A Grouped Fast Fourier Transform Algorithm Design For Selective Transformed Outputs

Chih-Peng Fan * and Guo-An Su
Department of Electrical Engineering, National Chung Hsing University,
250 Kuo-Kuang Road, Tai-chung, Taiwan, R.O.C.
cpfan@dragon.nchu.edu.tw *

Abstract - **In this paper, the grouped scheme is specially applied to compute the fast Fourier transform (FFT) when the portions of transformed outputs are calculated selectively. The grouped FFT algorithm applies the scheme of the grouped frequency indices to accelerate the computation of selected DFT outputs. The advantage of the grouped FFT algorithm is that it is more cost-effective than the convenient FFT algorithms when we need to compute parts of the transformed outputs, not all outputs. For computing all transformed outputs of the DFT, the computational complexity of the proposed FFT method is less than that of the radix-2 method. Meanwhile, the computational complexity of the proposed fast method approximates to that of the radix-4 FFT algorithm. By sharing coefficients of the twiddle factors in the same frequency group, the grouped FFT can be implemented with hardware sharing VLSI architectures.**

Keywords— **Fast Fourier Transform, Grouped Scheme, Selective Transformed Outputs**

I. INTRODUCTION

Discrete Fourier transform (DFT) has been the arithmetic transformation method for many scientific and engineering applications [1]. For example, we can use the DFT for the analysis, design, and implementation of discrete-time signal processing algorithms, and then we also use the DFT properties to inspect the spectrum of communication signals. If a direct DFT method is applied to calculate the complex input signals of the length N, it needs N^2 loops, about $4N^2$ additions and multiplications. Therefore, the costs of the computation and VLSI design of the DFT become larger when the transformed length N is large.

For computing the DFT with lower computational complexity, Cooley and Tukey [1] developed the radix-2 FFT algorithm in 1965. Among different FFT approaches, the fixed radix and the split radix methods are two widely used approaches, which provide the properties of regular computational structures. In the fixed radix-2 FFT algorithm, the radix-2 index mapping is used to divide the transform outputs into the even and odd parts. In the split radix-2/4 FFT algorithm, the first stage of butterfly computations uses the radix-2 method to obtain both even and odd index terms, and then the second-stage computation uses the radix-4 index mapping to factor the odd index terms.

In this paper, the grouped algorithm is specially proposed to compute the FFT efficiently when the portions of transformed outputs are calculated selectively. The rest of the paper is organized as follows. In Section II, we use an index description method to characterize the transform bases. Based on the indices of bases [2], the outputs of the DFT transformation can be pre-group together so that the grouped outputs can utilize the same twiddle factors with the same grouped butterfly. After using pre-grouped method, the fast computational scheme can reduce the computation complexity. In Section III, we describe the comparisons of the computational complexity for computing all transform outputs and selective grouped transform outputs. Finally we give a conclusion in Section IV.

II. PROPOSED SELECTIVE FFT ALGORITHM

Firstly, we describe the conventional DFT expression, which is rewritten with the "*mod*" operation, is shown in the following.

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

where $W_N^{nk} = e^{-j(2\pi/N)nk}$ and $n_k = n \cdot k \bmod N$. The base coefficient, W_N^{nk}, is often referred to the twiddle factor. In (1), the length of the input sequence N is a number of power of two, and k is the frequency index of the DFT, where k=0, 1, 2, … , N-1 and n=0,1, 2, … ,N-1. We use the transform index n_k to express the power coefficient of the twiddle factor $W_N^{nk \bmod N}$. In order to develop the grouped and selective FFT algorithm, we derive the whole algorithm from two parts: the first part refers to the grouped permutation process and the second part is the fast computational process. The derivation of the proposed fast method is described in the following subsections. In (1), we need N different twiddle factors to compute the whole DFT outputs. By evaluating all transforms outputs, the N twiddle factors are repeated with the same transform kernel. According to the same twiddle factors in the same group, we develop a grouped permutation process, and the permutation process can reduce the original length of the input sequence and the number of multiplications. Before the development of the proposed FFT algorithm, we group the

DFT transform outputs $X[k]$ by applying the same twiddle factors to compute (1). For developing the grouped FFT algorithm, we use the mathematical model, which is described shortly as follows. The divisible factors of N in the increasing order are shown as $\{d_0, d_1, d_2, ..., N\}$. After we take d_i mod N, we can give the following set,

$$S_B = \{d_0, d_1, d_2, ..., d_i\}. \quad (2)$$

Since N is power of 2, the last component in the set S_B is $N/2$. For each component in the set S_B, we define the corresponding kernel number as follows,

$$M_i = \frac{N}{d_i}, \text{ for } i \neq 0. \quad (3)$$

In (3), we define the case that $M_0 = N/P$ at $i = 0$. Next, we define the G_i to include the range from 0 to N-1. We define $G_0 = \{0\}$ at i=0. When $i \neq 0$, we assign the i-th kernel group of frequency indices to be defined as follows,

$$G_i = d_i \cdot R_i = \{d_i r_1, d_i r_2, d_i r_3, ..., d_i r_i\}, \quad (4)$$

where

$$R_i = \{r_1, r_2, r_3, ..., r_i\}. \quad (5)$$

In (5), the components of the R_i set are relatively prime with the kernel number M_i and the range of r_i is $0 < r_i < M_i$. Furthermore, the properties of (4) include $G_i \cap G_j = \phi$ and $\Sigma G_i = \{0, 1, 2, ..., N-1\}$. In (5), $r_i \neq r_j$, for $i \neq j$. After we obtain all frequency indices from all of the G_i sets, we can design the grouped FFT algorithm. From (4), we define that $k = r_{ik} \cdot d_i$, and substitute $r_{ik} \cdot d_i$ for n_k in (1) to obtain

$$n_k = n \cdot r_{ik} \cdot d_i \bmod N, \quad (6)$$

where d_i is the divisible factor of N, and the r_{ik} is one of the elements in the R_i set. Then, we replace (1) with (6) to obtain,

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{n \cdot r_{ik} \cdot d_i \bmod N}. \quad (7)$$

Let us define that

$$m_k = \frac{n_k}{d_i} = r_{ik} \cdot n \bmod \frac{N}{d_i} = r_{ik} \cdot n \bmod M_i, \quad (8)$$

where the kernel number M_i is defined in (3), and then (7) becomes as follows,

$$X[k] = \sum_{n=0}^{N-1} x[n] W_{N/d_i}^{n \cdot r_{ik} \bmod (N/d_i)}, = \sum_{n=0}^{N-1} x[n] W_{M_i}^{n \cdot r_{ik} \bmod M_i}. \quad (9)$$

where $W_{M_i}^{n \cdot r_{ik} \bmod M_i} = e^{-j2\pi (n \cdot r_{ik} \bmod M_i)/M_i}$.

In (9), by evaluating the frequency indices k in the same group G_i, we only require the same value of the M_i kernel base. Based on the grouped frequency indices, the length of the input sequence, N, can be segmented into N/M_i sub-sequences, where the length of each sub-sequence is M_i. Let us define that $n = q \cdot M_i + p$, where $q = 0 \sim N/M_i - 1$ and $p = 0 \sim M_i - 1$. Then, the (9) becomes

$$X[k] = \sum_{q=0}^{\frac{N}{M_i}-1} \sum_{p=0}^{M_i-1} x[qM_i + p] W_{M_i}^{(qM_i+p) \cdot r_{ik} \bmod M_i} = \sum_{p=0}^{M_i-1} x_k(p) W_{M_i}^{p \cdot r_{ik} \bmod M_i} = \sum_{p=0}^{M_i-1} x_k(p) W_{M_i}^{\frac{p}{d_i} \cdot k \bmod M_i}, \quad (10)$$

where

$$x_k(p) = \sum_{q=0}^{N/M_i-1} x[qM_i + p] \quad (11)$$

The expression in (10) shows the result of the permutation pre-process and indicates that the computation length of input sequences can be reduced from N to M_i, where M_i is equal to or less than N. We note that the pre-treatment in $x_k(p)$ only uses pure additional operations. For accelerating the implementation of $x_k(p)$ in (11), we use the parallel additions as follows,

$$x_k(p) = \sum_{q=0}^{N/4M_i-1} x[qM_i + p] + \sum_{q=N/4M_i}^{N/2M_i-1} x[qM_i + p] + \sum_{q=N/2M_i}^{3N/4M_i-1} x[qM_i + p] + \sum_{q=3N/4M_i}^{N/M_i-1} x[qM_i + p], \quad (12)$$

$$x_k(p) = \sum_{q=0}^{N/4M_i-1} \{(x[qM_i + p] + x[qM_i + p + N/2]) + (x[qM_i + p + N/4] + x[qM_i + p + 3N/4])\}. \quad (13)$$

By applying the folded input sequence in (13), the data flow diagram is shown in Figure 1.

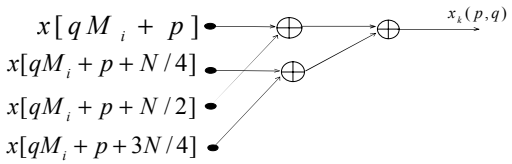

Figure 1 The Implementation of $x_k(p)$

In Figure 1, the kernel number M_i is different when the different group G_i and the transformed length N are given. Therefore, the different architectures for parallel inputs $x[n]$ can be used to trade off the computing performance and the chip area. After the input sequences $x[n]$ are dealt with the computing scheme in Figure 1, the data length of each grouped transformation is reduced from N to M_i with (10).

After permutation process in (11), the design with the regular and in-place address properties, which can save the

use of the memory in VLSI implement, is required for achieving the proposed FFT algorithm. So we use the FFT structure to develop the proposed method. Next, we give an example for explaining how we group the whole frequency indices k_i in the same group G_i. When *N*=16, we give $S_B=\{0,1,2,4,8\}$, $M_i=\{N/P,16,8,4,2\}$, $G_0=\{0\}$, $G_1=\{1,3,5,7,9,11,13,15\}$, $G_2=\{2,6,10,14\}$, $G_3=\{4,12\}$, and $G_4=\{8\}$. In the group G_1, we can employ $k=d_i(4k'+1)$ to denote the even part $\{1,5,9,13\}$ in the G_1 and apply $k=d_i(4k'+3)$ to describe the odd part $\{3,7,11,15\}$ in the same group. For the other groups, we use the same method to separate the even and odd parts. We note that the groups of G_0 and G_4 use only the addition and subtract operations to achieve the transform outputs of *X*[0] and *X*[8].

Let us define that $k=d_i(4k'+1)$, where $k'=0,1,2,...,M_i/4-1$ for k_i is even in a group. Substituting $k=d_i(4k'+1)$ for (11), then (11) can be rewritten as follows,

$$X[d_i(4k'+1)]=\sum_{p=0}^{M_i-1}x_k(p)W_{M_i}^{\frac{p}{d_i}d_i(4k'+1)\bmod M_i}$$
$$=\sum_{p=0}^{M_i/4-1}x_k(p)W_{M_i}^{p\cdot(4k'+1)\bmod M_i}+\sum_{p=M_i/4}^{M_i/2-1}x_k(p)W_{M_i}^{p\cdot(4k'+1)\bmod M_i}$$
$$+\sum_{p=M_i/2}^{3M_i/4-1}x_k(p)W_{M_i}^{p\cdot(4k'+1)\bmod M_i}+\sum_{p=3M_i/4}^{M_i-1}x_k(p)W_{M_i}^{p\cdot(4k'+1)\bmod M_i} \quad (14)$$

By using the periodic property of twiddle factor $W_{M_i}^{p\cdot(4k'+1)\bmod M_i}$ to simplify each term in the right hand of (14), then the (14) becomes

$$X[d_i(4k'+1)]=\sum_{p=0}^{M_i/4-1}\{[x_k(p)-x_k(p+M_i/2)]$$
$$-j\cdot[x_k(p+M_i/4)-x_k(p+3M_i/4)]\}W_{M_i}^{p}\cdot W_{M_i/4}^{p\cdot k'}. \quad (15)$$

The (15) can be used to compute the even part in the same group G_i, and the length of the folded input sequences can be reduced from M_i to $M_i/4$. According to the similar procedures described in (15), we define that $k=d_i(4k'+3)$, where $k'=0,1,2,...,M_i/4-1$, and k_i is odd in a group. Then we have

$$X[d_i(4k'+3)]=\sum_{p=0}^{M_i/4-1}\{[x_k(p)-x_k(p+M_i/2)]$$
$$+j\cdot[x_k(p+M_i/4)-x_k(p+3M_i/4)]\}W_{M_i}^{3p}\cdot W_{M_i/4}^{p\cdot k'}. \quad (16)$$

The (16) can be used to compute the odd part in the same group G_i, and the length of the folded input sequences can also be reduced from M_i to $M_i/4$. According to (15) and (16), we can accomplish the data flow diagram of (15) and (16) in the following diagram,

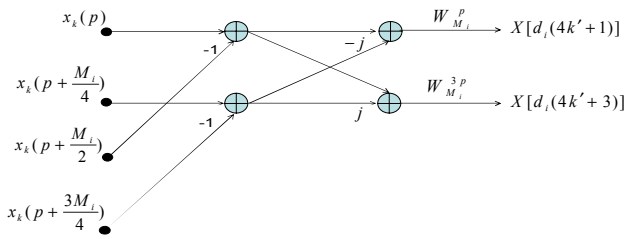

Figure 2 The data flow diagram for fast computations

After the input sequences $x_k(p)$ are dealt with the stage in Figure 2, the transformed length of each grouped outputs is reduced from M_i to $M_i/4$. In order to reduce the numbers of multiplications and accelerate the implementation for the group G_1, we apply the further fast algorithm in the following,

$$X[d_i(8k'+1+2b+4c)]=\sum_{p=0}^{M_i/8-1}\{[(x_k(p)-x_k(p+M_i/2))$$
$$-j\cdot W_{M_i}^{M_i\cdot b/2}(x_k(p+M_i/4)-x_k(p+3M_i/4))]$$
$$+W_{M_i}^{M_i/8}[W_{M_i}^{M_i\cdot(b+2c)/4}(x_k(p+M_i/8)-x_k(p+5M_i/8))$$
$$-j\cdot W_{M_i}^{3M_i\cdot(b+2c)/4}(x_k(p+3M_i/8)-x_k(p+7M_i/8))]\}$$
$$\cdot W_{M_i}^{p\cdot(1+2b+4c)}\cdot W_{M_i/8}^{p\cdot k'} \quad (17)$$

In (17), when the parameter (*b*,*c*)=(0,0), (*b*,*c*)=(0,1), (*b*,*c*)=(1,0) and (*b*,*c*)=(1,1), the results express the FFT scheme for $X[d_i(8k'+1)]$, $X[d_i(8k'+5)]$, $X[d_i(8k'+3)]$ and $X[d_i(8k'+7)]$, respectively. The paths of $X[d_i(8k'+1)]$ and $X[d_i(8k'+5)]$ indicate the even and odd sub-groups in the even part of the group G_1, and then the paths of $X[d_i(8k'+3)]$ and $X[d_i(8k'+7)]$ indicate the even and odd sub-groups in the odd part of the group G_1.

We show the whole data flow diagram of the proposed implementation in Figure 3. The grouped permutation process only uses the addition operations, and the different groups correspond to the different kernel number M_i, which leads to the different shortened sequences $x_k(p)_{M_i}$. For computing the transform outputs of *X*[0] and *X*[*N*/2], our design only needs additions or subtractions, and does not need the grouped process. In the proposed FFT computational scheme, the different output groups need the different shortened transformation lengths and the different computational complexities.

III. Comparisons of Computational Complexity

We show the comparison results for computational complexity, which are referred to the total and average numbers of complex multiplications among the proposed FFT structure and the other existing FFT structures.

Table I Total Numbers of Complex Multiplications for the Different FFT Algorithms with Length N

Transform Length (N)	Radix-2[3]	Radix-4 [3]	Proposed Method
64	83	66	61
128	277	---	168
256	579	450	443
512	1411	---	1118
1024	3331	2562	2721
2048	7683	---	6436
4096	17411	13314	14887
8192	38915	---	33834

Table II Average Numbers of Complex Multiplications per Output for the Different FFT Algorithms

Average numbers of complex multiplications(N=256)	Radix-2 [3]	Radix-4 [3]	Proposed Method
Group G_1 (128)	2.26	1.76	2.06
Group G_2 (64)			1.79
Group G_3 (32)			1.34
Group G_4 (16)			0.93
Group G_5 (8)			0.62
Group G_6 (4)			0.25

In the Table I, the computational complexity of the proposed FFT method is less than that of the radix-2 method, and then the computational complexity of the proposed fast method is less than that of the radix-4 FFT when the transformed length is less than 1024. In the Table II, the notation of group G_1 (128) indicates that there are the number of 128 frequency indices in the group G_1. The average numbers of complex multiplications in the radix2 and radix4 are evaluated by using the total numbers of complex complications to be divided by the transformed length N. The average numbers in the proposed method are obtained by dividing the total complex complications in all of the transform outputs in the same group by the numbers of the transform outputs in the same group. In Table II, the average computational complexity in the proposed FFT algorithm is less than that in the radix-2 method, but more than that in the radix-4 FFT algorithm in the group G_1 computations.

IV. Conclusion

In this paper, the grouped algorithm is specially employed to realize the FFT when the portions of transformed outputs are calculated selectively. The grouped FFT algorithm applies the scheme of the grouped frequency indices to accelerate the computation of selected outputs of the DFT. The advantage of the proposed FFT algorithm is that it acts more efficient than the convenient FFT algorithms when we need to compute parts of the transformed outputs, not all outputs. Meanwhile, for computing all transformed outputs, the computational complexity of the proposed FFT scheme approximates to that of the radix-4 FFT algorithm, and is less than that of the radix-2 FFT. For implementations, the proposed FFT algorithm is more regular than split radix FFT algorithms.

Acknowledgement

This work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC95-2220-E-005-006.

References

[1] Alan V. Oppenheim and Ronald W. Schafer, *Discrete Time Signal Processing*, Prentice Hall, 1989

[2] Chih-Peng Fan and Guo-An Su, "Novel Recursive Discrete Fourier Transform with Compact Architecture", IEEE APCCAS, vol. 2, pp. 1081-1084, 6-9 Dec., 2004.

[3] Daisuke Takahashi, "An Extended Split-Radix FFT Algorithm", IEEE Signal Processing Letters, vol. 8, no. 5, pp145-147, May 2001.

[4] Saad Bouguezel and M. Omair Ahmad, "A New Radix-2/8 FFT Algorithm for Length- $q\times 2^m$ DFTs, IEEE Transactions on Circuits and Systems — I: Regular Papers, vol. 51, no. 9, pp.1723-1732, September 2004.

[5] S. C. Chen, "Split Vector-Radix Fast Fourier Transform", IEEE Transactions on Signal Processing, vol. 40, no. 8, pp.2029- 2039, August 1992.

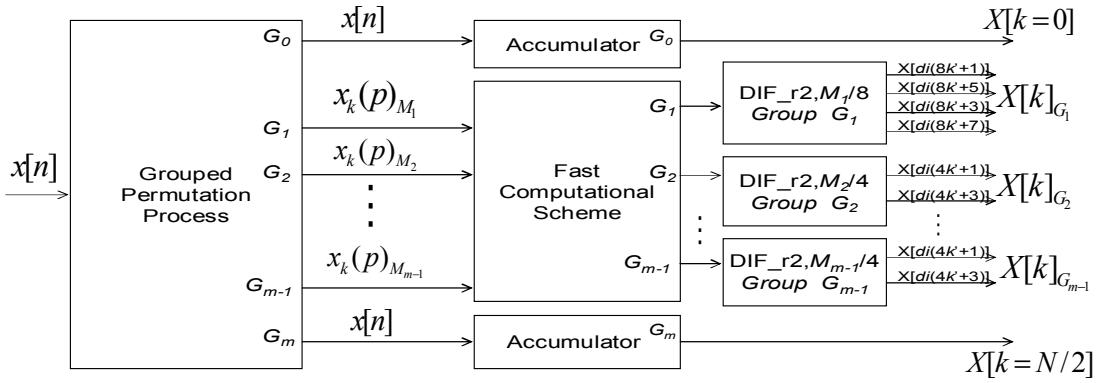

Figure 3 The whole data flow diagram of the proposed FFT implementation

A Low Multiplier and Multiplication Costs 256-point FFT Implementation with Simplified Radix-2^4 SDF Architecture

Chih-Peng Fan*, Mau-Shih Lee and Guo-An Su
Department of Electrical Engineering, National Chung Hsing University,
250 Kuo-Kuang Road, Tai-chung, Taiwan, R.O.C.
cpfan@dragon.nchu.edu.tw *

Abstract—**In this paper, we propose a low multiplier and multiplication complexities 256-point fast Fourier transform (FFT) architecture, especially for WiMAX 802.16a systems. Based on the radix-16 FFT algorithm, the proposed FFT architecture utilizes cascaded simplified radix-2^4 single-path delay feedback (SDF) structures. The control circuit of the proposed simplified radix-2^4 SDF FFT architecture is simple. The hardware requirement of the proposed FFT architecture only needs 1 complex multiplier and 56 complex adders for supporting 256-point computations. The computation complexity of multiplications and the hardware complexity of the proposed FFT architecture need less complexity than both complexities of the previous FFT structures in 256-point FFT applications. In hardware verifications, the output throughput rate of our FFT design processes up to 35.5M samples/sec with Xilinx Virtex2 1500 FPGA, and it processes up to 51.5M samples/sec with UMC 0.18μm standard cell technology. The throughput rate of this implementation is suitable for WiMAX 802.16a application, whose maximum sample rate is 32MHz.**

Keywords— Fast Fourier Transform (FFT), Single-Path Delay Feedback (SDF) Structure

I. INTRODUCTION

Recently, OFDM technologies are more and more significant in communication systems. For example, WiMAX [1], DVB-T[2], Wireless LAN [3], ADSL [4], and VDSL [5] all utilize OFDM techniques. The OFDM technique is advantageous that it efficiently uses channels, overcomes multi-path fading, and has simpler equalizers. The OFDM systems need the FFT and IFFT processors to perform real-time operations for modulation and demodulation of signals. In recent years, the developments of the fast computations of discrete cosine transform (DFT) become more important in order that the researchers [7-18] reduce the computational complexity of the FFT algorithms. For this reason, the computational complexity of the DFT computation has been decreased from $O(N^2)$ to O(NlogN) [6]. In OFDM communications, the number of carriers is directly proportional to the length of the FFT transformation in different OFDM systems. For instance, the FFT size in the WiMAX[1] systems supports 256-point transform length. So the design of the efficient 256-point FFT processor in WiMAX systems is necessarily required.

The pipelined FFT/IFFT processor architecture which has been designed in OFDM communication system has been studied since the 1970's. There are many kinds of the methods to implement the FFT hardware architectures. All hardware implementations of pipelined FFT can be categorized into three kinds of pipelined architectures, which include multiple delay commutator (MDC), single delay commutator (SDC), and single delay feedback (SDF) architectures. In comparison with three pipeline architectures, the SDF architecture is most suitable and its advantages are listed as follows, (1) The SDF architecture is very convenient to implement the different length FFT. (2) The number of the required registers in SDF architecture is less than that in MDC and SDC structures. (3) The controller of SDF architecture is easier than the other structures. In the circumstances of no increasing chip area and power consumption, the radix-2^N SDF architectures are better than the direct radix-4, radix-8 and radix-16 SDF architectures for N=2, 3 and 4, respectively. Thus, we would choose the SDF architecture to design the proposed 256-point FFT processor.

In this paper, we propose a novel SDF based 256-point FFT design with low multiplier and multiplication costs. The rest of the paper is organized as follows. In Section II, we begin with the derivations of the radix-16 algorithm. By using the structure of radix-8 FFT algorithm [7], the radix-16 FFT algorithm is described to guide the depiction of hardware pipelined architectures. Next, the proposed simplified radix-2^4 SDF pipeline FFT architecture is also shown in this section. In Section III, we propose a low-complexity 256-point FFT architecture with simplified radix-2^4 SDF structure. The proposed simplified radix-2^4 based FFT engine can reduce the computational complexity in multiplications, the hardware complexity and control circuit complexity. The results of hardware realizations are shown in this section. Then, the comparisons of computational complexity and hardware complexity are also described in Section III. Finally, we give a conclusion in Section IV.

II. Radix-16 Algorithm and Pipelined FFT Architecture

2.1 RADIX-16 ALGORITHM

Figure 1 shows the butterfly structure for computation of the radix-8 FFT, where $W_N^{nk} = e^{-i\frac{2\pi}{N}nk}$. Let k=2p and then replace the k index with the p index in radix-8 FFT structure.

Thus, the even parts of the radix-16 FFT algorithm can be described from (1) to (8) as follows,

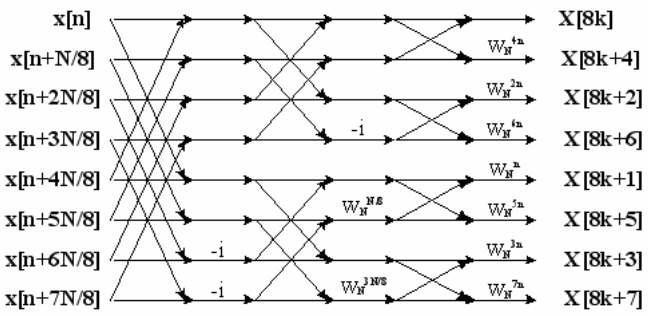

Fig. 1 Butterfly structure of radix-8 FFT [7]

$$X[16p]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])+(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])\}+\{(x[n+\frac{N}{8}]+x[n+\frac{5}{8}N])+(x[n+\frac{3}{8}N]+x[n+\frac{7}{8}N])\}\}+\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])+(x[n+\frac{5}{16}N]+x[n+\frac{13}{16}N])\}+\{(x[n+\frac{3}{16}N]+x[n+\frac{11}{16}N])+(x[n+\frac{7}{16}N]+x[n+\frac{15}{16}])\}\}\}\bullet W_{\frac{N}{16}}^{pn} \quad (1)$$

$$X[16p+2]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])-(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])\}-j\{(x[n+\frac{N}{8}]+x[n+\frac{5}{8}N])+(x[n+\frac{3}{8}N]+x[n+\frac{7}{8}N])\}\}+W_N^{N/8}\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])-(x[n+\frac{5}{16}N]+x[n+\frac{13}{16}N])\}-j\{(x[n+\frac{3}{16}N]-x[n+\frac{11}{16}N])+(x[n+\frac{7}{16}N]-x[n+\frac{15}{16}])\}\}\}\bullet W_{\frac{N}{16}}^{pn}W_N^{2n} \quad (2)$$

$$X[16p+4]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])+(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])\}-\{(x[n+\frac{N}{8}]+x[n+\frac{5}{8}N])+(x[n+\frac{3}{8}N]+x[n+\frac{7}{8}N])\}\}-j\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])-(x[n+\frac{5}{16}N]+x[n+\frac{13}{16}N])\}-\{(x[n+\frac{3}{16}N]-x[n+\frac{11}{16}N])+(x[n+\frac{7}{16}N]-x[n+\frac{15}{16}])\}\}\}\bullet W_N^{4n}W_{\frac{N}{16}}^{pn} \quad (3)$$

$$X[16p+6]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])-(x[n+\frac{N}{4}]-x[n+\frac{3N}{4}])\}+j\{(x[n+\frac{N}{8}]+x[n+\frac{5}{8}N])-(x[n+\frac{3}{8}N]+x[n+\frac{7}{8}N])\}\}+j\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])-(x[n+\frac{5}{16}N]-x[n+\frac{13}{16}N])\}+j\{(x[n+\frac{3}{16}N]+x[n+\frac{11}{16}N])-(x[n+\frac{7}{16}N]+x[n+\frac{15}{16}])\}\}\}\bullet W_N^{6n}W_{\frac{N}{16}}^{pn} \quad (4)$$

$$X[16p+8]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])+(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])\}+\{(x[n+\frac{N}{8}]+x[n+\frac{5}{8}N])+(x[n+\frac{3}{8}N]+x[n+\frac{7}{8}N])\}\}-\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])+(x[n+\frac{5}{16}N]+x[n+\frac{13}{16}N])\}+\{(x[n+\frac{3}{16}N]+x[n+\frac{11}{16}N])+(x[n+\frac{7}{16}N]+x[n+\frac{15}{16}])\}\}\}\bullet W_{\frac{N}{16}}^{pn}W_{\frac{N}{8}}^{n} \quad (5)$$

$$X[16p+10]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])-(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])\}-j\{(x[n+\frac{N}{8}]-x[n+\frac{5}{8}N])+(x[n+\frac{3}{8}N]-x[n+\frac{7}{8}N])\}\}-W_N^{N/8}\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])-(x[n+\frac{5}{16}N]+x[n+\frac{13}{16}N])\}-j\{(x[n+\frac{3}{16}N]-x[n+\frac{11}{16}N])+(x[n+\frac{7}{16}N]-x[n+\frac{15}{16}])\}\}\}\bullet W_N^{10n}W_{\frac{N}{16}}^{pn} \quad (6)$$

$$X[16p+12]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])+(x[n+\frac{N}{4}]+x[n+\frac{3N}{4}])\}-\{(x[n+\frac{N}{8}]-x[n+\frac{5}{8}N])+(x[n+\frac{3}{8}N]-x[n+\frac{7}{8}N])\}\}+j\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])+(x[n+\frac{5}{16}N]+x[n+\frac{13}{16}N])\}-\{(x[n+\frac{3}{16}N]-x[n+\frac{11}{16}N])+(x[n+\frac{7}{16}N]-x[n+\frac{15}{16}])\}\}\}\bullet W_N^{12n}W_{\frac{N}{16}}^{pn} \quad (7)$$

$$X[16p+14]=\sum_{n=0}^{\frac{N}{16}-1}\{\{\{(x[n]+x[n+\frac{N}{2}])-(x[n+\frac{N}{4}]-x[n+\frac{3N}{4}])\}+j\{(x[n+\frac{N}{8}]+x[n+\frac{5}{8}N])-(x[n+\frac{3}{8}N]+x[n+\frac{7}{8}N])\}\}-W_N^{3N/8}\{\{(x[n+\frac{N}{16}]+x[n+\frac{9}{16}N])-(x[n+\frac{5}{16}N]-x[n+\frac{13}{16}N])\}+j\{(x[n+\frac{3}{16}N]+x[n+\frac{11}{16}N])-(x[n+\frac{7}{16}N]+x[n+\frac{15}{16}])\}\}\}\bullet W_N^{14n}W_{\frac{N}{16}}^{pn} \quad (8)$$

Let $k=2p+1$ in Figure 1, the odd part of the radix-16 FFT can be achieved in the similar derivations of the even parts. Although the radix-16 FFT needs less computation complexity, the control circuit of the direct radix-16 SDF architecture for computing radix-16 FFT is very complex. Thus, the efficient simplified radix-2^4 SDF structure, which is described in the next section, will be applied to solve the computations of the 256-point FFT.

2.2 Simplified Radix-2^4 SDF Pipelined Architecture

The proposed simplified radix-2^4 SDF (SR2^4SDF) architecture is derived from radix-2 SDF FFT. The proposed SR2^4SDF architecture can be derived from the corresponding 4-stage radix-16 bufferfly architecture. The four SDF recursive computing stages correspond to the relative four-stage radix-16 FFT butterfly flow. The SR2^4SDF architecture needs $\log_{16}N$-1 multipliers, $4\log_4N$ adder, and N-1 registers. The control circuit of SR2^4SDF is simple, in comparison with the direct radix-16 SDF architecture. The hardware requirements of different pipelined SDF FFT architectures are listed in Table 1.

The architecture of the proposed SR2^4SDF is shown in Figure 2. The SR2^4SDF architecture is constructed by four radix-2 based processing elements (PE) and the coefficient multipliers. Each radix-2^4 PE performs two-point add/sub butterfly operation. Each PE needs two complex adders. The first twiddle factor, –j, performs just sign exchange between real and imaginary part data. The second twiddle factor, w1, performs three fixed-coefficient multiplications. Then, the third twiddle factor, w2, performs seven fixed-coefficient multiplications. For the reduction of the chip area and power consumption, the computations of w1 and w2 can be implemented with the canonical signed digital (CSD) circuits.

Finally, the last stage computation "⊗" is realized with a complex multiplier.

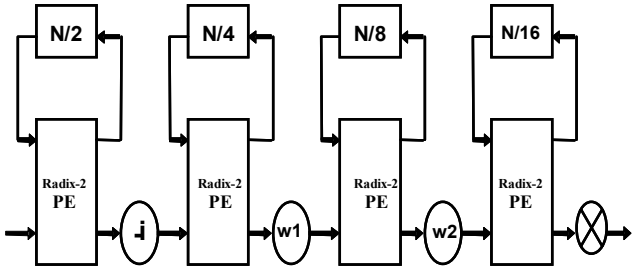

Fig.2 The simplified R2^{4}SDF architecture for N-point FFT

Table 1 Comparisons of Hardware Requirements for N-length FFT with Different SDF Architectures

	Complex Multiplier	Complex Adder	Register	Control Circuit
R2 SDF[11]	$\log_2 N - 2$	$\log_2 N$	N - 1	simple
R4 SDF[12]	$\log_4 N - 1$	$\log_4 N$	N - 1	medium
R2^2 SDF[8]	$\log_4 N - 1$	$4\log_4 N$	N - 1	simple
R2^3 SDF[9]	$\log_8 N - 1$	$4\log_4 N$	N - 1	simple
R248SDF[13]	$\log_4 N - 1$	$4\log_4 N$	N - 1	simple
R16 SDF	$\log_{16} N - 1$	$4\log_4 N$	N - 1	complex
SR2^4 SDF	$\log_{16} N - 1$	$4\log_4 N$	N - 1	simple

III. Low-complexity 256-point FFT Architecture

The control circuit of radix-2^4 FFT SDF architecture is simpler than that of the corresponding radix-16 SDF architecture. The radix-2^N FFT SDF architectures not only reduce the complexity of the control circuits, but also maintain the same number of multipliers, adders, subtracters and the same utilization of memory in the hardware architecture.

In Figure 3, the proposed FFT processor can be applied to 256-point FFT computations. The proposed efficient architecture needs 1 complex multiplier and 56 complex adders. According to the simplified radix-2^4 SDF architecture in Section 2.2, we know that the SR2^{4}SDF FFT architecture can be separated into 4-stage computations from the radix-16 algorithm. However, the complexity of the SR2^{4}SDF control circuit is simpler than that of the radix-16 SDF architecture, and then the output throughput rate of the SR2^{4}SDF architecture is equal to that of the complex radix-16 SDF architecture.

In Table 2, we show the comparisons of computational complexity among different architectures for computation of the 256-point FFT. For fair comparisons, each complex multiplication is equal to four real multiplications and two real additions. Our proposed architecture needs 1600 real multiplications and 4896 real additions for the 256-point FFT computations. Our proposed architecture needs the near computational complexity of multiplications compared with the previous 256-point FFT designs.

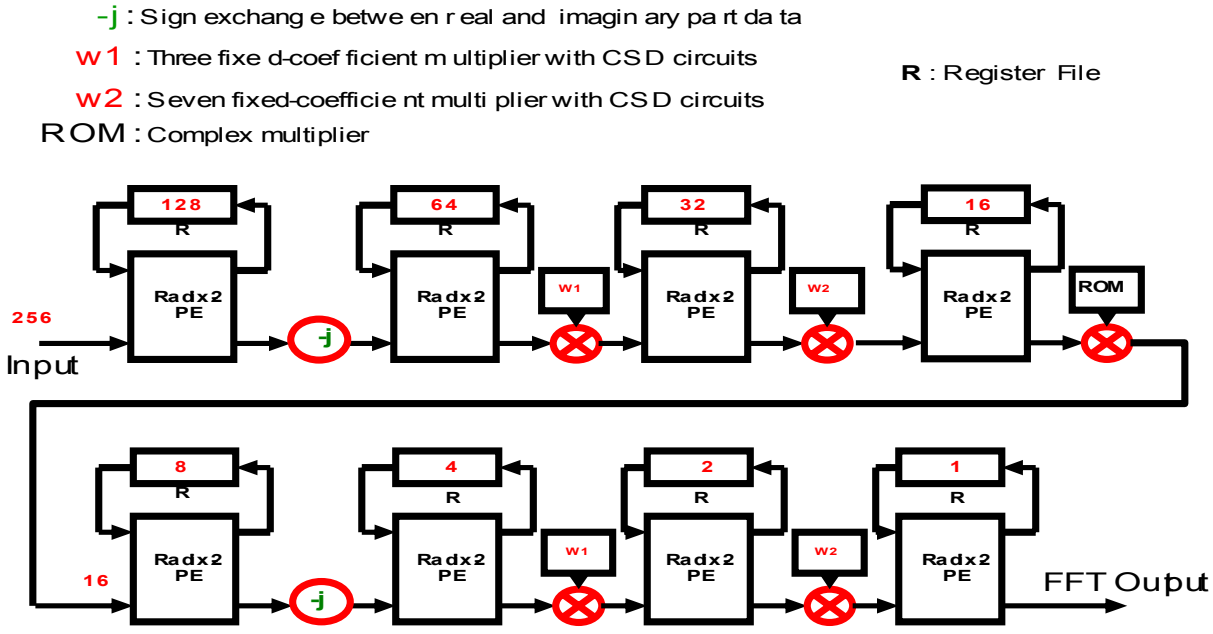

Fig. 3 The 256-point FFT processor with the simplified radix-2^4 SDF architecture

Table 2 Comparisons of Computational Complexity Among Different Architectures for Computation of 256-point FFT

Complexity / Architectures	Numbers of Real Multiplications	Numbers of Real Additions
Radix-2	2316	5380
Radix-4	1800	5080
Takahashi [17]	1660	5140
Bouguezel et al. [15]	1592	5072
Bouguezel [14]	1592	5072
Proposed	1600	4896

Table 3 Comparisons of Hardware Requirements for 256-point FFT with Different Architectures

Architectures	Complex Multipliers	Complex Adders	Register	Control Circuits
Oh and Lim [10]	2.1	16	255	Simple
He et al. [8]	3	16	255	Simple
Yeh and Jen [16]	3	16	255	Simple
Son et al.[18]	3	8	256	--------
Proposed	1	56	255	Simple

Table 3 shows the comparisons of hardware requirements for supporting 256-point FFT with different FFT architectures. Our proposed architecture needs 1 complex multiplier and 16 complex adders for the 256-point FFT computations in WiMAX systems. Table 4 and Table 5 show the results of hardware verifications of the proposed 256-point FFT processor.

Table 4 Hardware verification with FPGA

Target Device	Xilinx Virtex-II 1500 FG676 -4
FPGA gates	176127
Max. speed	35.76MHz

Table 5 Hardware verification with Standard Cell Design

Design Process	UMC 0.18μm 1P6M
Gates count	173875
Max. speed	51.5MHz
Power consumption	162.7mW@33.3MHz
Supply voltage	1.8V

IV. CONCLUSION

In this paper, we propose a low multiplier and multiplication complexities 256-point FFT architecture design for special application to WiMAX 802.16 systems. The proposed FFT architecture utilizes cascaded simplified radix-2^4 SDF structures. The control circuit of simplified radix-2^4 SDF FFT architecture is simple. The hardware requirement of the proposed FFT architecture needs 1 complex multiplier and 56 complex adders for supporting 256-point computations. The computation complexity of multiplications of our design needs less complexity than that of the previous 256-point FFT structures. In hardware verifications, the output throughput rate of the FFT design processes up to 35.5M samples/sec with Xilinx Virtex2 FPGA, and it processes up to 51.5M samples/sec with UMC 0.18μm standard cell design. The throughput rate of this scheme is suitable for WiMAX 802.16a application, whose maximum sample rate is 32MHz.

Acknowledgement

This work was supported by the National Science Council, Taiwan, R.O.C., under Grant NSC95-2220-E-005-006.

REFERENCES

[1] IEEE Std. 802.16-2004, Part 16: Air Interface for Fixed Broadband Wireless Access Systems, October 2004.

[2] ETSI EN 300 744 (v1.2.1):Digital Video Broadcasting (DVB); Framing Structure, Channel Coding and Modulation for Digital Terrestrial Television, Jul. 1999.

[3] IEEE 802.11, IEEE Standard for Wireless LAN Medium Access Control and Physical Layer Specifications, Nov. 1999.

[4] T1E1.4/98-007R4: Standards Project for Interfaces Relating to Carrier to Customer Connection of Asymmetrical Digital Subscriber Line (ADSL) Equipment, Jun. 1998.

[5] ETSI TS 101 270-2 (v1.1.1): Transmission and Multiplexing(TM); Access transmission systems on metallic access cables; Very high speed Digital Subscriber Line (VDSL); Part 2: Transceiver specification, Feb. 2001.

[6] Alan V. Oppenheim, Ronald W. Schafer and John R. Buck, "Discrete-Time Signal Processing", 1989.

[7] Lihong Jia, Yonghong Gao, Hannu Tenhunen, "Efficient VLSI Implementation of Radix-8 FFT Algorithm," IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 468-471, 1999.

[8] Shousheng He and Mats Torkelson, "A New Approach to Pipeline FFT Processor", Parallel Processing Symposium, 1996., Proceedings of IPPS '96, The 10th International,15-19, pp. 766 – 770, April 1996.

[9] Shousheng He and Mats Torkelson, "Designing Pipeline FFT Processor for OFDM (de)Modulation", International Symposium on Signals, Systems, and Electronics, pp. 257– 262, Oct. 1998.

[10] Jung-yeol Oh and Myoung-seob Lim, "A Radix-2^4 SDF Pipeline FFT Processor for OFDM Modulation", The First IEEE VTS APWCS (Asia Pacific Wireless Communications Symposium), Jan 2004.

[11] Wold E. H. and Despain A. M., "Pipeline and Parallel-Pipeline FFT Processors for VLSI Implementation", IEEE Trans. Comput., vol. 33, No.5, pp. 414-426, May 1984.

[12] Despain A. M., "Fourier Transform Computer Using CORDIC Iterations", IEEE Trans. Comput., vol. 23, No. 10, pp. 993-1001, Oct. 1974.

[13] Lihong Jia, Yonghong Gao, Jouni Isoaho and Hannu Tenhunen, "A New VLSI-Oriented FFT Algorithm and Implementation", IEEE ASIC Conference, pp. 337-341, 1998.

[14] Saad Bouguezel, "A New Radix-2/8 FFT Algorithm for Length-q×2m DFTs", IEEE Transactions on Circuits and Systems-I, vol. 51, no. 9, pp. 1723-1732, September 2004

[15] Saad Bouguezel, M. Omair Ahmad, and M.N.S. Swamy, "An Efficient Split-Radix FFT Algorithm", IEEE International Symposium on Circuits and Systems, vol. 4, IV65-68, May 2003.

[16] Wen-Chang Yeh and Chein-Wei Jen, "High-Speed and Low-Power Split-Radix FFT", IEEE Transactions on Signal Processing, vol. 51, no 3, pp. 864-874, March 2003.

[17] Daisuke Takahashi, "An Extended Split-Radix FFT Algorithm", IEEE Signal Processing Letters, vol. 8, no. 5, pp. 145-147, May 2001.

[18] Byung S. Son, Byung G. Jo, Myung H. Sunwoo, and Yong Serk Kim, "A High-Speed FFT Processor for OFDM Systems," IEEE International Symposium on Circuits and Systems, Volume: 3, vol. 3, pp. 281-284, 2002.