Fast Algorithm and Implementation of 2-D Discrete Cosine Transform

Nam Ik Cho, *Student Member, IEEE*, and Sang Uk Lee, *Member, IEEE*

Abstract—**In this paper, a new algorithm for the fast computation of a 2-D discrete cosine transform (DCT) is presented. It is shown that the $N \times N$ DCT, where $N = 2^m$, can be computed using only N 1-D DCT's and additions, instead of using $2N$ 1-D DCT's, as in the conventional row–column approach. Hence the total number of multiplications for the proposed algorithm is only half of that required for the row–column approach, and is also less than that of most of other fast algorithms, while the number of additions is almost comparable to that of others. It is also shown that only $N/2$ 1-D DCT modules are required for hardware parallel implementation of the proposed algorithm. Thus the number of actual multipliers being used is only a quarter of that required for the conventional approach.**

I. Introduction

SINCE DCT approaches the statistically optimal Karhunen–Loeve transform (KLT) for highly correlated signals, it is widely used in digital signal processing, especially for speech and image data compression [1], [2]. Thus many algorithms and VLSI architectures for the fast computation of DCT have been proposed [3]–[10]. For the fast computation of 2-D DCT, the conventional approach is the row–column method. This method requires $2N$ 1-D DCT's for the computation of the $N \times N$ DCT. However, for hardware parallel implementation of the conventional approach, a complicated matrix transposition architecture as well as $2N$ 1-D DCT modules is required. Thus for more efficient computation or parallel implementation of the 2-D DCT, the algorithms that work directly on the 2-D data set have been introduced [11]–[16]. The most efficient 2-D DCT algorithm appeared in the literature is the direct polynomial approach proposed by Duhamel [16], in which the number of multiplications is reduced to 50% of the conventional approach. On the other hand, the algorithms in [13] and [14] require 75%, and the indirect approach using the polynomial transform FFT and rotation proposed by Vetterli [15] requires between 50% and 75% of the conventional approach. More recently, a fast algorithm for the 4×4 DCT is proposed [17]. This algorithm is restricted to the size of 4×4 transform, because the derivation is very complicated to be generalized to any $2^m \times 2^m$ cases, where m is a positive integer. Hence it is useful for the computation of larger size transforms only by incorporating with the recursive 2-D DCT technique [14]. In the case of the 4×4 DCT, it requires the same number of multiplications as in [15] and [16]. However, for the larger size transforms, the 2-D recursive technique employing the 4×4 DCT [17] requires more multiplications than those of [15] and [16].

In this paper, a new fast 2-D DCT algorithm, which may be viewed as a modification and generalization of the 4×4 DCT [17], is proposed. It will be shown that the proposed algorithm requires only N 1-D DCT's for the computation of the $N \times N$ DCT, where $N = 2^m$. Hence the number of multiplications required for the proposed algorithm is only half of that required for the conventional approach, which is, in fact, the same number of multiplications as reported in [16]. However, as compared with Duhamel's algorithm [16], the proposed algorithm has advantage in that the computation structure is highly regular and systematic, and only real arithmetic is required. Also, we shall show that $N/2$ 1-D DCT modules are sufficient for the hardware parallel implementation of the algorithm. Hence the number of actual multipliers being used in hardware implementation is a quarter of that required for the conventional approach. Thus the proposed 2-D DCT algorithm is very suitable for the VLSI implementation.

The rest of the paper is organized as follows. In Section II, we will introduce a new fast 2-D DCT algorithm along with an examples for 8×8 DCT. Also, the examples for 4×4 DCT, 8×8 inverse DCT (IDCT), and 4×4 IDCT are provided. The comparison of the number of multiplications and additions with other fast algorithms [14]–[16] is also given in this section. In Section III, we shall discuss the parallel implementation of the algorithm. Finally, in Section IV, we give conclusions.

II. The Fast Algorithm and Its Parallel Implementation

In this section, we shall describe a new fast algorithm for 2-D DCT that requires only half the number of multiplications compared to the conventional row–column method. Also, we shall provide examples for 8×8 and 4×4 DCT's. The examples for IDCT are also given.

A. A Fast 2-D DCT Algorithm

For a given 2-D data sequence $\{x_{ij}: i, j = 0, 1, \cdots, N-1\}$, the 2-D DCT sequence $\{Y_{mn}: m, n = 0, 1, \cdots, N-1\}$ is given by

$$Y_{mn} = \frac{4}{N^2} u(m) u(n) \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x_{ij} \cos \frac{(2i+1)m}{2N} \pi \cos \frac{(2j+1)n}{2N} \pi \quad (1)$$

Manuscript received April 25, 1990. This paper was recommended by Associate Editor T. R. Hsing.

The authors are with the Department of Control and Instrumentation Engineering, Seoul National University, Seoul 151-742, Korea.

IEEE Log Number 9041758.

0098-4094/91/0300-0297$01.00 ©1991 IEEE

where

$$u(m)=\begin{cases}1/\sqrt{2}, & m=0,\\ 1, & \text{otherwise.}\end{cases}$$

We will neglect the scale factor $4u(m)u(n)/N^2$ for convenience. Then, let us define a denormalized form of Y_{mn} as

$$y_{mn}=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x_{ij}\cos\frac{(2i+1)m}{2N}\pi\cos\frac{(2j+1)n}{2N}\pi \quad (2.a)$$

i.e.,

$$Y_{mn}=\frac{4}{N^2}u(m)u(n)\cdot y_{mn}. \quad (2.b)$$

The main idea behind the 4×4 algorithm proposed in [17] is that the 4×4 DCT can be decomposed into four separate four-point 1-D DCT's by using the following relations:

$$\cos\frac{(2i+1)m}{2N}\pi\cos\frac{(2j+1)n}{2N}\pi=\frac{1}{2}\left(\cos\frac{(2i+1)m+(2j+1)n}{2N}\pi+\cos\frac{(2i+1)m-(2j+1)n}{2N}\pi\right). \quad (3)$$

In this paper, we shall make use of the above relation for the computation of the $N\times N$ DCT, where $N\geq 8$. Using the relation in (3), the $N\times N$ DCT can be separated into two transforms as given by

$$y_{mn}=1/2\left\{\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x_{ij}\cos\frac{(2i+1)m+(2j+1)n}{2N}\pi+\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x_{ij}\cos\frac{(2i+1)m-(2j+1)n}{2N}\pi\right\},$$

$$\text{for } m,n=1,1,2,\cdots,N-1. \quad (4)$$

For convenience, by defining new transforms A_{mn} and B_{mn} as

$$A_{mn}=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x_{ij}\cos\frac{(2i+1)m+(2j+1)n}{2N}\pi \quad (5.a)$$

$$B_{mn}=\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}x_{ij}\cos\frac{(2i+1)m-(2j+1)n}{2N}\pi \quad (5.b)$$

y_{mn} can be rewritten as

$$y_{mn}=1/2(A_{mn}+B_{mn}). \quad (6)$$

Now, we shall show that A_{mn} and B_{mn} can be expressed in terms of N 1-D DCT's by some data ordering and manipulations, so that the $N\times N$ DCT can be obtained from N separate 1-D DCT's. It is noted that the condition for the kernels of the transforms in (5) to be equivalent to that of 1-D DCT's is that $\{(2i+1)m\pm(2j+1)n\}$ should be expressed as $(2i+1)$ multiplied by some integer. To satisfy such a condition, we can see that $(2j+1)$ should be either a multiple of $(2i+1)$ modulo $2N$ or $2N$ minus a multiple of $(2i+1)$ mudulo $2N$, i.e.,

$$(2j+1)=p(2i+1) \bmod 2N \quad (7.a)$$

or

$$(2j+1)=2N-p(2i+1) \bmod 2N \quad (7.b)$$

where p is an odd integer ranging from 1 to $N-1$, because the value of p out of this range yields the same value of j as one of those produced by the p in the range. The relations in (7) are equivalent to

$$j=pi+(p-1)/2 \bmod N \quad (8.a)$$

or

$$j=N-1-pi-(p-1)/2 \bmod N,$$

$$\text{for } p=1,3,\cdots,N-1. \quad (8.b)$$

It can be easily shown that when i ranges from 0 to $N-1$, N sequences for j obtained by (8) are mutually different. Thus the 2-D input data set can be grouped into N different data sets, each of whose indexes satisfies the relations in (8). Then, we can see that the kernel of the transforms for each of these data sets is equivalent to that of 1-D DCT. To distinguish each of the sequences of j obtained by (8.a) or (8.b) for $p=1,3,5,\cdots,N-1$, let us denote them as

$$j(p;a)=pi+(p-1)/2 \bmod N \quad (9.a)$$

or

$$j(p;b)=N-1-pi-(p-1)/2 \bmod N,$$

$$\text{for } p=1,3,\cdots,N-1, \text{ and } i=0,1,2,\cdots,N-1. \quad (9.b)$$

That is, for given p, $\{j(p;a)\colon i=0,1,\cdots,N-1\}$ is the sequence of j obtained by (8.a) and $\{j(p;b)\colon i=0,1,\cdots,N-1\}$ is the sequence of j obtained by (8.b). Hence, by grouping the 2-D input sequence $\{x_{ij}\colon i,j=0,1,2,\cdots,N-1\}$ into N 1-D sequences $\{x_{ij(p;a)}\colon i=0,1,2,\cdots,N-1\}$ and $\{x_{ij(p;b)}\colon i=0,1,2,\cdots,N-1\}$ for $p=1,3,5,\cdots,N-1$, the 1-D transforms in (5) can be expressed as sum of 1-D DCT's. We will denote these 1-D data sequences by R_p^a and R_p^b, respectively. Then, they can be expressed as

$$R_p^a=\{x_{ij(p;a)}\colon i=0,1,2,\cdots,N-1,\ j(p;a)=pi+(p-1)/2 \bmod N\}. \quad (10.a)$$

$$R_p^b=\{x_{ij(p;b)}\colon i=0,1,2,\cdots,N-1,\ j(p;b)=N-1-pi-(p-1)/2 \bmod N\},$$

$$\text{for } p=1,3,5,\cdots,N-1. \quad (10.b)$$

However, for the proof of which we are in pursuit, it is necessary to know the exact result of $pi+(p-1)/2$ divided by N, while only the remainder of the division can be perceived from (10). In other words, we need to know the quotient of the division as well as the remainder. Hence, by introducing a new integer sequence q_{pi}, which is a quotient of $pi+(p-1)/2$ divided by N, we can rewrite (10) (without

"mod") as

$$R_p^a = \{x_{ij(p;a)}: i = 0,1,2,\cdots,N-1,$$
$$j(p;a) = pi + (p-1)/2 - Nq_{pi}\} \quad (11.a)$$

$$R_p^b = \{x_{ij(p;b)}: i = 0,1,2,\cdots,N-1,$$
$$j(p;b) = N-1-pi-(p-1)/2 + Nq_{pi}\},$$
$$\text{for} \quad p = 1,3,5,\cdots,N-1. \quad (11.b)$$

As an example, for $N = 8$, since p has the value of 1, 3, 5, and 7, the 8×8 2-D data set can be grouped into

$$R_1^a = \{x_{00}, x_{11}, x_{22}, x_{33}, x_{44}, x_{55}, x_{66}, x_{77}\} \quad (12.a)$$
$$R_1^b = \{x_{07}, x_{16}, x_{25}, x_{34}, x_{43}, x_{52}, x_{61}, x_{70}\} \quad (12.b)$$
$$R_3^a = \{x_{01}, x_{14}, x_{27}, x_{32}, x_{45}, x_{50}, x_{63}, x_{76}\} \quad (12.c)$$
$$R_3^b = \{x_{06}, x_{13}, x_{20}, x_{35}, x_{42}, x_{57}, x_{64}, x_{71}\} \quad (12.d)$$
$$R_5^a = \{x_{02}, x_{17}, x_{24}, x_{31}, x_{46}, x_{53}, x_{60}, x_{75}\} \quad (12.e)$$
$$R_5^b = \{x_{05}, x_{10}, x_{23}, x_{36}, x_{41}, x_{54}, x_{67}, x_{72}\} \quad (12.f)$$
$$R_7^a = \{x_{03}, x_{12}, x_{21}, x_{30}, x_{47}, x_{56}, x_{65}, x_{74}\} \quad (12.g)$$
$$R_7^b = \{x_{04}, x_{15}, x_{26}, x_{37}, x_{40}, x_{51}\, x_{62}, x_{73}\} \quad (12.h)$$

where the quotient sequences q_{pi}'s for $p = 1$, 3, 5, and 7 are

$$q_{1i} = \{0,0,0,0,0,0,0,0\} \quad (13.a)$$
$$q_{3i} = \{0,0,0,1,1,2,2,2\} \quad (13.b)$$
$$q_{5i} = \{0,0,1,2,2,3,4,4\} \quad (13.c)$$
$$q_{7i} = \{0,1,2,3,3,4,5,6\}. \quad (13.d)$$

Now, using the data groupings shown in (11), we can express A_{mn} and B_{mn} as

$$A_{mn} = \sum_{\substack{p=1 \\ (p:\text{ odd})}}^{N-1} \{T_p^a(m,n) + T_p^b(m,n)\} \quad (14.a)$$

$$B_{mn} = \sum_{\substack{p=1 \\ (p:\text{ odd})}}^{N-1} \{S_p^a(m,n) + S_p^b(m,n)\} \quad (14.b)$$

where $T_p^a(m,n)$, $T_p^b(m,n)$, $S_p^a(m,n)$, and $S_p^b(m,n)$ are defined as

$$T_p^a(m,n) = \sum_{x_{ij}\in R_p^a} x_{ij} \cos\frac{(2i+1)m + (2j+1)n}{2N}\pi \quad (15.a)$$

$$T_p^b(m,n) = \sum_{x_{ij}\in R_p^b} x_{ij} \cos\frac{(2i+1)m + (2j+1)n}{2N}\pi \quad (15.b)$$

$$S_p^a(m,n) = \sum_{x_{ij}\in R_p^a} x_{ij} \cos\frac{(2i+1)m - (2j+1)n}{2N}\pi \quad (15.c)$$

$$S_p^b(m,n) = \sum_{x_{ij}\in R_p^b} x_{ij} \cos\frac{(2i+1)m - (2j+1)n}{2N}\pi. \quad (15.d)$$

Then, from (6), y_{mn} can be rewritten as

$$y_{mn} = \sum_{\substack{p=1 \\ (p:\text{ odd})}}^{N-1} 1/2\{T_p^a(m,n) + T_p^b(m,n) + S_p^a(m,n) + S_p^b(m,n)\}. \quad (16)$$

Thus, in order to show that y_{mn} is the summation of 1-D DCT's, it remains to show that $T_p^a(m,n)$, $T_p^b(m,n)$, $S_p^a(m,n)$, and $S_p^b(m,n)$ can be expressed in terms of 1-D DCT's. In order to do so, by substituting the relation in (11.a) into (15.a), we have

$$T_{pa}(m,n) = \sum_{i=0}^{N-1} x_{ij(p;a)} \cos\frac{(2i+1)m + \{p(2i+1) - 2Nq_{pi}\}n}{2N}\pi$$

but it can be separated into two cases where n is even or odd, i.e.,

$$T_p^a(m,n) = \begin{cases} \displaystyle\sum_{i=0}^{N-1} x_{ij(p;a)} \cos\frac{(2i+1)(m+pn)}{2N}\pi, \\ \qquad \text{when } n \text{ is even} \quad (17.a) \\ \displaystyle\sum_{i=0}^{N-1} (-1)^{q_{pi}} x_{ij(p;a)} \cos\frac{(2i+1)(m+np)}{2N}\pi, \\ \qquad \text{when } n \text{ is odd.} \quad (17.b) \end{cases}$$

Also, substitution of the relation in (11.b) into (15.b) leads to

$$T_p^b(m,n) = \sum_{i=0}^{N-1} x_{ij(p;b)} \cos\frac{(2i+1)(m-np) + 2N(1+q_{pi})n}{2N}\pi$$

but this is also expressed separately depending on n, i.e.,

$$T_p^b(m,n) = \begin{cases} \displaystyle\sum_{i=0}^{N-1} x_{ij(p;b)} \cos\frac{(2i+1)(m-pn)}{2N}\pi, \\ \qquad \text{when } n \text{ is even} \quad (18.a) \\ \displaystyle -\sum_{i=0}^{N-1} (-1)^{q_{pi}} x_{ij(p;b)} \cos\frac{(2i+1)(m-np)}{2N}\pi, \\ \qquad \text{when } n \text{ is odd.} \quad (18.b) \end{cases}$$

In the same way, by substituting (11.a) into (15.c), we have

$$S_p^a(m,n) = \begin{cases} \displaystyle\sum_{i=0}^{N-1} x_{ij(p;a)} \cos\frac{(2i+1)(m-pn)}{2N}\pi, \\ \qquad \text{when } n \text{ is even} \quad (19.a) \\ \displaystyle\sum_{i=0}^{N-1} (-1)^{q_{pi}} x_{ij(p;a)} \cos\frac{(2i+1)(m-np)}{2N}\pi, \\ \qquad \text{when } n \text{ is odd.} \quad (19.b) \end{cases}$$

Also, by substituting (11.b) into (15.d), we have

$$S_p^b(m,n)=\begin{cases}\sum_{i=0}^{N-1} x_{ij(p;b)}\cos\dfrac{(2i+1)(m+pn)}{2N}\pi, \\ \qquad \text{when } n \text{ is even} \qquad (20.a)\\ -\sum_{i=0}^{N-1}(-1)^{q_{pi}}x_{ij(p;b)}\cos\dfrac{(2i+1)(m+np)}{2N}\pi, \\ \qquad \text{when } n \text{ is odd.} \qquad (20.b)\end{cases}$$

Then, by substituting (17)–(20) into (16), we can express y_{mn} as

$$y_{mn}=\begin{cases}1/2\sum\limits_{\substack{p=1\\(p:\text{ odd})}}^{N-1}\Big[\sum\limits_{i=0}^{N-1}(x_{ij(p;a)}+x_{ij(p;b)})\cos\dfrac{(2i+1)(m+np)}{2N}\pi \\ \quad +\sum\limits_{i=0}^{N-1}(x_{ij(p;a)}+x_{ij(p;b)})\cos\dfrac{(2i+1)(m-np)}{2N}\pi\Big], \\ \qquad \text{when } n \text{ is even} \qquad (21.a)\\ 1/2\sum\limits_{\substack{p=1\\(p:\text{ odd})}}^{N-1}\Big[\sum\limits_{i=0}^{N-1}(-1)^{q_{pi}}(x_{ij(p;a)}-x_{ij(p;b)})\cos\dfrac{(2i+1)(m+np)}{2N}\pi \\ \quad +\sum\limits_{i=0}^{N-1}(-1)^{q_{pi}}(x_{ij(p;a)}-x_{ij(p;b)})\cos\dfrac{(2i+1)(m-np)}{2N}\pi\Big], \\ \qquad \text{when } n \text{ is odd.} \qquad (21.b)\end{cases}$$

But, it can be seen that

$$\sum_{i=0}^{N-1}(x_{ij(p;a)}+x_{ij(p;b)})\cos\frac{(2i+1)(m+np)}{2N}\pi \quad (22.a)$$

and

$$\sum_{i=0}^{N-1}(x_{ij(p;a)}+x_{ij(p;b)})\cos\frac{(2i+1)(m-np)}{2N}\pi \quad (22.b)$$

correspond to one of 1-D DCT's of data sequence $\{x_{ij(p;a)}+x_{ij(p;b)}\}$ depending on m and n. That is, by defining

$$f_{pl}=\sum_{i=0}^{N-1}(x_{ij(p;a)}+x_{ij(p;b)})\cos\frac{(2i+1)l}{2N}\pi \quad (23)$$

we can see that (22.a) and (22.b) correspond to one of $+f_{pl}$ or $-f_{pl}$ for some $l=0,1,2,\cdots,N-1$. In the case of (21.b), by defining

$$g_{pl}=\sum_{i=0}^{N-1}(-1)^{q_{pi}}(x_{ij(p;a)}-x_{ij(p;b)})\cos\frac{(2i+1)l}{2N}\pi \quad (24)$$

it can be seen that

$$\sum_{i=0}^{N-1}(-1)^{q_{pi}}(x_{ij(p;a)}-x_{ij(p;b)})\cos\frac{(2i+1)(m+np)}{2N}\pi \quad (25.a)$$

and

$$\sum_{i=0}^{N-1}(-1)^{q_{pi}}(x_{ij(p;a)}-x_{ij(p;b)})\cos\frac{(2i+1)(m-np)}{2N}\pi \quad (25.b)$$

correspond to one of $\pm g_{pl}$ for some $l=0,1,2,\cdots,N-1$. Hence, for the computation of the $N\times N$ DCT sequence $\{y_{mn}:\ m,n=0,1,2,\cdots,N-1\}$, we need only $\{f_{pl}:\ l=0,1,2,\cdots,N-1\}$ and $\{g_{pl}:\ l=0,1,2,\cdots,N-1\}$ for $p=1,3,5,\cdots,N-1$. This implies that the computation of $N\times N$ DCT requires only the computation of N 1-D DCT's.

Now, after f_{pl} and g_{pl} are obtained, let us discuss the additions and other operations required for the computation of $N\times N$ DCT. From (21) and the definitions in (23) and (24), it is seen that y_{mn}'s are expressed in terms of the summation of f_{pl}'s and g_{pl}'s. In order to see the relationships that exist between y_{mn} and f_{pl}'s, g_{pl}'s, for some arbitrarily chosen m and n, let us give some examples for $N=8$ as follows:

$$y_{30}=1/2(f_{13}+f_{13}+f_{33}+f_{33}+f_{53}+f_{53}+f_{73}+f_{73}) \quad (26.a)$$

$$y_{52}=1/2(f_{17}+f_{13}-f_{35}+f_{31}-f_{51}+f_{55}-f_{73}-f_{77}) \quad (26.b)$$

$$y_{34}=1/2(f_{17}+f_{11}-f_{31}-f_{37}-f_{57}-f_{51}+f_{71}+f_{77}) \quad (26.c)$$

$$y_{26}=1/2(0+f_{14}-f_{34}-f_{30}+f_{50}+f_{54}-f_{74}+0) \quad (26.d)$$

$$y_{41}=1/2(g_{15}+g_{13}+g_{37}+g_{31}-g_{57}+g_{51}-g_{75}+g_{73}) \quad (26.e)$$

$$y_{03}=1/2(g_{13}+g_{13}-g_{37}-g_{37}-g_{51}-g_{51}-g_{75}-g_{75}) \quad (26.f)$$

$$y_{35}=1/2(0+g_{12}-g_{32}-g_{34}+g_{54}-g_{56}+g_{76}+g_{70}) \quad (26.g)$$

$$y_{57}=1/2(-g_{14}+g_{12}+g_{36}-g_{30}+0+g_{52}-g_{76}-g_{74}). \quad (26.h)$$

In the above example, addition operation in terms of f_{pl}'s and g_{pl}'s for computing y_{mn}'s looks complicated. However, we shall show that the addition operation can be implemented by butterfly stages as in ordinary fast algorithms for the discrete Fourier transform (DFT) and DCT. Now, in the case n is even, it can be seen that

$$\cos\frac{(2i+1)(m-n(N-p))}{2N}\pi=\pm\cos\frac{(2i+1)(m+np)}{2N}\pi \quad (27)$$

which implies that if

$$\sum_{i=0}^{N-1}(x_{ij(p;a)}+x_{ij(p;b)})\cos\frac{(2i+1)(m+np)}{2N}\pi=f_{pl} \quad (28.a)$$

for some $l = 0, 1, 2, \cdots, N-1$, then

$$\sum_{i=0}^{N-1} (x_{ij(N-p;a)} + x_{ij(N-p;b)}) \cdot \cos \frac{(2i+1)(m-n(N-p))}{2N} \pi = \pm f_{(N-p)l}. \quad (28.b)$$

In the same way, for n odd, it can be shown that if

$$\sum_{i=0}^{N-1} (-1)^{q_{pi}} (x_{ij(N;a)} - x_{ij(N;b)}) \cdot \cos \frac{(2i+1)(m+np)}{2N} \pi = g_{pl} \quad (29.a)$$

then

$$\sum_{i=0}^{N-1} (-1)^{q_{pi}} (x_{ij(N-p;a)} - x_{ij(N-p;b)}) \cdot \cos \frac{(2i+1)(m-n(N-p))}{2N} \pi = \pm g_{(N-p)(N-l)}. \quad (29.b)$$

These relationships reveal that f_{pl} always appears with $\pm f_{(N-p)l}$ in (26), allowing us to form a butterfly stage. In the case of g_{pl}, since it appears with $\pm g_{(N-p)(N-l)}$, we can also form a butterfly stage. For the example of $N = 8$, (26) can be rewritten as follows:

$$y_{30} = 1/2\{(f_{13} + f_{73}) + (f_{13} + f_{73}) + (f_{33} + f_{53}) + (f_{33} + f_{53})\} \quad (30.a)$$

$$y_{52} = 1/2\{(f_{17} - f_{77}) + (f_{13} - f_{73}) - (f_{35} + f_{55}) + (f_{31} - f_{51})\} \quad (30.b)$$

$$y_{34} = 1/2\{(f_{17} + f_{77}) + (f_{11} + f_{71}) - (f_{31} + f_{51}) - (f_{31} + f_{51})\} \quad (30.c)$$

$$y_{26} = 1/2\{(0+0) + (f_{14} - f_{74}) - (f_{34} - f_{54}) - (f_{30} + f_{50})\} \quad (30.d)$$

$$y_{41} = 1/2\{(g_{15} + g_{73}) + (g_{13} - g_{75}) + (g_{37} + g_{51}) + (g_{31} - g_{57})\} \quad (30.e)$$

$$y_{03} = 1/2\{(g_{13} - g_{75}) + (g_{13} - g_{75}) - (g_{37} + g_{51}) - (g_{37} + g_{51})\} \quad (30.f)$$

$$y_{35} = 1/2\{(0 + g_{70}) + (g_{12} + g_{76}) - (g_{32} + g_{56}) - (g_{34} - g_{54})\} \quad (30.g)$$

$$y_{57} = 1/2\{-(g_{14} + g_{74}) + (g_{12} - g_{76}) + (g_{36} + g_{52}) - (g_{30} + 0)\}. \quad (30.h)$$

Based on the formations shown above, as an example, the signal flow graph for an 8×8 DCT algorithm is shown in Fig. 1. The signal flow graph is separated into three parts for convenience, i.e., Fig. 1(a) is the signal flow graph from x_{ij}'s to f_{pl}'s and g_{pl}'s, Fig. 1(b) is from f_{pl}'s to y_{mn}'s, where n is even, and Fig. 1(c) is from g_{pl}'s to y_{mn}'s, where n is odd. From Fig. 1(a), it is seen that the 8×8 DCT requires only 8 1-D DCT's. From Fig. 1(b) and (c), it is also seen that the addition operations after the 1-D DCT stages can be implemented in butterfly form. The example for a 4×4 DCT is also shown in Fig. 2.

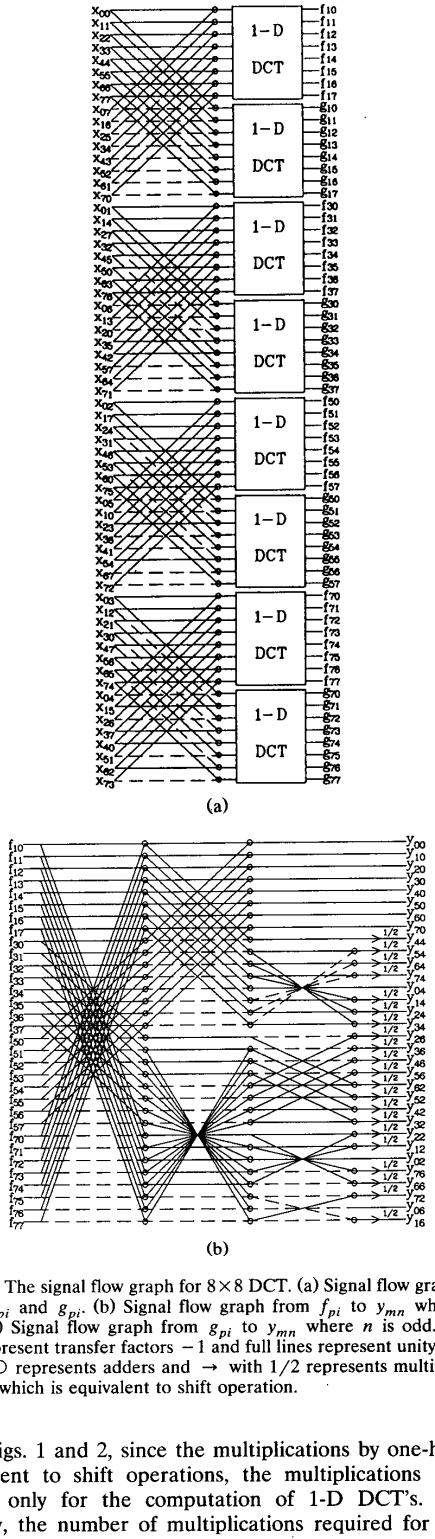
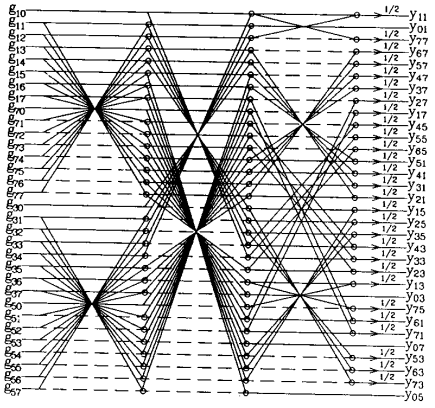


Fig. 1. The signal flow graph for 8×8 DCT. (a) Signal flow graph from x_{ij} to f_{pi} and g_{pi}. (b) Signal flow graph from f_{pi} to y_{mn} where n is even. (c) Signal flow graph from g_{pi} to y_{mn} where n is odd. Broken lines represent transfer factors -1 and full lines represent unity transfer factor. ○ represents adders and → with 1/2 represents multiplication by 1/2, which is equivalent to shift operation.

In Figs. 1 and 2, since the multiplications by one-half are equivalent to shift operations, the multiplications are required only for the computation of 1-D DCT's. Consequently, the number of multiplications required for $N \times N$ DCT is equivalent to that for N 1-D DCT's.



(c)

Fig. 1. (*Continued*)

B. Signal Flow Graph for IDCT

In the case of orthogonal transforms, if the scale factor is not taken into account, then the signal flow graph for the inverse transform is just the inverse of that for the forward transform. Similarly, in the proposed algorithm, the signal flow graph for the denormalized IDCT can be obtained by simply inverting the forward DCT. However, we need some modifications if we take into account the scale factor as shown in (1). As can be seen from the proposed signal flow graph for the forward transform in Figs. 1 or 2, there are some nodes that do not have their pairs. But this problem can be easily solved by multiplying the node variables by 2 when the direction of the flow is inverted. For example, in Fig. 1(b), it is seen that two nodes in the line of f_{50} and f_{70} do not have their pairs, and thus the node variables should be multiplied by 2 to keep the inverse flow correctly. Also, due to the scale factor required for 1-D IDCT, the zeroth input to 1-D IDCT should be multiplied by one-half, which is equivalent to shift operation. However, in the case of 1-D IDCT's for g_{pl}'s, the multiplications by one-half for g_{p0}'s and the multiplications by 2 for the node variables are cancelled out by each other. The signal flow graph for the 8×8 and 4×4 IDCT's are shown in Figs. 3 and 4, respectively. If one warts to maintain the scale factor for output x_{ij}'s correctly, it is necessary to multiply every node variables by one-half or to divide the input y_{mn}'s by $N^2/2$. However, in the case of fixed point computation, the former approach is better. In summary, when the output of the forward transform is used as input to these signal flow graphs for IDCT's shown in Figs. 3 and 4, they generate the same data sequence as the input to the forward transform.

C. Comparison with Other Fast 2-D DCT Algorithms

In this section, we will compare the number of multiplications and additions with those of other fast algorithms [14]–[16]. Let the number of multiplications and additions required for the proposed algorithm by M and A, respectively. Previously, we have shown that only N 1-D DCT's are required for the computation of the $N\times N$ DCT, and thus the number of multiplications is only half of that required for the conventional row–column approach. In implementing

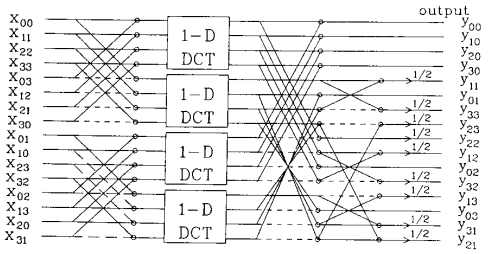


Fig. 2. The signal flow graph for 4×4 DCT.

the 1-D DCT's, we may use any existing 1-D DCT algorithms. When the highly efficient 1-D DCT algorithms proposed by Lee [6] or Hou [7] are used in the 1-D DCT computation, which require $(N/2)\log_2 N$ multiplications for an N-point 1-D DCT, the required number of multiplications for the $N\times N$ DCT is given by

$$M=(N^2/2)\log_2 N. \tag{31}$$

On the other hand, the required number of additions is the summation of those required for N 1-D DCT's and those for other additions as shown in Fig. 1. In other words, as can be seen from Figs. 1 or 2, we need additions for $(1+\log_2 N)$ butterfly stages and for the 1-D DCT stage. However, at the last stage, it is observed that the additions for y_{0j}'s, y_{i0}'s and $y_{(N/2)(N/2)}$ are not required. Also, we can see that g_{p0}'s and f_{p0}'s, except for f_{10} and f_{30}, do not require butterfly pairs. Thus the number of additions, except for those required for 1-D DCT stages, is shown to be $N^2(1+\log_2 N)-2N-(N-2)$. Since the number of additions required for 1-D DCT is $(3N/2)\log_2 N-N+1$ [6], [7], the total number of additions required for the $N\times N$ DCT is given by

$$A=(5N^2/2)\log_2 N-2N+2. \tag{32}$$

Thus, with the computations according to (31) and (32), we can compare the number of multiplications and additions with those required for other fast 2-D DCT algorithms, such as [14]–[16]. The results are summarized in Table I. It is seen that the proposed algorithm requires the same number of multiplications as in [16], which is the least of all and only half of that required for the conventional approach, while the number of additions is almost comparable to that of other algorithms.

III. Parallel Implementation of the Proposed Algorithm

For VLSI or hardware parallel implementation of an algorithm, reducing the number of multipliers is very important, because they occupy a large area of the chip. Also important considerations are regularity, modularity in the computation structure, and the complexity of data access scheme. In this context, we first describe an implementation scheme which reduces the number of multipliers being used for parallel implementation, and then discuss the problems such as modularity, regularity, and data access scheme of the architecture.

It was shown that the number of multiplications required for the proposed algorithm is equivalent to that required for N 1-D DCT's. Also, it seems that N 1-D DCT modules are required to compute $N\times N$ DCT in parallel. However, we

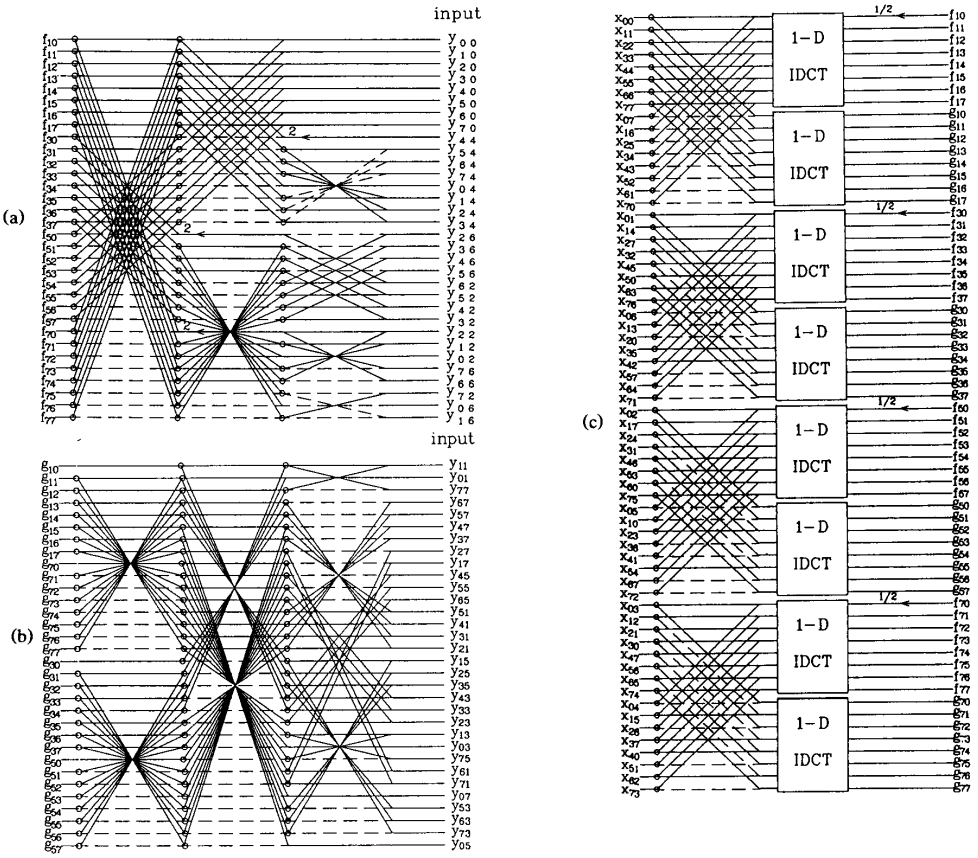


Fig. 3. The signal flow graph for 8×8 IDCT. (a) Signal flow graph from y_{mn} to f_{pi}, where n is even. (b) Signal flow graph from y_{mn} to g_{pi}, where n is odd. (c) Signal flow graph from f_{pi} and g_{pi} to x_{ij}.

TABLE I
COMPARISON OF THE NUMBER OF MULTIPLICATIONS AND ADDITIONS

	Number of multiplications					Number of additions				
	Conventional algorithm	Other fast algorithms			Proposed algorithm	Conventional algorithm	Other fast algorithms			Proposed algorithm
		[14]	[15]	[16]			[14]	[15]	[16]	
4×4	32	24	16	16	16	72	72	70	68	74
8×8	192	144	104	96	96	464	464	462	484	466
16×16	1024	768	568	512	512	2592	2592	2558	2531	2530
32×32	5120	3840	2840	2560	2560	13376	13376	12950	12578	12738

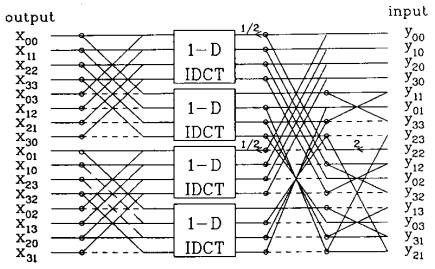


Fig. 4. The signal flow graph for the 4×4 IDCT.

shall show that $N/2$ 1-D DCT modules are sufficient by the use of multiplexers and demultiplexers. In Fig. 1(a), it is seen that the results for f_{pl} and g_{pl} remain to be the same even if the order of addition and 1-D DCT operation is reversed. Hence the signal flow graph in Fig. 1(a) is equivalent to Fig. 5, in which the order of addition and 1-D DCT operation is reversed for the data sets R_3^a, R_3^b, R_7^a, and R_7^b. Now, by using the multiplexers and demultiplexers as shown in Fig. 6, we can reduce the number of 1-D DCT modules to $N/2$. That is, in the upper part of Fig. 6, while the additions for R_1^a and R_1^b are in progress, 1-D DCT's for R_3^a and R_3^b can be started. Then, the results of additions for R_1^a and R_1^b are sent to 1-D

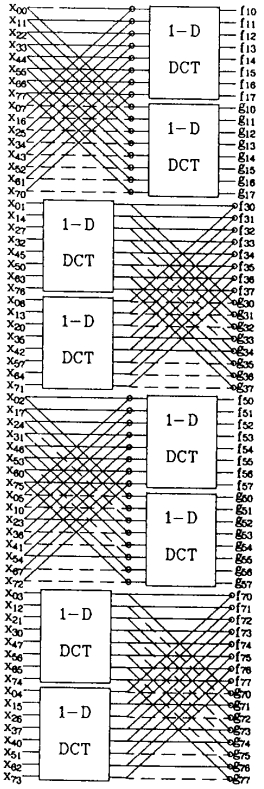


Fig. 5. Alternate signal flow graph of Fig. 1(a).

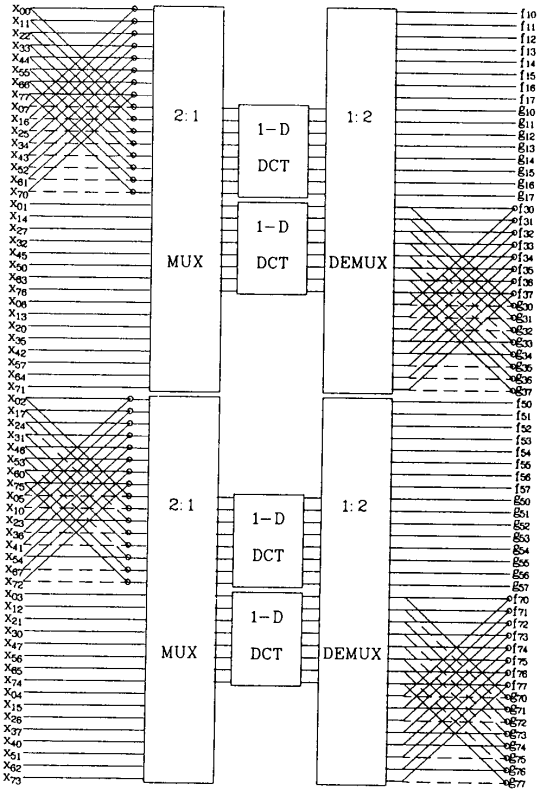


Fig. 6. Implementation of Fig. 1(a) with multiplexers and demultiplexers.

DCT processors. This is possible because the 1-D DCT architectures in [6] and [7] allow us to perform the multiple processing in parallel and pipelined environment. More specifically, since the data move successively in the pipelined structure, we can start the computation of next input data immediately after the current input data. Thus it is noted that the 1-D DCT module need not run twice the speed, yielding almost the same computation time as compared to that of implementation with N 1-D DCT modules. Similarly, in the lower part of Fig. 6, while the additions for R_5^a and R_5^b are being performed, 1-D DCT's for R_7^a and R_7^b can also be started. Then, the results of additions are sent to 1-D DCT modules throughout the multiplexer. The example for the parallel implementation of 4×4 DCT is also shown in Fig. 7. From Figs. 6 and 7, it is seen that the number of 1-D DCT modules required for $N \times N$ DCT is $N/2$, which is a quarter of that required for the conventional approach.

If we consider modularity and regularity, the proposed implementation scheme has advantage over other fast algorithms such as [15] and [16], in which the polynomial transform and the complex arithmetic are required. Hence, the proposed algorithm is believed to be more suitable for the VLSI implementation than other 2-D FDCT algorithms in terms of the number of multipliers, modularity, and regularity. However, there are another problems that should be addressed in the VLSI implementation. For example, like most of other fast algorithms, the proposed algorithm requires more complicated data access scheme and computation structure than the simple matrix–vector (row–column) approach [18]. But the matrix–vector approach results in the largest chip area. There are always trade-off's between the chip area and computation time, and hence there exist many variations in the implementations. Conclusively, it is very difficult to determine an appropriate criterion for the VLSI implementation of the algorithms. However, this problem is beyond the scope of this paper.

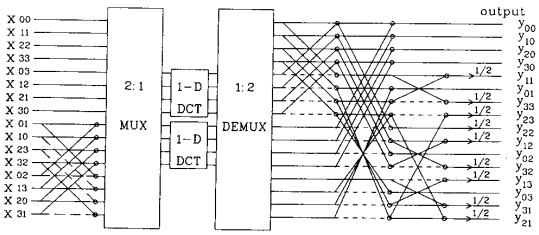


Fig. 7. Parallel implementation of the 4×4 DCT algorithm.

IV. Conclusions

In this paper, a fast algorithm for the 2-D DCT is proposed. It is shown that the $N \times N$ DCT is obtained from N 1-D DCT's with some additions and shift operations. Thus

the total number of multiplications required for the $N \times N$ DCT is N times that for the 1-D DCT, which is only half of that required for the conventional row–column approach. Hence the number of multiplication is the same as that of previously reported algorithm [16], which is known to be the best in terms of the number of multiplications, while the number of additions is comparable to others. However, the proposed algorithm has advantages that it has regular and systematic structure, and requires only real arithmetic, while the algorithm in [16] requires complex arithmetic. Also, in this paper, for the purpose of reducing the hardware complexity for the parallel implementation, an alternative scheme is described with slight increase in time complexity. The proposed scheme requires $N/2$ 1-D DCT modules, while the direct implementation requires N 1-D DCT modules. Since there are always trade-offs between the chip area and computation time, it is very difficult to compare the performance of the implementation of the fast algorithms. However, considering only the hardware complexity, the proposed algorithm is advantageous in that it requires very small number of multiplications and has regular and systematic structure compared to other fast algorithms.

Finally, another important aspect in a VLSI implementation is the precision of the algorithm, i.e., the amount of errors due to the fixed-point implementation. This problem is currently under investigation.

REFERENCES

[1] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-23, pp. 90–93, Jan. 1974.

[2] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*. New York: Springer-Verlag, 1975.

[3] W. H. Chen, C. H. Smith, and S. C. Fralick, "A fast computational algorithm for discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-25, pp. 1004–1009, Nov. 1977.

[4] M. J. Narashimha and A. M. Peterson, "On the computation of the discrete cosine transform," *IEEE Trans. Commun.*, vol. COM-26, pp. 934–936, June 1978.

[5] M. D. Wagh and H. Ganesh, "A new algorithm for the discrete cosine transform of arbitrary number of points," *IEEE Trans. Comput.*, vol. C-29, pp. 269–277, Apr. 1980.

[6] B. G. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-32, pp. 1243–1245, Dec. 1984.

[7] H. S. Hou, "A fast recursive algorithms for computing the discrete cosine transform," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-35, pp. 1455–1461, Oct. 1987.

[8] N. I. Cho and S. U. Lee, "DCT algorithms for VLSI parallel implementation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 38, pp. 121–127, Jan. 1990.

[9] M. Vetterli and H. Nussbaumer, "Simple FFT and DCT algorithms with reduced number of operations," *Signal Process*, vol. 6, pp. 267–278, Aug. 1984.

[10] P. Duhamel and H. H'Mida, "New 2^n DCT algorithms suitable for VLSI implementation," in *Proc. ICASSP'87*, pp. 1805–1808, 1987.

[11] J. Makhoul, "A fast cosine transform in one and two dimensions," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, pp. 27–34, Feb. 1980.

[12] F. A. Kamangar and K. R. Rao, "Fast algorithms for the 2-D discrete cosine transform," *IEEE Trans. Comput.*, vol. C-31, pp. 899–906, Sept. 1982.

[13] M. A. Haque, "A two-dimensional fast cosine transform," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. ASSP-33, pp. 1532–1539, Dec. 1985.

[14] C. Ma, "A fast recursive two dimensional cosine transform," *Intelligent Robots and Computer Vision: Seventh in a Series*, David P. Casasent, Ed., in *Proc. SPIE 1002*, pp. 541–548, 1988.

[15] M. Vetterli, "Fast 2-D discrete cosine transform," in *Proc. ICASSP'85*, Mar. 1985.

[16] P. Duhamel and C. Guillemot, "Polynomial transform computation of 2-D DCT," in *Proc. ICASSP '90*, pp. 1515–1518, Apr. 1990.

[17] N. I. Cho and S. U. Lee, "A fast 4×4 DCT algorithm for the recursive 2-D DCT," *IEEE Trans. Acoust., Speech, Signal Processing*, submitted for publication.

[18] M.-T. Sun, T.-C. Chen, and A. M. Gottlieb, "VLSI implementation of a 16×16 discrete cosine transform," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 610–617, Apr. 1989.

Nam Ik Cho (S'86) received the B.S. and M.S. degrees from Seoul National University, Seoul, Korea, in 1986 and 1988, respectively, in control and instrumentation engineering. He is currently working toward the Ph.D. degree at Seoul National University.

His research interest is in digital signal processing, including adapative filtering and VLSI implementation.

Sang Uk Lee (S'75–M'79) received the B.S. degree from Seoul National University in 1973, the M.S. degree from Iowa State University in 1976, and the Ph.D. degree from the University of Southern California, Los Angeles, in 1980, all in electrical engineering.

In 1980, he was with General Electric, Lynchburg, VA, and in 1981 he joined the M/A-COM Research Center, Rockville, MD. He is now with the Department of Control and Instrumentation at Seoul National University, where he is an Associate Professor. His current research interests are in the areas of image and speech signal processing, including VLSI and neural computing.

Dr. Lee is a member of Phi Kappa Phi.