INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 5, Issue 4 SEP 2016

VLSI DESIGN OF VEDIC MULTIPLIER BASED ARITHMETIC COSINE TRANSFORM ARCHITECTURE

M.SAIRAM REDDY[1] **G.DEEPIKA [2]**

reddy.sai101@gmail.com[1]

[1]PG Scholar, Dept of ECE, RRS College of Engineering and Technology, Patancheru, Medak, Telangana, India.

[2]Associate Professor, HOD, Dept of ECE, RRS College of Engineering and Technology, Patancheru, Medak, Telangana, India.

Abstract: In Image processing the Image compression can improve the performance of the digital systems by reducing the cost and time in image storage and transmission without significant reduction of the Image quality. This paper describes hardware architecture of low complexity Discrete Cosine Transform (DCT) architecture for image compression[6]. In this DCT architecture, common computations are identified and shared to remove redundant computations in DCT matrix operation. Vector processing is a method used for implementation of DCT. This reduction in computational complexity of 2D DCT reduces power consumption. The 2D DCT is performed on 8x8 matrix using two 1-Dimensional Discrete cosine transform blocks and a transposition memory [7]. Inverse discrete cosine transform (IDCT) is performed to obtain the image matrix and
reconstruct the original image. The proposed image compression algorithm is comprehended using MATLAB code. The VLSI design of the architecture is implemented Using Verilog HDL. The proposed hardware architecture for image compression employing DCT was synthesized using RTL complier and it was mapped using 180nm standard cells. . The Simulation is done using Modelsim. The simulation results from MATLAB and Verilog HDL are compared.

Keywords- Discrete Cosine Transform (DCT), Inverse Discrete Cosine Transform (IDCT), Joint Photographic Expert Group (JPEG), Low Power Design, Very Large Scale Integration (VLSI)

I. Introduction

Digital camera uses JPEG standard to compress the captured image data from sensors so as to reduce the storage requirements [1-5]. JPEG compresses image data in three steps. They are 8x8 block-wise discrete cosine transform (DCT), quantization and entropy coding. DCT transforms the image data from spatial domain intofrequency domain which are called DCT coefficients. Most of the visual information is stored in few low frequency DCT coefficients and they are used for further coding while high frequency coefficients are discarded. Quantization is used to bring further compression by representing DCT coefficients with no greater precision that is necessary to achieve desired image quality [6,7]. Quantized DCT coefficients are reordered in zig-zag fashion in increasing order of frequency. Finally entropy coding is done to eliminate the redundancy in quantized data representation.

DCT is a computation intensive algorithm and its direct implementation requires a large number of adders and multipliers. Distributed Arithmetic (DA) is a technique that reduces the computation for hardware implementation of digital signal processing algorithm [9]. DCT implementation using DA is done in literatures [10-12]. For the 64 DCT implementation of quantizer, which is done by dividing each coefficient by its correspondingquantizer step-size, a memory

module and a divider is required. Quantizer implemented in [15] uses16-bit multiplier and RAM, in [16] 13x10-bit multiplier and RAM are used whereas adders and shifters are used in [13] and [14]. A default quantization table is recommended by JPEG committee. But users are free to use their own quantization table for required perceptual quality and compression.

Video coding standards have evolved primarily through the development of the well-known ITU-T and ISO/IEC standards. The ITU-T produced H.261 [2] and H.263 [3], ISO/IEC produced MPEG-1 [4] and MPEG-4 Visual [5], and the two organizations jointly produced the H.262/MPEG-2 Video [6] and H.264/MPEG-4 Advanced Video Coding (AVC) [7] standards. The two standards that were jointly produced have had a particularly strong impact and have found their way into a wide variety of products that are increasingly prevalent in our daily lives. Throughout this evolution, continued efforts have been made to maximize compression capability and improve other characteristics such as data loss robustness, while considering the computational resources that were practical for use in products at the time of anticipated deployment of each standard.

The Discrete cosine transform (DCT) plays a vital role in video compression due to its near-optimal de correlation efficiency [1]. Several variations of integer DCT have been suggested in the last two decades to reduce the computational complexity. The new H.265/High Efficiency Video Coding (HEVC) standard 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. Shenet al 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.

II. Literature survey

HEVC Coding Design and Feature Highlights

The HEVC standard is designed to achieve multiple goals, including coding efficiency, ease of transport system integration and data loss resilience, as well as implementability using parallel processing architectures. The following subsections briefly describe the key elements of the design by which these goals are achieved, and

the typical encoder operation that would generate a valid bitstream.

A. Video Coding Layer

The video coding layer of HEVC employs the same hybrid approach (inter-/intrapicture prediction and 2-D transform coding) used in all video compression standards since H.261. Fig. 1 depicts the block diagram of a hybrid video encoder, which could create a bitstream conforming to the HEVC standard.

An encoding algorithm producing an HEVC compliant bit stream would typically proceed as follows. Each picture is split into block-shaped regions, with the exact block partitioning being conveyed to the decoder. The first picture of a video sequence (and the first picture at each clean random access point into a video sequence) is coded using only intra picture prediction (that uses some prediction of data spatially from region-to-region within the same picture, but has no dependence on other pictures). For all remaining pictures of a sequence or between random access points, interpicture temporally predictive coding modes are typically used for most blocks. The encoding process for inter picture prediction consists of choosing motion data comprising the selected reference picture and motion vector (MV) to be applied for predicting the samples of each block. The encoder and decoder generate identical inter picture prediction signals by applying motion compensation (MC) using the MV and mode decision data, which are transmitted as side information.

The residual signal of the intra- or interpicture prediction, which is the difference between the original block and its prediction, is transformed by a linear spatial transform. The transform coefficients are then scaled, quantized, entropy coded, and transmitted together with the prediction information. The encoder duplicates the decoder processing loop (see gray-shaded boxes in Fig. 1) such that both will generate identical predictions for subsequent data. Therefore, the quantized transform coefficients are constructed by inverse scaling and are then inverse transformed to duplicate the decoded approximation of the residual signal. The residual is then added to the prediction, and the result of that addition may then be fed into one or two loop filters to smooth out artifacts induced by block-wise processing and quantization. The final picture representation (that is a duplicate of the output of the decoder) is stored in a decoded picture buffer to be used for the prediction of subsequent pictures. In general, the order of encoding or decoding processing of pictures often differs from the order in which they arrive from the source; necessitating a distinction between the decoding order (i.e., bitstream order) and the output order (i.e., display order) for a decoder.

Video material to be encoded by HEVC is generally expected to be input as progressive scan imagery (either due to the source video originating in that format or resulting from deinterlacing prior to encoding). No explicit coding features are present in the HEVC design to support the use of interlaced scanning, as interlaced scanning is no longer used for displays and is becoming substantially less common for distribution. However, a metadata syntax has been provided in HEVC to allow an encoder to indicate that interlace-scanned video has been sent by coding each field (i.e., the even or odd numbered
lines of each video frame) of interlaced video as a separate picture or that it has been sent by coding each interlaced frame as an HEVC coded picture. This provides an efficient method of coding interlaced video without burdening decoders with a need to support a special decoding process for it.

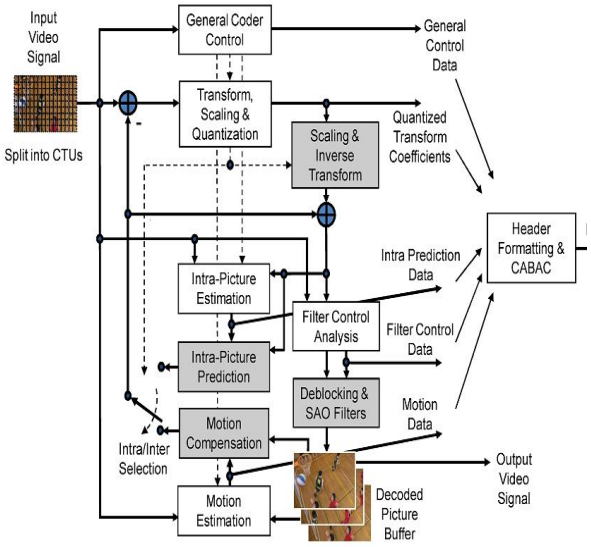


Fig. 1. Typical HEVC video encoder (with decoder modeling elements shaded in light gray).

III. 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. The eventual HEVC transform design involves coefficients of 8-bit size, but does not allow full factorization unlike other competing proposals. It however allows for both matrix multiplication and partial butterfly implementation. In this section, we have used the partial-butterfly algorithm of for the computation of integer DCT along with its efficient algorithmic transformation for hardware implementation.

A. Key Features of Integer DCT for HEVC

TheN-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

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

and

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

where

$$a(i) = x(i) + x(N-i-1)$$
$$b(i) = x(i) - x(N-i-1)$$

for i=0,1,⋯,N/2−1.X=[x(0),x(1),⋯,x(N−1)] is the input vector andY=[y(0),y(1),⋯,y(N−1)] isN-point DCT of X. CN/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

$$m_{N/2}^{i,j} = c_N^{2i+1,j} \text{ for } 0 \le i, j \le N/2-1$$

Where $C^{2i+1,j}_N$ is the (2i +1,j)th entry of the matrix C_N.Note that (1a) could be similarly decomposed, recursively, further using $C_{N/4}$ and $M_{N/4}$.

B. Hardware Oriented Algorithm

Direct implementation of (1) requires$N^2/4$ + $MUL_{N/2}$ multiplications,N^2 /4+N/2 + $ADD_{N/2}$ additions, and 2 shifts where MULN/2and ADDN/2are 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 Min (1b) are identical, the CMM problem can be implemented as a set ofN/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 eightpoint integer DCT are represented, respectively, as

$$C_4 = \begin{bmatrix} 64 & 64 & 64 & 64 \\ 83 & 36 & -36 & -83 \\ 64 & -64 & -64 & 64 \\ 36 & -83 & 83 & -36 \end{bmatrix}$$

and

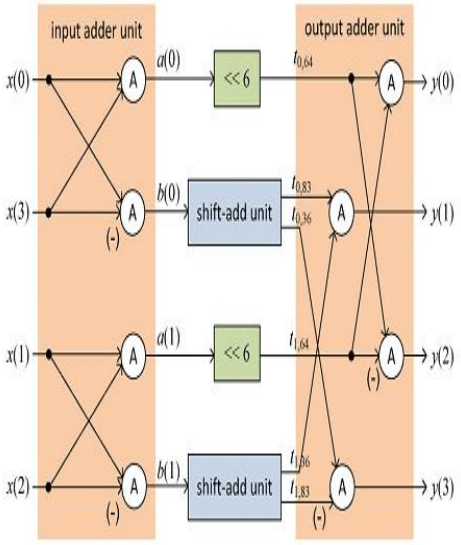
$$C_8 = \begin{bmatrix} 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\ 89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 \\ 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\ 75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 \\ 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\ 50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 \\ 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\ 18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 \end{bmatrix}.$$

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. Algorithms for Inverse DCT (IDCT) can also be derived in a similar way.

Proposed Architectures for Integer DCT Computation:

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 computesa(0),a(1),b(0), andb(1) according to STAGE-1 of the algorithm as described in Table I. The computations of ti,36 and ti,83 are performed by two SAUs according to STAGE-2 of the algorithm. The computation oft0,64 andt1,64 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.



(a)

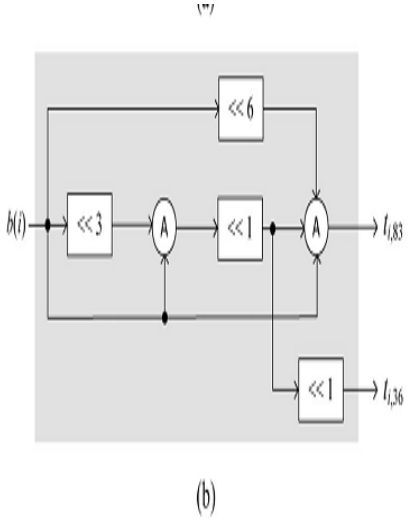


Fig. 1. Proposed architecture of four-point integer DCT. (a) Four-point DCT architecture. (b) Structure of SAU.

B. Proposed Architecture for Integer DCT of Length8and 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 computesa(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 2N−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 ti,89, ti,75, ti,50, and ti,18 for i =0,1,2,and3 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.

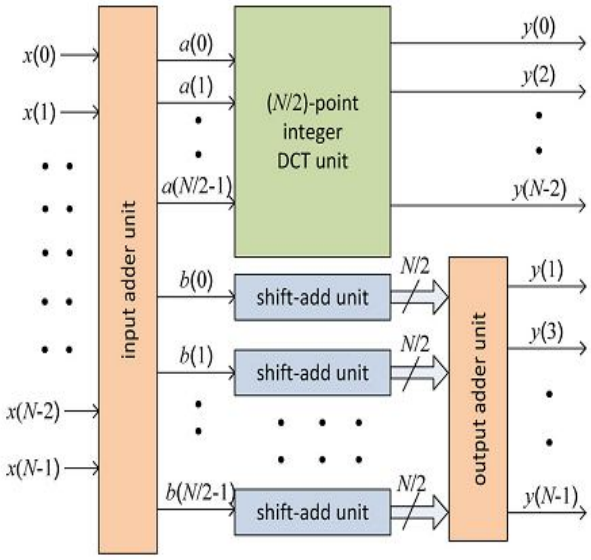
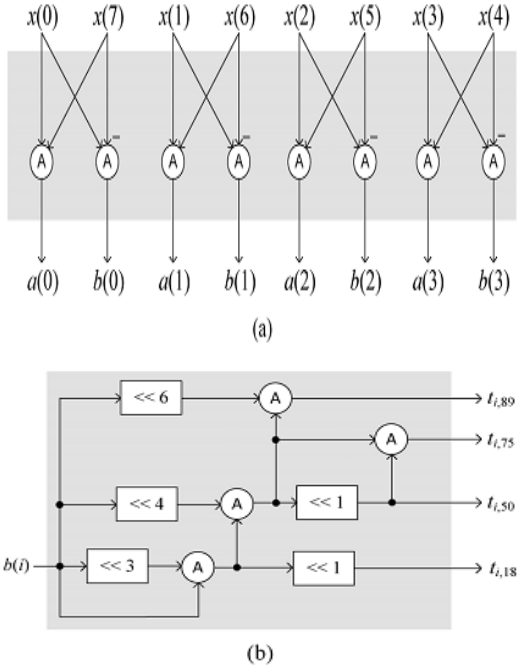


Fig. 2. Proposed generalized architecture for integer DCT of lengths N=8, 16, and 32.



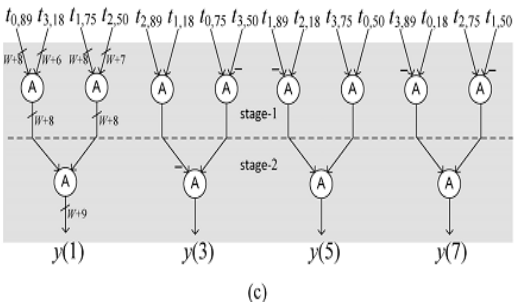


Fig. 3. Proposed architecture of eight-point integer DCT and IDCT. (a) Structure of IAU. (b) Structure of SAU. (c) Structure of OAU

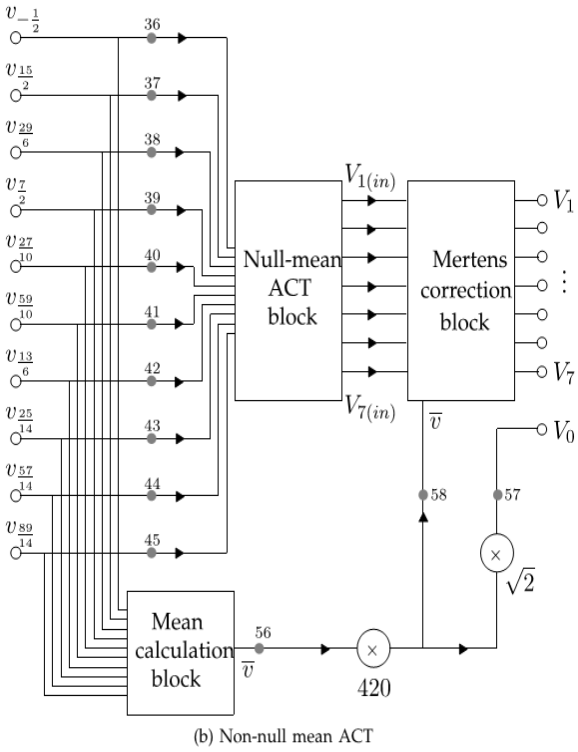


Fig 4 Non Null mean ACT

Integer constant multiplications are implemented as shift-and-add structures, therefore are not counted as multipliers. Note that the adder count also include the adders required for the Booth encoded structures

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 NAND 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,mNis used to decide the size of DCT computation. Specifically, for N=32,m32 is a 2-bits signal that is set to{00}, {01}, {10}, and {11}to compute four-, eight-, 16-, and 32-point DCT, respectively. The control unit generates sel 1 and sel 2, where sel 1 is used as control signals ofNMUXes and input ofNAND gates before IAU. sel 2 is used as the inputm(N/2) to two lower size reusable integer DCT units in a recursive manner. The combinational logics for control units are shown in Fig. 4(b) and (c) for N= 16 and 32, respectively. ForN=8,m8 is a 1-bit signal that is used as sel 1 while sel 2 is not required since fourpoint DCT is the smallest DCT. The proposed structure can compute one 32-point DCT, two 16-point DCTs, four eightpoint DCTs, and eight four-point DCTs, while the throughput remains the same as 32 DCT coefficients per cycle irrespectiveof the desired transform size..

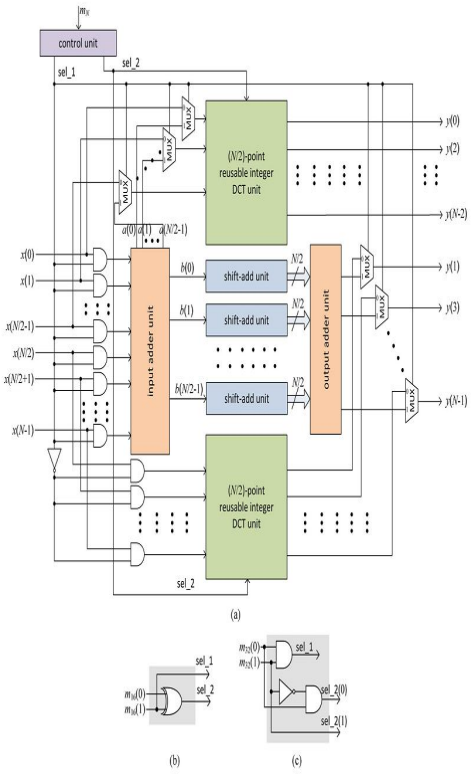


Fig. 5. Proposed reusable architecture of integer DCT. (a) Proposed reusable architecture for N= 8, 16, and 32. (b) Control unit for N= 16. (c) Control unit for N=32

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

Folded Structure for2-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 ENi fori=0,1,⋯,N−1. One can select the content of one of the rows of registers through the MUXes. During the firstNsuccessive cycles, the DCT module receives the successive columns of (N×N) block of input for the computation of STAGE-1, and stores the intermediate results in the registers of successive columns in the transposition buffer. In the nextNcycles, contents of successive rows of the transposition buffer are selected by the MUXes and fed as input to the 1-D DCT module.NMUXes are used at the input of the 1-D DCT module to select either the columns from the input buffer (during the first Ncycles) or the rows from the transposition buffer (during the next Ncycles).

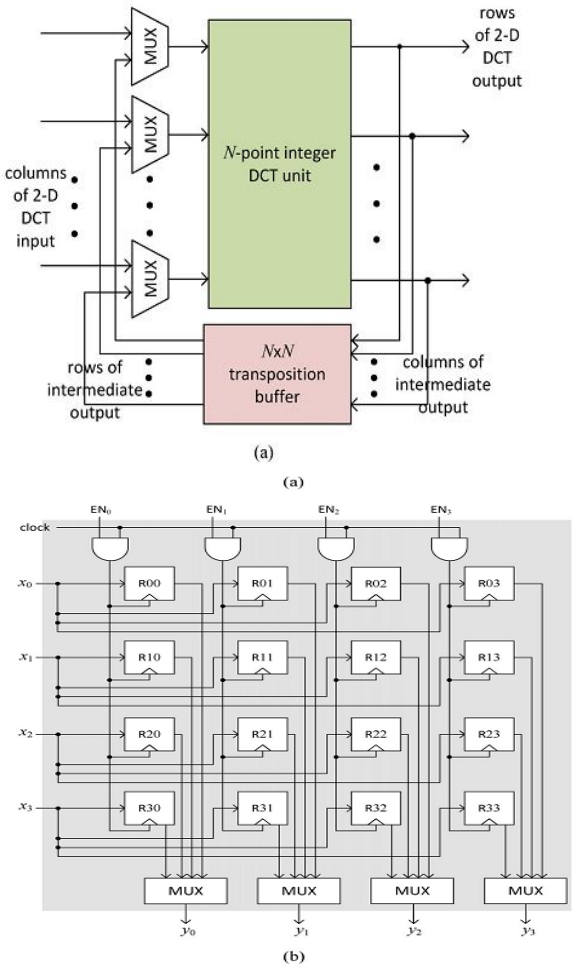


Fig. 6. Folded structure of (N×N)-point 2-D integer DCT. (a) Folded 2-D DCT architecture. (b) Structure of the transposition buffer for input size 4×4

Full-Parallel Structure for2-D Integer DCT

The full-parallel structure for (N×N)-point 2-D integer DCT is shown in Fig. 6(a). It consists of twoN-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 Nvalues in a cycle either rowwise 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,Nnumber of (2N−1):1 MUXes [shown in Fig. 6(d)] are used, where outputs of theith row and the ith column of RCs are fed as input to theith MUX. For the firstNsuccessive cycles, theith MUX provides output of Nsuccessive RCs on the ith row. In the next Nsuccessive cycles, the ith MUX provides output ofNsuccessive RCs on the ith column. By this arrangement, in the first Ncycles, we can read the output of Nsuccessive columns of RCs and in the next Ncycles,

we can read the output ofNsuccessive rows of RCs. The transposition buffer in this case allows both read and write operations concurrently. If for theNcycles, results are read and stored column-wise now, then in the nextNsuccessive cycles, results are read and stored in the transposition buffer row-wise. The first 1-D DCT module receives the inputscolumn-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 Ncycles, 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 nextNcycles, 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 ofNcycles required to fill in the transposition buffer for the first time.

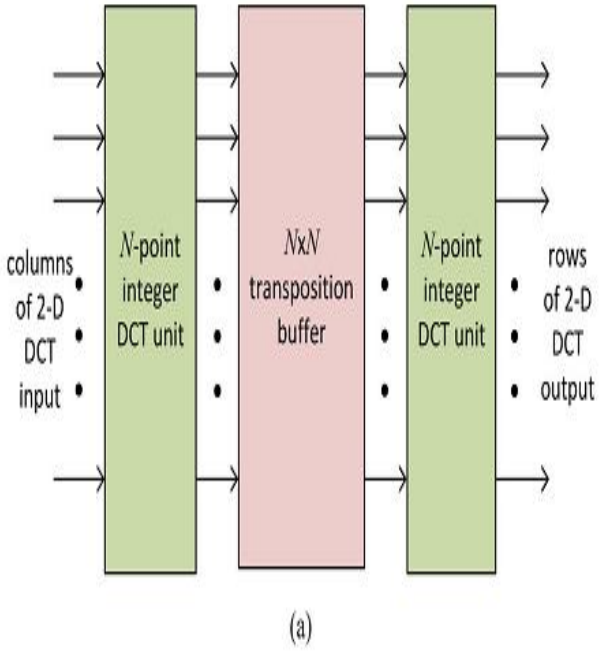


Fig. 7. Full-parallel structure of (N×N)-point 2-D integer DCT. (a) Fullparallel 2-D DCT architecture.

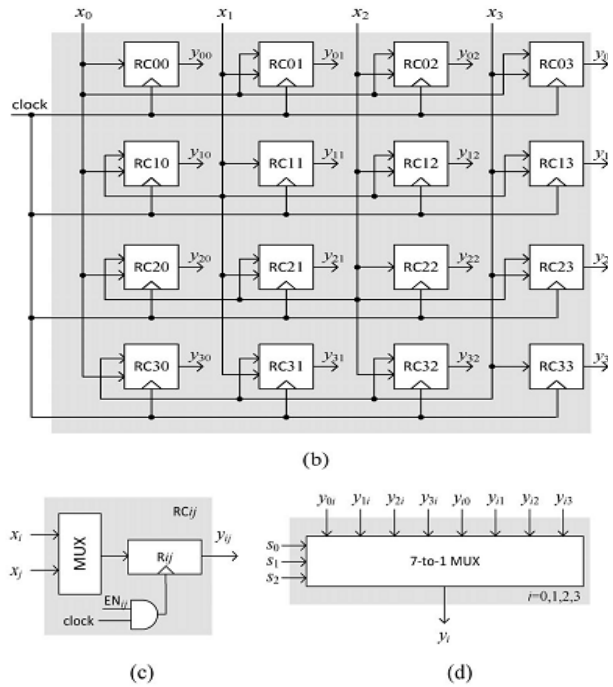
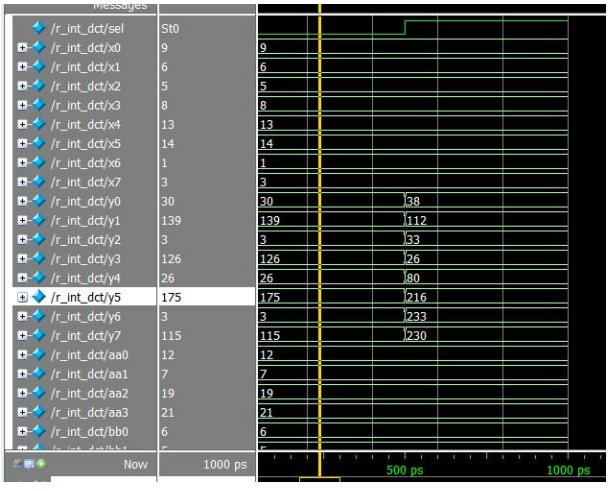


Fig.8. Full-parallel structure of (N×N)-point 2-D integer DCT. (b) Structure of the transposition buffer for input size 4×4. (c) Register cell RCij. (d) 7-to-1 MUX for 4×4 transposition buffer.

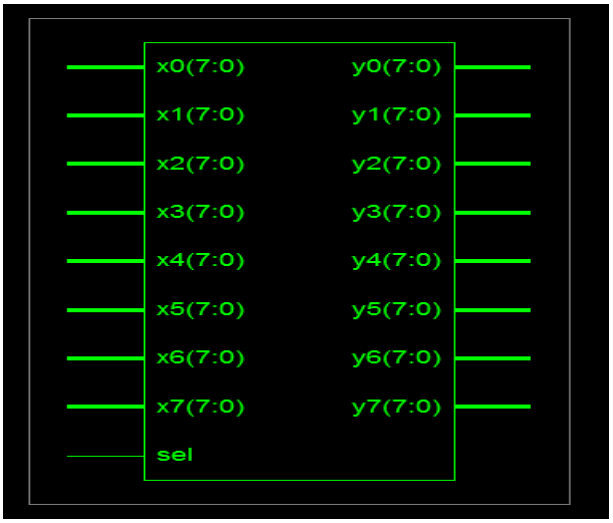
IV. RESULTS

Simulation Results:

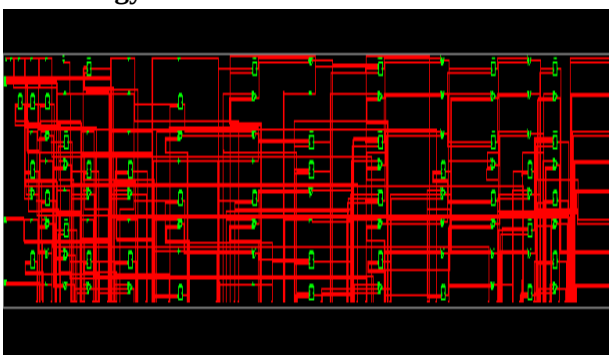


Synthesis Results:

RTL Schematic:



Technology Schematic:



V. CONCLUSION

In this paper, we presented a very low-complexity DCT approximation obtained via pruning. The resulting approximate transform requires only 10 additions and possesses performance metrics comparable with state-of-the-art methods, By means of computational simulation, VLSI hardware realizations, and a full HECV implementation, we demonstrated the practical relevance of our method as an image and video codec.

REFERENCES

[1] B. Bross, W.-J. Han, G. J. Sullivan, J.-R. Ohm, and T. Wiegand, High Efficiency Video

Coding (HEVC) Text Specification Draft 9, document JCTVC-K1003, ITU-T/ISO/IEC Joint Collaborative Team on Video Coding (JCT-VC), Oct. 2012.
[2] Video Codec for Audiovisual Services at px64 kbit/s, ITU-T Rec. H.261, version 1: Nov. 1990, version 2: Mar. 1993.
[3] Video Coding for Low Bit Rate Communication, ITU-T Rec. H.263, Nov. 1995 (and subsequent editions).
[4] Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s—Part 2: Video, ISO/IEC 11172-2 (MPEG-1), ISO/IEC JTC 1, 1993.
[5] Coding of Audio-Visual Objects—Part 2: Visual, ISO/IEC 14496-2 (MPEG-4 Visual version 1), ISO/IEC JTC 1, Apr. 1999 (and subsequent editions).
[6] Generic Coding of Moving Pictures and Associated Audio Information— Part 2: Video, ITU-T Rec. H.262 and ISO/IEC 13818-2 (MPEG 2 Video), ITU-T and ISO/IEC JTC 1, Nov. 1994.
[7] Advanced Video Coding for Generic Audio-Visual Services, ITU-T Rec. H.264 and ISO/IEC 14496-10 (AVC), ITU-T and ISO/IEC JTC 1, May2003 (and subsequent editions).
[8] H. Samet, "The quadtree and related hierarchical data structures," Comput. Survey, vol. 16, no. 2, pp. 187–260, Jun. 1984.
[9] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, 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.
[10] S. Wenger, "H.264/AVC over IP," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 645–656, Jul. 2003.
[11] T. Stockhammer, M. M. Hannuksela, and T. Wiegand, "H.264/AVC in wireless environments,"IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 657–673, Jul. 2003.
[12] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H.264/AVC standard,"IEEE Trans. Circuits Syst. Video Technol., vol. 17, no. 9, pp. 1103–1120, Sep. 2007.
[13] D. Marpe, H. Schwarz, and T. Wiegand, "Context-adaptive binary arithmetic coding in the H.264/AVC video compression standard,"IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 620–636, Jul. 2003.
[14] G. J. Sullivan,Meeting Report for 26th VCEG Meeting, ITU-T SG16/Q6 document VCEG-Z01, Apr. 2005.
[15] Call for Evidence on High-Performance Video Coding (HVC), MPEG
document N10553, ISO/IEC JTC 1/SC 29/WG 11, Apr. 2009.