Signal Processing 113 (2015) 68–79

Contents lists available at ScienceDirect

Signal Processing

journal homepage: www.elsevier.com/locate/sigpro

Datapath-regular implementation and scaled technique for $N=3\times 2^m$ DFTs

Weihua Zheng [a], Kenli Li [a,*], Keqin Li [a,b]

[a] College of Information Science and Engineering, Hunan University, Changsha, Hunan 410082, China
[b] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Article history:
Received 23 May 2014
Received in revised form
8 January 2015
Accepted 9 January 2015
Available online 19 January 2015

Keywords:
Datapath-regular
Radix-2/8
Radix-3
Scaled discrete Fourier transform

ABSTRACT

Discrete Fourier transform (DFT) is used widely in almost all fields of science and engineering, and is generally calculated using the fast Fourier transform (FFT) algorithm. In this paper, we present a fast algorithm for efficiently computing a DFT of size 3×2^m. The proposed algorithm decomposes the DFT, obtaining one length-2^m unscaled sub-DFT and two length-2^m sub-DFTs scaled by constant real numbers. For efficiently computing the scaled sub-DFTs, the constant real factors are attached to twiddle factors, combining them into new twiddle factors. By using this approach, the number of real multiplications is reduced compared with existing algorithms. To obtain regular datapath, a novel implementation method is presented aiming at the implementation of the proposed algorithm and making its datapath regular like the radix-2 FFT algorithm. The method can be applied to other algorithms with L-shape butterfly. Experimental result shows that, the proposed algorithm consumes less processing time than the existing algorithms for all scale DFTs, and than FFTW, a C subroutine library of FFTs, just for small scale DFTs.

1. Introduction

Discrete Fourier transform (DFT) is widely used in almost all fields of science and engineering, where frequency-domain representation of a signal has to be analyzed [1–4]. In communication area, it has gained extensive attention, because it is used in orthogonal frequency division multiplexing (OFDM) systems. However, direct computation of the DFT is a computationally intensive task. Therefore, highly efficient algorithms for computing the DFT are of great importance. Fast Fourier transform (FFT) is one class of efficient algorithms to compute a DFT.

FFT sequence lengths include powers-of-two and also non-powers-of-two. Hence, it is necessary to develop FFT algorithms to evaluate DFTs whose lengths are powers-of-two and also non-powers-of-two. Since the discovery of the FFT [5], considerable effort has been devoted to the development of the FFT algorithms for DFTs whose lengths are powers-of-two [6–10]. Bouguezel et al. [11] and Bi et al. [12] proposed a general class of split-radix FFT algorithms, in which higher split-radix FFT algorithms substantially reduce data transfer and twiddle factor evaluations as compared to lower split-radix FFT algorithms. Johnson and Frigo proposed a modified SRFFT (MSRFFT), reducing the number of operations as compared to SRFFT at the cost of more evaluations of twiddle factors, a non-standard permutation of data orders, and poorer computation precision [13]. Enlightened by MSRFFT, Zheng et al. [14] proposed two FFT algorithms, outperforming SRFFT in the three aspects of complexity, evaluations of twiddle factors, and computation precision, and also proposed an algorithm which has the lowest arithmetic complexity as compared to published algorithms. Simultaneously, considerable effort has been devoted to FFT algorithms for DFTs whose lengths are non-powers-of-two, such as the algorithms in [15–20] for DFTs whose lengths contain factor 3.

* Corresponding author.
E-mail addresses: zhengdavid@hnu.edu.cn (W. Zheng), lkl@hnu.edu.cn (K. Li), lik@newpaltz.edu (K. Li).

http://dx.doi.org/10.1016/j.sigpro.2015.01.008
0165-1684/© 2015 Elsevier B.V. All rights reserved.

When examining the amount of time spent on operations, it is interesting to note that time spent on load/store operations is more than that on actual arithmetic computations [1]. Reducing the number of floating-point operations is of less significance to execution time than that used on recent hardware [23–25]. Many applications require more convenience and more flexibility rather than just a lower computational complexity [26,27]. The highly symmetric structure of DFT lends itself nice to implement in various parallel schemes [28,29]. Considerable investigations have been carried out into architectures that efficiently compute FFT algorithms, and specifically into memory conflict of FFT processors [30–32]. Contrary to good behavior on general purpose computers, radix-2/4, radix-2/8, and radix-3/9 FFT algorithms have higher complexity on FFT processors, since the algorithms with L-shape butterflies are more irregular than algorithms with non-L-shape butterflies.

Applications in which lengths of DFTs are 3×2^m are arising recently [19,20,27]. The following are several examples.

1. FFT is an efficient tool to compute MDCT. The MPEG audio coding standard uses dynamically window modified discrete cosine transform (MDCT) to achieve high quality performance. In layer III of MDCT-I and MDCT-II, the length of data blocks is $N \neq 2^m$. The layer III specifies a longer block (N=36) and a shorter block (N=12).
2. In the applications of OFDM demodulation and modern microscopy, the sequence lengths may be non-powers-of-two.
3. A 1536-points FFT processor is used in Third Generation Partnership Project Long-Term Evolution.

The length-3×2^m DFT is a special case of the radix-6 FFT algorithm of Suzuki et al. [17] and the radix-3/6 FFT of Zheng and Li [19] when power of 3 is unity. The special case has been well researched by Bi [21] in 1998 and Bouguezel et al. [22] in 2004. However, there exists room for improvement in computational complexity for this special case.

Thus, we carry out a research for efficiently computing the length$-N = q \times 2^m$ DFT, where q is an odd integer. Our previous work [20] and this paper are two parts of the research. In our previous work [20], we proposed a framework in which, through the scaled DFT (SDFT) technique, the performance of FFTs can be improved in both computational complexity and accurate precision. In this paper, we propose an indexing scheme to map a standard FFT to a non-standard FFT, and an implementation method to iteratively implement the radix-2/8 FFT. The proposed indexing scheme makes the algorithm in [20] clearer. The proposed implementation method provides an iterative way for the implementation of the radix-2/8 FFT, which can take full advantage of the higher split-radix FFT and reduce data transfer and coefficient evaluations [22]. On the other hand, up till now, there does not exist any FFT processor whose architecture is specialized for the radix-2/8 FFT. Therefore, it is interesting to find an approach that uses the radix-2/8 FFT to efficiently compute length-N=3×2^m DFTs on a FFT processor.

In this paper, the length$-N = 3 \times 2^m$ DFT is decomposed with the proposed radix-3 FFT. By using the proposed radix-3 algorithm, two length-2^m sub-DFTs, which are scaled by -1.5 and $\sin(2\pi/3)$ respectively can be obtained. The scaled sub-DFTs are implemented efficiently with the proposed scaled radix-2/8 FFT. The proposed radix-3 and scaled radix-2/8 FFTs easily allow us to achieve the improvement in computational complexity.

The rest of the paper is organized as follows. Section 2 introduces the proposed algorithm in detail, including the proposed radix-3 FFT and the scaled radix-2/8 FFT. Section 3 presents a method to implement the radix-2/8 FFT and the proposed algorithm. Section 4 analyzes the performance of the proposed algorithm by comparing its computational complexity, accesses to lookup table, and execution time with the algorithms reported in the literature [2,21,22]. Finally, Section 5 offers concluding remarks.

2. Proposed radix-3 algorithm

In this section, we present a FFT algorithm for the computation of length$-N = 3 \times 2^m$ DFTs. In contrast with our previous work [20], an indexing scheme is proposed for mapping a standard FFT to a non-standard FFT, which is a key technique for the proposed algorithm.

2.1. Proposed radix 3 algorithm

Suppose that $x(n)$ is a sequence of length N, consisting of complex numbers. The DFT of this sequence is also a sequence, composed of the elements

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

where $W_N^{nk} = e^{-j2\pi nk/N}$ and $j = \sqrt{-1}$. Assume that the length N equals 3×2^m in the following discussion.

We now consider the decomposition of Eq. (1). The proposed radix-3 FFT algorithm provides the following decomposition:

$$\begin{aligned} X(k) = & \sum_{n=0}^{N/3-1} x(3n) W_{N/3}^{nk} + W_3^k \sum_{n=0}^{N/3-1} x(3n+N/3) W_{N/3}^{nk} \\ & + W_3^{-k} \sum_{n=0}^{N/3-1} x(3n-N/3) W_{N/3}^{nk}, \\ & \qquad k = 0, 1, \ldots, N-1, \end{aligned} \tag{2}$$

for the DFT. Two of the three sub-DFTs are rotated with twiddle factors W_3^k and W_3^{-k} respectively. Generally, these three sub-DFTs will be computed with a FFT algorithm. Since their lengths are powers-of-two, the algorithm that is used to compute the sub-DFTS can be the radix-2 FFT, the radix-4 FFT, the radix-2/4 FFT, the radix-2/8 FFT, mix-radix FFT, or MSRFFT, etc. In order to share operations between the two sub-DFTs with rotating factors, the decomposition of the proposed algorithm provides

$$X(3k) = \sum_{n=0}^{N/3-1} (x(3n) + u(n)) W_{N/3}^{3nk},$$
$$k = 0, 1, \ldots, N/3-1, \tag{3}$$

$$X(3k+N/3)=(X(3k)+F(3k))+(-1)^{(N/3 \bmod 3)}jG(3k),\quad k=0,1,\ldots,N/3-1, \tag{4}$$

$$X(3k-N/3)=(X(3k)+F(3k))-(-1)^{(N/3 \bmod 3)}jG(3k),\quad k=0,1,\ldots,N/3-1, \tag{5}$$

for Eq. (2), where

$$F(3k)=-1.5\sum_{n=0}^{N/3-1}u(n)W_{N/3}^{3nk},\quad k=0,1,\ldots,N/3-1, \tag{6}$$

and

$$G(3k)=\sin(2\pi/3)\sum_{n=0}^{N/3-1}v(n)W_{N/3}^{3nk},\quad k=0,1,\ldots,N/3-1. \tag{7}$$

The sequence $u(n)$ in Eqs. (3) and (6) and $v(n)$ in Eq. (7) can be represented in a matrix form

$$\begin{bmatrix}u(n)\\ v(n)\end{bmatrix}=\mathbf{H}_2\begin{bmatrix}x(3n+N/3)\\ x(3n-N/3)\end{bmatrix}, \tag{8}$$

where

$$\mathbf{H}_2=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}. \tag{9}$$

It is noteworthy that the sub-DFTs in Eqs. (3), (6) and (7) are three non-standard DFTs. However, these three DFTs can be directly computed with an algorithm which can evaluate powers-of-two DFTs. Assume that the output of the DFT in Eq. (6) directly computed through a FFT algorithm is $f(\eta)$. One can give the definition of $f(\eta)$ as the following:

$$f(\eta)=-1.5\sum_{n=0}^{N/3-1}u(n)W_{N/3}^{n\eta},\quad \eta=0,1,\ldots,N/3-1. \tag{10}$$

An indexing scheme is used, providing

$$F(3k)=f(3k \bmod N/3),\quad k=0,1,\ldots,N/3-1, \tag{11}$$

for mapping the sequence $f(\eta)$ in Eq. (10) to the sequence $F(3k)$ in Eq. (6). The indexing schemes used for computing Eqs. (3) and (7) are similar to Eq. (11). Assume that the output of the DFT in Eq. (7) directly computed through a FFT algorithm is $g(\eta)$. One can give the definition of $g(\eta)$ as the following:

$$g(\eta)=\sin(2\pi/3)\sum_{n=0}^{N/3-1}v(n)W_{N/3}^{n\eta},\quad \eta=0,1,\ldots,N/3-1. \tag{12}$$

An indexing scheme is used, providing

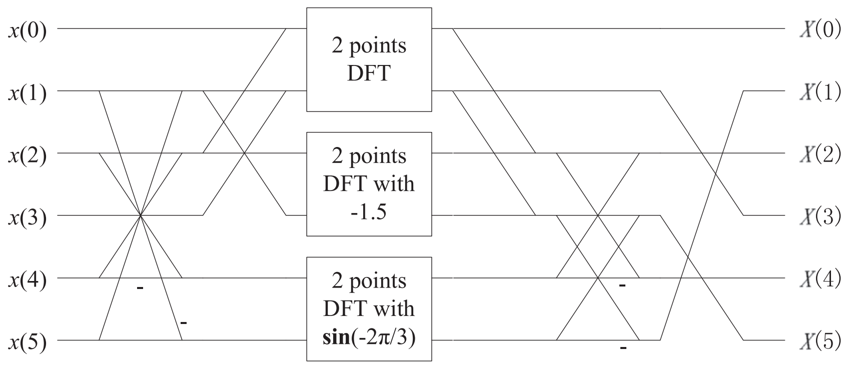

Fig. 1. Flowgraph for a size-3×2 DFT.

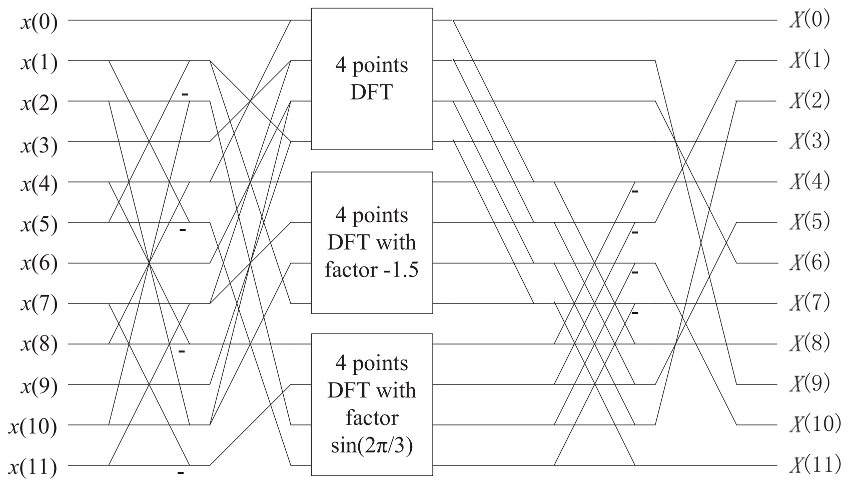

Fig. 2. Flowgraph for a size-3×4 DFT.

$$G(3k) = g(3k \bmod N/3), \quad k = 0, 1, \ldots, N/3-1, \tag{13}$$

for mapping the sequence $g(\eta)$ in Eq. (12) to the sequence $G(3k)$ in Eq. (7).

We now summarize the proposed scheme for computing the length$-N = 3 \times 2^m$ DFT. The proposed algorithm decomposes the DFT in Eq. (1) into three sub-DFTs in Eqs. (3)–(5). The sub-DFT $X(3k)$ is a general sub-DFT of size $N/3 = 2^m$. The sub-DFTs $F(3k)$ and $G(3k)$ are two scaled sub-DFTs. The general sub-DFT will be evaluated with the standard radix-2/8 algorithm and the two scaled sub-DFTs will be computed with the scaled radix-2/8 algorithm proposed in the next subsection. Figs. 1 and 2 show the flowgraph of a size-3×2 DFT and the flowgraph of a size-3×4 DFT respectively. The general flowgraph of a length-$N = 3 \times 2^m$ DFT is illustrated in Fig. 3. As compared to the algorithms in [17,21,22], when a length$-N = 3 \times 2^m$ DFT is decomposed decimation in time with these algorithms, the rotating factors of two of three sub-DFTs are W^k and W^{-k}, where k is the index of outputs. (The decomposition decimation in frequency is similar to this.) We have to load the rotating factors from a lookup table, which must spend an amount of time on access to memory. The proposed radix-3 FFT algorithm, as described above, decomposes the length$-N = 3 \times 2^m$ DFT with two real constant factors -1.5 and $\sin(2\pi/3)$, which can save much time from access to the lookup table since the constant factors can be stored in the registers on processors.

2.2. Scaled radix-2/8 algorithm

A length-$N = 2^m$ DFT can be computed efficiently by using the radix-2/4 FFT [33], the radix-2/8 FFT [22], or MSRFFT [13]. However, there does not exist a published algorithm that is specialized for length$-N = 2^m$ scaled DFTs. Hence, in this sub-section, a scaled radix-2/8 FFT is presented for computing powers-of-two scaled DFTs. A length-N DFT scaled by a real constant factor s is composed of the elements:

$$X(k) = s\sum_{n=0}^{N-1} x(n)W_N^{nk}, \quad k = 0, 1, \ldots, N-1. \tag{14}$$

In the proposed algorithm, s is -1.5 or $\sin(2\pi/3)$ and length N is power-of-two. The decomposition of the proposed scaled radix-2/8 FFT provides

$$\begin{aligned} X(k) = {} & U(k) + s \times W_N^k Z_1(k) \\ & + s \times W_N^{3k} Z_3(k) + s \times W_N^{5k} Z_5(k) + s \times W_N^{7k} Z_7(k), \\ & k = 0, 1, \ldots, N-1, \end{aligned} \tag{15}$$

for the scaled DFT in Eq. (14), where

$$U(k) = s\sum_{n=0}^{N/2-1} x(2n)W_{N/2}^{nk}, \quad k = 0, 1, \ldots, N/2-1, \tag{16}$$

$$Z_1(k) = \sum_{n=0}^{N/8-1} x(8n+1)W_{N/8}^{nk}, \quad k = 0, 1, \ldots, N/8-1, \tag{17}$$

$$Z_3(k) = \sum_{n=0}^{N/8-1} x(8n+3)W_{N/8}^{nk}, \quad k = 0, 1, \ldots, N/8-1, \tag{18}$$

$$Z_5(k) = \sum_{n=0}^{N/8-1} x(8n+5)W_{N/8}^{nk}, \quad k = 0, 1, \ldots, N/8-1, \tag{19}$$

$$Z_7(k) = \sum_{n=0}^{N/8-1} x(8n+7)W_{N/8}^{nk}, \quad k = 0, 1, \ldots, N/8-1. \tag{20}$$

$U(k)$, $Z_1(k)$, $Z_3(k)$, $Z_5(k)$, and $Z_7(k)$ are five sub-DFTs of the DFT in Eq. (14). $U(k)$ is a sub-DFT scaled by the factor s. In order to share common operations, the following eight equations are required:

$$\begin{aligned} X(k) = {} & U(k) + ((s \times W_N^k Z_1(k) + s \times W_N^{5k} Z_5(k)) \\ & + (s \times W_N^{3k} Z_3(k) + s \times W_N^{7k} Z_7(k))), \\ & k = 0, 1, \ldots, N/8-1, \end{aligned} \tag{21}$$

$$\begin{aligned} X(N/2+k) = {} & U(k) \\ & - ((s \times W_N^k Z1(k) + s \times W_N^{5k} Z5(k)) \\ & + (s \times W_N^{3k} Z3(k) + s \times W_N^{7k} Z7(k))), \\ & k = 0, 1, \ldots, N/8-1, \end{aligned} \tag{22}$$

$$\begin{aligned} X(N/4+k) = {} & U(N/4+k) \\ & -j((s \times W_N^k Z_1(k) + s \times W_N^{5k} Z_5(k)) \end{aligned}$$

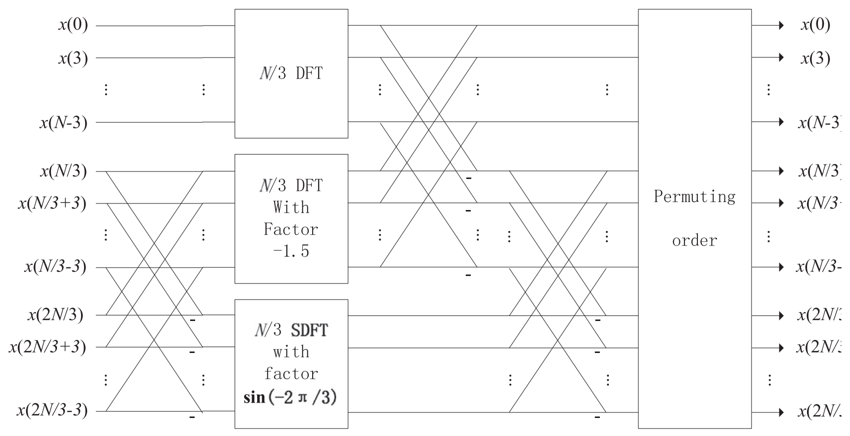

Fig. 3. General flowgraph of a DFT of length$-N = 3 \times 2^m$.

$$-(s\times W_N^{3k}Z_3(k)+s\times W_N^{7k}Z_7(k))),\quad k=0,1,\ldots,N/8-1, \tag{23}$$

$$X(3N/4+k)=U(N/4+k)+j((s\times W_N^{k}Z_1(k)+s\times W_N^{5k}Z_5(k))-(s\times W_N^{3k}Z_3(k)+s\times W_N^{7k}Z_7(k))),\quad k=0,1,\ldots,N/8-1, \tag{24}$$

$$X(N/8+k)=U(N/8+k)+(W_8^1(s\times W_N^{k}Z_1(k)-s\times W_N^{5k}Z_5(k))+W_8^3(s\times W_N^{3k}Z_3(k)-s\times W_N^{7k}Z_7(k))),\quad k=0,1,\ldots,N/8-1, \tag{25}$$

$$X(5N/8+k)=U(N/8+k)-(W_8^1(s\times W_N^{k}Z_1(k)-s\times W_N^{5k}Z_5(k))+W_8^3(s\times W_N^{3k}Z_3(k)-s\times W_N^{7k}Z_7(k))),\quad k=0,1,\ldots,N/8-1, \tag{26}$$

$$X(3N/8+k)=U(3N/8+k)-j(W_8^1(s\times W_N^{k}Z_1(k)-s\times W_N^{5k}Z_5(k))-W_8^3(s\times W_N^{3k}Z_3(k)-s\times W_N^{7k}Z_7(k))),\quad k=0,1,\ldots,N/8-1, \tag{27}$$

$$X(7N/8+k)=U(3N/8+k)+j(W_8^1(s\times W_N^{k}Z_1(k)-s\times W_N^{5k}Z_5(k))-W_8^3(s\times W_N^{3k}Z_3(k)-s\times W_N^{7k}Z_7(k))),\quad k=0,1,\ldots,N/8-1. \tag{28}$$

There are many common operations in Eqs. (21)–(28). For example, the evaluation of $X(N/2+k)$ requires only a complex subtraction, since other operations are contained in the evaluation of $X(k)$. Eqs. (21)–(28) can be further represented in the following matrix form:

$$\begin{bmatrix} X(k)\\ X(N/2+k)\\ X(N/4+k)\\ X(3N/4+k)\\ X(N/8+k)\\ X(5N/8+k)\\ X(3N/8+k)\\ X(7N/8+k)\end{bmatrix}=(\mathbf{H}_2\otimes\mathbf{I}_4)\begin{bmatrix} U(k)\\ a_e(k)\\ U(N/4+k)\\ a_o(k)\\ U(N/8+k)\\ b_e(k)\\ U(3N/8+k)\\ b_o(k)\end{bmatrix}, \tag{29}$$

where $\otimes$ represents Kronecker product, and

$$\mathbf{H}_2=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}, \tag{30}$$

$$\mathbf{I}_4=\begin{bmatrix}\mathbf{I}_2 & 0\\ 0 & \mathbf{I}_2\end{bmatrix}, \tag{31}$$

$$\mathbf{I}_2=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}. \tag{32}$$

$a_o(k)$ and $a_e(k)$ in Eq. (29) can be expressed in a matrix form:

$$\begin{bmatrix} a_e(k)\\ b_e(k)\\ a_o(k)\\ b_o(k)\end{bmatrix}=\mathbf{T}_4(\mathbf{I}_2\otimes\mathbf{H}_2)\mathbf{S}_4(\mathbf{H}_2\otimes\mathbf{I}_2)\mathbf{R}_4\begin{bmatrix} Z_1(k)\\ Z_5(k)\\ Z_3(k)\\ Z_7(k)\end{bmatrix}, \tag{33}$$

where

$$\mathbf{R}_4=\begin{bmatrix} s\times W_N^{k} & 0 & 0 & 0\\ 0 & s\times W_N^{5k} & 0 & 0\\ 0 & 0 & s\times W_N^{3k} & 0\\ 0 & 0 & 0 & s\times W_N^{7k}\end{bmatrix}, \tag{34}$$

$$\mathbf{S}_4=\begin{bmatrix} 1 & 0 & 0 & 0\\ 0 & W_8^1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & W_8^3\end{bmatrix}, \tag{35}$$

$$\mathbf{T}_4=\begin{bmatrix}\mathbf{I}_2 & 0\\ 0 & -j\mathbf{I}_2\end{bmatrix}. \tag{36}$$

The DFT $U(k)$ in Eq. (16) will be recursively decomposed if its length is greater than 4. The DFTs $Z_1(k)$, $Z_3(k)$, $Z_5(k)$, and $Z_7(k)$ in Eqs. (17)–(20) will be evaluated with the standard radix-2/8 algorithm. Let k vary from 0 to $N/8-1$, all outputs of sequence $X(k)$ can be obtained from Eq. (29), or Eqs. (21)–(28).

We now summarize the scheme that the proposed scaled radix-2/8 algorithm is used for computing a length$-N=2^m$ scaled DFT. The scaled DFT in Eq. (14) is decomposed into a scaled sub-DFT of length$-N/2$ and four general sub-DFTs of length$-N/8$. The scaled sub-DFT of length$-N/2$ will be recursively decomposed and calculated according to the scaled radix-2/8 algorithm. The four general sub-DFTs of length$-N/8$ are computed with the standard radix-2/8 FFT. The flowgraph of the proposed scaled radix-2/8 length-32 FFT is shown in Fig. 4, containing four butterflies: a special butterfly when $k=0$, a special butterfly when $k=N/16$, and two general butterflies.

3. Datapath-regular implementation of radix-2/8 FFT algorithm

From the architecture point of view, the regularity of a FFT algorithm, a property that the operations which are implemented at a certain position will appear at their counterparts, is more significant than computational complexity. Regularities of FFT algorithms can be categorized by datapath and computation [34]. The regularity of datapath is the property that signal data flows from one stage to next stage along the same path. The regularity of computation is the property that the same computations are implemented in the same circumstances. Cooley–Tukey's FFT is very regular. The radix-2/4 FFT follows Cooley–Tukey's FFT, and the radix-2/8 FFT follows the radix-2/4 FFT. Owing to higher complexity, algorithms with L-shape butterflies, such as the radix-2/4 FFT and the radix-2/8 FFT, are rarely implemented on FFT processors. Multicore parallel processors also favor those algorithms with non-L-shape butterflies. The regular structure results in faster implementation on general processors and

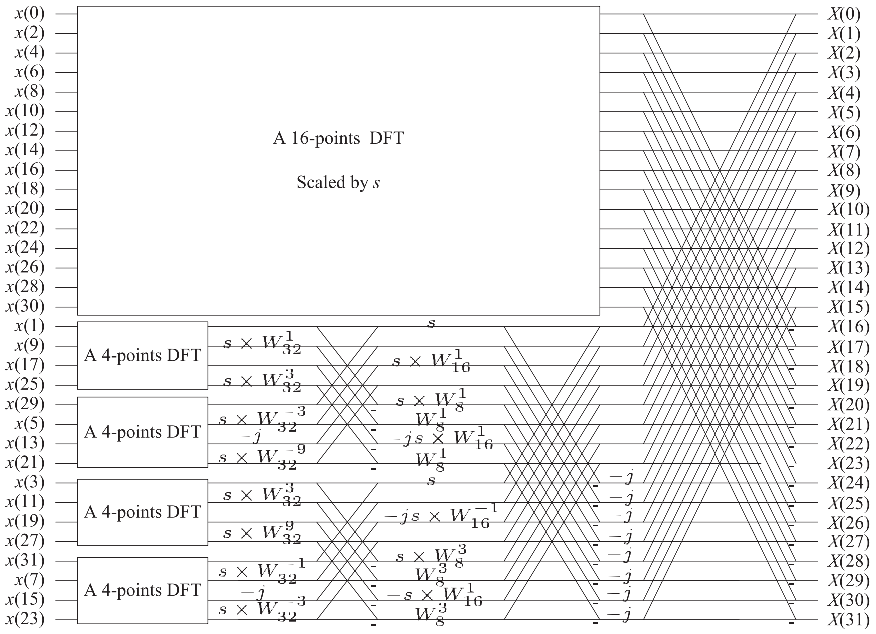

Fig. 4. Flowgraph of a length-32 DFT scaled by *s*.

simpler implementation on FFT processors. Therefore, it is interesting for the radix-2/8 FFT to be implemented more regularly. The irregularity of datapath handled by permuting the orders of intermediate data will result in a notably slower software implementation [35,36], though it is attractive for hardware implementations. In this section, we present an implementation method that makes the datapath of the radix-2/8 FFT regular like the radix-2 FFT. The method does not need to permute the order of immediate data.

3.1. Butterfly unit of two-to-two

The main idea of the proposed implementation of the radix-2/8 FFT lies in the observation that the flowgraph of the radix-2/8 FFT is the same as that of the radix-2 FFT and the difference between these two FFT algorithms lies in the locations of twiddle factors. To make datapath of the radix-2/8 FFT regular like the radix-2 FFT, each of general L-shape butterflies of the radix-2/8 FFT is divided into 4 types of 8 butterfly units of two-to-two (BU-2). A BU-2 is similar to a radix-2 FFT butterfly, with two inputs and two outputs, a complex addition, a complex subtraction, but 0, 1, or 3 complex multiplications as compared to 1 complex multiplication of a general radix-2 FFT butterfly. The first type of BUs-2 is the same as a special radix-2 FFT butterfly for the computation of length-2 DFTs, appearing at the right of the general L-shape butterfly of the radix 2/8 FFT in Fig. 5. Assume that the two inputs of a BU-2 are $x(m)$ and $x(n)$, and the BU-2 is computed in place. We give an expression of this type of BUs-2 as follows:

$$\begin{cases} X(m) = x(m) + x(n), \\ X(n) = x(m) - x(n), \end{cases} \tag{37}$$

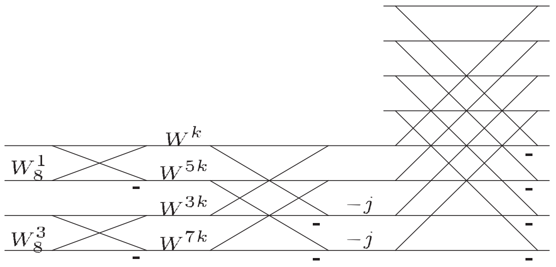

Fig. 5. A general butterfly of radix-2/8 FFT algorithm.

where $X(m)$ and $X(n)$ are the outputs of this type of BUs-2. The second type of BUs-2 is the same as another special class of radix-2 FFT butterflies, corresponding to the middle part of the general L-shape butterfly in Fig. 5. One can provide

$$\begin{cases} X(m) = x(m) + x(n), \\ X(n) = -j(x(m) - x(n)), \end{cases} \tag{38}$$

for the expression of this type of BUs-2. The third and fourth types of BUs-2 correspond to the left part of the general L-shape in Fig. 5. We can give their expressions as follows:

$$\begin{cases} X(m) = (x(m)W_8^1 + x(n))W_{4N}^{k}, \\ X(n) = (x(m)W_8^1 - x(n))W_{4N}^{5k}), \end{cases} \tag{39}$$

and

$$\begin{cases} X(m) = (x(m)W_8^3 + x(n))W_{4N}^{3k}, \\ X(n) = (x(m)W_8^3 - x(n))W_{4N}^{7k}, \end{cases} \tag{40}$$

where the variable N, which is equal to twice the number of BUs-2 contained in the BU-2 block, is the width of the BU-2 block, and the variable k is the ordinal number of the BU-2 in the BU-2 block, in the range from 0 to $N/8-1$. In Eqs. (37)–(40), the variable n is equal to $m+N/2$. A BU-2 block consists of certain a type of BUs-2, which is similar to a butterfly block of the radix-2 FFT. From Fig. 5, one can easily read all BUs-2 in a general radix-2/8 butterfly: four BUs-2 with the first type, two BUs-2 with the second type, one BU-2 with the third type, and one BU-2 with the fourth type.

3.2. BU-2 block indexing

One of the BU-2 blocks of the radix-2/8 FFT matches one of the butterfly blocks of the radix-2 FFT. However, in contrast with the fact that one class of the radix-2 FFT butterflies compose one type of radix-2 butterfly blocks, the four types of BUs-2 compose the four types of BU-2 blocks of the radix-2/8 FFT. The flowgraph of a length-16 DFT is depicted in Fig. 6, where the numbers in circles are the types of BUs-2. It is easy to read all BUs-2 and their types, and all BU-2 blocks and their widths in this figure.

In the following discussion of BU-2 block indexing, a BU-2 block of width N, consisting of certain a type of BUs-2, is named $B(N,t)$, where $t = 0, 1, 2, 3$ indicates the type of BUs-2 of the BU-2 block. As illustrated in Fig. 6, every two smaller BU-2 blocks can be referred to as a decomposition of their right bigger BU-2 block. A BU-2 block of width N is divided into two smaller BU-2 blocks of width $N/2$. Thus, the type t of a smaller BU-2 block can be determined by the BU-2 type of a bigger BU-2 block. The decomposition of BU-2 blocks of radix-2/8 FFT has the following rules:

1. A BU-2 block of width N and type $t=0$ is decomposed into two length$-N/2$ BU-2 blocks of types $t=0$ and $t=1$ respectively, i.e., $B(N,0)\rightarrow B(N/2,0)B(N/2,1)$.
2. A BU-2 block of width N and type $t=1$ is also decomposed into two smaller blocks with types $t=2$ and $t=3$ respectively, i.e., $B(N,0)\rightarrow B(N/2,2)B(N/2,3)$.
3. A BU-2 block of width N and BU-2 type $t=2$ or $t=3$ is decomposed into two length$-N/2$ BU-2 blocks with BU-2 type $t=0$, i.e., $B(N,2$ **or** $3)\rightarrow B(N/2,0)B(N/2,0)$.

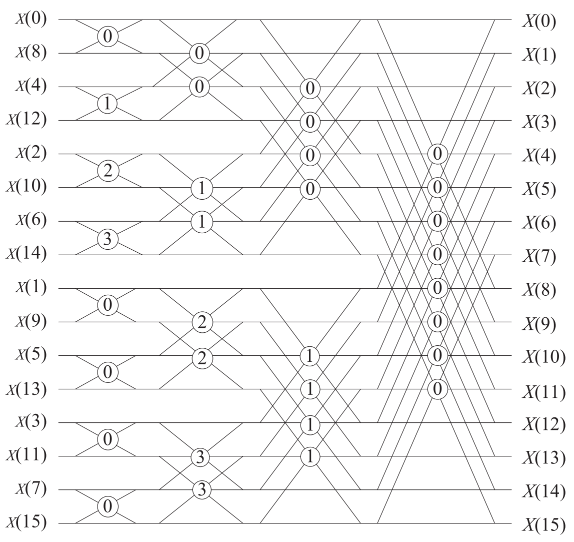

Fig. 6. Flowgraph of a length-16 DFT.

In the above description, → denotes the decomposition of BU-2 blocks. The relationship between BU-2 type and BU-2 block decomposition is illustrated in Fig. 7. The numbers in circles denote BU-2 types. For clarity, the widths of blocks are omitted.

We can obtain the BU-2 type of a BU-2 block in two ways: reading it from a table in which all BU-2 types of BU-2 blocks are pre-calculated and stored in advance, or evaluating it directly. In the way of pre-calculation and table storage, a BU-2 block needs 2 bits memory to store its BU-s type, and a length-N DFT only needs $N/16$ bytes memory to store all BU-2 blocks (only quarter BU-2 blocks require to be stored, because the BU-2 blocks in the other parts can refer to BU-2 types of these quarter BU-2 blocks). Fig. 8 shows the table to store BU-2 types of quarter BU-2 blocks of a 64-points DFT. When we want to refer to BU-2 type of the jth BU-2 block from the table, whatever stage the block is in, if $j<N/4$, the jth entry in the table is the type of the block; otherwise, the type of the block is represented by the (j mod $N/16$) th entry in the table.

In the way of the direct computation of BU-2 blocks of a length-$N=2^m$ DFT, the computation needs to iteratively go through m stages. In the ith stage, there are 2^{i-1} blocks needed to be dealt with, where $i\in[1,m]$. Set $\alpha=2^{i-2}$ (or $\alpha=2^{\lfloor \log_2^j \rfloor}$, where $\lfloor x \rfloor$ is the largest integer not greater than x), and $\beta=j$. Let $\xi(j)$ be the BU-2 type of the jth block of any stage. The $\xi(j)$ can be evaluated through the following three steps:

```
1. While ($\beta \geq 8$)
    (a) While ($\beta < \alpha$) $\alpha \leftarrow \alpha/2$;
    (b) $\alpha \leftarrow \alpha/4$;
    (c) $\beta \leftarrow \beta$ mod $\alpha$.
2. If $\beta < 4$, then $\xi(j)=\beta$.
3. If 4 $\leq \beta < 8$, then $\xi(j)=0$.
```

The way to directly compute the BU-2 type of a block does not require any extra memory. All arithmetic operations can be performed by bitwise shift operations.

There are three properties which are significant for both the pre-calculation and the direction evaluation. All BU-2 types of all BU-2 blocks can be obtained iteratively through these three following properties:

1. If $j<4$, then $\xi(j)=j$.
2. If $4\leq j<8$, then $\xi(j)=0$.
3. $\xi(j \mid n\alpha)=\xi(j)$ where n=0, 1, 2, …, or 7, and $\mid$ denotes the bitwise OR operation.

3.3. Three loops program for length-2^m DFTs

There are two methods to implement a FFT algorithm, i.e., a recursive method and an iterative method. For a FFT algorithm, the recursion provides a method for theoretical analysis and decomposition of a DFT into smaller sub-DFTs. The iteration is

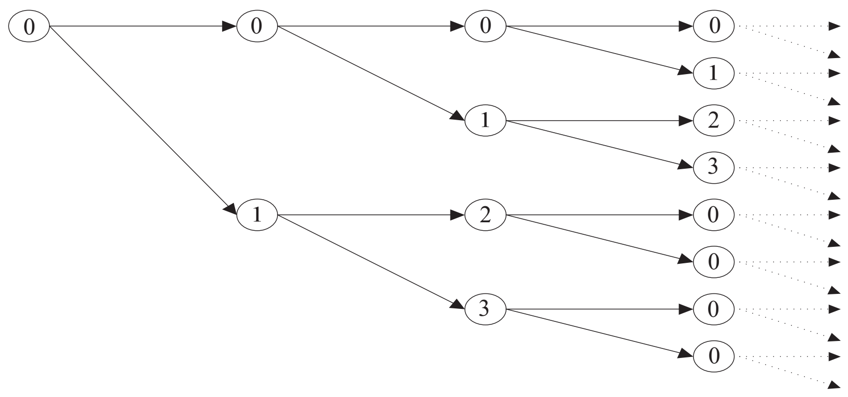

Fig. 7. Block decomposition of radix-2/8 FFT algorithm.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	2	3	0	0	0	0	0	1	0	1	0	1	0	1

Fig. 8. Table to store category identifications of a length-$N=64$ DFT.

much faster than the recursion on general processors. It is often that FFTs are implemented in the iterative method on general processors.

For the computation of a length-$N=2^m$ unscaled DFT, a three loops iterative C/C++ program has been developed and implemented. In the program, there are m stages required to be gone through. In the ith stage, the DFT is decomposed into 2^{i-1} BU-2 blocks, where $i=1,2,\dots,m$. The BU-2 types of the BU-2 blocks can be obtained by reading from a table or directly computing. What need to be done for each BU-2 block is to implement all BUs-2 according to their type. The program goes through m stages in the outer loop, implements all BU-2 blocks for every stage in the middle loop, and deals with all BUs-2 of every BU-2 block in the inner loop. Before the three loops, the coefficient table is evaluated and stored, and the input terms are permuted according to the bit reverse algorithm. If we want to get the category of a block from the lookup table, the table needs to be evaluated before the three loops in this program. Algorithm 1 shows the structure of the program. By comparing with the program in [37], it can be seen that the datapath of this program is regular.

Algorithm 1. Regular implementation of the radix-2/8 DIT FFT algorithm.

```
Require: Input sequence of the length-$N=2^m$ DFT, $x(n)$
Ensure: Output sequence of the DFT, $x(n)$
  Initiate coefficient table
  Permute the orders of intputs
  $span \leftarrow 1$
  $number\_block \leftarrow N/2$
  for $i \leftarrow m$ to 1 do
    for $j \leftarrow number\_block-1$ to 0 do
      $m \leftarrow j \times span \times 2$
      $n \leftarrow m+span$
      for $k \leftarrow 0$ to $span$-1 do
        Calculate the outputs of the BU-2 of inputs $x(m)$ and $x(n)$
          (in terms of the category of the $j$th block)
        $m \leftarrow m+1$
        $n \leftarrow n+1$
      end for
    end for
    $number\_block \leftarrow number\_block/2$
    $span \leftarrow span \times 2$
  end for
  return $x(n)$
```

For a length-2^m sub-DFT scaled by a real constant s, the three loops program is fundamentally similar to Algorithm 1, except that (1) the coefficients must be the products of the scaling factor s and the original coefficients when $j=2$ or 3 for all i, and (2) the results of BUs-2 must be multiplied by the scaling factor s for $i=1$ and $j=0$ or 1.

We now summarize the proposed implementation scheme. BU-2 and BU-2 block are two basic executing units in the proposed method. The BUs-2 of the radix-2/8 FFT are classified into four types by their multiplications, and the BU-2 blocks are categorized into four types accordingly. The BU-2 type of a BU-2 block is determined by the decomposition of a bigger BU-2 block. The BU-2 type of a BU-2 block can be calculated and stored in a table in advance before the computation of a DFT, and they can also be computed directly.

4. Performance analysis

In this section, we consider the performance of the proposed algorithm for computing a DFT of length-3×2^m by analyzing and comparing it with the algorithms reported in [2,19,21,22]. In addition, we implement the proposed algorithm in two ways (recursive implementation and iterative implementation) on an actual processor and compare their execution time.

4.1. Arithmetic complexity

Let A_N and M_N be respectively the numbers of real additions and multiplications required by a DFT of length-N. Let M_N^s be the number of real multiplications required by a scaled DFT of length-N. Assume that a complex multiplication requires four real multiplications and two real additions.

A size-$N=2^m$ sub-DFT is implemented with the standard radix-2/8 FFT algorithm discussed in Section 3.3. The numbers of operations required by the DFTs with lengths-$N=2$, 4, and 8 are given in Table 1.

When $N>8$, the decomposition of the radix-2/8 FFT consists of dividing a DFT of size N into one sub-DFT of size $N/2$ and four sub-DFTs of size $N/8$. This is achieved by $N/8-2$ general butterflies and 2 special butterflies (when $k=0$ and $k=N/16$). A general butterfly of the standard radix-2/8 FFT algorithm requires 20 real multiplications and 44 real additions. The special butterfly when $k=0$ requires 4 real multiplications and 36 real additions. When $k=N/16$, the special butterfly requires 16 real multiplications and 40 real additions. Therefore, it can be seen that the expressions of the numbers of real multiplications and real additions required by the standard radix-2/8 FFT algorithm for length-$N=2^m$ DFTs are respectively

$$M_N=\tfrac{5}{2}N-20+M_{N/2}+4M_{N/8},\quad N>8, \tag{41}$$

$$A_N=\tfrac{11}{2}N-12+A_{N/2}+4A_{N/8},\quad N>8. \tag{42}$$

A size-$N=2^m$ scaled DFT is implemented with the proposed scaled radix-2/8 FFT algorithm described in Section 2, where the required number of real additions is equal to that required by an unscaled DFT implemented with the standard radix-2/8 FFT algorithm. The general butterfly and special butterfly when $k=N/16$ of the scaled radix-2/8 FFT algorithm require the same number of real multiplications as that required by an unscaled DFT implemented with the standard radix-2/8 FFT algorithm. Four extra real multiplications are required by the scaled radix-2/8 FFT algorithm for special butterfly $k=0$. The expression of the number of real multiplications required by the scaled radix-2/8 FFT algorithm is

$$M_N^s=\tfrac{5}{2}N-16+M_{N/2}^s+4M_{N/8},\quad N>8, \tag{43}$$

where $M_1^s=2, M_2^s=4, M_4^s=8$, and $M_8^s=16$.

For a DFT of length-$N=3\times 2^m$, the implementation with the proposed algorithm contains the evaluations of a length-2^m unscaled sub-DFT, two length-2^m scaled sub-DFTs, and $N/3$ 3-points DFTs. A 3-points DFT requires only 12 real additions (it is different from the general 3-points FFT algorithm). Thus, the arithmetic complexity required by the proposed algorithm for computing a DFT of length-$N=3\times 2^m$ is

$$M_N=M_{N/3}+2\times M_{N/3}^s,\quad N>3, \tag{44}$$

$$A_N=4N+3A_{N/3}.\quad N>3. \tag{45}$$

The arithmetic complexity of the proposed algorithm and the algorithms reported in [21,22] are given in Table 2. From the table, it can be seen that the proposed algorithm saves floating-point operations compared with the algorithms in [21,22] when lengths are greater than 24. Besides the special case when the lengths of the sub-DFTs are 24, the algorithms in [21,22] have to multiply every input/output elements of corresponding scaled sub-DFTs by scaling factors, which requires slightly less than 2^{m+2} real multiplications. However, the scaled sub-DFT is implemented with the proposed scaled radix-2/8 FFT, and the scaling factor s is attached the twiddle factors of the FFT, which requires only $8m$ real multiplications. Thus, we can see that the proposed algorithm reduces slightly less than $2^{m+2}-8m$ real multiplications for the length-3×2^m DFT compared to the algorithms in [21,22].

Two comparisons of the proposed algorithm and the radix-3/6 FFT in [19] are given in Tables 3 and 4 respectively. Table 3 compares the computational complexity for length$-N=6^m$ DFTs. Table 4 compares the computational complexity for length$-N=3\times 2^m$ DFTs. From these two tables, one can see that the radix-3/6 FFT is suitable for length$-N=6^m$ DFTs, and the proposed algorithm is more efficient than the radix-3/6 FFT for computing the length$-N=3\times 2^m$ DFTs. Total, as compared to our previous paper [19], the proposed algorithm saves the number of operations by $2N-8m-16$ for the length$-N=3\times 2^m$ DFT. As compared to our previous paper [20], the proposed algorithm optimizes the number of data transfers, address generations, and twiddle factor evaluations or accesses to the lookup table. Further, the proposed algorithm achieve the computational complexity being equal to that of [20] when the powers-of-two sub-DFTs in [20] are implemented with the radix-2/4 FFT. In addition, the proposed algorithm

Table 1
Arithmetic complexity of 2-, 4-, and 8-points DFTs.

N	Multiplications	Additions
2	0	4
4	0	16
8	4	52

Table 2
Arithmetic complexity for $N=3\times 2^m$.

N	Algorithm in [22]			Algorithm in [23]			Proposed algorithm		
	M_N	A_N	Total	M_N	A_N	Total	M_N	A_N	Total
24	36	252	288	36	252	288	36	252	288
48	128	624	752	128	624	752	104	624	728
96	356	1500	1856	372	1500	1872	292	1500	1792
192	960	3504	4464	936	3528	4464	768	3528	4296
384	2404	8028	10 432	2348	8100	10 448	1964	8100	10 064
768	5824	18 096	23 920	5696	18 288	23 984	4840	18 288	23 128
1536	13 668	40 284	53 952	13 220	40 812	54 032	11 508	40 812	52 320
3072	31 424	88 752	120 176	30 232	90 072	120 304	26 768	90 072	116 840
6144	71 012	193 884	264 896	68 316	196 980	265 296	61 180	196 980	258 160
12 288	158 400	420 528	578 928	151 856	427 776	579 632	137 592	427 776	565 368
24 576	349 540	906 588	1 256 128	334 164	923 196	1 257 360	305 732	923 196	1 228 928

Table 3
Computational complexity for length-$N = 6^m$ DFTs with $m \in (l, l+2)$ [19].

N	In [19]			Proposed algorithm		
	M_N	A_N	Total	M_N	A_N	Total
$4 \times 9 = 36$	128	464	592	128	464	592
$8 \times 9 = 72$	276	1092	1368	260	1108	1368
$4 \times 27 = 108$	656	1952	2608	656	1952	2608
$8 \times 27 = 216$	1340	4364	5704	1316	4444	5760
$16 \times 27 = 432$	2912	9768	12 680	3168	9656	12 824
$16 \times 81 = 648$	5668	16 468	22 136	5636	15 796	21 432
$32 \times 81 = 1296$	11 704	35 824	47 528	12 888	35 472	48 360
$64 \times 81 = 2592$	25 268	78 100	103 368	28 372	76 948	105 320
$16 \times 243 = 3888$	44 720	127 576	172 296	48 960	129 476	178 436
$32 \times 243 = 7776$	92 796	273 164	365 960	105 700	278 396	384 096

Table 4
Computational complexity for length-$N = 3 \times 2^m$ DFTs.

Algorithm	Total
Radix-3/6 FFT in [19]	$4Nm+8$
Proposed algorithm	$4Nm-2N+8m+24$

reduces the power of quantization noise as compared with the algorithms in [20,21,22]. However, considering the readers of signal processing, the discussion does not appear in this paper.

4.2. Execution time

Execution time of a DFT is determined by many factors, such as arithmetic complexity, accesses to lookup table, hardware architecture, and iterative or recursive implementation. The proposed algorithm is implemented in two ways, a recursive method and an iterative method. We compare their execution times with FFTW and the algorithms in [21,22]. The comparison of the recursive implementation is conceptive. The iterative implementation has real-life meaning. The recursive programs are performed repeatedly 1000 times on processor E6700 (32 kB data cache and 32 kB code cache respectively with 8 ways 64 bytes lines). The execution times are shown in Table 5. It can be seen that the proposed algorithm conceptively consumes less execution time than the algorithms in [21,22].

The iterative algorithm and FFTW are performed repeatedly 100 times on CPU E6700 with Studio 2005 in Windows 7. FFTW runs in two models: MEASURE and ESTIMATE. (MEASURE model will optimize its program for the next run by measuring the performance of FFTW with different parameters in the real-life environment. ESTIMATE model configures the parameters of program by estimating.) Each implementation of both FFTW and the proposed algorithm must load the inputs. If FFTW loads the inputs only once for the 100 repeated executions, its execution time will be reduced rapidly. Execution time of the first run of FFTW is viewed as the initiating time and is not contained into the total execution time. Time to initiate the two tables (storing block categories and coefficients) has been included into execution time of the proposed algorithm. Time to free and destroy memory and objects is ignored. Owing to each experiment in FFTW's execution a different execution time, the execution time of FFTW with MEASURE model is counted by its smallest value. Table 6 shows the execution time of FFTW and the proposed algorithm. From this table, we can see that the proposed algorithm consumes less time than FFTW when N is small. When N is large, the proposed algorithm consumes execution time that is between those spent by MEASURE and ESTIMATE model of FFTW. The reason why FFTW with MEASURE model consumes less time when N is large lies in that FFTW can adapt to hardware to maximize performance but our program cannot. To allow readers to understand how the proposed algorithm saves the computation time, we have provided the c/c++ code as supplemental material for readers' read and comparison. The code is developed with Microsoft Visual Studio 2008 on Windows 7, and requires the support of version 3.3 of FFTW. The code contains three main functions:

Table 5
Execution time of recursive implementation (repeatedly 1000 times).

N	In [21]	In [22]	Proposed
192	0.391	0.368	0.323
384	0.891	0.857	0.822
768	2.098	1.988	1.892
1536	4.686	4.537	4.177
3072	10.412	10.148	9.230
6144	22.941	22.313	20.515
12 288	50.201	49.171	45.027
24 576	108.884	106.620	98.409
49 152	237.600	234.000	213.400

Table 6
Execution time repeatedly 100 times.

N	Proposed	FFTW with	
		MEASURE	ESTIMATE
3×256	0.000	≥ 0.015	0.047
3×512	0.015	≥ 0.031	0.078
3×1024	0.031	0.047	0.078
3×2048	0.078	≥ 0.078	0.124
3×4096	0.156	≥ 0.078	0.188

- program3x2mTest(): test the validity of the program.
- Execute_FFTW(): test time of FFTW on 3×2^m DFTs.
- Executing_time(): test time of the proposed algorithm on 3×2^m DFTs.

5. Conclusions

In this paper, we have proposed an algorithm for computing DFTs of length-$N = 3 \times 2^m$. We have also presented a method to implement the radix-2/8 FFT algorithm and the proposed algorithm. The algorithm divides decimation-in-frequency the DFT into three length$-N/3$ sub-DFTs. The two length$-N/3$ sub-DFTs scaled by constant factors are evaluated by the proposed scaled radix-2/8 algorithm. The proposed algorithm reduces the number of real multiplications compared with the related algorithms. The implementation method makes datapath of the radix-2/8 FFT algorithm regular like the radix-2 FFT algorithm, and gives theoretic support to its implementation in both hardware and software. Experimental results show that the proposed algorithm consumes less execution time. The idea of the proposed algorithm can be applied to a DFT of length-$q_1 \times q_2$, reducing its number of real operations, where q_1 and q_2 are co-prime with each other. The proposed implementation method can be applied to other algorithms with L-shape butterflies, making their datapath regular like fixed-radix FFT algorithms.

Acknowledgments

The authors are grateful to anonymous reviewers for their suggestions to improve the paper. This research was partially funded by the Key Program of National Natural Science Foundation of China (Grant nos. 61133005 and 61432005), and the National Natural Science Foundation of China (Grant nos. 61070057, 61173013, 61472136, 61370095, and 61472124), Ministry of Education of China (Grant no. 708066), the Ph.D. Programs Foundation of Ministry of Education of China (20100161110019), Project supported by the National Science Foundation for Distinguished Young Scholars of Hunan (12JJ1011).

Appendix A. Supplementary data

Supplementary data associated with this article can be found in the online version at http://dx.doi.org/10.1016/j.sigpro.2015.01.008.

References

[1] P. Duhamel, M. Vetterli, Fast Fourier transforms: a tutorial review and a state of the art, Signal Process. 19 (4) (1990) 259–299.
[2] M. Frigo, S. Johnson, The design and implementation of FFTW3, Proc. IEEE 93 (2) (2005) 216–231.
[3] G.S. Engel, T.R. Calhoun, E.L. Read, T.-K. Ahn, T. Man cal, Y.-C. Cheng, R.E. Blankenship, G.R. Fleming, Evidence for wavelike energy transfer through quantum coherence in photo synthetic systems, Nature 446 (7137) (2007) 782–786.
[4] G. Bi, S.K. Mitra, S. Li, Sampling rate conversion based on DFT and DCT, Signal Process. 93 (2 (February)) (2013) 476–486.
[5] J. Cooley, J. Tukey, An algorithm for the machine calculation of complex Fourier series, Math. Comput. 19 (90) (1965) 297–301.
[6] P. Duhamel, H. Hollmann, Split-radix FFT algorithm, Electron. Lett. 20 (January) (1984) 14–16.
[7] F. Taylor, G. Papadourakis, A. Skavantzos, A. Stouraitis, A radix-4 FFT using complex RNS arithmetic, IEEE Trans. Comput. 100 (6) (1985) 573–576.
[8] I. Kamar, Y. Elcherif, Conjugate pair fast Fourier transform, Electron. Lett. 25 (5 (April)) (1989) 324–325.
[9] C.-P. Fan, G.-A. Su, Pruning fast Fourier transform algorithm design using group-based method, Signal Process. 87 (11) (2007) 2781–2798.
[10] T.-Y. Sung, H.-C. Hsin, Y.-P. Cheng, Low-power and high-speed CORDIC-based split-radix FFT processor for OFDM systems, Digit. Signal Process. 20 (2) (2010) 511–527.
[11] S. Bouguezel, M.O. Ahmad, M.N.S. Swamy, A general class of split-radix FFT algorithms for the computation of the DFT of length-2^m, IEEE Trans. Signal Process. 55 (8 (August)) (2007) 4127–4138.
[12] G. Bi, G. Li, X. Li, A unified expression for split-radix DFT algorithms, in: IEEE International Conference on Communications, Circuits and Systems (ICCCAS), 2010, pp. 323–326.
[13] S.G. Johnson, M. Frigo, A modified split-radix FFT with fewer arithmetic operations, IEEE Trans. Signal Process. 55 (1) (2007) 111–119.
[14] W. Zheng, K. Li, K. Li, Scaled radix-2/8 algorithm for efficient computation of length$-N = 2^m$ DFTs, IEEE Trans. Signal Process. 62 (MAY) (2014) 2492–2503.
[15] C. Rader, Discrete Fourier transforms when the number of data samples is prime, Proc. IEEE 56 (6) (1968) 1107–1108.
[16] S. Winograd, On computing the discrete Fourier transform, Math. Comput. 32 (141) (1978) 175–199.
[17] Y. Suzuki, T. Sone, K. Kido, A new FFT algorithm of radix 3, 6, and 12, IEEE Trans. Acoust. Speech Signal Process. ASSP-34 (2 (April)) (1986) 380–383.
[18] C. Temperton, A new set of minimum-add small-N rotated DFT modules, J. Comput. Phys. 75 (1) (1988) 190–198.
[19] W. Zheng, K. Li, Split-radix algorithm for length 6^m DFT, IEEE Signal Process. Lett. 20 (2013) 713–716.
[20] W. Zheng, Kenli Li, Keqin Li, A fast algorithm based on SRFFT for length $N = q \times 2^m$ DFTs, IEEE Trans. Circuits Syst. II: Express Briefs 61 (2013) 110–114.
[21] G. Bi, Fast algorithms for DFT of composite sequence lengths, Signal Process. 70 (2) (1998) 139–145.
[22] S. Bouguezel, M. Ahmad, M. Swamy, A new radix-2/8 FFT algorithm for length$-q \times 2^m$ DFTs, IEEE Trans. Circuits Syst. I: Regul. Pap. 51 (9) (2004) 1723–1732.
[23] K. Maharatna, A.S. Dhar, S. Banerjee, A VLSI array architecture for realization of DFT, DHT, DCT and DST, Signal Process. 81 (9) (2001) 1813–1822.
[24] M. Frigo, C. Leiserson, H. Prokop, S. Ramachandran, Cache oblivious algorithms, in: 40th Annual Symposium on Foundations of Computer Science, 1999, IEEE, New York, pp. 285–297.
[25] L. Jeesung, L. Hanho, A high-speed two-parallel radix-2^4 FFT/IFFT processor for MB-OFDM UWB systems, IEICE Trans. Fundam. Electron. Commun. Comput. Sci. 91 (4) (2008) 1206–1211.
[26] B. Jo, M. Sunwoo, New continuous-flow mixed-radix (CFMR) FFT processor using novel in-place strategy, IEEE Trans. Circuits Syst. I: Regul. Pap. 52 (5) (2005) 911–919.
[27] C. Hsiao, Y. Chen, C. Lee, A generalized mixed-radix algorithm for memory-based FFT processors, IEEE Trans. Circuits Syst. II: Express Briefs 57 (1) (2010) 26–30.
[28] D. Cohen, Simplified control of FFT hardware, IEEE Trans. Acoust. Speech Signal Process. 24 (6) (1976) 577–579.
[29] E. Sejdić, I. Djurović, L. Stanković, Fractional Fourier transform as a signal processing tool: an overview of recent developments, Signal Process. 91 (6) (2011) 1351–1369.
[30] D. Bhandarkar, Analysis of memory interference in multiprocessors, IEEE Trans. Comput. 100 (9) (1975) 897–908.
[31] D. Harper III, Increased memory performance during vector accesses through the use of linear address transformations, IEEE Trans. Comput. 41 (2) (1992) 227–230.
[32] D. Reisis, N. Vlassopoulos, Conflict-free parallel memory accessing techniques for FFT architectures, IEEE Trans. Circuits Syst. I: Regul. Pap. 55 (11) (2008) 3438–3447.
[33] R. Yavne, An economical method for calculating the discrete Fourier transform, in: Proceedings of the AFIPS Fall Joint Computer Conference, vol. 33, 1968, pp. 115–125.

[34] L. Yuan, X. Tian, Y. Chen, Pruning split-radix fft with time shift, in: IEEE International Conference on Electronics, Communications and Control (ICECC), 2011, pp. 1581–1586.

[35] P. Duhamel, Implementation of split-radix FFT algorithms for complex, real, and real symmetric data, IEEE Trans. Acoust. Speech Signal Process. 34 (2 (April)) (1986) 285–295.

[36] J. Kwong, M. Goel, A high performance split-radix FFT with constant geometry architecture, in: Design, Automation & Test in Europe Conference & Exhibition (DATE), 2012. IEEE, Dresden, pp. 1537–1542.

[37] H. Sorensen, M. Heideman, C. Burrus, On computing the split-radix FFT, IEEE Trans. Acoust. Speech Signal Process. 34 (1) (1986) 152–156.