Balanced Binary-Tree Decomposition for Area-Efficient Pipelined FFT Processing

Hyun-Yong Lee, *Student Member, IEEE*, and In-Cheol Park, *Senior Member, IEEE*

***Abstract*—This paper presents an area-efficient algorithm for the pipelined processing of fast Fourier transform (FFT). The proposed algorithm is to decompose a discrete Fourier transform (DFT) into two balanced sub-DFTs in order to minimize the total number of twiddle factors to be stored into tables. The radix in the proposed decomposition is adaptively changed according to the remaining transform length to make the transform lengths of sub-DFTs resulting from the decomposition as close as possible. An 8192-point pipelined FFT processor designed for digital video broadcasting—terrestrial (DVB-T) systems saves 33% of general multipliers and 23% of the total size of twiddle factor tables compared to a conventional pipelined FFT processor based on the radix-2^2 algorithm. In addition to the decomposition, several implementation techniques are proposed to reduce area, such as a simple index generator of twiddle factor and add/subtract units combined with the two's complement operation.**

***Index Terms*—Balanced binary-tree decomposition, digital video broadcasting—terrestrial (DVB-T), fast Fourier transform (FFT), orthogonal frequency division multiplexing (OFDM), pipelined processing.**

I. Introduction

FAST Fourier transform (FFT) is an important signal processing block being widely used in communication systems, especially in orthogonal frequency division multiplexer (OFDM) systems such as digital video broadcasting—terrestrial (DVB-T) [24], digital audio broadcasting (DAB), very high-speed digital subscriber line (VDSL) and IEEE 802.16e. These applications require large-point FFT processing for multiple carrier modulation, such as 8192/2048-point FFT in DVB-T and 8192/4096/2048/1024/512/256-point FFT in VDSL.

Many FFT algorithms have been proposed, such as radix-2^2 [11], radix-2^3 [12], radix-$4 + 2$ [13], split-radix [14], [15], vector-radix [16], and prime factor algorithms [17], to reduce the hardware complexity of the multiplier, butterfly (BF) and controller. 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 the FFT processing. The twiddle factor, which is actually the sine and cosine values of a phase, can be computed on demand by using the COordinate Rotation DIgital Computer (CORDIC) algorithm [18]. As the CORDIC algorithm takes multiple cycles to compute a twiddle factor or requires a lot of hardware resources. However, the on-the-fly twiddle factor computation 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 processor, the table becomes so large that it occupies significant area.

In this paper, we propose a new decomposition algorithm to minimize the total size of tables needed to store twiddle factors, while preserving the advantages of previous algorithms. The proposed algorithm is to decompose a discrete Fourier transform (DFT) such that the transform lengths of sub-DFTs resulting from the decomposition are almost equal. In addition, we present a simple method to generate the table address required to read a specific twiddle factor, and a twiddle factor recoding method to achieve area-efficient multipliers.

To efficiently process FFT, a number of hardware architectures have been proposed, including serial processing on a single processor [2], pipelined processing [7]–[10], and parallel processing [3]–[6]. Among them, the pipelined processing is preferred, as it can provide a high performance with a moderate hardware cost. Based on the proposed decomposition algorithm, an 8192-point pipelined FFT processor is implemented, which saves 33% of general multipliers and 23% of the table size of twiddle factors compared to a conventional pipelined radix-2^2 FFT processor.

The rest of this paper is organized as follows. Section II describes the fundamentals of DFT decomposition and previous FFT algorithms. In Section III, a binary tree representation is defined to illustrate the decomposition procedure of FFT algorithms, and an efficient decomposition algorithm is proposed through the analysis of hardware complexity. To reduce area in practical implementation, several techniques are presented in Section IV, including a pre-encoding scheme for constant multiplication and an address generation scheme to efficiently access twiddle factor tables. An 8192-point FFT processor designed for DVB-T applications is described in Section V and compared with previous implementations. Finally, concluding remarks are made in Section VI.

II. FFT Algorithms

The N-point DFT of a set of N-point signals $\{x(n)\}$ is defined as

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

Manuscript received February 6, 2006; revised August 23, 2006. This work was supported by Institute of Information Technology Assessment through the ITRC and by IC Design Education Center (IDEC). This paper was recommended by Associate Editor S.-M. Phoong.

The authors are with the Division of Electrical Engineering, the School of Electrical Engineering and Computer Science, Korea Advanced Institute of Science and Technology (KAIST), Daejeon 305-701, Korea (e-mail: hylee@ics.kaist.ac.kr; icpark@ee.kaist.ac.kr).

Digital Object Identifier 10.1109/TCSI.2006.888764

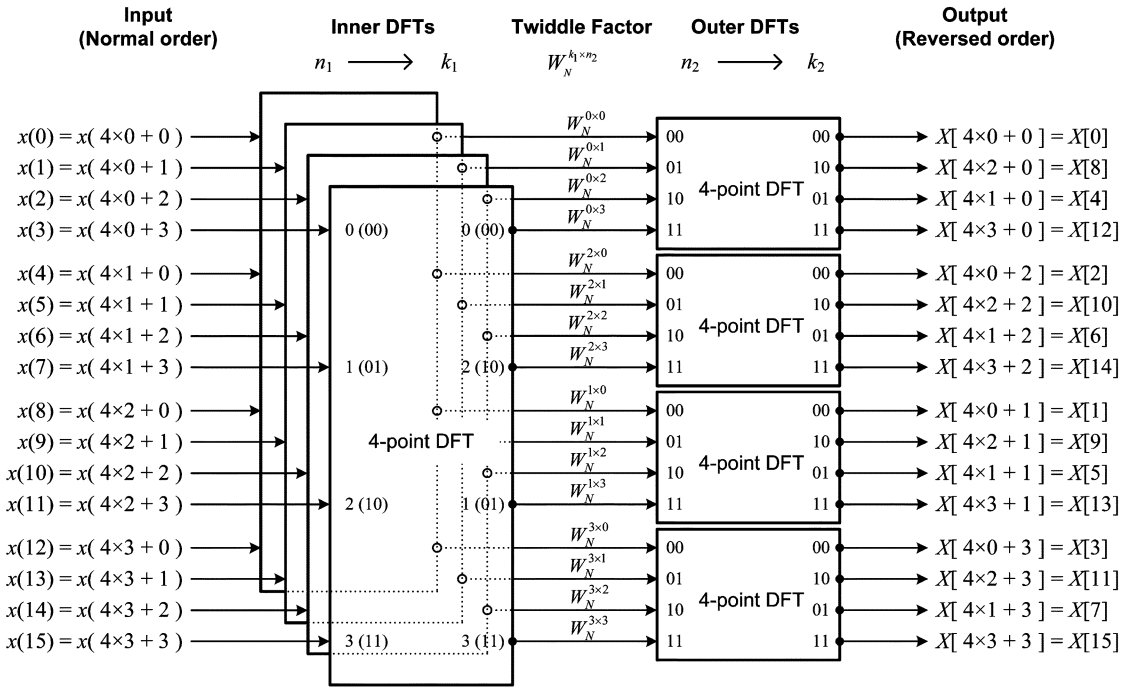


Fig. 1. Signal flow graph of CT algorithm when the transform length is a power of 2.

where $W_N = \exp(-j2\pi/N)$. To reduce the computational complexity, Cooley and Tukey proposed an efficient decomposition algorithm [1] in 1965. In this section, we briefly explain the Cooley–Tukey (CT) algorithm and previous FFT algorithms derived from the CT algorithm.

A. CT Algorithm

The CT algorithm can be expressed as

$$X[k_1 + N_1 k_2] = \sum_{n_2=0}^{N_2-1} \left[\left(\sum_{n_1=0}^{N_1-1} x(N_2 n_1 + n_2) W_{N_1}^{k_1 n_1} \right) W_N^{k_1 n_2} \right] W_{N_2}^{k_2 n_2}, \quad 0 \le n_2, k_1 < N_1; \quad 0 \le n_1, k_2 < N_2; \quad N = N_1 \times N_2. \tag{2}$$

In this paper, N, N_1, and N_2 are assumed to be powers of two to concentrate on FFT, that is, $N = 2^{l+r}, N_1 = 2^l$ and $N_2 = 2^r$, where l and r are positive integers. Note that two summations indexed by n_1 and n_2 are involved in (2), which are referred to as *inner DFTs* and *outer DFTs*, respectively. As a result, we can break the N -point DFT into the N_1-point and N_2-point DFTs. The signal flow of 16-point DFT derived from (2) is depicted in Fig. 1, where the left-hand and right-hand sides represent the 4-point inner DFT processing and 4-point outer DFT processing, respectively. Employing the CT decomposition, the DFT computation can be divided into three steps:

1) *Inner DFT Processing:* The inner DFT operation is equivalent to N_2 DFTs of length N_1. As there are N_2 inner DFTs each of which processes N_1 input signals, the N input signals, $\{x(N_2 n_1 + n_2)\}$, are partitioned into N_2 groups according to n_2. An inner N_1-point DFT processes a group of input signals indexed with the same n_2. As the output sequence of the conventional FFT processing is usually the bit-revered order of the input sequence, the output sequence of the inner DFT is assumed to be bit-revered in Fig. 1.
2) *Twiddle factor multiplication:* Each output of the inner DFT is multiplied by $W_N^{k_1 n_2}$ called a *twiddle factor*. Since the twiddle factor is indexed by $k_1 \times n_2$, the sequences of k_1 and n_2 are important to determine the index. As can be seen in Fig. 1, the sequence of n_2 is independent of the structure of the inner DFT, and remains in the normal sequential order. In case of k_1, however, the sequence of k_1 is the fully bit-reversed order of n_1, as the N_1-point inner DFT generates a fully bit-reversed output sequence.
3) *Outer DFT processing:* The summation with index n_2 is equivalent to N_1 DFTs of length N_2. The i-th outer DFT processes all the i-th output signals of the inner DFTs multiplied by the corresponding twiddle factor. The structure of inner DFTs does not affect the computation of outer DFTs, and only n_2 is concerned in computing an outer DFT.

B. Previous FFT Algorithms

To reduce the computational complexity of DFT, a number of FFT algorithms have been developed by recursively applying the CT decomposition algorithm. The FFT algorithms can be categorized by how to apply the CT decomposition. Let us begin with the radix-r FFT algorithms [7] in which the transform of length N is recursively decomposed into N/r and r until all the remaining transform lengths are less than or equal to r. Fig. 2 shows the signal flow graphs of 16-point FFT corresponding to radix-2 and radix-4 algorithms, where the decomposition order is indicated at the top and twiddle factors involved in the multiplication are marked with black dots on the circle of radius 1 at the bottom. The radix r plays a major role in determining the efficiency and complexity. Compared to the radix-2 algorithm, the

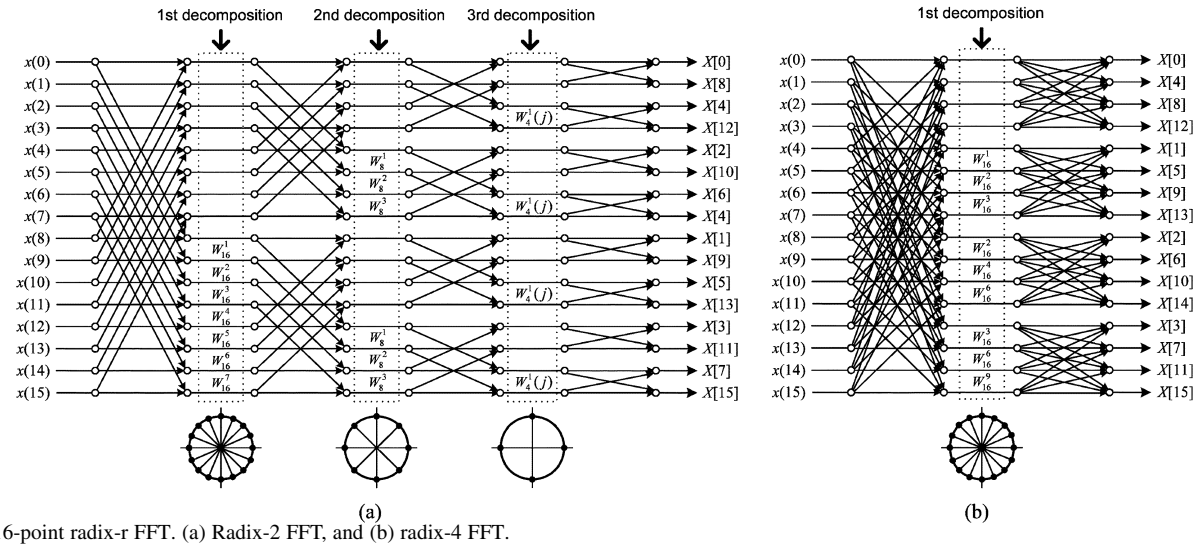


Fig. 2. 16-point radix-r FFT. (a) Radix-2 FFT, and (b) radix-4 FFT.

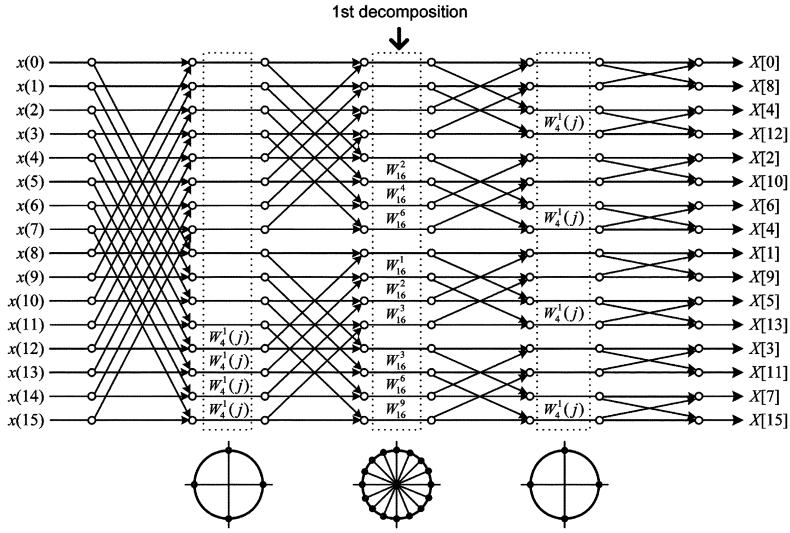


Fig. 3. 16-point radix-2^2 FFT.

radix-4 algorithm results in an architecture that has half stages and half multipliers, but the BF structure and signal flow control become more complex. In general, a high radix reduces the number of processing stages but increases the hardware complexity of a stage significantly.

The second class is the radix-2^n algorithms proposed to overcome the drawback of high-radix algorithms. In the radix-2^2 algorithm [11] shown in Fig. 3, the basic unit of decomposition consists of the radix-2 BF type-I (BF2-I), which is equal to the conventional radix-2 BF, and radix-2 BF type-II (BF2-II) that has additional multiplexers. Although the overall signal flow is almost the same as the radix-2 algorithm, the number of stages requiring twiddle factor multiplications is reduced to half like the radix-4 algorithm. In case of the radix-2^3 algorithm [12], the basic unit of decomposition consists of the BF2-I, BF2-II, and radix-2 BF type-III (BF2-III) that is associated with an additional constant multiplier in BF2-II. Therefore, the general multiplier required in every third stage is replaced with a constant multiplier. The radix-2^n algorithm can be explained by applying the CT algorithm two times. The transform is first decomposed into 2^k-point DFTs like the conventional radix-2^n algorithm, and then each 2^k-point sub-DFT is decomposed into k DFTs of length 2.

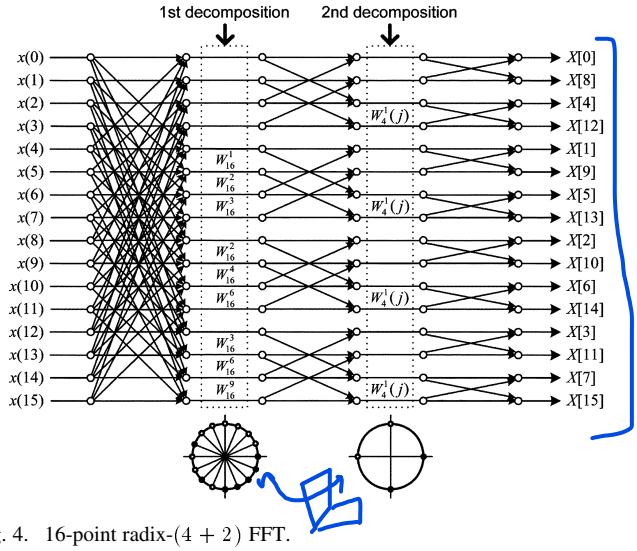


Fig. 4. 16-point radix-$(4+2)$ FFT.

The third class is the mixed-radix algorithms that can be derived by mixing different radixes, such as the radix-$(4+2)$ FFT algorithm [13] to which the radix-4 and radix-2 algorithms are alternately applied. The signal flow graph of 16-point radix-$(4+2)$ FFT is shown in Fig. 4. Although this algorithm results in

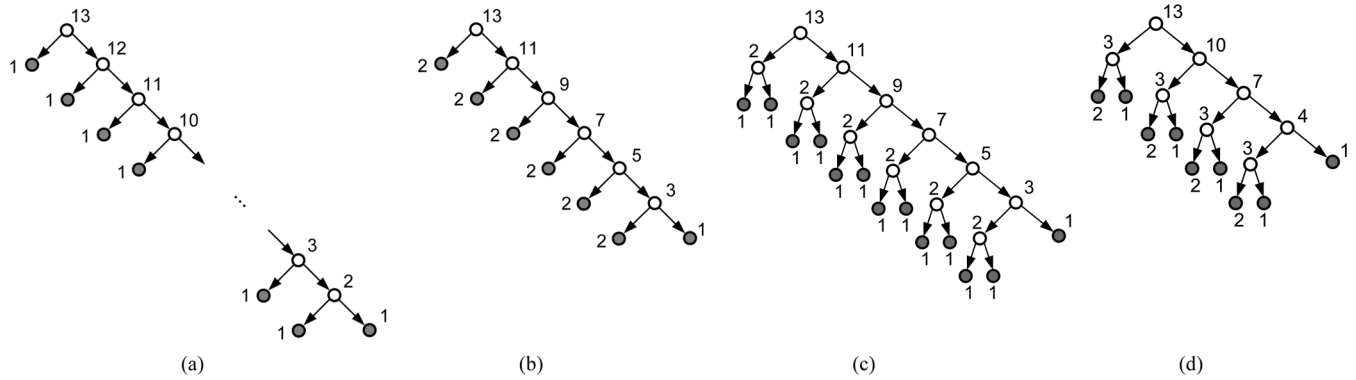


Fig. 5. Binary tree representation of the previous FFT algorithms. (a) Radix-2 DIF algorithm, (b) radix-4 DIF algorithm, (c) radix-22 DIF algorithm, and (d) radix-$(4 + 2)$ DIF algorithm.

more efficient processing, the hardware complexity is similar to that of the radix-2^3 algorithm.

Besides the above algorithms, there are many other algorithms such as split-radix [14], [15], vector-radix [16] and prime factor algorithms [17]. In the split-radix algorithm, the total number of complex multiplications can be reduced, but each stage becomes irregular. Since these algorithms are not efficient in terms of pipelined processing as well as control complexity due to the irregularity, they are not considered any more in this paper.

In the previous algorithms, a DFT is sequentially decomposed starting from the time or frequency domain. According to the decomposition direction, they can be classified into decimation-in-time (DIT) and decimation-in-frequency (DIF) algorithms.

III. Proposed Decomposition Algorithm

The proposed balanced decomposition is described in this section. Introducing a binary tree representation to show the overall decomposition procedure, we explain how to compute the size of twiddle factor tables and why the proposed balanced decomposition is effective in reducing the twiddle factor tables.

A. Binary Tree Representation

The CT decomposition algorithm is to divide a large-point DFT into smaller-point DFTs. In order to intuitively represent the recursive decomposition, we introduce a binary tree representation. In the binary tree representation, a weight w is assigned to each node to represent a set of 2^w-point DFTs. Let $w(u)$ be the weight of a node u. If a 2^{l+r}-point DFT is decomposed into 2^l-point DFTs and 2^r-point DFTs by applying (2), the node of weight $(l + r)$ has a left child of weight l and a right child of weight r. The left child corresponds to the set of 2^r inner DFTs of 2^l points and the right child to the set of 2^l outer DFTs of 2^r points. Therefore, the weight of the parent node is always equal to the sum of the weights of their child nodes. When the tree structure is finalized, all the leaves correspond to BFs, and internal nodes represent the twiddle factor multiplication required to connect the two set of child DFTs.

With this definition, a binary tree can be constructed for the given transform length. Fig. 5 illustrates four different binary trees constructed for 8192-point FFT each of which represents a previous FFT algorithm. As shown in Fig. 5, the binary trees corresponding to the previous FFT algorithms are not balanced. In this paper, the previous FFT algorithms are classified into unbalanced decomposition to distinguish them with the proposed balanced decomposition.

TABLE I
The $\pi/2$ Symmetric Property of Twiddle Factors

Twiddle factors	Values	Sign inversion		Real and imaginary swapping
		cos(i)	sin(i)	
W_N^i, $0 \le i < N/4$	cos(i)-$j$sin(i)	0	1	0
$W_N^{N/4+i}$, $0 \le i < N/4$	-sin(i)-$j$cos(i)	1	1	1
$W_N^{2N/4+i}$, $0 \le i < N/4$	-cos(i)+$j$sin(i)	1	0	0
$W_N^{3N/4+i}$, $0 \le i < N/4$	sin(i)+$j$cos(i)	0	0	1

B. 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 in this section that all the twiddle factors required in a pipeline stage are saved in a table with considering the symmetric property of twiddle factors. To analyze the size of a table, we consider the $\pi/2$ symmetric property of twiddle factors summarized in Table I. Of course, the $\pi/4$ symmetric property can be employed, but the analysis is almost the same except constant scaling.

In the binary tree representation, a leaf u of weight 1 means a set of 2-point DFTs that do not require multiplication as shown in Fig. 6(a), and an internal node represents twiddle factor multiplication. As shown Fig. 6(b), an internal node of weight 2 requires 4 multiplications. Three of them are trivial multiplications by $W_4^0 = 1$ and the other is a multiplication by $W_4^1 = -j$, meaning that only one twiddle factor is sufficient if we take into account the $\pi/2$ symmetry. As the multiplication by - j can be realized by exchanging the real and imaginary values as shown in Table I, the internal node of weight 2 can be implemented without multiplications. However, we count the number as 1 to make the analysis simple. As shown Fig. 6(c), an internal node of weight 3 requires two twiddle factors. In general, the number of twiddle factors needed for an internal node u is not greater

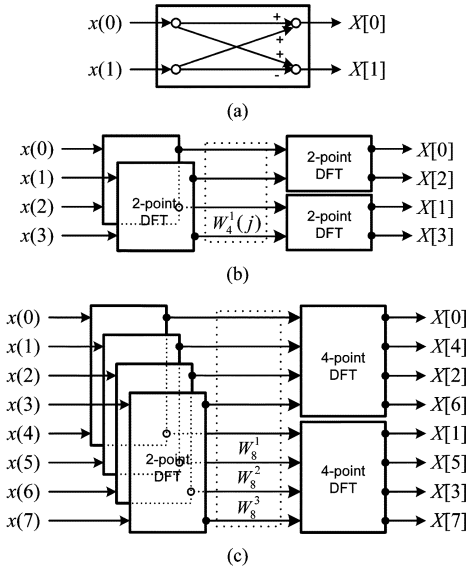


Fig. 6. The number of twiddles factors. (a) 2-point DFT, (b) 4-point DFT, and (c) 8-point DFT.

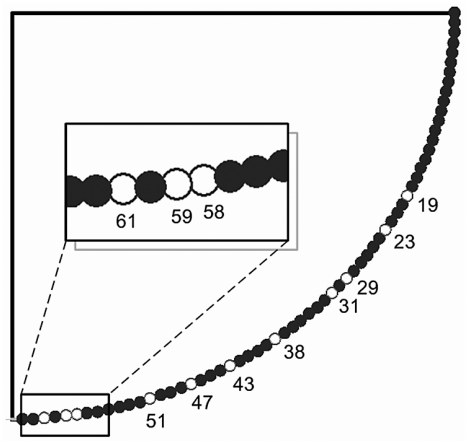


Fig. 7. Twiddle factors used in an internal node of weight 8 that is decomposed into two child nodes of weight 4.

than $2^{w(u)-2}$, if the $\pi/2$ symmetry is taken into account. Therefore, the table size needed for an internal node u can be defined as

$$c(u) = 2^{w(u)-2}. \tag{3}$$

Equation (3) does not mean that all the entries of a table are occupied by valid twiddle factors. As shown in Fig. 7, some points within $\pi/2$, which are marked with white dots, are not used as twiddle factors. This is evident from Fig. 1 showing how the twiddle factor index is generated. As multiplying k_1 by n_1 generates the index, some values such as prime numbers cannot be produced. If we employ the conventional ROM that has consecutive 2^k entries as a twiddle factor table, the table size defined in (3) makes sense even though some entries are useless.

As the total size of tables can be found by summing all the costs of non-leaf nodes, the cost of a binary tree T is defined as

$$C(T) = \sum_{\forall u \in N-\mathrm{L}} c(u) \tag{4}$$

where N is the set of all nodes and L is the set of leaves.

C. Balanced Decomposition

The first objective of the proposed algorithm is to minimize the tree cost when the transform length is given, and the next is to make all the leaf nodes become 2-point DFTs in order to produce a simple hardware structure. The second objective can be realized by allowing all the leaves in the binary tree representation to have a weight of 1. The cost optimization problem can be defined as follows.

Definition 1: Cost Optimization Problem: Given the root weight, find a binary tree associated with the minimum cost in which all the leaves have weight 1.

In case of single decomposition, the optimal solution can be found by minimizing the cost sum of two child nodes.

Theorem 1–Local Optimization: Consider a tree T consisting of a node u and its two child nodes, l and r. The cost of T is minimized if the node u is decomposed such that the weights of the children are as balanced as possible, i.e., $w(l) = \lceil w(u)/2 \rceil$ and $w(r) = \lfloor w(u)/2 \rfloor$, or $w(l) = \lfloor w(u)/2 \rfloor$ and $w(r) = \lceil w(u)/2 \rceil$, where $\lceil x \rceil$ and $\lfloor x \rfloor$ represent a minimal integer not less than x and a maximal integer not greater than x, respectively.

Proof: From (3) and (4), the cost sum of nodes, u, l and r, is given by $C\ (T) = c(u) + c(l) + c(r) = (2^{w(u)} + 2^{w(l)} + 2^{w(r)})/4$. By differentiating this function, we can find that the minimal value is obtained if the $w(l)$ and $w(r)$ are equal to each other. As only integer values are allowed for $w(l)$ and $w(r)$, the weights should be as close as possible to each other. QED.

A binary tree satisfying Theorem 1 can be constructed by symmetrically combining two nodes starting from leaves to the root (bottom-up approach). If the number of leaves is a power of two, it is apparent that this construction results in a tree of the minimal cost. As the root weight in such a construction scheme is restricted to a power of two, the bottom-up approach is not suitable for 1024, 2048, 4096, and 8192-point FFTs. To find a solution applicable to a root node whose weight is not a power of two, we can try a greedy algorithm that recursively applies Theorem 1 from the root to leaves (top-down approach).

Theorem 2–Global Optimization: The tree cost is minimized if all the internal nodes are decomposed such that the weights of their child nodes are as balanced as possible.

Proof: To verify that the top-down greedy approach leads to an optimal solution, it should be proved that the cost of a balanced tree is always not greater than that of any imbalanced tree. Due to Theorem 1, the principle of optimality holds during the top-down approach of tree construction, and thus this theorem is intuitively true. The exact proof is so lengthy that it is not described in this paper. QED.

In other words, a *balanced tree* leads to a solution that is optimal in terms of the total size of twiddle factor tables. For small-point FFTs, the balanced decomposition produces the same signal flow graphs as the radix-2^n algorithm. For example, the signal flow graphs resulting from the balanced decomposition for 16-point and 64-point FFTs are exactly equal to those from the radix-2^2 and radix-2^3 algorithms, respectively. If the transform length is greater than 64, however, the balanced decomposition leads to different signal flow graphs associated with smaller multipliers and twiddle factor tables.

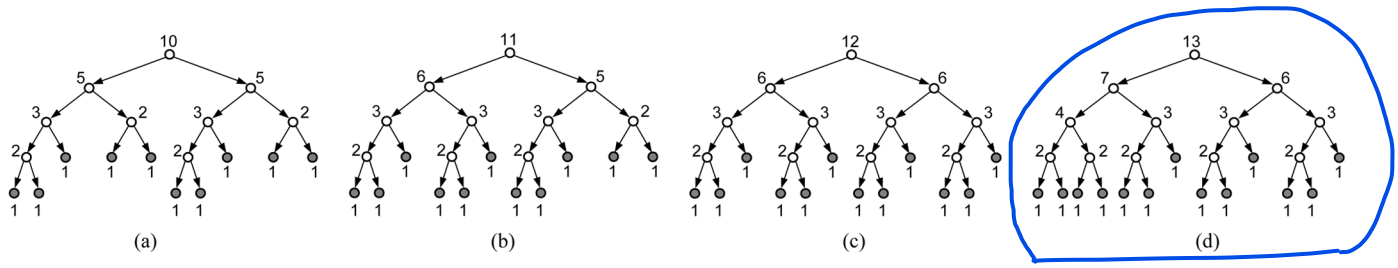


Fig. 8. Binary trees resulting from the proposed decomposition. (a) 1024-point FFT, (b) 2048-point FFT, (c) 4096-point FFT, and (d) 8192-point FFT.

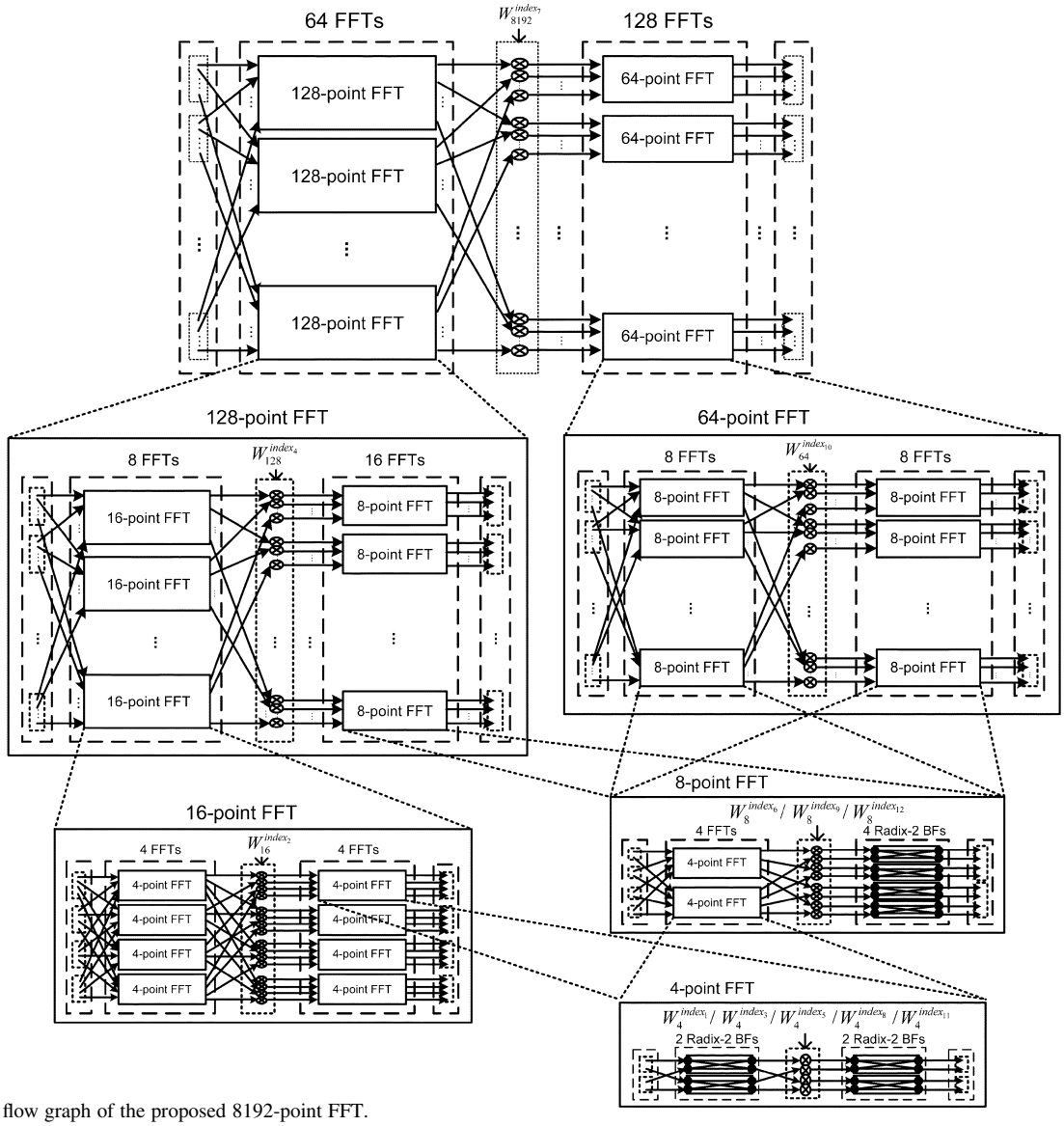


Fig. 9. Signal flow graph of the proposed 8192-point FFT.

Based on the proposed decomposition, the balanced binary trees corresponding to 1024-, 2048-, 4096-, and 8192-point FFTs are derived as shown in Fig. 8. Since the decomposition proceeds toward both of time and frequency domains, the proposed algorithm is neither DIT nor DIF.

In Fig. 9, the signal flow graph resulting from the proposed balanced decomposition is illustrated for 8192-point FFT. The graph is subdivided to reduce the drawing complexity.

IV. Area-Efficient Implementation of Pipeline Stages

This section describes several techniques that are effective in reducing the area required to implement a pipeline stage. A general pipeline stage based on the single delay feedback (SDF) structure [8] is shown in Fig. 10, which consists of a number of blocks such as address generator, twiddle factor ROM, FIFO, BF operator, and twiddle factor multiplier. The operation of the SDF structure is not described in this paper, and we recommend the reader to refer [8] for detailed description. In this section, we describe how each block can be implemented to reduce area.

A. Address Generation for Twiddle Factor Tables

Let us consider the case that a node u is decomposed into two child nodes, l and r. If we include the trivial multiplication by W_N^0, there are N twiddle factor multiplications in node u,

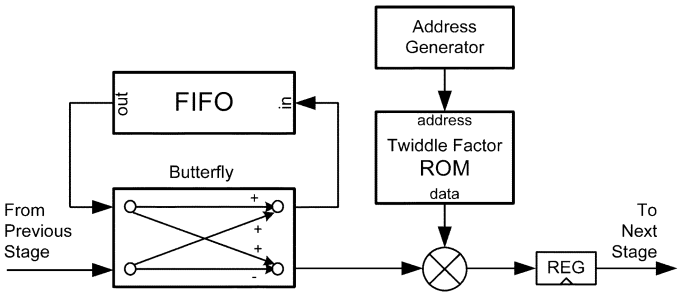


Fig. 10. General pipeline stage based on SDF.

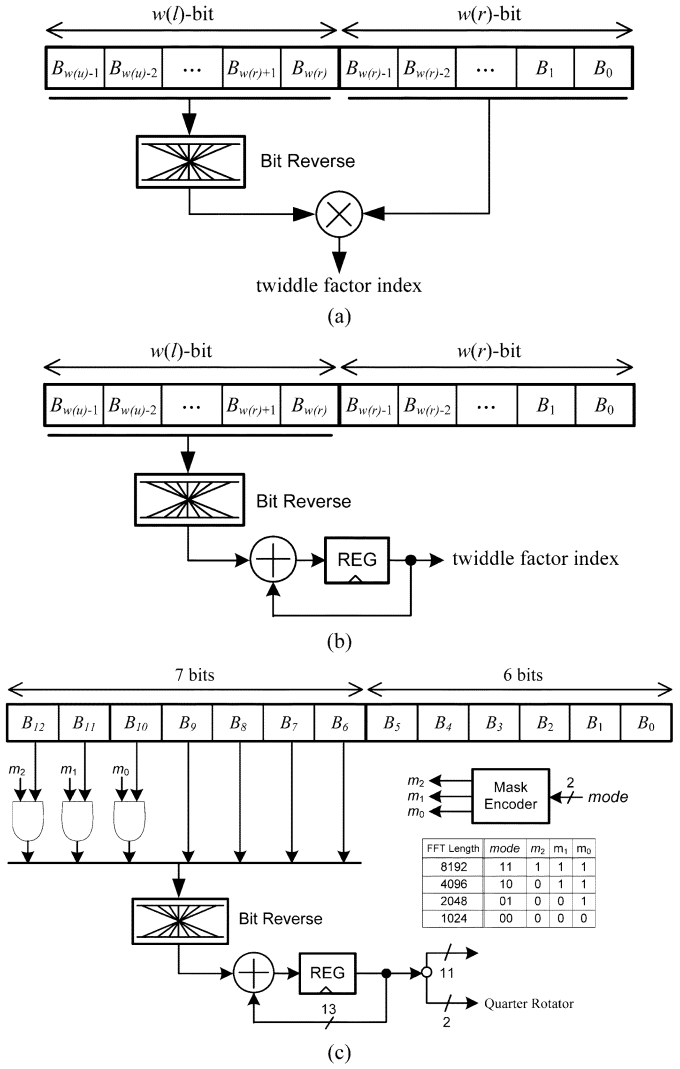


Fig. 11. Address generation for a twiddle factor table. (a) Direct approach, (b) incremental approach, and (c) an example for configurable FFT lengths.

where N is $2^{w(u)}$. As shown in Fig. 1, the index of a twiddle factor is determined by multiplying the output sequence of the inner DFT by the input sequence of the outer DFT. As the output sequence of a DFT is the bit-reversed input sequence of the DFT, multiplying the bit-reversed input sequence of the inner DFTs by the input sequence of the outer DFT can generate the index.

The above observation leads to the circuit diagram shown in Fig. 11(a), where a $w(u)$-bit counter is employed to generate input sequences of inner and outer DFTs. The upper $w(l)$ bits and the lower $w(r)$ bit correspond to input sequences of the inner DFT and the outer DFTs, respectively. The table index can be computed by multiplying the value of the bit-reversed $w(l)$ bits by the value of $w(r)$ bits. An incremental approach is shown in Fig. 11(b), which is more efficient as the multiplier is replaced with an adder. Since the value of $w(r)$ bits increases by one, the index can be generated by accumulating the value of the bit-reversed $w(l)$ bits.

It is possible to support variable-point transforms with a FFT processor designed for the longest-point FFT. For example, we can support 4192-point FFT with an 8192-point FFT processor by simply reducing the weights of the root node by one and recursively reducing by one all the left child nodes whose parent weights are decreased. Reducing the weight of a node can be achieved by fixing the most significant bit (MSB) of the counter to zero. Fig. 12 shows the binary trees of 4096/2048/1024-point FFTs that can be re-constructed from the 8196-point FFT. Note that the binary tree corresponding to 1024-point FFT is not balanced. Although some trees derived from the longest-point FFT can be imbalanced, the imbalance does not lead to any additional hardware overhead, as the hardware cost is determined by the longest-point FFT.

Fig. 11(c) shows an address generator designed for the 7th stage of the proposed 8192-point FFT processor. To support the 4096/2048/1024-point FFTs, the upper 3 bits of the counter can be masked to zero according to the mode signal. The output length of the adder is 13 bits, and only the lower 11 bits are actually used as the table index. The upper 2 bits are used to control the quarter rotator that calculates the $\pi/2$ symmetry.

B. Pre-Encoding of Twiddle Factors

Twiddle factor multiplication is the most complex computation in FFT processing. To provide high throughput, we exploit the modified Booth multiplication that generates a partial product per two multiplier bits. As summarized in Table II, three consecutive multiplier bits are examined in the modified Booth multiplication to generate a partial product. In the conventional modified Booth multiplication, the sine and cosine values stored in the twiddle factor table are first read and then encoded to generate the partial products as shown in the third column of Table II. We can reduce the Booth encoder and encoding time by directly storing the Booth encoded values into the table, instead of the native sine and cosine values. The scheme is more efficient in terms of area and latency, because the Booth encoder can be completely avoided in the multiplier implementation. However, in the conventional encoding, a Booth code requires 3 bits to represent five operations, $\{-2A, -A, 0, +A, +2A\}$ as shown in the third column of Table II, meaning that the number of bits to be stored into the table increases by about 50% compared to that of the native sine and cosine values.

To store encoded values into the table without increasing the word size, we propose a new encoding. One observation is that the current encoding is closely related to the next adjacent encoding because of the bit overlapped between two sets of multiplier bits. The MSB of the current set is the least significant bit of the next set. If the MSB of the current set is 0, the operation of the next set is in $\{0, A, -2A, -A\}$. Otherwise, the next bit-set is restricted to four operations, $\{+2A, +A, 0, -A\}$. According to the sign of the current encoding, therefore, the next encoding is restricted to four operations that can be represented

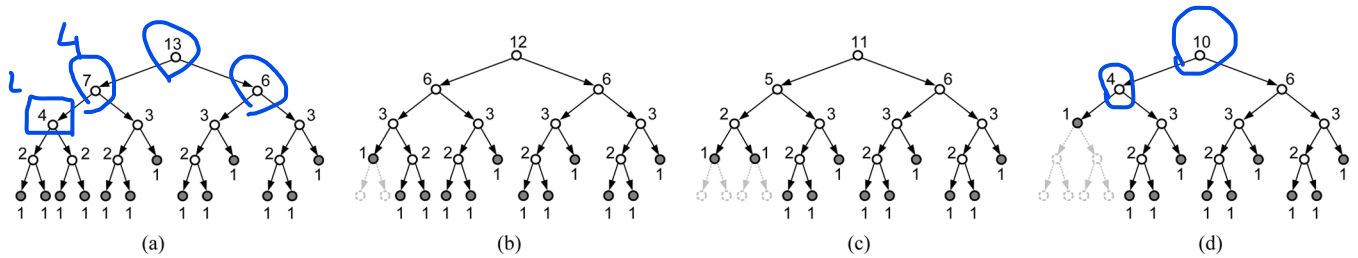


Fig. 12. Binary trees reconstructed from the 8192-point FFT. (a) 8192-point FFT, (b) 4096-point FFT, (c) 2048-point FFT, and (d) 1024-point FFT.

TABLE II
AREA-EFFICIENT ENCODING FOR MODIFIED BOOTH MULTIPLICATION

Multiplier Bits			Operation	Modified Booth Encoding (B_i)			Proposed Encoding	
b_{2i+1}	b_{2i}	b_{2i-1}		sign	$2A$	A	p_{2i+1}	p_{2i}
0	0	0	+0	0	0	0	0	0
0	1	0	$+A$	0	0	1	0	1
1	0	0	$-2A$	1	1	0	1	0
1	1	0	$-A$	1	0	1	1	1
0	0	1	$+A$	0	0	1	1	1
0	1	1	$+2A$	0	1	0	1	0
1	0	1	$-A$	1	0	1	0	1
1	1	1	-0	1	0	0	0	0

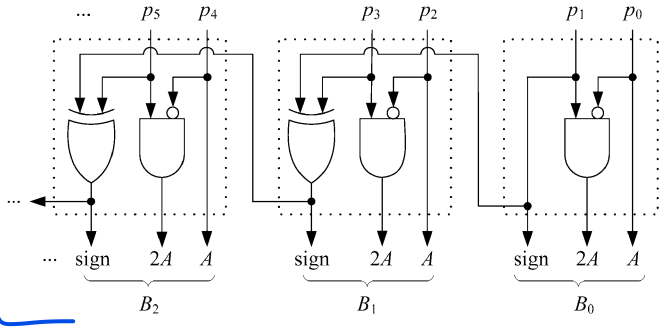


Fig. 13. Logic circuit for converting the proposed encoding to the modified Booth encoding.

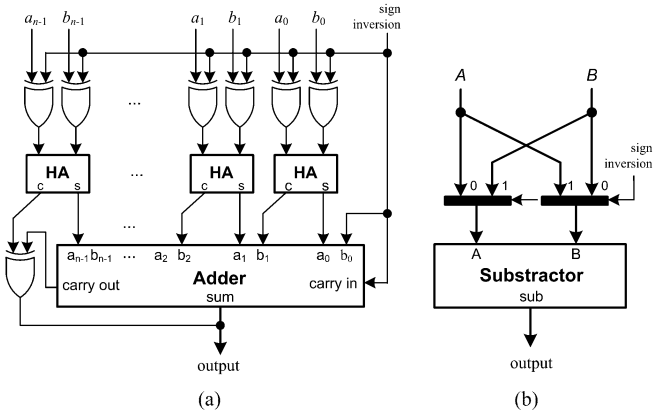


Fig. 14. Adder and subtractor combined with two's complementation. (a) Combined adder, and (b) combined subtractor.

with only 2 bits. The proposed 2-bit encoding is shown in the forth column of Table II.

The original modified Booth encoding can be produced from the proposed 2-bit encoding by using the simple logic shown in Fig. 13. The only drawback is that the new encoding should be serially computed in producing the original encoding. As the serial computation can increase the delay of multiplication, it can increase the delay of a pipeline stage, resulting in performance degradation. In a pipeline stage, however, the serial computation can be parallelized with the BF operation, because the twiddle factor multiplication is performed after the BF operation is completed. As a result, the new encoding can reduce the complexity of booth encoding logic without sacrificing performance.

C. Combined Quarter Rotator

The $\pi/2$ symmetric property of twiddle factors, which is effective in reducing the ROM size, is implemented by employing a *quarter rotator* that rotates the result of twiddle factor multiplication by W_4^0, W_4^1, W_4^2, or W_4^3. The quarter rotator defined in Table I can be achieved by the two's complementation and the swapping of real and imaginary values. The two MSBs of the index generator are used to control the quarter rotator, and the other bits to read the twiddle factor table. While the swapping can be implemented using simple multiplexers, the two's complementation requires an adder to perform subtraction. Since the two's complementation follows the final addition/subtraction of the complex multiplier, they can be combined to reduce area and enhance performance, as shown in Fig. 14. The combined circuits are derived from $-(A+B) = (\sim A) + (\sim B) + 2$ and $-(A-B) = B - A$, where $\sim x$ represents the bit inversion of x.

D. Area-Efficient FIFO Memory

The FIFO memory used in the delay feedback can be implemented by serially connecting registers. Since this structure shifts the contents every cycle, it induces large area and power consumption for a deep FIFO memory. The function of a FIFO memory can be emulated by using a RAM memory associated with the tail and head pointers. In this case, the memory contents can be accessed using the pointers instead of shifting the contents. To reduce area and power, therefore, the RAM memory has to be employed in implementing a deep FIFO memory. Since the FIFO memory reads and writes data simultaneously in a cycle, a straightforward implementation requires a RAM memory equipped with two ports.

As the two-port memory is more expensive than the single-port memory, we describe in this subsection the FIFO memory that can be constructed with single-port memories. We assume that the single-port memory provides separate data ports for read and write accesses, but only one of them can be accessed in a cycle due to the single address port, which is usually valid in most embedded memories. The first method is to use two single-port memories each of which has half entries. By alternating the read and write operations, we can emulate the function of the FIFO memory. The next method is to employ a single memory

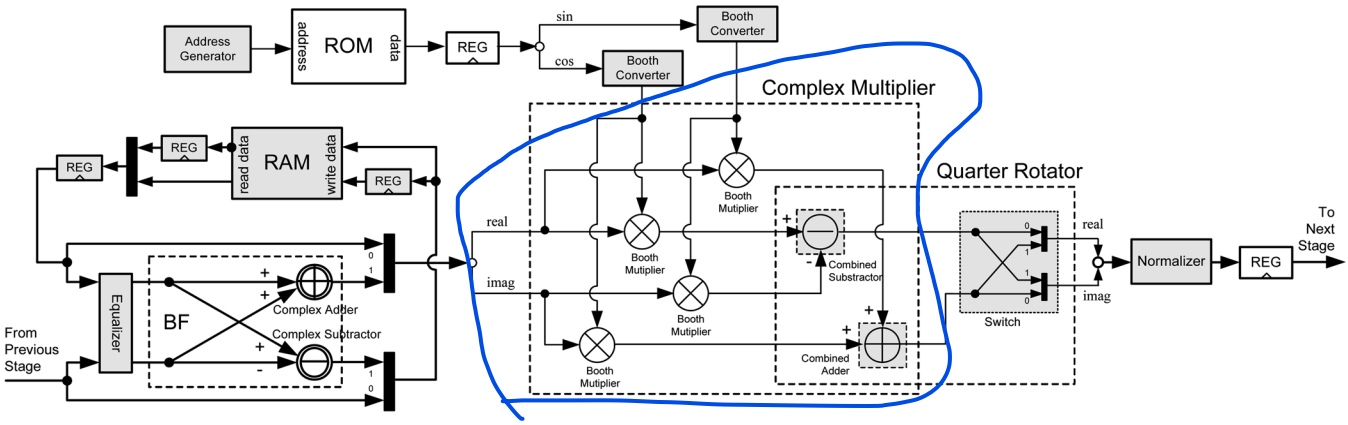


Fig. 15. Detailed structure of a pipeline stage based on the proposed implementation techniques.

TABLE III
COMPARISON OF TABLE SIZE AND MULTIPLIERS

FFT length	Radix-2 algorithm				Radix-2^2 algorithm				Radix-2^3, 4+2 Algorithm				Proposed Algorithm			
	TS[a]	TS(%)	GM[b]	CM[c]	TS	TS(%)	GM	CM	TS	TS(%)	GM	CM	TS	TS(%)	GM	CM
1024	508	149	28	2	340	100	16	10	292	86	12	6	272	80	12	4
2048	1020	150	32	2	680	100	20	10	584	86	12	6	536	79	12	6
4096	2044	150	36	2	1364	100	20	12	1170	86	12	8	1056	77	12	6
8192	4092	150	40	2	2728	100	24	12	2340	86	16	8	2100	77	16	6

[a] Table Size [b] General Multipliers [c] Constant Multipliers

in order to reduce the area. Since only a read or write operation is allowed in a cycle, the read and write operation is alternated every cycle. The word size of the memory is doubled in this case in order not to lose the performance by allowing the read and write of two entries at a time. A delay register is inserted in front of the write port to parallelize two serial data coming from the BF operator, and a delay register and a multiplexer are added to serialize two data read from the RAM memory. The second method needing only one memory is usually more efficient than the first one, because each memory includes peripheral logics such as row and column decoders.

Fig. 15 shows the detailed structure of a pipeline stage obtained by applying the above implementation techniques. As data are represented in the semi-floating point format [22], an equalizer is employed to align two input values of each BF and a normalizer to generate normalized output values. As the ROM is usually slower than logic circuits, the sine and cosine values read from the twiddle factor ROM are latched into registers in order to enhance the performance by reducing the critical path delay.

V. EXPERIMENTAL RESULTS

Table III compares the hardware costs of the proposed and previous FFT algorithms, which indicates that the proposed decomposition algorithm provides a structure that is more efficient than those of previous FFT algorithms. The proposed algorithm reduces the total size of twiddle factor ROMs as well as the number of multipliers. In case of 8192-point FFT, the proposed algorithm reduces 33% general multipliers, 50% constant multipliers and 23% table size compared to the radix-2^2 algorithm being widely used in hardware implementation. In general, the more the length increases, the more the proposed FFT algorithm reduces the table size.

By using the proposed balanced tree decomposition, an 8192-point FFT processor is designed based on the single-path delay feedback (SDF) structure [8]. As shown in Fig. 16, the overall structure is designed to support 8192/4096/2048/1024-point FFTs. The wordlength of the processor is determined by performing SQNR (Signal to Quantization Noise Ratio) simulation. For the semi-floating point representation, the wordlength is set to 12 bits to achieve a SQNR of 55 dB.

The size of a twiddle factor table is decided by the cost of the corresponding node defined in (2). For a node of weight 2, the twiddle factor is either $W_4^0 = 1$ or $W_4^1 = -j$, which can be computed by employing only the quarter rotator. For the node of weight 3, a constant complex multiplier is additionally required for the multiplication of W_8^1. If the node weight is more than 3, a general multiplier associated with a ROM table is used. The proposed 8192-point FFT processor requires a ROM table of 2048 entries, and three small tables of 32, 16, and 4 entries.

The proposed 8192-point FFT processor is described in a hardware description language, and synthesized with a 0.18-μm CMOS standard cell library. Fig. 17 shows the layout. The maximum operating frequency is 112 MHz, and the gate count is 43 392 excluding memories. The execution time to compute 8192-point FFT is 410.2 μs at 20 MHz, which is enough to meet the specification of DVB-T standard (896 μs). From the 7th stage, the FIFO memory is implemented using registers, because a small-sized memory can be implemented more efficiently with registers. For the same reason, the ROM tables for the second, forth, and tenth stages are implemented with combinational logics. Therefore, only a single ROM table exists for the 7th stage in the layout.

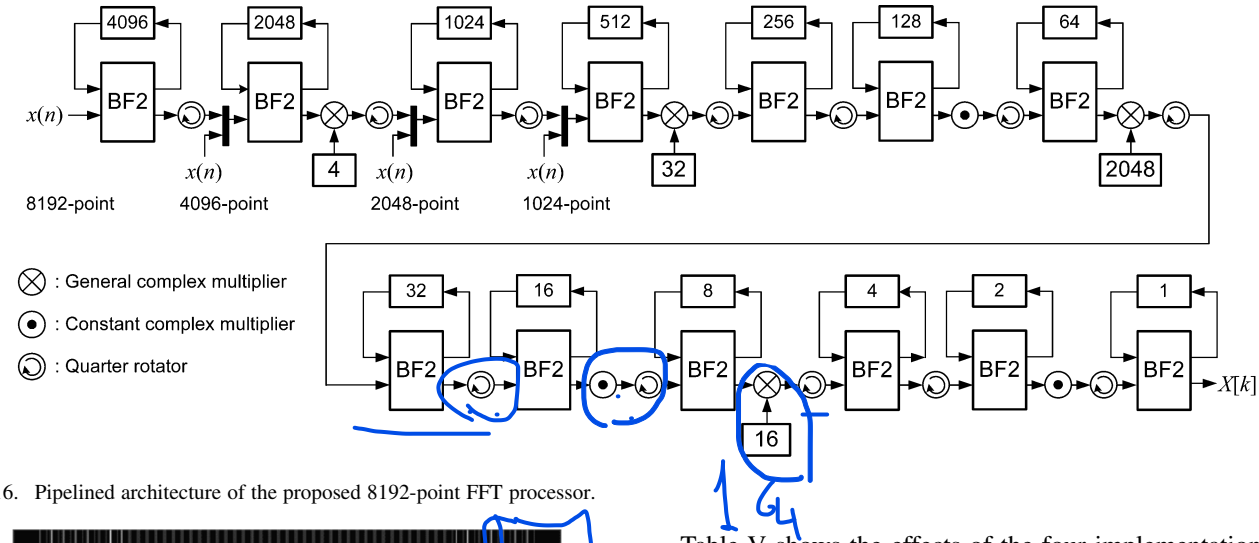


Fig. 16. Pipelined architecture of the proposed 8192-point FFT processor.

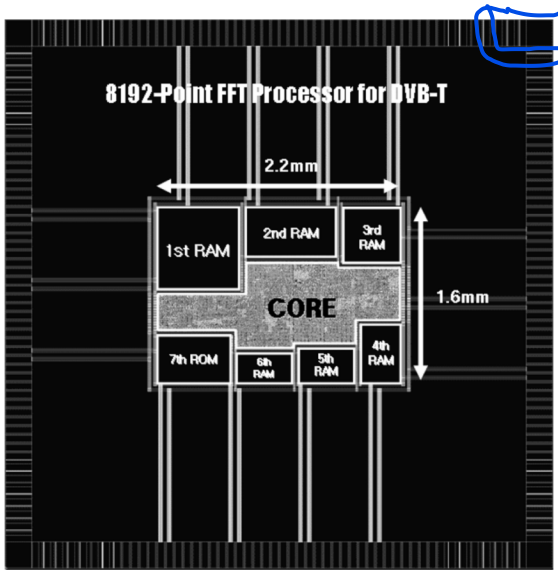


Fig. 17. Layout of the proposed 8192-point pipelined FFT processor.

To compare fairly with the previous works, core area and power consumption are normalized as follows [21], [23]:

$$\text{Normalized Area} = \frac{\text{Area}}{(\text{Technology}/0.18\ \mu\text{m})^2} \tag{5}$$

$$\frac{\text{FFTs}}{\text{Energy}} = \frac{\text{Technology}/0.18\ \mu\text{m}}{\text{Power} \times \text{ExecutionTime} \times 10^3} \tag{6}$$

where (6) means the normalized number of 8192-point FFTs that can be computed with 1 mW.

In Table IV, the performance characteristics of the proposed processor are summarized and compared with the previous works. The power consumption and operating frequency are obtained from the post-layout simulation including parasitic capacitance and resistance. Due to the architectural difference, the power consumption of the proposed processor is higher than [21]. However, the normalized FFTs per energy of (6) indicates slightly better energy efficiency, since the execution time of the proposed processor is shorter than [21]. As a result, the proposed processor achieves better area and power efficiency compared to the other FFT processors. Moreover, the proposed processor results in much higher operating frequency.

Table V shows the effects of the four implementation techniques described in Section IV. All the techniques are more efficient than the conventional implementations, but their contributions to the overall area reduction, except the FIFO implementation, are not remarkable because the area is dominated by multipliers, butterflies and memories. The area reduction is mainly resulted from the proposed balanced decomposition that is effective in reducing the sizes of twiddle factor tables and the number of multipliers.

VI. Conclusion

In this paper, we have presented a new decomposition algorithm to reduce the size of twiddle factor tables required in pipelined FFT processing. The proposed algorithm called balanced decomposition is to recursively decompose a DFT into sub-DFTs such that the transform lengths of the resulting sub-DFTs are almost equal. We have proved that the recursive decomposition leads to a FFT algorithm minimizing the total size of twiddle factor tables. From the viewpoint of the binary tree representation introduced to represent the decomposition procedure graphically, the proposed algorithm results in a balanced binary tree, as the recursive decomposition progresses in both left and right directions, while the previous FFT algorithms lead to imbalanced binary trees due to their unidirectional decomposition procedures. In addition to the table size, the balanced decomposition is also effective in reducing hardware resources such as general multipliers and constant multipliers. An 8192-point pipelined FFT processor designed for DVB-T applications saves 33% of general multipliers, 50% of constant multipliers and 23% of the total size of twiddle factor tables, compared to a pipelined radix-2^2 FFT processor. To reduce area further without compromising performance, we have presented a number of implementation schemes, such as incremental index generation, pre-encoding of twiddle factors to be stored into tables, RAM-based FIFO, and adder/subtractor combined with sign inversion.

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] J. Ja'Ja' and R. M. Owens, "An architecture for a VLSI FFT processor," *VLSI J. Integr.* , vol. 1, no. 4, pp. 305–316, 1983.

TABLE IV
PERFORMANCE OF THE PROPOSED FFT PROCESSOR COMPARED WITH PREVIOUS IMPLEMENTATIONS

	S. He [11]	E. Bidet [19]	L. Jia [20]	Y. Lin [21]	Proposed
FFT sizes (≥ 1024)	1024	8192, 2048	8192, 1024	8192, 1024	8192, 4096, 2048, 1024
Technology	0.5 μm	0.5 μm	0.6 μm	0.18 μm	0.18 μm
Supply voltage	N.A.	3.3 V	3.3 V	1.8 V	1.8 V
Max. frequency	30 MHz	23 MHz	N.A.	56 MHz	112 MHz
FFT algorithm	Radix-2^2	Radix-4	Split radix-2/4/8	Radix-2^3	Balanced binary-tree
Processor architecture	Pipelined SDF[a]	Pipelined SDC[b]	Pipelined SDF	Single (Radix-2^3)	Pipelined SDF
Max. Throughput @ *R* Hz	R	R	R	0.57R @ 8192-point	R
Datapath width	2 × 16 bit	2 × 12 bit	2 × 12 bit	2 × 11 bit	2 × 12 bit
Data Representation	Fixed point	CBFP[c]	N.A.	Dynamic scaling	Semi-floating point
Logic count	21K standard cells (without memories)	1.5M transistors (with memories)	1.3M transistors (with memories)	N.A.	43,392 gates (without memories)
Core area	40 mm^2	100 mm^2	107 mm^2	4.84 mm^2	3.52 mm^2
Normalized core area	5.18	12.96	9.63	4.84	3.52
Execution time @ 20MHz, 8K-point	N.A.	400 μs	409.6 μs	717.35 μs	410.2 μs
Power consumption @ 20MHz	N.A.	600 mW (chip)	650 mW (chip)	25.2 mW (core)	40.7 mW (core)
Normalized FFTs/Energy	N.A	11.57	12.54	55.32	59.90
Measurement	Post-Layout Simulation	Chip	Simulation	Chip	Post-Layout Simulation

[a] Single-Path Delay Feedback [b] Single-Path Delay Commutator [c] Convergent Block Floating Point

TABLE V
GATE COUNT OR AREA COMPARISON OF FOUR TECHNIQUES

Proposed Techniques	Conventional Implementation	Proposed Implementation	Reduction Rate
Twiddle Factor Address Generation	518 gates	270 gates	48%
Booth Pre-Encoding	180 gates	92 gates	49%
Combined Quarter Rotator	2,400 gates	1,824 gates	24%
Single-Port RAM-Based FIFO	1.78 mm^2	1.59 mm^2	12%

[3] H. S. Lee, "Systolic array architecture for VLSI FFT processor," *Int. J. Mini Microcomput.*, vol. 6, no. 3, pp. 49–54, 1984.

[4] V. Boriakoff, "FFT computation with systolic arrays, a new architecture," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 41, no. 4, pp. 278–284, Apr. 1994.

[5] 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, Sep. 1991.

[6] H. Stone, "Parallel processing with the perfect shuffle," *IEEE Trans. Comput.*, vol. C-20, no. 2, pp. 153–161, Feb. 1971.

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

[8] 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.

[9] A. M. Despain, "Fourier transform computer using CORDIC iterations," *IEEE Trans. Comput.*, vol. C-23, no. 10, pp. 993–1001, Oct. 1974.

[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. SC-19, no. 5, pp. 702–709, Oct. 1984.

[11] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC'98)*, May 1998, pp. 131–134.

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

[13] 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.

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

[15] D. Takahashi, "An extended split-radix FFT algorithm," *Electron. Lett.*, vol. 8, no. 5, pp. 145–147, May 2001.

[16] H. R. Wu and F. J. Paoloni, "The Structure of vector radix fast Fourier transforms," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 37, no. 9, pp. 1415–1424, Sep. 1989.

[17] I. J. Good, "The relationship between two fast Fourier transforms," *IEEE Trans. Comput.*, vol. C-20, no. 3, pp. 333–341, May 1979.

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

[19] E. Bidet, D. Castelain, C. Joanblanq, and P. Senn, "A fast single-chip implementation of 8192 complex point FFT," *IEEE J. Solid-State Circuits*, vol. 30, no. 3, pp. 300–305, Mar. 1995.

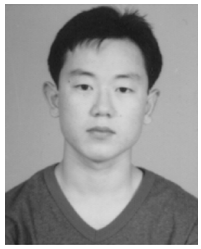
[20] L. Jia, Y. Gao, J. Isoaho, and H. Tenhunen, "A new VLSI-oriented FFT algorithm and implement," in *Proc. 11th Annu. IEEE Int. ASIC Conf.*, Sep. 1998, pp. 337–341.

[21] Y. Lin, H. Liu, and C. Lee, "A dynamic scaling FFT processor for DVB-T applications," *IEEE J. Solid-State Circuits*, vol. 39, no. 11, pp. 2005–2013, Nov. 2004.

[22] T. Lenart and V. Owall, "A 2048 complex point FFT processor using a novel data scaling approach," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS'03)*, May 2003, pp. IV.45–IV.48.

[23] B. M. Bass, "A low-power, high-performance, 1024-point FFT processor,," *IEEE J. Solid-State Circuits*, vol. 34, no. 3, pp. 380–387, Mar. 1999.

[24] *Digital Video Broadcasting (DVB); Framing Structure, Channel Coding and Modulation for Digital Terrestrial Television*, ETSI EN 300 744, 2004, ETSI, v.1.5.1.



Hyun-Yong Lee (S'01) received the B. Tech. and M.S. degree in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 2001 and 2003, respectively. Currently, he is working toward the Ph.D. degree in the Department of Electrical Engineering and Computer Science at KAIST.

His research interests include VLSI design for signal processing and communication.

In-Cheol Park (S'88–M'92–SM'02) received the B.S. degree in electronic engineering from Seoul National University, Seoul, Korea, in 1986, the M.S. and Ph.D. degrees in electrical engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejon, Korea, in 1988 and 1992, respectively.

Since June 1996, he has been an Assistant Professor and is 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 on high-speed circuit design. His current research interest includes computer-aided design (CAD) algorithms for high-level synthesis, and VLSI architectures for general-purpose microprocessors.

Dr. Park received the Best Paper Award at ICCD in 1999, and the Best Design Award at ASP-DAC in 1997.