High-Speed and Low-Power Split-Radix FFT

Wen-Chang Yeh and Chein-Wei Jen

***Abstract*—This paper presents a novel split-radix fast Fourier transform (SRFFT) pipeline architecture design. A mapping methodology has been developed to obtain regular and modular pipeline for split-radix algorithm. The pipeline is repartitioned to balance the latency between complex multiplication and butterfly operation by using carry-save addition. The number of complex multiplier is minimized via a bit-inverse and bit-reverse data scheduling scheme. One can also apply the design methodology described here to obtain regular and modular pipeline for the other Cooley–Tukey-based algorithms.**

For an $N(=2^n)$-point FFT, the requirements are $\log_4 N-1$ multipliers, $4\log_4 N$ complex adders, and memory of size $N-1$ complex words for data reordering. The initial latency is $N+2\cdot\log_2 N$ clock cycles. On the average, it completes an N-point FFT in N clock cycles. From post-layout simulations, the maximum clock rate is 150 MHz (75 MHz) at 3.3 v (2.7 v), 25°C (100°C) using a 0.35-μm cell library from Avant!. A 64-point SRFFT pipeline design has been implemented and consumes 507 mW at 100 MHz, 3.3 v, and 25°C. Compared with a radix-2^2 FFT implementation, the power consumption is reduced by an amount of 15%, whereas the speed is improved by 14.5%.

***Index Terms*—Low power FFT, split-radix FFT.**

I. Introduction

THE FAST Fourier transform (FFT) and its inverse (IFFT) are essential in the field of digital signal processing. Recently, due to the popularity of the orthogonal frequency division multiplex (OFDM) system, the demand for high-speed and low-power FFT emerges from various applications. According to the European digital video/audio broadcasting (DVB-T/DAB) standards, an OFDM system may require FFT length ranging from 256- to 8192-point. Wireless local area network (WLAN) and HIPERLAN/2 systems require high-speed and low-power FFT/IFFT design [1], [2]. The fourth-generation cellular phone and the forthcoming new WLAN systems may also incorporate OFDM system to deliver higher bandwidth [3]. Hence, it is important to design high-performance and low-power FFT for these applications.

For a static CMOS circuit, the power consumption is usually determined by the dynamic power, which can be written as

$$P_{\text{dynamic}} = \alpha \cdot C_L \cdot V_{dd}^2 \cdot f_{clk} \tag{1}$$

where α is the switching probability, C_L is the capacitance being charged or discharged when switching, V_{dd} is the supply voltage, and f_{clk} is the clock rate. At architecture level, α can be regarded as the *operation count* for a specific module, and C_L is proportional to the part of a module that is being active. For example, for a multiplier, if the operation count is MUL# within T clock cycles, then α is proportional to $MUL\#/T$.

As the α in (1) is usually determined by algorithm and its corresponding architecture, it is desirable that the chosen FFT algorithm has the least computational complexity as well as the corresponding hardware complexity. Among various FFT algorithms, the Cooley–Tukey algorithm [4] is very popular because it can reduce the computational complexity from $O(N^2)$ to $O(N\log_2 N)$, and the regularity of the algorithm makes it suitable for VLSI implementation. To further reduce the computational complexity, radix-4, split-radix [5], radix-2^2 [6], radix-2/4/8 [7], and higher radix versions have been proposed. In general, all of these algorithms decompose a length-N ($=2^n$) FFT into odd half and even half recursively and effectively reduce the number of complex multiplications by utilizing symmetric properties of the FFT kernel. The split-radix algorithm is the best in terms of the multiplicative complexity for N-point FFT when the multiplications with ± 1, $\pm j$ are skipped. However, split-radix algorithm is inherently irregular because radix-2 stages are used for even half components, and radix-4 stages are used for odd half components, which results in an "L"-shaped butterfly unit.

Due to the irregularity of the butterfly unit, it is hard to design a regular and modular hardware pipeline for the split-radix algorithm. In [8], a two-dimensional (2-D) processor array was proposed to implement split-radix algorithm. Its hardware complexity grows at $O(N\cdot\log N)$, which makes such a design impractical for large N. In [9], a one-dimensional (1-D) linear array design was proposed. Although its hardware complexity has been reduced to $O(\log N)$ only, the hardware requirement is more than twice as large as those of the other radix-4 FFT implementations proposed in [6], [10], and [11].

In this paper, we present a SRFFT pipeline architecture that implements the split-radix algorithm efficiently and is suitable for VLSI implementation. The number of multipliers has been minimized to $\log_4 N-1$ by means of sharing the multiplier between two adjacent stages. The sharing is achieved by using the bit-inverse and bit-reverse (BIBR) data scheduling scheme proposed here. The hardware complexity is equivalent to that of the radix-2^2 pipeline architecture [6].

In order to balance the latency between complex multiplication and butterfly operation, the complex multiplier has been pipelined into two stages. The first stage is based on Wallace tree and modified Booth recoding, and the second stage is the final addition of the multiplication. The second stage is then merged

Manuscript received May 30, 2001; revised October 2, 2002. The work was supported by the National Science Council of Taiwan, R.O.C., under Grants NSC 89–2218-E009–078 and NSC89–2218-E009–085. The associate editor coordinating the review of this paper and approving it for publication was Prof. Chaitali Chakrabarti.

W.-C. Yeh is with ZyDAS Technology Corporation, Hsinchu, Taiwan, R.O.C. (e-mail: wcyeh@zydas.com.tw).

C.-W. Jen is with the Department of Electronics Engineering and Institute of Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C. (e-mail: cwjen@twins.ee.nctu.edu.tw).

Digital Object Identifier 10.1109/TSP.2002.806904

into the succeeding two butterfly units to balance the latency. The repartition transforms the carry-propagation additions at the end of multiplication into carry-save additions, which can further reduce power consumption and increase performance without increasing hardware cost.

The proposed design approach can be generalized to other 2^n-point FFT algorithms based on Cooley-Tukey decomposition to obtain regular pipeline architecture as well. For comparison, we have implemented a 64-point SRFFT pipeline and a 64-point radix-2^2 FFT pipeline. The post-layout simulation shows that the SRFFT design can operate at 150 MHz, 3.3 v, and 25°C. We have achieved 15% power reduction and 14.5% performance improvement when compared with the radix-2^2 design under equal conditions.

The organization of this paper is as follows. In Section II, we will analyze the relationships between FFT algorithms and pipeline architecture. The proposed SRFFT architecture and the multiplier folding scheme will be discussed in Section III. In Section IV, we will present the design of delay-balanced pipeline architecture. Post-layout simulation and its analysis are given in Section V.

II. Analysis of Radix-2^n FFT Algorithms

A. Basic Formulation and Low Radix Algorithms

Given an input sequence x_n, an N-point discrete Fourier transform is defined as

$$A_k = \sum_{n=0}^{N-1} x_n \cdot W_N^{nk} \quad k = 0, 1, 2, \ldots, N-1 \tag{2}$$

where the n is the time index, and the k is the frequency index. The coefficient W_N^{nk} is defined as

$$W_N^{nk} = \cos\left(\frac{2\pi \cdot nk}{N}\right) - j \cdot \sin\left(\frac{2\pi \cdot nk}{N}\right). \tag{3}$$

For Cooley–Tukey radix-2 decimation-in-frequency (DIF) decomposition (2) is decomposed into even and odd frequency components

$$A_{2k} = \sum_{n=0}^{(N/2)-1} \left(x_n + x_{n+(N/2)}\right) \cdot W_{N/2}^{nk} \tag{4}$$

$$A_{2k+1} = \sum_{n=0}^{(N/2)-1} (x_n - x_{n+(N/2)}) W_N^n \cdot W_{N/2}^{nk}. \tag{5}$$

In (5), the W_N^n is usually referred to as *twiddle factor*. The SRFFT algorithm [5] further decomposes the odd frequency component into $4k+1$ and $4k+3$ frequency components, shown in (6) and (7) at the bottom of the page. By applying (4), (6), and (7) recursively, the split-radix FFT can be obtained. On the other hand, the radix-4 algorithm can be obtained by decomposing (4) and (5) into A_{4k}, A_{4k+2}, A_{4k+1}, and A_{4k+3} frequency components. Table I shows the multiplicative complexity of the radix-2, radix-4, and split-radix algorithms. For non-4^n length FFT, an additional radix-2 stage is used for the radix-4 algorithm. The split-radix algorithm shows a clear advantages over the other algorithms. To understand the relationship among the three algorithms intuitively, we can examine their signal flow graphs (SFGs) as shown in Fig. 1. From the figure, we can see the following.

- Both the radix-4 and split-radix are superior to the radix-2 algorithm because "$-j$" terms are extracted.

  The complex multiplications with $-j$ are accomplished by exchanging the real and the imaginary parts of the incoming data and then inverting the sign of the imaginary part.
- The split-radix algorithm is superior to the radix-4 algorithm because more "$-j$" terms are extracted in the SFG.

  Note that four twiddle factors are moved from the end of the second butterfly stage to the end of the third butterfly stage, and two of them become trivial multiplications.

TABLE I
Nontrivial Complex Multiplications Required for 2^n-Point FFT Algorithms

DFT size	Radix-2	Radix-4	Split-Radix
8	2	2	2
16	10	8	8
32	34	28	26
64	98	76	72
128	258	204	186
256	642	492	456
512	1538	1196	1082
1024	3586	2732	2504
2048	8194	6316	5690
4096	18434	13996	12744
8192	40962	31404	28218

B. High Radix Algorithms

If multiplicative complexity lower than the split-radix algorithm is desirable, higher radix FFT algorithms should

$$A_{4k+1} = \sum_{n=0}^{(N/4)-1} \left(x_n - j \cdot x_{n+(N/4)} - x_{n+(N/2)} + j \cdot x_{n+(3N/4)}\right) W_N^n \cdot W_{(N/4)}^{nk} \tag{6}$$

$$A_{4k+3} = \sum_{n=0}^{(N/4)-1} \left(x_n + j \cdot x_{n+(N/4)} - x_{n+(N/2)} - j \cdot x_{n+(3N/4)}\right) W_N^{3n} \cdot W_{(N/4)}^{nk}. \tag{7}$$

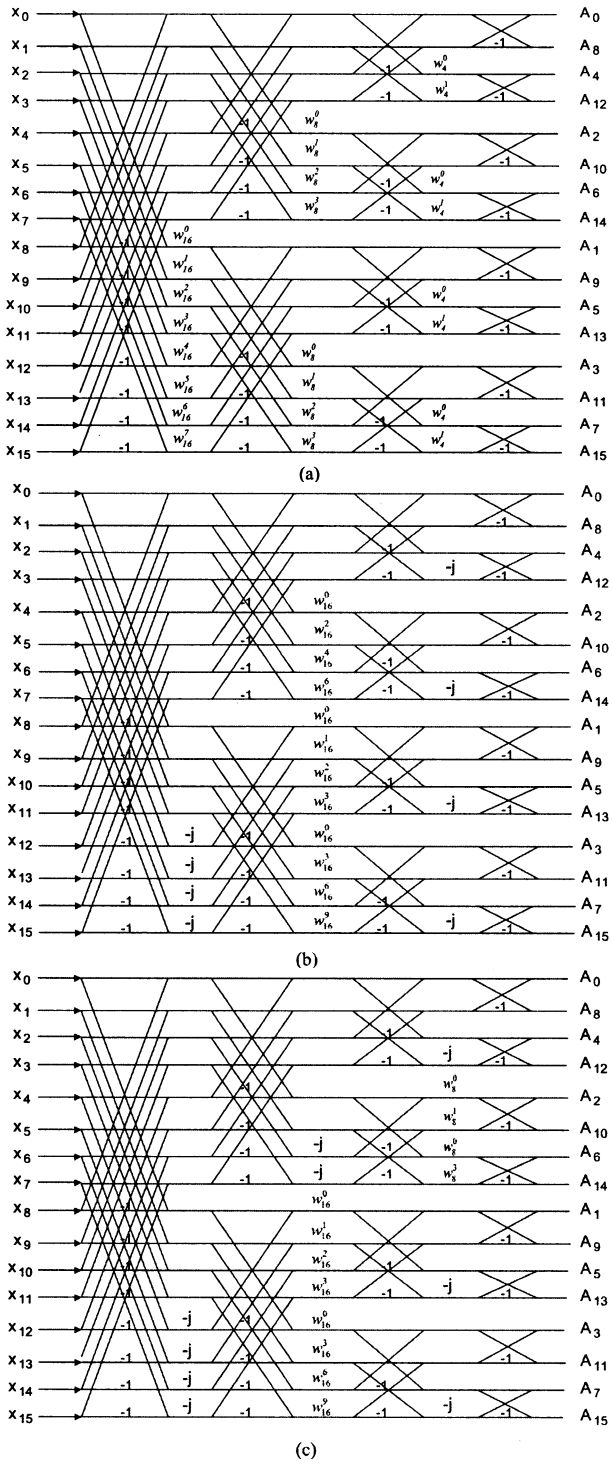

Fig. 1. Signal flow graph of 16-point (a) radix-2 FFT, (b) radix-4 FFT, and (c) split-radix FFT.

be used. However, high-radix FFT algorithms often increase the circuit complexity and are not easy to implement. To discuss and to compare the efficiency of high-radix algorithms, radix-8 and radix-2/8 algorithms are examined here. The radix-2/8 algorithm can be considered to be an extension to split-radix(radix-2/4) algorithm by decomposing (6) and (7) one more radix-2 stage. The SFGs of radix-8 and radix-2/8 algorithms are shown in Fig. 2. Besides ± 1 and $\pm j$ terms, W_8^1 and W_8^3 are also extracted from the SFGs. Due to the symmetric

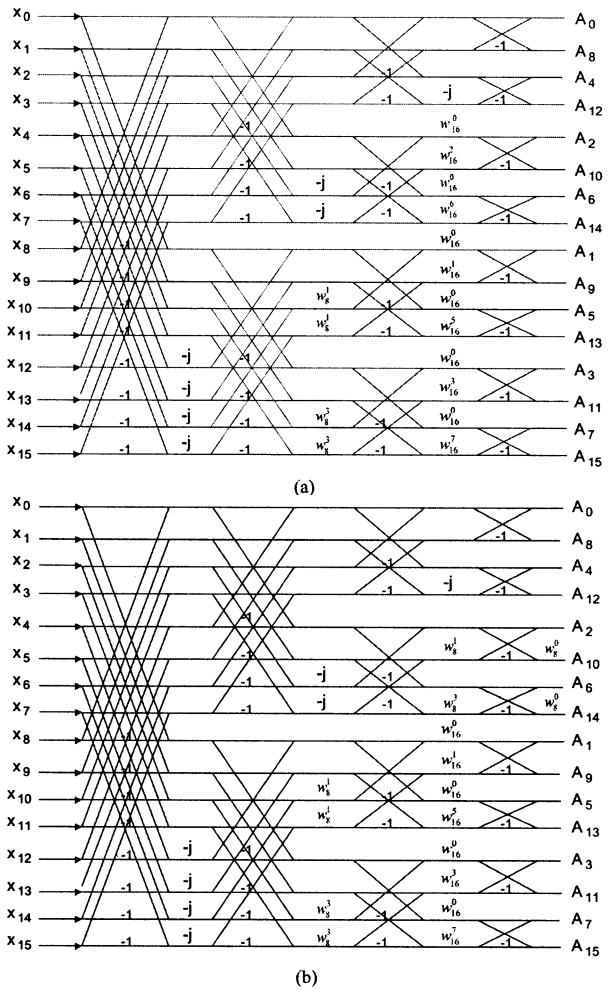

Fig. 2. Signal flow graph of 16-point (a) radix-8 FFT and (b) radix-2/8 FFT.

properties of cosine and sine functions, the values of W_8^1 and W_8^3 can be written as $(\sqrt{2}/2)(1-j)$ and $-(\sqrt{2}/2)(1+j)$, respectively. Thus, a complex multiplication with one of the two coefficients can be computed using a constant multiplier and additions. The number of complex multiplications and the number of constant multiplications are summarized in Table II.

Obviously, the radix-2/8 algorithm has lower multiplicative complexity than the radix-8 algorithm because it extracts more W_8^1 and W_8^3 coefficients in the SFG. According to our hardware implementation, we found that the area of the constant multiplier is one and half times of a real multiplier. Hence, one constant multiplication is approximately equivalent to 0.4 complex multiplications. To compare the multiplicative complexity between low radix algorithms and high radix algorithms, one can multiply the number of constant multiplications by 0.4 and calculate the equivalent number of complex multiplications.

It is interesting to observe that the multiplicative complexity of the radix-8 algorithm derived in this work is the same as that of the radix-2/4/8 algorithm reported in [7]. Actually, if we examine Fig. 2(a) more carefully, we will find that the SFG of the radix-8 algorithm is equivalent to that of radix-2/4/8 algorithm. The radix-2/4/8 algorithm implements the radix-8 butterfly using three radix-2 stages instead of one single butterfly. Therefore, we may term the radix-2/4/8 algorithm the radix-2^3

TABLE II
NONTRIVIAL MULTIPLICATIONS REQUIRED FOR RADIX-8 AND RADIX-2/8 FFT ALGORITHMS

DFT size	Radix-8		Radix-2/8	
	Complex	Constant	Complex	Constant
8	0	2	0	2
16	6	4	4	6
32	20	8	16	14
64	48	32	44	38
128	152	64	120	94
256	376	128	308	214
512	824	384	736	494
1024	2104	768	1724	1126
2048	4792	1536	3976	2494
4096	10168	4096	8964	5494
8192	23992	8192	19952	12046

because of the equivalence at algorithm level. The butterfly unit consists of three radix-2 butterfly stages. The radix-2^3 design proposed in [6] can be regarded as another radix-2/4/8 design. Based on this observation, we will not evaluate the performance of radix-2/4/8 algorithm any further.

The difference equations used here to compute the number of complex multiplication for an N $(= 2^n)$-point FFT are

$$M8_n = 8 \cdot M8_{n-3} + 7 \cdot 2^{n-3} - 8 \tag{8}$$

for the radix-8 algorithm, and

$$M28_n = M28_{n-1} + 4 \cdot M28_{n-3} + 2^{n-1} - 4 \tag{9}$$

for the radix-2/8 algorithm. One can use these equations to verify the correctness of Table II.

C. Tradeoff Among the Algorithms

Based on the discussion in the previous sections, we can see that if the multiplications with ± 1 and $\pm j$ are removed and the multiplications with $\pm(\sqrt{2}/2)(1 \mp j)$ can be realized using constant multiplication, the radix-2/8 algorithm will have the lowest multiplicative complexity among all the discussed algorithms. On the other hand, the fixed-radix algorithms have more regular SFGs than the mixed-radix algorithms. However, the radix-4 and radix-8 algorithms can be applied to 4^n-point and 8^n-point FFT's only, unless a radix-2 stage is also employed in the pipeline. Such a limitation also exists in other fixed-radix FFT algorithms but does not in the split-radix or the radix-2/8 algorithm.

The additive complexity has not been analyzed because it is basically the same for all the algorithms based on Cooley–Tukey decomposition, as discussed in [5]. Thus, the number of additions can be further reduced if we can minimize the number of multiplications and implement each multiplication using as few additions as possible. To summarize, mixed radix algorithms are quite attractive if regular and efficient hardware architecture can be found. We will present the proposed pipeline architecture for the split-radix algorithm in Section III and the delay-balanced pipeline to remove unnecessary computation in Section IV.

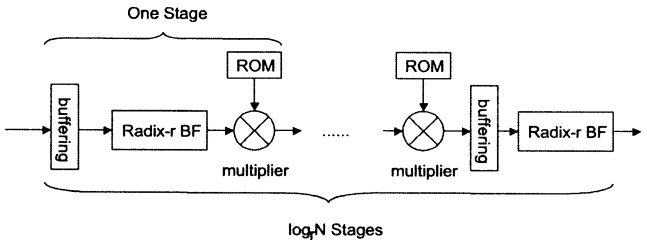

Fig. 3. Conventional radix-r pipeline architecture.

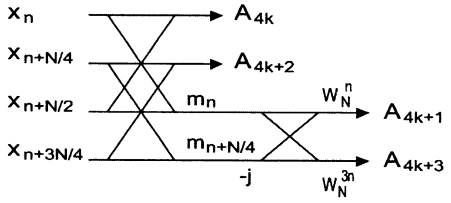

Fig. 4. "L"-shaped split-radix butterfly network.

III. PIPELINE ARCHITECTURE FOR SRFFT

A. Previous Work

The one-dimensional linear array is very popular because it possesses regularity, modularity, local connection, and high throughput with moderate hardware complexity. Fig. 3 shows the commonly used radix-r 1-D pipeline architecture [11]. The number of butterflies and the number of multipliers are proportional to $\log_r N$. However, the scheme shown in the figure cannot be applied to mixed-radix algorithms directly. The problem arises from the irregularity of the butterfly stage for mixed-radix algorithms. Take the split-radix algorithm as an example; both (6) and (7) can be rewritten in alternative form by defining $(x_n - x_{n+(N/2)})$ in (5) as m_n.

$$A_{4k+1} = \sum_{n=0}^{(N/4)-1} \left(m_n - j \cdot m_{n+(N/4)}\right) W_N^n \cdot W_{N/4}^{nk} \tag{10}$$

$$A_{4k+3} = \sum_{n=0}^{(N/4)-1} \left(m_n + j \cdot m_{n+(N/4)}\right) W_N^{3n} \cdot W_{N/4}^{nk}. \tag{11}$$

If (4), (10), and (11) are mapped directly into hardware, the shape of the butterfly unit would look like the one shown in Fig. 4. Such an "L"-shaped butterfly unit is difficult to be integrated into pipeline architecture, and a similar problem also arises in radix-2/8 algorithm [7]–[9]. In [9], a delay-commutator(DC)-based design was proposed to implement the split-radix algorithm. Although the design achieves the multiplicative complexity of the split-radix algorithm, the hardware requirement is considerably much higher than the other pipeline architectures.

Table III compares the hardware requirement and the multiplicative complexity for several classical and new implementations [6], [9]–[13]. The taxonomy is adopted from [6]. Split- radix single-path delay-feedback is denoted SRSDF, and SRMDC denotes split-radix multiple-path delay-commutator. Because there are two different radix-2^2 SDF designs, we

TABLE III
HARDWARE REQUIREMENT COMPARISON FOR N-POINT FFT ALGORITHM, $N = 2^n$

Architecture	Complex Multiplier	Complex Adder	Complex MEM size	Multiplicative Complexity	Complex MEM Access	Proposed in
R2MDC	$2(log_4$N-1)	$4log_4$N	1.5N-2	Radix-2	$N \cdot log_2$N	[10]
R4MDC	$3(log_4$N-1)	$8log_4$N	2.5N-4	Radix-4	$N \cdot log_4$N	[10]
R4SDC	$(log_4$N-1)	$3log_4$N	2N-2	Radix-4	$N \cdot log_4$N	[13]
SRMDC	$4(log_4$N-1)	$12log_4$N-8	1.5N-2	Split-Radix	$N \cdot log_2$N	[9]
R2SDF	$2(log_4$N-1)	$4log_4$N	N-1	Radix-2	$N \cdot log_2$N	[11]
R4SDF	$(log_4$N-1)	$8log_4$N	N-1	Radix-4	$N \cdot log_4$N	[12]
R2^2SDFI	$(log_4$N-1)	$4log_4$N	N-1	Radix-4	$N \cdot log_2$N	[6]
R2^2SDFII	$(log_4$N-1)	$4log_4$N	N-1	Radix-4	$N \cdot log_2$N	here
SRSDF	$(log_4$N-1)	$4log_4$N	N-1	Split-Radix	$N \cdot log_2$N	here

denote the first one proposed in [6] as R2^2SDFI and the second one proposed in this work as R2^2SDFII. We will derive the SRSDF and R2^2SDFII architectures in the following sections.

B. Memory System Design

In order to compute DFT via FFT, the input data and the intermediate results have to be reordered using memory. The required memory size is directly proportional to N, and the number of memory access is proportional to $N \cdot \log_r N$. Therefore, it is important to reduce both the memory size and the number of memory access.

Take radix-2 FFT algorithm as an example. The computation of (4) cannot start until both x_n and $x_{n+(N/2)}$ are available. For word-sequential I/O, the two samples will be separated by $N/2$ clock cycles if one sample is available per clock cycle. Consequently, the first $N/2$ samples have to be stored in a local memory until the other data sample $x_{n+(N/2)}$ arrives. Similar constraints also exist in the other FFT algorithms.

Two different buffering strategies have been developed for pipeline FFT architecture. One is delay-commutator (DC) architecture, and the other one is delay-feedback (DF) architecture [6], as shown in Fig. 5. The DC approach is shown in Fig. 5(a). At the first $N/2$ cycles, the first $N/2$ samples are stored in "$N/2$ FIFO_I". At the next $N/2$ cycles, the butterfly receives $x_{n+(N/2)}$ from the input and x_n from "$N/2$ FIFO_I" and generates outputs according to (4) and (5). Meanwhile, one of the results generated by the first butterfly is stored into "$N/2$ FIFO_II," and the other one is fed to the multiplier directly. During the N cycles, data are stored into the "$N/2$ FIFO_I" FIFO in the first $N/2$ cycles and then are read from the FIFO in the second $N/2$ cycles. As a result, the utilization rate of each FIFO is only 50%.

For DF style as shown in Fig. 5(b), the incoming samples are stored in the "$N/2$ FIFO" during the first $N/2$ cycles. When $x_{n+(N/2)}$ arrives, the inputs of the radix-2 butterfly unit will receive $x_{n+(N/2)}$ from the input and x_n from the feedback FIFO

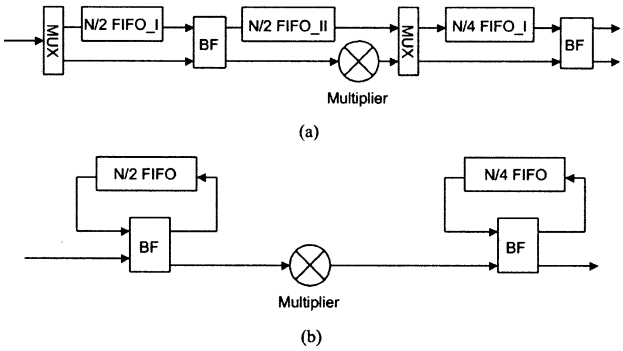

Fig. 5. Buffering strategy. (a) Radix-2 multiple-path DC style. (b) Radix-2 single-path DF style.

for computation. One of the outputs of the butterfly unit is fed back to the "$N/2$ FIFO" again, which explains the name "delay-feedback." Data is both read from and written to each memory cell of the FIFO every clock cycle. The utilization rate of each FIFO is increased to 100%.

Table III also compares the memory requirement. The delay-feedback buffering strategy can implement radix-2, radix-4, radix-2^2, and split-radix algorithms with only $(N-1)$ memory words. On the other hand, DC buffering strategy requires $1.5N$ words or more. Note that each word actually comprises of real-part and imaginary-part for complex FFT. Apparently, DF strategy is preferred as the memory size is proportional to N, whereas the number of multiplier, pipeline register, or butterfly unit is of order $\log_r N$ only.

Except for the size of the memory, it is also important to minimize the access to the memory in order to reduce the power consumption. In general, the number of memory access is related to the radix of FFT algorithm because $\log_r N$ radix-r stages are used for an N-point FFT pipeline, as shown in Fig. 3. Global memory access occurs only when data enter or leave each stage, and therefore, it is proportional to the number of stages in a pipeline. Taking the radix-2^2 pipeline as an example,

we can see that it has the same number of global memory access as radix-2 pipeline, although it performs radix-4 FFT algorithm. The number of local memory access inside a radix-r stage also depends on the design of BF. For example, a radix-4 BF operation can be implemented using one radix-4 stage or two radix-2 stages with local memory buffer. Compared with a single radix-4 stage, the latter one has lower cost but more local memory access. Table III also shows the number of global memory access for the pipeline based architectures.

C. Proposed Delay-Feedback Pipeline Architecture

Based on the discussions in the previous sections, we can conclude that a good pipeline architecture for the FFT should possess the following features.

- At algorithm level, it should achieve the multiplicative complexity as low as possible.
- It should be suitable for any power of two length FFT.
- At the architecture level, use the delay-feedback buffering strategy to minimize the memory size.
- It should have modular and regular modules, local routing, and low control complexity.

We achieve these features by *projection mapping*, which is a technique from systolic array [14]. At first, we observe that conventional fixed-radix pipeline architecture can be obtained through folding the SFG at the vertical direction. Fig. 6(a) uses radix-4 SFG as an example. The BF_I unit implements common butterfly operation. The BF_II unit operates in two modes. For the first mode, it works just as the BF_I unit. For the second mode, the input data will be multiplied by "$-j$" before normal butterfly operation. Due to the spatial regularity of the SFG of radix-4 algorithm as shown in Fig. 1(b), the coefficient "$-j$" exists at the beginning of odd stages only. Therefore, BF_I is used at even stages, and BF_II is used at odd stages. The obtained radix-4 FFT pipeline can use either the DC or DF buffering strategy to store intermediate data. When single-path delay-feedback (SDF) buffering strategy is chosen, the R2^{2}SDFII FFT pipeline listed in Table III is obtained. Clearly, it uses the same number of butterfly units, multipliers, and the same memory size as the R2^{2}SDFI FFT pipeline. Although the R2^{2}SDFI architecture was obtained through a different manner, we believe that the two R2^{2}SDF designs are equivalent at both the algorithm and at the architecture levels.

Based on the above procedure, we do the projection mapping for split-radix in the same way, and the result is shown in Fig. 6(b). The BF_II unit is used at every stage except for the first stage, and two multipliers are used. Compared with the R2^{2}SDFII pipeline, the obtained pipeline architecture for the split-radix algorithm uses more BF_II units and one more multiplier. The overall structure is still regular and suitable for VLSI implementation, and the control of the BF_II units and multipliers is similar to the R2^{2}SDFII pipeline. Note that buffering style in the figure is not specified as it can be determined by the designer. We will use DF style throughout this paper.

The problem of the obtained pipeline architecture for the split-radix algorithm is that the number of multipliers is the same as radix-2 pipeline architecture, which equals $2(\log_4 N - 1)$, as shown in Table III. The utilization rate of the second multiplier drops to 25% only, which is very inefficient. To reduce the number of multipliers, a multiplier sharing scheme is developed here. At first, we try to use only one multiplier for two successive stages, as shown in Fig. 7. A resource conflict problem arises if we use a conventional data scheduling scheme where the sum of a butterfly is sent to the next stage and the difference of a butterfly is fed into the buffer. The definitions of *sum of a butterfly* and *difference of a butterfly* are in accordance with a common radix-2 butterfly. The data coming from the two butterfly units will both require multiplication simultaneously in certain cases when a conventional data scheduling scheme is used.

To solve this problem, we propose a *bit-inverse and bit-reverse* (BIBR) data scheduling scheme. The BIBR scheme exchanges the order of output sequences from butterfly stages. Thus, for each butterfly unit, it stores the sum into the buffer and propagates the difference to the next stage first. The final output data sequence becomes BIBR order, instead of bit-reverse for the conventional scheme. Table IV compares the two data scheduling schemes. It is clear from the table that when the input data sequence x_n is in normal order for both schemes, the output sequence A_k will be in bit-reverse order for the conventional scheme and in BIBR order for the proposed scheme.

Finally, the SRSDF pipeline architecture described in Table III is obtained. A similar design procedure can be applied to obtain SDF pipeline for any algorithm based on Cooley–Tukey decomposition. However, the BIBR scheduling scheme does not eliminate the multiplier resource conflict for the R28SDF pipeline for the radix-2/8 algorithm when we try to use only $\log_8 N - 1$ multipliers. The R28SDF FFT pipeline still requires $(\log_2 N - 2)$ complex multipliers and constant multipliers. Further study is required on the issue of reducing the hardware requirement for the R28SDF pipeline.

IV. Design of Delay-Balanced Pipeline

A. Two-Stage Pipelined Complex Multiplier Design

The design of a complex multiplier is essential for any complex FFT implementation because it consists of four real multiplications and two real additions. In addition to the large area, the latency of the FFT pipeline is often limited by the latency of the complex multiplication as well. Hence, many efforts have been devoted to reducing the latency of complex multiplication and minimizing the power consumption.

The equation for complex multiplication can be written as

$$x \times W = A_R + jA_I = (x_R \cdot W_R - x_I \cdot W_I) + j(x_R \cdot W_I + x_I \cdot W_R) \tag{12}$$

where the x refers to the incoming data from the previous stage, W is the twiddle factor, and A is the result of multiplication. Real part and imaginary part data are denoted using subscripts $_R$ and $_I$, respectively. To obtain either A_R or A_I, two real multiplications and one real addition are required. Rather than design a new complex multiplier, we decided to repartition the pipeline to balance the latency because the basic problem for all of the previous designs is that the latency of a complex multiplication is in general twice as long as that of a butterfly operation. Although three-multiplication scheme can be used to reduce the

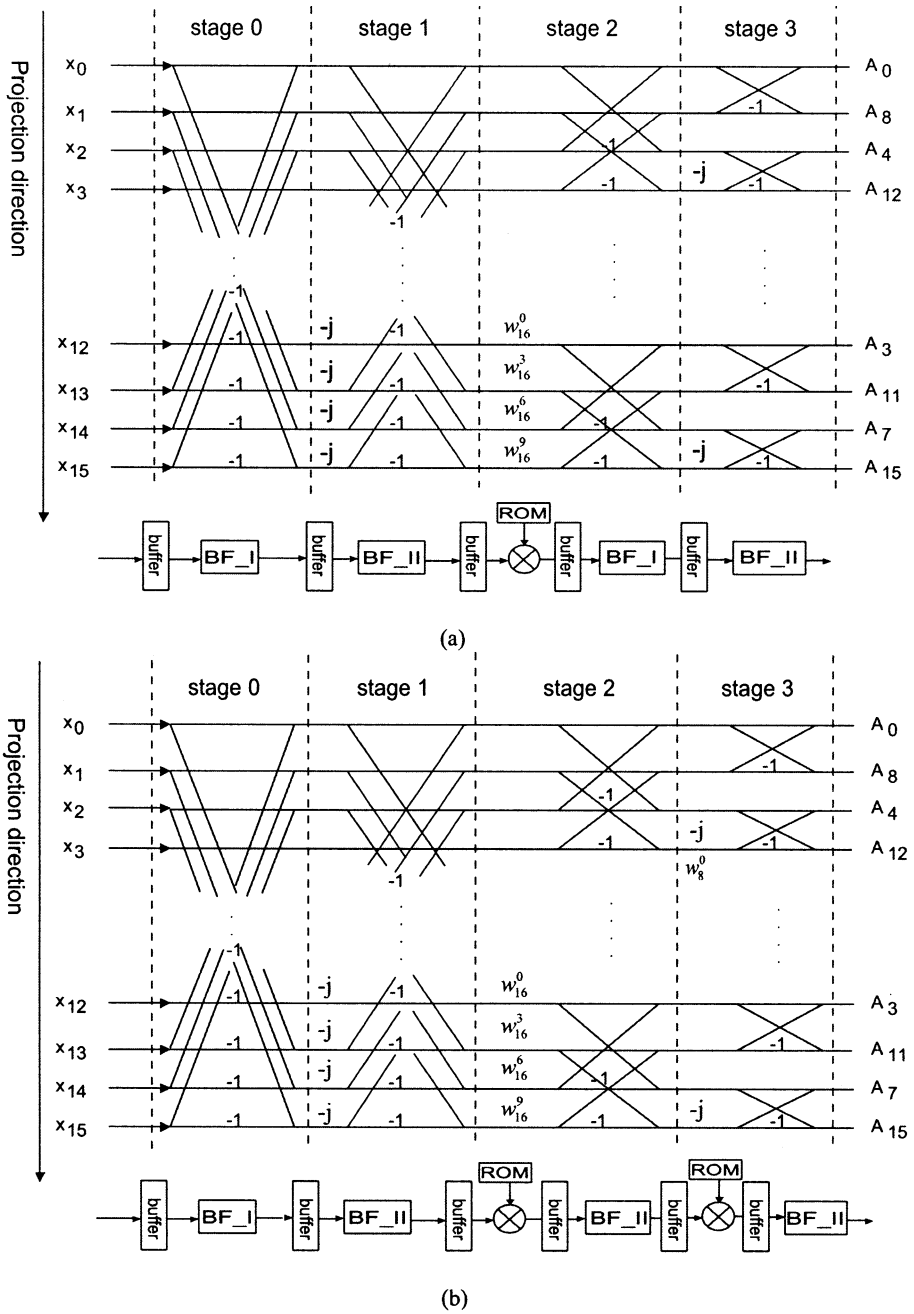

Fig. 6. Projection mapping of (a) Radix-4 SFG. (b) Split-radix SFG.

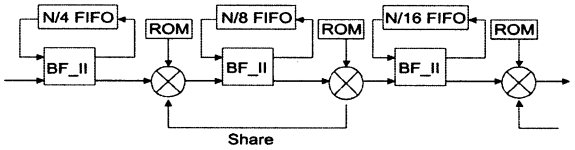

Fig. 7. Multiplier sharing scheme for the two successive stages of SRFFT pipeline.

area, it will further increase the latency of complex multiplication due to the encoding/recoding process [15]. Therefore, the repartition focuses on the balance of multiplication and butterfly operation, and it can be applied to either the common complex multiplication or the three-multiplication scheme.

At first, the partial products of the two multiplications are generated using modified-Booth encoding [16], [17] and then fed into a Wallace tree [18], [19]. The remaining two rows of partial products from the Wallace tree are converted to a two's-complement format using a final adder. Fig. 8(a) shows the structure of the generated multiplier before repartition. CPA denotes carry-propagation addition as the final addition has to be accomplished via a CPA adder.

The latency of each block in Fig. 8(a) is estimated as follows. Based on the multiplier optimization algorithms proposed in [20] and [21], the latency of several multipliers with different wordlengths are listed in Table V. Note that M is the wordlength of W_R (W_I), L is the wordlength of x_R (x_I), and the latency is normalized with respect to the delay of two inputs XOR gate.

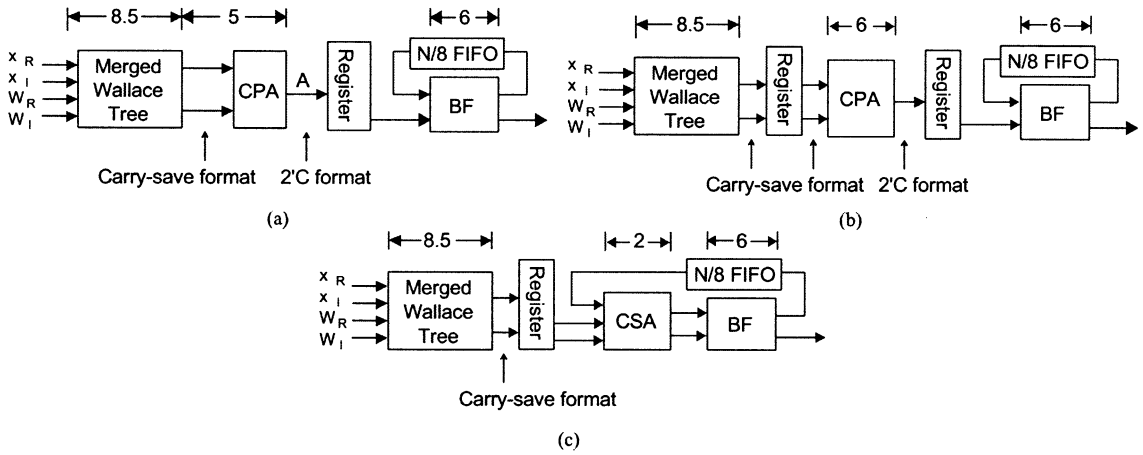

Fig. 8. Multiplier sharing scheme for successive two stages. (a) Multiplier before partition. (b) Add one pipeline stage. (c) Merge the final addition with butterfly.

TABLE IV
COMPARISON OF OUTPUT DATA SEQUENCE A_k FOR CONVENTIONAL SCHEME AND THE PROPOSED SCHEME

Input data x_n order	Conventional order (BR)	Proposed order (BIBR)
000	000	111
001	100	011
010	010	101
011	110	001
100	001	110
101	101	010
110	011	100
111	111	000

TABLE V
LATENCY OF WALLACE TREE AND FINAL ADDER FOR $(x_R \cdot W_R - x_I \cdot W_I)$ OR $(x_R \cdot W_I + x_I \cdot W_R)$ OPERATION

Size MxL	Merged Wallace Tree (XOR gate)	Final Adder (XOR gate)
12x14	8.5	5.0
12x16	8.5	5.0
12x18	8.5	5.0
12x20	8.5	5.5
12x24	8.5	5.5
12x28	8.5	5.5

As the latency of a Wallace tree is related to the height of the array formed by the partial products rather than the width, W_R and W_I are encoded using modified Booth encoding to generate partial products. The height is $M + 2$ for the merged Wallace tree with modified Booth encoding. However, if L is smaller than M, then x should be encoded to reduce the height. Because the wordlength of the twiddle factors is fixed at 12-bit ($= M$) in our design, the latency of the Wallace tree is fixed at 8.5 XOR gates. The latency of the final adder ranges from 5.0 to 5.5 XOR gates. If a butterfly unit uses fast CPAs for butterfly operation, its latency will be approximately $\log_2 L + 1$ XOR gates [22]. Therefore, the latency of the butterfly is around five or six XOR gates. The latency of each stage before partition is also shown in Fig. 8(a) based on the above estimation. If we partition the complex multiplier into two stages to balance the delay and do not modify the butterfly unit as shown in Fig. 8(b), the repartition will cause the area to increase due to the insertion of additional pipeline registers and will increase the total latency of the FFT. To avoid these problems, the final addition of the multiplier is merged into the butterfly units, as shown in Fig. 8(c). The butterfly operation has been modified to take the final addition into consideration.

As discussed in Section III-B, the butterfly operates in two phases. During the first phase, the incoming data from the previous stage are stored in a buffer. During the second phase, the data are computed according to (4) and (5), or (10) and (11), to complete the butterfly operation for split-radix algorithm. The modification here is done as follows. During the first phase, the carry-save format data from the previous stage are converted to two's-complement format using the adders and subtractors of the original butterfly unit. The data is then stored into the buffer as usual. During the second phase, the incoming data $x_{n+(N/2)}$ $(m_{n+N/4})$ in carry-save format and the stored data x_n (m_n) in two's-complement format are sent to carry-save adders, i.e., one row of full-adders, to replace the original CPA with a carry-save addition (CSA). The CSA is modified such that it can be used to compute either $(x_n \pm x^s_{n+(N/2)})$ or $(m_n \mp j \cdot m^s_{n+N/4})$. The superscript "$s$" denotes that the data is in carry-save format. The outputs of the CSA are then fed to the original butterfly unit to complete the computation. Note that the latency of a CSA is fixed at two XOR-gate delays, regardless of the wordlength. Therefore, the latency becomes 8.5 XOR gates for the multiplier stage and 8 $(= 2 + 6)$ XOR gates for the butterfly unit.

B. Benefits of the Delay Balanced Pipeline Design

The benefits of the proposed design are manifold. At first, the delay of the multiplier and the butterfly unit is now balanced. Second, the total number of the clock cycle will remain the same as that of the original design. Third, since the CPA of each mul-

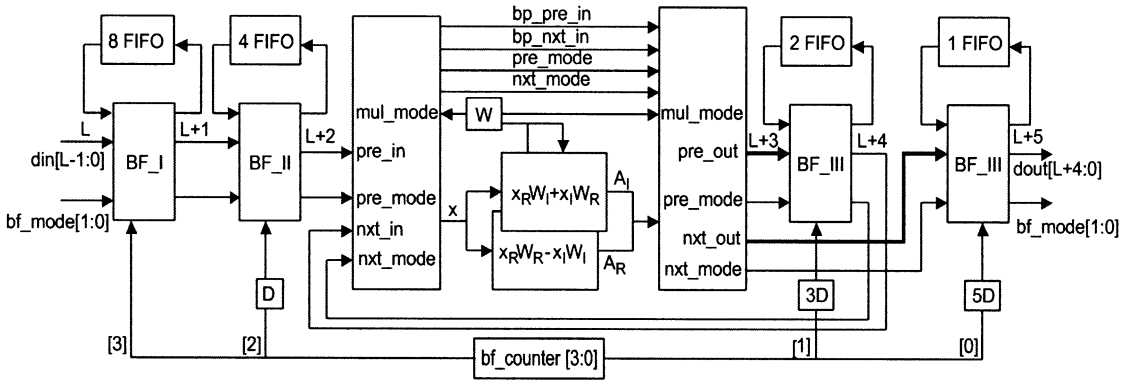

Fig. 9. Example of a delay-balanced SRSDF for $N = 16$ with minimized number of multiplier. The word-length L is increased by one bit at each stage. The bp_pre_in and bp_nxt_in are used to bypass pre_in and nxt_in, respectively, when either of them does not require a multiplication.

TABLE VI
HARDWARE COMPARISON BETWEEN THE ORIGINAL DESIGN AND THE DELAY-BALANCED DESIGN

Design	Original		Delay balanced	
Resource	CPA	Pipeline Register	CSA	Pipeline Register
Area	$Llog_2L$	2L	2L	4L
Note	For L=16, the cost is approximately the same.			

tiplication has been converted to CSA, the power consumption can be reduced by an amount of

$$\text{Reduced\_CPA}_{pwr} = 2\times(\text{CPA}_{pwr} - \text{CSA}_{pwr})\times\text{CMul\#} \quad (13)$$

where CPA_{pwr} denotes the power consumed by a CPA operation, CSA_{pwr} denotes the power consumed by a CSA, and CMul# denotes the number of complex multiplication for an FFT. Note that there are two CPAs for each complex multiplication. In addition to the power saving due to the conversion from CPA to CSA, the reduced latency can also be employed to reduce the power consumption via voltage scaling techniques.

Moreover, the repartition will not increase the total hardware cost. Table VI compares the cost of the original unbalanced design and the proposed balanced one. Assume that the cost of one bit full adder (FA) is equivalent to one bit register. For the original design, the CPA costs $L\log_2 L$ FAs, and there are $2L$ pipeline registers in our SRSDF architecture because each multiplier is connected to two butterfly units. When the CPA is merged into the butterfly units, we need two sets of CSA for the two butterfly units, and the number of pipeline register is doubled to $4L$ due to the carry-save format. The hardware cost is about the same for the two designs when $L = 16$.

C. Control Signals for Delay-Balanced SRSDF

A delay-balanced SRSDF pipeline for $N = 16$ is shown in Fig. 9. We use both local control signals and global control signals to synchronize the whole pipeline. The operation for a butterfly unit is determined by the received **bf_mode** and **bf_bypass** signals and is described as follows.

The bf_mode signal is a 2-bit local control signal that propagates with data. The three different states indicated by the bf_mode signal are defined in Table VII. When a butterfly unit receives st_normal from previous stage, it will generate sum according to (4) and generate difference according to (5). It will also generate a new bf_mode signal for the next stage according to Table VII. Similarly, if it receives st_mulj, it will multiply the input data with "$-j$" and then generate the sum and the difference. If a st_csa signal is received, the butterfly will send the incoming data into a carry-save adder and then generate correct outputs, as explained in Section IV-A. Therefore, three different butterfly units, BF_I, BF_II, and BF_III are used in this design. BF_I can accept st_normal, and BF_II can accept st_normal and st_mulj. BF_III can accept any one of the three states defined in the Table. The bf_bypass signal in the Table is generated by a $\log_2 N-1$-bit counter labeled with "bf_counter" in Fig. 9. The MSB is connected to the first butterfly stage so that the first stage will change its state every $N/2$ clock cycles. Similarly, the $(\log_2 N - 1 - k)$th bit of the counter will be connected to the kth butterfly stage, as shown in the figure. When bf_bypass is "1," the data will be bypassed and stored into buffer. When it is "0," the butterfly will carry out one of the three possible BF operations specified by bf_mode.

The operation of the multiplier is determined by the bf_mode signals from the previous and the next butterfly units and the mul_mode signal from the twiddle factor table, which is labeled as "W" in Fig. 9. The bf_mode from the previous butterfly will be connected to pre_mode port, and the other one from the next butterfly will be connected to nxt_mode port, as shown in the figure. When none of the bf_mode signals is in st_csa state, the data from both butterfly units are simply bypassed. When one of the bf_mode is in st_csa state, the multiplier will multiply this incoming data with a twiddle factor, and the other one will be bypassed. Note that the pre_mode and nxt_mode signals will not be both in st_csa state simultaneously because the proposed BIBR data scheduling scheme is used. The twiddle factor table follows similar rules to deliver correct twiddle factor for multiplication. The mul_mode signal from the table is set to one whenever current multiplication is trivial and can be skipped. Only if the trivial multiplications are eliminated can the multiplicative complexity of the SRSDF be as low as indicated in Table I.

TABLE VII
STATES INDICATED BY BF_MODE AND THE STATE TRANSITION TABLE

bf_mode	State	Next State	
		bf_bypass='1'	bf_bypass='0'
00	st_normal	st_normal	st_mulj
01	st_mulj	st_csa	st_csa
11	st_csa	st_normal	st_mulj

TABLE VIII
AREA PERCENTAGES OF THE MODULES OF THE 64-POINT SRFFT

Category	Control	Datapath			Feedback
		BF	MUL	ROM	memory
Area	0.51%	31.89%	46.57%	2.06%	18.94%

For an SRSDF design without delay balancing, the *bf_mode* local control signal is the additional control overhead compared with a R2^{2}SDF or R4SDF design. Note that the folding of multiplier is achieved by using BIBR scheduling and therefore does not increase any control complexity except for the multiplexers required at the input and output of a multiplication stage. The *st_csa* state is needed only when delay-balanced architecture is employed. The other control signals, such as *bf_counter* or *mul_mode*, are commonly employed in pipeline-based design. Thus, the overall control complexity is increased slightly due to the local control signal. The area percentages of control, datapath, and feedback memory for the 64-point SRSDF FFT are summarized in Table VIII. The size of control logic is very small compared with the other components. If the hardware for data multiplexing in multipliers and BF units is considered as control logic, the percentage of control logic will be increased to approximately seven. Most of the increase comes from the multiplexers for data routing in multipliers.

V. SIMULATION AND ANALYSIS

A. R2^{2}SDFII and SRSDF FFT Pipelines

For comparison, we have implemented a 64-point ($N = 64$) R2^{2}SDFII pipeline and a 64-point delay-balanced SRSDF FFT pipeline because they have similar hardware cost and performance, as shown in Table III. Note that although R2^{2}SDFII is obtained through the procedure described in this work, its performance should be similar to the R2^{2}SDF described in [6]. The input uses 12-bit ($L = 12$) for the real part and 12-bit for the imaginary part. To avoid overflow, the wordlength is adjusted by one bit at each stage, as shown in Fig. 9. The output is 20-bit for both real and imaginary parts for a 64-point FFT pipeline. The wordlength of the twiddle factors is fixed at 12-bit ($= M$) for real part and imaginary part. The result of multiplication exceeding the required wordlength is truncated directly. These parameters, including the N, L, and M, are configurable. The design is described using C/C++ language at first to verify the functionality and the effects of fixed point arithmetic. The C/C++ program is then converted to Verilog and then synthesized using the design analyzer from Synopsys. Automatic placement and route is done by Apollo from Avant!.

TABLE IX
PERFORMANCE COMPARISON OF 64-POINT FFT PIPELINE

Architecture	Gate Count	Critical Path	Power (mW) @100MHz, 3.3v
R2^{2}SDFII	40682	7.13ns	594.24
SRSDF	38168	6.09ns	507.85
Improvement	6%	14.5%	15%

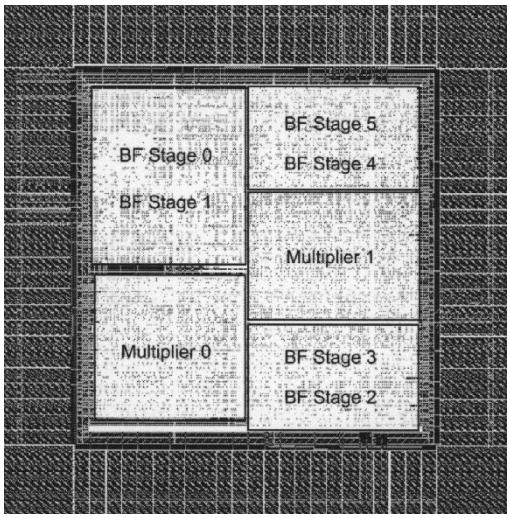

Fig. 10. Layout view of the 64-point SRFFT.

B. Area, Power, and Timing Performance

Table IX summaries the performance of the two designs. The SRSDF design can still function properly at 150 MHz at 3.3 V or 75 MHz at 2.7 V, whereas R2^{2}SDFII cannot. The layout view of the 64-point SRSDF FFT chip is shown in Fig. 10. It is pad limited with a core area of 1902 μm (H) $\times$ 1820 μm (W). The gate count and the critical path are reported by the synopsys design analyzer, and the power consumption is reported by PowerMill. The functionality of both SRSDF and R2^{2}SDFII are verified by the post-layout simulation done by TimeMill. The SRSDF FFT pipeline can achieve a higher clock rate because of the well balance of the multiplication and the butterfly operation. The novel SRSDF FFT architecture achieves about 15% power saving and 14.5% speed improvement compared with the R2^{2}SDFII FFT pipeline. The power consumption for SRSDF or R2^{2}SDFII does agree with the model predicted by (1). For example, the power consumed by the SRSDF is 259 mW when supply voltage is reduced to 2.7 V. If we take the area into consideration, then the power reduction contributed by the delay-balanced pipeline and the split-radix algorithm is about 10%. However, when N is 64, the number of multiplication for the two designs differs by four only. It means that most of the 10% power reduction should come from the delay-balanced design. The strength of the split-radix algorithm will not be significant when N is small.

VI. Conclusion and Future Work

This paper presents a novel delay-balanced SRSDF pipeline architecture, which is regular and extensible for any 2^n-point FFT. Most of the conventional radix-r FFT pipeline has the restriction that the length of FFT has to be power of r. We remove such restriction by using split-radix algorithm. Compared with the R2^{2}SDFII design, it saves 15% power consumption and 6% hardware cost and reduces the critical path by 14.5%, according to the post-layout simulation based on the 0.35 μm Avant! cell library [23]. Thus, it does not only achieve the minimum hardware requirement but also saves the power and increases the maximum clock rate at the same time.

The 1-D linear array for the other FFT algorithms can be obtained via similar mapping procedure, and the delay-balanced pipeline architecture can also be used when higher clock rate and lower power consumption are desirable. The comparison of the fixed-radix and mixed-radix algorithms also provides useful information for a designer.

For the radix-2/8 algorithm, we also propose a R28SDF pipeline architecture in this work. It has low computational complexity but a high hardware cost as well. We will develop a cost-effective solution for R28SDF architecture in the future.

As mentioned in Section III-B, the number of global memory access for low radix algorithms can be reduced by using high radix pipeline structure. This is also considered as our future research direction because it may save significant power for OFDM systems with long length FFT.

Acknowledgment

The authors would like to acknowledge the valuable suggestions from the reviewers and the associate editor.

References

[1] N. Weste and D. J. Skellern, "VLSI for OFDM," *IEEE Commun. Mag.*, vol. 36, pp. 127–131, Oct. 1998.
[2] R. van Nee and R. Prasad, *OFDM for Wireless Multimedia Communications*. Norwell, MA: Archtech House, 2000.
[3] M. Engels, W. Eberle, and B. Gyselinckx, "Design of a 100 Mbps wireless local area network," in *Proc. IEEE URSI Int. Symp. Signals, Syst., Electron.*, 1998, pp. 253–256.
[4] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 5, no. 5, pp. 87–109, 1965.
[5] P. Duhamel, "Algorithms meeting the lower bounds on the multiplicative complexity of length- 2^n DFT's and their connection with practical algorithms," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, Sept. 1990.
[6] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)Modulation," in *Proc. IEEE URSI Int. Symp. Signals, Syst., Electron.*, 1998, pp. 257–262.
[7] L. Jia, Y. Gao, J. Isoaho, and H. Tenhunen, "A new VLSI-oriented FFT algorithm and implementation," *Proc. Eleventh Annu. IEEE Int. ASIC Conf.*, pp. 337–341, 1998.
[8] M. A. Richards, "On hardware implementation of split-radix FFT," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, pp. 1575–1581, Oct. 1988.
[9] J. García, J. A. Michell, and A. M. Burón, "VLSI configurable delay commutator for a pipeline split radix FFT architecture," *IEEE Trans. Signal Processing*, vol. 47, pp. 3098–3107, Nov. 1999.
[10] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.
[11] E. H. Wold and A. M. Despain, "Pipeline and parallel pipeline FFT processors for VLSI implementation," *IEEE Trans. Comput.*, vol. C-33, pp. 414–426, May 1984.
[12] A. M. Despain, "Fourier transform computer using CORDIC iterations," *IEEE Trans. Comput.*, vol. C-23, pp. 993–1001, Oct. 1974.
[13] G. Bi and E. V. Jones, "A pipelined FFT processor for word-sequential data," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, pp. 1982–1985, Dec. 1989.
[14] K. K. Parhi, *VLSI Digital Signal Processing Systems: Design and Implementation*. New York: Wiley, 1999, ch. 7, p. 189.
[15] B. W. Y. Wei, H. Du, and H. Chen, "A complex-number multiplier using radix-4 digits," *Proc. IEEE 12th Symp. Comput. Arithmetic*, pp. 84–90, 1995.
[16] A. D. Booth, "A signed binary multiplication technique," *Quart. J. Mechan. Appl. Math.*, vol. 4, pp. 236–240, 1951.
[17] O. L. MacSorley, "High speed arithmetic in binary computers," *Proc. IRE*, vol. 49, pp. 67–91, 1961.
[18] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Comput.*, vol. C–13, pp. 14–17, Feb. 1964.
[19] L. Dadda, "Some schemes for parallel multiplier," *Alta Frequenza*, vol. 34, pp. 349–356, Mar. 1965.
[20] P. F. Stelling, C. U. Martel, V. G. Oklobdzija, and R. Ravi, "Optimal circuits for parallel multipliers," *IEEE Trans. Comput.*, vol. 47, pp. 273–285, Mar. 1998.
[21] W.-C. Yeh and C.-W. Jen, "A high-speed booth encoded parallel multiplier design," *IEEE Trans. Comput.*, vol. 49, pp. 692–701, July 2000.
[22] K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*. New York: Wiley, 1976, ch. 3, p. 81.
[23] "Passport 0.35 Micron, 3.3 Volt," Avant! Corp., Optimum Silicon SC Library, CB35OS142, 1998.

Wen-Chang Yeh was born in Hsinchu, Taiwan, R.O.C., in 1975. He received the B.S. and Ph.D. degrees in electronics engineering from National Chiao Tung University, Hsinchu, in 1997 and 2001, respectively.

He is now working for ZyDAS Technology Corporation, Hsinchu, Taiwan, R.O.C. His current research interests include computer arithmetic, digital signal processing for communication system, computer architecture, and system-level design.

Chein-Wei Jen received the B.S. degree from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1970, the M.S. degree from Stanford University, Stanford, CA, in 1977, and the Ph.D. degree from National Chiao Tung University in 1983.

He is currently with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University as a Professor. From 1985 to 1986, he was with the University of Southern California, Los Angeles, as a Visiting Researcher. His current research interests include VLSI design, digital signal processing, processor architecture, and design automation. He has held six patents and published over 40 journal papers and 90 conference papers in these areas.