Harware-Efficient Implementation of DCT Architectures for Image Communications

Abhishek kumar verma[1], Ambresh Patel[2]
M.Tech (VLSI) Scholar, RKDF University Bhopal, India[1]
Assistant Professor, Department of Electricals & Electronic Engineering, RKDF University Bhopal, India[2]

Abstract: In this paper, the area- and power-efficient architectures of integer discrete cosine transform (DCT) of different lengths to be used in High Efficiency Video Coding (HEVC) are implemented. An efficient constant matrix-multiplication scheme can be used to derive parallel architectures for 1-D integer DCT of different lengths. The proposed structure could be reusable for DCT of lengths 4, 8, 16, and 32 with a throughput of 32 DCT coefficients per cycle irrespective of the transform size. Moreover, we propose power-efficient structures for folded and full-parallel implementations of 2-D DCT. The Discrete Cosine Transform (DCT) plays a vital role in video compression due to its near-optimal decorrelation efficiency

Keywords: Discrete Cosine Transform (DCT), H.265, High Efficiency Video Coding (HEVC), Integer Discrete Cosine Trans-Form (DCT), Video Coding.

I. INTRODUCTION

The Discrete Cosine Transform (DCT) plays a vital role in video compression due to its near-optimal decorrelation efficiency [1]. Several variations of integer DCT have been suggested in the last two decades to reduce the computational complexity [2]–[6]. The new H.265/High Efficiency Video Coding (HEVC) standard [7] has been recently finalized and poised to replace H.264/AVC [8]. Some hardware architectures for the integer DCT for HEVC have also been proposed for its real-time implementation. Ahmed et al. [9] decomposed the DCT matrices into sparse sub-matrices where the multiplications are avoided by using the lifting scheme. Shen et al. [10] used the multiplier less multiple constant multiplication (MCM) approach for four-point and eight-point DCT, and have used the normal multipliers with sharing techniques for 16 and 32-point DCTs. Park et al. [11] have used Chen's factorization of DCT where the butterfly operation has been implemented by the processing element with only shifters, adders, and multiplexors. Budagavi and Sze [12] proposed a unified structure to be used for forward as well as inverse transform after the matrix decomposition. One key feature of HEVC is that it supports DCT of different sizes such as 4, 8, 16, and 32. Therefore, the hardware architecture should be flexible enough for the computation of DCT of any of these lengths.

The existing designs for conventional DCT based on constant matrix multiplication (CMM) and MCM can provide optimal solutions for the computation of any of these lengths, but they are not reusable for any length to support the same throughput processing of DCT of different transform lengths. Considering this issue, we have analyzed the possible implementations of integer DCT for HEVC in the context of resource requirement and reusability, and based on that, we have derived the proposed algorithm for hardware implementation. We have designed scalable and reusable architectures for 1-D and 2-D integer DCTs for HEVC that could be reused for any of the prescribed lengths with the same throughput of processing irrespective of transform size.

In the next section, we present algorithms for hardware implementation of the HEVC integer DCTs of different lengths 4, 8, 16, and 32. In Section III, we illustrate the design of the proposed architecture for the implementation of four-point and eight-point integer DCT along with a generalized design of integer DCT of length N , which could be used for the DCT of length N = 16 and 32. Moreover, we demonstrate the reusability of the proposed solutions in this section. In Section IV, we propose power-efficient designs of transposition buffers for full-parallel and folded implementations of 2-D Integer DCT. In Section V the simulation results of the proposed architectures for HEVC are discussed.

II. ALGORITHM FOR HARDWARE IMPLEMENTATION OF INTEGER DCT FOR HEVC

In the Joint Collaborative Team-Video Coding (JCT-VC), which manages the standardization of HEVC, Core Experiment 10 (CE10) studied the design of core transforms over several meeting cycles [13]. The eventual HEVC transform design [14] involves coefficients of 8-bit size, but does not allow full factorization unlike other competing proposals [13]. It however allows for both matrix multiplication and partial butterfly implementation. In this section, we have used the partial-butterfly algorithm of [14] for the computation of integer DCT along with its efficient algorithmic transformation for hardware implementation.

A. Key Features of Integer DCT for HEVC

The N -point integer DCT[1] for HEVC given by [14] can be computed by a partial butterfly approach using a (N/2)-point DCT and a matrix–vector product of (N/2) × (N/2) matrix with an (N/2)-point vector as:

$$\begin{bmatrix} y(0) \\ y(2) \\ \cdot \\ \cdot \\ \cdot \\ y(N-4) \\ y(N-2) \end{bmatrix} = \mathbf{C}_{N/2} \begin{bmatrix} a(0) \\ a(1) \\ \cdot \\ \cdot \\ \cdot \\ a(N/2-2) \\ a(N/2-1) \end{bmatrix} \quad \textbf{(1a)}$$

And

$$\begin{bmatrix} y(1) \\ y(3) \\ \cdot \\ \cdot \\ \cdot \\ y(N-3) \\ y(N-1) \end{bmatrix} = \mathbf{M}_{N/2} \begin{bmatrix} b(0) \\ b(1) \\ \cdot \\ \cdot \\ \cdot \\ b(N/2-2) \\ b(N/2-1) \end{bmatrix} \quad \textbf{(1b)}$$

Where

$$a(i) = x(i) + x(N-i-1)$$
$$b(i) = x(i) - x(N-i-1) \quad \textbf{(2)}$$

for i = 0, 1, · · · ,N/2−1. X = [x(0), x(1), · · · , x(N −1)] is the input vector and Y = [y(0), y(1), · · · , y(N−1)] is N -point DCT of X. $C_{N/2}$ is (N/2)-point integer DCT kernel matrix of size (N/2) × (N/2). MN/2 is also a matrix of size (N/2) × (N/2) and its (i, j)[th] entry is defined as

$$mi,jN/2 = c_{2i}+1,jN \text{ for } 0 \le i, j \le N/2-1 \quad \textbf{(3)}$$

Where C_{2i}+1,jN is the $(2_i + 1, j)^{th}$ entry of the matrix CN. Note that (1a) could be similarly decomposed, recursively, further using CN/4 and MN/4. We have referred to the direct implementation of DCT based on (1)–(3) as the reference algorithm in the remainder of this paper.

B. Hardware Oriented Algorithm

Direct implementation of (1) requires N2/4 + MULN/2 multiplications, N2/4 + N/2 + ADDN/2 additions, and 2 shifts where MULN/2 and ADDN/2 are the number of multiplications and additions/subtractions of (N/2)-point DCT, respectively. Computation of (1) could be treated as a CMM problem [15]–[17]. Since the absolute values of the coefficients in all the rows and columns of matrix M in (1b) are identical, the CMM problem can be implemented as a set of N/2 MCMs that will result in a highly regular architecture and will have low-complexity implementation. The kernel matrices for four-, eight-, 16-, and 32-point integer DCT for HEVC are given in [14], and 4- and eight-point integer DCT are represented, respectively, as

$$\mathbf{C}_4 = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix} \quad \textbf{(4)}$$

And

$$\mathbf{C}_8 = \begin{bmatrix} 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 \\ 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\ 75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 \\ 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\ 50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 \\ 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\ 18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 \end{bmatrix} \quad \textbf{(5)}$$

Based on (1) and (2), hardware oriented algorithms for DCT computation can be derived in three stages as in Table I. For 8-, 16-, and 32-point DCT, even indexed coefficients of [y(0), y(2), y(4), · · · y(N − 2)] are computed as 4-, 8-, and 16-point DCTs of [a(0), a(1), a(2), · · · a(N/2−1)], respectively, according to (1a). In Table II, we have listed the arithmetic complexities of the reference algorithm and the MCM-based algorithm for four-, eight-, 16-, and 32-point DCT.

TABLE I: 3-STAGEs Hardware Oriented Algorithms for the Computation of 4-, 8-, 16-, and

	4-Point DCT Computation
STAGE-1	$a(i) = x(i) + x(3-i),\ b(i) = x(i) - x(3-i)$ for $i = 0$ and 1.
STAGE-2	$m_{i,0} = 9b(i) = (b(i) << 3) + b(i),\ t_{i,01} = 64a(i) = a(i) << 6,\ t_{i,02} = 83b(i) = (b(i) << 6) + (m_{i,0} << 1) + b(i),$ $t_{i,10} = 36b(i) = m_{i,9} << 2$ for $i = 0$ and 1.
STAGE-3	$y(0) = t_{0,04} + t_{1,04},\ y(2) = t_{0,04} - t_{1,04},\ y(1) = t_{0,03} + t_{1,10},\ y(3) = t_{0,10} - t_{1,03}.$
	8-Point DCT Computation
STAGE-1	$a(i) = x(i) + x(7-i),\ b(i) = x(i) - x(7-i)$ for $0 \le i \le 3$.
STAGE-2	$m_{i,0} = 9b(i) = (b(i) << 3) + b(i),\ m_{i,25} = 25b(i) = (b(i) << 4) + m_{i,9},\ t_{i,18} = 18b(i) = m_{i,9} << 1,$ $t_{i,31} = 50b(i) = m_{i,25} << 1,\ t_{i,75} = 75b(i) = t_{i,50} + m_{i,25},\ t_{i,89} = 89b(i) = (b(i) << 6) + m_{i,25}$ for $0 \le i \le 3$.
STAGE-3	$y(1) = t_{0,89} + t_{1,75} + t_{2,50} + t_{3,18},\ y(3) = t_{0,75} - t_{1,18} - t_{2,89} - t_{3,50},$ $y(5) = t_{0,50} - t_{1,89} + t_{2,18} + t_{3,75},\ y(7) = t_{0,18} - t_{1,50} + t_{2,75} - t_{3,89}.$
	16-Point DCT Computation
STAGE-1	$a(i) = x(i) + x(15-i),\ b(i) = x(i) - x(15-i)$ for $0 \le i \le 7$.
STAGE-2	$m_{i,0} = 9b(i) = (b(i) << 3) + b(i),\ t_{i,9} = 9b(i) = m_{i,0} + b(i),\ m_{i,18} = 18b(i) = t_{i,9} << 1,\ m_{i,72} = 72b(i) = t_{i,9} << 3,$ $t_{i,25} = 25b(i) = (b(i) << 4) + t_{i,9},\ t_{i,43} = 43b(i) = m_{i,18} + t_{i,25},\ t_{i,57} = 57b(i) = t_{i,25} + (b(i) << 5),$ $t_{i,71} = 71b(i) = m_{i,72} - (b(i) << 1),\ t_{i,80} = 80b(i) = m_{i,72} + m_{i,8},\ t_{i,87} = 87b(i) = (t_{i,43} << 1) + b(i),$ $t_{i,90} = 90b(i) = m_{i,72} + m_{i,18}$ for $0 \le i \le 7$.
STAGE-3	$y(1) = t_{0,80} + t_{1,87} + t_{2,80} + t_{3,70} + t_{4,57} + t_{5,43} + t_{6,25} + t_{7,9},\ y(3) = t_{0,87} + t_{1,57} + t_{2,9} - t_{3,43} - t_{4,80} - t_{5,90} - t_{6,70} - t_{7,25},$ $y(5) = t_{0,80} + t_{1,9} - t_{2,70} - t_{3,87} - t_{4,25} + t_{5,57} + t_{6,90} + t_{7,43},\ y(7) = t_{0,70} - t_{1,43} - t_{2,87} + t_{3,9} + t_{4,90} + t_{5,25} - t_{6,80} - t_{7,57},$ $y(9) = t_{0,57} - t_{1,80} - t_{2,25} + t_{3,90} - t_{4,9} - t_{5,87} + t_{6,43} + t_{7,70},\ y(11) = t_{0,43} - t_{1,90} + t_{2,57} + t_{3,25} - t_{4,87} + t_{5,70} + t_{6,9} - t_{7,80},$ $y(13) = t_{0,25} - t_{1,70} + t_{2,90} - t_{3,80} + t_{4,43} + t_{5,9} - t_{6,57} + t_{7,87},\ y(15) = t_{0,9} - t_{1,25} + t_{2,43} - t_{3,57} + t_{4,70} - t_{5,80} + t_{6,87} - t_{7,90}.$
	32-Point DCT Computation
STAGE-1	$a(i) = x(i) + x(31-i),\ b(i) = x(i) - x(31-i)$ for $0 \le i \le 15$.
STAGE-2	$m_{i,2} = 2b(i) = b(i) << 1,\ t_{i,4} = 4b(i) = b(i) << 2,\ t_{i,8} = 9b(i) = (b(i) << 3) + b(i),\ m_{i,18} = 18b(i) = t_{i,9} << 1,$ $m_{i,72} = 72b(i) = t_{i,9} << 3,\ t_{i,13} = 13b(i) = t_{i,9} + t_{i,4},\ m_{i,52} = 52b(i) = t_{i,13} << 2,\ t_{i,22} = 22b(i) = m_{i,18} + t_{i,4},$ $t_{i,31} = 31b(i) = (b(i) << 5) - b(i),\ t_{i,38} = 38b(i) = (t_{i,9} << 2) + m_{i,2},\ t_{i,46} = 46b(i) = (t_{i,22} << 1) + m_{i,2},$ $t_{i,54} = 54b(i) = m_{i,52} + m_{i,2},\ t_{i,61} = 61b(i) = (t_{i,31} << 1) - b(i),\ t_{i,67} = 67b(i) = t_{i,54} + t_{i,13},$ $t_{i,73} = 73b(i) = m_{i,72} + b(i),\ t_{i,78} = 78b(i) = m_{i,52} + (t_{i,13} << 1),\ t_{i,85} = 82b(i) = t_{i,78} + t_{i,4},$ $t_{i,88} = 85b(i) = m_{i,72} + t_{i,13},\ t_{i,84} = 88b(i) = t_{i,22} << 2,\ t_{i,90} = 90b(i) = m_{i,72} + m_{i,18}$ for $0 \le i \le 15$.
STAGE-3	$y(1) = t_{0,90} + t_{1,90} + t_{2,88} + t_{3,85} + t_{4,82} + t_{5,78} + t_{6,73} + t_{7,67} + t_{8,61} + t_{9,54} + t_{10,46} + t_{11,38} + t_{12,31} + t_{13,22} + t_{14,13} + t_{15,4},$ $y(3) = t_{0,90} + t_{1,82} + t_{2,67} + t_{3,46} + t_{4,22} - t_{5,4} - t_{6,31} - t_{7,54} - t_{8,73} - t_{9,85} - t_{10,90} - t_{11,88} - t_{12,78} - t_{13,61} - t_{14,38} - t_{15,13},$ $y(5) = t_{0,88} + t_{1,67} + t_{2,31} - t_{3,13} - t_{4,54} - t_{5,82} - t_{6,90} - t_{7,78} - t_{8,46} - t_{9,4} + t_{10,38} + t_{11,73} + t_{12,90} + t_{13,85} + t_{14,61} + t_{15,22},$ $y(7) = t_{0,85} + t_{1,46} - t_{2,13} - t_{3,67} - t_{4,90} - t_{5,73} - t_{6,22} + t_{7,38} + t_{8,82} + t_{9,88} + t_{10,54} - t_{11,4} - t_{12,61} - t_{13,90} - t_{14,78} - t_{15,31},$ $y(9) = t_{0,82} + t_{1,22} - t_{2,54} - t_{3,90} - t_{4,61} + t_{5,13} + t_{6,78} + t_{7,85} + t_{8,31} - t_{9,46} - t_{10,90} - t_{11,67} + t_{12,4} + t_{13,73} + t_{14,88} + t_{15,38},$ $y(11) = t_{0,78} - t_{1,4} - t_{2,82} + t_{3,73} + t_{4,13} - t_{5,85} + t_{6,67} - t_{7,22} - t_{8,88} + t_{9,61} + t_{10,31} + t_{11,90} + t_{12,54} - t_{13,38} - t_{14,90} - t_{15,46},$ $y(13) = t_{0,73} - t_{1,31} - t_{2,90} + t_{3,22} + t_{4,78} + t_{5,67} - t_{6,38} - t_{7,90} + t_{8,13} + t_{9,82} + t_{10,61} - t_{11,46} - t_{12,88} + t_{13,4} + t_{14,85} + t_{15,54},$ $y(15) = t_{0,67} - t_{1,54} + t_{2,78} + t_{3,38} + t_{4,85} - t_{5,22} - t_{6,90} + t_{7,4} + t_{8,90} + t_{9,13} - t_{10,88} - t_{11,31} + t_{12,82} + t_{13,46} - t_{14,73} - t_{15,61},$ $y(17) = t_{0,61} - t_{1,73} - t_{2,18} + t_{3,82} + t_{4,31} - t_{5,88} - t_{6,13} + t_{7,90} - t_{8,4} - t_{9,90} + t_{10,22} + t_{11,85} - t_{12,38} - t_{13,78} + t_{14,54} + t_{15,67},$ $y(19) = t_{0,54} - t_{1,85} - t_{2,4} + t_{3,88} - t_{4,46} - t_{5,61} + t_{6,92} + t_{7,13} - t_{8,90} + t_{9,38} + t_{10,82} - t_{11,78} - t_{12,22} + t_{13,90} - t_{14,31} - t_{15,73},$ $y(21) = t_{0,46} - t_{1,90} + t_{2,38} + t_{3,54} - t_{4,90} + t_{5,31} + t_{6,61} - t_{7,98} + t_{8,22} + t_{9,67} - t_{10,90} + t_{11,13} + t_{12,73} - t_{13,82} + t_{14,4} + t_{15,78},$ $y(23) = t_{0,38} - t_{1,88} + t_{2,73} - t_{3,4} + t_{4,67} + t_{5,90} - t_{6,46} + t_{7,31} + t_{8,85} - t_{9,78} + t_{10,13} + t_{11,61} - t_{12,90} + t_{13,54} + t_{14,22} - t_{15,82},$ $y(25) = t_{0,31} - t_{1,78} + t_{2,90} - t_{3,61} + t_{4,4} + t_{5,54} - t_{6,88} + t_{7,82} - t_{8,38} - t_{9,22} + t_{10,73} - t_{11,90} + t_{12,67} - t_{13,13} - t_{14,46} + t_{15,85},$ $y(27) = t_{0,22} - t_{1,61} + t_{2,85} - t_{3,90} + t_{4,73} - t_{5,38} - t_{6,4} + t_{7,46} - t_{8,78} + t_{9,90} - t_{10,82} + t_{11,54} - t_{12,13} + t_{13,31} + t_{14,67} - t_{15,88},$ $y(29) = t_{0,13} - t_{1,38} + t_{2,61} - t_{3,78} + t_{4,88} - t_{5,90} + t_{6,85} - t_{7,73} + t_{8,54} - t_{9,31} + t_{10,4} + t_{11,22} - t_{12,46} + t_{13,67} - t_{14,82} + t_{15,90},$ $y(31) = t_{0,4} - t_{1,13} + t_{2,22} - t_{3,31} + t_{4,38} - t_{5,46} + t_{6,54} - t_{7,61} + t_{8,67} - t_{9,73} + t_{10,78} - t_{11,82} + t_{12,85} - t_{13,88} + t_{14,90} - t_{15,90}.$

III. PROPOSED ARCHITECTURES FOR INTEGER DCT COMPUTATION

In this section, we present the proposed architecture for the computation of integer DCT of length 4. We alsopresent a generalized architecture for integer DCT of higherlengths.

A. Proposed Architecture for Four-Point Integer DCT

The proposed architecture for four-point integer DCT is shown in Fig.1(a). It consists of an input adder unit (IAU), a shift-add unit (SAU), and an output adder unit (OAU). The IAU computes a(0), a(1), b(0), and b(1) according to Stage-1 of the algorithm as described in Table I. The computations of t_i, 36 and t_i, 83 are performed by two SAUs according to Stage-2 of the algorithm. The computation of t_0, 64 and t_1, 64 does not consume any logic since the shift operations could be rewired in hardware. The structure of SAU is shown in Fig. 1(b). Outputs of the SAU are finally added by the OAU according to Stage-3 of the algorithm.

B. Proposed Architecture for Integer DCT of Length 8 and Higher Length DCTs

The generalized architecture for N-point integer DCT based on the proposed algorithm is shown in Fig.2. It consists of four units, namely the IAU, (N/2)-point integer DCT unit, SAU, and OAU. The IAU computes a(i) and b(i) for i = 0, 1, ..., N/2 − 1 according to Stage-1 of the algorithm of Section II-B. The SAU provides the result of multiplication of input sample with DCT coefficient by Stage-2 of the algorithm. Finally, the OAU generates the output of DCT from a binary adder tree of $\log_2 N - 1$ stages. Fig.3(a)–(c), respectively, illustrates the structures of IAU, SAU, and OAU in the case of eight-point integer DCT. Four SAUs are required to compute $t_{i,89}$, $t_{i,75}$, $t_{i,50}$, and $t_{i,18}$ for i = 0, 1, 2, and 3 according to Stage-2 of the algorithm. The outputs of SAUs are finally added by two-stage adder tree according to Stage-3 of the algorithm. Structures for 16- and 32-point integer DCT can also be obtained similarly.

C. Reusable Architecture for Integer DCT

The proposed reusable architecture for the implementation of DCT of any of the prescribed lengths is shown in Fig. 4(a). There are two (N/2)-point DCT units in the structure. The input to one (N/2)-point DCT unit is fed through (N/2) 2:1 MUXes that selects either [a(0), ..., a(N/2 − 1)] or [x(0), ..., x(N/2 − 1)], depending on whether it is used for N -point DCT computation or for the DCT of a lower size. The other (N/2)-point DCT unit takes the input [x(N/2), ...,x(N − 1)] when it is used for the computation of DCT of N/2 point or a lower size, otherwise, the input is reset by an array of (N/2) AND gates to disable this (N/2)-point DCT unit. The output of this (N/2)-point DCT unit is multiplexed with that of the OAU, which is preceded by the SAUs and IAU of the structure. The N AND gates before IAU are used to disable the IAU, SAU, and OAU when the architecture is used to compute (N/2)-point DCT computation or a lower size. The input of the control unit, m_N is used to decide the size of DCT computation. Specifically, for N = 32, m_{32} is a 2-bits signal that is set to {00}, {01}, {10}, and {11} to compute four-, eight-, 16-, and 32-point DCT, respectively.

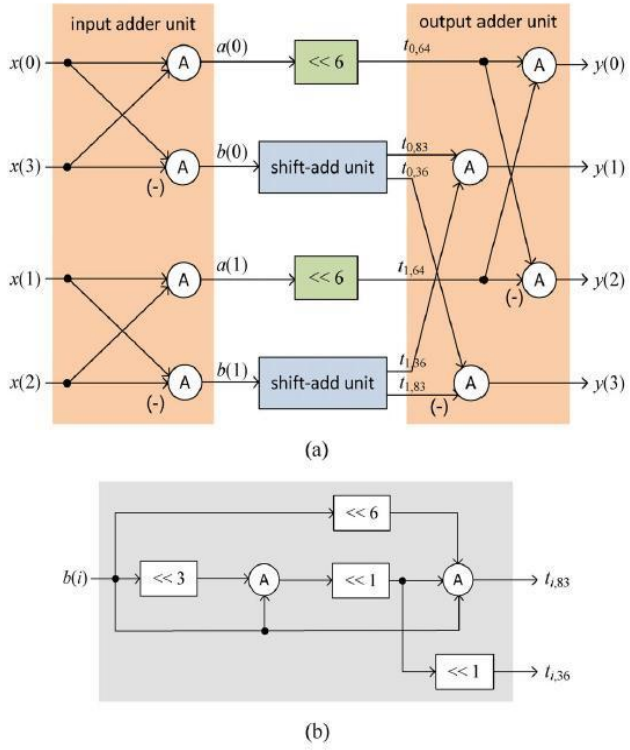

Fig. 1. Proposed architecture of four-point integer DCT. (a) Four-point DCT architecture. (b) Structure of SAU.

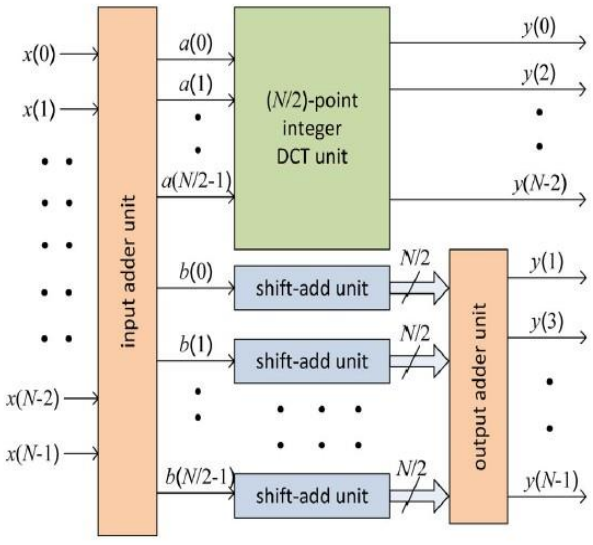

Fig. 2. Proposed generalized architecture for integer DCT of lengths N = 8,16, and 32.

The control unit generates sel 1 and sel 2, where sel 1 is used as control signals of N MUXes and input of N AND gates before IAU. sel 2 is used as the input $m_{(N/2)}$ to two lower size reusable integer DCT units in a recursive manner. The combinational logics for control units are 4(b) and (c) for N = 16 and 32, respectively. For N = 8, m_8

is a 1-bit signal that is used as sel 1 while sel 2 is not required since four-point DCT is the smallest DCT. The proposed structure can compute one 32-point DCT, two 16-point DCTs, four eight-point DCTs, and eight four-point DCTs, while the throughput remains the same as 32 DCT coefficients per cycle irrespective of the desired transform size.

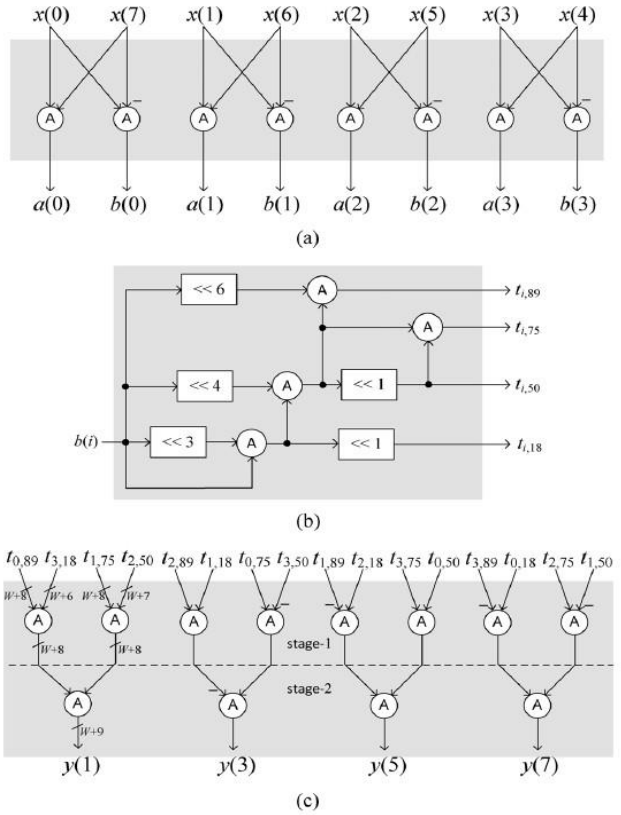

Fig. 3. Proposed architecture of eight-point integer DCT and IDCT. (a) Structure of IAU. (b) Structure of SAU. (c) Structure of OAU.

IV. PROPOSED STRUCTURE FOR 2-D INTEGER DCT

Using its separable property, an (N × N)-point 2-D integer DCT could be computed by the row-column decomposition technique in two distinct stages.

- **Stage 1:** N -point 1-D integer DCT is computed for each column of the input matrix of size (N × N) to generate an intermediate output matrix of size (N × N).
- **Stage 2:** N -point 1-D DCT is computed for each row of the intermediate output matrix of size (N × N) to generate desired 2-D DCT of size (N × N).

We present here a folded architecture and full-parallel architecture for the 2-D integer DCT, along with the necessary transposition buffer to match them without internal data movement.

A. Folded Structure for 2-D Integer DCT

The folded structure for the computation of (N × N)-point 2-D integer DCT is shown in Fig. 5(a). It consists of one N -point 1-D DCT module and a transposition buffer. The structure of the proposed 4 × 4 transposition buffer is shown in Fig. 5(b). It consists of 16 registers arranged in four rows and four columns. (N × N) transposition buffer can store N values in any one column of registers by enabling them by one of the enable signals EN_i for i = 0, 1, · · ·, N − 1. One can select the content of one of the rows of registers through the MUXes. During the first N successive cycles, the DCT module receives the successive columns of (N ×N) block of input for the computation of STAGE-1,

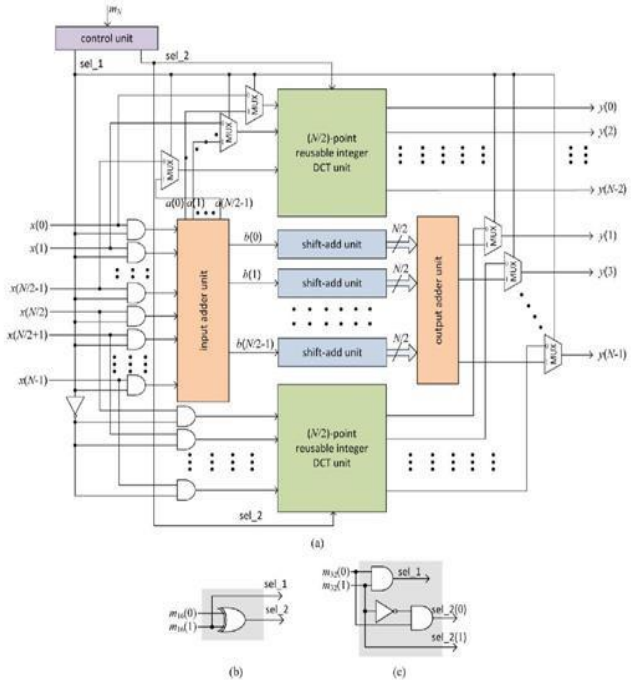

Fig. 4. Proposed reusable architecture of integer DCT. (a) Proposed reusable architecture for N = 8, 16, and 32. (b) Control unit for N = 16. (c) Control unit for N = 32.

registers of successive columns in the transposition buffer. In the next N cycles, contents of successive rows of the transposition buffer are selected by the MUXes and fed as input to the 1-D DCT module. N MUXes are used at the input of the 1-D DCT module to select either the columns from the input buffer (during the first N cycles) or the rows from the transposition buffer (during the next N cycles).

B. Full-Parallel Structure for 2-D Integer DCT

The full-parallel structure for (N × N)-point 2-D integer DCT is shown in Fig. 6(a). It consists of two N -point 1-D DCT modules and a transposition buffer. The structure of the 4 × 4 transposition buffer for full-parallel structure is shown in Fig.6(b). It consists of 16 register cells (RC) [shown in Fig.6(c)] arranged in four rows and four columns. N × N transposition buffer can store N values in a cycle either row-wise or column-wise by selecting the inputs by the MUXes at the input of RCs. The output from RCs can also be collected either row-wise or column-wise. To read the output from the buffer, N number of (2N − 1):1 MUXes [shown in Fig. 6(d)] are used, where outputs of the ith row and the ith column of RCs are fed as input to the ith MUX. For the first N successive cycles, the ith MUX provides

output of N successive RCs on the ith row. In the next N successive cycles, the ith MUX provides output of N successive RCs on the ith column. By this arrangement, in the first N cycles, we can read the output of N successive columns of RCs and in the next N cycles, we can read the output of N successive rows of RCs. The transposition buffer in this case allows both read and write operations concurrently. If for the N cycles,

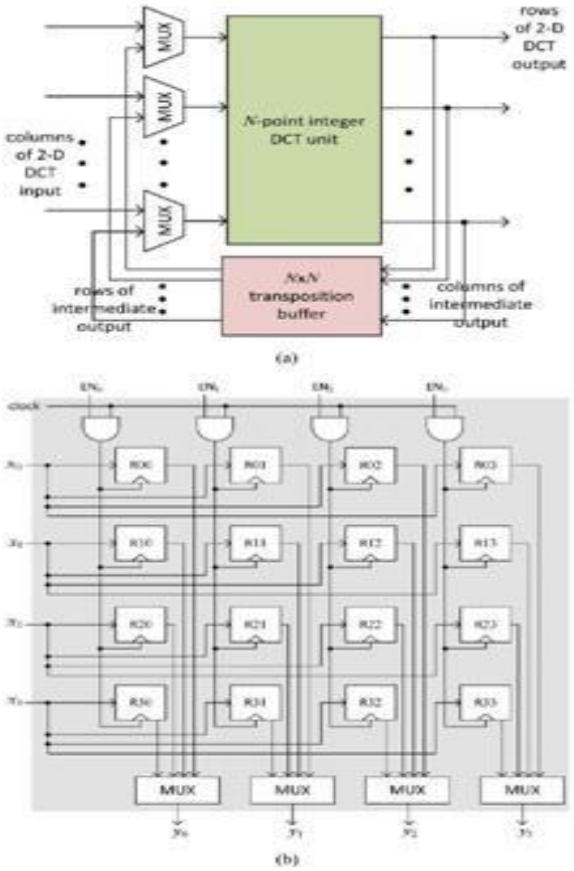

Fig.5. Folded structure of (N ×N)-point 2-D integer DCT. (a) Folded 2-D DCT architecture. (b) Structure of the transposition buffer for input size 4×4.

results are read and stored column-wise now, then in the next N successive cycles, results are read and stored in the transposition buffer row-wise. The first 1-D DCT module receives the inputs column-wise from the input buffer. It computes a column of intermediate output and stores in the transposition buffer. The second 1-D DCT module receives the rows of the intermediate result from the transposition buffer and computes the rows of 2-D DCT output row-wise. Suppose that in the first N cycles, the intermediate results are stored column-wise and all the columns are filled in with intermediated results, then in the next N cycles, contents of successive rows of the transposition buffer are selected by the MUXes and fed as input to the 1-D DCT module of the second stage. During this period, the output of the 1-D DCT module of first stage is stored row-wise. In the next N cycles, results are read and written column-wise. The alternating column-wise and row-wise read and write operations with the transposition buffer continues. The transposition buffer in this case introduces a pipeline latency of N cycles required to fill in the transposition buffer for the first time.

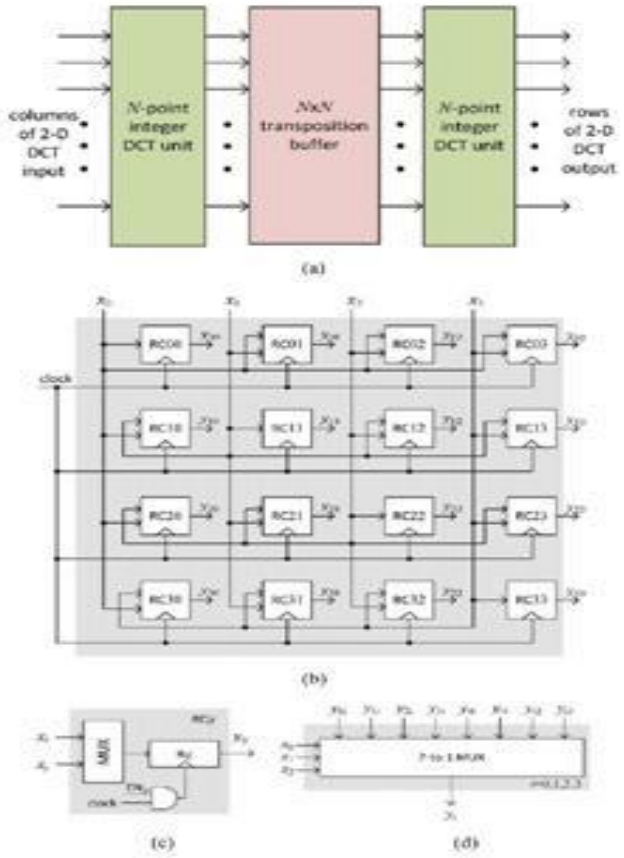

Fig.6. Full-parallel structure of (N × N)-point 2-D integer DCT. (a) Fullparallel 2-D DCT architecture. (b) Structure of the transposition buffer for input size 4×4. (c) Register cell RCij. (d) 7-to-1 MUX for 4×4 transposition buffer.

V. IMPLEMENTATION RESULTS

A. Simulation Results of 1-D Integer DCT

We have coded the architecture derived from the reference algorithm of Section II as well as the proposed architectures for different transform lengths in Verilog, and simulated by using Xilinx. The word length of input samples is chosen to be 16 bits.

B. Simulation Results of 2-D Integer DCT

We also simulated the folded and full-parallel structures for 2-D integer DCT. The 2-D full-parallel structure yields 32 samples in each cycle after initial latency of 32 cycles providing double the throughput of the folded structure. However, the full-parallel architecture consumes more power than the folded architecture since it has two 1-D DCT units and nearly the same complexity of transposition buffer while the throughput of full-parallel design is double the throughput of folded design.

VI. SUMMARY AND CONCLUSION

In this paper, we have proposed area- and power-efficient architectures for the implementation of DCT of different lengths to be used in HEVC. The computation of N -point 1-D DCT involves an (N/2)-point 1-D DCT and a vector-matrix multiplication with a constant matrix of size (N/2) × (N/2). We have used the proposed architecture to derive a reusable architecture for DCT that can compute the DCT of lengths 4, 8, 16, and 32 with throughput of 32 output coefficients per cycle. We have proposed power-efficient architectures for folded and full-parallel implementations of 2-D DCT, where no data movement takes place within the transposition buffer.

VII. REFERENCES

[1] N. Ahmed, T. Natarajan, and K. Rao, "Discrete cosine transform," IEEE Trans. Comput., vol. 100, no. 1, pp. 90–93, Jan. 1974.

[2] W. Cham and Y. Chan, "An order-16 integer cosine transform," IEEE Trans. Signal Process., vol. 39, no. 5, pp. 1205–1208, May 1991.

[3] Y. Chen, S. Oraintara, and T. Nguyen, "Video compression using integer DCT," in Proc. IEEE Int. Conf. Image Process., Sep. 2000, pp. 844–845.

[4] J. Wu and Y. Li, "A new type of integer DCT transform radix and its rapid algorithm," in Proc. Int. Conf. Electric Inform. Control Eng., Apr. 2011, pp. 1063–1066.

[5] A. M. Ahmed and D. D. Day, "Comparison between the cosine and Hartley based naturalness preserving transforms for image watermarking and data hiding," in Proc. First Canad. Conf. Comput. Robot Vision, May 2004, pp. 247–251.

[6] M. N. Haggag, M. El-Sharkawy, and G. Fahmy, "Efficient fast multiplication-free integer transformation for the 2-D DCT H.265 standard," in Proc. Int. Conf. Image Process., Sep. 2010, pp. 3769–3772.

[7] B. Bross, W.-J. Han, J.-R. Ohm, G. J. Sullivan, Y.-K. Wang, and T. Wiegand, High Efficiency Video Coding (HEVC) Text Specification Draft 10 (for FDIS and Consent), JCT-VC L1003, Geneva, Switzerland, Jan. 2013.

[8] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 560–576, Jul. 2003.

[9] A. Ahmed, M. U. Shahid, and A. Rehman, "N Point DCT VLSI Architecture for Emerging HEVC Standard," in Proc. VLSI Design, vol. 2012, Article 752024, pp. 1–13, 2012.

[10] S. Shen, W. Shen, Y. Fan, and X. Zeng, "A unified 4/8/16/32-point integer IDCT architecture for multiple video coding standards," in Proc. IEEE Int. Conf. Multimedia Expo, Jul. 2012, pp. 788–793.

[11] J.-S. Park, W.-J. Nam, S.-M. Han, and S. Lee, "2-D large inverse transform (16x16, 32x32) for HEVC (high efficiency video coding),".

[12] M. Budagavi and V. Sze, "Unified forward+inverse transform architecture for HEVC," in Proc. Int. Conf. Image Process., Sep. 2012, pp. 209–212.

[13] P. Topiwala, M. Budagavi, A. Fuldseth, R. Joshi, and E. Alshina. (2011, Nov.). JCTVC-G040, CE10: Summary Report on Core Transform Design [Online]. Available: http://phenix.int-evry.fr/jct/doc end user/documents/7 Geneva/wg11/JCTVC-G040-v3.zip.

[14] A. Fuldseth, G. Bjøntegaard, M. Budagavi, and V. Sze. (2011, Nov.). JCTVC-G495, CE10: Core Transform Design for HEVC: Proposal for Current HEVC Transform [Online]. Available: http://phenix.int-evry.fr/jct/doc end user/documents/7 Geneva/wg11/JCTV%C-G495-v2.zip.

[15] M. Potkonjak, M. B. Srivastava, and A. P. Chandrakasan, "Multiple constant multiplications: Efficient and versatile framework and algorithms for exploring common subexpression elimination," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 15, no. 2, pp. 151–165, Feb. 1996.

[16] M. D. Macleod and A. G. Dempster, "Common subexpression elimination algorithm for low-cost multiplierless implementation of matrix multipliers," Electron. Lett., vol. 40, no. 11, pp. 651–652, May 2004.

[17] N. Boullis and A. Tisserand, "Some optimizations of hardware multiplication by constant matrices," IEEE trans. Comput., vol. 54, no. 10, pp. 1271–1282, Oct. 2005.

[18] F. Bossen, Common HM Test Conditions and Software Reference Configurations, JCT-VC L1100, Geneva, Switzerland, Jan. 2013.