Scaled Radix-2/8 Algorithm for Efficient Computation of Length-$N = 2^m$ DFTs

Weihua Zheng, Kenli Li, and Keqin Li, *Senior Member, IEEE*

***Abstract*—This paper presents a scaled radix-2/8 fast Fourier transform (FFT) (SR28FFT) algorithm for computing length-$N = 2^m$ discrete Fourier transforms (DFTs) scaled by complex number rotating factors. The idea of the SR28FFT algorithm is from the modified split radix FFT (MSRFFT) algorithm, and its purpose is to furnish other algorithms with high efficiency but without shortcomings of the MSRFFT algorithm. A novel radix-2/4 FFT (NR24FFT) algorithm and a novel radix-2/8 FFT (NR28FFT) algorithm are proposed. These two algorithms use SR28FFT to calculate their sub-DFTs of odd-indexed terms. Several aspects of the two algorithms such as computational complexity, computation accuracy, and coefficient evaluations or accesses to the lookup table all are improved. The bit-reverse method can be used for their order permutation and no extra memory is required to store their extra coefficients by the two novel algorithms, which contribute significantly to the performance of the FFT algorithms. The SR28FFT algorithm can also be applied to other algorithms whose decomposition contains sub-DFTs of powers-of-two. The Appendix presents an algorithm named SR28FFT-2 for further reducing the number of arithmetic operations, and NR24FFT and NR28FFT algorithms based on SR28FFT-2 requires fewer real operations than that required by the MSRFFT algorithm.**

***Index Terms*—Fast Fourier Transform (FFT), radix-2/4, radix-2/8, scaled radix-2/8.**

I. Introduction

FOR many years, the time to perform an FFT algorithm was dominated by real number arithmetic, and considerable effort was devoted towards proving and achieving lower bounds on the exact count of arithmetic operations (real additions and multiplications), herein called "flops" (floating-point operations), required by a discrete Fourier transform (DFT) of a given size [1]–[6]. It has been shown in the literature that direct use of the Cooley-Tukey [7] approach to developing a higher radix FFT algorithm leads to a reduction in the arithmetic complexity compared to that of the lower radix FFT algorithm. This can be seen by comparing, for example, the radix-2, radix-4, radix-8, and radix-16 FFT algorithms. Since programs with regular structure are generally more compact and have faster implementation, for powers-of-two DFTs, one often uses recursively the same decomposition in each stage, thus loading full radix-2 or radix-4 program. When the DFT length is not a power of radix, a smaller radix is used towards the end of the composition. This is so-called the mixed-radix method.

In 1968, Yavne [8] proposed an algorithm that was subsequently rediscovered simultaneously by various authors [9]–[11] in 1984 and became known as the "split radix" FFT (SRFFT) algorithm, i.e., the radix-2/4 FFT algorithm, for $N = 2^m$, and achieved a record flop count of $4N \lg N - 6N + 8$ for $N > 1$, an improvement by 20% over the classic "radix-2" algorithm (flops $\sim 5N \lg N$) [7]. The papers [12]–[14] proposed a class of split-radix FFT algorithms, which have the same arithmetic complexity as the radix-2/4 FFT algorithm. This class of algorithms are the best compromise between structure and computational complexity. The radix-2/4 FFT algorithm and the radix-2/8 FFT algorithm [15], [16] are two well known algorithms in this class of algorithms. In 2000, Grigoryan and Agaian [17] presented an algorithm which reduces the number of operations for DFTs of lengths longer than 256 at expense of a more complicated structure compared with the radix-2/4 FFT algorithm; however, it is computationally less efficient for shorter lengths. In 2007, Johnson and Frigo presented a modified version of the split-radix FFT (MSRFFT) that reduces the flop count by a further ~5.6% (1/18) compared to the standard SRFFT, i.e., MSRFFT reduces the order of magnitude of operations from $4N \lg N$ to $34N \lg N/9$.

The situation is more complicated since much more parameters have to be taken into consideration than the sole operations count [18]–[20]. The numerical accuracy, the size of memory, and twiddle factor evaluations [21] or accesses to the lookup table are also the main aspects of performance of FFT algorithms. It is seen that the following four aspects will influence the performance of the MSRFFT algorithm.

- MSRFFT requires more memory to store its coefficients if coefficients are pre-computed and stored in a lookup table in advance.
- Computing a DFT with the MSRFFT algorithm requires arithmetic operations more than computing the same length DFT with the SRFFT algorithm, if coefficients are directly computed [22], [23] rather than loaded from the lookup table.
- MSRFFT is not suitable for using the standard bit-reverse method [24] to permute its order, since it is an

Manuscript received August 28, 2013; revised January 06, 2014; accepted February 24, 2014. Date of publication March 11, 2014; date of current version April 21, 2014. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Zhiyuan Yan. This work was supported in part by the National Natural Science Foundation of China (Grant Nos. 61133005, 61173045, 61370098, 61370095), and by the project of the National Science Foundation for Distinguished Young Scholars of Hunan (12JJ1011) *(Corresponding author: Kenli Li.)*

W. Zheng and Kenli Li are with College of Information Science and Engineering, Hunan University, Changsha 410082, China (e-mail: zhengdavid@hnu.edu.cn; lkl@hnu.edu.cn).

Keqin Li is with the College of Information Science and Engineering, Hunan University, Changsha 410082, China, and also with the Department of Computer Science, State University of New York, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TSP.2014.2310434

algorithm based on conjugate pair SRFFT algorithm [25], [26]. MSRFFT's order permutation requires quite a few extra memory and memory access for its software implementation and more complex circuits for its hardware implementation.

- MSRFFT has errors within 10% of the standard conjugate-pair split-radix algorithm [27], since the use of the tangent function, which is singular, or equivalently the division by a cosine, raises questions about the numerical accuracy.

The purpose of this paper is to develop algorithms for efficiently computing length-$N-2^m$ DFTs. An efficiently auxiliary algorithm, the scaled radix-2/8 FFT (SR28FFT) algorithm, is developed for efficiently computing length-$N = 2^m$ sub-DFTs scaled by complex number rotating factors. The SR28FFT algorithm decomposes a DFT according to the standard radix-2/8 FFT algorithm, and extracts a common factor 1 or w_8^1 from the next recursion for each butterfly. Owing to no real operations and tangent function required by multiplications and divisions of rotating factor w_N^n and common factor 1 or w_8^1, no extra memory is required to store extra coefficients, and numerical accuracy is also not an issue. The MSRFFT algorithm is based on conjugate pair SRFFT algorithm. Its inputs (or outputs) of one of two odd-indexed sub-DFTs has to be rotated forward 1 position in each recursive stage. The special rotation results in that the algorithm cannot utilize the standard bit-reverse method to permute its order. In contrast, the SR28FFT algorithms are based on the standard radix-2/8 FFT algorithm. Its inputs/outputs do not need to perform any extra rotation. Therefore, the bit-reverse method can be used to permute its order. The SR28FFT algorithm has been used in the radix-2/4 FFT and the radix-2/8 FFT, i.e., a novel radix-2/4 FFT (NSR24FFT) algorithm and a novel radix-2/8 FFT (NSR28FFT) algorithm are proposed. These two algorithms overcome the shortcomings of the MSRFFT algorithm and achieve the desired improvements.

The rest of this paper is organized as follows. Section II describes the proposed SR28FFT algorithm. Section III discusses two novel algorithms, the novel radix-2/4 FFT algorithm and the novel radix-2/8 FFT algorithm. Section IV analyzes performance of algorithms. Section V evaluates computation accuracy by computing L_2 relative error. Section VI draws the conclusion. The Appendix presents an algorithm based on the SR28FFT algorithm, considering only the improvement in arithmetic operations and neglecting other performance.

II. Scaled Radix-2/8 FFT Algorithm

Given a length-$N = 2^m$ sequence $x(n)$, its DFT is also a length-N sequence defined by

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

where $w_N = e^{-j2\pi/N}$ and $j = \sqrt{-1}$.

A. Proposed Scaled Radix-2/8 FFT Algorithm

In this subsection, we propose an algorithm based on the standard radix-2/8 FFT algorithm for scaled DFTs of size-$N = 2^m$, which is named SR28FFT algorithm. The algorithm can be used in the radix-2/4 FFT algorithm, the radix-2/8 FFT algorithm, etc.

The idea of SR28FFT is based on the fact that a reduction of operations can be obtained for $N \geq 16$ if a length-$N = 2^m$ DFT defined in (1) is a sub-DFT with a rotating factor and a factor $s(N, n)$ is extracted from the DFT. The factor $s(N, n)$ can be attached to the rotating factor of sub-DFTs to combine a new rotating factor without any floating-point operation if the rotating factor is a complex number. The factor $s(N, n)$ is defined by

$$s(N, n) = \begin{cases} 1, & 0 < (n \bmod N/4) < N/8; \\ w_8^1, & N/8 \leq (n \bmod N/4). \end{cases} \tag{2}$$

The DFT in (1) can be re-expressed by the following equation in which the factor $s(N, n)$ is contained:

$$X(k) = \sum_{n=0}^{N-1} \frac{y(n)}{s(N, n)} w_N^{nk}, \quad k = 0, 1, \ldots, N-1, \tag{3}$$

where

$$y(n) = x(n) s(N, n). \tag{4}$$

The DFT in (3) is necessary only for a sub-DFT or a DFT with a complex rotating factor. If the sub-DFT or the DFT does not have a complex rotating factor, the operations involved in the factor $s(N, n)$ become redundant and unnecessary.

Now, we discuss the decomposition of the DFT in (3). The following decompositions are provided so that the DFT of (3) can be computed more efficiently:

$$X(2k) = \sum_{n=0}^{N/2-1} b(n)/s(N, n) w_{\frac{N}{2}}^{nk}, \; k = 0, 1, \ldots, (N/2-1), \tag{5}$$

for the even-indexed terms, and

$$\begin{aligned} &X(8k+1) \\ &\quad = \sum_{n=0}^{N/8-1} \left(b_e(n) + b_e(N/8+n)\right) w_N^n s(N/8, n) w_{\frac{N}{8}}^{nk}, \\ &\qquad k = 0, 1, \ldots, (N/8-1); \end{aligned} \tag{6}$$

$$\begin{aligned} &X(8k+5) \\ &\quad = \sum_{n=0}^{N/8-1} \left(b_e(n) - b_e(N/8+n)\right) w_N^{5n} s(N/8, n) w_{\frac{N}{8}}^{nk}, \\ &\qquad k = 0, 1, \ldots, (N/8-1); \end{aligned} \tag{7}$$

$$\begin{aligned} &X(8k+3) \\ &\quad = \sum_{n=0}^{N/8-1} \left(b_o(n) - j b_o(N/8+n)\right) w_N^{3n} s(N/8, n) w_{\frac{N}{8}}^{nk}, \\ &\qquad k = 0, 1, \ldots, (N/8-1); \end{aligned} \tag{8}$$

$$\begin{aligned} &X(8k+7) \\ &\quad = \sum_{n=0}^{N/8-1} \left(b_o(n) + j b_o(N/8+n)\right) w_N^{7n} s(N/8, n) w_{\frac{N}{8}}^{nk}, \\ &\qquad k = 0, 1, \ldots, (N/8-1); \end{aligned} \tag{9}$$

for the odd-indexed terms, where $b(n) = x(n) + x(N/2+n)$. The sequence $b_e(n)$ in (6) and (7) and $b_o(n)$ in (8) and (9) can be expressed in a matrix form

$$\begin{bmatrix} b_e(n) \\ b_o(n) \end{bmatrix} = \mathbf{H_2} \begin{bmatrix} u(n) \\ -ju(N/4+n) \end{bmatrix}, \tag{10}$$

where $\mathbf{H_2}$ is a second-order Hadamard matrix defined by

$$\mathbf{H_2} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{11}$$

The sequence $b(n)$ in (5) and $u(n)$ in (10) can be expressed in a matrix form

$$\begin{bmatrix} b(n) \\ u(n) \end{bmatrix} = \mathbf{H_2} \begin{bmatrix} x(n) \\ x(N/2+n) \end{bmatrix}. \tag{12}$$

No matter whether the value of the factor $s(N/8, n)$ in (6)–(9) is 1 or w_8^1, the product of the factor $s(N/8, n)$ and w_N^n, w_N^{3n}, w_N^{5n}, or w_N^{7n} does not require extra real operations and can directly map to w_N^η, where $0 \le \eta < N/4$. The sub-DFT in (5) is a DFT scaled by $1/s(N, n)$. Straightforward computation of the sub-DFT will result in the same number of operations required by the standard radix 2/8 FFT algorithm. To reduce the number of operations, the four length-$N/8$ sub-DFTs will be of course decomposed recursively using (5)–(9). Besides that, it is necessary for the length-$N/2$ sub-DFT in (5) to be further decomposed with a appropriate strategy. The effective decompositions of the length-$N/2$ sub-DFT in (5) provide

$$X(4k) = \sum_{n=0}^{N/4-1} c(n)s(N/4, n)/s(N, n)w_{\frac{N}{4}}^{nk}, \quad k = 0, 1, \ldots, (N/4-1), \tag{13}$$

for the even-indexed terms,

$$X(16k+2) = \sum_{n=0}^{N/16-1} (c_e(n) + c_e(N/16+n))\, w_N^{2n} s(N/16, n) w_{\frac{N}{16}}^{nk}, \quad k = 0, 1, \ldots, (N/16-1); \tag{14}$$

$$X(16k+10) = \sum_{n=0}^{N/16-1} (c_e(n) - c_e(N/16+n))\, w_N^{10n} s(N/16, n) w_{\frac{N}{16}}^{nk}, \quad k = 0, 1, \ldots, (N/16-1); \tag{15}$$

$$X(16k+6) = \sum_{n=0}^{N/16-1} (c_o(n) - jc_o(N/16+n))\, w_N^{6n} s(N/16, n) w_{\frac{N}{16}}^{nk}, \quad k = 0, 1, \ldots, (N/16-1); \tag{16}$$

$$X(16k+14) = \sum_{n=0}^{N/16-1} (c_o(n) + jc_o(N/16+n))\, w_N^{14n} s(N/16, n) w_{\frac{N}{16}}^{nk}, \quad k = 0, 1, \ldots, (N/16-1); \tag{17}$$

for the odd-indexed terms, where $c(n) = b(n) + b(N/4+n)$, and

$$\begin{bmatrix} c_e(n) \\ c_e(N/16+n) \\ c_o(n) \\ c_o(N/16+n) \end{bmatrix} = (\mathbf{S_2} \bigotimes \mathbf{I_2}) \mathbf{H_4} \begin{bmatrix} v(n) \\ v(N/16+n) \\ v(N/8+n) \\ v(3N/16+n) \end{bmatrix}, \tag{18}$$

with

$$\mathbf{S_2} = \begin{bmatrix} \frac{s(N/2,n)}{s(N,n)} & 0 \\ 0 & \frac{s(N/2,N/16+n)}{s(N,N/16+n)} \end{bmatrix}, \tag{19}$$

and

$$\mathbf{H_4} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & -1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}. \tag{20}$$

The $c(n)$ in (13) and $v(n)$ in (18) can be expressed in the following matrix form

$$\begin{bmatrix} c(n) \\ v(n) \end{bmatrix} = \mathbf{H_2} \begin{bmatrix} b(n) \\ b(N/4+n) \end{bmatrix}. \tag{21}$$

The length-$N/4$ DFT in (13) and the four length-$N/16$ sub-DFTs in (14)–(17) will be decomposed recursively using (5)–(9).

We now summarize the proposed scheme for the DFT in (3). The DFT is first decomposed into five smaller sub-DFTs defined in (5)–(9). Then, four sub-DFTs in (6)–(9) are decomposed recursively according to (5)–(9), and the length-$N/2$ sub-DFT in (5) is decomposed recursively according to (13)–(17). All sub-DFTs in (13)–(17) are decomposed according to (5)–(9). Fig. 2 shows signal flowgraph of 32-points DFTs. For clarity, the special butterfly when $n = 0$ (including $n = N/16$) and the operations of input terms involved in $s(32, n)$ are omitted. Fig. 1 shows flowgraphs of the general butterfly. Considering computation involved in $s(N, n)$ in (4), it is the best for the SR28FFT algorithm to be used in DFTs or sub-DFTs which are scaled by complex number rotating factors, so DFTs' scaling factors and the factors $s(N, n)$ can be combined into new rotating factors without any floating-point operation. In Section III, we will discuss two algorithms which use the SR28FFT algorithm for computing their sub-DFTs of odd-indexed terms. The first one is a novel radix-2/4 FFT algorithm, and the second one is a novel radix-2/8 FFT algorithm.

B. Special Case When $n = 0$ and $n = N/32$

A further improvement of arithmetic complexity can be obtained if the special case when $n = N/16$ is taken into account. Considering the special case when $n = N/16$, a new extracted factor is defined by

$$s'(N, n) = \begin{cases} 1, & 0 \le (n \bmod N/4) < N/8, n \ne N/16; \\ \cos(2\pi/16), & (n \bmod N/4) = N/16, 3N/16; \\ w_8^1, & N/8 \le (n \bmod N/4), n \ne 3N/16. \end{cases} \tag{22}$$

For saving real multiplications, all $s(N, n)$ in (5)–(9) and (14)–(17) are replaced by $s'(N, n)$. The expression $s(N, n)/s(2N, n)$ in (19) will be replaced by $s(N, n)/s'(2N, n)$. When $n = N/16$, the rotating factors in

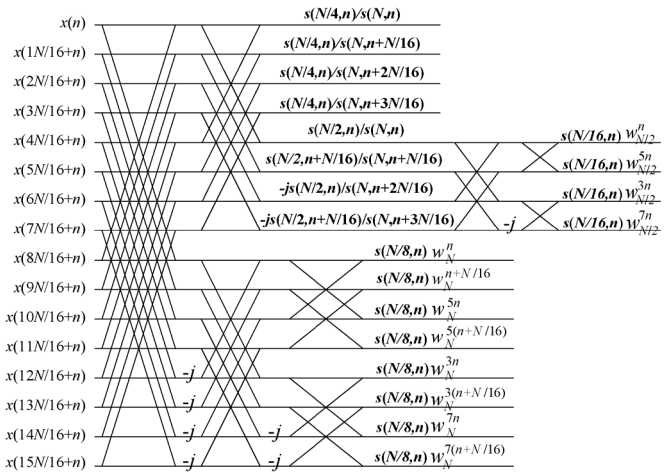

Fig. 1. General butterfly of proposed SR28FFT algorithm.

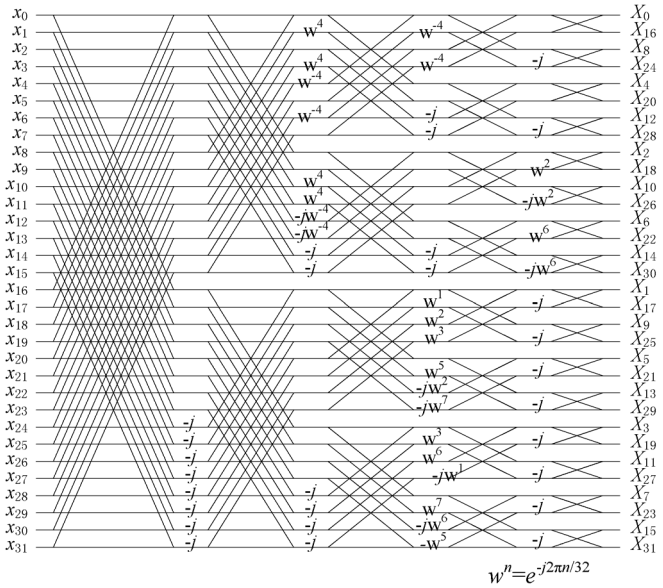

Fig. 2. Signal flowgraph of 32-points DFT of proposed SR28FFT algorithm.

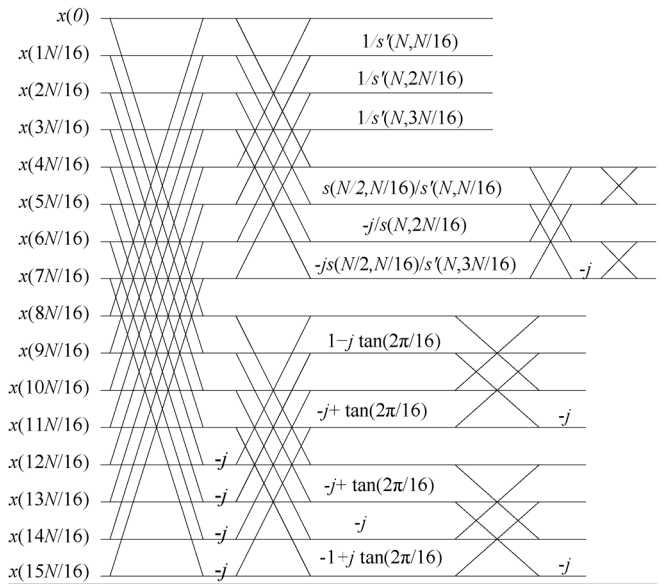

Fig. 3. Special butterfly when $n = 0$ of proposed SR28FFT algorithm.

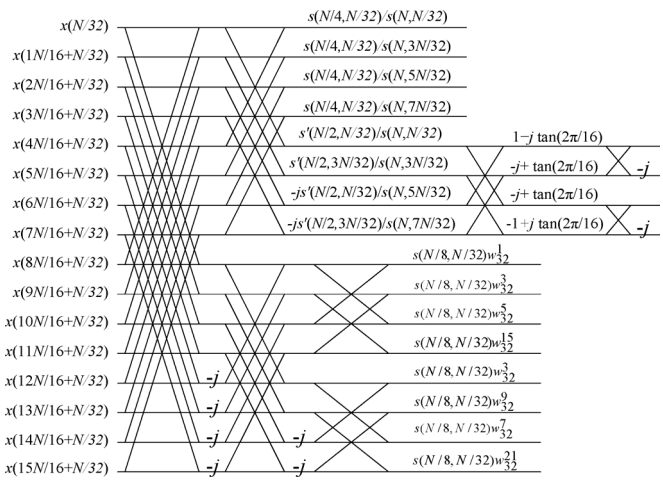

Fig. 4. Special butterfly when $n = N/32$ of proposed SR28FFT algorithm.

(6)–(9) will be distributed into the left brackets and attached to the original coefficients. The new four rotating factors are $[\pm j]w_{16}^{\pm 1}$, where $[\pm j]$ is an alternative. Therefore, the common factor $\cos(2\pi/16)$ can be extracted. When $n = N/32$, for (14)–(17), the rotating factors are $w_{16}^{1,3,5,\ and\ 7}$ can be distributed into the left brackets, and the factor $\cos(2\pi/16)$ from the new rotating factors is extracted and combined with $s(N, N/32)$. The extracted factor from the sub-DFT in (13) is $s(N, n)$ rather than $s'(N, n)$, i.e., the expression involved in $s(N, n)$ and $s'(N, n)$ in (13) is $s(N/2, n)/s'(2N, n)$. Figs. 3 and 4 show flowgraphs of the general butterfly and the two special butterflies when $n = 0$ and $n = N/32$ for the SR28FFT algorithm. In Fig. 4, $s(N/8, N/32) = 1$, $s(N/4, N/32) = w_8^1$, and $s'(N/2, N/32) = \cos(2\pi/16)$.

III. Two Novel FFT Algorithms

The proposed SR28FFT algorithm can be used by the radix-2/4 FFT algorithm and the radix-2/8 FFT algorithm for efficiently computing their sub-DFTs of odd-indexed terms. Besides the radix-2/4 FFT algorithm and the radix-2/8 FFT algorithm, many other algorithms such as radix-2, radix-4, radix-8, etc can also use the SR28FFT algorithm to reduce their operations count. In this section, we present two algorithms, a novel radix-2/4 FFT (NR24FFT) algorithm and a novel radix-2/8 FFT (NR28FFT) algorithm, in which the SR28FFT algorithm is used to reduce their arithmetic complexities.

A. Novel Radix-2/4 FFT Algorithm

With the NR24FFT algorithm, the DFT in (1) is decomposed decimation-in-frequency (the decimation-in-time decomposition is similar to the decimation-in-frequency decomposition). When $N \geq 4$, the DFT in (1) provides the following decompositions,

$$X(2k) = \sum_{n=0}^{N/2-1} u(n) w_{\frac{N}{2}}^{nk}, \quad k = 0, 1, \ldots, (N/2-1), \quad (23)$$

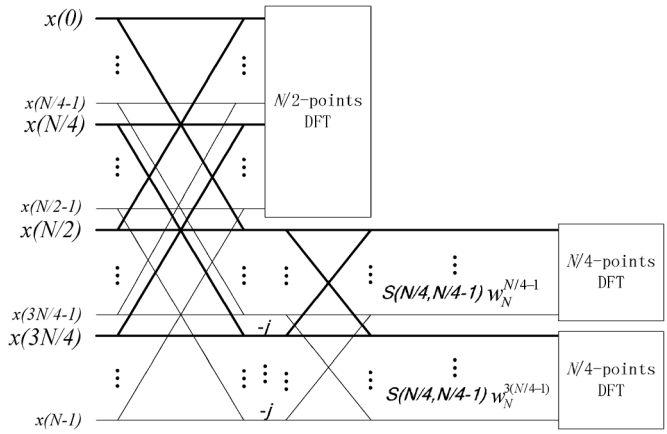

Fig. 5. First stage of proposed NR24FFT algorithm.

for the even-indexed terms,

$$X(4k+1) = \sum_{n=0}^{N/4-1} a_e(n) w_N^n s(N/4,n) w_{\frac{N}{4}}^{nk}, \quad k = 0,1,\ldots,(N/4-1); \quad (24)$$

and

$$X(4k+3) = \sum_{n=0}^{N/4-1} a_o(n) w_N^{3n} s(N/4,n) w_{\frac{N}{4}}^{nk}, \quad k = 0,1,\ldots,(N/4-1); \quad (25)$$

for the odd-indexed terms. The sequence $u(n)$ in (23), $a_e(n)$ in (24), and $a_o(n)$ in (25) can be expressed in a matrix form

$$\begin{bmatrix} u(n) \\ v(n) \end{bmatrix} = \mathbf{H_2} \begin{bmatrix} x(n) \\ x(N/2+n) \end{bmatrix}, \quad (26)$$

and

$$\begin{bmatrix} a_e(n) \\ a_o(n) \end{bmatrix} = \mathbf{H_2} \begin{bmatrix} v(n) \\ -jv(N/2+n) \end{bmatrix}. \quad (27)$$

A length-$N = 2^m$ DFT is thus decomposed into a length-$N/2$ DFT given by (23) and two length-$N/4$ DFTs given by (24) and (25). The length-$N/2$ sub-DFT in (23) will be decomposed recursively using (23)–(25) till the size is reduced to a 2-points DFT. The two length-$N/4$ sub-DFTs in (24) and (25) are decomposed and implemented recursively with the SR28FFT algorithm using (5)–(9). Fig. 5 shows the decomposition in the first stage for the NR24FFT algorithm.

B. Novel Radix-2/8 FFT Algorithm

In this subsection, we present a novel radix-2/8 FFT (NS28FFT) algorithm for length-$N = 2^m$ DFTs. Like the NS24FFT algorithm, the NS28FFT algorithm also uses SR28FFT to reduce the number of operations. When $N \geq 8$, the NR28FFT algorithm provides the following decompositions so that the DFT in (1) can be computed more efficiently than the standard radix-2/8 FFT algorithm.

$$X(2k) = \sum_{n=0}^{N/2-1} u(n) w_{\frac{N}{2}}^{nk}, \; k = 0,1,\ldots,(N/2-1), \quad (28)$$

for the even-indexed terms,

$$X(8k+1) = \sum_{n=0}^{N/8-1} \left(a_e(n) + w_8^1 a_e(N/8+n)\right) s(N/8,n) w_N^n w_{\frac{N}{8}}^{nk}, \quad k = 0,1,\ldots,(N/8-1); \quad (29)$$

$$X(8k+5) = \sum_{n=0}^{N/8-1} \left(a_e(n) - w_8^1 a_e(N/8+n)\right) s(N/8,n) w_N^{5n} w_{\frac{N}{8}}^{nk}, \quad k = 0,1,\ldots,(N/8-1); \quad (30)$$

$$X(8k+3) = \sum_{n=0}^{N/8-1} \left(a_o(n) + w_8^3 a_o(N/8+n)\right) s(N/8,n) w_N^{3n} w_{\frac{N}{8}}^{nk}, \quad k = 0,1,\ldots,(N/8-1); \quad (31)$$

$$X(8k+7) = \sum_{n=0}^{N/8-1} \left(a_o(n) - w_8^3 a_o(N/8+n)\right) s(N/8,n) w_N^{7n} w_{\frac{N}{8}}^{nk}, \quad k = 0,1,\ldots,(N/8-1); \quad (32)$$

for the odd-indexed terms. The sequences a_e and a_o in (29)–(32) have been defined in (27). The sequences $u(n)$ in (28) and $v(n)$ in (27) have been defined in (26). The length-$N/2$ sub-DFT in (28) is decomposed using (28)–(32) until the size is reduced to a 4-, 2-, or 1-points DFT. The four length-$N/8$ sub-DFTs in (29)–(32) are computed with the SR28FFT algorithm, and are decomposed using (5)–(9). Fig. 6 shows the decomposition of the first stage of the NR28FFT algorithm.

We now summarize the proposed NR24FFT algorithm and the proposed NR28FFT algorithm. The DFT in (1) is first decomposed according to the standard radix-2/4 FFT algorithm or the standard radix 2/8 FFT algorithm, then the sub-DFTs of odd-indexed terms are computed with the SR28FFT algorithm, and sub-DFT of even-indexed terms is evaluated recursively with the algorithms themselves. By using the SR28FFT algorithm, more efficient computation is obtained compared with the standard radix-2/4 FFT algorithm and the standard radix-2/8 FFT algorithm.

IV. Performance Analysis

In this section, we consider the performance of the proposed algorithms by analyzing and comparing them with the existing algorithms such as the standard radix-2/4, radix-2/8, radix-8, and etc [9], [16]. The analysis and comparison includes not only the arithmetic operations but also the coefficient evaluations or accesses to the lookup table, since it is also a main aspect of performances of FFT algorithms.

A. Butterfly Analysis

The SR28FFT algorithm has three classes of butterflies, a general butterfly and two special butterflies when $n = 0$ and $n = N/32$, as illustrated in Figs. 1, 3, and 4. Since the SR28FFT algorithm is based on the standard radix-2/8 FFT algorithm, the arithmetic operations of these three classes of butterflies can compare with those required by the radix-2/8 FFT algorithm.

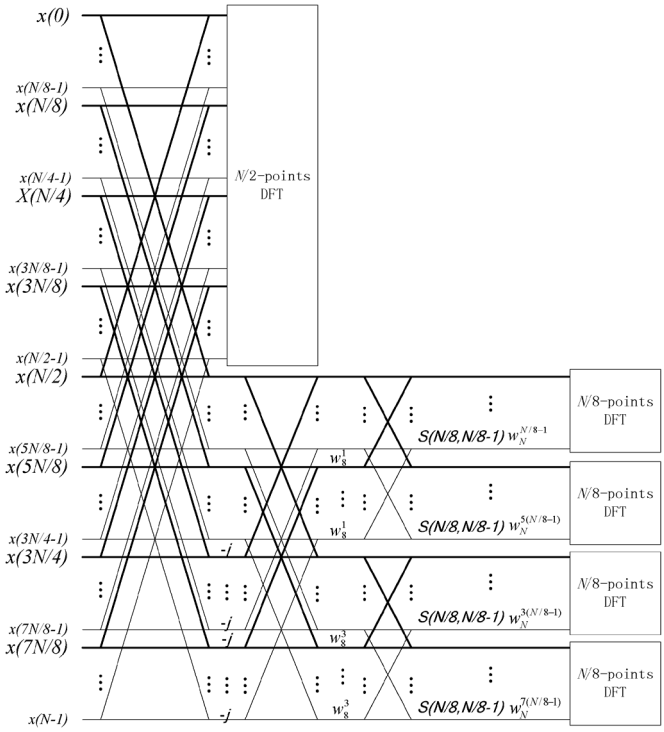

Fig. 6. First stage of proposed NR28FFT algorithm.

The analysis of the three classes of butterflies are listed as follows.

1) The general butterfly of the proposed SR28FFT algorithm, as illustrated in Fig. 1, requires 4 complex multiplications by w_8^1. To finish the same transformation, the standard radix-2/8 FFT algorithm requires 6 complex multiplications by w_8^1 or w_8^3. Since the proposed SR28FFT algorithm just changes the multiplications by w_8^1 or w_8^3 and other operations required by the SR28FFT algorithm is the same as those required by the standard radix-2/8 FFT algorithm in the general butterfly, two complex multiplications by w_8^1 or w_8^3 are saved by the SR28FFT algorithm, i.e., a reduction of 4 real multiplications and 4 real additions is achieved by the general butterfly.
2) For the special butterfly when $n = 0$ of the SR28FFT algorithm, as illustrated in Fig. 3, all complex multiplications are implemented with 16 real additions and 20 real multiplications. In order to implement the same works, the standard radix-2/8 requires 16 real additions and 24 real multiplications. It can been see that this special butterfly achieves a reduction of 4 real multiplications.
3) The special butterfly when $n = N/32$, as illustrated in Fig. 4, all complex multiplications are implemented with 32 real additions and 52 real multiplications. In order to implement the same works, the standard radix-2/8 requires 32 real additions and 56 real multiplications. Obviously, the special butterfly also achieves a reduction of 4 real multiplications.

It is seen that a reduction of operations is easily obtained by the three classes of butterflies of the SR28FFT algorithm.

B. Arithmetic Complexity

For a FFT processor that does not adopt dynamic voltage and frequency (DVS) components [28], if arithmetic units cannot be reduced and the number of operations of algorithms is reduced, no benefits can be gained. So, a reduction of operations caused by the special butterflies when $n = 0$ and when $n = N/32$ is useless for the FFT processor [29], [30]. The one-butterfly implementation and the three-butterfly implementation have different advantages and disadvantages for the software and hardware implementation. To make a complete comparison, we will count the number of real operations of the proposed SR28FFT algorithm respectively for the one-butterfly implementation and the three-butterfly implementation.

Let A_N and M_N, respectively, be the number of real additions and the number of multiplications required by the SR28FFT algorithm for a length-N DFT. Suppose that a butterfly-unit of two-to-two (BU-2) is an arithmetic unit with two inputs of complex number, two outputs of complex number, one complex addition, and one complex subtraction. A BU-2 is implemented with 4 real additions. The proposed decomposition is achieved by three classes of butterflies that are illustrated in Figs. 1, 3, and 4. Every butterfly requires 24 BUs-2 and some complex multiplications. The complex multiplications of the general butterfly contain 32 real additions and 56 real multiplications. The complex multiplications of the special butterfly when $n = 0$ contain 16 real additions and 20 real multiplications. The complex multiplications of the special butterfly when $n = N/32$ contain 32 real additions and 52 real multiplications. The proposed decomposition consists of dividing a length-N DFT to one length-$N/4$ DFT, four length-$N/8$ sub-DFTs, and four length-$N/16$ sub-DFTs in the first stage. This is achieved by performing $N/16 - 2$ general butterflies and the 2 special butterflies, i.e., the special case when $n = 0$ and the special case when $n = N/32$. The decomposition process is repeated successively for each of the resulting DFTs until the size is reduced to a 8-, 4-, 2-, or 1-point DFT. Therefore, it is seen that the expressions for the number of real multiplications and real additions of the one-butterfly implementation of the proposed FFT algorithm are, respectively,

$$\begin{cases} A_N = A_{\frac{N}{4}} + 4A_{\frac{N}{8}} + 4A_{\frac{N}{16}} + 8N, & N \geq 16; \\ M_N = M_{\frac{N}{4}} + 4M_{\frac{N}{8}} + 4M_{\frac{N}{16}} + 7N/2, & N \geq 16. \end{cases} \tag{33}$$

For small lengths $N = 1$, 2, 4, and 8 DFTs, the SR28FFT algorithm require 0, 4, 16, and 52 real additions, and 0, 0, 0, and 4 real multiplications for implementing them. The arithmetic complexity of the one-butterfly implementation is given in Table I. We can see from this table that the number of multiplications of the one-butterfly implementation required by the SR28FFT algorithm are less than that required by the standard radix-2/4 FFT algorithm and the standard radix-2/8 FFT algorithm, and the required total flops (floating-point operations) is also less than that required by the two classical algorithms.

It is seen that the number of real additions and real multiplications of the three-butterfly implementation of the SR28FFT are, respectively,

TABLE I
COMPARISON OF ARITHMETIC COMPLEXITY OF ONE-BUTTERFLY

N	Radix-2/4			Radix-2/8			SR28FFT		
	Addition	Multiplication	Total	Addition	Multiplication	Total	Addition	Multiplication	Total
16	174	92	266	178	88	266	178	88	266
32	434	228	662	442	216	658	434	208	642
64	1038	540	1578	1058	504	1562	1042	488	1530
128	2418	1252	3670	2474	1176	3650	2434	1136	3570
256	5518	2844	8362	5650	2680	8330	5538	2568	8106
512	12402	6372	18774	12698	5976	18674	12434	5712	18146
1024	27534	14108	41642	28226	13240	41466	27634	12648	40282
2048	60530	30948	91478	62090	29080	91170	60706	27696	88402
4096	131982	67356	199338	135410	63224	198634	132290	60104	192394
8192	285810	145636	431446	293370	136664	430034	286514	129808	416322
16384	615310	313116	928426	631842	293944	925786	616722	278824	895546
32768	1318002	669924	1987926	1353706	628760	1982466	1320642	595696	1916338
65536	2810766	1427228	4237994	2887634	1339256	4226890	2816226	1267848	4084074

$$\begin{cases} A_N = A_{\frac{N}{4}} + 4A_{\frac{N}{8}} \\ \quad + 4A_{\frac{N}{16}} + 8N - 16, & N \geq 16; \\ M_N = \begin{cases} M_{\frac{N}{4}} + 4M_{\frac{N}{8}} + 4M_{\frac{N}{16}} \\ \quad + 7N/2 - 36, & N \geq 16, N \neq 32; \\ M_{\frac{N}{4}} + 4M_{\frac{N}{8}} + 4M_{\frac{N}{16}} \\ \quad + 7N/2 - 40, & N = 32; \end{cases} \end{cases} \tag{34}$$

With the initial conditions $M_2 = 0$, $A_2 = 4$, $M_4 = 0$, $A_4 = 16$, $M_8 = 4$, and $A_8 = 52$, we compute the (34) using Matlab, and obtain

$$\begin{aligned} M_N = {} & 1.16619676817862Nm - 3.60353192324843N \\ & - 0.00011868471889(-1)^m N \\ & + 0.00012597547822(-1)^j N \\ & + 0.00216426860976(-1)^k N \\ & - 0.00055409734728(-1)^l N \\ & + 0.72227590213094m \\ & - 2.55402321736061(-1)^m m \\ & + 0.04417056619718(-1)^j m \\ & - 1.55211776207181(-1)^k m \\ & - 3.83428848950868(-1)^l m \\ & + 1.88253879668650 \\ & + 19.50174660153815(-1)^m \\ & - 2.20517515688403(-1)^j \\ & 3.63499512596155(-1)^k \\ & + 23.06858051608533(-1)^l, \\ & \quad for\ N \geq 8; \end{aligned} \tag{35}$$

and

$$A_N = \frac{8}{3}N\log_2^N - \frac{16}{9}N + 2 - (-1)^{\log_2^N}\frac{2}{9}, \quad N \geq 8; \tag{36}$$

where $m = \log_2^N$, $j = ((m \bmod 3) == 0)$, $k = (((m+3) \bmod 4) == 0)$, and $l = (((m+3) \bmod 5) == 0)$. (Note: the integer value of the boolean value "true" is 1; the integer value of the boolean value "false" is 0). The expression of A_N is the same as the number of additions required by the SRFFT or the MSRFFT algorithm to compute a length-N DFT [31], [32]. Tables II gives the arithmetic complexity of the three-butterflies implementation. It is seen from this table that the number of multiplications needed by the proposed SR28FFT algorithm is less than those needed by the standard radix-2/4 FFT algorithm and the standard radix-2/8 FFT algorithm, the number of real additions required by the SR28FFT algorithm is equal to that required by the standard radix-24 FFT algorithm and less than that required by the standard radix-2/8 FFT algorithm, and the total flops required by the SR28FFT algorithm is less than those required by the two classical algorithms.

The NR24FFT algorithm decomposes a length-N DFT into a length-$N/2$ sub-DFT and two length-$N/4$ sub-DFTs. The length-$N/2$ sub-DFT is decomposed and evaluated recursively until the size is reduced to a 2-points DFT with the NR24FFT algorithm itself, and two length-$N/4$ sub-DFTs are computed with the SR28FFT algorithm. The decomposition is achieved by performing $N/4 - 2$ general butterflies and 2 special butterfly when $n = 0$ and $n = N/8$. The number of operations of the butterflies performed by the NR28FFT algorithm are the same as that performed by the SRFFT algorithm. The general butterfly requires 16 real additions and 8 real multiplications. The special butterfly when $n = 0$ requires 12 real additions, and the special butterfly when $n = N/16$ requires 4 real multiplications and 16 real additions. Let $A_N^{2/4}$ and $M_N^{2/4}$ be the number of real additions and real multiplications required by a length-N DFT by the proposed NR24FFT algorithm respectively. Therefore, the arithmetic complexity of the NR24FFT algorithm can be obtained easily by follows:

$$\begin{cases} A_N^{\frac{2}{4}} = A_{\frac{N}{2}}^{\frac{2}{4}} + 2A_{\frac{N}{4}} + 4N - 4, & N \geq 4; \\ M_N^{\frac{2}{4}} = \begin{cases} M_{\frac{N}{2}}^{\frac{2}{4}} + 2M_{\frac{N}{4}} + 2N - 8, & N = 4; \\ M_{\frac{N}{2}}^{\frac{2}{4}} + 2M_{\frac{N}{4}} + 2N - 12, & N > 4. \end{cases} \end{cases} \tag{37}$$

For the length-$N = 2$ DFT, $A_2^{2/4} = 4$ and $M_2^{2/4} = 0$.

TABLE II
COMPARISON OF ARITHMETIC COMPLEXITY OF THREE-BUTTERFLY

N	Radix-2/4			Radix-2/8			SR28FFT		
	Addition	Multication	Total	Addition	Multication	Total	Addition	Multication	Total
16	144	24	168	144	24	168	144	20	164
32	372	84	456	372	84	456	372	76	448
64	912	248	1160	920	240	1160	912	224	1136
128	2164	660	2824	2188	636	2824	2164	584	2748
256	5008	1656	6664	5072	1592	6664	5008	1468	6476
512	11380	3988	15368	11556	3812	15368	11380	3540	14920
1024	25488	9336	34824	25928	8896	34824	25488	8248	33736
2048	56436	21396	77832	57468	20364	77832	56436	18880	75316
4096	123792	48248	172040	126208	45832	172040	123792	42580	166372
8192	269428	107412	376840	274964	101876	376840	269428	94668	364096
16384	582544	236664	819208	594936	224272	819208	582544	208400	790944
32768	1252468	517012	1769480	1279980	489500	1769480	1252468	455160	1707628
65536	2679696	1121400	3801096	2740272	1060824	3801096	2679696	986732	3666428

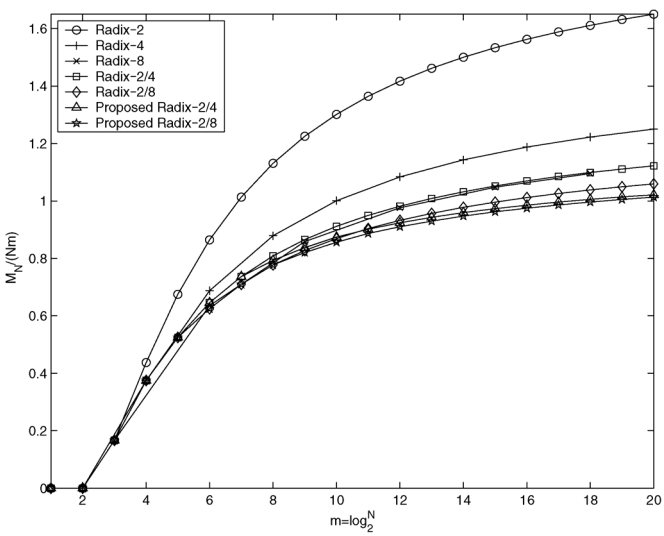

Fig. 7. Comparison of number of real multiplications.

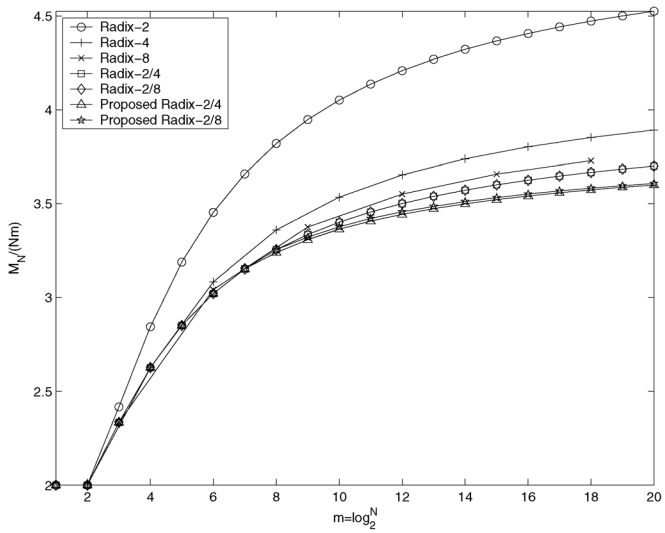

Fig. 8. Comparison of flops (real additions and real multiplications) count.

The NR28FFT algorithm decomposes a length-N DFT into a length-$N/2$ sub-DFT and four length-$N/8$ sub-DFTs. The decomposition is achieved by performing $N/8-2$ general butterflies and 2 special butterfly when $n=0$ and $n=N/16$. The decomposition is repeated successively for the length-$N/2$ sub-DFTs until the size is reduced to a 2- or 4-points DFT. Four length-$N/8$ sub-DFTs are decomposed and evaluated with the SR28FFT algorithm. The number of operations of the butterflies performed by the NR28FFT algorithm are the same as that performed by the standard radix-2/8 FFT algorithm. The general butterfly requires 44 real additions and 20 real multiplications. The special butterfly when $n=0$ requires 4 real multiplications and 36 real additions, and the special butterfly when $n=N/16$ requires 16 real multiplications and 40 real additions. Let $A_N^{2/8}$ and $M_N^{2/8}$ be the number of real additions and real multiplications required for a length-N DFT by the NR28FFT algorithm respectively. Therefore, it can be seen that the arithmetic complexity of the proposed NR28FFT algorithm is

$$A_N^{\frac{2}{8}}=\begin{cases}A_{\frac{N}{2}}^{\frac{2}{8}}+4A_{\frac{N}{8}}+11N/2-8, & N=8;\\ A_{\frac{N}{2}}^{\frac{2}{8}}+4A_{\frac{N}{8}}+11N/2-12, & N>8.\end{cases}$$
$$M_N^{\frac{2}{8}}=\begin{cases}M_{\frac{N}{2}}^{\frac{2}{8}}+4M_{\frac{N}{8}}+5N/2-20, & N=8;\\ M_{\frac{N}{2}}^{\frac{2}{8}}+4M_{\frac{N}{8}}+5N/2-24, & N>8.\end{cases}\tag{38}$$

For length $N=2$ and 4 DFTs, the algorithm requires 4 or 16 real additions, and 0 and 4 real multiplications for computing them. Two comparisons are given in Figs. 7 and 8 respectively. It is clear seen from these two figures that the NR24FFT algorithm and the NR28FFT algorithm require less number of real multiplications and less flops (real additions and real multiplications) count among radix-2, radix-4, radix-8, radix-2/4, radix-2/8, and the two proposed FFT algorithms.

C. Twiddle Factors

It is obvious that two real coefficients are required when the 4mult-2add (a complex multiplication is performed using 4 real multiplications and 2 real additions) scheme is used to compute a complex multiplication. In counting the number of real coefficient evaluations or accesses to the lookup table required by the proposed SR28FFT algorithm, it is assumed that the coefficients required by the special butterflies, such as $\sqrt{2}/2$, $\cos(2\pi/16)$, $\sin(2\pi/16)$, $\cos(2\pi/16)\sqrt{2}/2$, and $\sin(2\pi/16)\sqrt{2}/2$ are initialized and kept in the internal registers of the processor during the processing time of the corresponding algorithm. For the general butterfly, sixty real coefficients are required to be evaluated or loaded from the lookup table by the SR28FFT algorithm. To finish the same task, the standard radix-2/8 FFT algorithm requires to perform the same evaluations or accesses to the lookup table. For special butterfly when $n = 0$, no coefficient is required to be computed or loaded by the SR28FFT algorithm and the standard radix-2/8 FFT algorithm. For special butterfly when $n = N/32$, four or no coefficient is required to be computed or loaded by the SR28FFT algorithm, and eight coefficients are required by the standard radix-2/8 FFT algorithm, if all the real and imaginary parts of w_{32}^1, w_{32}^3, w_{32}^5, and w_{32}^7 are or are not initiated and stored in the internal registers. It is seen that all or half of evaluations or accesses of real and imaginary parts of w_{32}^1, w_{32}^3, w_{32}^5, and w_{32}^7 are saved by the SR28FFT algorithm compared with the standard radix-2/8 FFT algorithm. Therefore, by counting the required number of the butterflies in the SR28FFT algorithm, the number of real coefficient evaluations or accesses to the lookup table for the two algorithms is

$$T_N = \begin{cases} T_{\frac{N}{4}} + 4T_{\frac{N}{8}} + 4T_{\frac{N}{16}}, & N = 16; \\ T_{\frac{N}{4}} + 4T_{\frac{N}{8}} + 4T_{\frac{N}{16}} + 4, & N = 32; \\ T_{\frac{N}{4}} + 4T_{\frac{N}{8}} + 4T_{\frac{N}{16}} + 16(N/16 - 2) + 4, & N \geq 32; \end{cases} \tag{39}$$

where for comparing under the same conditions, it is assumed that all the real and imaginary parts of w_{32}^1, w_{32}^3, w_{32}^5, and w_{32}^7 are not initiated into the internal registers. For various values of N, the number of real coefficient evaluations or accesses to the lookup table of the proposed SR28FFT algorithm is compared to that required by the standard-2/8 FFT algorithm in Table III. It is seen from the table that savings of over 35% in the evaluation of twiddle factors or in the access to the lookup table can easily be achieved by the proposed algorithm. Note that, when the lookup table is used, similar savings are obtained by the proposed algorithm in the address generation for reading the twiddle factors.

D. Real Input Signals

When the NR24FFT algorithm and the NR28FFT algorithm deal with real input signals, the numbers of multiplication are half of those required for complex input signals, and the numbers of real addition are half of those required for complex input signals minus $N-2$, which is similar to the MSRFFT algorithm, the radix-2/8 FFT algorithm, and the radix-2/4 FFT algorithm for real input signals.

V. Floating Point Accuracy

Accuracy is one of the most important factors of system performance. In this section, we discuss the accuracy on the computing results of the L_2 (root-mean-square) relative error $\sqrt{\sum |\Delta X(k)|^2}/\sqrt{\sum |X(k)|^2}$ of length-$N = 2^m$ DFTs.

TABLE III
Comparison of Real Coefficient Evaluations or Accesses to Lookup Table

N	Standard radix-2/8	SR28FFT	Savings (%)
32	16	4	75.00
64	64	36	43.75
128	176	104	40.91
256	480	280	41.67
512	1232	748	39.29
1024	2944	1836	37.64
2048	6896	4304	37.59
4096	15904	10016	37.02
8192	35856	22804	36.40
16384	79808	50932	36.18
32768	176176	112824	35.96
65536	385120	247720	35.68
131072	835408	538812	35.50

TABLE IV
L_2 Relative Error for Length-$N = 2^m$ DFTs ($\times 10^{-7}$)

N	Standard radix-2/8 (A)	NR24FFT (B)	NR28FFT (C)	Improvements % in B over A	Improvements % in C over A
64	1.031	1.025	1.028	0.62	0.31
128	1.077	1.003	1.004	6.92	6.77
256	1.104	1.118	1.103	-1.27	0.12
512	1.215	1.196	1.186	1.57	2.40
1024	1.279	1.255	1.254	1.85	1.93
2048	1.368	1.319	1.324	3.60	3.23
4096	1.424	1.400	1.401	1.63	1.56
8192	1.494	1.490	1.483	0.30	0.76
16384	1.558	1.531	1.533	1.76	1.63
32768	1.626	1.580	1.573	2.85	3.28
65536	1.686	1.654	1.664	1.93	1.34
131072	1.743	1.697	1.716	2.67	1.56

We compute the L_2 relative error of the related algorithms compared to the exact results, for pseudo random inputs $x(n) \in [-0.5, 0.5)$, in 32-bits single precision, on Pentium E6700 with Windows 7 and Microsoft Visual Studio 2008. The coefficients are pre-computed and stored in a lookup table, by rounding/truncating the 64-bits double precision coefficients into 32-bits single precision coefficients. The exact results are from the implementation of a common approach in the case of 64-bits double precision. The comparing results, in Table IV, show that the computation accuracy is slightly improved by the proposed NR24FFT and NR28FFT algorithms compared with the standard radix-2/8 FFT algorithm. It is well known that the amount of the roundoff has randomness and determinism. The general improvement on quantized loss lies in the determinism, since the SR28FFT algorithm reduces the number of complex multiplication by w_8^1 or w_8^3. The improvement of "-1.27%" is perhaps resulted in by the randomness.

VI. Conclusion

In this paper, we have proposed an auxiliary algorithm, the SR28FFT algorithm, which is based on the standard radix-2/8 FFT algorithm, for efficiently computing the length-2^m sub-DFTs scaled by complex number rotating factors. Two novel algorithms based on the SR28FFT algorithm, i.e., the NR24FFT algorithm and the NR28FFT algorithm, have been proposed for computing length-$N = 2^m$ DFTs. It has been shown that, the

proposed SR28FFT algorithm reduces the number of operations compared with the standard radix-2/8 FFT algorithm and the standard radix-2/4 FFT algorithm, and several aspects of the FFT algorithms such as arithmetic complexity, computation accuracy, and coefficient evaluations or accesses to the lookup table have been improved by the NR24FFT algorithm and the NR28FFT algorithm compared with the standard radix-2/4 FFT algorithm and the standard radix-2/8 FFT algorithms. Moreover, the NR24FFT algorithm and the NR28FFT algorithm can use the bit-reverse method to permute their data order, and has no extra coefficients required to be evaluated and stored. In addition, the SR28FFT-2 algorithm in the Appendix further reduces the number of arithmetic operations compared to that required by the SR28FFT algorithm. If the NR24FFT algorithm and the NR28FFT algorithm use the SR28FFT-2 algorithm, their computational complexity are lower than the MSRFFT algorithm. The SR28FFT algorithm and the SR28FFT-2 algorithm can also be applied to other algorithms such as radix-4, radix-8, etc to efficiently compute their similar length-2^m sub-DFTs.

APPENDIX

In this appendix, we only consider the improvement of computational complexity of the proposed algorithm, and ignore the improvements of other performance. For differentiating the SR28FFT algorithm in Section II, the algorithm in this appendix is named "SR28FFT-2". In the SR28FFT-2 algorithm, the number of extracting factors is two. The first one is similar to the factor in (2), defined in the following.

$$s_1(N,n)=\begin{cases}1, & N\le 4;\\ s_1(N/8,n), & 0<(n \bmod N/4)<N/8;\\ s_1(N/8,n)w_8^1, & N/8\le (n \bmod N/4).\end{cases} \tag{40}$$

Another one is defined by

$$s_2(N,n)=\begin{cases}1, & N\le 8;\\ s_2(N/8,n)\cos(2\pi(n \bmod N/8)/N), & \text{otherwise.}\end{cases} \tag{41}$$

Let $\varsigma(N,n)$ be the product of $s_1(N,n)$ and $s_2(N,n)$, i.e.,

$$\varsigma(N,n)=s_1(N,n)s_2(N,n),\quad n=0,1,\ldots,N-1. \tag{42}$$

Replacing the definition of $s(N,n)$ in (2) by the definition in (42), the DFT in (3) can be used for further decomposition in this appendix. The following decompositions are provided so that the DFT of (3) can be computed efficiently:

$$X(2k)=\sum_{n=0}^{N/2-1} b(n)/\varsigma(N,n)w_{\frac{N}{2}}^{nk},\quad k=0,1,\ldots,(N/2-1), \tag{43}$$

for the even-indexed terms, and

$$X(8k+1)=\sum_{n=0}^{N/8-1}(b_e(n)+b_e(N/8+n))\,tn_N^n w_{\frac{N}{8}}^{nk},\quad k=0,1,\ldots,(N/8-1); \tag{44}$$

$$X(8k+5)=\sum_{n=0}^{N/8-1}(b_e(n)-b_e(N/8+n))\,\frac{w_N^{5n}}{\cos(2\pi n/N)}w_{\frac{N}{8}}^{nk},\quad k=0,1,\ldots,(N/8-1); \tag{45}$$

$$X(8k-1)=\sum_{n=0}^{N/8-1}(b_o(n)+jb_o(N/8+n))\,tn_N^{-n} w_{\frac{N}{8}}^{nk},\quad k=0,1,\ldots,(N/8-1); \tag{46}$$

$$X(8k-5)=\sum_{n=0}^{N/8-1}(b_o(n)-jb_o(N/8+n))\,\frac{w_N^{-5n}}{\cos(2\pi n/N)}w_{\frac{N}{8}}^{nk},\quad k=0,1,\ldots,(N/8-1); \tag{47}$$

for the odd-indexed terms, where sequences $b(n)$ in (43), $b_e(n)$ in (44) and (45), and $b_o(n)$ in (46) and (47) have been defined in Eqs. (10)–(12), and $tn_N^{\pm n}$ in (44) and (46) can be defined by

$$tn_N^{\pm n}=\frac{w_N^{\pm n}}{\cos(2\pi n/N)}. \tag{48}$$

Similar to the SR28FFT algorithm, the length-$N/2$ sub-DFT in (43) needs to be decomposed further. The following decomposition

$$X(4k)=\sum_{n=0}^{N/4-1} c(n)\frac{\varsigma(N/4,n)}{\varsigma(N,n)}w_{\frac{N}{4}}^{nk},\ k=0,1,\ldots,(N/4-1), \tag{49}$$

is provided for its even-indexed terms, and the decompositions

$$X(16k+2)=\sum_{n=0}^{N/16-1}(c_e(n)+c_e(N/16+n))\,w_N^{2n}\frac{s_2(N/16,n)}{\alpha(N,n)}w_{\frac{N}{16}}^{nk},\quad k=0,1,\ldots,(N/16-1); \tag{50}$$

$$X(16k+10)=\sum_{n=0}^{N/16-1}(c_e(n)-c_e(N/16+n))\,w_N^{10n}\frac{s_2(N/16,n)}{\alpha(N,n)}w_{\frac{N}{16}}^{nk},\quad k=0,1,\ldots,(N/16-1); \tag{51}$$

$$X(16k-10)=\sum_{n=0}^{N/16-1}(c_o(n)-jc_o(N/16+n))\,w_N^{-10n}\frac{s_2(N/16,n)}{\alpha(N,n)}w_{\frac{N}{16}}^{nk},\quad k=0,1,\ldots,(N/16-1); \tag{52}$$

$$X(16k-2)=\sum_{n=0}^{N/16-1}(c_o(n)+jc_o(N/16+n))\,w_N^{-2n}\frac{s_2(N/16,n)}{\alpha(N,n)}w_{\frac{N}{16}}^{nk},\quad k=0,1,\ldots,(N/16-1); \tag{53}$$

are provided for the odd-indexed terms, where $c(n)=b(n)+b(N/4+n)$. The sequence c_e in (50) and (51) and the sequence

TABLE V
NUMBER OF REAL MULTIPLICATIONS

N	MSRFFT	SR28FFT-2	NR24FFT
16	24	20	24
32	84	72	84
64	240	208	240
128	628	548	628
256	1544	1372	1544
512	3668	3296	3652
1024	8480	7688	8432
2048	19252	17596	19108
4096	43064	39636	42664
8192	95252	88120	94228
16384	208720	193984	206256
32768	453876	423508	448020
65536	980584	917964	967048

c_o in (53) and (52) have been defined in (18) with $\mathbf{S_2}$ re-defined by

$$\mathbf{S_2} = \begin{bmatrix} \frac{\alpha(N,n)}{s(N,n)} & 0 \\ 0 & \frac{\alpha(N,n)w_8^1}{s(N,N/16+n)} \end{bmatrix}, \tag{54}$$

where

$$\alpha(N,n) = \begin{cases} s_2(N,n), & for\ \frac{w_8^1}{s_1(N,n)} = \pm 1\ or\ \pm j; \\ s_2(N,N/16+n), & for\ \frac{w_8^1}{s_1(N,N/16+n)} = \pm 1\ or\ \pm j. \end{cases} \tag{55}$$

The four length-$N/16$ sub-DFTs in (50)–(53) will be decomposed recursively using (43)–(46). The length-$N/4$ DFT in (13) is calculated using the method of the SR28FFT algorithm.

The SR28FFT-2 algorithm is similar to the SR28FFT algorithm. However, there are the following two points that the SR28FFT-2 algorithm can reduce the number of real multiplications compared with the SR28FFT algorithm.

1) For each $n < N/8$, except for the special cases when $n = 0$, the computation of $X(8k+1)$ and $X(8k+7)$ of the SR28FFT algorithm requires 4 more real multiplications than that of $X(8k+1)$ and $X(8k-1)$ of the SR28FFT-2 algorithm, and the computation of $X(4k)$ of the SR28FFT algorithm requires 2 fewer real multiplications than that of the SR28FFT-2 algorithm. In other words, a general butterfly or a special butterfly when $n = N/32$ of the SR28FFT-2 algorithm can save 4 real multiplications compared to that of the SR28FFT algorithm
2) As far as the SR28FFT algorithm is concerned, the recursive decomposition of $X(4k)$ in the special case when $n = 0$ cannot save the number of operations, since in (13) the number of real operations involved in $s\prime(N/4,n)/s\prime(N,n)$ is four more real multiplications than those involved in $s(N/4,n)/s\prime(N,n)$, which offsets the real multiplication saved by the special butterfly when $n = 0$ in the next stage. However, in (49), the number of real multiplications involved in $\varsigma(N/4,n)/\varsigma(N,n)$ does not require extra real multiplications compared with that involved in $1/\varsigma(N,n)$, namely, the recursive decomposition of $X(4k)$ can in the special of $n = 0$ save the four real multiplications.

That is, except for the special butterfly when $n = 0$, each butterfly of the new algorithm requires 4 real multiplications less than the SR28FFT algorithm. The following equation gives the number of real multiplications required by the SR28FFT-2 algorithm:

$$M_N = M_{\frac{N}{4}} + 4M_{\frac{N}{8}} + 4M_{\frac{N}{16}} + 13N/4 - 36. \tag{56}$$

For small lengths $N = 2$, 4, 8, and 16 DFTs, the SR28FFT-2 algorithm require 0, 0, 4, and 20 real multiplications for implementing them. The three algorithms, MSRFFT, SR28FFT, and NR24FFT, require the same number of real additions for a DFT. Table V gives a comparison of the number of real multiplications required by these three algorithms, showing that the SR28FFT-2 algorithm requires less real multiplication than the MSRFFT algorithm for $N \geq 16$, and the NR24FFT algorithm requires less real multiplications than the MSRFFT algorithm when $N \geq 512$. In reality, the NR28FFT algorithm also requires less real operations than MSRFFT.

ACKNOWLEDGMENT

The authors greatly appreciate all editors and reviewers for their constructive comments, which helped us to further improve the quality of the manuscript.

REFERENCES

[1] P. Duhamel and M. Vetterli, "Fast Fourier transforms: A tutorial review and a state of the art," *Signal Process.*, vol. 19, no. 4, pp. 259–299, 1990.
[2] C. Rader, "Discrete Fourier transforms when the number of data samples is prime," *Proc. IEEE*, vol. 56, no. 6, pp. 1107–1108, 1968.
[3] S. Winograd, "On computing the discrete Fourier transform," *Math. Computat.*, vol. 32, no. 141, pp. 175–199, 1978.
[4] Y. Suzuki, T. Sone, and K. Kido, "A new FFT algorithm of radix 3, 6, 12," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-34, no. 2, pp. 380–383, Apr. 1986.
[5] W. Zheng and K. Li, "Split radix algorithm for length 6^m DFT," *IEEE Signal Process. Lett.*, vol. 20, pp. 713–716, Jul. 2013.
[6] W. Zheng, K. Li, and K. Li, "A fast algorithm based on SRFFT for length $N = q \times 2^m$ DFTs," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 61, pp. 110–114, Feb. 2014.
[7] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, no. 90, pp. 297–301, 1965.
[8] R. Yavne, "An economical method for calculating the discrete Fourier transform," in *Proc. AFIPS Fall Joint Comput. Conf.*, 1968, vol. 33, pp. 115–125.
[9] P. Duhamel and H. Hollmann, "Split-radix FFT algorithm," *Electron. Lett.*, vol. 20, pp. 14–16, Jan. 1984.
[10] M. Vetterli and H. J. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Process.*, vol. 6, no. 4, pp. 267–278, 1984.
[11] J. Martens, "Recursive cyclotomic factorization-a new algorithm for calculating the discrete Fourier transform," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 32, no. 4, pp. 750–761, 1984.
[12] R. Stasinski, "The techniques of the generalized fast Fourier transform algorithm," *IEEE Trans. Signal Process.*, vol. 39, no. 5, pp. 1058–1069, 1991.
[13] S. Bouguezel, M. O. Ahmad, and M. Swamy, "A general class of split-radix FFT algorithms for the computation of the DFT of length- 2^m," *IEEE Trans. Signal Process.*, vol. 55, no. 8, pp. 4127–4138, 2007.
[14] G. Bi, G. Li, and X. Li, "A unified expression for split-radix DFT algorithms," in *Proc. IEEE Int. Conf. Commun., Circuits Syst. (ICCAS)*, 2010, pp. 323–326.
[15] D. Takahashi, "An extended split-radix FFT algorithm," *IEEE Signal Process. Lett.*, vol. 8, no. 5, pp. 145–147, 2001.
[16] S. Bouguezel, M. Ahmad, and M. Swamy, "A new radix-2/8 FFT algorithm for length- $q \times 2^m$ DFTs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 51, no. 9, pp. 1723–1732, Sep. 2004.

[17] A. Grigoryan and S. Agaian, "Split manageable efficient algorithm for Fourier and Hadamard transforms," *IEEE Trans. Signal Process.*, vol. 48, no. 1, pp. 172–183, 2000.
[18] M. Frigo and S. Johnson, "FFTW: An adaptive software architecture for the FFT," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1998, vol. 3, pp. 1381–1384.
[19] W. Chang and T. Nguyen, "On the fixed-point accuracy analysis of FFT algorithms," *IEEE Trans. Signal Process.*, vol. 56, no. 10, pp. 4673–4682, 2008.
[20] T. Cho, H. Lee, J. Park, and C. Park, "A high-speed low-complexity modified radix- 2^5 FFT processor for gigabit WPAN applications," in *Proc. IEEE Circuits Syst. Int. Symp. (ISCAS)*, 2011, pp. 1259–1262.
[21] K. J. Bowers, R. A. Lippert, R. O. Dror, and D. E. Shaw, "Improved twiddle access for fast Fourier transforms," *IEEE Trans. Signal Process.*, vol. 58, no. 3, pp. 1122–1130, 2010.
[22] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans.*, no. 3, pp. 330–334, 1959.
[23] V. Kantabutra, "On hardware for computing exponential and trigonometric functions," *IEEE Trans. Comput.*, vol. 45, no. 3, pp. 328–339, 1996.
[24] C. Burrus, "Bit reverse unscrambling for a radix- 2^m FFT," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP)*, 1987, vol. 12, pp. 1809–1810.
[25] I. Kamar and Y. Elcherif, "Conjugate pair fast Fourier transform," *Electron. Lett.*, vol. 25, no. 5, pp. 324–325, Apr. 1989.
[26] R. A. Gopinath, "Comment on conjugate pair fast Fourier transform," *Electron. Lett.*, vol. 25, no. 16, p. 1084, Aug. 1989.
[27] J. C. Schatzman, "Accuracy of the discrete Fourier transform and the fast Fourier transform," *SIAM J. Scientif. Comput.*, vol. 17, no. 5, pp. 1150–1166, 1996.
[28] E. G. Larsson and O. Gustafsson, "The impact of dynamic voltage and frequency scaling on multicore DSP algorithm design [exploratory DSP]," *IEEE Signal Process. Mag.*, vol. 28, no. 3, pp. 127–144, 2011.
[29] K. Maharatna, E. Grass, and U. Jagdhold, "A 64-point Fourier transform chip for high-speed wireless LAN application using OFDM," *IEEE J. Solid-State Circuits*, vol. 39, no. 3, pp. 484–493, 2004.
[30] W.-C. Yeh and C.-W. Jen, "High-speed and low-power split-radix FFT," *IEEE Trans. Signal Process.*, vol. 51, no. 3, pp. 864–874, 2003.
[31] H. Sorensen, M. Heideman, and C. Burrus, "On computing the split-radix FFT," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 34, no. 1, pp. 152–156, 1986.
[32] S. G. Johnson and M. Frigo, "A modified split-radix FFT with fewer arithmetic operations," *IEEE Trans. Signal Process.*, vol. 55, no. 1, pp. 111–119, 2007.

Weihua Zheng received the Master's degree from the National University of Defense Technology, China, in 2010.

He is currently working towards the Ph.D. degree at Hunan University of China. He is an associate professor of computer science and technology at Hunan University of Technology. His research interests include fast Fourier transform, audio signal processing, image processing, parallel computing.

Kenli Li received the Ph.D. degree in computer science from Huazhong University of Science and Technology, China, in 2003.

He was a visiting scholar at University of Illinois at Urbana-Champaign from 2004 to 2005. He is currently a full professor of computer science and technology at Hunan University and associate director of National Supercomputing Center in Changsha. His major research includes parallel computing, grid and cloud computing, and DNA computing. He has published more than 90 papers in international conferences and journals, such as IEEE-TC, IEEE-TPDS, JPDC, ICPP, CCGrid.

Dr. Li is an outstanding member of CCF.

Keqin Li (SM'96) is a SUNY Distinguished Professor of computer science. He is an Intellectual Ventures endowed visiting chair professor at Tsinghua University, China. His research interests are mainly in design and analysis of algorithms, parallel and distributed computing, and computer networking. He has over 285 refereed research publications.

Professor Li is currently or has served on the editorial board of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON COMPUTERS, *Journal of Parallel and Distributed Computing*, *International Journal of Parallel, Emergent and Distributed Systems*, *International Journal of High Performance Computing and Networking*, *International Journal of Big Data Intelligence*, and *Optimization Letters*.