*Advanced Science Letters*
**Vol. 22, 3662–3666, 2016**

# 2048-Point Fast Fourier Transform Processing Based on Twiddle Factor Reduction and Dynamic Data Scaling

Bang Chul Jung[1,*], Choul-Young Kim[1], Hyoungho Ko[1], and Ji-Hoon Kim[2]

[1]*Department of Electronics Engineering, Chungnam National University, Daejeon, 34134, Republic of Korea*
[2]*Department of Electrical and Information Engineering, Seoul National University of Science and Technology, Seoul, 01811, Republic of Korea*

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 proposed algorithm can reduce the table size to half, compared to the radix-$2^2$ algorithm, while retaining the simple structure. In addition, a new dynamic data scaling approach is presented to reduce hardware complexity without degrading signal-to-quantization-noise ratio (SQNR). To verify the proposed algorithm, a 2048-point pipelined FFT processor is designed using a 0.18 $\mu$m CMOS process. By combining the proposed algorithm and the radix-$2^2$ algorithm, the table size is reduced to 35% and 53% compared to the radix-2 and radix-$2^2$ algorithms, respectively. The FFT processor occupies 1.95 mm$^2$ and achieves SQNR of more than 55 dB without increasing the internal wordlength progressively using the proposed dynamic data scaling.

**Keywords:** FFT (Fast Fourier Transform), Pipelined Processing, Data Scaling.

## 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 a system requires large-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, various FFT algorithms have been proposed such as radix-$2^2$, radix-$2^3$ as well as radix-2 and radix-4 algorithms.[1,2] 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 to be stored into tables. In the implementation of a large-point FFT processor, however, the tables become large enough to occupy significant area and power consumption.[3] In this paper, a new FFT algorithm is proposed to overcome the problem of the large table requirement, which not only reduces the table size by a factor of two compared to radix-$2^2$ algorithm but also retains the simple structure of 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 radix-$2^2$ algorithm.

When a fixed-point representation is employed to implement a FFT processor, the wordlength has a significant influence on the accuracy and dynamic range. Although a long wordlength is required to achieve high signal-to-quantization-noise ratio (SQNR), it results in a large hardware complexity as the word sizes of memories and computational units such as complex multipliers and complex adders should be increased in proportion to the wordlength.[4] An efficient dynamic data scaling technique is also presented in this paper to lower the hardware complexity without degrading SQNR.

## 2. PROPOSED FFT ALGORITHM

The proposed algorithm can be derived by applying the Cooley and Tukey radix-2 decimation-in-frequency (DIF) decomposition two times.[5] 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.

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

*Author to whom correspondence should be addressed.

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

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

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

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

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

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

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 $n_3(=2m)$, the sum is arranged as follows.

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

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

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

where $m$ is an integer between 0 and $N/8-1$.

By substituting Eqs. (6) and (7) to Eq. (4), we obtain the following expression.

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

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

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

If $n_3$ is odd, then $H(\cdot)$ is arranged below.

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

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 in 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

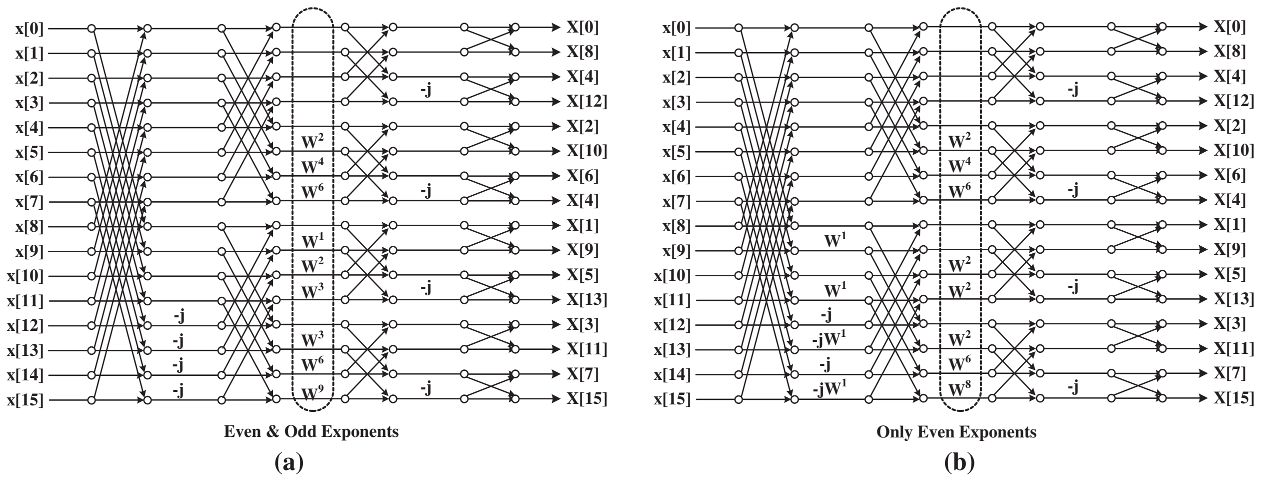

Fig. 1. Signal flow graphs of 16-point FFT. (a) Radix-$2^2$ algorithm and (b) proposed algorithm.

multiplied in Eq. (8) become even values, while the exponents of the twiddle factors in other FFT algorithms such as radix-2, radix-$2^2$, and radix-$2^3$ have both even and odd values. Compared to the radix-$2^2$ algorithm, the proposed algorithm associated with only even exponents reduces the size of twiddle factor table by half at the cost of an additional constant multiplier per two stages, as shown in Figure 1 that illuminates the signal flow graphs of 16-point FFT corresponding to the radix-$2^2$ algorithm and the proposed algorithm.

## 3. DYNAMIC DATA SCALING

As a butterfly contains an adder and a subtractor, one bit should be increased in the result 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 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.

Another data scaling approach is to dynamically scale the internal wordlength. One of the approaches is the block floating point (BFP) method.[6] When a pipelined architecture is used, however, the BFP method is not suitable because of its huge latency to normalize all outputs from a certain stage. Instead, a method called convergent block floating point (CBFP) has been proposed for pipelined architectures.[7] As shown in Figure 2, the CBFP method also suffers from large memory overhead and increased latency caused by the intermediate buffer as well as complex normalization. Furthermore, the intermediate buffer in the CBFP logic has to store full-precision values because the normalization can be performed after the scaling factor is known.[8] Although a data scaling method that does not need additional buffers and latency has been proposed, it still requires the complex normalization that should be implemented with a number of compare and shift units connected in series at the output of each stage.[9]

The proposed data scaling technique is based on an observation that there is no need to scale down the internal value if overflow does not occur in computing the value. Even if overflow occurs, scaling down to half, which can be achieved by a simple operation of 1-bit right shift, is all to accommodate the overflow. Based on this observation, we present an efficient dynamic scaling technique. The main idea of the proposed algorithm is to conditionally scale down the output value of a complex multiplication if overflow occurs in the computation, and tagging this information on the internal word. We examine the output value of a complex multiply unit to check whether it can be represented in $n$ bits or not as shown in Figure 3(a). The overflow can be easily detected by performing an Exclusive-OR operation for two most significant bits (MSBs) shown in Figure 3(b). If overflow occurs in either the real value or the imaginary value, both the real value and the imaginary value is scaling down to half, which leads to less hardware complexity.

The internal word format of the proposed dynamic scaling method is shown in Figure 4(a). The data field in the internal word format is to represent the scaled data value, and the tag field is to indicate how many times the scalings are applied from the original input values to generate the corresponding data. If the proposed data scaling method is applied to the $L$-th stage, at most $\lceil \log_2 L \rceil$ bits are enough for the tag field. Therefore,

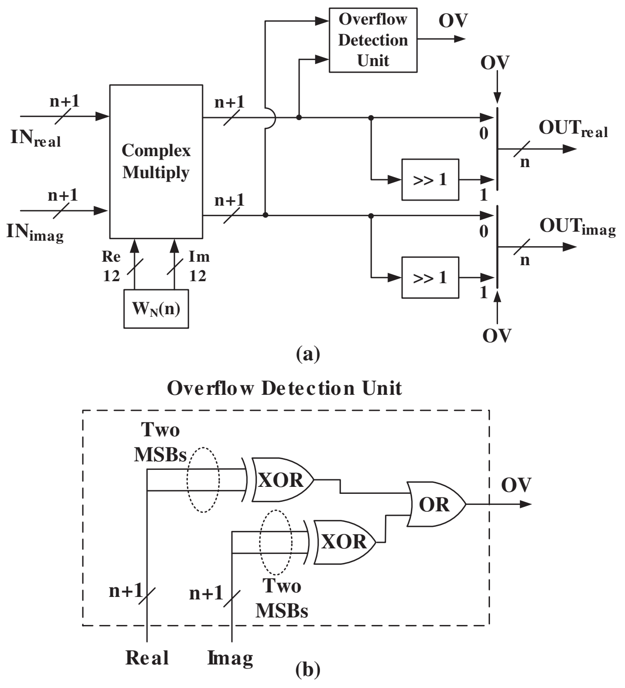

Fig. 3. Proposed dynamic scaling technique. (a) Dynamic scaling. (b) Overflow detection.

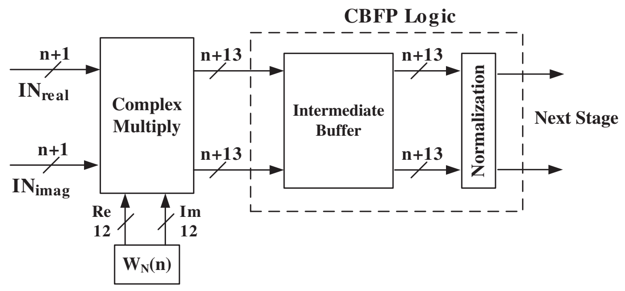

Fig. 2. CBFP logic in a pipeline stage.

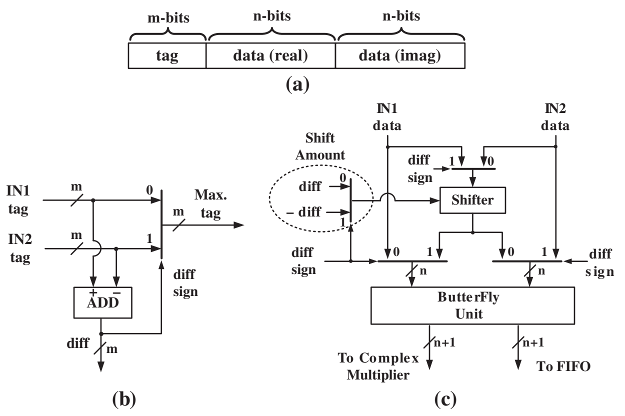

Fig. 4. Proposed processing unit. (a) Internal word format. (b) Tag processing. (c) Data processing.

the number of bits in the tag field increases gradually in log scale.

In general, the two data words participating in a butterfly computation have different tag values. As shown in Figures 4(b and c), the difference of the two tag values is calculated first, and then one data word with the smaller scale is shifted by the difference to make the scales of two data words equal. The tag value of the output word of a butterfly computation is initially set to the larger tag of the two input words. After the complex multiplication is completed, the output tag value is increased by one if overflow is detected.

At the final pipeline stage of $N$-point FFT, each output has the different tag value in general because each value experiences a different number of scalings. To obtain appropriate precision, the output is scaled up by the amount of the corresponding tag value. As the proposed conditional scaling technique makes the internal wordlength short, it leads to a lower hardware complexity without severe information loss.

## 4. PROPOSED 2048-POINT PIPELINED FFT

In pipelined DIF FFT processing, the twiddle factor table is largest at the first stage and reduced by a factor of two at the successive stages. Reducing the table sizes at the first several stages can be significant because the original table sizes are large enough to pay off the additional constant multipliers. The reduction is not considerable, however, at the latter stages. At each pipeline stage, 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 is not compensated by the table reduction at a certain stage, the radix-$2^2$ algorithm should be applied from that stage to the last stage.

By combining the proposed algorithm with radix-$2^2$ algorithm, we designed a 2048-point pipelined FFT processor of which Single-path Delay Feedback (SDF) structure is shown in Figure 5.[1] The overall table size can be reduced to almost half, compared to the structure that uses only radix-$2^2$ algorithm, by applying the proposed algorithm to the first four stages. In this case, two constant multipliers are required to compute non-trivial multiplications by $W_{2048}^1$ and $W_{512}^1$. In implementing the 2048-point FFT processor, the wordlength of the twiddle factors is set to 12 bits by performing several simulations. The longer twiddle factors are not cost efficient, as the SQNR performance is not increased notably but the costs of multipliers and tables are increased significantly.[4]

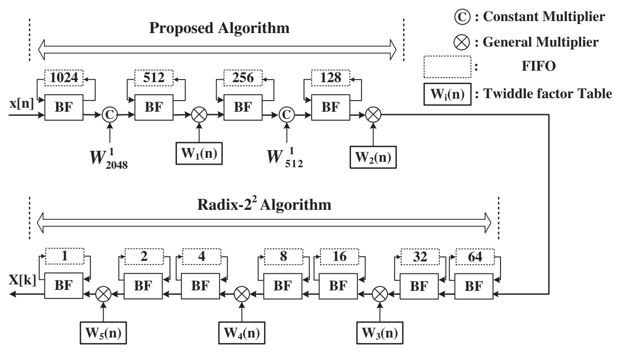

Fig. 5. SDF pipeline architecture of the proposed 2048-point FFT processor.

Table I. Complexity of constant multipliers.

| Constant operand | MSD representation |
|---|---|
| cos(2π/2048) | 1.000000000000 |
| sin(2π/2048) | 0.000000001100 |
| cos(2π/512) | 1.000000000000 |
| sin(2π/512) | 0.000000110010 |

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, the constants are expressed in the minimal signed digit (MSD) representation, as shown in Table I. Due to the sparse non-zero bits in the sine and the cosine values, the constant multipliers can be implemented with a few adders. By employing these two simple constant multipliers, we can reduce the required sizes of two largest tables to half. As the two tables takes more than 75% of the total table sizes required in the radix-$2^2$ algorithm, the reduction plays a significant role in lowering the overall complexity of the 2048-point FFT processor.

## 5. IMPLEMENTATIONS

We should the format of Sensor Letters. The hardware complexities required in the proposed algorithm and the previous algorithms are compared in Table II for the case of 2048-point FFT. The required table size indicates the total number of entries of the ROM tables. The $\pi/2$ symmetric property of the twiddle factors is considered in counting the table size. We can reduce the table size further if we employ the $\pi/4$ symmetric property. As the reduction ratio is independent of what symmetric property is used, the reduction ratio shown in Table II also applies to the case of $\pi/4$ symmetry. As indicated in Table II, the proposed algorithm needs the minimal table size compared to other algorithms and the overhead is just two constant multipliers which can be implemented with a few adders.

Assuming that the input is represented in 12 bits, we compare four scaling schemes shown in Table III. Table IV shows that the internal wordlength configurations and SQNR performances resulting from the scaling methods. If no scaling is used, the wordlength is increased progressively, one bit per stage to

Table II. Hardware complexity comparison for 2048-point FFT.

| FFT algorithm | Constant multiplier | General multiplier | Required table size |
|---|---|---|---|
| Radix-2 | 0 | 10 | 1023 (100%) |
| Radix-$2^2$ | 0 | 5 | 682 (66.7%) |
| Radix-$2^3$ | 3 | 4 | 584 (57.1%) |
| Proposed | 2 | 5 | 362 (35.4%) |

Table III. Scaling configurations.

| Case | Scaling method |
|---|---|
| I | Always scaling-to-half |
| II | No scaling + scaling-to-half |
| III | Proposed dynamic scaling always |
| IV | No scaling + proposed dynamic scaling |

Table IV. SQNR and internal wordlength of computational units.

| | Stage Number | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Case | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | SQNR (dB) |
| I | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 18.9 |
| II | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 19 | 19 | 19 | 56.1 |
| III | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 12 | 51.2 |
| IV | 12 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 13 | 55.8 |

Table V. Memory requirement for 2048-point FFT.

| Scaling method | FIFO memory requirement | Intermediate buffer requirement |
|---|---|---|
| Convergent block floating point (CBFP) | 2047 | 2046 |
| Proposed dynamic scaling | 2047 | 0 |

avoid overflow. On the contrary, both the proposed dynamic scaling technique and the scaling-to-half method maintain the internal wordlengths constant. As denoted in Table IV, the proposed dynamic scaling technique (case III) can improve SQNR impressively, about 30 dB improvement, compared to the scaling-to-half method (case I) although those two configurations have similar hardware complexity. If the first stage is allowed to extend one bit as in case IV, we can obtain a SQNR performance of more than 55 dB using the proposed dynamic scaling technique. To achieve a SQNR of more than 55 dB by progressively increasing the internal wordlength, the wordlength should be lengthened to 19 bits, leading to huge hardware complexity at the latter stages as in case II.

Compared to the CBFP method that requires additional buffers to store a group of values to be normalized, the proposed dynamic scaling method requires less memory as well as less computational delay. Table V shows memory sizes required to process 2048-point FFT. The memory requirement indicates the total sizes of FIFO memories and intermediate buffers. The memory overhead resulting from the CBFP method is enormous, as the full-precision values should be stored in intermediate buffers and the size of the buffers is comparable with that of the FIFO memories. In addition, the latency is also increased considerably by the intermediate buffers.

We designed a 2048-point pipelined FFT processor using a 0.18 $\mu$m 4-Metal CMOS process. The internal wordlengths are configured as indicated in case IV in Table III. The proposed FFT processor occupies 1.95 mm$^2$ and the gate count is 75,809 excluding memories and ROMs. The FIFO buffers are implemented using RAM memories, and small-sized RAM and ROM memories are replaced with registers and logic circuitry, respectively.

## 6. CONCLUSIONS

We should the format of Sensor Letters. We have proposed a new FFT algorithm to reduce the size of twiddle factor tables and an efficient dynamic scaling method to lower overall hardware complexity in the implementation of large-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 simple 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 large-point FFT computation, especially in terms of area and power consumption. Based on the proposed FFT algorithm, we can design a 2048-point pipelined FFT processor that reduces the total size of twiddle factor tables to 35% and 53% compared to the radix-2 and radix-$2^2$ algorithms, respectively. In addition, the proposed dynamic scaling technique enables the proposed processor to achieve SQNR of more than 55 dB without increasing the internal wordlength progressively.

**Acknowledgments:** This study was supported by the Research Program funded by the Seoul National University of Science and Technology.

## References and Notes

1. S. He and M. Torkelson, Design and implementation of a 1024-point pipeline FFT processor, *Proceedings IEEE Custom Integrated Circuits, Conference* **(1998)**, pp. 131–134.
2. S. He and M. Torkelson, Designing pipeline FFT processor for OFDM (de)modulation, *Proceedings IEEE URSI International Symposium Signals, System Electron.* **(1998)**, pp. 257–262.
3. W. Li and L. Wanhammar, A pipeline FFT processor, *Proceedings IEEE Workshop on Signal Processing Systems* **(1999)**, pp. 654–662.
4. S. Johansson, S. He, and P. Nilsson, Wordlength optimization of a pipelined FFT processor, *42nd Midwest Symposium on Circuits and Systems* **(1999)**, pp. 501–503.
5. J. W. Cooley and J. W. Tukey, *Math. Computation* 19, 297 **(1965)**.
6. Y. W. Lin, H. Y. Liu, and C. Y. Lee, *IEEE J. Solid-State Circuits* 39, 2005 **(2004)**.
7. E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, *IEEE J. Solid-State Circuits* 20, 300 **(1995)**.
8. J.-R. Choi, S.-B. Park, D.-S. Han, and S.-H. Park, A 2048 complex point FFT architecture for digital audio broadcasting system, *Proceedings of IEEE International Symposium on Circuits and Systems* **(2000)**, pp. 693–696.
9. T. Lenart and V. Owall, A 2048 complex point FFT processor using a novel data scaling approach, *Proceedings of IEEE International Symposium on Circuits and Systems* **(2003)**, pp. IV-45–IV-48.

Received: 15 February 2016. Accepted: 10 March 2016.