Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2009, Article ID 485817, 9 pages
doi:10.1155/2009/485817

Research Article

Scaled AAN for Fixed-Point Multiplier-Free IDCT

P. P. Zhu,[1] J. G. Liu,[1] S. K. Dai,[1,2] and G. Y. Wang[1]

[1] *State Key Lab for Multi-Spectral Information Processing Technologies, Institute for Pattern Recognition and Artificial Intelligence, Huazhong University of Science and Technology, Wuhan 430074, China*
[2] *Information College, Huaqiao University, Quanzhou, Fujian 362011, China*

Correspondence should be addressed to J. G. Liu, jgliu@ieee.org

Received 6 May 2008; Revised 13 October 2008; Accepted 9 February 2009

Recommended by Ulrich Heute

An efficient algorithm derived from AAN algorithm (proposed by Arai, Agui, and Nakajima in 1988) for computing the Inverse Discrete Cosine Transform (IDCT) is presented. We replace the multiplications in conventional AAN algorithm with additions and shifts to realize the fixed-point and multiplier-free computation of IDCT and adopt coefficient and compensation matrices to improve the precision of the algorithm. Our 1D IDCT can be implemented by 46 additions and 20 shifts. Due to the absence of the multiplications, this modified algorithm takes less time than the conventional AAN algorithm. The algorithm has low drift in decoding due to the higher computational precision, which fully complies with IEEE 1180 and ISO/IEC 23002-1 specifications. The implementation of the novel fast algorithm for 32-bit hardware is discussed, and the implementations for 24-bit and 16-bit hardware are also introduced, which are more suitable for mobile communication devices.

1. Introduction

Discrete cosine transforms (DCTs) are widely used in speech coding and image compression. Among four types of discrete cosine transforms (type-I, type-II, type-III, and type-IV), DCT-II and DCT-III are frequently adopted in Codecs. 1D DCT-II (also known as forward DCT) and DCT-III (also known as Inverse DCT) are defined as follows:

$$\text{DCT-II:}\quad X(n) = \sum_{k=0}^{N-1} \left(c\sqrt{\frac{2}{N}}\right) x(k) \cos\frac{n(2k+1)\pi}{2N}, \quad (n = 0, 1, \ldots, N-1),$$

$$\text{DCT-III:}\quad x(k) = \sum_{n=0}^{N-1} \left(c\sqrt{\frac{2}{N}}\right) X(n) \cos\frac{n(2k+1)\pi}{2N}, \quad (k = 0, 1, \ldots, N-1), \tag{1}$$

where

$$c = \frac{1}{\sqrt{2}} \quad \text{for } u = 0, \text{ otherwise } 1. \tag{2}$$

Many existing image and video coding standards (such as JPEG, H.261, H.263, MEPG-1, MPEG-2, and MPEG-4 part 2) require the implementation of an integer-output approximation of the 8×8 inverse discrete cosine transform (IDCT) function, defined as follows:

$$x(k,l) = \sum_{n=0}^{7-1} \sum_{m=0}^{7-1} \left(\frac{c_n \cdot c_m}{4}\right) X(n,m) \cdot \cos\left[\frac{n(2k+1)\pi}{16}\right] \cdot \cos\left[\frac{m(2l+1)\pi}{16}\right] \quad (k,l = 0, 1, \ldots, 7), \tag{3}$$

where

$$c_u = \frac{1}{\sqrt{2}} \quad \text{for } u = 0, \text{ otherwise } 1, \qquad c_v = \frac{1}{\sqrt{2}} \quad \text{for } v = 0, \text{ otherwise } 1. \tag{4}$$

$X(n,m)$ $(n,m = 0, \ldots, 7)$ denote input IDCT coefficients, and the reconstructed pixel values are

$$\hat{x}(k,l) = \left\lfloor x(k,l) + \frac{1}{2} \right\rfloor \quad (k,l = 0, \ldots, 7). \tag{5}$$

In this paper, we will propose an efficient algorithm for implementing (3). The Inverse DCT is supposed to decode data modeled by different encoders with low drift.

Some classical DCT/IDCT algorithms have been proposed, such as, Lee [1], AAN [2], and LLM [3] algorithms. However, the slightly irregular structures of these classical algorithms require many floating-point multipliers and adders, which take much time for both hardware and software to implement. Therefore, many fast algorithms for DCT/IDCT were proposed in the past years [4–12]. In order to decrease the implementation complexity, some researchers developed the recursive transform algorithms by taking advantage of the local connectivity and the simple structures in the circuit realizations, which are particularly suitable for very large scale integration (VLSI) implementations [4–8]. However, comparing with the other fast algorithms, longer computational time and larger round-off errors limit the use of the recursive algorithms. To reduce the computational complexity, looking-up tables and accumulators instead of multipliers are used to compute inner products. This method is widely used in many DSP applications, such as DFT, DCT, convolutions, and digital filters [9, 10]. However, the hardware will probably encounter the out of memory problem, especially for mobile devices, because the looking-up tables require large storage memories.

Considering low-power implementations of IDCT on mobile devices with no or less floating-point multipliers and the requirements of higher precisions, less computational complexities, and less storage memories in application, some multiplier-free DCTs are presented. Among them, multiplier-free approximation of DCT based on lifting scheme is developed [11, 12]. The method replaces the butterfly computational structures in the original DCT signal flow graph with lifting structures. The advantage of the lifting structures is that each lifting step is a biorthogonal transform, and its inverse transform also has similar lifting structures, which means we just need to subtract what was added at forward transform to invert a lifting step. Hence, the original signals can still be perfectly constructed even if the floating-point multiplications result in the lifting steps are rounded to integers, as long as the same procedure is applied to both the forward and inverse transforms. In order to implement multiplier-free algorithm, the algorithm approximating the floating-point lifting coefficients by hardware friendly dyadic values can be implemented by only shift and addition operators. This kind of approximation of original DCT is called BinDCT. However in most cases BinDCT is not the best choice, because forward transforms and inverse transforms are always not implemented by the same procedures. Moreover, BinDCT introduces more multiplication operators into the signal flow graph, which decrease the computational precision remarkably. If we just use BinDCT for inverse transform and original DCT for forward transform, then the differences between original data and recover data cannot be neglected. It means that BinDCT cannot perform well to recover date modulated by other forward DCTs.

In this paper, we propose our novel multiplier-free IDCT algorithm derived from conventional AAN algorithm [2]. The algorithm contains no multiplications and is implemented only by fixed-point integer-arithmetic. In order to improve the precision and reduce the computational complexity, we adopt the scale factors to modulate a coefficient matrix and a compensation matrix.

The paper is organized as follows. In Section 2, we present the improvement of the 1D IDCT algorithm through deleting the multiplication operators in the conventional algorithm and replacing them with addition and shift operators. Then we discuss the 8×8 2D IDCT and its 32-bit hardware implementation in Section 3. Considering low-power implementations of IDCT on mobile devices with no or less floating multipliers, we propose the 8×8 2D IDCTs based for 24-bit and 16-bit hardware in Section 4. We then show the performances of our proposed algorithms, including their computational complexities and precisions in Section 5. Finally, we give the conclusions in Section 6.

2. Implementation of 1D IDCT

In this section, firstly we give a general method which is able to reform many existing 1D IDCTs. Then we try to use this approach to reform the traditional AAN algorithm. Finally, considering the characteristics of AAN flow graph, we propose a new and more efficient algorithm.

2.1. General Method to Reform Existing 1D IDCTs. The butterfly computational structures which are always found in most of the existing IDCTs, such as ones in [1–3], can be interpreted by the following equation:

$$T = (u \cdot a\cos\alpha + v \cdot b\cos\beta)\cos\gamma. \tag{6}$$

Here u and v are scale factors; a and b are integer inputs. Let $w_1 = u \cdot \cos\alpha \cdot \cos\gamma$ and $w_2 = v \cdot \cos\beta \cdot \cos\gamma$, then

$$T = aw_1 + bw_2. \tag{7}$$

The details of how to replace the multiplications in (7) with additions and shifts are given below.

Without losing of generality, assume that w_1 and w_2 are positive numbers and transform them into binary numbers,

$$\begin{aligned} w_1 &= m_0 + 2^{-1} \times m_1 + \cdots + 2^{-t+1} \times m_{t-1} + 2^{-t} \times m_t \\ &\qquad (m_i = 0 \text{ or } 1,\ i = 0, 1, \ldots, t), \\ w_2 &= n_0 + 2^{-1} \times n_1 + \cdots + 2^{-t+1} \times n_{t-1} + 2^{-t} \times n_t \\ &\qquad (n_i = 0 \text{ or } 1,\ i = 0, 1, \ldots, t), \end{aligned} \tag{8}$$

then

$$\begin{aligned} T &= aw_1 + bw_2 \\ &= (am_0 + bn_0) + 2^{-1} \times (am_1 + bn_1) \\ &\quad + \cdots + 2^{-t} \times (am_t + bn_t). \end{aligned} \tag{9}$$

If $am_i + bn_i$ $(i = 0, 1, \ldots, t)$ are calculated out, then T can be obtained by t shifts and t additions. Because m_i, n_i $(i = 0, 1, \ldots, t)$ are equal to either 0 or 1, there are totally 4 combinations with a and b. They are 0, a, b, $a + b$. So T can

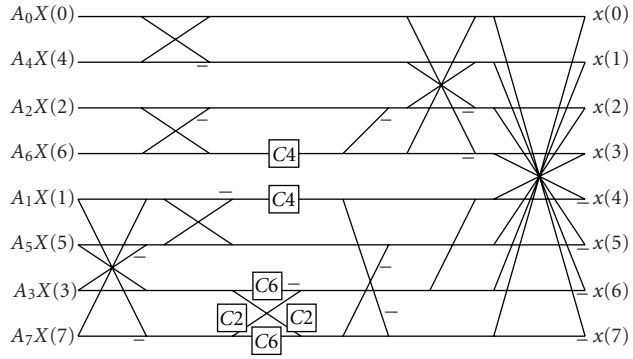

Figure 1: The flow graph of AAN $n = 8$.

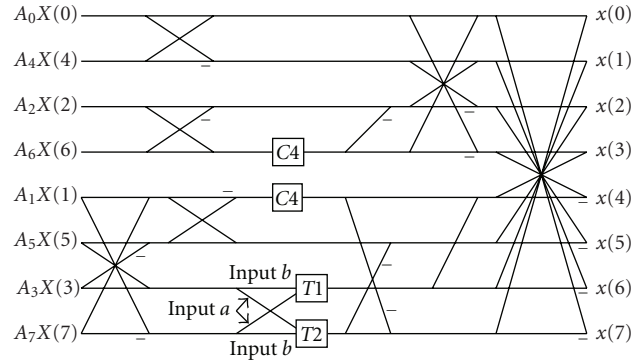

Figure 2: The revised flow graph of AAN $n = 8$.

actually be calculated by $t - s$ additions and $t - s$ shifts of a, b, and $a + b$, where s denotes the amount of $am_i + bn_i$ equal to 0. Since w_1 and w_2 are constants, the values of m_i, n_i $(i = 0, 1, \ldots, t)$ can be known in advance. So the optimal scheme of additions and shifts can be designed to decrease the amount of operations.

Most algorithms of the fast IDCT and DCT only deal with each separate multiplication using additions and shifts but our proposed method implements the linear combinations of multiplications via additions and shifts, which remarkably simplifies the computational complexity.

2.2. Reformation of AAN Algorithm Based on the General Method. The 1D IDCT flow graph of AAN fast algorithm [2], when $n = 8$, is shown in Figure 1, where the symbol ci denotes $\cos(i\pi/16)$, and the scale coefficients A_i $(i = 0, \ldots, 7)$ are defined as follows:

$$\begin{aligned}
A_0 &= \frac{1}{2\sqrt{2}} \approx 0.3535533906,\\
A_1 &= \frac{\cos(7\pi/16)}{2\sin(3\pi/8) - \sqrt{2}} \approx 0.4499881115,\\
A_2 &= \frac{\cos(\pi/8)}{\sqrt{2}} \approx 0.6532814824,\\
A_3 &= \frac{\cos(5\pi/16)}{\sqrt{2} + 2\cos(3\pi/8)} \approx 0.2548977895,\\
A_4 &= \frac{1}{2\sqrt{2}} \approx 0.3535533906,\\
A_5 &= \frac{\cos(3\pi/16)}{\sqrt{2} - 2\cos(3\pi/8)} \approx 1.2814577239,\\
A_6 &= \frac{\cos(3\pi/8)}{\sqrt{2}} \approx 0.2705980501,\\
A_7 &= \frac{\cos(\pi/16)}{\sqrt{2} + 2\sin(3\pi/8)} \approx 0.3006724435.
\end{aligned} \tag{10}$$

We reform the above algorithm flow graph based on our method described in Section 2. The new flow graph is shown in Figure 2.

In Figure 2, the butterfly computational structures are replaced with the formulas of $T1$ and $T2$. The formulas of $T1$ and $T2$, corresponding with w_1 and w_2, and optimal schemes of additions and shifts are given as follows:

$$\begin{aligned}
T1 &= a \cdot \cos\left(\frac{\pi}{8}\right) - b \cdot \cos\left(\frac{3\pi}{8}\right)\\
&= a \cdot w_1 - b \cdot w_2,
\end{aligned} \tag{11}$$

where $w_1 = \cos(\pi/8)$ and $w_2 = \cos(3\pi/8)$.

Optimal schemes of additions and shifts are

$$\begin{aligned}
x_1 &= a - (a \gg 3);\\
&\quad // \text{ B1 } (0.111)_2\\
x_2 &= (b \gg 3) + (b \gg 7);\\
&\quad // \text{ B2 } (0.0010001)_2\\
x &= x_1 + (a \gg 4) - (x_1 \gg 6) + (x_1 \gg 14);\\
&\quad // \text{ B1 } (0.11101100100000111)_2\\
x_3 &= x_2 - (x_2 \gg 10) + (b \gg 2);\\
&\quad // \text{ B2 } (0.01100001111101111)_2\\
x &= x - x_3;
\end{aligned} \tag{12}$$

8 additions and 8 shifts are used in the step to implement the computation of $T1$.

In the codes, x, x_1, x_2, and x_3 are all variables, and symbol "$\gg$" denotes the right-shift operator. The binary numbers B1 and B2 following the codes are the coefficients of Input a and Input b, respectively. The result of each code line can be expressed with B1, B2, a, and b. Now we explain this step in details.

Factors w_1 and w_2 can be expressed as binary numbers. Considering the precision and complexity of the computation, we choose 2^{17} as the denominators, then

$$\begin{aligned}
w_1 &= \cos\left(\frac{\pi}{8}\right) \approx \frac{121095}{131072}\\
&= (0.11101100100000111)_2,\\
w_2 &= \cos\left(\frac{3\pi}{8}\right) \approx \frac{50159}{131072}\\
&= (0.01100001111101111)_2.
\end{aligned} \tag{13}$$

When Input a is right-shifted r bits, the value is equal to $2^{-r} \cdot a$. So the code "$x_1 = a - (a \gg 3)$;" can be expressed by mathematic expression as follows:

$$x_1 = a - 2^{-3} \cdot a = (1 - 2^{-3}) \cdot a = (0.111)_2 \cdot a. \tag{14}$$

So the binary number B1 following the code as the coefficient of Input a is equal to $(0.111)_2$.

In the same way, the code "$x_2 = (b \gg 3) + (b \gg 7)$;" means

$$\begin{aligned} x_2 &= 2^{-3} \cdot b + 2^{-7} \cdot b \\ &= \left(2^{-3} + 2^{-7}\right) \cdot b \\ &= (0.0010001)_2 \cdot b. \end{aligned} \tag{15}$$

The binary number B2 is equal to $(0.0010001)_2$.

Then these codes implement the computation of $T1$:

$$\begin{aligned} x &= (0.11101100100000111)_2 \cdot a \\ &\quad - (0.01100001111101111)_2 \cdot b \\ &= w_1 \cdot a - w_2 \cdot b. \end{aligned} \tag{16}$$

With the similar method, we implement the computations of $T2$ and $\sqrt{2}/2$:

$$\begin{aligned} T2 &= a \times \cos\left(\frac{3\pi}{8}\right) + b \times \cos\left(\frac{\pi}{8}\right) \\ &= a \times w_1 + b \times w_2, \end{aligned} \tag{17}$$

where $w_1 = \cos(3\pi/8)$ and $w_2 = \cos(\pi/8)$.

Optimal schemes of additions and shifts are

$$\begin{aligned} &x_1 = b - (b \gg 3); \\ &\qquad // \text{ B2 } (0.111)_2 \\ &x_2 = (a \gg 3) + (a \gg 7); \\ &\qquad // \text{ B1 } (0.0010001)_2 \\ &x = x_1 + (b \gg 4) - (x_1 \gg 6) + (x_1 \gg 14); \\ &\qquad // \text{ B2 } (0.11101100100000111)_2 \\ &x_3 = x_2 - (x_2 \gg 10) + (a \gg 2); \\ &\qquad // \text{ B1 } (0.01100001111101111)_2 \\ &x = x + x_3; \end{aligned} \tag{18}$$

8 additions and 8 shifts are used in the step to implement the computation of $T2$:

$$\frac{\sqrt{2}}{2} \approx \frac{46341}{65536} = (0.1011010100000101)_2. \tag{19}$$

Optimal schemes of additions and shifts are

$$\begin{aligned} &x_1 = a + (a \gg 2); && // \ (1.01)_2 \\ &x_2 = x_1 \gg 2; && // \ (0.0101)_2 \\ &x_3 = a - x_2; && // \ (0.1011)_2 \\ &x_4 = x_1 + (x_2 \gg 6); && // \ (1.0100000101)_2 \\ &x = x_3 + (x_4 \gg 6); && // \ (0.1011010100000101)_2. \end{aligned} \tag{20}$$

In these codes, x, x_1, x_2, and x_3 are all variables, a is the input, and the output is approximately equal to $\sqrt{2}/2 \cdot a$. 4 additions and 4 shifts are used to implement the computation of each$\sqrt{2}/2$, and 8 additions and 8 shifts are needed to implement overall computations of $\sqrt{2}/2$ in the 1D IDCT computation. The computational complexity of 1D IDCT is tabulated in Table 1.

Table 1: The statistics of computational complexity.

	Additions	Shifts
Outside	26	0
$T1$	8	8
$T2$	8	8
$\sqrt{2}/2$	8	8
Total 1D	26 + 8 + 8 + 8 = 50	8 + 8 + 8 = 24

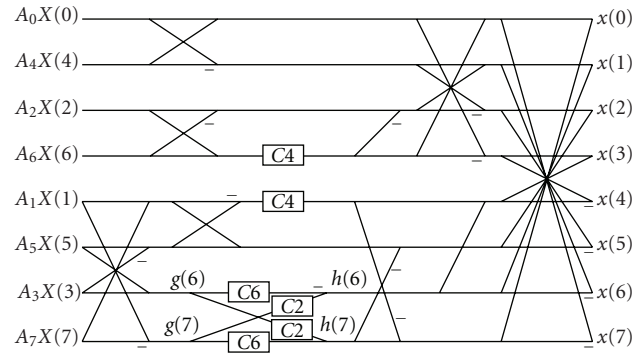

Figure 3: The revised flow graph of AAN based on the character, $n = 8$.

Table 2: The statistics of computational complexity.

	Additions	Shifts
Outside	28	0
$T1$	5	6
$T2$	5	6
$\sqrt{2}/2$	8	8
Total 1D	28 + 5 + 5 + 8 = 46	6 + 6 + 8 = 20

For the 1D IDCT, the total number of additions and shifts is 50 and 24, respectively.

2.3. Revision Based on the Characteristics of AAN Algorithm. Obviously, Input a and Input b in Figure 2 are computed twice, in order to reduce the redundancy, another algorithm is presented in Figure 3.

Details of the implementation of Figure 3 are presented as follows

$$\begin{aligned} h(6) &= g(7) \times \cos\left(\frac{\pi}{8}\right) - g(6) \times \cos\left(\frac{3\pi}{8}\right) \\ &= t_d - t_a, \\ h(7) &= g(6) \times \cos\left(\frac{\pi}{8}\right) + g(7) \times \cos\left(\frac{3\pi}{8}\right) \\ &= t_b + t_c, \end{aligned} \tag{21}$$

where $t_a = g(6) \times \cos(3\pi/8)$, $t_b = g(6) \times \cos(\pi/8)$, $t_c = g(7) \times \cos(3\pi/8)$, and $t_d = g(7) \times \cos(\pi/8)$. We also deal with the computations of $h(6)$ and $h(7)$ with the similar method introduced in Section 2.2 and choose 2^{17} as the denominators again.

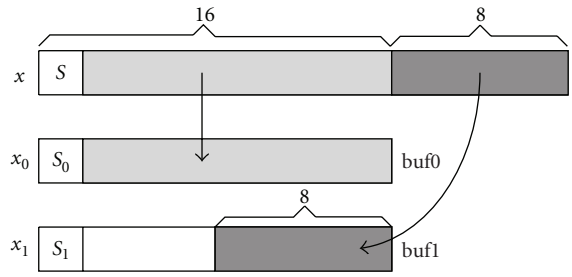

Figure 4: The standard storage data structure.

Optimal schemes of additions and shifts are

$$\begin{aligned}
&t_1 = g(6) - (g(6) \gg 4); && // (0.1111)_2\\
&t_2 = t_1 + (g(6) \gg 3); && // (1.0001)_2\\
&t_3 = t_1 + (t_2 \gg 10); && // (0.11110000010001)_2\\
&t_a = (g(6) \gg 1) - (t_3 \gg 3); && // (0.01100001111101111)_2\\
&t_b = t_3 - (t_1 \gg 6); && // (0.11101100100001)_2\\
&t_1 = g(7) - (g(7) \gg 4); && // (0.1111)_2\\
&t_2 = t_1 + (g(7) \gg 3); && // (1.0001)_2\\
&t_2 = t_1 + (t_2 \gg 10); && // (0.11110000010001)_2\\
&t_c = (g(7) \gg 1) - (t_3 \gg 3); && // (0.01100001111101111)_2\\
&t_d = t_3 - (t_1 \gg 6); && // (0.11101100100001)_2\\
&h(6) = t_d - t_a;\\
&h(7) = t_b + t_c.
\end{aligned} \tag{22}$$

In order to reduce the computational complexity, we let

$$\begin{aligned}
w_1 &= \cos\left(\frac{\pi}{8}\right) \approx \frac{121096}{131072}\\
&= (0.11101100100001)_2
\end{aligned} \tag{23}$$

instead of

$$\begin{aligned}
w_1 &= \cos\left(\frac{\pi}{8}\right) \approx \frac{121095}{131072}\\
&= (0.11101100100000111)_2.
\end{aligned} \tag{24}$$

The computational complexity of the improved method is tabulated in Table 2.

For the 1D IDCT, the total number of additions and shifts is 46 and 20, respectively.

3. Implementation of 2D IDCT

In order to implement an 8×8 2D IDCT defined in (3), we decompose it into a cascade of 1D IDCTs which are applied to each row and column in the 8×8 IDCT coefficient matrix. The algorithm of 1D IDCT has been discussed in Section 2. In this section, we will focus on the scale coefficients A_i $(i = 0, \ldots, 7)$ and the matrices derived from them which remarkably affect the computational precision of the algorithm.

3.1. Choice of Coefficient Matrices and Compensation Matrices. To ensure the precision of the transform, we use a coefficient matrix and a compensation matrix to scale the input $X(n, m)$ $(n, m = 0, \ldots, 7)$ in the preprocessing. The details are given as follows.

p_1 and p_2 are defined as scale factors, and $\mathbf{coef}[i][j]$ $(i, j = 0, 1, \ldots, 7)$ denotes the original floating-point matrix. $\mathbf{coef}[i][j]$ is defined as

$$\mathbf{coef}[i][j] = A_i \times A_j. \tag{25}$$

Because the input data are multiplied by floating-point matrix $\mathbf{coef}[i][j]$, we just scale the floating-point matrix $\mathbf{coef}[i][j]$ by 2^{p_1} to obtain integer coefficient matrix $\mathbf{coef0}[i][j]$ which is suitable for fixed-point computation. Considering rounding of fixed-point matrix, matrix $\mathbf{coef0}[i][j]$ is presented as

$$\begin{aligned}
\mathbf{coef}[i][j] &= \mathbf{coef}[i][j] \times (1 \ll P_1),\\
\mathbf{coef0}[i][j] &= (\text{int})(\mathbf{coef}[i][j] + 0.5).
\end{aligned} \tag{26}$$

In order to improve the computational precision, we expect p_1 as greater as possible. However, the value of p_1 is limited by the bit width of register. So we use the compensation matrix $\mathbf{coef1}[i][j]$ to improve the precision of computations. The compensation matrix is obtained as

$$\begin{aligned}
\mathbf{coef}[i][j] &= \mathbf{coef}[i][j] - \mathbf{coef0}[i][j],\\
\mathbf{coef1}[i][j] &= (\text{int})(\mathbf{coef}[i][j] \times (1 \ll P_2) + 0.5).
\end{aligned} \tag{27}$$

Since the compensation matrix $\mathbf{coef1}[i][j]$ stores p_2-bit data information, in some extent, the introduction of $\mathbf{coef1}[i][j]$ improves p_2-bit precision of the computations.

Matrix $\mathbf{block}[i][j]$ $(i, j = 0, 1, \ldots, 7)$ is defined as an 8×8 coefficient matrix. Then the preprocessing can be expressed with the compensation matrix $\mathbf{coef1}[i][j]$ as follows:

$$\begin{aligned}
\mathbf{block}[i][j] = \mathbf{block}[i][j] &\times \mathbf{coef0}[i][j]\\
&+ ((\mathbf{block}[i][j] \times \mathbf{coef1}[i][j]) \gg P_2).
\end{aligned} \tag{28}$$

Considering proper rounding at the final stage of the transform, we add 2^{p_1-1} to all output data and right-shift them by p_1 bits. Observing Figure 3 the flow graph of 1D IDCT algorithm, we find that if we add 2^t to $X(0)$, then all output $x(k, l)$ $(k, l = 0, \ldots, 7)$ are all added by 2^t. So we just need to add a bias 2^{p_1-1} to DC term $X(0, 0)$:

$$\mathbf{block}[0][0] = \mathbf{block}[0][0] + (1 \ll (P_1 - 1)), \tag{29}$$

and simply right-shift all output $x(k, l)$ $(k, l = 0, \ldots, 7)$ by p_1 bits for rounding of the 2D IDCT.

3.2. Implementation for 32-Bit Hardware. For 8-bit DCT coefficients, corresponding IDCT coefficients are at most 11-bit data. Due to the additions in the flow-graph, for 32-bit hardware implementation we let $p_1 = 18$, $p_2 = 3$, so we get the coefficient matrix and compensation matrix as follows:

$$\mathbf{coef0} = \begin{bmatrix} 32768 & 41706 & 60547 & 23624 & 32768 & 118768 & 25080 & 27867 \\ 41706 & 53081 & 77062 & 30068 & 41706 & 151163 & 31920 & 35468 \\ 60547 & 77062 & 111877 & 43652 & 60547 & 219455 & 46341 & 51491 \\ 23624 & 30068 & 43652 & 17032 & 23624 & 85627 & 18081 & 20091 \\ 32768 & 41706 & 60547 & 23624 & 32768 & 118768 & 25080 & 27867 \\ 118768 & 151163 & 219455 & 85627 & 118768 & 430476 & 90901 & 101004 \\ 25080 & 31920 & 46341 & 18081 & 25080 & 90901 & 19195 & 21328 \\ 27867 & 35468 & 51491 & 20091 & 27867 & 101004 & 21328 & 23699 \end{bmatrix},$$

$$\mathbf{coef1} = \begin{bmatrix} 0 & -2 & 3 & 3 & 0 & 0 & -4 & -1 \\ -2 & 3 & 2 & 2 & -2 & 0 & 2 & -2 \\ 3 & 2 & 0 & 2 & 3 & 0 & 0 & 2 \\ 3 & 2 & 2 & 2 & 3 & -1 & 2 & -1 \\ 0 & -2 & 3 & 3 & 0 & 0 & -4 & -1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ -4 & 2 & 0 & 2 & -4 & 0 & 1 & 3 \\ -1 & -2 & 2 & -1 & -1 & 0 & 3 & -2 \end{bmatrix}. \tag{30}$$

After preprocessing, we process 64 coefficients according to the flow graph shown in Figure 3 by row-column way. Finally, we right-shift the output back to the original scale as

$$\mathbf{block}[i][j] = (\mathbf{block}[i][j]) \gg 18. \tag{31}$$

The multiplications of the coefficient matrix and the compensation matrix in (28) can also be implemented with the method of shifts and additions.

There are positive and negative elements in the compensation matrix $\mathbf{coef1}[i][j]$. The purpose of this kind of design is to reduce the absolute values of these elements and decrease the computational complexity. Due to the limited space, the details of optimal schemes of additions and shifts for all elements in the matrices are omitted.

4. Implementations of 2D IDCT for 24-Bit and 16-Bit Hardware

In Section 3, we implemented the 2D IDCT for 32-bit hardware. However in some cases it cannot be applied. Take mobile devices for example. The bit width of these devices is not enough to complete a 32-bit computation. So we present the implementations of IDCTs for 24-bit and 16-bit hardware in this section.

4.1. Implementation for 24-Bit Hardware. To implement the above algorithm in 24-bit frame with the same idea, let $p_1 = 11$ and $p_2 = 5$, then get the coefficient matrix and compensation matrix:

$$\mathbf{coef0} = \begin{bmatrix} 256 & 326 & 473 & 185 & 256 & 928 & 196 & 218 \\ 326 & 415 & 602 & 235 & 326 & 1181 & 249 & 277 \\ 473 & 602 & 874 & 341 & 473 & 1714 & 362 & 402 \\ 185 & 235 & 341 & 133 & 185 & 669 & 141 & 157 \\ 256 & 326 & 473 & 185 & 256 & 928 & 196 & 218 \\ 928 & 1181 & 1714 & 669 & 928 & 3363 & 710 & 789 \\ 196 & 249 & 362 & 141 & 196 & 710 & 150 & 167 \\ 218 & 277 & 402 & 157 & 218 & 789 & 167 & 185 \end{bmatrix},$$

$$\mathbf{coef1} = \begin{bmatrix} 0 & -6 & 1 & -14 & 0 & -4 & -2 & -9 \\ -6 & -10 & 2 & -3 & -6 & -1 & 12 & 3 \\ 1 & 2 & 1 & 1 & 1 & 16 & 1 & 9 \\ -14 & -3 & 1 & 2 & -14 & -1 & 8 & -1 \\ 0 & -6 & 1 & -14 & 0 & -4 & -2 & -9 \\ -4 & -1 & 16 & -1 & 4 & 3 & 5 & 3 \\ -2 & 12 & 1 & 8 & -2 & 5 & -1 & -12 \\ -9 & 3 & 9 & -1 & -9 & 3 & -12 & 5 \end{bmatrix}. \tag{32}$$

After preprocessing, we process 64 coefficients according to the flow graph shown in Figure 3 by row-column way. Finally, we right-shift the output back to the original scale as

$$\mathbf{block}[i][j] = \mathbf{block}[i][j] \gg 11. \tag{33}$$

The computations of 1D IDCTs for 24-bit hardware and 32-bit hardware are nearly the same. The only difference is just the bit width.

4.2. Implementation for 16-Bit Hardware. Considering that the IDCT coefficients are at most 11-bit data, it is impossible to implement the IDCT within 16-bit width to satisfy the practical requirement of precision just through modifying the scale factors p_1 and p_2.

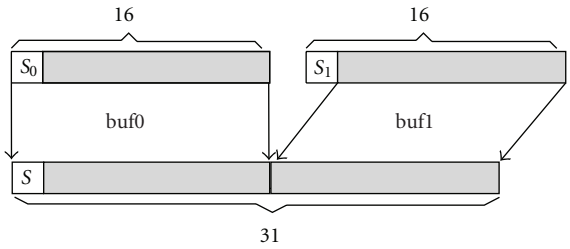

Figure 5: The storage data structure for preprocessing.

In order to complete calculations of the IDCT according to Figure 3 within 16-bit width, we use a combination of two bytes as a unit to deal with calculations. We denote two 16-bit buffers as buf0 and buf1 to store the high 16 bits and low 8 bits of the original 24-bit datum, respectively, which can be expressed formally as follows.

Let the original 24-bit datum as x, and the data stored in buf0 and buf1 are x_0 and x_1, respectively. Then

$$x = x_0 \cdot 2^8 + x_1. \tag{34}$$

In order to use unique data structure to express data, we define the standard storage format, in which x_0 and x_1 must satisfy the equations as follows:

$$-32768 \le x_0 \le 32767, \qquad 0 \le x_1 \le 255. \tag{35}$$

This process also can be demonstrated by Figure 4.

In Figure 4, S, S_0, and S_1 are all sign bits, and $S_0 = S$ and $S_1 = 0$ when data are stored in the standard storage format.

Due to the limit of 16-bit width, we cannot implement preprocessing according to (28). We also use a combination of two bytes as a unit which contains 30 data bits and 1 sign bit, and the method of additions and shifts to deal with calculations in preprocessing. The data structure is showed in Figure 5.

Because there are 30 data bits, we do not need the compensation matrix but we use a new coefficient matrix $\mathbf{coef\_16}[i][j]$, which is defined as follows:

$$\begin{aligned} \mathbf{coef}\,[i]\,[j] &= \mathbf{coef}\,[i]\,[j] \times (1 \ll P), \\ \mathbf{coef\_16}\,[i]\,[j] &= (\text{int})\,(\mathbf{coef}\,[i]\,[j] + 0.5), \end{aligned} \tag{36}$$

where $p = p_1 + p_2 = 11 + 5 = 16$. The coefficient matrix is presented as follows:

$\mathbf{coef\_16} =$

$$\begin{bmatrix} 8192 & 10426 & 15137 & 5906 & 8192 & 29692 & 6270 & 6967 \\ 10426 & 13270 & 19266 & 7517 & 10426 & 37791 & 7980 & 8867 \\ 15137 & 19266 & 27969 & 10913 & 15137 & 54864 & 11585 & 12873 \\ 5906 & 7517 & 10913 & 4258 & 5906 & 21407 & 4520 & 5023 \\ 8192 & 10426 & 15137 & 5906 & 8192 & 29692 & 6270 & 6967 \\ 29692 & 37791 & 54864 & 21407 & 29692 & 107619 & 22725 & 25251 \\ 6270 & 7980 & 11585 & 4520 & 6270 & 22725 & 4799 & 5332 \\ 6967 & 8867 & 12873 & 5023 & 6967 & 25251 & 5332 & 5925 \end{bmatrix}. \tag{37}$$

Then the preprocessing can be expressed as follows with the new coefficient matrix $\mathbf{coef\_16}[i][j]$:

$$\begin{aligned} \mathbf{block}\,[i]\,[j] &= \mathbf{block}\,[i]\,[j] \times \mathbf{coef\_16}\,[i]\,[j], \\ \mathbf{block}\,[i]\,[j] &= \mathbf{block}\,[i]\,[j] \gg P_2. \end{aligned} \tag{38}$$

After preprocessing, we transform the data into standard format mentioned above and complete the computation of IDCT.

As discussed above, in fact the algorithm for 16-bit hardware is theoretically the same as for 24-bit hardware; there are just some differences in the implementations. In other words, we complete the 24-bit computations within 16-bit width, so that we could gain the same precisions.

5. Performances of Our Proposed Algorithms

In order to evaluate the performances of the novel algorithm, we test it referring to IEEE 1180 [13] and ISO/IEC 23002-1 [14] specifications. We use the specified pseudorandom input matrices X_i (Q denotes the testing quantity, $i = 1, 2, \ldots, Q$), test out our algorithm, and investigate five metrics:

peak pixel err:

$$p = \max_{k,l,i} \left| \hat{x}_i\,(k,l) - x_{0i}\,(k,l) \right|,$$

peak mean square err:

$$e = \max_{k,l} \left[\frac{1}{Q} \sum_{i=0}^{Q-1} \left(\hat{x}_i(k,l) - x_{0i}(k,l) \right)^2 \right],$$

over mean square err:

$$n = \frac{1}{64} \sum_{k=0}^{7} \sum_{l=0}^{7} \left[\frac{1}{Q} \sum_{i=0}^{Q-1} \left(\hat{x}_i(k,l) - x_{0i}(k,l) \right)^2 \right],$$

peak mean err:

$$d = \max_{k,l} \left[\frac{1}{Q} \sum_{i=0}^{Q-1} \left(\hat{x}_i\,(k,l) - x_{0i}\,(k,l) \right) \right],$$

over mean err:

$$m = \frac{1}{64} \sum_{k=0}^{7} \sum_{l=0}^{7} \left[\frac{1}{Q} \sum_{i=0}^{Q-1} \left(\hat{x}_i\,(k,l) - x_{0i}\,(k,l) \right) \right]. \tag{39}$$

Here, $\hat{x}_i(k,l)$ $(k,l = 0, \ldots, 7)$ denote the reconstructed pixel values of the specified pseudorandom input matrices X_i, and $x_{0i}(k,l)$ $(k,l = 0, \ldots, 7)$ denote reference values correspondingly.

The IEEE 1180 indicates that these metrics must satisfy these accuracy requirements: $p \le 1$, $e \le 0.06$, $n \le 0.02$, $d \le 0.015$, and $m \le 0.0015$. The performances of our proposed algorithm are presented in Tables 3 and 4. Because computational precisions of the algorithms for 24-bit and 16-bit hardware are the same, we just give one table to show the results.

From the previous tables, it is obvious that the precisions of IDCT for 24-bit and 16-bit hardware are lower than

Table 3: IDCT precision performance for 32-bit hardware.

Q	L, H	Negation	p(ppe) ≤ 1	e(pmse) ≤ 0.06	n(omse) ≤ 0.02	d(pme) ≤ 0.015	m(ome) ≤ 0.0015
10 000	5, 5	No	1.000000	0.000300	0.000064	0.000300	0.000064
10 000	5, 5	Yes	1.000000	0.000400	0.000070	0.000400	0.000070
10 000	256, 255	No	1.000000	0.000800	0.000252	0.000400	0.000061
10 000	256, 255	Yes	1.000000	0.000800	0.000244	0.000500	0.000094
10 000	300, 300	No	1.000000	0.000800	0.000300	0.000500	0.000087
10 000	300, 300	Yes	1.000000	0.000700	0.000278	0.000400	0.000038
10 000	384, 383	No	1.000000	0.000700	0.000234	0.000400	0.000022
10 000	384, 383	Yes	1.000000	0.000800	0.000238	0.000400	0.000034
10 000	512, 511	No	1.000000	0.000600	0.000231	0.000400	0.000022
10 000	512, 511	Yes	1.000000	0.000600	0.000241	0.000400	0.000031
1 000 000	5, 5	No	1.000000	0.000105	0.000064	0.000105	0.000064
1 000 000	5, 5	Yes	1.000000	0.000115	0.000067	0.000115	0.000067
1 000 000	256, 255	No	1.000000	0.000332	0.000256	0.000116	0.000062
1 000 000	256, 255	Yes	1.000000	0.000347	0.000255	0.000115	0.000064
1 000 000	300, 300	No	1.000000	0.000348	0.000252	0.000107	0.000054
1 000 000	300, 300	Yes	1.000000	0.000357	0.000253	0.000118	0.000056
1 000 000	384, 383	No	1.000000	0.000331	0.000241	0.000080	0.000042
1 000 000	384, 383	Yes	1.000000	0.000324	0.000239	0.000095	0.000042
1 000 000	512, 511	No	1.000000	0.000307	0.000236	0.000081	0.000034
1 000 000	512, 511	Yes	1.000000	0.000310	0.000232	0.000061	0.000032

Table 4: IDCT precision performance for 24-bit or 1-bit hardware.

Q	L, H	Negation	p(ppe) ≤ 1	e(pmse) ≤ 0.06	n(omse) ≤ 0.02	d(pme) ≤ 0.015	m(ome) ≤ 0.0015
10 000	5, 5	Yes	1.000000	0.001300	0.000558	0.001000	0.000098
10 000	5, 5	No	1.000000	0.001500	0.000603	0.001500	0.000184
10 000	256, 255	Yes	1.000000	0.013600	0.008787	0.002700	0.000066
10 000	256, 255	No	1.000000	0.013400	0.008839	0.002300	0.000114
10 000	300, 300	Yes	1.000000	0.014700	0.008817	0.002400	0.000061
10 000	300, 300	No	1.000000	0.014700	0.008787	0.002100	0.000244
10 000	384, 383	Yes	1.000000	0.014600	0.008742	0.003400	0.000092
10 000	384, 383	No	1.000000	0.014600	0.008763	0.002100	0.000003
10 000	512, 511	Yes	1.000000	0.011800	0.008511	0.002500	0.000005
10 000	512, 511	No	1.000000	0.012900	0.008586	0.001800	0.000052
10 00000	5, 5	Yes	1.000000	0.001242	0.000559	0.001242	0.000139
10 00000	5, 5	No	1.000000	0.001292	0.000561	0.001292	0.000141
10 00000	256, 255	Yes	1.000000	0.012672	0.008839	0.001420	0.000125
10 00000	256, 255	No	1.000000	0.012660	0.008839	0.001189	0.000119
10 00000	300, 300	Yes	1.000000	0.012367	0.008761	0.001123	0.000103
10 00000	300, 300	No	1.000000	0.012380	0.008753	0.001069	0.000094
10 00000	384, 383	Yes	1.000000	0.012075	0.008641	0.000921	0.000070
10 00000	384, 383	No	1.000000	0.012136	0.008649	0.000771	0.000068
10 00000	512, 511	Yes	1.000000	0.012110	0.008620	0.000875	0.000020
10 00000	512, 511	No	1.000000	0.012138	0.008620	0.000663	0.000055

the precision of IDCT for 32-bit hardware but still meet corresponding specifications very well [11, 12].

We estimate the implementation costs of our algorithm in Table 5 in terms of 1D and 2D computational complexities. The 1D computational complexity means the complexity of executing our 8-point IDCT algorithm, and the 2D computational complexity includes the computations of 16 iterations of 1D IDCTs, an addition for rounding and the right shifts at the end of the transform.

In many image and video Codecs, it is also possible to simply merge factors involved in IDCT scaling with the factors used by the corresponding inverse-quantization

Table 5: Computational complexities of 1D and 2D IDCT.

	Additions	Shifts
1D IDCT	46	20
2D IDCT	737	384

process. In such cases, scaling can be executed without taking any extra resources. So we do not take account of these computational complexities in Table 5.

6. Conclusions

In this paper, we propose a new general method to compute the fast IDCT, which can be applied to most existing IDCT algorithms with butterfly computational structures. We also introduce a specific algorithm derived from AAN algorithm. Considering characteristics of the AAN algorithm, coefficient matrices and compensation matrices are brought into the algorithm to improve its precision. Through varying the scale coefficients p_1 and p_2, we can modify the precision to meet the different requirements of hardware, such as 32-bit hardware and 24-bit hardware. However, in order to implement our algorithm using 16-bit hardware with satisfied precision, we have to revise our design which includes a new coefficient matrix, a new kind of data structure, and some new methods of data manipulation.

The new IDCT algorithm achieves a good compromise between the precision and computational complexity. The idea of optimizing the IDCT algorithm can also be extended to other similar fast algorithms.

Acknowledgments

This work was supported partly by NSFC under Grants 60672060, the Research Fund for the Doctoral Program of Higher Education, and HiSilicon Technologies Co. Ltd.

References

[1] B. Lee, "A new algorithm to compute the discrete cosine transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 6, pp. 1243–1245, 1984.

[2] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for images," *Transactions of the IEICE*, vol. 71, no. 11, pp. 1095–1097, 1988.

[3] C. Loeffler, A. Ligtenberg, and G. S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '89)*, vol. 2, pp. 988–991, Glasgow, UK, May 1989.

[4] A. Elnaggar and H. M. Alnuweiri, "A new multidimensional recursive architecture for computing the discrete cosine transform," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 1, pp. 113–119, 2000.

[5] L.-P. Chau and W.-C. Siu, "Recursive algorithm for the realization of the discrete cosine transform," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS '00)*, vol. 5, pp. 529–532, Geneva, Switzerland, May 2000.

[6] C.-H. Chen, B.-D. Liu, J.-F. Yang, and J.-L. Wang, "Efficient recursive structures for forward and inverse discrete cosine transform," *IEEE Transactions on Signal Processing*, vol. 52, no. 9, pp. 2665–2669, 2004.

[7] C.-H. Chen, B.-D. Liu, and J.-F. Yang, "Condensed recursive structures for computing multidimensional DCT/IDCT with arbitrary length," *IEEE Transactions on Circuits and Systems I*, vol. 52, no. 9, pp. 1819–1831, 2005.

[8] J. Lee, N. Vijaykrishnan, and M. J. Irwin, "Inverse discrete cosine transform architecture exploiting sparseness and symmetry properties," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 5, pp. 655–662, 2006.

[9] J.-S. Chiang, Y.-F. Chiu, and T.-H. Chang, "A high throughput 2-dimensional DCT/IDCT architecture for real-time image and video system," in *Proceedings of the 8th IEEE International Conference on Electronics, Circuits and Systems (ICECS '01)*, vol. 2, pp. 867–870, Msida, Malta, September 2001.

[10] R. Kutka, "Fast computation of DCT by statistic adapted look-up tables," in *Proceedings of the IEEE International Conference on Multimedia and Expo (ICME '02)*, vol. 1, pp. 781–784, Lausanne, Switzerland, August 2002.

[11] T. D. Tran, "The binDCT: fast multiplierless approximation of the DCT," *IEEE Signal Processing Letters*, vol. 7, no. 6, pp. 141–144, 2000.

[12] J. Liang and T. D. Tran, "Fast multiplierless approximations of the DCT with the lifting scheme," *IEEE Transactions on Signal Processing*, vol. 49, no. 12, pp. 3032–3044, 2001.

[13] CAS Standards Committee of the IEEE Circuits and Systems Society, "IEEE standard specifications for the implementations of 8×8 inversediscrete cosine transform," March 1991.

[14] ISO/IEC 23002-1:2006, JTC1/SC29/WG11 N7815, "Information technology—MPEG video technologies—part 1: accuracy requirements for implementation of integer-output 8×8 inverse discrete cosine transform".