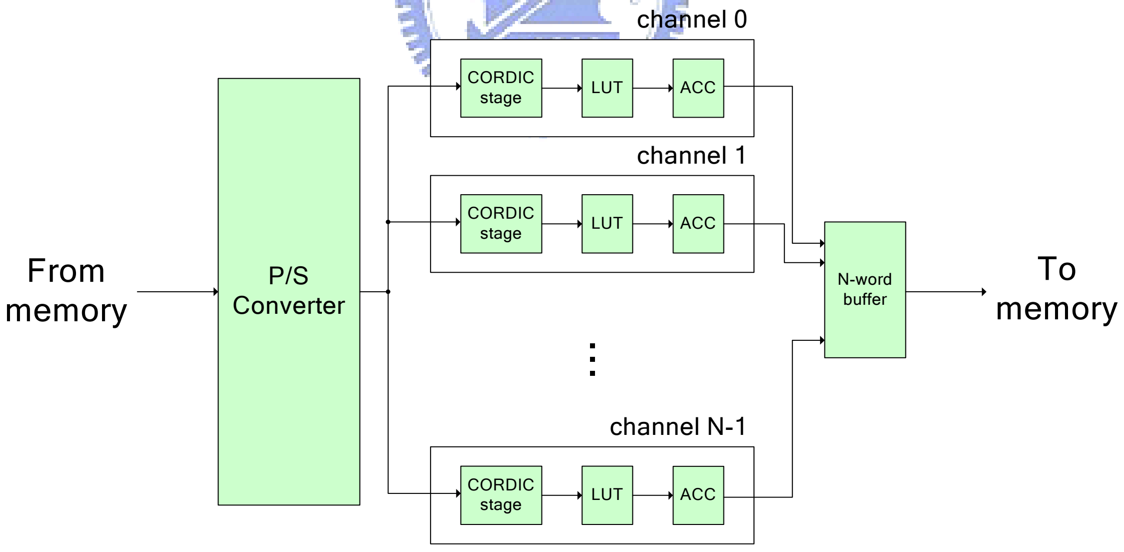


Fig. 4.9 Duplicate datapaths in CORDIC-DA stage

Structure		CORDIC-DA				R2SDF	Parallel FFT
		1 channel	2 channels	4 channels	8 channels		
Max. clock rate		2.34 ns 427.4 MHz	2.36 ns 423.7 MHz	2.41 ns 414.9 MHz	2.50 ns 400.0 MHz	5.47 ns 182.8 MHz	6.47 ns 154.6 MHz
Latency	Clock cycle	20	20	20	20	8	2
	Timing	46.8 ns	47.2 ns	48.2 ns	50 ns	43.8 ns	12.9 ns
Gate count		14,792	25,142	44,922	81,852	25,666	36,394
Static power analysis		52.9 mW @ 400 MHz	52.2 mW @ 200 MHz	55.8 mW @ 100 MHz	49.1 mW @ 50 MHz	7.2 mW @ 20 MHz	4.84 mW @ 20 MHz
Simulation-based power analysis		59.1 mW @ 400 MHz	51.5 mW @ 200 MHz	45.8 mW @ 100 MHz	41.6 mW @ 50 MHz	18.1 mW @ 20 MHz	12.0 mW @ 20 MHz
Power density (mW / K gates)		4.00	2.05	1.02	0.51	0.71	0.33
Normalized power (mW / MHz)		0.15	0.26	0.46	0.83	0.905	0.6

Table 4.2 Summary of branch FFTs

Chapter 5 A Case Study: 802.11a Wireless LAN 64-Point FFT Processor

5.1 Design Environment

In the previous chapters, the proposed CORDIC-DA structure is described. An FFT test chip for long-length application is developed to verify the proposed FFT processor. This test chip follows the specification of 802.11a wireless LAN 64-point FFT processor. According to the results shown in the previous chapters, the parallel 8-point FFT implementation is better than the proposed structure at power and speed. Thus, another test chip is also developed with the parallel 8-point FFT implementation to verify the argument. The specified FFT processor is a 16-bit 64-point FFT processor, working at the sampling frequency of 20 MHz. The proposed architecture was modeled in VHDL and functionally verified using Mentor Graphics' Modelsim simulator.

After functional validation, the processors were synthesized for TSMC 0.18μm single-poly six-metal CMOS technology using Synopsys Design Compiler. After synthesis, floor planning, P&R, and layout were carried out using Cadence SOC

Encounter, as shown in Fig. 5.1 and Fig. 5.2. Finally, the post-simulation power analysis on the netlists exported from SOC Encounter is carried out using Synopsys PrimePower.

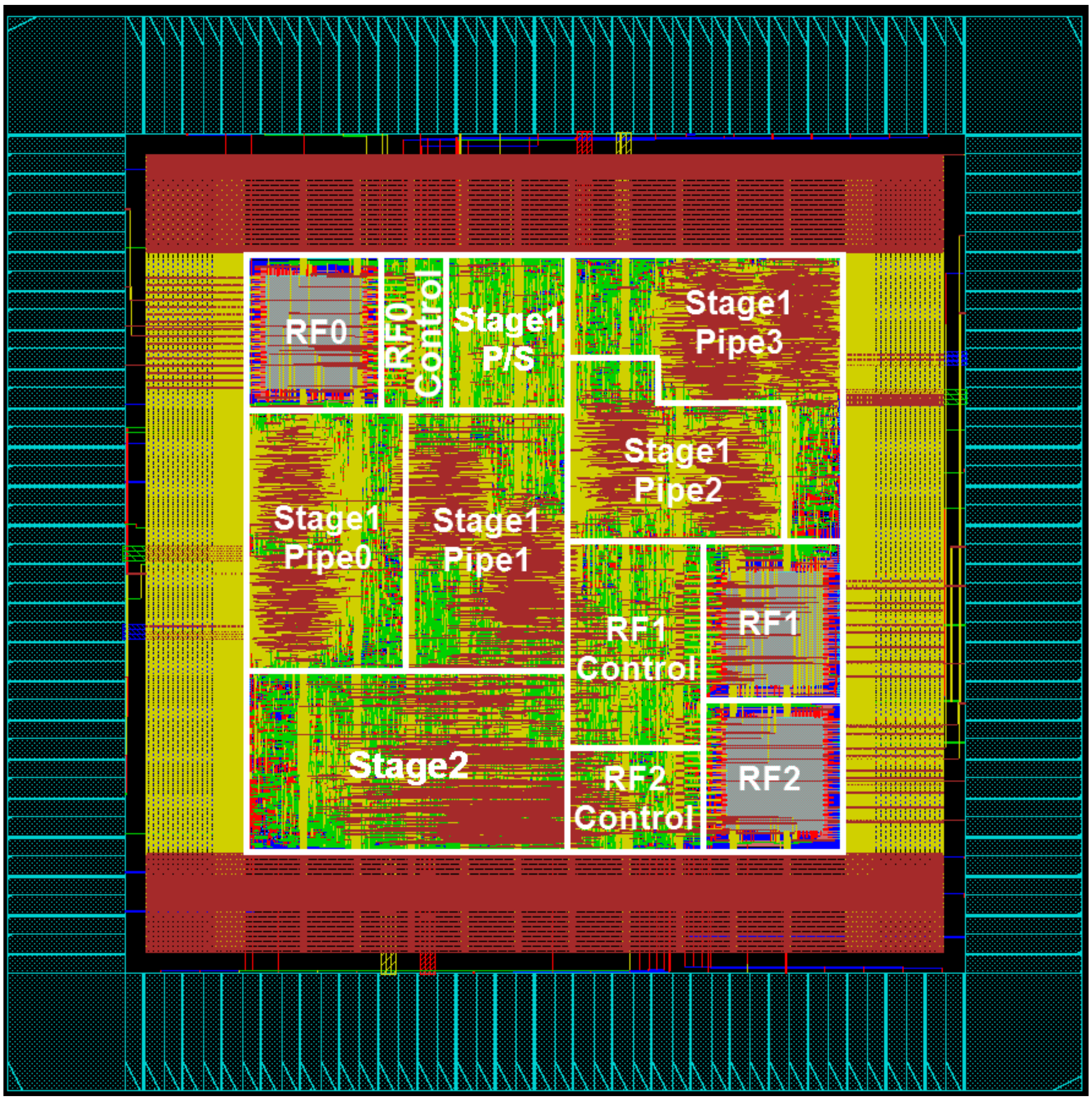


Fig. 5.1 Layout view of 64-point FFT with CORDIC-DA structure

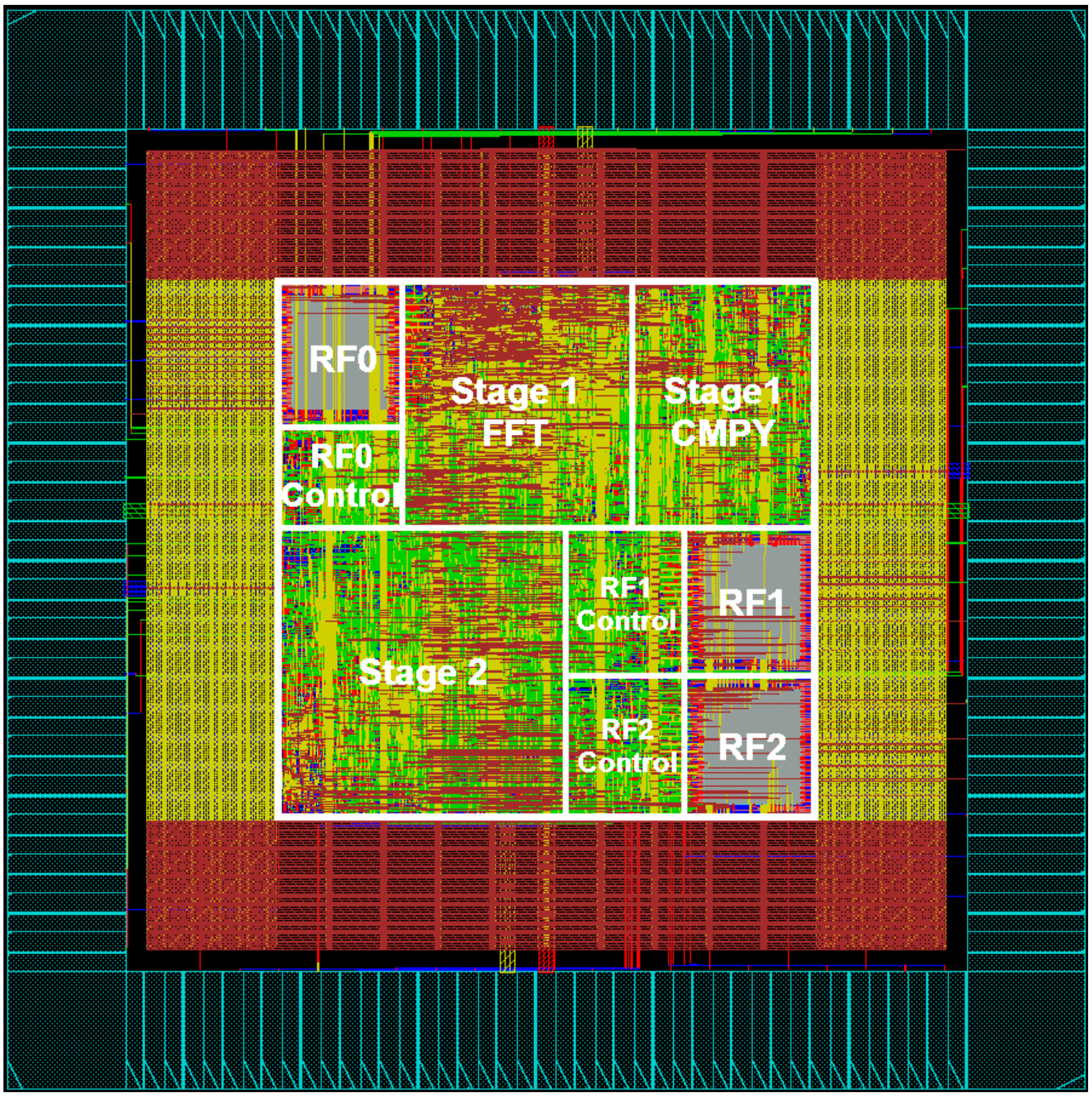


Fig. 5.2 Layout view of 64-point FFT with parallel FFT structure

5.1.1 Clock Issues

At first, the 1-channel version with dual-memory architecture is developed, as shown in Fig. 5.3. A 64-point FFT operation can be completed within 2 iterations. The 1-channel version requires 20 cycles to complete one result so that total $20 \times 64=1280$ cycles are required for each iteration. With extra cycles spent on memory access, a total number of over 2560 cycles is required to complete a 64-point FFT operation. An average of more than 40 cycles is required per output at 20 MHz. Thus, a very high frequency clock source of over 20×40=800 MHz is required. Considering the clock source generation, this implementation is impractical.

Notice that no twiddle factor rotation is needed during the second FFT iteration. Thus, the second 8-point FFT processing unit can be structured with no phase rotator. Considering the unbalanced computational requirement of the 2 branch FFTs, the 2 iterations can be structured individually. Separating them into 2 pipeline stages will help decreasing the area costs and lowering the clock rate by less than a half. Although the study is concentrated on recursive architectures, the high clock rate makes the chips hard to realize. The chips have to be implemented with frequency about 200MHz or even lower thus a flattened architecture is needed. A refined version with flattened pipeline architecture was built, as shown in Fig. 5.4. The second stage is structured with 8-point parallel DIT FFT.

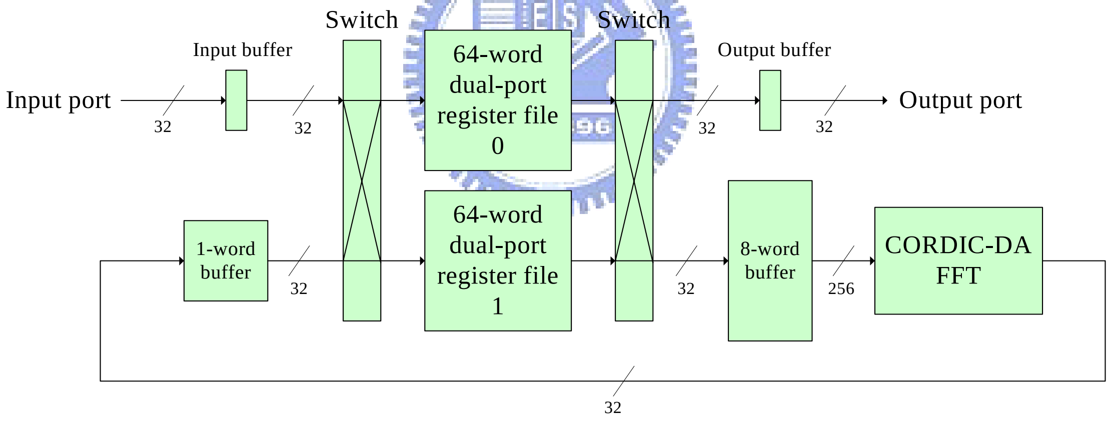


Fig. 5.3 Block diagram of 1-channel version CORDIC-DA FFT with recursive architecture

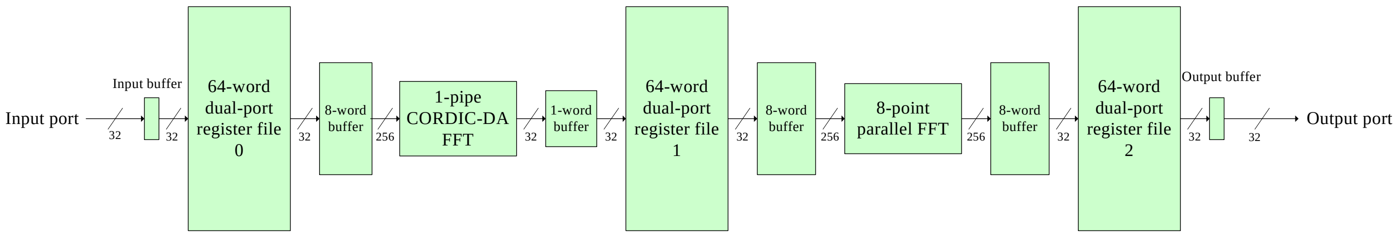


Fig. 5.4 Block diagram of CORDIC-DA with 2 pipeline stages

According to the bad performance result, we tried to develop a 2-channel version of CORDIC-DA FFT. As the result described in the previous chapter, the 2-channel version is better at power consumption and clock frequency, thus we developed a VHDL model which is parameterized on number of channels with the GENERIC statement. Available options are 1, 2, 4, 8 channels, and clock rates are 400, 200, 100, 50 MHz respectively. Considering power consumption and area costs, the 4-channel version was chosen to realize.

In these designs, two clock domains are adopted: the bit-serial domain and bit-parallel domain. The bit-serial domain, in which the circuits work at 1:1 frequency as the external clock source, contains the CORDIC-DA FFT and the P/S converter. The other components are of the bit-parallel domain, in which the circuits work at 20MHz, and the clock source is supplied by in-chip digital clock divider.

5.2 Verification

A functional test environment was built using Matlab which generates random test patterns for verification, as shown in Fig. 5.5. The testbench compares the simulation results to the golden patterns, which were also generated by Matlab, and run-time analyzes PSNR (Peak Signal-to-Noise Ratio) which is defined as:

$$MSE = \frac{1}{N}\sum_{n=0}^{N-1}\left(Result(n) - Golden(n)\right)^2$$

$$PSNR = 20\log_{10}\left(\frac{2^b}{MSE}\right) \tag{32}$$

where N is the number of the point of FFT and b is the bitwidth of data bus.

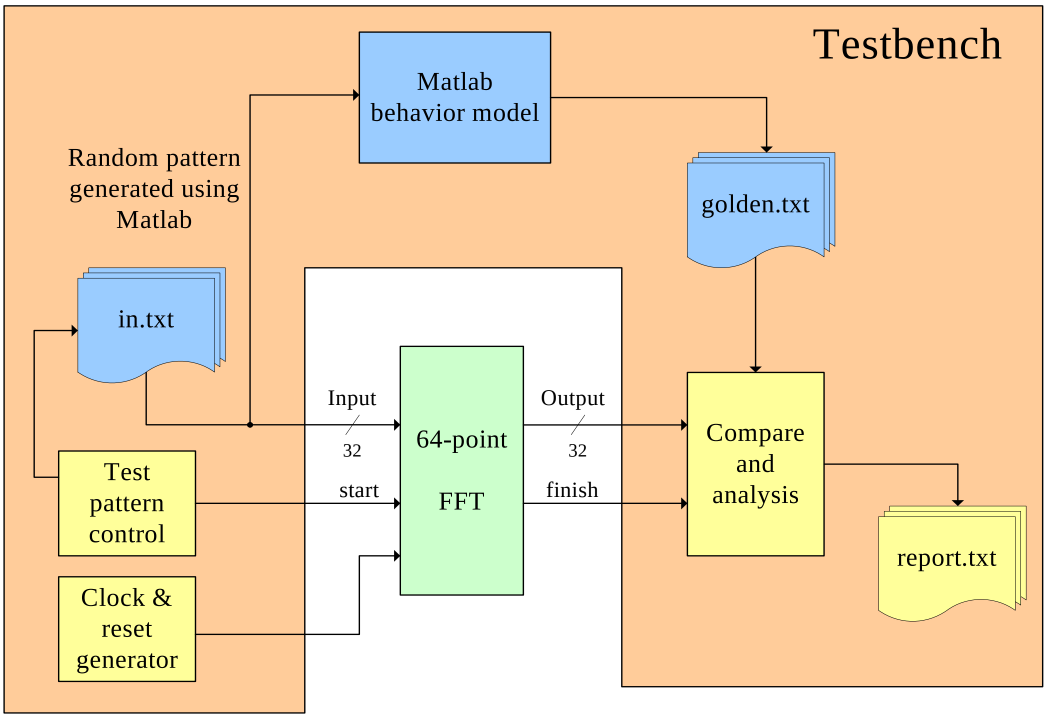


Fig. 5.5 Testbench for 64-point FFT

5.3 Test Strategy

We adopted full scan test for these chips. The test circuits were inserted during compilation using Synopsys DFT Compiler. The existing flip-flops inside the chips were replaced with scan flip-flops, and additional memory wrappers by the I/O port of dual-port register files were inserted. Test vectors are generated by Synopsys TetraMAX. The fault coverage of the CORDIC-DA FFT and parallel FFT are 98.67% and 97.46% respectively. The chips can not achieve high fault coverage possibly because of the hard-macro memory modules that are treated as black boxes. A BIST (Built-In Self Test) strategy may help increasing fault coverage.

5.4 Design Comparison

In such a case of 64-point FFT processor, the parallel FFT is superior than the proposed CORDIC-DA in many views, including timing, costs, power, and accuracy. Table 5.1 lists the summary of the two chips. The 4-channel version of CORDIC-DA spends more area mainly because the duplicated datapath. The PSNR of CORDIC-DA FFT is a bit lower than parallel FFT. It may possibly because of the shift-accumulate mechanism in the DA stage. The shifted bits in the accumulator are discarded right away for each cycle so that the accuracy of the final result will be affected.

Design		CORDIC-DA (4-channel)	Parallel FFT
Clock rate		100 MHz	20 MHz
Datapath width		16 bits	16 bits
Latency		10020 ns	9900 ns
PSNR	Avg.	100.39 dB	103.27 dB
	Min.	98.77 dB	101.37 dB
	Max.	102.81 dB	105.75 dB
Synthesized gate count		79585 (with testing circuits)	69603 (with testing circuits)
Core size		1200 x 1200 um^2	1060 x 1060 um^2
Die Size		2350 x 2350 um^2	2350 x 2350 um^2
Timing		100 MHz (bit-serial zone) 20 MHz (bit-parallel zone)	20 MHz
Core power		45.5 mW @ 100 MHz	16.8 mW @ 20 MHz
Die power		62.6 mW @ 100 MHz	28.6 mW @ 20 MHz

Table 5.1　Summary of two test chips

Chapter 6 Conclusion and Future Work

6.1 Conclusion

In this thesis, a study on recursive architecture FFT processor has been presented, and a new structure with CORDIC-DA technique has been proposed. Under the specification of 802.11a, this structure is extremely superior to the conventional radix-8 FFT processor implementations on area, but inferior on clock, performance, power, etc. We had made efforts on optimize the CORDIC-DA structure in many ways, including:

1) Twiddle factor rotator integration
2) Register balancing
3) LUT size reduction
4) Datapath duplication
5) Recursion flattening

However, the performance is still not as we expected. The main issue is the bit-serial arithmetic adopted in DA that is a trade-off between area cost and timing. The high clock rate required by CORDIC-DA is a derivative problem from bit-serial arithmetic. First, the clock source is hard to generated without an in-chip PLL circuit. Second, the high clock rate will make the logic synthesizer to insert more buffers and strengthen the driving ability of cells that consume more power than a low clock rate condition. Finally, we can make a conclusion that under the bit-parallel recursive architecture long-length FFT, the 8-point parallel FFT is the best implementation for the branch FFT considering the trade-off among many aspects, including performance, power, cost, and complexity.

Although the proposed structure works worse for the recursive architecture FFT, it is still recommendable in specific applications. The 1-channel version of CORDIC-DA is good at area costs regardless of the high clock rate. If there is an application that requires very small area cost, the 1-channel CORDIC-DA may be one of the best solutions. In the view of performance, if a serial-in serial-out application which requires high-speed FFT computations and regardless of power consumption, the CORDIC-DA may be suitable for implementation.

6.2 Future Work

In previous section, a conclusion for implementing radix-8 FFT processors is described that the parallel FFT should be the first consideration. However, there is still much research to do with the proposed CORDIC-DA structure. First, the test chip is implemented with a high specification of 16-bit wordlength while the output is also 16-bit and 2^6 scaled. The datapath can be designed more carefully if a precise error analysis was done. Hence, the resource cost will be reduced while keeping the same SQNR performance. The memory access is another issue. For a bigger N, the memory access will be more complex, and how to improve the efficiency and simplify the memory access scheme in the CORDIC-DA structure is left for future work.

Although the performance of the CORDIC-DA structure is not good enough as we expected, the idea proposed in this thesis is still advisable for optimizing bit-serial arithmetic architecture and DA-twiddle factor integration. In the future, longer-length FFT processors using parallel FFT implementations will be constructed, and we will keep study on the optimization of long-length FFT processors based on short-length FFT and recursive architecture.

REFERENCES

[1] *Information Technology- Telecommunications and Information Exchange between Systems- Local and Metropolitan Area Networks- Specific Requirements Part 11: Wireless Lan Medium Access Control (MAC) and Physical Layer (PHY) Specifications Amendment 1: High-Speed Physical Layer in the 5 GHz Band*, IEEE P802.11a/D7.0.

[2] Y. H. Hu, "CORDIC-Based VLSI Architectures for Digital Signal Processing," *IEEE Signal Processing Magazine*, pp. 16-35, July 1992.

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

[4] Stanley A. White, "Applications of Distributed Arithmetic to Digital Signal Processing: A Tutorial Review," *IEEE ASSP Magazine*, pp. 4-19, July 1989.

[5] A. Berkeman, V. Öwall, and M. Torkelson, "A Low Logic Depth Complex Multiplier Using Distributed Arithmetic," *IEEE J. of Solid-State Circuits*, vol. 35, no. 4, pp. 656-659, Apr. 2000.

[6] C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. Electron. Comput.* vol. 13, pp. 14-17, Feb. 1964.

[7] A. D. Booth, "A Signed Binary Multiplication Technique," *Q. J. Mech. Appl. Math.*, vol. 4, pp. 236-240, 1951.

[8] M. Alidina, J. Monteiro, S. Devas, A. Ghosh, and M. Papaefthymiou, "Precomputational-Based Sequential Logic Optimization for Low Power," *IEEE Trans. on VLSI Systems*, vol. 2, no.4, pp. 426-436, Dec. 1994.

[9] Koushik Maharatna, Eckhard Grass, and Ulrich Jagdhold, "A 64-Point Fourier Transform Chip for High-Speed Wireless LAN Application Using OFDM," *IEEE J. of Solid-State Circuits*, vol. 39, no. 3, pp. 484-493, Mar. 2004.

[10] Tome Chen, Glen Sunada, and Jian Jin, “COBRA: A 100-MOPS Single-Chip Programmable and Expandable FFT,” *IEEE trans. on VLSI Systems*, vol. 7, no. 2, pp. 174-182, Jun. 1999.

[11] Bevan M. Baas, “A Low-Power, High-Performance, 1024-Point FFT Processor,” *IEEE J. of Solid-State Circuits*, vol. 34, no. 3, pp. 380-387, Mar. 1999.

[12] Y. W. Lin, H. Y. Liu, and C.Y. Lee, “A Dynamic Scaling FFT Processor for DVB-T Applications,” *IEEE J. of Solid-State Circuits*, vol. 39, no. 11, pp. 2005-2013, Nov. 2004.

[13] Y. W. Lin, H. Y. Liu, and C. Y. Lee, “A 1-GS/s FFT/IFFT Processor for UWB Applications,” *IEEE J. of Solid-State Circuits*, vol.40, no.8, pp. 1726-1735, Aug. 2005.

[14] Soontorn Oraintara, Y. J. Chen, Truong Q. Nguyen, “Integer Fast Fourier Transform,” *IEEE trans. on Signal Processing*, vol. 50, no. 3, pp. 607-618, Mar. 2002.

[15] R. Thamvichai, T. Bose, and M. Radenkovic, “Fast Integer Fourier Transform (FIFT) Based on Lifting Matrics,” ISCAS 2003, pp. IV-85-IV88, May 2003.

[16] G. Zhong, F. Xu, and Alan N. Willson, Jr., “A Power-Scalable Reconfigurable FFT/IFFT IC Based on a Multi-Processor Ring,” *IEEE J. of Solid-State Circuits*, vol. 41, no. 2, pp. 483-495, Feb. 2006.

[17] S. He and Mats Torkelson, “Designing Pipeline FFT Processor for OFDM (de)Modulation,” ISSSE 98, pp. 257-262, Oct. 1998.

[18] L. D. Van and C. C. Yang, “High-Speed Area-Efficient Recursive DFT/IDFT Architectures,” ISCAS 2004, vol. 3, pp. III-357-III-360, May 2004.

[19] J. C. Kuo, C. H. Wen, and A. Y. Wu, “Implementation of a Programmable 64~2048-Poing FFT/IFFT Processor for OFDM-Based Communication Systems,” ISCAS 2003, vol. 2, pp. II-121-II-124, May 2003.

[20] S. Magar, S. Shen, G. Luikuo, M. Fleming, and R. Aguilar, "An Application Specific DSP Chip Set for 100 MHz Data Rates," ICASSP 1998, vol.4, pp.1989-1992, Apr. 1988.

[21] E. H. World and A. M. Despain, "Pipeline and parallel pipeline FFT processors for VLSI implementation," *IEEE trans. Comput.* vol. C-33, pp. 414-426, May 1984.

[22] M. D. Ercegovac and T. Lang, "Redundant and On-Line CORDIC: Application to Matrix Triangularization and SVD," *IEEE trans. on Computers,* vol. 39, no. 6, pp. 725-740, Jun. 1990.

[23] N. Takagi, T Asada, and S. Yajima, "Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation," *IEEE trans. on Computers*, vol. 40, no. 9, pp. 989-994, Sep, 1991.

[24] H. Dawid and H. Meyr, "The Differential CORDIC Algorithm: Constant Scale Factor Redundant Implementation without Correcting Iterations," *IEEE trans. on Computers*, vol. 45, no. 3, pp. 307-318, Mar. 1996.

[25] D. Timmermann, H. Hahn, and B. J. Hosticka, "Low Latency Time CORDIC Algorithms," *IEEE trans. on Computers*, vol.41, no. 8, pp. 1010-1015, Aug. 1992.

[26] J. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330-334, Sep. 1959.

[27] P. D. Welch, "A Fixed-Point Fast Fourier Transform Error Analysis," *IEEE trans. on audio and electroacoustics,* vol. AU-17, no. 2, pp. 151-157, Jun. 1969

[28] A. V. Oppenheim and C. J. Weinstein, "Effects of Finite Register Length in Digital Filtering and the Fast Fourier Transform," *proc. of the IEEE,* vol. 60, no. 8, pp. 957-976, Aug. 1972.

[29] M. Sundaramurthy and V. U. Reddy, "Some Results in Fixed-Point Fast Fourier

Transform Error Analysis," *IEEE trans. on computers*, pp. 305-308, Mar. 1997.

[30] L. R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Englewood Cliffs, NJ: Prentice Hall, 1975

[31] Synopsys Design Compiler User Guide, Version W-2004.12, Dec. 2004.

[32] Synopsys Design Compiler Reference Manual: Constraints and Timing, Version W-2004.12, Dec. 2004.

[33] Artisan Standard Library 0.13um-0.25um Register File Generator User Manual, Release ug_2004q2v0.

[34] Artisan Standard Library SRAM Generator User Manual, Release ug_2004q1v0.

[35] Artisan Standard Library ROM Generator User Manual, Release ug_2003q4v2.