Hindawi Publishing Corporation
VLSI Design
Volume 2012, Article ID 752024, 13 pages
doi:10.1155/2012/752024

Research Article

N Point DCT VLSI Architecture for Emerging HEVC Standard

Ashfaq Ahmed, Muhammad Usman Shahid, and Ata ur Rehman

Department of Electronics & Telecommunication, Politecnico di Torino, 10129 Torino, Italy

Correspondence should be addressed to Ashfaq Ahmed, ashfaq.ahmed@polito.it

Received 21 December 2011; Revised 13 April 2012; Accepted 6 May 2012

Academic Editor: Andrey Norkin

This work presents a flexible VLSI architecture to compute the N-point DCT. Since HEVC supports different block sizes for the computation of the DCT, that is, 4×4 up to 32×32, the design of a flexible architecture to support them helps reducing the area overhead of hardware implementations. The hardware proposed in this work is partially folded to save area and to get speed for large video sequences sizes. The proposed architecture relies on the decomposition of the DCT matrices into sparse submatrices in order to reduce the multiplications. Finally, multiplications are completely eliminated using the lifting scheme. The proposed architecture sustains real-time processing of 1080P HD video codec running at 150 MHz.

1. Introduction

As the technology is evolving day by day, the size of hardware is shrinking with an increase of the storage capacity. High-end video applications have become very demanding in our daily life activities, for example, watching movies, video conferencing, creating and saving videos using high definition video cameras, and so forth. A single device can support all the multimedia applications which seemed to be dreaming before, for example, new high-end mobile phones and smart phones. As a consequence, new highly efficient video coders are of paramount importance. However, high efficiency comes at the expense of computational complexity. As pointed out in [1, 2], several blocks of video codecs, including the transform stage [3], motion estimation and entropy coding [4], are responsible for this high complexity. As an example the discrete-cosine-transform (DCT), that is used in several standards for image and video compression, is a computation intensive operation. In particular, it requires a large number of additions and multiplications for direct implementation.

HEVC, the brand new and yet-to-release video coding standard, addresses high efficient video coding. One of the tools employed to improve coding efficiency is the DCT with different transform sizes. As an example, the 16-point DCT of HEVC is shown in [5]. In video compression, the DCT is widely used because it compacts the image energy at the low frequencies, making easy to discard the high frequency components. To meet the requirement of real-time processing, hardware implementations of 2-D DCT/inverse DCT (IDCT) are adopted, for example, [6]. The 2-D DCT/IDCT can be implemented with the 1-D DCT/IDCT and a transpose memory in a row-column decomposition manner. In the direct implementation of DCT, float-point multiplications have to be tackled, which cause precision problems in hardware. Hence, we propose a Walsh-Hadamard transform-based DCT implementation [7]. Then, inspired by the DCT factorizations proposed in [8, 9], we factorize the remaining rotations into simpler steps through the lifting scheme [10]. The resulting lifting scheme-based architecture, inspired by [11–13], is simplified, exploiting the techniques proposed in [9, 14] to achieve a multiplierless implementation. Other techniques can be employed to achieve multiplierless solutions, such as the ones proposed in [8, 15–18], but they are not discussed in this work. In this work, the proposed multisize DCT architecture supports all the block sizes of HEVC and is proposed for the real-time processing of 1080P HD video sequences.

The rest of this paper is organized as follows. Section 2 provides reviews of 2-D DCT transform. Section 3 shows the matrix decompositions for different DCT sizes. Section 4 presents the proposed hardware architecture. The VLSI implementation and the simulation results in Section 5. Finally, Section 6 concludes this paper.

2. Review of 2D Transform

According to [19], the DCT in the so-called II-form for an N-point block of samples $x = \{x_0, x_1, \ldots, x_{N-1}\}$ is obtained as

$$X_k = \sqrt{\frac{2}{N}}\epsilon_k \sum_{n=0}^{N-1} x(n) \cdot \cos\left[\frac{\pi(2n+1)k}{2N}\right], \quad (1)$$

where

$$\epsilon_k = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } k = 0, \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

and $k = 0, 1, 2, \ldots, N-1$. The same result expressed in (1) can be rewritten as the product of a matrix (C_N^{II}) for x as follws:

$$X = C_N^{II} \cdot x^t, \quad (3)$$

where $(\cdot)^t$ is the transposition operator. The DCT matrix can be expressed in terms of a reduced number of angles by exploiting the symmetry properties of the trigonometric functions. For 16 points, DCT matrix can be represented as

$$C_{16}^{II} = \frac{1}{2\sqrt{2}} \cdot \hat{C}_{16}^{II}, \quad (4)$$

where $\hat{C}_{16}^{II}$ is shown in (20), with

$$c_{p,q} = \cos\left(\frac{p \cdot \pi}{2^q}\right), \qquad s_{p,q} = \sin\left(\frac{p \cdot \pi}{2^q}\right), \quad (5)$$

where $4 \le 2^q \le 2N$ and p is an odd integer such that $p < 2^{q-2}$.

In [20], it is shown that every even-odd transform can be represented in terms of any other evenodd transform through a conversion matrix. In particular, in [7] it is shown that the DCT can be expressed in terms of the Walsh-Hadamard transform (WHT) [21], and the conversion matrix has a block diagonal structure. In [22], it is shown that WHT can be implemented with a fast algorithm based on a butterfly structure. Among the possible WHT matrix representations, the Walsh-ordered one is applied by deriving it from the corresponding Hadamard-ordered matrix. The N-order Hadamard matrix can be expressed as

$$H_N = \begin{bmatrix} H_{N/2} & H_{N/2} \\ H_{N/2} & -H_{N/2} \end{bmatrix}, \quad (6)$$

where $H_1 = 1$. The corresponding Walsh matrix W_N is obtained by applying a two step procedure [23] as follows:

(1) bit reverse the order of the rows of H_N,

(2) gray coding to the row indices.

Therefore, (4) can be written as

$$C_N^{II} = \frac{1}{\sqrt{N}} \cdot B_N \cdot T_N \cdot B_N \cdot W_N, \quad (7)$$

where B_N is the N-point bit reversal matrix, and

$$T_N = \begin{bmatrix} T_{N/2} & 0 \\ 0 & U_{N/2} \end{bmatrix} \quad (8)$$

is a block diagonal matrix with a recursive structure, $T_2 = I_2$ is the 2×2 identity matrix and

$$U_2 = \begin{bmatrix} c_{1,3} & s_{1,3} \\ -s_{1,3} & c_{1,3} \end{bmatrix}. \quad (9)$$

It is worth noting that $U_{N/2}$ can be factorized in terms of Givens rotations, where the Givens rotation matrix for a rotation angle θ is

$$\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}. \quad (10)$$

In particular $U_{N/2}$ can be decomposed in the product of permutation matrices $P_{N/2}$ and Givens rotation matrices $V_{N/2,q}$ with $3 \le q \le m$ and $m = \log_2(N) + 1$ as

$$U_{N/2} = P_{N/2} \cdot V_{N/2,m} \cdot \ldots \cdot V_{N/2,q} \cdot \ldots \cdot V_{N/2,3} \cdot P_{N/2}. \quad (11)$$

The permutation matrix $P_{N/2}$ is obtained by applying the following permutation:

$$\Phi_{N/2} = \begin{pmatrix} 0 & 1 & \cdots & \frac{N}{2} - 1 \\ \phi_{N/2}(0) & \phi_{N/2}(1) & \cdots & \phi_{N/2}\left(\frac{N}{2} - 1\right) \end{pmatrix} \quad (12)$$

to the rows or to the columns of $I_{N/2}$, the $N/2 \times N/2$ identity matrix. It is worth observing that $\phi_{N/2}(x)$ can be defined recursively as

$$\phi_{N/2}(x) = \begin{cases} 2\phi_{N/4}(x), & 0 \le x \le \frac{N}{4} - 1, \\ 2\phi_{N/4}(x) + 1, & \frac{N}{4} \le x \le N - 1, \end{cases} \quad (13)$$

where $\phi_2(0) = 0$ and $\phi_2(1) = 1$.

The Givens rotations matrices can be described as follows: $V_{N/2,m}$ contains $N/4$ Givens rotations disposed in $N/4$ concentric squares with the rotation angle increasing from the outer square to the inner one. The definition of each $c_{p,q}$ is given in (5). In the outermost circle $p = 1$, whereas the value of p increases from the outer square to the inner one. Since p is the multiplied value in the numerator of (5), the angle of the Givens rotations increases from outer to inner squares. The general structure of $V_{N/2,m}$ is shown in (15). In the next sections, the expression shown in (15) will be detailed for different values of N

$$V_{N/2,m} = \begin{pmatrix} c_{1,m} & 0 & 0 & \cdots & 0 & 0 & s_{1,m} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & c_{p,m} & \cdots & s_{p,m} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & 0 & -s_{p,m} & \cdots & c_{p,m} & 0 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s_{1,m} & 0 & 0 & \cdots & 0 & 0 & c_{1,m} \end{pmatrix}. \quad (14)$$

The remaining matrices $V_{N/2,q}$, for $q \geq 3$, are

$$V_{N/2,q} = \begin{pmatrix} V_{N/2^r,q} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & V_{N/2^r,q} \end{pmatrix}, \tag{15}$$

where $r = m - q + 1$ and $V_{2,3} = U_2$. Finally, each Givens rotation can be factorized into lifting steps by the means of the lifting scheme [10] as suggested in [9] as follows:

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} = \begin{pmatrix} 1 & \frac{1-\cos\theta}{\sin\theta} \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 \\ -\sin\theta & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & \frac{1-\cos\theta}{\sin\theta} \\ 0 & 1 \end{pmatrix}. \tag{16}$$

3. Matrix Decompositions for Different N

Matrices for different DCT are derived using the factorization presented in Section 2. In the following paragraphs factorizations for N ranging from 4 to 32 are explicitly shown.

3.1. 4×4 *DCT.* Equation (6) can be given as

$$H_4 = \begin{pmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{pmatrix}. \tag{17}$$

Equation (7) can be given as

$$C_4^{II} = \frac{1}{2} \cdot B_4 \cdot T_4 \cdot B_4 \cdot W_4, \tag{18}$$

where

$$T_4 = \begin{pmatrix} I_2 & 0 \\ 0 & U_2 \end{pmatrix}, \tag{19}$$

where U_2 is shown in (9) and I_2 is a 2×2 identity matrix. W_4 is a Walsh-Hadamard matrix which is calculated using (17)

$$\widehat{C}_{16}^{II} = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ c_{1,5} & c_{3,5} & c_{5,5} & c_{7,5} & s_{7,5} & s_{5,5} & s_{3,5} & s_{1,5} & -s_{1,5} & -s_{3,5} & -s_{5,5} & -s_{7,5} & -c_{7,5} & -c_{5,5} & -c_{3,5} & -c_{1,5} \\ c_{1,4} & c_{3,4} & s_{3,4} & s_{1,4} & -s_{1,4} & -s_{3,4} & -c_{3,4} & -c_{1,4} & -c_{1,4} & -c_{3,4} & -s_{3,4} & -s_{1,4} & s_{1,4} & s_{3,5} & c_{3,4} & c_{1,4} \\ c_{3,5} & s_{7,5} & s_{1,5} & -s_{5,5} & -c_{5,5} & -c_{1,5} & -c_{7,5} & -s_{3,5} & s_{3,5} & c_{7,5} & c_{1,5} & c_{5,5} & s_{5,5} & -s_{1,5} & -s_{7,5} & -c_{3,5} \\ c_{1,3} & s_{1,3} & -s_{1,3} & -c_{1,3} & -c_{1,3} & -s_{1,3} & s_{1,3} & c_{1,3} & c_{1,3} & s_{1,3} & -s_{1,3} & -c_{1,3} & -c_{1,3} & -s_{1,3} & s_{1,3} & c_{1,3} \\ c_{5,5} & s_{1,5} & -c_{7,5} & -c_{3,5} & -s_{3,5} & s_{7,5} & c_{1,5} & s_{5,5} & -s_{5,5} & -c_{1,5} & -s_{7,5} & s_{3,5} & c_{3,5} & c_{7,5} & -s_{1,5} & -c_{5,5} \\ c_{3,4} & -s_{1,4} & -c_{1,4} & -s_{3,4} & s_{3,4} & c_{1,4} & s_{1,4} & -c_{3,4} & -c_{3,4} & s_{1,4} & c_{1,4} & s_{3,4} & -s_{3,4} & -c_{1,4} & -s_{1,4} & c_{3,4} \\ c_{7,5} & -s_{5,5} & -c_{3,5} & -s_{1,5} & c_{1,5} & s_{3,5} & -c_{5,5} & -s_{7,5} & s_{7,5} & c_{5,5} & -s_{3,4} & -c_{1,5} & -s_{1,5} & c_{3,5} & s_{5,5} & -c_{7,5} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ s_{7,5} & -c_{5,5} & -s_{3,5} & c_{1,5} & -s_{1,5} & -c_{3,5} & s_{5,5} & c_{7,5} & -c_{7,5} & -s_{5,5} & c_{3,5} & s_{1,5} & -c_{1,5} & s_{3,5} & c_{5,5} & -s_{7,5} \\ s_{3,4} & -c_{1,4} & s_{1,4} & c_{3,4} & -c_{3,4} & -s_{1,4} & c_{1,4} & -s_{3,4} & -s_{3,4} & c_{1,4} & -s_{1,4} & -c_{3,4} & c_{3,4} & s_{1,4} & -c_{1,4} & s_{3,4} \\ s_{5,5} & -c_{1,5} & s_{7,5} & s_{3,5} & -c_{3,5} & c_{7,5} & s_{1,5} & -c_{5,5} & c_{5,5} & -s_{1,5} & -c_{7,5} & c_{3,5} & -s_{3,5} & -s_{7,5} & c_{1,5} & -s_{5,5} \\ s_{1,3} & -c_{1,3} & c_{1,3} & -s_{1,3} & -s_{1,3} & c_{1,3} & -c_{1,3} & s_{1,3} & s_{1,3} & -c_{1,3} & c_{1,3} & -s_{1,3} & -s_{1,3} & c_{1,3} & -c_{1,3} & s_{1,3} \\ s_{3,5} & -c_{7,5} & c_{1,5} & -c_{5,5} & s_{5,5} & s_{1,5} & -s_{7,5} & c_{3,5} & -c_{3,5} & s_{7,5} & -s_{1,5} & -s_{5,5} & c_{5,5} & -s_{1,5} & c_{7,5} & -s_{3,5} \\ s_{1,4} & -s_{3,4} & c_{3,4} & -c_{1,4} & c_{1,4} & -c_{3,4} & s_{3,4} & -s_{1,4} & -s_{1,4} & s_{3,4} & -c_{3,4} & c_{1,4} & -c_{1,4} & c_{3,4} & -s_{3,4} & s_{1,4} \\ s_{1,5} & -s_{3,5} & s_{5,5} & -s_{7,5} & c_{7,5} & -c_{5,5} & c_{3,5} & -c_{1,5} & c_{1,5} & -c_{3,5} & c_{5,5} & -c_{7,5} & s_{7,5} & -s_{5,5} & s_{3,5} & -s_{1,5} \end{bmatrix} \tag{20}$$

3.2. 8×8 *DCT.* Equation (6) can be given as

$$H_8 = \begin{pmatrix} H_4 & H_4 \\ H_4 & H_4 \end{pmatrix}. \tag{21}$$

Equation (7) can be given as

$$C_8^{II} = \frac{1}{2\sqrt{2}} \cdot B_8 \cdot T_8 \cdot B_8 \cdot W_8, \tag{22}$$

where

$$T_8 = \begin{pmatrix} I_2 & 0 & 0 \\ 0 & U_2 & 0 \\ 0 & 0 & U_4 \end{pmatrix}, \tag{23}$$

where U_2 is shown in (9) and I_2 is a 2×2 identity matrix. W_8 is a Walsh-Hadamard matrix which is calculated using (21). According to (11), $m = \log_2(8) + 1 = 4$ and $3 \leq q \leq 4$. So U_4 can be written as

$$U_4 = P_4 \cdot V_{4,4} \cdot V_{4,3} \cdot P_4, \tag{24}$$

where, according to (14)

$$V_{4,4} = \begin{pmatrix} c_{1,4} & 0 & 0 & s_{1,4} \\ 0 & c_{3,4} & s_{3,4} & 0 \\ 0 & -s_{3,4} & c_{3,4} & 0 \\ -s_{1,4} & 0 & 0 & c_{1,4} \end{pmatrix} \tag{25}$$

and, according to (15), $r = m - q + 1 = 4 - 3 + 1 = 2$ and $V_{N/2^2,q} = V_{8/4,3} = V_{2,3} = U_2$, so

$$V_{4,3} = \begin{pmatrix} U_2 & 0 \\ 0 & U_2 \end{pmatrix}. \tag{26}$$

Using (12) and (13) the permutation is computed as

$$\Phi_4 = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 0 & 2 & 1 & 3 \end{pmatrix}. \tag{27}$$

Finally the permutation matrix P_4 is obtained by applying the permutation, shown in (27), to the columns of 4×4 identity matrix as,

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{28}$$

3.3. 16×16 *DCT.* Equation (6) can be given as

$$H_{16} = \begin{pmatrix} H_8 & H_8 \\ H_8 & -H_8 \end{pmatrix}. \tag{29}$$

Equation (7) can be given as

$$C_8^{II} = \frac{1}{4} \cdot B_{16} \cdot T_{16} \cdot B_{16} \cdot W_{16}, \tag{30}$$

where

$$T_{16} = \begin{pmatrix} I_2 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 \\ 0 & 0 & U_4 & 0 \\ 0 & 0 & 0 & U_8 \end{pmatrix}, \tag{31}$$

where U_2 and U_4 are shown in (9) and (24). To calculate U_8, according to (11), $m = \log_2(16) + 1 = 5$ and $3 \le q \le 5$. So U_8 can be written as

$$U_8 = P_8 \cdot V_{8,5} \cdot V_{8,4} \cdot V_{8,3} \cdot P_8, \tag{32}$$

where, according to (14),

$$V_{8,5} = \begin{pmatrix} c_{1,5} & 0 & 0 & 0 & 0 & 0 & 0 & s_{1,5} \\ 0 & c_{3,5} & 0 & 0 & 0 & 0 & s_{3,5} & 0 \\ 0 & 0 & c_{5,5} & 0 & 0 & s_{5,5} & 0 & 0 \\ 0 & 0 & 0 & c_{7,5} & s_{7,5} & 0 & 0 & 0 \\ 0 & 0 & 0 & -s_{7,5} & c_{7,5} & 0 & 0 & 0 \\ 0 & 0 & -s_{5,5} & 0 & 0 & c_{5,5} & 0 & 0 \\ 0 & -s_{3,5} & 0 & 0 & 0 & 0 & c_{3,5} & 0 \\ -s_{1,5} & 0 & 0 & 0 & 0 & 0 & 0 & c_{1,5} \end{pmatrix}. \tag{33}$$

Similarly, to calculate $V_{8,4}$, according to (15), $r = m - q + 1 = 5 - 4 + 1 = 2$ and $V_{N/2^2,q} = V_{16/4,4} = V_{4,4}$, so

$$\mathrm{V}_{8,4} = \begin{pmatrix} V_{4,4} & 0 \\ 0 & V_{4,4} \end{pmatrix}, \tag{34}$$

where $V_{4,3}$ is calculated in (25). For $V_{8,3}$, $r = m - q + 1 = 5 - 3 + 1 = 3$ and $V_{N/2^3,q} = V_{16/8,3} = V_{2,3} = U_2$, so

$$V_{8,3} = \begin{pmatrix} U_2 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 \\ 0 & 0 & U_2 & 0 \\ 0 & 0 & 0 & U_2 \end{pmatrix}. \tag{35}$$

Using (12) and (13) the permutation is computed as

$$\Phi_8 = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 4 & 2 & 6 & 1 & 5 & 3 & 7 \end{pmatrix}. \tag{36}$$

Finally the permutation matrix P_8 is obtained by applying the permutation, shown in (37), to the columns of 8×8 identity matrix as

$$P_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{37}$$

3.4. 32×32 *DCT.* Equation (6) can be given as

$$H_{32} = \begin{pmatrix} H_{16} & H_{16} \\ H_{16} & -H_{16} \end{pmatrix}. \tag{38}$$

Equation (7) can be given as

$$C_{32}^{II} = \frac{1}{4\sqrt{2}} \cdot B_{32} \cdot T_{32} \cdot B_{32} \cdot W_{32}, \tag{39}$$

where

$$T_{32} = \begin{pmatrix} I_2 & 0 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 & 0 \\ 0 & 0 & U_4 & 0 & 0 \\ 0 & 0 & 0 & U_8 & 0 \\ 0 & 0 & 0 & 0 & U_{16} \end{pmatrix}, \tag{40}$$

where U_2, U_4 and U_8 are shown in (9), (24), and (32). To calculate U_{16}, according to (11), $m = \log_2(32) + 1 = 6$ and $3 \le q \le 56$. So U_{16} can be written as

$$U_{16} = P_{16} \cdot V_{16,6} \cdot V_{16,5} \cdot V_{16,4} \cdot V_{16,3} \cdot P_{16}, \tag{41}$$

where, according to (14),

$$V_{16,6} = \begin{pmatrix} c_{1,6} & 0 & 0 & 0 & 0 & 0 & 0 & s_{1,6} \\ 0 & c_{3,6} & 0 & 0 & 0 & 0 & s_{3,6} & 0 \\ 0 & 0 & \ddots & 0 & 0 & \iddots & 0 & 0 \\ 0 & 0 & 0 & c_{15,6} & s_{15,6} & 0 & 0 & 0 \\ 0 & 0 & 0 & -s_{15,6} & c_{15,6} & 0 & 0 & 0 \\ 0 & 0 & \iddots & 0 & 0 & \ddots & 0 & 0 \\ 0 & -s_{3,6} & 0 & 0 & 0 & 0 & c_{3,6} & 0 \\ -s_{1,6} & 0 & 0 & 0 & 0 & 0 & 0 & c_{1,6} \end{pmatrix}. \tag{42}$$

Similarly, to calculate $V_{16,5}$, according to (15), $r = m-q+1 = 6-5+1 = 2$ and $V_{N/2^2,q} = V_{32/4,5} = V_{8,5}$, so

$$V_{16,5} = \begin{pmatrix} V_{8,5} & 0 \\ 0 & V_{8,5} \end{pmatrix}, \tag{43}$$

where $V_{8,3}$ is calculated in (33). For $V_{16,4}$, $r = m-q+1 = 6-4+1 = 3$ and $V_{N/2^3,q} = V_{32/8,4} = V_{4,4}$, so

$$V_{16,4} = \begin{pmatrix} V_{4,4} & 0 & 0 & 0 \\ 0 & V_{4,4} & 0 & 0 \\ 0 & 0 & V_{4,4} & 0 \\ 0 & 0 & 0 & V_{4,4} \end{pmatrix}, \tag{44}$$

where $V_{4,4}$ is calculated in (25). For $V_{16,3}$, $r = m-q+1 = 6-3+1 = 4$ and $V_{N/2^4,q} = V_{32/16,3} = V_{2,3} = U_2$, so

$$V_{16,3} = \begin{pmatrix} U_2 & 0 & 0 & 0 \\ 0 & U_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & U_2 \end{pmatrix}. \tag{45}$$

Using (12) and (13) the permutation is computed as

$$\Phi_{16} = \begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ 0 & 8 & 4 & 12 & 2 & 10 & 6 & 14 & 1 & 9 & 5 & 13 & 3 & 11 & 7 & 15 \end{pmatrix} \tag{46}$$

finally the permutation matrix P_{16} is obtained by applying the permutation, shown in (47), to the columns of 16×16 identity matrix as

$$P_{16} = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix}. \tag{47}$$

4. Proposed Architecture

The complete hardware architecture of the DCT is shown in Figure 1. Each frame is loaded in the input frame memory. The complete frame is divided into $N \times N$ blocks. The control unit reads the rows of each block from the input memory. At the same time, the control unit passes a "1" to the input multiplexers. Also the address and other control signals are passed to the DCT block. After complete calculation of the DCT, the transformed row is input to the transpose memory, along with its corresponding address. In this way, for the first N clock cycles, the rows from the input memory are input to the DCT and are written on the corresponding addresses in the transpose memory. After the N clock cycles, the control unit passes a "0" for the input multiplexers for the next N clock cycles. So in this way, each column from the transpose memory is input to DCT block, and the outputs of the DCT block are written back in the transpose memory, on the same location from where they are read. When all the columns are read and processed by the DCT, the control unit again starts reading the next $N \times N$ block from the input memory and at the same time, each row from the transpose memory is written to the output transformed memory. In this way all the $N \times N$ blocks are read, processed, and written in the output transformed memory.

When the last row is processed through the DCT, it is written to the transpose memory. At the same time, the first column from the transpose memory is read in order to be processed through DCT block. As the last row was not written, so the last data of the first column is not valid. So "Data0" multiplexer is used for forwarding. In this way, the first output of last transformed row of a $N \times N$ block is forwarded to the input to DCT and also written to the transpose memory.

4.1. DCT Block. DCT block is the main block of the complete architecture. The DCT block takes the input data, the corresponding control signals, and the corresponding addresses. The internal architecture of the DCT block is shown in Figure 2.

DCT block has 4 pipeline stages. The data is passed through the Hadamard block. The Hadamard block is designed with a fully parallel architecture. The Hadamard block takes 32 data at its inputs and passes to the butterfly_32, while the first 16 are input to the butterfly_16, the first 8 are input to the butterfly_8, and the first 4 are input to the butterfly_4 as well. Multiplexers are placed at the inputs of each different size butterfly in order to have correct result from Hadamard block. The select signals for the multiplexers are controlled by the control unit. The Hadamard block has 32 outputs. To have a Walsh transform from a Hadamard one, the bit_reversal and gray_code blocks are placed after the Hadamard block.

In the bit_reversal block, the data at input port number X is moved to output port number Y, where the output port number Y is determined by representation the input port number X and reversing the bits. For example, in case of DCT_16, the Hadamard block will produce 16 valid outputs. So the 16 inputs to the bit_reversal block are shuffled according to bit_reversal rule, for example, $X = 0$ means $X =$ "0000" and the bit_reversal is also $Y =$ "0000". So the first input port will be connected to the first output port. Similarly, for $X =$ "0001", the bit reversal will $Y =$ "1000" which means that the second input port is connected to the eight output port. In this way all the inputs are connected to the outputs according to bit reversal rule. As the architecture supports four different sizes of DCT, it means that the bit

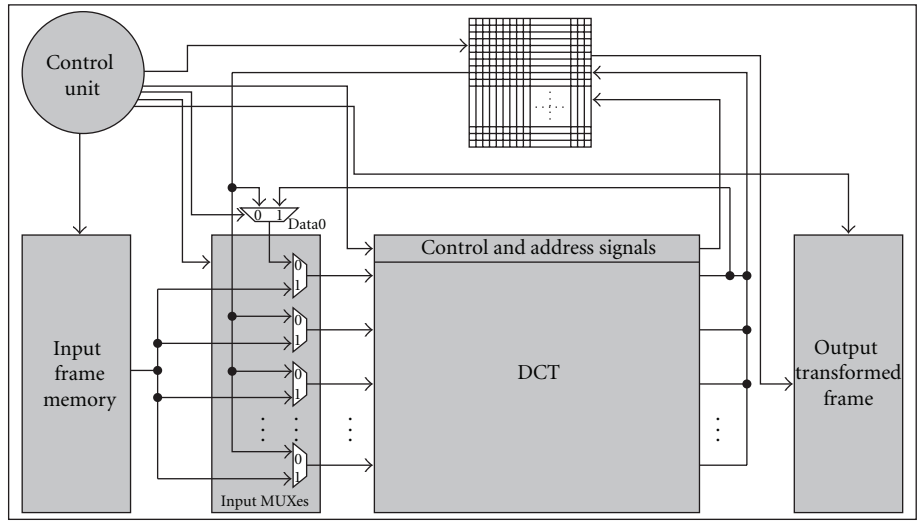

Figure 1: Top level hardware architecture of DCT.

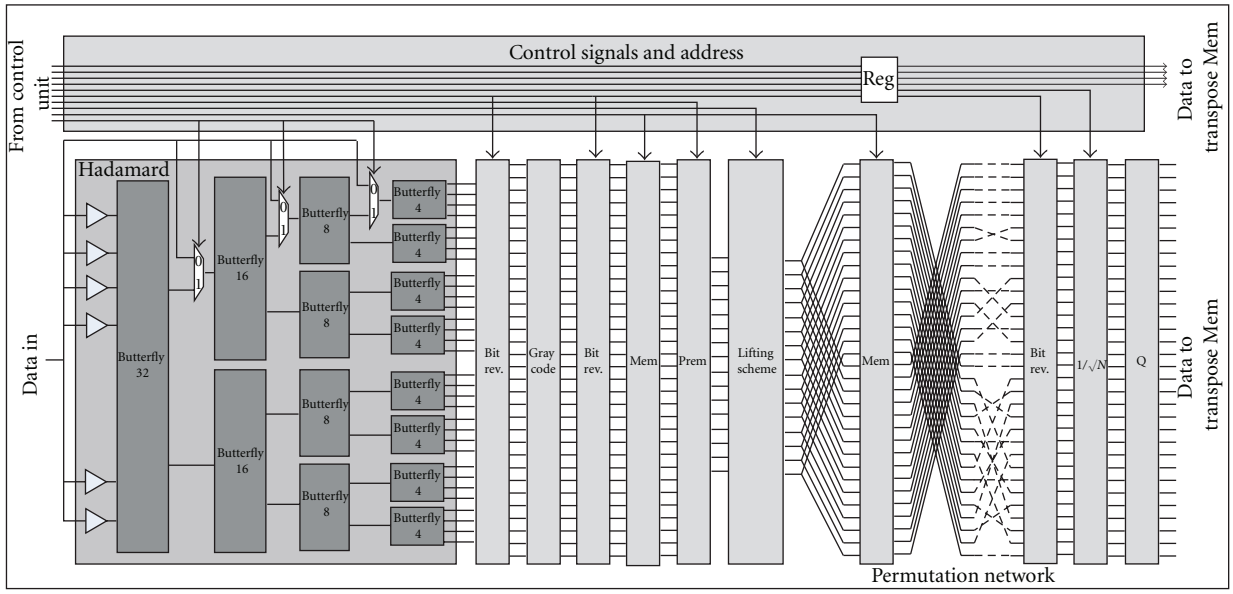

Figure 2: DCT block internal structure.

reversal rule will be different for each DCT size. For example, for DCT_4, X = "01" will be connected to Y = "10", that is, 2nd output port, while in case of DCT_16, it will be connected to the 8th output port. So multiplexers are placed in order to support all the DCT sizes in the bit reversal block.

Gray code block works in the same principle as bit reversal block, but according to gray code law. In gray code block, the output port is determined by applying gray code on the input addresses. For example, for DCT_32 if X = "01101", Y = "01011". So the input port number 13 is connected with the output port number 11. Gray code calculation does not depend on the DCT size. For example for DCT_16, if X = "1101", Y = "1011", which means that input port 13 is connected to the output port number 11, which is same as that for DCT_32.

The architecture of mem.block is shown in Figure 3. The memory block connects the first 16 inputs directly to the output ports, while the last 16 outputs are multiplexed with the latched inputs and the direct inputs. The last 16 outputs are used in case of DCT_32, while the last 16 last inputs are bypassed in case of DCT_4, DCT_8, and DCT_16.

The permutation block is implemented using (27), (36), and (46). The block takes 32 inputs, and it sends 16 of the inputs to the outputs according to the permutation law. The first 16 inputs to the permutation block are passed to the outputs in the first clock cycle, while in the second clock cycle the last 16 inputs of the permutation network are passed to the outputs. In case of $DCT < 32$, the selection line of the multiplexers is always set to "0", while in case of $DCT = 32$, the selection line remains "0" in first clock cycle, while remains "1" in the next clock cycle.

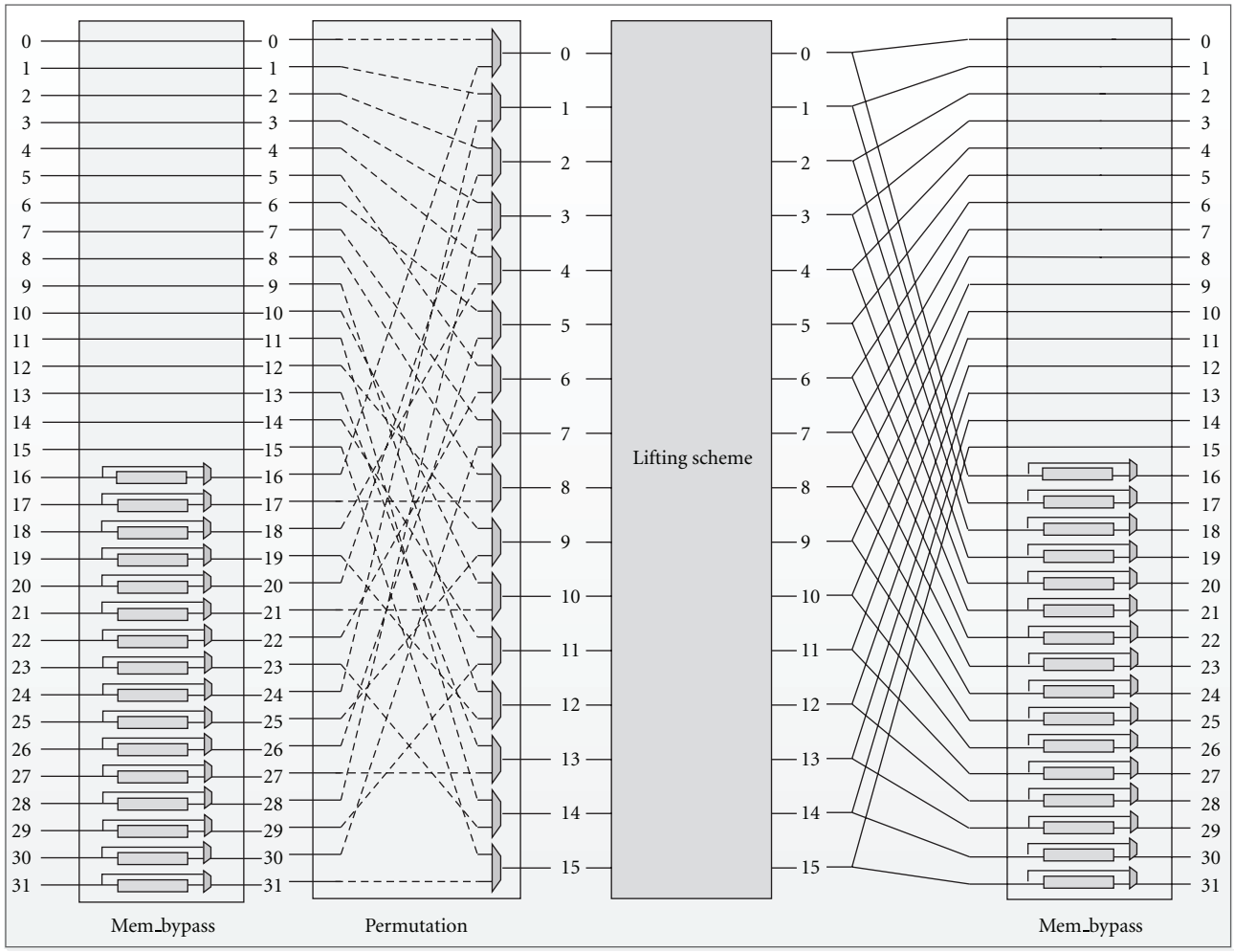

Figure 3: Memory block and permutation block.

The lifting scheme is implemented using (19), (23), (31), and (40). The lifting block is implemented with a folded architecture. Where the fully parallel lifting block is used for DCT sizes of 4, 8, and 16, while DCT_32, the block is reused. Each row of DCT_32 takes 2 clock cycles for completion. During the first clock cycle, the upper 16 inputs are processed by the lifting scheme and are stored in the memory block. In the next clock cycle, the lower 16 inputs are processed by the lifting block and the result along with the previously calculated stored values is forwarded to the next block. The lifting block is shown in Figure 4.

Lifting scheme is designed for 15 Givens rotations. The basic lifting structure, shown in Figure 5, is implemented using (16), where

$$\frac{a}{2^m} \approx \frac{1-\cos\theta}{\sin\theta}, \qquad \frac{b}{2^n} \approx -\sin\theta, \tag{48}$$

as suggested in [9].

The lifting structures takes two values at the inputs. For each Givens rotation, the a, b, m, and n are approximated integer values, in order to have the approximated DCT equal to the actual DCT. As a and b are integers, the multiplications are implemented using adders and shift operations. The result of each lifting structure is quantized to 16-bits resolution to have a reasonable PSNR value. So the final outputs of the lifting block are 16-bit wide. The results of some lifting structures are bypassed using the multiplexers at their outputs. In fact, the select line for the multiplexers will always remain "0" for DCT_4, DCT_8, and DCT_16, while for DCT_32, the select line remains "0" for first clock cycle and "1" the for the next clock cycle.

In case of DCT_32, the Hadamard block produces 32 results in parallel. The outputs of the Hadamard block are fed into the bit reversal block, gray code block, and in the following bit reversal block, and the output of the bit reversal block is fed into the memory block. The memory block passes the upper 16 inputs directly to the permutation block, while the lower 16 inputs are stored in the registers. The permutation block forwards the upper 16 inputs to the lifting scheme, where the select lines for the multiplexers in the lifting scheme are set to "0", and the lifting scheme outputs the results and the results are stored in the following memory

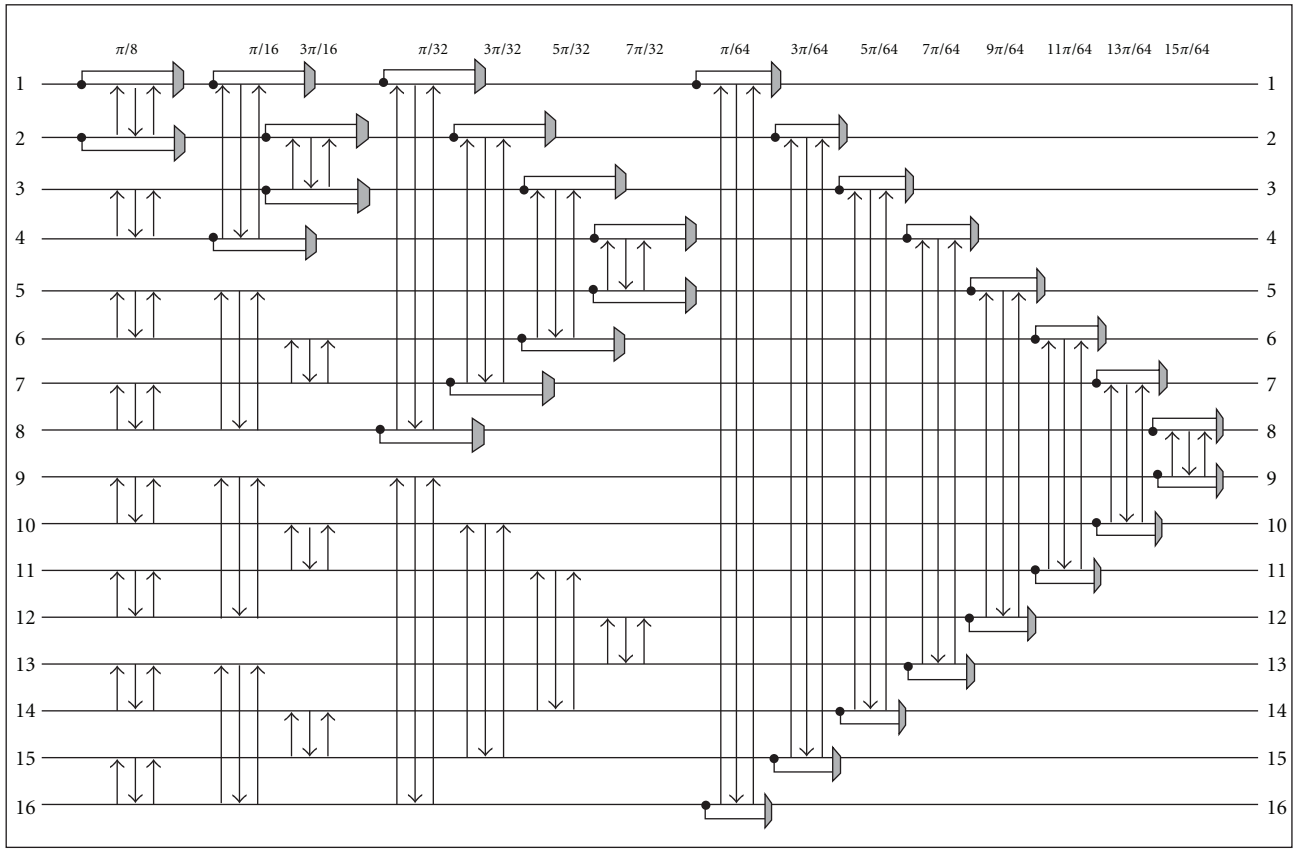

Figure 4: Folded lifting scheme.

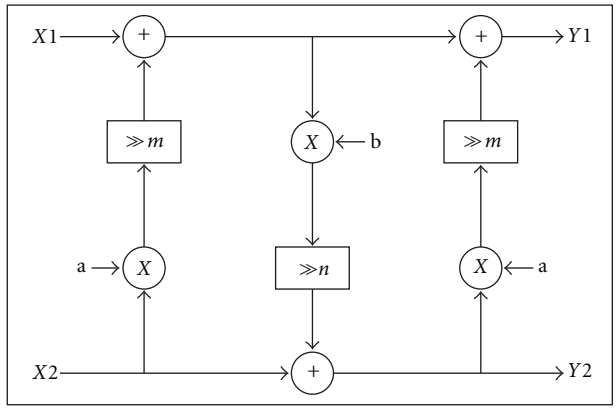

Figure 5: Basic lifting structure.

block. In the next clock cycle the lower 16 values stored in the first memory block are passed to the lifting scheme, through the permutation block. The selection line for the multiplexers in the lifting scheme are set to "1". The 16 results are calculated and are passed to the memory block. At the same time, the memory block forwards the previously stored values along with the new arrived ones.

In case of DCT_4, DCT_8, and DCT_16, the lower 16 values from the first memory blocks are invalid and never used. So the valid upper 16 inputs are fed into the lifting scheme via permutation network. The selection line for the lifting scheme multiplexers is always set to "0" in case of $DCT < 32$.

$DCT < 32$ takes one clock cycle to calculate one row or one column, while $DCT = 32$ takes 2 clock cycles. The outputs of the second memory block are passed to the third bit reversal block, passing through a fully parallel permutation network. The outputs of the bit reversal are then divided by square root of N, where N is the DCT size. The square roots are calculated as

$$\frac{1}{\sqrt{N}} \approx \begin{cases} \dfrac{1}{2}, & N = 2, \\ \dfrac{8+4-1}{32}, & N = 8, \\ \dfrac{1}{4}, & N = 16, \\ \dfrac{8+4-1}{64}, & N = 32. \end{cases} \tag{49}$$

The hardware architecture of the one square root block is shown in Figure 6. The input is divided by $\sqrt{N}$ and the calculated values are fed into the output multiplexer, where the valid result is sent to output depending on the DCT size. Finally, the outputs from the square root block are quantized from 16 bits to 13 bits using the Q block.

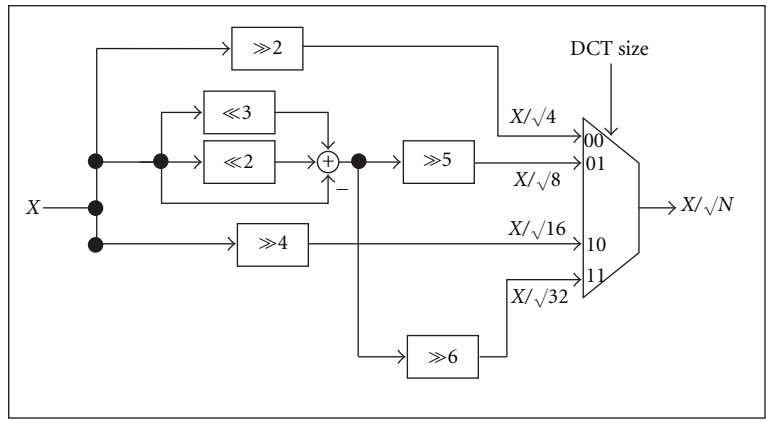

Figure 6: $1/\sqrt{N}$ block.

4.2. Transpose Buffer. Transpose buffer is designed using registers. The buffer is designed to support maximum DCT size, that is, DCT_32. So the buffer is of size $N \times N \times B$, where $N = 32$ and $B = 13$, where B is the width of each data. So a total of 13 kbits memory is utilized to implement transpose buffer. The inputs of the buffer are the clock, reset, transpose signal, the row number, the column number, read enable signal, and the write enable signal. During the direct cycle, all the rows from the input frame memory are transformed through DCT block, and the results are stored on the corresponding rows in the transpose buffer. When all the rows of a input frame memory are transformed and written to the transform buffer, the columns of the transform buffer are read and the columns are transformed via DCT block, and the results are again written back to the transform buffer in transpose way, that is, on each column. When all the columns of the transform buffer are read, transformed, and written back to the buffer, the rows of the input frame memory are read row wise, and at the same time the rows of the transform buffer are written to the output frame memory. In this way, the complete frame is transformed and written to the output memory.

4.3. Input MUXes Block. The DCT transforms the rows of the block and the results are stored in the buffer. So the select signal for the input MUXes block is set to "1". After N clock cycles, where N is the size of DCT, the select signal of the input MUXes is set to "0", so that the columns from the transform buffer is fed into the DCT block. So, input MUXes block switches the inputs for the DCT block for the direct or transformed cycles.

4.4. Data0 Multiplexer. During the direct cycles, the rows from the input memory are transformed and the results are written in the transform buffer. When the last row from the input memory is transformed, first column is read in the next clock cycle from the transform memory. At this point, the last data of the first column is not the valid one, as the last transformed row has not yet been written to the memory. So data0 multiplexer is used for forwarding, where the first data out of the 32 transformed dates is selected from the data0 memory. The select line of the data0 multiplexer is set to "1" for just one clock cycle during transformation of $N \times N$ block, that is, when the first column is read from transform buffer and the last row is transformed via DCT block, otherwise the select signal is always set to "0".

4.5. Control Unit. Control Unit controls the activities of all the blocks in each clock cycle. This unit is responsible for a correct sequence of operations. Control unit is designed using 4 memories, where each memory contains the control signals for each DCT size. There are 4 counters in the unit, where each counter produces the addresses for its corresponding memories. In response to the addresses, the memories output the control signals. The outputs of the memories are multiplexed, where the selection line of the multiplexer decides which input to go out. The hardware architecture of the control unit is shown in Figure 7. MEM_CU_4 is a $8 \times 128 = 1$ kbits size, MEM_CU_8 is a $16 \times 128 = 2$ kbits size, and MEM_CU_16 is a $32 \times 128 = 16$ kbits bits size while MEM_CU_32 is a $128 \times 128 = 16$ kbits size. The memories contains N control signals for direct cycle and N clock cycles for the transpose cycle, where N is the DCT size. But MEM_CU_32 contains $2(N + N)$ control signals, because each row or column takes two clock cycles for completion in case of DCT_32. So the control unit generates 128-bits wide control signals, for all the functioning blocks of the complete DCT in each clock cycle.

5. Results

The computation of N-point DCT by the means of the WHT factorization requires the following.

(1) $(N/2) \cdot \log_2(N)$ 2-inputs/2-output butterfly for the N-point WHT.

(2) $1 + (N/2) \cdot [\log_2(N) - 2]$ 2-input/2-output lifting-based (3-lifting-step) structures for the Givens rotations.

The WHT is implemented with fully parallel architectures using maximum number of resources, while the lifting

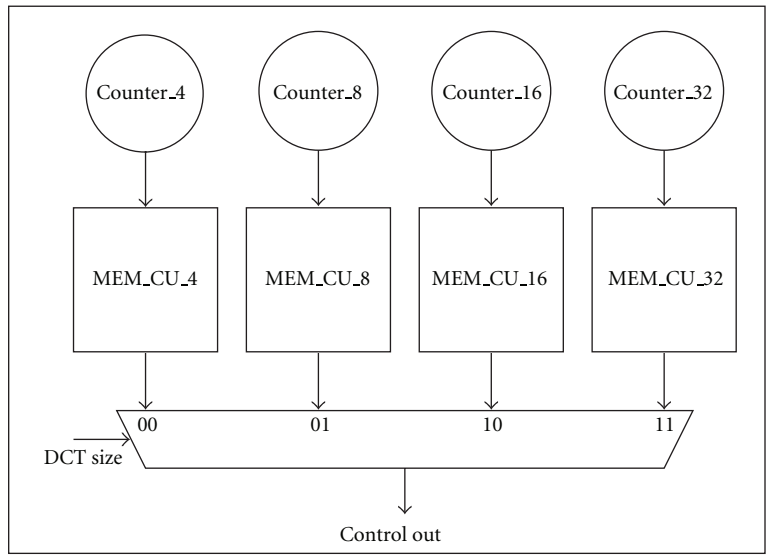

Figure 7: Control unit.

scheme is implemented with a folded architecture. Hence, 80 2-input/2-output butterflies are used to implement WHT for $N = 32$. The number of adders required to implement the 80 2-input/2-output butterflies is 160. The total number of 2-input/2-output lifting structures to implement the 15 Givens rotation is 49, but with folded architecture we have reused the data path and reduced the number of lifting structures to 32.

The factorization of the matrices is applied to H.265 DCT. The lifting coefficients are approximated with the following condition:

$$\left\lfloor \frac{1}{N} \cdot B_N \cdot T_N \cdot B_N \cdot W_N \cdot 2^7 \right\rfloor = Q, \tag{50}$$

where Q is the N-point DCT obtained from MATLAB function dctmtx, scaled with 2^7. Table 1 shows the approximated values, calculated from the conditions in (48), of all the coefficients and the number of bits required to normalize the results.

According to [14], multiplications for $a_{1,3} = 51$ and $b_{1,3} = -98$ can be implemented with a minimum number of additions resorting to the n-dimensional reduced adder graph (RAG-n) technique. The total number of adders required for all the coefficients is shown in Table 1.

The DCT block contains $2 \times N/2 \times \log_2(N) = 160$ adders to implement the Hadamard block. Using Tables 1 and 2, the number of adders use to implement the DCT can be calculated. The first stage of lifting steps ($\pi/8$ rotation) requires $3 \times 8 = 24$ adders to implement the lifting structure, and further $6 \times 8 = 48$ adders are required to implement all the steps involving $a_{1,3}$ and $b_{1,3}$. The first stage of lifting steps ($\pi/16$ rotation) requires $3 \times 4 = 12$ adders to implement the lifting structure, and further $8 \times 4 = 32$ adders are required to implement all the steps involving $a_{1,4}$ and $b_{1,4}$. The first stage of lifting steps ($3\pi/16$ rotation) requires $3 \times 4 = 12$ adders to implement the lifting structure, and further $9 \times 4 = 36$ adders are required to implement all the steps involving $a_{3,4}$ and $b_{3,4}$. The first stage of lifting steps ($\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $6 \times 2 = 12$ adders are required to implement all the steps involving $a_{1,5}$ and $b_{1,5}$. The first stage of lifting steps ($3\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $7 \times 2 = 14$ adders are required to implement all the steps involving $a_{3,5}$ and $b_{3,5}$.

The first stage of lifting steps ($5\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $2 \times 2 = 4$ adders are required to implement all the steps involving $a_{5,5}$ and $b_{5,5}$. The first stage of lifting steps ($7\pi/32$ rotation) requires $3 \times 2 = 6$ adders to implement the lifting structure, and further $9 \times 2 = 18$ adders are required to implement all the steps involving $a_{7,5}$ and $b_{7,5}$. The first stage

Table 1: Number of adders for lifting coefficients.

a	Adders	b	Adders
51	2	−98	2
101	3	−569	2
311	3	−200	3
25	2	−50	2
152	2	−297	3
64	0	−121	2
183	3	−325	3
25	2	−50	2
19	2	−38	2
63	1	−124	1
178	3	−345	4
115	3	−219	3
71	2	−132	1
169	3	−305	3
99	3	−172	3

Table 2: Approximated valued of lifting structures coefficients and the number of bits required for quantization of results.

Givens rotations	m, n	a	b
$\pi/8$	8	51	−98
$\pi/16$	10	101	−569
$3\pi/16$	10	311	−200
$\pi/32$	9	25	−50
$3\pi/32$	10	152	−297
$5\pi/32$	8	64	−121
$7\pi/32$	9	183	−325
$\pi/64$	10	25	−50
$3\pi/64$	8	19	−38
$5\pi/64$	9	63	−124
$7\pi/64$	10	178	−345
$9\pi/64$	9	115	−219
$11\pi/64$	8	71	−132
$13\pi/64$	9	169	−305
$15\pi/64$	8	99	−172

of lifting steps ($\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 6 adders are required to implement all the steps involving $a_{1,6}$ and $b_{1,6}$. The first stage of lifting steps ($3\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 6 adders are required to implement all the steps involving $a_{3,6}$ and $b_{3,6}$. The first stage of lifting steps ($5\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 3 adders are required to implement all the steps involving $a_{5,6}$ and $b_{5,6}$. The first stage of lifting steps ($7\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 10 adders are required to implement all the steps involving $a_{7,6}$ and $b_{7,6}$. The first stage of lifting steps ($9\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 9 adders are required to implement all the steps involving $a_{9,6}$ and $b_{9,6}$. The first stage of lifting steps ($11\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 5 adders are required to implement all the steps involving $a_{11,6}$ and $b_{11,6}$. The first stage of lifting steps ($13\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 9 adders are required to implement all the steps involving $a_{13,6}$ and $b_{13,6}$. The first stage of lifting steps ($15\pi/64$ rotation) requires 3 adders to implement the lifting structure, and further 9 adders are required to implement all the steps involving $a_{15,6}$ and $b_{15,6}$. The square root block contains 2 adders, and there are 32 square root blocks. Therefore, 64 adders are calculating square roots in parallel. The total number of adders required for Hadamard and lifting scheme is $160 + 72 + 44 + 48 + 18 + 20 + 10 + 24 + 9 + 9 + 6 + 13 + 12 + 15 + 12 + 12 + 64 = 548$.

Figure 8 shows the experiment setup carried out to calculate the PSNR. The original frames are transformed using the proposed DCT. Then the transformed data is quantized to 13 bits. The quantized coefficients are then passed through the inverse quantization block and inverse DCT. The PSNR is then calculated between the original frames and the reconstructed frames. The inverse quantization is taken using (51)

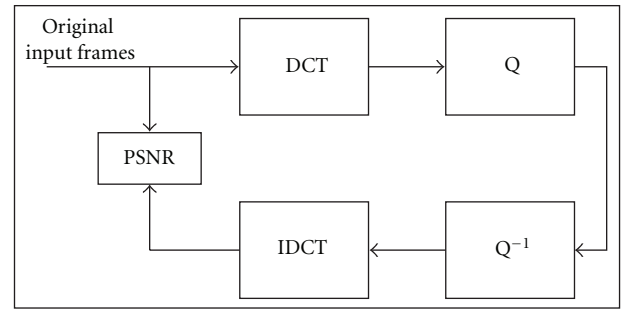

Figure 8: PSNR calculation and experiment setup.

$$x = \left(C_N^{II}\right)^{-1} \cdot X^t. \tag{51}$$

Table 3 shows the PSNR values for different sequences with different DCT sizes. The PSNR is calculated as shown in (52) and (53)

$$\text{PSNR} = 20 \cdot \log\left(\frac{\text{MAX}_\text{I}^2}{\sqrt{\text{MSE}}}\right), \tag{52}$$

where MAX_I is the maximum possible value of the image, and MSE, mean square error, can be defined as

$$\text{MSE} = \frac{1}{m \cdot n} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2, \tag{53}$$

where $I(i, j)$ and $K(i, j)$ are the input frame and the reconstructed frame, after inverse quantization and IDCT, respectively. From Table 3, it is quite clear that the DCT is showing great efficiency with respect to PSNR. PSNR of Y frames is very close to 50 dB.

In Tables 4, 5, and 6, the number of multiplications, additions, and shifts, required to calculate different sizes of DCT, are shown. The proposed architecture has no multiplications, where all the multiplications are implemented using shifts and adds. As it can be observed, the number of additions required to compute the 32-point DCT with the proposed architecture is less than the original DCT implementation and the other proposed ones.

The net list is written in VHDL language. Synopsys Design Vision is used for synthesis purpose. The code is synthesized on 90 nm standard cell library at a clock frequency of 150 MHz. Table 7 shows the results of the synthesis.

The time required by the proposed architecture to completely process an $N \times N$ macro block is

$$T_{\text{MB}} = \frac{2}{f_{\text{clk}}} \cdot (\delta \cdot N + 4), \tag{54}$$

where $\delta = 2$ if $N = 32$ and $\delta = 1$ otherwise. Thus, the total time to process one $W \times H$ pixel frame is

$$T = \frac{W \cdot H}{N^2} \cdot T_{\text{MB}} \cdot K = \frac{W \cdot H}{N^2} \cdot K \cdot \frac{2\delta \cdot N + 8}{f_{\text{clk}}}, \tag{55}$$

Table 3: PSNR (dB) of different sequences for different DCT sizes.

Sequences	DCT size											
	$N = 4$			$N = 8$			$N = 16$			$N = 32$		
	Y	U	V	Y	U	V	Y	U	V	Y	U	V
BQ terrace_1920 × 1080_60	47.1	44.4	44.1	48.8	45.9	44.1	48.9	45.4	45.3	48.8	45.7	44.1
BQ square_416 × 240_60	47.4	45.1	45.2	48.1	45.2	45.2	48.1	45.7	44.1	48.2	45.2	44.7
BQ mall_832 × 480_60	46.9	44.1	44.5	47.9	44.9	45.5	48.7	44.6	44.9	48.4	45.7	45.3
Basketball drive_1920 × 1080_50	47.8	44.5	44.1	48.2	44.8	45.7	48.5	45.1	45.7	48.3	45.1	45.3
Basketball drill_832 × 480_50	47.0	45.4	45.6	48.6	45.0	44.8	48.2	45.9	45.0	48.7	45.4	45.5

Table 4: Proposed architecture.

N	M	A	S
4	0	17	5
8	0	74	39
16	0	232	132
32	0	548	249

* N is the DCT size.
* M is the number of multiplications.
* A is the number of additions.
* S is the number of shifts.

Table 5: Comparison for $N = 32$.

	M	A	S
Original	1024	992	0
Proposed	0	548	249

Table 6: Comparison for $N = 16$.

	M	A	S
[5]	0	242	58
Proposed	0	232	132

* N is the DCT size.
* M is the number of multiplications.
* A is the number of additions.
* S is the number of shifts.

Table 7: Synthesis results.

Parameter	Value
Technology	90 nm
Frequency	150 MHz
Area	0.42 mm^2
Power	884.1 μW
Memory	36 kbits

where K accounts for the chroma subsampling, for example, $K = 3$ for 4 : 4 : 4, $K = 2$ for 4 : 2 : 2, and $K = 1.5$ for 4 : 2 : 0. So from (55), taking $H = 1920$, $W = 1080$, and $K = 1.5$ we obtain $T = 2.8$ ms and $T = 20.7$ ms for $N = 32$ and $N = 4$, respectively. As a consequence, in the worst case ($N = 4$) the proposed architecture sustains up to 48 frames per second.

6. Conclusion

In this work, a dynamic N-point DCT for HEVC is proposed. A partially folded architecture is adopted to maintain speed and to save area. The DCT supports 4, 8, 16, and 32 points. The simulation results show that the PSNR is very close to 50 dB, which is reasonably good. Multiplications are removed from the architecture by introducing lifting scheme and approximating the coefficients.

References

[1] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560–576, 2003.

[2] K. Ugur, K. Andersson, A. Fuldseth et al., "High performance, low complexity video coding and the emerging hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 20, no. 12, pp. 1688–1697, 2010.

[3] S. Song, C. Seo, and K. Kim, "A unified transform unit for H.264," in *Proceedings of the International SoC Design Conference (ISOCC '08)*, pp. 130–133, November 2008.

[4] S. Saponara, M. Martina, M. Casula, L. Fanucci, and G. Masera, "Motion estimation and CABAC VLSI co-processors for real-time high-quality H.264/AVC video coding," *Microprocessors and Microsystems*, vol. 34, no. 7-8, pp. 316–328, 2010.

[5] M. N. Haggag, M. El-Sharkawy, and G. Fahmy, "Efficient fast multiplication-free integer transformation for the 2-D DCT H.265 standard," in *Proceedings of the 17th IEEE International Conference on Image Processing (ICIP '10)*, pp. 3769–3772, September 2010.

[6] Y. M. Huang, J. L. Wu, and C. T. Hsu, "A refined fast 2-D discrete cosine transform algorithm with regular butterfly structure," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 2, pp. 376–383, 1998.

[7] D. Hein and N. Ahmed, "On a real-time Walsh-Hadamard cosine transform image processor," *IEEE Transactions on Electromagnetic Compatibility*, vol. EMC-20, pp. 453–457, 1978.

[8] 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.

[9] Y. J. Chen, S. Oraintara, and T. Nguyen, "Video compression using integer DCT," in *International Conference on Image Processing (ICIP'00)*, pp. 844–845, September 2000.

[10] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *Journal of Fourier Analysis and Applications*, vol. 4, no. 3, pp. 247–268, 1998.

[11] M. Martina, G. Masera, G. Piccinini, and M. Zamboni, "A VLSI architecture for IWT (Integer Wavelet Transform)," in *Proceedings of the 43rd IEEE Midwest Symposium on Circuits and Systems*, pp. 1174–1177, August 2000.

[12] M. Martina, G. Masera, G. Piccinini, and M. Zamboni, "Novel JPEG 2000 compliant DWT and IWT VLSI implementations," *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 35, no. 2, pp. 137–153, 2003.

[13] M. Martina and G. Masera, "Folded multiplierless lifting-based wavelet pipeline," *IET Electronics Letters*, vol. 43, no. 5, pp. 27–28, 2007.

[14] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 9, pp. 569–577, 1995.

[15] M. Martina and G. Masera, "Low-complexity, efficient 9/7 wavelet filters VLSI implementation," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 53, no. 11, pp. 1289–1293, 2006.

[16] M. Martina and G. Masera, "Multiplierless, folded 9/7–5/3 wavelet VLSI architecture," *IEEE Transactions on Circuits and Systems II*, vol. 54, no. 9, pp. 770–774, 2007.

[17] R. Joshi, Y. A. Reznik, and M. Karczewicz, "Efficient large size transforms for high-performance video coding," in *33rd Proceedings of the Applications of Digital Image Processing*, Proceedings of the SPIE, San Diego, Calif, USA, August 2010.

[18] M. Martina, G. Masera, and G. Piccinini, "Scalable low-complexity B-spline discrete wavelet transform architecture," *IET Circuits, Devices and Systems*, vol. 4, no. 2, pp. 159–167, 2010.

[19] V. Britanak, P. C. Yip, and K. R. Rao, *Discrete Cosine and Sine Transforms: General Properties, Fast Algorithms and Integer Approximations*, Elsevier, 2007.

[20] H. W. Jones, D. N. Hein, and S. C. Knauer, "The Ratio CO/CO_2 of oxidation on a burning carbon surface," pp. 87–98, November 1978.

[21] N. Ahmed and K. R. Rao, *Orthogonal Transforms for Digital Signal Processing*, Springer, 1975.

[22] J. W. Manz, "A sequency-ordered fast Walsh transform," *IEEE Transactions on Audio Electroacoustics*, vol. AU-20, pp. 204–205, 1972.

[23] A. T. S. Claire and C. Sabido-David, "Unified matrix treatment of the fast Walsh-Hadamad transfor," *IEEE Transactions on Computers*, vol. 25, no. 11, pp. 1142–1146, 1976.