TR-93-005

An Efficient Prime-Factor Algorithm for the Discrete Cosine Transform and Its Hardware Implementations*

PeiZong Lee† and Fang-Yu Huang

An Efficient Prime-Factor Algorithm for the Discrete Cosine Transform and Its Hardware Implementations*

PeiZong Lee† and Fang-Yu Huang
Institute of Information Science
Academia Sinica
#128, Sec. 2, Yen-Chiu-Yun Road
Nankang, Taipei, Taiwan, R. O. C.

June 4, 1993

Abstract

The prime-factor decomposition is a fast computational technique for many important digital signal processing operations, such as the convolution, the discrete Fourier transform, the discrete Hartley transform, and the discrete cosine transform (DCT). In this paper, we present a new prime-factor algorithm for the DCT. We also design a prime-factor algorithm for the discrete sine transform which is based on the prime-factor DCT algorithm.

Hardware implementations for the prime-factor DCT are also studied. We are especially interested in the hardware designs which are suitable for the VLSI implementations. We will show three hardware designs for the prime-factor DCT, including a VLSI circuit fabricated directly according to the signal-flow graph, a linear systolic array, and a mesh-connected systolic array. These three designs show the trade-off between cost and performance. The methodology, which deals with general $(N_1 \cdot N_2)$-point DCTs, where N_1 and N_2 are mutually prime, is illustrated by converting a 15-point DCT problem into a (3×5)-point 2-D DCT problem.

*** A preliminary version of this technical report is presented at the **IEEE International Conference on Acoustics, Speech, and Signal Processing**, Minneapolis, Minnesota, U.S.A., April 27–30, 1993.

*This work was partially supported by the NSC under Grants NSC 81-0408-E-001-505.
†PeiZong Lee: leepe@iis.sinica.edu.tw, TEL: +886 (2) 788-3799, FAX: +886 (2) 782-4814.

1 Introduction

This paper is concerned with designing a fast prime-factor algorithm for the discrete cosine transform (DCT). The DCT, which performs much like the theoretically optimal Karhunen-Loeve transform for the first-order Markov stationary random data, has found wide applications in speech and image processing as well as telecommunication signal processing for the purpose of data compression, feature extraction, and filtering [9]. In order to compute the DCT efficiently, fast algorithms have been intensively studied [1] [8] [12] [13] [16] [19] [28] [30]. A complete survey of the DCT algorithms can be seen elsewhere [14] [21].

It is our goal in this paper to present a new prime-factor DCT algorithm. The prime-factor decomposition is a fast computational technique for many important digital signal processing operations, such as the convolution [20], the discrete Fourier transform (DFT) [2] [3] [4] [5] [10] [22] [24] [25] [26] [27], the discrete Hartley transform (DHT) [6] [18] [23], and the DCT [6] [13] [35]. It has both theoretical and practical significance. Its main theoretical rationale is to convert a large-size one-dimension problem, by employing certain appropriate index mappings, into a multidimensional one. Then, we can deal with the resulting groups of small-size problems in each dimension.

For practical considerations, since in a typical DSP processor the memory for data storage is expensive and usually not large, it is more feasible to process a small-size problem one at a time. In addition, when this approach is combined with efficient short-length algorithms, such as Rader's algorithm [20], or Winograd type minimum multiplication algorithms in DFT [31], or Heideman's small odd-length DCT modules [7], etc., it would be of practical interest in reducing the scalar multiplication complexity.

However, the actual realizations of the index mappings are always time-consuming. Therefore, many researchers have been seeking efficient implementation methods and have obtained some encouraging results [17] [32] [33] [34].

Although, previously Yang and Narasimha [35] have proposed a prime-factor DCT algorithm,

its index mapping is very complicated. Yang and Narasimha note that an N-point DCT can be implemented by an N-point DFT, and in addition, there exist prime-factor algorithms for the DFT. Therefore, it is possible to derive a prime-factor DCT algorithm.

From our point of view, an index mapping not only should be easy to understand, but also should be efficient in running. Lee [13] has achieved this goal. However, his input index mapping is realized by constructing and combining two index tables, which could occupy additional memory space and would be infeasible in variable-size applications.

Chakrabarti and JáJá [6] develop a systolic architecture for implementing Lee's algorithm. Because they want to compute the DCT from the DHT, they modify the index mappings, which are essentially the same as Lee's. However, they did not discuss the actual implementation for these index mappings.

In this paper, the input index mapping we adopt is the Ruritanian mapping, since its efficient realization can be based on the previous research efforts. In addition, the resulting algorithm complexity is by no means increased. As for the output index mapping, we employ the same one as Lee's, for which might be the most natural one in view of the DCT transform kernel's structure. Because the Ruritanian mapping can be implemented by existing fast algorithms, our algorithm is thus suitable for on-line computation for variable different size prime-factor DCTs.

We also consider hardware implementations for the prime-factor DCT. We are especially interested in the hardware designs which are suitable for the VLSI (Very Large Scale Integration) implementations. We will show three hardware designs for the prime-factor DCT, including a VLSI circuit fabricated directly according to the signal-flow graph, a linear systolic array, and a mesh-connected systolic array. These three designs show the trade-off between cost and performance.

Finally, we generalize Wang's method [29] to design an algorithm for computing the discrete sine transform (DST) from the DCT. We also design a prime-factor DST algorithm based on the prime-factor DCT algorithm.

The rest of this paper is organized as follows. In Section 2, the DCT is introduced. In Section 3, we derive the prime-factor DCT algorithm. In Section 4, we propose three hardware implementations. In Section 5, we derive the prime-factor DST based on the prime-factor DCT. In Section 6, we give some concluding remarks. Finally, in the Appendix, we give the proof of two main theorems.

2 Background

For a given input data sequence x_n, $0 \leq n \leq N-1$, the DCT output data sequence X_k, $0 \leq k \leq N-1$, is defined by

$$X_k = \sqrt{\frac{2}{N}}\, \epsilon(k) \sum_{n=0}^{N-1} x_n \cos(\frac{\pi(2n+1)k}{2N}) \,; \tag{1}$$

and the IDCT (inverse discrete cosine transform) is defined by

$$x_n = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} \epsilon(k)\, X_k \cos(\frac{\pi(2n+1)k}{2N}) \,; \tag{2}$$

where

$$\epsilon(k) = \begin{cases} \frac{1}{\sqrt{2}}\,, & \text{for } k = 0; \\ 1\,, & \text{for } 1 \leq k \leq N-1\,. \end{cases}$$

We assume throughout this paper that N is a product of two relatively prime integers N_1 and N_2. For convenience, we will ignore the scaling factor $\epsilon(k)$ and the normalization factor $\sqrt{\frac{2}{N}}$ in Equations (1) and (2), since they can be done in a separate step. In the following, we deal with the simplified version of Equation (2) :

$$x_n = \sum_{k=0}^{N-1} X_k \cos(\frac{\pi(2n+1)k}{2N})\,. \tag{3}$$

We want to convert Equation (3) into the form in Equation (4) by taking appropriate input and output index mappings:

$$x_{(n_1,n_2)} = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} Y_{(k_1,k_2)} \cos(\frac{\pi(2n_1+1)k_1}{2N_1}) \cos(\frac{\pi(2n_2+1)k_2}{2N_2})\,, \tag{4}$$

where $0 \leq n_1 < N_1$, $0 \leq n_2 < N_2$, and $Y_{(k_1,k_2)}$ are the result of certain modifications on $X_{(k_1,k_2)}$.

The input index mapping connects the input index k, $0 \leq k < N$, to (k_1, k_2), $0 \leq k_1 < N_1$ and $0 \leq k_2 < N_2$. The output index mapping connects the output index n, $0 \leq n < N$, to (n_1, n_2), $0 \leq n_1 < N_1$ and $0 \leq n_2 < N_2$. Since the DCT is orthonormal, the forward transform also can be realized by taking the transpose of the inverse transform.

3 Derivation of the Prime-Factor Decomposition

The input index mapping we adopt is the Ruritanian mapping, which was also used in the prime-factor DFT algorithms [3] [4] [5] [10] [22], and several researchers have studied its efficient implementation methods [17] [32] [33] [34]. That is,

$$\forall\, k \in [0,\, N-1],\ \exists\,!\,(k_1, k_2) \in [0,\, N_1 - 1] \times [0,\, N_2 - 1],$$

such that

$$k = \psi(k_1, k_2) = (N_2 k_1 + N_1 k_2) \bmod N\ ,$$

where $N = N_1\, N_2$, N_1 and N_2 are relatively prime.

In other words, based on the Ruritanian mapping, the integers k on the interval $[0,\, N-1]$ correspond one-to-one to the lattice points in the region $[0,\, N_1 - 1] \times [0,\, N_2 - 1]$. For the sake of our derivation, these lattice points are divided into five disjoint groups.

Let

$$\begin{aligned} f(k_1, k_2) &= N_2 k_1 + N_1 k_2\ , \\ g(k_1, k_2) &= N_2 k_1 - N_1 k_2\ . \end{aligned}$$

Define

$$\begin{aligned} E &= \{(k_1, k_2) \,|\, k_1 = 0 \text{ or } k_2 = 0\}\ , \\ A &= \{(k_1, k_2) \,|\, f(k_1, k_2) < N \text{ and } g(k_1, k_2) > 0;\ k_1 k_2 \neq 0\}\ , \\ B &= \{(k_1, k_2) \,|\, f(k_1, k_2) < N \text{ and } g(k_1, k_2) < 0;\ k_1 k_2 \neq 0\}\ , \end{aligned}$$

$$C = \{(k_1,k_2) \,|\, f(k_1,k_2) > N \text{ and } g(k_1,k_2) < 0;\ k_1k_2 \neq 0\},$$
$$D = \{(k_1,k_2) \,|\, f(k_1,k_2) > N \text{ and } g(k_1,k_2) > 0;\ k_1k_2 \neq 0\}.$$

Let

$$W = \{k \,|\, k = \psi(k_1,k_2),\ (k_1,k_2) \in E\},$$
$$U = \{k \,|\, k = f(k_1,k_2),\ (k_1,k_2) \in (A \cup B)\},$$
$$V = \{k \,|\, k = g(k_1,k_2),\ (k_1,k_2) \in (A \cup D)\}.$$

Theorem 1 $\{k \,|\, k \in [0,\ N-1],\ k \textit{ is an integer}\} = W \cup U \cup V.$

Proof : In the Appendix.

From Theorem 1, we can rewrite Equation (3) as follows:

$$x_n = (\sum_{k\in W} + \sum_{k\in U} + \sum_{k\in V}) X_k \cos(\frac{\pi(2n+1)k}{2N}). \tag{5}$$

Let

$$X_{(k_1,k_2)} = X_k, \text{ where } k = \psi(k_1,k_2),$$
$$X^f_{(k_1,k_2)} = X_k, \text{ where } k = f(k_1,k_2) \text{ and } (k_1,k_2) \in (A \cup B),$$
$$X^g_{(k_1,k_2)} = X_k, \text{ where } k = g(k_1,k_2) \text{ and } (k_1,k_2) \in (A \cup D).$$

Because, when (k_1,k_2) is in E, $f(k_1,k_2) = |g(k_1,k_2)| = \psi(k_1,k_2)$, we have $\cos(\frac{\pi(2n+1)f(k_1,k_2)}{2N}) = \cos(\frac{\pi(2n+1)g(k_1,k_2)}{2N}) = \cos(\frac{\pi(2n+1)\psi(k_1,k_2)}{2N})$. Therefore, Equation (5) becomes:

$$\begin{aligned} x_n = \frac{1}{2}\{ & \sum_{(k_1,k_2)\in E} X_{(k_1,k_2)} [\cos(\frac{\pi(2n+1)\,f(k_1,k_2)}{2N}) + \cos(\frac{\pi(2n+1)\,g(k_1,k_2)}{2N})] + \\ & 2 \sum_{(k_1,k_2)\in(A\cup B)} X^f_{(k_1,k_2)} \cos(\frac{\pi(2n+1)\,f(k_1,k_2)}{2N}) + \\ & 2 \sum_{(k_1,k_2)\in(A\cup D)} X^g_{(k_1,k_2)} \cos(\frac{\pi(2n+1)\,g(k_1,k_2)}{2N})\}. \end{aligned} \tag{6}$$

Now, we define

$$Y_{(k_1,k_2)} = \begin{cases} X_{(k_1,k_2)}, & \text{if } (k_1,k_2) \in E \\ X_{(k_1,k_2)} + X_{(k_1,N_2-k_2)}, & \text{if } (k_1,k_2) \in A \\ X_{(k_1,k_2)} + X_{(N_1-k_1,k_2)}, & \text{if } (k_1,k_2) \in B \\ -X_{(N_1-k_1,N_2-k_2)} + X_{(N_1-k_1,k_2)}, & \text{if } (k_1,k_2) \in C \\ -X_{(N_1-k_1,N_2-k_2)} + X_{(k_1,N_2-k_2)}, & \text{if } (k_1,k_2) \in D\,. \end{cases} \tag{7}$$

Theorem 2

$$x_n = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} Y_{(k_1,k_2)} \cos(\frac{\pi(2n+1)k_1}{2N_1}) \cos(\frac{\pi(2n+1)k_2}{2N_2})\,.$$

Proof : In the Appendix.

Next, we introduce the output index mapping, which was proposed by Lee [13]. For the completeness of this presentation, we list it here for easy reference.

The output index mapping: $\varphi(n) = (n_1, n_2)$, where

$$\begin{aligned} \bar{n}_1 &= n \bmod 2N_1 \\ \bar{n}_2 &= n \bmod 2N_2 \\ n_1 &= \begin{cases} \bar{n}_1, & \text{if } \bar{n}_1 < N_1 \\ 2N_1 - 1 - \bar{n}_1, & \text{otherwise} \end{cases} \\ n_2 &= \begin{cases} \bar{n}_2, & \text{if } \bar{n}_2 < N_2 \\ 2N_2 - 1 - \bar{n}_2, & \text{otherwise.} \end{cases} \end{aligned} \tag{8}$$

Lee [13] has shown that $\varphi : [0,\, N-1] \longrightarrow [0,\, N_1-1] \times [0,\, N_2-1]$ is a one-to-one mapping function. In addition,

$$\cos(\frac{\pi(2n+1)k_1}{2N_1}) = \cos(\frac{\pi(2n_1+1)k_1}{2N_1}) \quad \text{and} \quad \cos(\frac{\pi(2n+1)k_2}{2N_2}) = \cos(\frac{\pi(2n_2+1)k_2}{2N_2}).$$

Therefore, it is easy to obtain the following theorem.

Theorem 3

$$x_{(n_1,n_2)} = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} Y_{(k_1,k_2)} \cos(\frac{\pi(2n_1+1)k_1}{2N_1}) \cos(\frac{\pi(2n_2+1)k_2}{2N_2})\,. \qquad \square$$

Prime-factor algorithm for the IDCT:

Input: An N-point data sequence X_k, $0 \leq k < N$, where $N = N_1 N_2$, N_1 and N_2 are mutually prime.

Output: An N-point data sequence x_n, $0 \leq n < N$.

Step 1: Apply the Ruritanian mapping on X_k to construct an $(N_1 \times N_2)$-point 2-D data matrix $X_{(k_1,k_2)}$, where $0 \leq k_1 < N_1$ and $0 \leq k_2 < N_2$.

Step 2: Modify the 2-D data matrix $X_{(k_1,k_2)}$ to get the 2-D data matrix $Y_{(k_1,k_2)}$ according to Equation (7).

/* This is proved to be correct in Theorem 2. */

Step 3: /* Row-column evaluation */

1. Execute the N_1-point IDCT for each of the N_2 columns of $Y_{(k_1,k_2)}$, and the result is $T_{(k_1,k_2)}$.
2. Execute the N_2-point IDCT for each of the N_1 rows of $T_{(k_1,k_2)}$, and the result is $x_{(n_1,n_2)}$, where $0 \leq n_1 < N_1$ and $0 \leq n_2 < N_2$.

/* This is proved to be correct in Theorem 3. */

Step 4: Apply the output index mapping in Equation (8) to get the final result x_n. □

In the following, we use an example to demonstrate our method.

Example : In this example, $N = 15$, and $N_1 = 3$, $N_2 = 5$. Figure 1 shows the signal flow graph for implementing the 15-point 1-D IDCT, which can be converted into the (3×5)-point 2-D IDCT. First, when given a 15-point 1-D input data sequence X_k, $0 \leq k < 15$, we transform this 1-D data sequence by the Ruritanian mapping to a (3×5)-point 2-D data matrix $X_{(k_1,k_2)}$, $0 \leq k_1 < 3$ and $0 \leq k_2 < 5$. Table 1 shows the Ruritanian mapping for this case. The five disjoint sets of lattice

k_2 / k / k_1	0	1	2	3	4
0	0	3	6	9	12
1	5	8	11	14	2
2	10	13	1	4	7

Table 1: The Ruritanian mapping: $k = (5k_1 + 3k_2) \bmod 15$, when $N = 15$, and $N_1 = 3$, $N_2 = 5$.

points in the region $[0, 2] \times [0, 4]$, which are divided by $k_1 k_2 = 0$, $5k_1 + 3k_2 > 15$, $5k_1 + 3k_2 < 15$, $5k_1 - 3k_2 > 0$, and $5k_1 - 3k_2 < 0$, include

$$\begin{aligned} E &= \{(0,0),(0,1),(0,2),(0,3),(0,4),(1,0),(2,0)\}; \\ A &= \{(1,1),(2,1)\}; \\ B &= \{(1,2),(1,3)\}; \\ C &= \{(1,4),(2,4)\}; \\ D &= \{(2,2),(2,3)\}. \end{aligned}$$

Next, we modify the 2-D data matrix $X_{(k_1,k_2)}$ to get the 2-D data matrix $Y_{(k_1,k_2)}$ according to Equation (7). Then, we apply the row-column evaluation for the computation: first, we deal with five 3-point IDCTs; then, take the transpose of the result; and then, deal with three 5-point IDCTs. Finally, the output data sequence $x(n)$, $0 \le n < 15$, can be obtained by using the output index mapping in Equation (8). Table 2 shows the output index mapping for this case.

Note that, the signal flow graph in Figure 1 performs the IDCT if the signals flow from left to right, and performs the DCT if the signals flow from right to left.

n_2 / n / n_1	0	1	2	3	4
0	0	11	12	6	5
1	10	1	7	13	4
2	9	8	2	3	14

Table 2: The output index mapping when $N = 15$, and $N_1 = 3$, $N_2 = 5$.

4 Hardware Implementations

In this section, we consider several hardware implementations for the prime-factor IDCT. We are especially interested in the hardware designs which are suitable for the VLSI implementations.

Fig. 1 shows the signal-flow graph for the implementation of a 15-point prime-factor IDCT. It is possible to fabricate such a 15-point prime-factor IDCT circuit directly into a single chip, which may contain millions of transistors, by using current VLSI technologies. The input data sequences can then be pipelined and enter into this chip, and an output data sequence is produced in every cycle time. This design, of course, has achieved the best performance and the maximum throughput. We will call this implementation, which is fabricated directly according to the signal-flow graph, *Design I.*

However, when the problem size grows, it is difficult to fabricate a large-size prime-factor IDCT circuit directly into a single chip. This is because it requires a network including several butterfly-like interconnections and a transpose network that are very costly to layout in most circuit technologies including VLSI.

After carefully studying the prime-factor algorithm, we find that Step 1 and Step 4 only deal with $O(N)$ memory references. Step 2 only requires $O(N)$ addition or subtraction operations. However, Step 3, which implements the row-column evaluation, requires at least $O(N \log N)$ multiplication and addition operations by any fast IDCT algorithm [8] [12] [14]. It is clear that Step 3 is the bottleneck of the algorithm. Steps 1, 2, and 4 can be implemented by the host computer, or

by a single processing element, or by a single processing element each. Therefore, in the following we concentrate our effort in solving Step 3 by using feasible VLSI algorithms.

In Fig. 1, we find that there are five 3-point IDCT components and three 5-point IDCT components. In general, there are N_2 N_1-point IDCT components and N_1 N_2-point IDCT components. In practice, in order to save hardware without slowing down the processing speed, we can use only one 3-point (N_1-point) IDCT component and one 5-point (N_2-point) IDCT component as shown in Fig. 2, provided the data sequences are carefully arranged.

In Fig. 2, Steps 1, 2, and 4 are each implemented by a single processing element. Step 3 requires a 3-point (N_1-point) IDCT component, a transpose buffer, and a 5-point (N_2-point) IDCT component. The data matrix generated after Step 2 is pipelined and enters into the 3-point (N_1-point) IDCT component column by column. The output data matrix of the 3-point (N_1-point) IDCT component is stored in a transpose buffer. After this data matrix enters the transpose buffer, this data matrix in the transpose buffer is then pipelined and enters into the 5-point (N_2-point) IDCT component row by row. The resulting data matrix is then sent to a processing element for generating the output data sequence by using the output index mapping.

We now consider the fabrication of an arbitrary N-point IDCT component. It is true that there is no fast algorithm for the N-point IDCT, except if N has a value of 2 to a power. However, in practice, the value of N may be varied. Previously, if N did not have a value of 2 to a power, then either we should pad additional zeros so that we could use a fast IDCT algorithm, or we should use the conventional matrix-vector multiplication algorithm to solve the N-point IDCT.

Suppose that we base this on an $O(N \log N)$ fast sequential IDCT algorithm for laying out the IDCT chip. If $N_1 \neq N_2$, then the layout of the N_1-point IDCT component is quite different from the layout of the N_2-point IDCT component. In addition, as mentioned before, because this fast sequential algorithm requires a network with several butterfly-like interconnections, the circuit layout becomes expensive. Therefore, if a chip includes both the N_1-point IDCT component and the N_2-point IDCT component, then the circuit design cycle will certainly be lengthened.

Recently, Kung and Leiserson [11] used matrix-vector multiplication algorithm to solve the discrete Fourier transform (DFT). Although it requires $O(N^2)$ multiply-and-add operations for solving the N-point DFT in the sequential algorithm, the algorithm can be implemented by a linear systolic array with N processing elements in $O(N)$ time. This result is promising in comparison with any sequential fast Fourier transform algorithm with $O(N \log N)$ operation complexity.

We are interested in the systolic array implementations, because the systolic arrays are especially suitable for VLSI implementations. A systolic array is a special-purpose parallel device, composed of a few simple types of process elements. Because its interconnection pattern exhibits regularity and locality, it is especially suitable for VLSI implementations. Several array structures have been proposed, including linear array, mesh-connected arrays, and hexagonal arrays. In a typical application, such arrays would be attached as peripheral devices to a host computer, which inserts input values into them and extracts output values from them [11].

Like the DFT, the IDCT also can be implemented by the conventional matrix-vector multiplication. In addition, we also can use a linear systolic array with N processing elements to implement the N-point IDCT in $O(N)$ time. We will call this linear systolic array implementation *Design II.*

However, as each processing element requires N registers or memory storage to store part of the N-point IDCT coefficient matrix, each processing element needs additional memory addressing and control hardware, which will increase the circuit complexity.

We now consider the time complexity and the area complexity of this linear systolic array implementation. It requires $O(N)$ units of time for solving the N-point IDCT with this linear systolic array with N processing elements. The area complexity of this linear systolic array which has the same order as the total number of registers or memory storage is $O(N^2)$. Therefore, the VLSI performance measure AT^2 of this linear systolic array implementation is $O(N^4)$, where A means the circuit area and T means the execution time. However, this performance can not be satisfied.

We now show that the computation required by the row-column evaluation can be represented

by two matrix-matrix multiplications. Suppose that the $N_1 \times N_2$ data matrix Y is generated after Step 2. Then, by the row-column evaluation,

$$x^T_{N_1 \times N_2} = C_{N_2 \times N_2} \, (C_{N_1 \times N_1} \, Y_{N_1 \times N_2})^T,$$

where $C_{N_1 \times N_1}$ and $C_{N_2 \times N_2}$ are the N_1-point and the N_2-point IDCT coefficient matrices, respectively. That is,

$$T_{N_1 \times N_2} = C_{N_1 \times N_1} \, Y_{N_1 \times N_2} \quad \text{and} \quad x^T_{N_1 \times N_2} = C_{N_2 \times N_2} \, T^T_{N_1 \times N_2}.$$

Thus, the operation complexity of the prime-factor IDCT is $O(N\,(N_1 + N_2))$ in the sequential algorithm. This is better than the operation complexity $O(N^2)$ of the IDCT implemented by the matrix-vector multiplication.

In the following, we show how to use mesh-connected systolic arrays for implementing the row-column evaluation. Because the matrix-matrix multiplication can be implemented by a three-nested for-loop algorithm, we can design a mesh-connected systolic array to implement it. The method we use to design systolic arrays is based on the theoretical result developed by Lee (of this paper) and Kedem [15], which presented the formal necessary and sufficient conditions for the correct implementations of nested for-loop algorithms on systolic arrays of arbitrary dimensions. In addition, the method allows us to design processing elements with a constant number of registers, thus making the array modularly extensible. That is, we can use one type of processing element, with a constant number of registers independent of the input problem size, to construct a systolic array for a target algorithm. This approach is attractive because the algorithm can be executed faster than it could be on a computer using conventional memory addressing techniques.

We will omit details of transforming the matrix-matrix multiplication algorithm onto mesh-connected systolic arrays in this presentation. However, for completeness, we will describe, but not in complete detail and formalism, our target systolic arrays. The interested readers are referred to the theoretical results as stated in [15].

In order to implement the row-column evaluation, three components are required: an $N_1 \times N_1$ mesh-connected systolic array for implementing the matrix-matrix multiplication $T_{N_1 \times N_2} =$

$C_{N_1 \times N_1} Y_{N_1 \times N_2}$; a transpose component for implementing $T^T_{N_1 \times N_2}$ from $T_{N_1 \times N_2}$; and an $N_2 \times N_2$ mesh-connected systolic array for implementing the matrix-matrix multiplication $x^T_{N_1 \times N_2} = C_{N_2 \times N_2} T^T_{N_1 \times N_2}$.

Fig. 3 shows the complete data flow of the 15-point IDCT systolic array implementation. When given an input data sequence X_k, $0 \leq k < 15$, we transform this 1-D data sequence by the Ruritanian mapping to a (3×5)-point 2-D data matrix $X_{(k_1,k_2)}$ by using only one processing element, for $0 \leq k_1 < 3$ and $0 \leq k_2 < 5$. Next, we modify the 2-D data matrix $X_{(k_1,k_2)}$ to the 2-D data matrix $Y_{(k_1,k_2)}$ according to Equation (7) also by using only one processing element.

Then, we use three systolic array components to implement the row-column evaluation. First, we deal with the matrix-matrix multiplication $T_{3\times5} = C_{3\times3} Y_{3\times5}$ using a 3×3 mesh-connected systolic array. The data entries $C^3_{(i,j)} = \cos(\frac{\pi(2i+1)j}{2\cdot3})$ of the 3-point IDCT coefficient matrix are stored in the 3×3 mesh-connected systolic array initially. The data matrix $Y_{(k_1,k_2)}$, which requires it to be aligned skewedly, flows from north to south. The intermediate data matrix $T_{(k_1,k_2)}$, which also requires it to be aligned skewedly and whose initial value $T^0_{(k_1,k_2)}$ is zero, flows from west to east. In each processing element of the 3×3 mesh-connected systolic array, it contains a multiply-and-add circuit which implements $T_{(k_1,k_2)} = T_{(k_1,k_2)} + C^3_{(k_1,i)} * Y_{(i,k_2)}$, as shown in Fig. 4 ($O = I_2 + R * I_1$), for some integers k_1, k_2, and i.

Second, the intermediate data matrix $T_{(k_1,k_2)}$ then passes through an adjustment buffer, which adds one unit of delay between two adjacent data entries of the data flow of $T_{(k_1,k_2)}$. This one unit of delay between two adjacent data entries is necessary when the data matrix $T_{(k_1,k_2)}$ enters into the transpose component to obtain the transposed skewed data matrix $T^T_{(k_1,k_2)}$.

The 3×5 transpose component computes $T^T_{3\times5}$ from $T_{3\times5}$, where T is the input matrix of size 3×5 ($N_1 \times N_2$) and T^T is the output matrix of size 5×3 ($N_2 \times N_1$). The (k_1, k_2)-th entry of T ($T_{(k_1,k_2)}$) will enter into the transpose component from the k_1-th row, then will shift to the east buffer cell in each of the time units until to the (N_2-1-k_2)-th column of the transpose component, and then will shift to the north buffer cell in each of the time units. The resulting skewed data

matrix $T^T_{N_1 \times N_2}$ can then be obtained.

Third, we deal with the matrix-matrix multiplication $x^T_{3\times5} = C_{5\times5}\, T^T_{3\times5}$ using a 5×5 mesh-connected systolic array. The data entries $C^5_{(i,j)} = \cos(\frac{\pi(2i+1)j}{2\cdot5})$ of the 5-point IDCT coefficient matrix are stored in the 5×5 mesh-connected systolic array initially. The skewed data matrix $T^T_{(k_1,k_2)}$ flows from south to north. The data matrix $x_{(n_1,n_2)}$, which requires it to be aligned skewedly and whose initial value $x^0_{(n_1,n_2)}$ is zero, flows from east to west. The operations performed in each processing element of the 5×5 mesh-connected systolic array are the same as the case in the 3×3 mesh-connected systolic array as shown in Fig. 4. In effect, the circuit of the processing element in the 3×3 mesh-connected systolic array is the same as the one in the 5×5 mesh-connected systolic array. This is the advantage of designing the modularly extensible systolic array, which allows us to use only one type of processing element to construct systolic arrays for a target algorithm with different problem sizes.

The resulting data matrix $x_{(n_1,n_2)}$ can then undergo the output index mapping according to Equation (8), and the output data sequence x_n, $0 \leq n < 15$, can then be obtained. This step, however, can be implemented by using only one processing element.

We now analyze the time complexity and the area complexity of the mesh-connected systolic array implementation which implements the row-column evaluation. It requires $O(N_1 + N_2)$ units of time for solving two matrix-matrix multiplications and one transpose operation. Because each processing element only contains a constant number of registers, the area complexity of this implementation is $O(N_1^2 + N_1N_2 + N_2^2)$. Therefore, the VLSI performance measure AT^2 of this implementation is $O((\max\{N_1, N_2\})^4)$, which is much better than the case in *Design II*.

5 Computation of the IDST by using the IDCT

In this section, we show how to compute the IDST (inverse discrete sine transform) by using the IDCT. Wang [29] showed that an N-point DST could be computed by an N-point DCT. In the following, we generalize his method to compute the IDST by using the IDCT. Therefore, the prime-factor IDST can be computed by the prime-factor IDCT. Since the DST is orthonormal, the forward transform also can be realized by taking the transpose of the inverse transform.

Recall that in Equation (2), the IDCT output data sequence x_n^c of X_k^c, $0 \leq n, k \leq N-1$, is defined as

$$x_n^c = \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} \epsilon^c(k)\, X_k^c \cos(\frac{\pi(2n+1)k}{2N}) ,$$

where

$$\epsilon^c(k) = \begin{cases} \frac{1}{\sqrt{2}} , & \text{for } k = 0; \\ 1 , & \text{for } 1 \leq k \leq N-1 . \end{cases}$$

The IDST output data sequence x_m^s of X_h^s, $0 \leq m, h \leq N-1$, is defined as

$$x_m^s = \sqrt{\frac{2}{N}} \sum_{h=0}^{N-1} \epsilon^s(h)\, X_h^s \sin(\frac{\pi(2m+1)(h+1)}{2N}) , \tag{9}$$

where

$$\epsilon^s(h) = \begin{cases} 1 , & \text{for } 0 \leq h \leq N-2; \\ \frac{1}{\sqrt{2}} , & \text{for } h = N-1. \end{cases}$$

Note that, for the sake of distinction, we use the superscript 'c' for denoting the cosine transform, and 's' for denoting the sine transform.

We now state how to compute the IDST by using the IDCT. First, by substituting

$$k = N-1-h , \quad \text{for } 0 \leq h \leq N-1 ,$$

into Equation (9), we have

$$x_m^s = \sqrt{\frac{2}{N}} \sum_{k=N-1}^{0} \epsilon^s(N-1-k)\, X_{N-1-k}^s \sin(\frac{\pi(2m+1)(N-k)}{2N}) .$$

Since

$$\sin(\frac{\pi(2m+1)(N-k)}{2N}) = (-1)^m \cos(\frac{\pi(2m+1)k}{2N}) \text{ and } \epsilon^s(N-1-k) = \epsilon^c(k),$$

we obtain that

$$\begin{aligned}(-1)^m x^s_m &= \sqrt{\frac{2}{N}} \sum_{k=N-1}^{0} \epsilon^c(k) X^s_{N-1-k} \cos(\frac{\pi(2m+1)k}{2N}) \\ &= \sqrt{\frac{2}{N}} \sum_{k=0}^{N-1} \epsilon^c(k) X^s_{N-1-k} \cos(\frac{\pi(2m+1)k}{2N}) .\end{aligned}$$

That is, the data sequence $\{y_m = (-1)^m x^s_m\}$, for $0 \le m \le N-1$, can be obtained by doing the IDCT on the data sequence $\{Y_k = X^s_{N-1-k}\}$, for $0 \le k \le N-1$.

Suppose that N is a product of two mutually prime integers. In the following, we list the procedure for computing the prime-factor IDST by using the prime-factor IDCT.

Prime-factor algorithm for the IDST:

Step 1: Compute the data sequence $\{Y_k = X^s_{N-1-k}\}$, for $0 \le k \le N-1$;

Step 2: compute the IDCT output data sequence $\{y_m\}$, $0 \le m \le N-1$, for the data sequence $\{Y_k\}$, $0 \le k \le N-1$, by using the prime-factor IDCT algorithm;

Step 3: compute $x^s_m = (-1)^m y_m$, for $0 \le m \le N-1$.

Fig. 5 shows the procedure for computing the 15-point prime-factor IDST from the 15-point prime-factor IDCT.

6 Conclusion

We have presented in this paper a new prime-factor DCT algorithm, which consists of four steps: the input index mapping; the modification; the row-column evaluation; and the output index mapping. The input index mapping we adopted is the Ruritanian mapping method, which is much simpler than the ones designed by Yang and Narasimha [35], and Lee [13]. The output index mapping we employed is the same one as Lee's [13], for which might be the most natural one in view of the DCT transform kernel's structure. We also designed a prime-factor DST algorithm, which is based on the prime-factor DCT algorithm.

We then considered hardware implementations for the prime-factor DCT. We have shown three hardware designs for the prime-factor DCT. The first one, which is a VLSI circuit fabricated directly according to the signal-flow graph, might be not easy to implement when the problem size becomes large. The second one is a linear systolic array implementation, which contains N processing elements and can solve an N-point DCT problem in $O(N)$ systolic steps. The third one is a mesh-connected systolic array implementation, which contains $O(N_1^2+N_1N_2+N_2^2)$ processing elements and can solve the row-column evaluation in $O(N_1+N_2)$ systolic steps, where $N=N_1N_2$, and N_1 and N_2 are mutually prime. If we consider the VLSI performance measure AT^2 on the second and the third designs, then the latter one, which is $O((\max\{N_1, N_2\})^4)$, is better than the former one, which is $O(N^4)$.

Appendix

Lemma 1 *For each* (k_1,k_2) *in* $[1,N_1-1]\times[1,N_2-1]$, *neither* $f(k_1,k_2)=N$ *nor* $g(k_1,k_2)=0$.

Proof : Suppose that, for some (k_1,k_2) in $[1,N_1-1]\times[1,N_2-1]$, we have $f(k_1,k_2)=N_2k_1+N_1k_2=$

N. Then, $N_2k_1 = N_1(N_2 - k_2)$. Because N_1 and N_2 are relatively prime, k_1 must be a multiple of N_1. However, this is a contradiction, since k_1 is in $[1, N_1 - 1]$.

Similarly, we can proof that $g(k_1, k_2) \neq 0$, for all (k_1, k_2) in $[1, N_1 - 1] \times [1, N_2 - 1]$. □

Lemma 2 *(1) If (k_1, k_2) is in $(A \cup B)$, then $\psi(k_1, k_2) = f(k_1, k_2)$.*
(2) If (k_1, k_2) is in $(C \cup D)$, then $\psi(k_1, k_2) = g(k_1, N_2 - k_2)$.

Proof :

(1) Because when (k_1, k_2) is in $(A \cup B)$, $N > f(k_1, k_2) = N_2k_1 + N_1k_2 = \psi(k_1, k_2)$.

(2) Because when (k_1, k_2) is in $(C \cup D)$, $N < f(k_1, k_2) = N_2k_1 + N_1k_2 < 2N$. Therefore,

$$\psi(k_1, k_2) = f(k_1, k_2) - N = N_2k_1 - N_1(N_2 - k_2) = g(k_1, N_2 - k_2).$$ □

Lemma 3 *(1) $f(k_1, k_2) = 2N - f(N_1 - k_1, N_2 - k_2)$.*
(2) $g(k_1, k_2) = -g(N_1 - k_1, N_2 - k_2)$.

Proof : (1) $f(N_1 - k_1, N_2 - k_2) = N_2(N_1 - k_1) + N_1(N_2 - k_2) = 2N - f(k_1, k_2)$.
(2) $g(N_1 - k_1, N_2 - k_2) = N_2(N_1 - k_1) - N_1(N_2 - k_2) = -g(k_1, k_2)$. □

Lemma 4 *(1) (k_1, k_2) is in A if and only if $(N_1 - k_1, N_2 - k_2)$ is in C.*
(2) (k_1, k_2) is in B if and only if $(N_1 - k_1, N_2 - k_2)$ is in D.
(3) (k_1, k_2) is in C if and only if $(k_1, N_2 - k_2)$ is in A.
(4) (k_1, k_2) is in D if and only if $(k_1, N_2 - k_2)$ is in D.

Proof : We show the only-if part of (1), other parts can be proved similarly.

If (k_1, k_2) is in A. Then, $f(k_1, k_2) < N$ and $g(k_1, k_2) > 0$. We now show that $(N_1 - k_1, N_2 - k_2)$ is in C. First, because (k_1, k_2) is in $[1, N_1 - 1] \times [1, N_2 - 1]$, so is $(N_1 - k_1, N_2 - k_2)$. Second, from Lemma

3-(1), $f(N_1-k_1, N_2-k_2) = 2N - f(k_1,k_2) > N$; and from Lemma 3-(2), $g(N_1-k_1, N_2-k_2) = -g(k_1,k_2) < 0$. Therefore, by definition, (N_1-k_1, N_2-k_2) is in C. □

Proof of Theorem 1 :

From Lemma 1 and by definition, we obtain that all of the lattice points in $[1, N_1-1]\times[1, N_2-1]$ is just equal to $A \cup B \cup C \cup D$. Therefore, the lattice points on the region $[0, N_1-1]\times[0, N_2-1]$ can be divided into five disjoint groups: E; A; B; C; and D. From the definition of the Ruritanian mapping,

$$
\begin{aligned}
& \{\, k \mid k \in [0, N-1],\ \ k \text{ is an integer} \,\} \\
= \ & \{\, k \mid k = \psi(k_1,k_2),\ (k_1,k_2) \text{ in } E \,\} \cup \\
& \{\, k \mid k = \psi(k_1,k_2),\ (k_1,k_2) \text{ in } (A \cup B) \,\} \cup \\
& \{\, k \mid k = \psi(k_1,k_2),\ (k_1,k_2) \text{ in } C \,\} \cup \\
& \{\, k \mid k = \psi(k_1,k_2),\ (k_1,k_2) \text{ in } D \,\} \\
= \ & \{\, k \mid k = \psi(k_1,k_2),\ (k_1,k_2) \text{ in } E \,\} \cup \\
& \{\, k \mid k = f(k_1,k_2),\ (k_1,k_2) \text{ in } (A \cup B) \,\} \cup \quad \text{/* Lemma 2-(1) */} \\
& \{\, k \mid k = g(k_1, N_2-k_2),\ (k_1,k_2) \text{ in } C \,\} \cup \quad \text{/* Lemma 2-(2) */} \\
& \{\, k \mid k = g(k_1, N_2-k_2),\ (k_1,k_2) \text{ in } D \,\} \quad \text{/* Lemma 2-(2) */} \\
= \ & \{\, k \mid k = \psi(k_1,k_2),\ (k_1,k_2) \text{ in } E \,\} \cup \\
& \{\, k \mid k = f(k_1,k_2),\ (k_1,k_2) \text{ in } (A \cup B) \,\} \cup \\
& \{\, k \mid k = g(k_1,k_2),\ (k_1,k_2) \text{ in } A \,\} \cup \quad \text{/* Lemma 4-(3) */} \\
& \{\, k \mid k = g(k_1,k_2),\ (k_1,k_2) \text{ in } D \,\} \quad \text{/* Lemma 4-(4) */} \\
= \ & W \cup U \cup V \,. \qquad \square
\end{aligned}
$$

Lemma 5

$$x_n = \frac{1}{2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} Y_{(k_1,k_2)} \left[\cos\left(\frac{\pi(2n+1)\, f(k_1,k_2)}{2N}\right) + \cos\left(\frac{\pi(2n+1)\, g(k_1,k_2)}{2N}\right) \right].$$

Proof :

(right-hand side) $\times$ 2

$$
\begin{aligned}
= \quad & \textstyle\sum_{(k_1,k_2)\in E} X_{(k_1,k_2)} \left[\cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) \right] + \\
& \{ [\textstyle\sum_{(k_1,k_2)\in A} X_{(k_1,k_2)} + \sum_{(k_1,k_2)\in B} X_{(k_1,k_2)} + \sum_{(k_1,k_2)\in C} (-X_{(N_1-k_1,N_2-k_2)}) + \sum_{(k_1,k_2)\in D} (-X_{(N_1-k_1,N_2-k_2)})] \\
& + [\textstyle\sum_{(k_1,k_2)\in A} X_{(k_1,N_2-k_2)} + \sum_{(k_1,k_2)\in B} X_{(N_1-k_1,k_2)} + \sum_{(k_1,k_2)\in C} X_{(N_1-k_1,k_2)} + \sum_{(k_1,k_2)\in D} X_{(k_1,N_2-k_2)}] \} \\
& \times \left[\cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) \right] \\
= \quad & \textstyle\sum_{(k_1,k_2)\in E}^{[1,2]} X_{(k_1,k_2)} \left[\cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) \right] + \\
& [\textstyle\sum_{(k_1,k_2)\in A}^{[3]} X_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in B}^{[4]} X_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in C}^{[5]} (-X_{(N_1-k_1,N_2-k_2)}) \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in D}^{[6]} (-X_{(N_1-k_1,N_2-k_2)}) \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right)] + \\
& [\textstyle\sum_{(k_1,k_2)\in A}^{[7]} X_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in B}^{[8]} X_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in C}^{[9]} (-X_{(N_1-k_1,N_2-k_2)}) \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in D}^{[10]} (-X_{(N_1-k_1,N_2-k_2)}) \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right)] + \\
& [\textstyle\sum_{(k_1,k_2)\in A}^{[11]} X_{(k_1,N_2-k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in B}^{[12]} X_{(N_1-k_1,k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in C}^{[13]} X_{(N_1-k_1,k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in D}^{[14]} X_{(k_1,N_2-k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right)] + \\
& [\textstyle\sum_{(k_1,k_2)\in A}^{[15]} X_{(k_1,N_2-k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in B}^{[16]} X_{(N_1-k_1,k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in C}^{[17]} X_{(N_1-k_1,k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + \\
& \textstyle\sum_{(k_1,k_2)\in D}^{[18]} X_{(k_1,N_2-k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right)] \\
= \quad & \textstyle\sum_{(k_1,k_2)\in E}^{[1,2]} X_{(k_1,k_2)} \left[\cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) \right] + \\
& [\textstyle\sum_{(k_1,k_2)\in A}^{[3]} X^f_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + && \text{/* Lemma 2-(1) */} \\
& \textstyle\sum_{(k_1,k_2)\in B}^{[4]} X^f_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) f(k_1,k_2)}{2N}\right) + && \text{/* Lemma 2-(1) */} \\
& \textstyle\sum_{(k_1,k_2)\in C}^{[5]} (-X^f_{(N_1-k_1,N_2-k_2)}) \cos\left(\frac{\pi(2n+1)(2N-f(N_1-k_1,N_2-k_2))}{2N}\right) + && \text{/* Lemmas 2-(1), 3-(1) */} \\
& \textstyle\sum_{(k_1,k_2)\in D}^{[6]} (-X^f_{(N_1-k_1,N_2-k_2)}) \cos\left(\frac{\pi(2n+1)(2N-f(N_1-k_1,N_2-k_2))}{2N}\right)] + && \text{/* Lemmas 2-(1), 3-(1) */} \\
& [\textstyle\sum_{(k_1,k_2)\in A}^{[7]} X^f_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + && \text{/* Lemma 2-(1) */} \\
& \textstyle\sum_{(k_1,k_2)\in B}^{[8]} X^f_{(k_1,k_2)} \cos\left(\frac{\pi(2n+1) g(k_1,k_2)}{2N}\right) + && \text{/* Lemma 2-(1) */} \\
& \textstyle\sum_{(k_1,k_2)\in C}^{[9]} (-X^f_{(N_1-k_1,N_2-k_2)}) \cos\left(\frac{\pi(2n+1)(-g(N_1-k_1,N_2-k_2))}{2N}\right) + && \text{/* Lemmas 2-(1), 3-(2) */}
\end{aligned}
$$

$\sum_{(k_1,k_2)\in D}^{[10]} (-X^f_{(N_1-k_1,N_2-k_2)}) \cos(\frac{\pi(2n+1)(-g(N_1-k_1,N_2-k_2))}{2N})] +$ /* Lemmas 2-(1), 3-(2) */

$[\sum_{(k_1,k_2)\in A}^{[11]} X^g_{(k_1,k_2)} \cos(\frac{\pi(2n+1)f(k_1,k_2)}{2N}) +$ /* Lemma 2-(2) */

$\sum_{(k_1,k_2)\in B}^{[12]} X^g_{(N_1-k_1,N_2-k_2)} \cos(\frac{\pi(2n+1)(2N-f(N_1-k_1,N_2-k_2))}{2N}) +$ /* Lemmas 2-(2), 3-(1) */

$\sum_{(k_1,k_2)\in C}^{[13]} X^g_{(N_1-k_1,N_2-k_2)} \cos(\frac{\pi(2n+1)(2N-f(N_1-k_1,N_2-k_2))}{2N}) +$ /* Lemmas 2-(2), 3-(1) */

$\sum_{(k_1,k_2)\in D}^{[14]} X^g_{(k_1,k_2)} \cos(\frac{\pi(2n+1)f(k_1,k_2)}{2N})] +$ /* Lemma 2-(2) */

$[\sum_{(k_1,k_2)\in A}^{[15]} X^g_{(k_1,k_2)} \cos(\frac{\pi(2n+1)g(k_1,k_2)}{2N}) +$ /* Lemma 2-(2) */

$\sum_{(k_1,k_2)\in B}^{[16]} X^g_{(N_1-k_1,N_2-k_2)} \cos(\frac{\pi(2n+1)(-g(N_1-k_1,N_2-k_2))}{2N}) +$ /* Lemmas 2-(2), 3-(2) */

$\sum_{(k_1,k_2)\in C}^{[17]} X^g_{(N_1-k_1,N_2-k_2)} \cos(\frac{\pi(2n+1)(-g(N_1-k_1,N_2-k_2))}{2N}) +$ /* Lemmas 2-(2), 3-(2) */

$\sum_{(k_1,k_2)\in D}^{[18]} X^g_{(k_1,k_2)} \cos(\frac{\pi(2n+1)g(k_1,k_2)}{2N})]$ /* Lemmas 2-(2) */

/* Applying Lemma 4-(1) and 4-(2), [3] = [5], [4] = [6], [15] = [17], [16] = [18], [7] + [9] = 0, [8] + [10] = 0, [11] + [13] = 0, [12] + [14] = 0. */

$= \sum_{(k_1,k_2)\in E} X_{(k_1,k_2)} [\cos(\frac{\pi(2n+1)\,f(k_1,k_2)}{2N}) + \cos(\frac{\pi(2n+1)\,g(k_1,k_2)}{2N})] +$

$2\sum_{(k_1,k_2)\in(A\cup B)} X^f_{(k_1,k_2)} \cos(\frac{\pi(2n+1)\,f(k_1,k_2)}{2N}) +$

$2\sum_{(k_1,k_2)\in(A\cup D)} X^g_{(k_1,k_2)} \cos(\frac{\pi(2n+1)\,g(k_1,k_2)}{2N})$

$=$ (right-hand side of Equation (6)) $\times$ 2 . □

Proof of Theorem 2 :

Since

$$
\begin{aligned}
f &= \cos(\frac{\pi(2n+1)\,f(k_1,k_2)}{2N}) + \cos(\frac{\pi(2n+1)\,g(k_1,k_2)}{2N}) \\
&= \cos(\frac{\pi(2n+1)(N_2k_1+N_1k_2)}{2N}) + \cos(\frac{\pi(2n+1)(N_2k_1-N_1k_2)}{2N}) \\
&= 2\cos(\frac{\pi(2n+1)N_2k_1}{2N}) \cos(\frac{\pi(2n+1)N_1k_2}{2N}) \\
&= 2\cos(\frac{\pi(2n+1)k_1}{2N_1}) \cos(\frac{\pi(2n+1)k_2}{2N_2})
\end{aligned}
$$

from Lemma 5, we have

$$x_n = \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} Y_{(k_1,k_2)} \cos(\frac{\pi(2n+1)k_1}{2N_1}) \cos(\frac{\pi(2n+1)k_2}{2N_2}) .$$ □

References

[1] N. Ahmed, T. Natarajan, and K. R. Rao. Discrete Cosine transform. *IEEE Transactions on Computers*, C-23(1):90–93, January 1974.

[2] C. S. Burrus. Index mappings for multidimensional formulation of the DFT and convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-25(3):239–242, June 1977.

[3] C. S. Burrus. Bit reverse unscrambling for a radix-2^M FFT. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1809–1810. IEEE, 1987.

[4] C. S. Burrus. Unscrambling for fast DFT algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1086–1087, July 1988.

[5] C. S. Burrus and P. W. Eschenbacher. An in-place, in-order prime factor FFT algorithm. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-29(4):806–817, August 1981.

[6] C. Chakrabarti and J. JáJá. Systolic architectures for the computation of the discrete Hartley and the discrete Cosine transforms based on prime factor decomposition. *IEEE Transactions on Computers*, C-39(11):1359–1368, November 1990.

[7] M. T. Heideman. Computation of an odd-length DCT from a real-valued DFT of the same length. *IEEE Transactions on Signal Processing*, 40(1):54–61, January 1992.

[8] H. S. Hou. A fast recursive algorithm for computing the discrete Cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(10):1455–1461, October 1987.

[9] A. K. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, Englewood Cliffs, NJ, 1989.

[10] D. P. Kolba and T. W. Parks. A prime factor FFT algorithm using high-speed convolution. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-25(4):281–294, August 1977.

[11] H. T. Kung and C. E. Leiserson. *Introduction to VLSI Systems*, chapter 8.3 Algorithms for VLSI Processor Arrays. Addison-Wesley, Reading, MA, C. Mead and L. Conway edition, 1980.

[12] B. G. Lee. A new algorithm to compute the discrete Cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-32(6):1243–1245, December 1984.

[13] B. G. Lee. Input and ouput index mappings for a prime-factor decomposed computation of discrete Cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(2):237–244, February 1989.

[14] P.-Z. Lee and F. Y. Huang. New recursive algorithms for computing the 1-D and 2-D discrete Cosine transforms. Technical Report TR-93-001, Institute of Information Science, Academia Sinica, 1993. Submitted for publication, a preliminary version is presented in the *Fifth Digital Signal Processing Workshop*, Starved Rock State Park, IL., September, 1992.

[15] P.-Z. Lee and Z. M. Kedem. Mapping nested loop algorithms into multi-dimensional systolic arrays. *IEEE Transactions on Parallel and Distributed Systems*, 1:64–76, January 1990.

[16] W. Li. A new algorithm to compute the DCT and its inverse. *IEEE Transactions on Signal Processing*, 39(6):1305–1313, June 1991.

[17] D. P. K. Lun, R. Chan, and W. C. Siu. Yet a fast address generation scheme for the computation of prime factor algorithms. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1499–1502. IEEE, 1990.

[18] D. P. K. Lun and W. C. Siu. On prime factor mapping for the discrete Hartley transform. *IEEE Transactions on Signal Processing*, 40(6), June 1992.

[19] M. J. Narasimha and A. M. Peterson. On the computation of the discrete Cosine transform. *IEEE Transactions on Communications*, COM-26:934–936, June 1978.

[20] H. J. Nussbaumer. *Fast Fourier Transform and Convolution Algorithms*, volume 2 of *Springer Series in Information Sciences*. Springer-Verlag, Berlin, Heidelberg, and New York, 1981.

[21] R. K. Rao and P. Yip. *Discrete Cosine Transform: Algorithms, Advantages, and Applications*. Academic Press, Inc., 1990.

[22] J. H. Rothweiler. Implementation of the in-order prime factor transform for variable sizes. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-30(1):105–107, February 1982.

[23] H. V. Sorensen, D. L. Jones, C. S. Burrus, and M. T. Heideman. On computing the discrete Hartley transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 33(4):1231–1238, October 1985.

[24] H. V. Sorensen, D. L. Jones, M. T. Heideman, and C. S. Burrus. Real-valued fast Fourier transform algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-35(6):849–863, June 1987.

[25] C. Temperton. A note on prime factor FFT algorithms. *Journal of Computational Physics*, 52:198–204, 1983.

[26] C. Temperton. Implementation of a self-sorting in-place prime factor FFT algorithm. *Journal of Computational Physics*, 58:283–299, 1985.

[27] C. Temperton. Implementation of a prime factor FFT algorithm on CRAY-1. *Parallel Computing*, 6:99–108, 1988.

[28] M. Vetterli and H. J. Nussbaumer. Simple FFT and DCT algorithm with reduced number of operations. *Signal Processing*, 6(4):267–278, 1984.

[29] Z. Wang. A fast algorithm for the discrete Sine transform implemented by the fast Cosine transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-30:814–815, 1982.

[30] Z. Wang. Fast algorithms for the discrete W transform and for the discrete Fourier transform. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, ASSP-32(4):803–816, August 1984.

[31] S. Winograd. On computing the discrete Fourier transform. *Mathematics of Computation*, 32(141):175–199, January 1978.

[32] K. L. Wong, R. Chan, D. P. K. Lun, and W. C. Siu. Efficient address generation for prime factor algorithms. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 38(9):1518–1528, September 1990.

[33] K. L. Wong and W. C. Siu. Fast address generation for the computation of prime factor algorithms. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1091–1094. IEEE, 1989.

[34] K. L. Wong and W. C. Siu. Data routing networks for systolic/pipeline realization of prime factor mapping. *IEEE Transactions on Computers*, 40(9):1072–1074, September 1991.

[35] P. P. N. Yang and M. J. Narasimha. Prime factor decomposition of the discrete Cosine transform. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 772–775. IEEE, 1985.

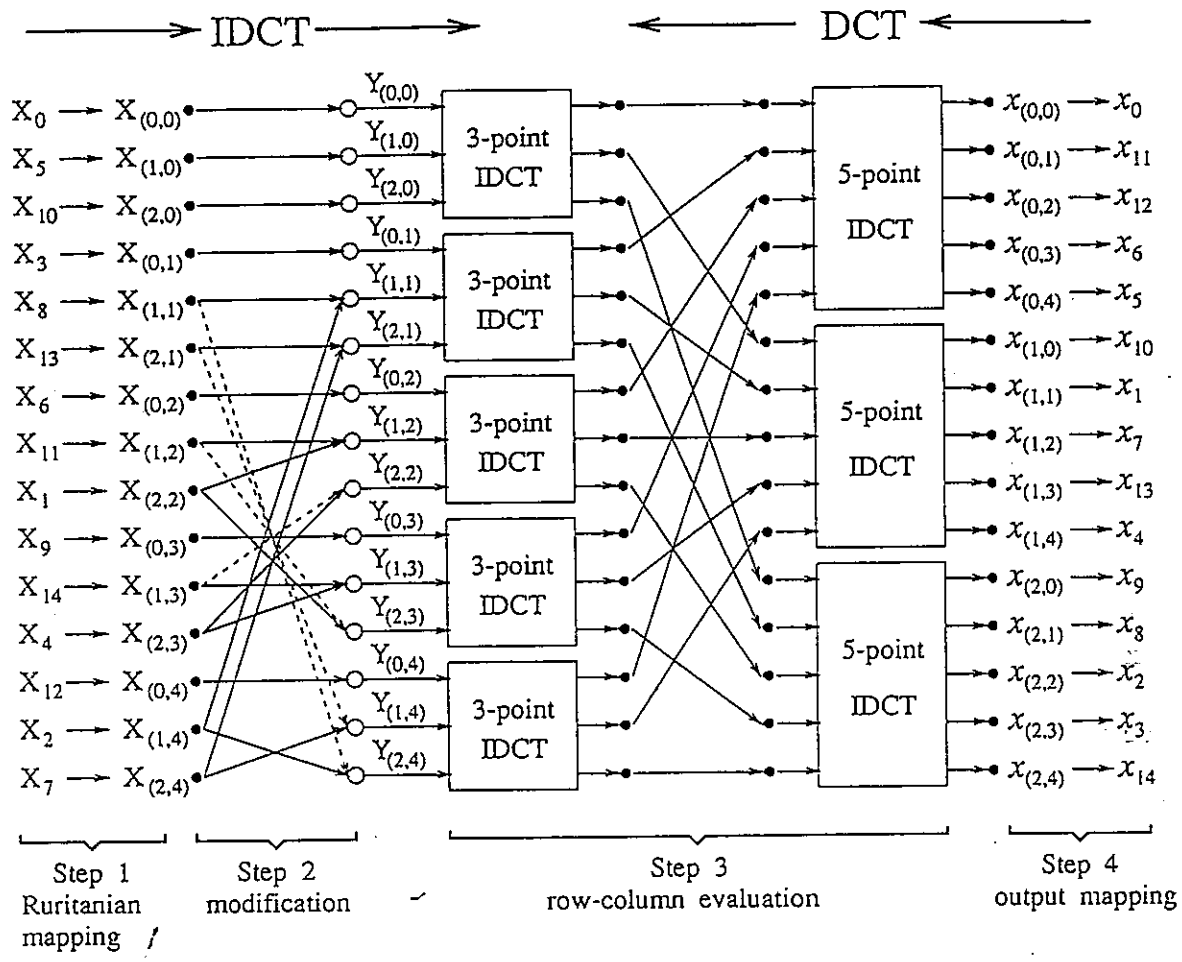

Fig. 1. The signal-flow graph for implementing the 15-point IDCT, which can be decomposed into the (3×5)-point 2-D IDCT. If signals flow from left to right, it performs the IDCT; if signals flow from right to left, it performs the DCT. Solid lines represent transfer factor 1, while dashed lines represent transfer factor -1. Circles $\circ$ represent adders.

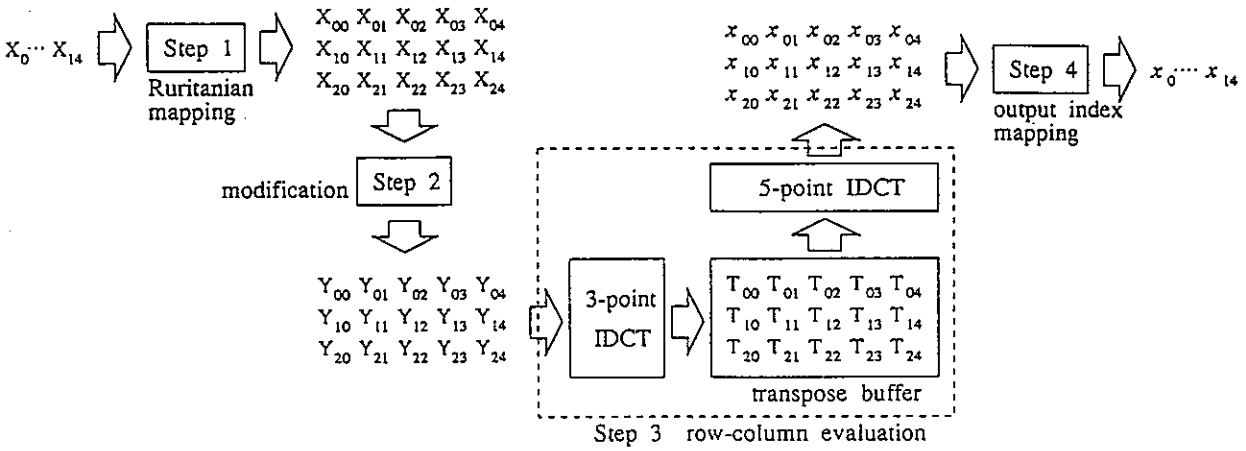

Fig. 2. One 3-point IDCT component, one transpose buffer, and one 5-point IDCT component are necessary for computing the row-column evaluation for the 15-point prime-factor IDCT.

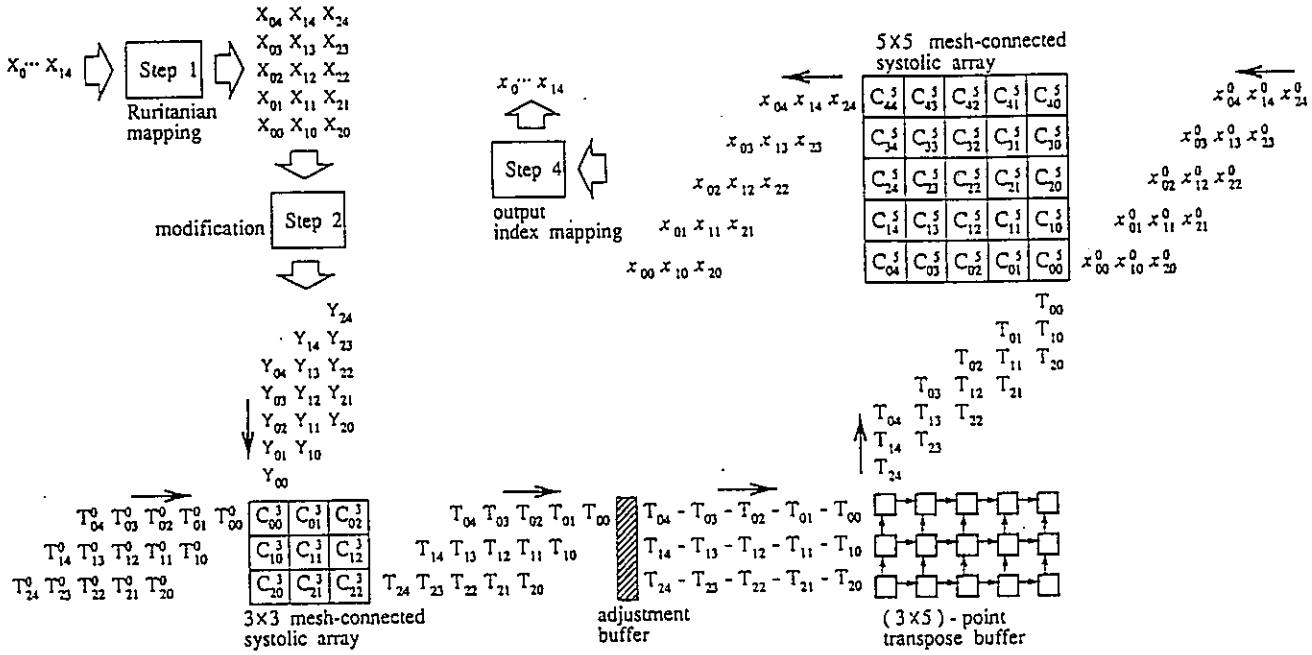

Fig. 3. The complete data-flow diagram of the mesh-connected systolic array implementation for computing the 15-point prime-factor IDCT.

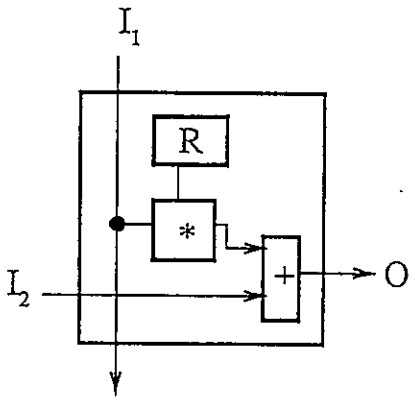

Fig. 4. Each processing element of the mesh-connected systolic array contains a multiply-and-add circuit, which implements $O = I_2 + R * I_1$.

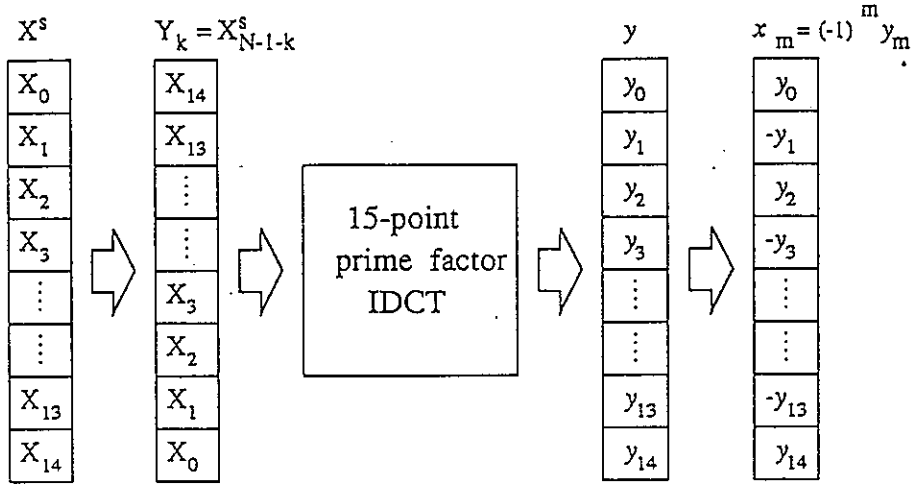

Fig. 5. The procedure for computing the 15-point prime-factor IDST from the 15-point prime-factor IDCT.