Twiddle Factor Transformation for Pipelined FFT Processing

In-Cheol Park, WonHee Son, and Ji-Hoon Kim
School of EECS, Korea Advanced Institute of Science and Technology, Daejeon, Korea
icpark@ee.kaist.ac.kr, {wson,sckid}@eeinfo.kaist.ac.kr

Abstract

This paper presents a novel transformation technique that can derive various fast Fourier transform (FFT) in a unified paradigm. The proposed algorithm is to find a common twiddle factor at the input side of a butterfly and migrate it to the output side. Starting from the radix-2 FFT algorithm, the proposed common factor migration technique can generate most of previous FFT algorithms without using mathematical manipulation. In addition, we propose new FFT algorithms derived by applying the proposed twiddle factor moving technique, which reduce the number of twiddle factors significantly compared with the previous algorithms being widely used for pipelined FFT processing.

1. Introduction

Fast Fourier Transform (FFT) is an important signal processing block being widely adopted in communication systems, especially in orthogonal frequency division multiplexing (OFDM) systems such as digital video broadcasting, digital subscriber line (xDSL) and WiMAX (IEEE 802.16). These applications require large-point FFT processing for multiple carrier modulation, such as 1024/2048/8192-point FFTs.

Based on the Cooley-Tukey (CT) decomposition [1], many FFT algorithms, such as radix-2^2 [2], radix-2^3 [3], radix-(4+2) [4], split-radix [5] as well as radix-4 algorithms, have been proposed using the complex mathematical relationship to reduce the hardware complexity. As the algorithms were derived based on intensive mathematical manipulation, it is not straightforward to understand the mathematical meaning and apply them to derive new FFT algorithms. The computational complexity and the hardware requirement are greatly dependent on the FFT algorithms in use.

In this paper, we propose a novel transformation which provides a general view that can unify almost all the FFT algorithms based on the CT decomposition. The proposed transformation is to find a common twiddle factor at the input stage of a butterfly pair of a stage and move it to other stages, which enables reinterpretation of the previous FFT algorithms and intuitive generation of new FFT algorithms without resorting to complicated mathematical manipulation. Two new FFT algorithms are presented in this paper, which are all derived using the proposed twiddle factor transformation to minimize the total size of tables required to store twiddle factors as well as the computational complexity especially in implementing pipelined FFT processors.

The rest of this paper is organized as follows. Section 2 describes the fundamentals of FFT decomposition focused on the Cooley-Tukey algorithm, and presents the previous FFT algorithms briefly. In Section 3, the proposed common twiddle factor migration is described in detail. Section 4 explains how to derive the previous FFT algorithm by using the proposed transformation. New FFT algorithms derived based on the proposed algorithms are presented and compared with the previous FFT algorithms in Section 5, and concluding remarks are made in Section 6.

2. Previous FFT algorithms

The N-point Discrete Fourier Transform (DFT) of an N-point sequence $\{x(n)\}$ is defined as

$$X[k] = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \; 0 \le k < N \qquad (1)$$

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

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

To reduce the computational complexity, a number of FFT algorithms have been developed by recursively applying the Cooley-Tukey decomposition algorithm [1] proposed in 1965.

2.1. Cooley-Tukey (CT) decomposition

The CT decomposition can be expressed as

$$X[k_1 + N_1 k_2] = \sum_{n_2=0}^{N_2-1} [(\sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) W_{N_1}^{k_1 n_1}) W_N^{k_1 n_2}] W_{N_2}^{k_2 n_2},$$
$$0 \le n_1, k_1 < N_1, \quad 0 \le n_2, k_2 < N_2, \quad N = N_1 \times N_2 \quad (3)$$

In this paper, N, N_1, and N_2 are assumed to be powers of 2, that is, $N = 2^{l+r}$, $N_1 = 2^l$, and $N_2 = 2^r$, where l and r are positive integers. Note that (3) involves two summations indexed by n_1 and n_2, which are referred as *inner DFTs* and *outer DFTs*, respectively. As a result, we can partition the N-point DFT into the N_1-point and N_2-point DFTs. Each output of the inner DFT is multiplied by twiddle factor $W_N^{k_1 n_2}$.

2.2. Previous FFT algorithms

Various FFT algorithms can be categorized by how to apply the CT decomposition. Let us begin with the radix-r FFT algorithms [6] in which the transform of length N is recursively decomposed into N/r and r until all the remaining transform lengths become less than or equal to r. The decomposition is sequentially conducted starting from the time or frequency domain, which classifies the decomposition into decimation-in-time (DIT) or decimation-in-frequency (DIF) decomposition. Fig. 1 shows the signal flow graphs of 16-point DIF FFT corresponding to the radix-2 algorithm. The value of r, radix, plays a major role in determining the efficiency and complexity.

The second class is the radix-2^n algorithms proposed to avoid the drawback of high-radix algorithms. The radix-2^2 algorithm [2] not only reduces the computational complexity but also retains the simple structure of the radix-2 algorithm. Although the overall signal flow is almost the same as the radix-2 algorithm, the number of stages requiring twiddle factor multiplication is reduced to half like the radix-4 algorithm. Due to the low computational complexity as well as the simple signal flow graph, many FFT processors recently developed are based on the radix-2^n algorithms [8][9].

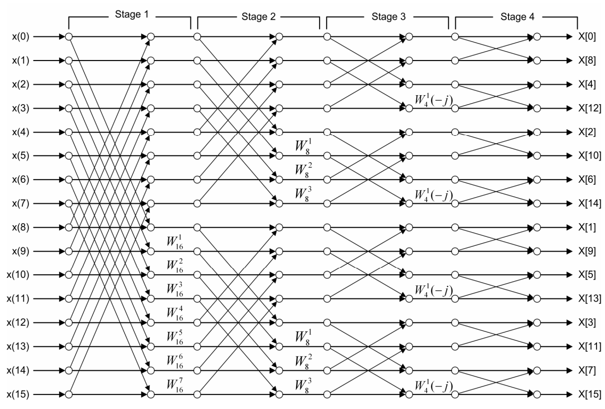


Figure 1. 16-point radix-2 FFT

3. Common twiddle factor migration

This section introduces common twiddle factors with their basic properties and explains the proposed common twiddle factor migration in detail. Starting from the radix-2 DIF FFT, we describe how to find and move the common twiddle factor to derive other FFT algorithms.

3.1. Common twiddle factors

Let us consider a butterfly pair shown in Fig. 2(a). If there is a factor common in the two input twiddle factors, it can be moved to the output side without affecting the functionality of the butterfly pair as shown in Fig. 2(b). As all twiddle factors of a stage have the same base, finding a common twiddle factor at the input side can be achieved by examining the exponents of the input-side twiddle factors. Moving the common twiddle factor to the output side is equivalent to multiplying the original twiddle factors of the output side by the migrated common factor, and the multiplication is actually to add the exponent of the common twiddle factor to those of the output-side twiddle factors when they have the same base.

Let us consider the signal flow graph of the 16-point radix-2 DIF FFT shown in Fig. 1. The bases of the twiddle factors are different depending on the stage.

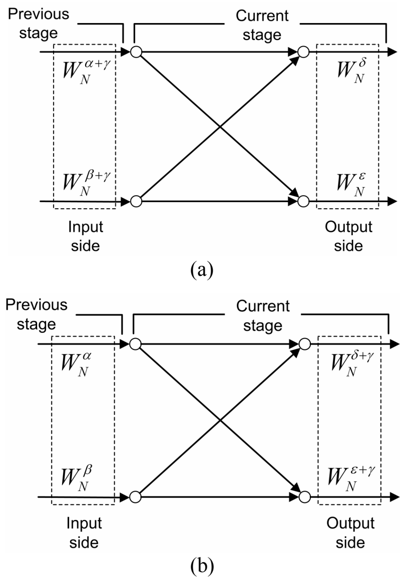


Figure 2. Butterfly pair (a) before and (b) after moving a common twiddle factor

To make it easy to manipulate the common factors, we normalize the bases to the largest one of the first stage. The normalization can be achieved based on the fact of $W_N^{\alpha} = W_{kN}^{k\alpha}$, where k is a non-zero integer. If the bases are normalized, specifying only the exponents is sufficient to represent the twiddle factors. For a 2^n-point FFT, n bits are sufficient to represent all the exponents of twiddle factors.

3.2. Finding common twiddle factors

Suppose that all the twiddle factors of a signal flow graph are normalized and represented with their exponents. To extract a common exponent at the input side of a butterfly pair, the exponents at the input side can be compared by performing the bit-wise AND operation. The common exponent obtained by using this method is called a *Boolean common factor*. Though we can imagine a more complex method that calculates the common factor algebraically, we deal with only the Boolean common factors in this paper, for simple manipulation.

The Boolean common factors can be classified into three types. First, a Boolean common factor obtained by the bit-wise AND operation of the input exponents is called a *max-common factor*, as it has the longest number of non-zero bits among common factors achievable with the input exponents. We can make another kind of common factor called a *min-common factor* by taking a single non-zero bit from the max-common factor. Taking more than one non-zero bits in a max-common factor results in a different common factor, called a *part-common factor* in this paper.

Fig. 3(a) redraws the lower part of the second stage of the 16-point FFT signal flow graph specifying only the exponents. It shows the relationship between the input twiddle factors in a stage, where we can see that the most significant bit (MSB) is always zero, the second MSB is not common, and the two least significant bits, colored gray, are common for all the butterfly pairs and a combination of the two bits is associated with a separate butterfly pair. Note that the exponent difference between the input twiddle factors of a butterfly pair is N/4=4. Fig. 3(b) shows a part of the third stage that has non-zero exponents at the input side.

As this relationship is held irrespective of stage location, we can treat all the butterfly pairs in a stage as a whole instead of dealing with each butterfly pair individually. Therefore, all the exponents of max-common twiddle factors in a stage can be denoted as a symbolic representation where two MSBs are zero, $(i-1)$ LSBs are zero for stage i, and between them the j-th bit of stage i is represented with $b_{i,j}$.

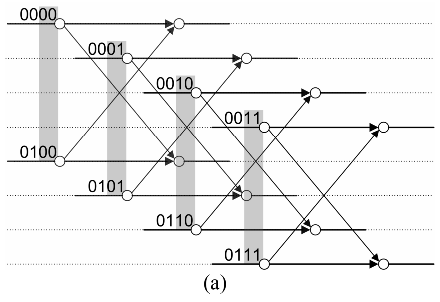
3.3. Moving range of a common twiddle factor

A max-common factor can move only to the very next stage, because the migrated max-common factor is not common any longer at the next stage. In contrast, a min-common or a part-common factor may be common at the next stage, implying that it is possible to move a min-common factor or a part-common factor by more than one stage.

The maximal moving range of a min-common factor is defined as the *moving span*. In general, a min-common factor corresponding to $b_{i,j}$ has a moving span of n-2-j, and thus the span is independent of what stage the bit is in. This implies that the moving span of a lower significant bit is longer than that of a higher significant bit. For a part-common factor, the moving span is determined by choosing the minimal value among the moving spans of min-common factors belonging to the part-common factor. When s_{ij} represents the moving span of a min-common factor corresponding to $b_{i,j}$, the moving span matrix $S = [\ s_{ij}\ ]$ with $0 < i \le n,\ 0 \le j < n$ is specified as follows.

$$s_{ij} = \begin{cases} n-2-j & \text{if } 0 \le j < n\text{-}2 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

3.4. Moving Operation and Relevant Matrices



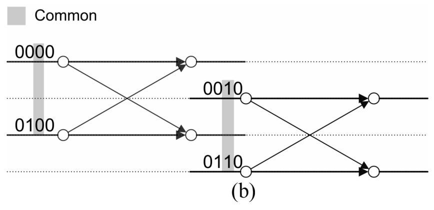


Figure 3. (a) The lower part of the second stage (b) a part of the third stage in 16-point DIF FFT signal flow graph

The common factor migration can be described as a matrix operation denoted by $\circledast$. Assume that the exponential term of a stage is represented by (b_{n-1}, b_{n-2}, …, b_0). Prior to describing the operation, let us define two operand matrices for 2^n-point FFT. One operand is an $n \times n$ matrix named *symbolic exponent matrix* $E = [e_{ij}]$, $0 < i \le n,\ 0 \le j < n$, whose entry is specified as follows.

$$e_{ij} = \begin{cases} 1 & \text{if there is an exponent whose } b_j \text{ is } 1 \text{ at the } i\text{-th stage} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

The twiddle factors of the radix-2 DIF FFT algorithm are used for constructing *E*. In practice, the *i*-th row vector of *E* can be obtained by performing the bit-wise OR operation of all the exponents at the *i*-th stage. The symbolic exponent matrix for 2^n-point FFT denoted by *E* is given as follows.

$$E = \begin{bmatrix} 0 & e_{1,n\text{-}2} & e_{1,n\text{-}3} & \cdots & e_{11} & e_{10} \\ 0 & e_{2,n\text{-}2} & e_{2,n\text{-}3} & \cdots & e_{21} & 0 \\ & & \cdots & & & \\ 0 & e_{n\text{-}2,n\text{-}2} & e_{n\text{-}2,n\text{-}3} & \cdots & 0 & 0 \\ 0 & e_{n\text{-}1,n\text{-}2} & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{bmatrix} \tag{6}$$

The other operand is an $n \times n$ matrix $M = [m_{ij}]$ named *moving matrix*, which is specified as follows,

$$m_{ij} = \begin{cases} r & \text{if the min-common factor corresponding to } b_j \\ & \text{at the } i\text{-th stage is to be moved by } r \text{ stages} \\ 0 & \text{if not to be moved} \end{cases} \tag{7}$$

where $0 \le r \le s_{ij}$.

The new symbolic exponent matrix *K* resulting from moving operation $\circledast$ is expressed as follows,

$$K = M \circledast E \tag{8}$$

where the moving operation migrates e_{ij} from the *i*-th stage to the (i+m_{ij})-th stage. As we can treat all the butterfly pairs in a stage as a whole instead of dealing with each butterfly pair individually, we are able to describe the migration by taking the symbolic exponent matrix instead of a matrix containing all the exponents of every stage.

4. Derivation of previous FFT algorithms

Most of the previous FFT algorithms can be explained by the proposed common factor moving method. Starting from the signal flow graph of the radix-2 DIF, we describe in this section how to apply the proposed method to generate the previous FFT algorithms such as radix-2^2 and DIT.

4.1. Radix-2^2 FFT algorithm

The radix-2^2 algorithm [2] is usually implemented by employing a butterfly pair accompanied with an additional multiplexer at every odd stage and a conventional one at every even stages. As the former can be implemented without a twiddle factor multiplier, the number of stages requiring twiddle factor multiplication is reduced to half. Therefore, the radix-2^2 algorithm leads to a more efficient hardware implementation than the radix-2 algorithm does.

Starting from the radix-2 DIF FFT algorithm, we first calculate the max-common factors at every odd stage and then move them to the next even stage at the right side. The FFT algorithm resulting from this procedure is exactly the same as the radix-2^2 algorithm if the property of $W_N^{N/4} = -j$ is applied. The following is the corresponding moving matrix for *N*=2048.

$$M_{2048} = \begin{bmatrix} 0&0&1&1&1&1&1&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&1&1&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&1&1&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \tag{9}$$

4.2. Radix-2 DIT FFT algorithm

Starting from the radix-2 DIF algorithm, we can derive the radix-2 DIT algorithm by moving every min-common factor to the last stage indicated by its moving span. The corresponding moving matrix for *N*=1024 is given as follows.

$$M_{1024} = \begin{bmatrix} 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 4 & 5 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (10)$$

5. New pipelined FFT algorithms and simulation results

The previous algorithms such radix-2^2 and radix-2^3 algorithms could reduce the hardware complexity, but they did not seriously take into account the number of twiddle factors. The CORDIC algorithm [7] enables the on-demand computation of twiddle factors, but requires multiple cycles to compute a twiddle factor, which is not suitable for high performance FFT processing. To achieve high performance, therefore, the twiddle factors are usually stored into ROM tables. In the implementation of a large-point FFT processing block, the tables become large enough to occupy significant area.

Among variety of FFT architectures, pipelined architecture is preferred since it can provide high performance with a moderate hardware cost [2][7]. Fig. 4 depicts a typical pipelined architecture based on the single-path delay feedback (SDF). The pipelined architectures for FFT processing are summarized well in [2]. In this section, two new pipelined FFT algorithms derived by applying the proposed common factor migration are presented.

5.1. Calculation of twiddle factor table size

Let us derive the hardware cost of twiddle factors required in a pipeline stage of FFT processing. For simplicity, it is assumed that all the twiddle factors required in a pipeline stage are stored in a ROM table with considering the symmetric property of twiddle factors.

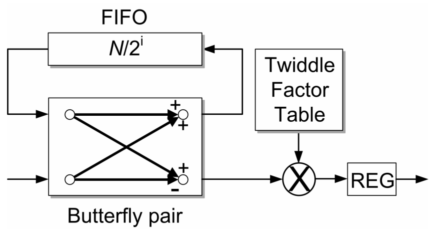


Figure 4. A typical pipelined architecture based on SDF (i-th stage)

In general, the i-th stage of 2^n-point FFT requires 2^{n-i} twiddle factors if the FFT is processed based on the radix-2 algorithm. This fact can be induced by looking at the number of 1's in the symbolic exponent matrix. If we take into account the $\pi/2$ symmetric property, two MSB bits are ignored in counting the number of non-zero bits, which means that the i-th stage of 2^n-point FFT requires 2^{n-1-i} twiddle factors. Therefore, the twiddle factor table required in the stage can be implemented using a ROM of 2^{n-1-i} entries. Although all the entries of the ROM are not occupied by valid twiddle factors, the ROM size defined above makes sense if we consider the conventional ROM that has consecutive entries of a power of two.

5.2. Modified radix-2^2 FFT algorithm

This algorithm is derived by imposing a constraint on the migration of the some common factors when deriving the radix-2^2 algorithm from the radix-2 DIF algorithm. If the right-most min-common factor at every odd stage is not allowed to move to its consecutive even stage in the common factor migration, the number of twiddle factors required in the even stage can be reduced to half. The moving matrix for 2048-point FFT is as follows.

Table 1. Comparison of ROM size and hardware complexity with previous works

FFT length	1024				2048				8192			
	TE[a]	TE(%)	GM[b]	CM[c]	TE	TE(%)	GM	CM	TE	TE(%)	GM	CM
Radix-2	508	149	7	1	1020	149	8	1	4092	150	10	1
Radix-2^2	340	100	4	0	680	100	4	1	2728	100	5	1
Radix-2^3, 4+2	292	86	3	3	584	86	3	3	2340	86	4	4
Proposed (A)[d]	178	**52.4**	4	3	344	**50.6**	4	4	1368	**50.2**	5	5
Proposed (B)[e]	116	**34.1**	4	0	164	**24.1**	5	0	516	**18.9**	6	0

[a]Total number of entries in twiddle factor tables
[b]Total number of general complex multipliers, [c]Total number of constant complex multipliers
[d]Modified radix-2^2 FFT algorithm, [e]Evenly-distributed radix-2^2 FFT algorithm

$$M_{2048} = \begin{bmatrix} 0&0&1&1&1&1&1&1&1&1&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&1&1&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \qquad (11)$$

The reduction in the table size is achieved by employing an additional constant multiplier at every second stage. Since the table size reducible is huge and the complexity of the constant multipliers is low in the former stages, we can improve the area-efficiency significantly by applying the modified radix-2^2 algorithm to only the first several stages with small increase in computational hardware complexity.

5.3. Evenly-distributed radix-2^2 algorithm

Even-distribution of non-zero bits can reduce the computational complexity. If we selectively move min-common factors at the lower bit positions when generating the radix-2^2 FFT algorithm, we obtain a new FFT algorithm which reduces the table size while keeping the hardware complexity low. The moving matrix for 2048-point FFT is presented in (12). The hardware complexity and the table size of this FFT algorithm are summarized in Table 1.

$$M_{2048} = \begin{bmatrix} 0&0&1&1&3&3&3&3&5&7&9\\ 0&0&0&0&2&2&2&2&6&8&0\\ 0&0&1&1&1&1&1&1&7&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&1&5&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix} \qquad (12)$$

6. Conclusion

In this paper, we have presented a new transformation technique that unifies most of FFT algorithms. The proposed method is to migrate the common twiddle factor from a certain stage to other stage. We have shown that most of radix-2 based FFT algorithms can be generated by applying the proposed technique to radix-2 DIF FFT. Furthermore, we have presented two novel FFT algorithms derived by applying the proposed algorithm, which can drastically reduce the total size of twiddle factor tables while not significantly increasing the hardware complexity. For example, an 8192-point pipelined FFT processor designed based on the proposed transformation could reduce the total size of twiddle factor tables by about 80% when compared to the conventional radix-2^2 FFT widely adopted in hardware implementation.

Acknowledgements

This work was supported by Institute of Information Technology Assessment through the ITRC.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for machine computation of complex Fourier series," *Math, Comp.,* vol. 19, pp. 297–301, Apr. 1965.

[2] S. He and M. Torkelson, "Design and Implementation of a 1024-point pipeline FFT processors," in *Proc. IEEE Custom Integr. Circuits Conf.,* May 1998, pp. 131–134.

[3] S. He and M. Torkelson, "Designing pipeline FFT processor for OFDM (de)modulation," in *Proc. IEEE URSI Int. Symp. Signals, Syst. Electron.,* Sept. 1998, pp. 257–262.

[4] 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.

[5] P. Duhamel and H. Hollomann, "Split radix FFT algorithm," *Electron. Lett.,* vol. 20, no. 1, pp. 14–16, Jan. 1984.

[6] L. R. Rabiner and B. Gold, *Theory and Application of Digital Signal Processing.* Englewood Cliffs, NJ:Prentice-Hall, 1975.

[7] A. M. Despain, "Fourier transform computers using CORDIC iterations," *IEEE Trans. Comput.,* vol. C-23, pp. 993–1001, Oct. 1974.

[8] 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.

[9] W. H. Chang and T. Nguyen, "An OFDM-specified lossless FFT architecture," *IEEE Trans. Circuits Syst. I,* vol. 53, no. 6, pp. 1235–1243, June 2006.