IEICE TRANS. FUNDAMENTALS, VOL.E90–A, NO.11 NOVEMBER 2007

PAPER

Long-Point FFT Processing Based on Twiddle Factor Table Reduction

Ji-Hoon KIM[†], *Nonmember and* **In-Cheol PARK**[†a)], *Member*

SUMMARY In this paper, we present a new fast Fourier transform (FFT) algorithm to reduce the table size of twiddle factors required in pipelined FFT processing. The table size is large enough to occupy significant area and power consumption in long-point FFT processing. The proposed algorithm can reduce the table size to half, compared to the radix-2^2 algorithm, while retaining the simple structure. To verify the proposed algorithm, a 2048-point pipelined FFT processor is designed using a 0.18 μm CMOS process. By combining the proposed algorithm and the radix-2^2 algorithm, the table size is reduced to 34% and 51% compared to the radix-2 and radix-2^2 algorithms, respectively. The FFT processor occupies 1.28 mm^2 and achieves a signal-to-quantization-noise ratio (SQNR) of more than 50 dB.

key words: *discrete Fourier transform (DFT), fast Fourier transform (FFT), low-power design, orthogonal frequency division multiplexing (OFDM), pipelined processing*

1. Introduction

The fast Fourier transform (FFT) is a major signal processing block being widely used in communication systems, especially in orthogonal frequency division multiplexing (OFDM) systems such as digital video broadcasting, digital subscriber line and WiMAX (IEEE 802.16). As such as system requires long-point FFT computation for multiple carrier modulation, usually more than 1024 points, it is desirable to reduce computational complexity as well as hardware complexity.

To reduce the computational complexity of FFT processing, various FFT algorithms have been proposed such as radix-2^2 [1], radix-2^3 [2], radix-4+2 [3], split-radix [4] as well as radix-2 and radix-4 algorithms. Although the previous algorithms could reduce the computational hardware resources such as multipliers and adders, they did not seriously take into account the number of twiddle factors required in FFT processing. The twiddle factors can be computed on the fly by using the CORDIC (COordinate Rotation DIgital Computer) algorithm [5]. As the CORDIC algorithm takes multiple cycles and requires a lot of hardware resources, the twiddle factors are usually stored in tables that are generally implemented with ROMs. In the implementation of a long-point FFT processor, however, the tables become large enough to occupy significant area and power consumption [6].

In this paper, a new FFT algorithm is proposed to overcome the large table requirement, which not only reduces the table size by a factor of two compared to the radix-2^2 algorithm but also retains the simple structure of the radix-2 algorithm. Since additional computations incurred by applying the proposed algorithm can be implemented with a few adders, the overall computational complexity is almost the same as that of the radix-2^2 algorithm.

For efficient FFT processing, a number of hardware architectures have been proposed, such as serial processing on a single processor [7], pipelined processing [8]–[10], and parallel processing [11]–[13]. Among them, the pipelined processing is preferred, as it can provide high performance with a moderate cost. Based on the proposed algorithm, a pipelined 2048-point FFT processor is designed. It can also support 1024/512/256-point FFT processing. By applying the proposed algorithm to the first several stages, the table size is reduced to 34% and 51% compared to the radix-2 and radix-2^2 algorithms, respectively.

The rest of this paper is organized as follows. In Sect. 2, a new FFT algorithm is proposed. The hardware issues in the proposed algorithm are described in Sect. 3. A 2048-point FFT processor based on the proposed algorithm is presented in Sect. 4.

2. Proposed FFT Algorithm

The proposed algorithm can be derived by applying the radix-2 decimation-in-frequency (DIF) decomposition presented by Cooley and Tukey [14] two times. The N-point Discrete Fourier Transform (DFT) of a sequence $x(n)$ is defined as

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

where $x(n)$ and $X(k)$ are complex numbers. The twiddle factor is defined as follows.

$$W_N^{kn} = e^{-j\left(\frac{2\pi kn}{N}\right)} = \cos\left(\frac{2\pi kn}{N}\right) - j\sin\left(\frac{2\pi kn}{N}\right) \tag{2}$$

The two decompositions can be expressed if n and k are replaced with 3-dimensional linear index maps shown below.

$$\begin{aligned} n &= \frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3 \\ k &= k_1 + 2k_2 + 4k_3 \end{aligned} \tag{3}$$

Manuscript received September 20, 2006.
Manuscript revised May 25, 2007.
Final manuscript received July 18, 2007.

[†]The authors are with the Division of Electrical Engineering, the School of Electrical Engineering and Computer Sciences, Korea Advanced Institute of Science and Technology, 373-1, Guseong-dong, Yuseong-gu, Daejeon, 305-701 Korea.
a) E-mail: icpark@ee.kaist.ac.kr

DOI: 10.1093/ietfec/e90–a.11.2526

Using the above index maps, Eq. (1) can be rewritten as

$$\begin{aligned} X(k) &= X(k_1 + 2k_2 + 4k_3) \\ &= \sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \sum_{n_1=0}^{1} x\left(\frac{N}{2}n_1 + \frac{N}{4}n_2 + n_3\right) \\ &\qquad \cdot W_N^{(\frac{N}{2}n_1+\frac{N}{4}n_2+n_3)(k_1+2k_2+4k_3)} \\ &= \sum_{n_3=0}^{\frac{N}{4}-1} \sum_{n_2=0}^{1} \left\{B\left(\frac{N}{4}n_2 + n_3, k_1\right) \cdot W^{(\frac{N}{4}n_2+n_3)k_1}\right\} \\ &\qquad \cdot W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} \end{aligned} \tag{4}$$

where $B(\bullet)$ represents the following butterfly structure.

$$\begin{aligned} &B\left(\frac{N}{4}n_2 + n_3, k_1\right) \\ &= x\left(\frac{N}{4}n_2 + n_3\right) + (-1)^{k_1} x\left(\frac{N}{4}n_2 + n_3 + \frac{N}{2}\right) \end{aligned} \tag{5}$$

The main idea of the proposed algorithm is to take into account the value of n_3 in the summation of n_2. For even n_3 (= 2 m), the sum is arranged as follows.

$$\begin{aligned} &\sum_{n_2=0}^{1} \left\{B\left(\frac{N}{4}n_2 + n_3, k_1\right) \cdot W^{(\frac{N}{4}n_2+n_3)k_1}\right\} \\ &\qquad \cdot W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} \\ &= \left\{B(n_3, k_1) + (-1)^{k_2}(-j)^{k_1} B\left(n_3 + \frac{N}{4}, k_1\right)\right\} \\ &\qquad \cdot W_N^{2m(k_1+2k_2)} W_N^{4n_3k_3} \end{aligned} \tag{6}$$

If n_3 is odd (= $2m + 1$), the sum becomes

$$\begin{aligned} &\sum_{n_2=0}^{1} \left\{B\left(\frac{N}{4}n_2 + n_3, k_1\right) \cdot W^{(\frac{N}{4}n_2+n_3)k_1}\right\} \\ &\qquad \cdot W_N^{(\frac{N}{4}n_2+n_3)(2k_2+4k_3)} \\ &= \left\{W_N^{k_1} B(n_3, k_1)\right. \\ &\left. + (-1)^{k_2}(-j)^{k_1} W_N^{k_1} B\left(n_3 + \frac{N}{4}, k_1\right)\right\} \\ &\qquad \cdot W_N^{2k_2} W_N^{2m(k_1+2k_2)} W_N^{4n_3k_3} \end{aligned} \tag{7}$$

where m is an integer between 0 and $N/8 - 1$. By substituting Eq. (6), Eq. (7) to Eq. (4), we obtain the following expression.

$$\begin{aligned} X(k) &= X(k_1 + 2k_2 + 4k_3) \\ &= \underbrace{\sum_{n_3=0}^{\frac{N}{4}-1} [H(k_1, k_2, n_3) W_N^{2m(k_1+2k_2)}]}_{\frac{N}{4}\text{-point DFT}} W_N^{4n_3k_3} \end{aligned} \tag{8}$$

In Eq. (8), the expression of $H(\bullet)$ also depends on the value of n_3. For even n_3, $H(\bullet)$ is expressed as

$$\begin{aligned} &H(k_1, k_2, n_3) \\ &= B(n_3, k_1) + (-1)^{k_2}(-j)^{k_1} B\left(n_3 + \frac{N}{4}, k_1\right) \end{aligned} \tag{9}$$

If n_3 is odd, $H(\bullet)$ can be arranged as follows.

$$\begin{aligned} &H(k_1, k_2, n_3) \\ &= \left[W_N^{k_1} B(n_3, k_1)\right. \\ &\left. + (-1)^{k_2}(-j)^{k_1} W_N^{k_1} B\left(n_3 + \frac{N}{4}, k_1\right)\right] \cdot W_N^{2k_2} \end{aligned} \tag{10}$$

As k_1 is either 0 or 1, Eq. (9) indicates that the butterfly has a trivial multiplication of $-j$ at the input side if n_3 is even and Eq. (10) implies that an additional constant multiplication of W_N^1 is required at the input side if n_3 is odd. By performing the constant multiplications at the input side, all the exponents of the twiddle factors $W_N^{2m(k_1+2k_2)}$ and $W_N^{2k_2} \cdot W_N^{2m(k_1+2k_2)}$ to be multiplied in Eq. (8) become even values. This implies that, in the proposed algorithm, the table required in every second stage needs to store only the twiddle factors of which exponents are even. However, in the radix-2, radix-2^2, and radix-2^3 algorithms, most of the twiddle factor tables have twiddle factors associated with both even and odd exponents.

The N-point Inverse DFT (IDFT) of a sequence $X(k)$ is defined as

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-kn}, \quad 0 \le n < N \tag{11}$$

As shown in Eq. (11), the inverse fast Fourier transform (IFFT) can be processed just as the same way as we do for FFT with changing only the sign of the imaginary part of the twiddle factors.

3. Proposed Table Size Reduction in Implementation

In pipelined radix-2 DIF FFT processing, the number of required twiddle factors is largest at the first stage and reduced gradually at the successive stages. The number of twiddle factors required in a stage is determined by two factors. The first factor is the maximum exponent value (MEV) of twiddle factors, and the other is the greatest common divisor (G.C.D) value among the exponents of the twiddle factors, that is, *minimum stride*. Considering these two factors, the number of required twiddle factors, $N_{required}$, can be represented as follows.

$$N_{required} = \frac{MEV}{minimum\ stride} \tag{12}$$

However, the ROM size is usually larger than $N_{required}$ since the number of ROM entries is usually constrained to a power of two. The ROM size, N_{ROM}, is also largest at the first stage. Figure 1 illustrates the signal flow graphs of 16-point FFT corresponding to the radix-2, radix-2^2, radix-2^3 and the proposed algorithms. As shown in Fig. 1, the twiddle factor table is needed at every stage in the radix-2 algorithm.

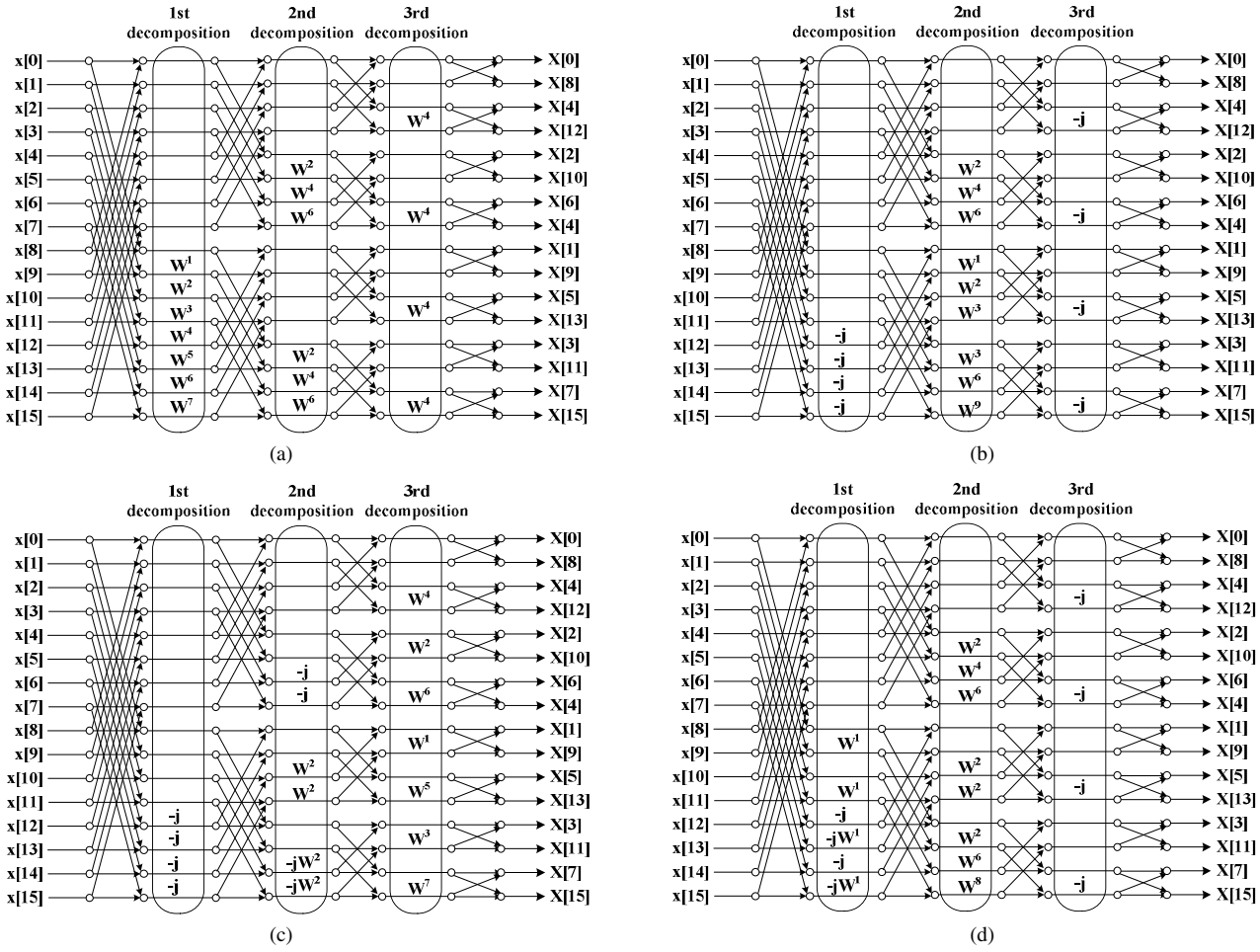

Fig. 1 Signal flow graphs for 16-point FFT. (a) Radix-2 algorithm. (b) Radix-2^2 algorithm. (c) Radix-2^3 algorithm. (d) Proposed algorithm.

Table 1 Decomposition factors comparison results.[1)]

	1st decomposition			2nd decomposition			3rd decomposition		
	minimum stride[2)]	*MEV*	N_{ROM}	*minimum stride*	*MEV*	N_{ROM}	*minimum stride*	*MEV*	N_{ROM}
Radix-2	1	7	8	2	6	4	N.A	4	1
Radix-2^2	N.A	4	1	1	9	16	N.A	4	1
Radix-2^3	N.A	4	1	N.A	6	1	1	7	8
Proposed	N.A	5	1	2	8	4	N.A	4	1

[1)] No Symmetric property in twiddle factors is considered.
[2)] N.A means this decomposition does not need the table for the twiddle factors — the constant multiplier or $\pi/2$ rotator exists.

Compared to the radix-2 algorithm, the radix-2^2 or radix-2^3 algorithm reduces the number of tables by introducing constant multipliers.

To reduce the ROM table size, in general, non-trivial twiddle factors should be regularly distributed. This regularity can be determined by the value of *minimum stride*. Table 1 shows the number of ROM entries required in 16-point FFT processing shown in Fig. 1 for the various FFT algorithms. Since the value of the *minimum stride* in the proposed algorithm is bigger than one, we can reduce the total number of entries of the ROM at least by a factor of two compared to the famous radix-2^2 algorithm that has a twiddle factor table in every second stage. Considering the $\pi/2$ symmetric property of the twiddle factors shown in Fig. 2 in counting the number of entries, the total number of table entries required for the case of 8192-point and 2048-point FFT processing are compared in Table 2. By swapping the real and imaginary parts and changing their sign if required to compensate the $\pi/2$ symmetry, all twiddle factors can be obtained from the stored twiddle factors. As denoted in

Table 2, the proposed algorithm can effectively reduce the number of table entries compared to the other FFT algorithms.

In the proposed algorithm, the reduction of table entries is achieved at the cost of an additional constant multiplier per every second stage. Therefore, the effectiveness of the proposed algorithm can be determined by the complexity of the additional constant multiplier. The complexity of the constant multiplier depends on the number of non-zero bits in the binary representation of the constant. To minimize the number of non-zero bits, rounded constants are expressed in the minimal signed digit (MSD) representation [15], as shown in Table 3. Due to the sparse non-zero bits in the sine and cosine values, the constant multipliers can be implemented with a few adders. In case of W_{8192}^{1} multiplication, the corresponding constant multipliers can be implemented with only two adders as shown in Fig. 3(a). Similarly, the constant multipliers for W_{2048}^{1} and W_{512}^{1} can be implemented with four and six adders, respectively, as shown in Figs. 3(b), (c). The number of adders needed for implementing the constant multiplier is specified in Table 3. The constant multiplier for W_{8}^{1} is used in every third stage in the radix-2^3 algorithm.

Although the proposed algorithm is very effective in long-point FFT processing due to the low complexity of the additional constant multiplier, the reduction in table size is not considerable at the later stages of the pipelined FFT processor. For the two consecutive pipeline stages, we have to decide whether to apply the proposed algorithm or not with considering both the cost of the additional constant multiplier and the table size reducible by the proposed algorithm. When the cost of the additional constant multiplier in a certain stage is not compensated by the table reduction in the next stage, the radix-2^2 algorithm should be applied from that stage. The hardware complexity defined by the number of adders in constant multipliers and the number of general multipliers is summarized in Table 2 for 8192-point and 2048-point FFT. The proposed algorithm is applied to the first 8 stages of the total 13 stages and to the first 6 stages of the total 11 stages for 8192-point FFT and 2048-point FFT, respectively. The remaining stages are processed by the radix-2^2 algorithm.

As indicated in Table 2, the proposed algorithm combined with the radix-2^2 algorithm can process long-point FFT with comparable hardware complexity to other hardware-efficient FFT algorithms. In addition to the low hardware complexity, the required total number of table entries is minimal in the proposed algorithm combined with the radix-2^2 algorithm. We can reduce the table size further if we employ the $\pi/4$ symmetric property [16]. As the reduction ratio is independent of what symmetric property is used, the reduction ratio shown in Table 2 also applies to the case of $\pi/4$ symmetry. Reducing the table sizes at the first several stages by applying the proposed algorithm can be significant because the original table sizes at the first several stages are large enough to pay off the additional constant multipliers.

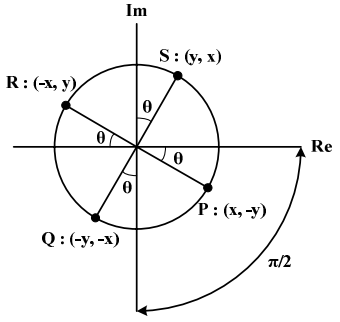

Fig. 2 $\pi/2$ symmetry in twiddle factors.

Table 3 Complexity of constant multipliers.

Constant Operand	MSD Representation	# of required adders
cos(2π/8192)	1.00000000000	2
sin(2π/8192)	0.00000000010	
cos(2π/2048)	1.00000000000	4
sin(2π/2048)	0.00000000110	
cos(2π/512)	1.00000000000	6
sin(2π/512)	0.00000011001	
cos(2π/128)	1.0000000001$\bar{0}$	8
sin(2π/128)	0.00001100100	
cos(2π/8)	0.10110101000	10
sin(2π/8)	0.10110101000	

Table 2 Complexity comparison with other FFT algorithms.

	8192-point			2048-point		
	TN[1)]	CM[2)]	GM[3)]	TN	CM	GM
Radix-2	4092 (150%)	10	40	1020 (150%)	10	32
Radix-2^2	2728 (100%)	10	20	680 (100%)	10	16
Radix-2^3/Radix-4+2	2340 (85.8%)	40	16	584 (85.9%)	30	12
Proposed + Radix-2^2	1368 (50.1%)	30	20	344 (50.6%)	28	16

[1)]Total # of entries in twiddle factor tables.
[2)]Total # of adders in constant multipliers (in MSD representation).
[3)]Total # of general multipliers.

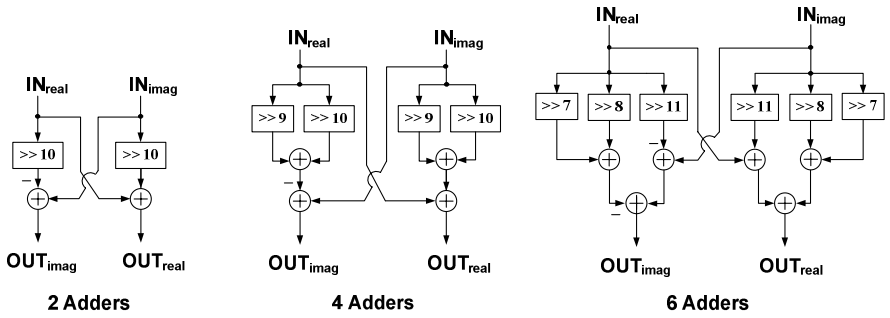

Fig. 3 Low-complexity constant multipliers for (a) W^1_{8192} multiplication, (b) W^1_{2048} multiplication and (c) W^1_{512} multiplication.

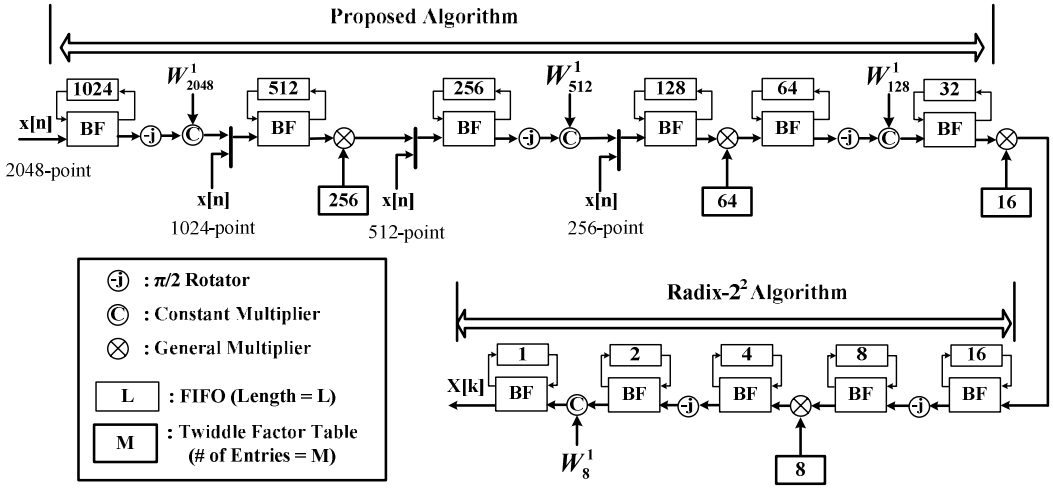

Fig. 4 2048/1024/512/256-point pipelined FFT/IFFT processor structure.

4. Overall Structure of Pipelined 2048-Point FFT/IFFT Processor

By combining the proposed algorithm with the radix-2^2 algorithm as described in Sect. 3, we designed a 2048-point pipelined FFT/IFFT processor of which single-path delay feedback (SDF) structure [1] is shown in Fig. 4. Also, it can process 1024/512/256-point FFT/IFFT. Although the designed processor is based on the SDF structure, the proposed algorithm can be applied to other pipelined structures, such as multi-path delay commutator (MDC) [1]. The $\pi/2$ symmetric property of the twiddle factors is considered in deciding the size of ROM tables. In the FFT processor, three constant multipliers are required to compute non-trivial multiplications of W^1_{2048}, W^1_{512}, W^1_{128} whose MSD representations are shown in Table 3. In implementing the 2048-point FFT processor, the bit-width of the twiddle factors is set to 12 bits by performing several simulations. The longer wordlength is not cost efficient, as the SQNR performance is not increased notably but the cost of multipliers and tables are increased significantly [17].

As a butterfly (BF) contains an adder and a subtractor, the sum should be increased by one bit to avoid overflow, increasing the hardware complexity of memories and computational units. The simplest way to avoid the increase of the internal wordlength is to scale down the output value of each stage to half. If all the internal wordlengths are set to the wordlength of the input, however, the resulting SQNR is very low because of severe information loss. To achieve a SQNR enough to meet the standard specification, therefore, this approach needs an internal wordlength that is much longer than the input wordlength, increasing the overall hardware complexity significantly. In the designed FFT/IFFT processor, the dynamic data scaling method [18], a sort of semi-floating point representation, is adopted to lower the complexity of the computational units. Table 4 shows the internal wordlength configuration and SQNR performance of the designed FFT/IFFT processor when the input is represented in 12 bits.

Table 4 SQNR and internal wordlength of computational units.

Stage Number											SQNR
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	(dB)
12	13	13	13	13	13	13	13	13	13	13	52.8

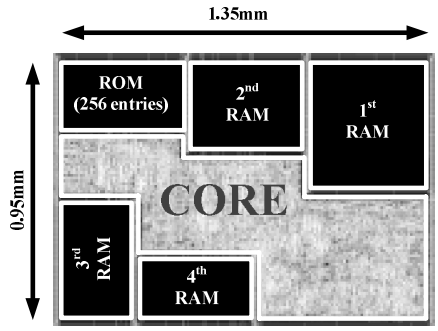

Fig. 5 Layout of the proposed 2048-point FFT processor.

Table 5 Hardware costs of functional blocks included in the proposed FFT processor.

	Gate Count	Area (mm^2)
General Complex Multiplier[1)]	3228	0.034
W^1_{8192} Multiplier	173	0.002
W^1_{2048} Multiplier	390	0.004
W^1_{512} Multiplier	544	0.005
W^1_{128} Multiplier	814	0.008
W^1_8 Multiplier	1163	0.012

[1)]Consists of four general multipliers

Table 6 Areas of ROM tables.

Type	# of entries	Area (mm^2)
Memory macro	2048	0.332
	1024	0.208
	512	0.143
	256	0.107
Logic circuitry	128	0.041
	64	0.020
	32	0.011
	16	0.007

The proposed 2048/1024/512/256-point FFT/IFFT processor was described in a hardware description language and synthesized with a 0.18 μm 4-metal CMOS standard cell library. Figure 5 shows the layout of the proposed FFT/IFFT processor. The proposed FFT processor occupies 1.28 mm^2 and the gate count is 51,510 excluding RAM and ROM memories. The FIFO buffers are implemented using RAM memories, and small-sized RAM and ROM memories are replaced with registers and logic circuitry, respectively. The proposed processor achieves a SQNR of more than 50 dB as indicated in Table 4. For 2048-point FFT/IFFT processing, the latency is 2057 cycles. In addition, the proposed FFT/IFFT processor can process the 1024/512/256-point FFT/IFFT by simply changing the input location.

The area occupied by the core including small-sized RAM and ROM memories is 40.3% of the total area and the hardware costs of the various functional blocks are listed in Table 5. Table 6 shows the areas occupied by ROM tables where the large-size ROMs are generated by the memory compiler. It shows that, at the later stages, the hardware cost of the additional constant multiplier becomes bigger than that of ROM tables. Therefore, the radix-2^2 algorithm is applied to those stages. Based on the measurements shown in the Table 5 and Table 6, we have estimated the areas required for the various FFT algorithms listed in Table 7, where we can conclude that the proposed algorithm is the most area-efficient because of its minimum area occupied by ROM tables and the negligible overhead of additional constant multipliers. Although the radix-2^3 and radix-4+2 algorithms lead to comparable areas in Table 7, their total areas are in practice larger than the proposed one because they need more complex control circuit. Furthermore, it is obvious that the proposed algorithm combined with the radix-2^2 algorithm can lead to more area-efficient implementation for the 8192-point processing compared to other algorithms since the area occupied by the ROM outweighs the area of the additional constant multipliers.

Table 7 Comparison of estimated areas[1)] (mm^2).

	8192-point			2048-point		
	ROM[2)]	CM[3)]	GM[4)]	ROM	CM	GM
Radix-2	0.874	0.012	0.340	0.334	0.012	0.272
Radix-2^2	0.532	0.012	0.170	0.200	0.012	0.136
Radix-2^3/Radix-4+2	0.455	0.048	0.136	0.168	0.036	0.102
Proposed + Radix-2^2	0.347	0.031	0.170	0.139	0.029	0.136

[1)]Area occupied by RAM is the same for all algorithms.
[2)]Area occupied by ROM tables.
[3)]Area occupied by constant multipliers.
[4)]Area occupied by complex general multipliers.

5. Conclusion

We have proposed a new FFT algorithm to reduce the size of twiddle factor tables required in long-point pipelined FFT processors. By applying the proposed FFT algorithm to the first several stages, the table size required in pipelined FFT processing is reduced approximately by half at the cost of a few constant multipliers compared to the radix-2^2 algorithm. Since the constant multipliers can be implemented by a few adders, the proposed algorithm is efficient in long-point FFT computation, especially in terms of area and power consumption. Based on the proposed FFT algorithm, we can design a 2048/1024/512/256-point pipelined FFT/IFFT processor that reduce the total size of twiddle factor tables to 34% and 51% compared to the radix-2 and radix-2^2 algorithms, respectively. The designed FFT/IFFT processor achieves a SQNR of more than 50 dB for 2048-point FFT processing.

Acknowledgments

This work was supported by Institute of Information Technology Assessment through the ITRC and by IC Design Education Center (IDEC).

References

[1] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," Proc. IEEE Custom Integrated Circuits. Conf., pp.131–134, May 1998.
[2] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," Proc. IEEE URSI Int. Symp. Signals, Syst. Electron., pp.257–262, Sept. 1998.
[3] Y. Jung, H. Yoon, and J. Kim, "New efficient FFT algorithm and pipeline implementation results for OFDM/DMT applications," IEEE Trans. Consum. Electron., vol.49, no.1, pp.14–20, Feb. 2003.
[4] P. Duhamel and H. Hollomann, "Split radix FFT algorithm," Electron. Lett., vol.20, no.1, pp.14–16, Jan. 1984.
[5] A.M. Despain, "Fourier transform computers using CORDIC iterations," IEEE Trans. Comput., vol.C-23, pp.993–1001, Oct. 1974.
[6] W. Li and L. Wanhammar, "A pipeline FFT processor," Proc. IEEE Workshop on Signal Processing Systems, pp.654–662, Oct. 1999.
[7] J. Ja'Ja' and R.M. Owens, "An architecture for a VLSI FFT processor," Integr. VLSI J., vol.1, no.4, pp.305–316, 1983.
[8] L.R. Rabiner and B. Gold, Theory and Application of Digital Signal Processing, Prentice-Hall, Englewood Cliffs, NJ, 1975.
[9] E.H. Wold and A.M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementation," IEEE Trans. Comput., vol.C-33, no.5, pp.414–426, May 1984.
[10] E. Swartzlander, V.K.W. Young, and S.J. Joseph, "A radix 4 delay commutator for fast Fourier transform processor implementation," IEEE J. Solid-State Circuits, vol.19, no.5, pp.702–709, Oct. 1984.
[11] H.S. Lee, "Systolic array architecture for VLSI FFT processor," Int. J. of Minu and Microcompt., vol.6, no.3, pp.49–54, 1984.
[12] V. Boriakoff, "FFT computation with systolic arrays, a new architecture," IEEE Trans. Circuits Syst. II, vol.41, no.4, pp.278–284, April 1994.
[13] M.K. Lee, K.W. Shin, and J. Kyu, "A VLSI array processor for 16-point FFT," IEEE J. Solid-State Circuits, vol.26, no.4, pp.1286–1292, Sept. 1991.
[14] J.W. Cooley and J.W. Tukey, "An algorithm for the machine computation of the complex Fourier series," Math. Comput., vol.19, pp.297–301, April 1965.
[15] I.C. Park and H.J. Kang, "Digital filter synthesis based on algorithm to generate all minimal signed digit representations," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.21, no.12, pp.1525–1529, Dec. 2002.
[16] Y.W. Lin, H.Y. Liu, and C.Y. Lee, "A 1-GS/s FFT/IFFT processor for UWB applications," IEEE J. Solid-State Circuits, vol.40, no.8, pp.1726–1735, Aug. 2005.
[17] S. Johansson, S. He, and P. Nilsson, "Wordlength optimization of a pipelined FFT processor," 42nd Midwest Symposium on Circuits and Systems, pp.501–503, Aug. 1999.
[18] T. Lenart and V. Owall, "A 2048 complex point FFT processor using a novel data scaling approach," Proc. IEEE International Symposium on Circuits and Systems, pp.IV-45–IV-48, May 2003.

Ji-Hoon Kim received the B.S. (*summa cum laude*) degree in electrical engineering and computer science from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2004. Currently, he is working toward the Ph.D degree in electrical engineering and computer science at KAIST. Since 2004, he has been a Research Assistant at KAIST, where he worked on developing high-performance CTC (Convolutional Turbo Code) decoder, FFT/IFFT processor for future mobile communication systems, and multihtreaded embedded processor. His current research interests include the algorithm, architecture, and VLSI design for communication system and Multi-Processor System-on-Chip (MPSoC) design.

In-Cheol Park received the B.S. degree in electronic engineering from Seoul National University, in 1986, and the M.S. and Ph.D. degrees in electrical engineering from the Korea Advanced Institute of Science and Technology (KAIST) in 1988, 1992, respectively. Since June 1996, he has been an Assistant Professor and now a Professor in the Department of Electrical Engineering and Computer Science at KAIST. Prior to joining KAIST, he was with IBM T.J. Watson Research Center, Yorktown, NY, from May 1995 to May 1996, where he researched high-speed circuit design. His current research interests include computer-aided design algorithm for high-level synthesis and very large scale integration architectures for general-purpose microprocessors. Dr. Park received the best paper award at the ICCD in 1999 and the best design award at the ASP-DAC in 1997.