IEICE TRANS. FUNDAMENTALS, VOL.E96–A, NO.7 JULY 2013

PAPER

A Unified Forward/Inverse Transform Architecture for Multi-Standard Video Codec Design

Sha SHEN[†], **Weiwei SHEN**[†], *Nonmembers*, **Yibo FAN**[†a)], *Member*, *and* **Xiaoyang ZENG**[†], *Nonmember*

SUMMARY This paper describes a unified VLSI architecture which can be applied to various types of transforms used in MPEG-2/4, H.264, VC-1, AVS and the emerging new video coding standard named HEVC (High Efficiency Video Coding). A novel design named configurable butterfly array (CBA) is also proposed to support both the forward transform and the inverse transform in this unified architecture. Hadamard transform or 4/8-point DCT/IDCT are used in traditional video coding standards while 16/32-point DCT/IDCT are newly introduced in HEVC. The proposed architecture can support all these transform types in a unified architecture. Two levels (architecture level and block level) of hardware sharing are adopted in this design. In the architecture level, the forward transform can share the hardware resource with the inverse transform. In the block level, the hardware for smaller size transform can be recursively reused by larger size transform. The multiplications of 4 or 8-point transform are implemented with Multiplierless MCM (Multiple Constant Multiplication). In order to reduce the hardware overhead, the multiplications of 16/32 point DCT are implemented with ICM (input-muxed constant multipliers) instead of MCM or regular multipliers. The proposed design is 51% more area efficient than previous work. To the author's knowledge, this is the first published work to support both forward and inverse 4/8/16/32-point integer transform for HEVC standard in a unified architecture.

key words: *HEVC, integer DCT/IDCT, Hadamard transform, input-muxed constant multiplier, multi-standard video coding*

1. Introduction

As the video coding technology advances, various types of transforms have been adopted in miscellaneous video coding standards. Discrete Cosine Transform (DCT) is the most popular transform for block based video coding due to its ability to concentrate the energy of video residual data into low frequency domain.

Floating-point 8×8 2D DCT is used in the video coding standards like MPEG-1 [1], MPEG-2 [2] and MPEG-4 [3]. Due to the limited bit precision in any actual digital circuit system, the floating-point DCT may cause data mismatch problem between encoder and decoder.

In order to avoid this mismatch, integer 8×8 2D DCT is used in later video standards like H.264 [4], AVS [5] and VC-1 [6]. H.264 and VC-1 also use smaller size integer DCT such as integer 4×4 2D DCT, which can improve coding efficiency for the video sequences with complex texture. In addition, Hadamard transform is used in H.264 to process the DC coefficients of intra 16×16 or chroma prediction mode.

Manuscript received December 21, 2012.
Manuscript revised February 24, 2013.

[†]The authors are with State Key Lab of ASIC and System, Fudan University, Shanghai, 200240, China.
a) E-mail: fanyibo@fudan.edu.cn (Corresponding author)

DOI: 10.1587/transfun.E96.A.1534

High Efficiency Video Coding (HEVC) standard [7] is an emerging new video coding standard which is jointly developed by the two video standardization organizations: MPEG and ITU. It is considered as the successor of H.264. Compared with H.264, [8] reported that HEVC can reduce up to 44% bit rate with the same picture quality. In order to achieve this bit rate reduction, HEVC adopts many new coding tools including larger size integer DCT such as 16×16 and 32×32 2D DCT.

Two different methods can be used to perform 2D DCT: direct 2D method [9], [10] or Row Column Decomposition Method (RCDM). In [9], a direct 2D method is proposed for floating-point 8×8 2D IDCT. A direct 2D method for 4×4 H.264 integer DCT is also proposed in [10]. Direct 2D method has the advantage of higher processing capacity. It also avoids the usage of transpose memory. But its internal connection and computational logic is quite complex. This method is not suitable for larger size transform such as 16×16 or 32×32 2D transform. RCDM sequentially perform 1D transform twice instead of direct 2D transform. RCDM can greatly reduce the hardware cost in case of larger size transform. The proposed design in this paper also adopts RCDM.

Various VLSI architectures based on RCDM have been proposed for multiple-standard 1D transforms [11]–[17]. A low-cost hardware-sharing architecture of 1D inverse transform is proposed for H.264 and AVS in [11]. An offset matrix is exploited to reduce computational complexity. [12] extends this idea to support more standards such as VC-1 and MPEG-2/4. The offset matrix will become too complex if more standards are to be supported. In [13] and [14], a flexible architecture is proposed for multi-standard transform. But most previous works [11]–[14] can only support 4/8-point 1D transform. In [15], the regular multiplier based architecture is proposed to support 4/816/32-point IDCT. The regular multiplier is used and it is much larger than the constant multiplier in terms of silicon area. A VLSI architecture for HEVC 16/32-point IDCT is proposed in [16]. The proposed fast algorithm in [16] is based on the obsolete Working Draft 2 of HEVC and can no longer be applied to the latest HEVC standard. A multiplierless design is proposed for HEVC 16-point DCT in [17]. It is optimized for 16-point DCT only and cannot support other transform size.

In order to address the above problems, we propose a unified VLSI architecture for various types of transforms. It can support both the existing video coding standards like H.264, AVS, MPEG-2/4, VC-1 and the emerging HEVC

standard. A novel design named configurable butterfly array (CBA) is also proposed to support both the forward transform and the inverse transform in this unified architecture.

The hardware resource sharing in this design is carried out in two different levels: the architecture level or the block level. In the top architecture level, the forward and inverse transform can share the same hardware blocks such as multiplication blocks, adder tree. The CBA block can be configured to support either forward transform or inverse transform.

Hardware sharing is also carried out inside the multiplication/add tree blocks. The hardware for smaller size transform can be reused for larger size transform. One hardware sharing technique called "input-muxed constant multiplier" is used to implement the multiplication circuits of 16/32-point DCT. It can reduce the hardware cost significantly in comparison with the multiplication blocks used in [15]. This architecture is also flexible to support more future video standards as long as the DCT-like transform matrix is used.

The rest of this paper is organized as follows. In Sect. 2, the idea and the fast computational algorithm of integer IDCT/DCT is reviewed. Hadamard transform is also introduced in this section. Section 3 presents the proposed VLSI architecture of the unified 1D forward/inverse transform. The result of VLSI implementation and comparisons with previous designs are shown in Sect. 4. A conclusion is drawn in Sect. 5.

2. Reviews of 1D DCT/IDCT Transforms

1D integer DCT/IDCT has been widely used in the latest video coding standards like H.264/AVC, AVS, VC-1 and HEVC. 1D integer DCT can be defined as:

$$Y = A_N \times X \tag{1}$$

where X is the input signal, Y is the transform result and A_N is the $N \times N$ integer transform matrix defined by each video standard. 1D integer IDCT can be also defined in a similar equation as:

$$X = Y \times A_N \tag{2}$$

In HEVC, the size of transform matrix can be 4×4, 8×8, 16×16 or 32×32. Many fast computational algorithms have been proposed for floating-point DCT [18]–[20]. Matrix factorization is the core idea of these fast algorithms. The fast algorithm proposed by Chen [18] is a fundamental work. The $N\times N$ transform matrix A_N can be decomposed in a recursive form, which is shown in Eq. (3). Here P_N is the permutation matrix and B_N is the butterfly operation. The transform matrix A_N is divided into even part matrix ($A_{N/2}$) and odd part matrix ($R_{N/2}$) while matrix $A_{N/2}$ can be further divided in the same fashion. Matrix $R_{N/2}$ can also be factorized and decomposed into several matrices, which is shown in [18].

$$[A_N] = [P_N] \times \begin{bmatrix} A_{N/2}, & 0 \\ 0, & R_{N/2} \end{bmatrix} \times [B_N] \tag{3}$$

where P_N (N = 4, 8, 16 or 32) is the permutation matrix and it is used to permute the output vector Y. Matrix P_N is defined as:

$$\begin{aligned} P_N(i, j) &= 1 \text{ for } i = 2 \times j \text{ or } i = (j - N/2) \times 2 + 1 \\ &= 0 \text{ otherwise} \end{aligned} \tag{4}$$

where $0 \le i \le N - 1$ and $0 \le j \le N - 1$.

B_N (N = 4, 8, 16 or 32) is called as the butterfly matrix and it is used to calculate the sum and difference of each pair of the input signals. Matrix B_N is defined as:

$$B_N(i, j) = \begin{cases} 1 & \text{for } i = j \text{ and } i < N/2 \\ 1 & \text{for } i = N - 1 - j \\ -1 & \text{for } i = j \text{ and } i \ge N/2 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

where $0 \le i \le N - 1$ and $0 \le j \le N - 1$.

The integer transform matrix used in H.264, VC-1, AVS or HEVC can reduce computational complexity. But the integer transform matrix is no longer orthogonal due to the integer approximation of the transform coefficients. So the fast algorithms for floating-point DCT as mentioned above cannot be applied without modification.

After the thorough inspection of various integer transform matrices used in AVS, VC-1, H.264 and HEVC, we can see that Eq. (3) can still be applied to integer transform due to its symmetric/asymmetric feature. The integer transform matrix can be recursively divided into smaller matrices. The permutation and butterfly operation remain the same. The only exception is that the generalized method of decomposing the odd part matrix ($R_{N/2}$) can no longer be used for integer transform.

The 4×4 integer transform matrix for HEVC is shown in Eq. (6), which is shown below:

$$A_4 = \begin{bmatrix} c00 & c00 & c00 & c00 \\ c08 & c24 & -c24 & -c08 \\ c00 & -c00 & -c00 & c00 \\ c24 & -c08 & c08 & -c24 \end{bmatrix} \tag{6}$$

The transform matrix of size 8×8, 16×16 or 32×32 can be expressed as Eqs. (7), (8) or (9).

$$[A_8] = [P_8] \times \begin{bmatrix} A_4, & 0 \\ 0, & R_4 \end{bmatrix} \times [B_8] \tag{7}$$

$$[A_{16}] = [P_{16}] \times \begin{bmatrix} A_8, & 0 \\ 0, & R_8 \end{bmatrix} \times [B_{16}] \tag{8}$$

$$[A_{32}] = [P_{32}] \times \begin{bmatrix} A_{16}, & 0 \\ 0, & R_{16} \end{bmatrix} \times [B_{32}] \tag{9}$$

$$R_4 = \begin{bmatrix} c28 & c20 & c12 & c04 \\ -c20 & -c04 & -c28 & c12 \\ c12 & c28 & -c04 & c20 \\ -c04 & c12 & -c20 & c28 \end{bmatrix} \tag{10}$$

The odd part matrix R_4, R_8 and R_{16} are shown in Eqs. (10), (11) and (12). c00, c01, ..., c30, c31 are the 32 transform coefficients defined by each video coding standard. The exact values of these 32 transform coefficients are

Table 1 Transform coefficients of various video coding standards (even part).

	c00	c02	c04	c06	c08	c10	c12	c14	c16	c18	c20	c22	c24	c26	c28	c30
HEVC	64	90	89	87	83	80	75	70	64	57	50	43	36	25	18	9
MEPG-2/4	362	-	502	-	473	-	426	-	362	-	284	-	196	-	100	-
4-p VC-1	17	-	-	-	22	-	-	-	17	-	-	-	10	-	-	-
8-p VC-1	12	-	16	-	16	-	15	-	12	-	9	-	6	-	4	-
4-p H.264	1	-	-	-	2	-	-	-	1	-	-	-	1	-	-	-
8-p h.264	8	-	12	-	8	-	10	-	8	-	6	-	4	-	3	-
AVS	8	-	10	-	10	-	9	-	8	-	6	-	4	-	2	-
Hadamard	1	-	-	-	1	-	-	-	1	-	-	-	1	-	-	-

Table 2 Transform coefficients of various video coding standards (odd part).

	c01	c03	c05	c07	c09	c11	c13	c15	c17	c19	c21	c23	c25	c27	c29	c31
HEVC	90	90	88	85	82	78	73	67	61	54	46	38	31	22	13	4

shown in Table 1 and Table 2.

$$R_8 = \begin{bmatrix} c30 & c26 & c22 & c18 & c14 & c10 & c06 & c02 \\ -c26 & -c14 & -c02 & -c10 & -c22 & c30 & c18 & c06 \\ c22 & c02 & c18 & -c26 & -c06 & -c14 & c30 & c10 \\ -c18 & -c10 & c26 & c02 & c30 & -c06 & -c22 & c14 \\ c14 & c22 & -c06 & -c30 & c02 & -c26 & -c10 & c18 \\ -c10 & c30 & c14 & -c06 & c26 & c18 & -c02 & c22 \\ c06 & -c18 & c30 & c22 & -c10 & c02 & -c14 & c26 \\ c02 & c06 & -c10 & c14 & -c18 & c22 & -c26 & c30 \end{bmatrix} \quad (11)$$

4×4 Hadamard transform is used in H.264 to further improve the coding efficiency for luma or chroma DC coefficients. It can be regarded as a special integer DCT in which case all the transform coefficients are either 1 or -1. It is quite straightforward to integrate the Hadamard transform with DCT. The transform coefficients of 4×4 Hadamard transform are also shown in Table 1.

3. VLSI Architecture for Unified 1D Forward/Inverse Transform

In this section, the unified VLSI architecture for 1D integer forward/inverse transform is described with more details. This architecture can support all types of transforms like 8-point floating-point DCT/IDCT, 4/8/16/32-point integer DCT/IDCT and 4-point Hadamard.

3.1 Top Architecture and Pipeline Design

The top level architecture and pipeline design of the forward/inverse transform are shown in Fig. 1. This architecture can support 32-point DCT/IDCT. Smaller size transforms such as 4/8/16-point DCT/IDCT or 4-point Hadamard transform can also be supported by the same hardware.

The input port X in Fig. 1 represents the 32 input signals that are to be transformed. The output port Y represents the 32 transformed results. If the transform size is set as N and N is less than 32, only first N-th input signals are used and only the first N-th output signals are valid outputs. Forward/inverse transforms with different transform size (4/8/16/32-point) can be supported by setting the input ports (BP32, BP16, BP8 and type) to a proper value. The exact setting is described with more details in Sect. 3.2.

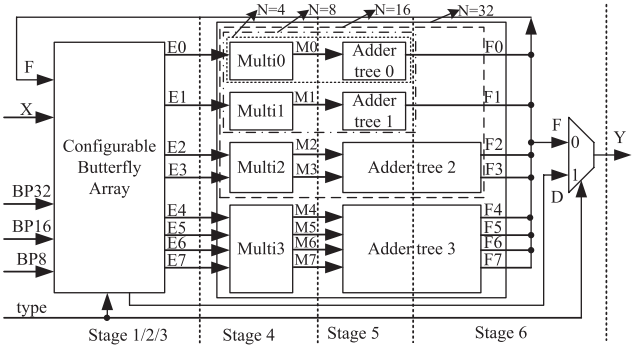


Fig. 1 Top architecture of the unified forward/inverse transform.

The hardware modules used in Fig. 1 can be categorized into three types: (a) configurable butterfly array (CBA). This block is used to perform the butterfly operation which is defined in Eq. (5). It can be configured to support different transform type or transform size. This block will be described in Sect. 3.2 with more details. (b) The multiplication blocks. Four multiplication blocks named Multi0, Multi1, Multi2 and Multi3 are used in this proposed design. (c) Adder tree blocks. Four adder tree blocks are used to sum up the outputs of the corresponding multiplication blocks.

When this design is configured to support 4-point transform, only Multi0 and adder tree 0 are used and other pipeline stages are bypassed. Multi0/1 and adder tree 0/1 are used to calculate 8-point transform. Multi0/1/2 and adder tree 0/1/2 are used to calculate 16-point transform. As shown in Fig. 1, the multiplication blocks and add tree

$$R_{16} = \begin{bmatrix} c31 & c29 & c27 & c25 & c23 & c21 & c19 & c17 & c15 & c13 & c11 & c09 & c07 & c05 & c03 & c01 \\ -c29 & -c23 & -c17 & -c11 & -c05 & -c01 & -c07 & -c13 & -c19 & -c25 & -c31 & c27 & c21 & c15 & c09 & c03 \\ c27 & c17 & c07 & c03 & c13 & c23 & -c31 & -c21 & -c11 & -c01 & -c09 & -c19 & -c29 & c25 & c15 & c05 \\ -c25 & -c11 & -c03 & -c17 & -c31 & c19 & c05 & c09 & c23 & -c27 & -c13 & -c01 & -c15 & -c29 & c21 & c07 \\ c23 & c05 & c13 & c31 & -c15 & -c03 & -c21 & c25 & c07 & c11 & c29 & -c17 & -c01 & -c19 & c27 & c09 \\ -c21 & -c01 & -c23 & c19 & c03 & c25 & -c17 & -c05 & -c27 & c15 & c07 & c29 & -c13 & -c09 & -c31 & c11 \\ c19 & c07 & -c31 & -c05 & -c21 & c17 & c09 & -c29 & -c03 & -c23 & c15 & c11 & -c27 & -c01 & -c25 & c13 \\ -c17 & -c13 & c21 & c09 & -c25 & -c05 & c29 & c01 & c31 & -c03 & -c27 & c07 & c23 & -c11 & -c19 & c15 \\ c15 & c19 & -c11 & -c23 & c07 & c27 & -c03 & -c31 & c01 & -c29 & -c05 & c25 & c09 & -c21 & -c13 & c17 \\ -c13 & -c25 & c01 & -c27 & -c11 & c15 & c23 & -c03 & c29 & c09 & -c17 & -c21 & c05 & -c31 & -c07 & c19 \\ c11 & c31 & -c09 & c13 & c29 & -c07 & c15 & c27 & -c05 & c17 & c25 & -c03 & c19 & c23 & -c01 & c21 \\ -c09 & c27 & c19 & -c01 & c17 & c29 & -c11 & c07 & -c25 & -c21 & c03 & -c15 & -c31 & c13 & -c05 & c23 \\ c07 & -c21 & -c29 & c15 & -c01 & c13 & -c27 & -c23 & c09 & -c05 & c19 & c31 & -c17 & c03 & -c11 & c25 \\ -c05 & c15 & -c25 & -c29 & c19 & -c09 & c01 & -c11 & c21 & -c31 & -c23 & c13 & -c03 & c07 & -c17 & c27 \\ c03 & -c09 & c15 & -c21 & c27 & c31 & -c25 & c19 & -c13 & c07 & -c01 & c05 & -c11 & c17 & -c23 & c29 \\ -c01 & c03 & -c05 & c07 & -c09 & c11 & -c13 & c15 & -c17 & c19 & -c21 & c23 & -c25 & c27 & -c29 & c31 \end{bmatrix} \quad (12)$$

blocks used for 4/8/16/32-point transform are marked with dot line, dot-dash line, dash line and solid line. The CBA block can also be used for 4/8/16/32-point transform. So the hardware sharing among different transform sizes can be well achieved.

In order to achieve higher working frequency, the proposed design is divided into 6 pipeline stages. Three butterfly operations (defined by the butterfly matrix B_8, B_{16} and B_{32}) are performed in configurable butterfly array. Each butterfly operation is arranged as one pipeline stage. Totally three pipeline stages are used for configurable butterfly array. The multiplication blocks (Multi0, Multi1, Multi2 and Multi3) are arranged as the 4th pipeline stage. Adder Tree0 and Adder Tree1 are arranged in the 5th pipeline stage. The computational complexity of Add Tree3 or Add Tree4 is much bigger than Adder Tree0 or Adder Tree1. So Add Tree3 and Add Tree4 are divided into two separate pipeline stages: the 5th and 6th stage. The inside details of Add Tree3 and Add Tree4 will be introduced in Sect. 3.3.

3.2 VLSI Implementation of Configurable Butterfly Array

The butterfly operation is required by either the forward or the inverse transform. Three types of butterfly matrixes are used for a 32-point transform: B_8, B_{16} and B_{32}. These three matrixes are defined by Eq. (5). In case of 32-point forward transform, the 32-point butterfly operation (B_{32}) is performed first and followed by the 16-point butterfly operation (B_{16}). The 8-point butterfly operation (B_8) is performed at last. In case of 32-point inverse transform, the 8-point butterfly operation (B_8) is performed first and followed by the 16-point butterfly operation (B_{16}). The 32-point butterfly operation (B_{32}) is performed at last.

A flexible architecture named configurable butterfly array is proposed to support the butterfly operation required by both the forward transform and the inverse transform. The detailed diagram of CBA is depicted in Fig. 2. CBA is divided into 3 pipeline stages by the vertical dash-lines in Fig. 2. Ten identical butterfly blocks BF0-BF9 are used in CBA.

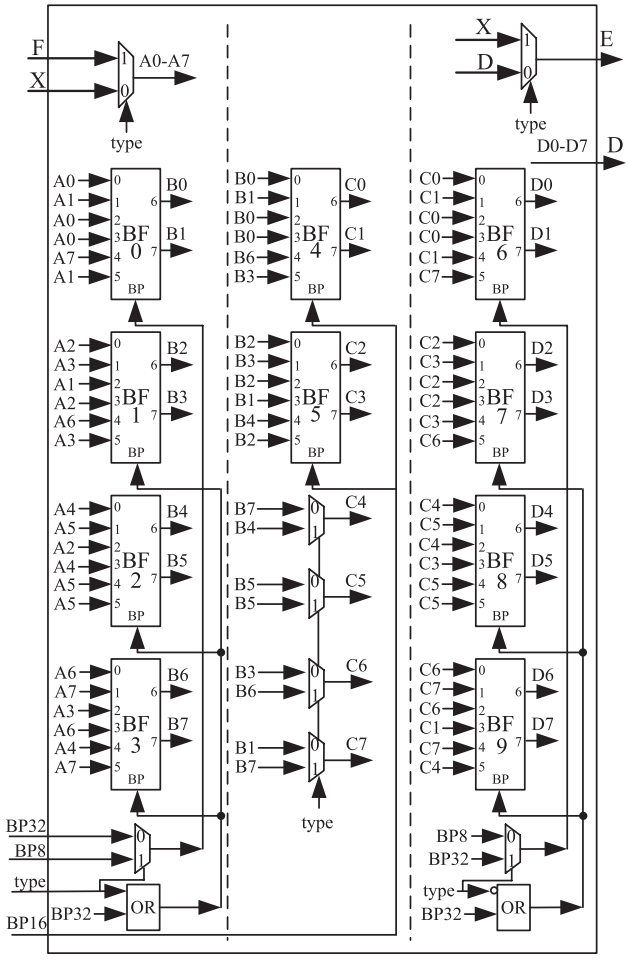


Fig. 2 Configurable butterfly array.

The input ports (BP32, BP16 and BP8) are used to con-

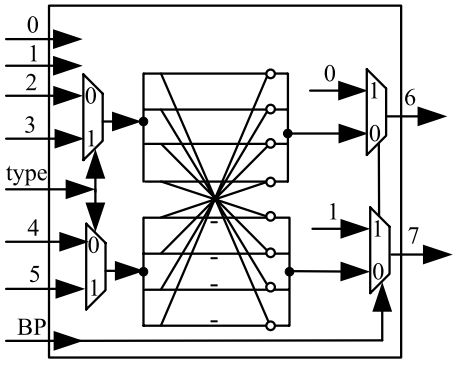


Fig. 3 Butterfly block (BF) with bypass function.

figure the supported transform size. If the transform size is set as 4, BP32, BP16 and BP8 are all set as 1. If the transform size is set as 8, BP32 and BP16 are set as 1 while BP8 is set as 0. If the transform size is set as 16, BP32 is set as 1 while BP8 and BP16 are set as 0. If the transform size is set as 32, BP32, BP16 and BP8 are all set as 0.

The input port "type" is used to configure the supported transform type. When "type" is set as 0, the system in Fig. 1 is configured to support forward transform and the outputs of adder tree blocks (bus signal F in Fig. 1) are selected as the final transform results. BF0-BF3 are configured to perform 32-point butterfly operation and BF6 is used to perform 8-point butterfly operation.

When "type" is set as 1, the system in Fig. 1 is configured to support inverse transform. The outputs of configurable butterfly array (bus signal D in Fig. 1) are selected as the final transform results. BF0 is configured to perform 8-point and BF6-BF9 are configured to perform 32-point butterfly operation. BF4 and BF5 are used to perform 16-point butterfly operation.

The internal architecture of BF block is shown in Fig. 3. There are six data input ports (0-5) and two output ports (6-7) in BF block. Input port "type" is used to configure the transform type (forward or inverse transform. Input port "BP" is used to support the bypass fuction. Each BF block can perform an 8-point butterfly operation. When the input "BP" is set as 1, the 8-point butterfly block is bypassed and the input ports 0/1 are set as output signals. Otherwise the 8-point butterfly operation is performed for either the forward transform or the inverse transform according the input signal "type.

Signal A, B and C in Fig. 2 represent the intermediate bus signals. In order to simplify the diagram in Fig. 2, all these bus signals are divided into 8 groups. Each group is named as A0-A7, B0-B7 or C0-C7. For example, the input bus signal X in Fig. 2 actually consists of 32 separate input signals (x_0, $x_1, \ldots, x_{31}$). If each input signal is assumed to be 16-bit and the design is configured to support forward transform, A0 consists of four input signals (x_0, x_1, x_2, x_3). A0 will be a 64-bit bus signal. The bus width of A1-A7 is the same as that of A0. Bus signals B/C are also divided into 8 groups in a similar way.

3.3 VLSI Implementation for Multiplication and Add Tree Blocks

Previous work [13] has shown that the multiplication and adder tree circuit account for more than 80% of the whole hardware area. In most previous works [11]–[14], [21] on integer transform, multiplication is performed by multiplierless Multiple Constant Multiplication (MCM). MCM requires less hardware resource than regular multipliers when the transform size is small. The experimental results in Table 4 will show that MCM is more area efficient for 4or 8×8 transform block. Therefore the multiplications block Multi0 and Multi1 in this proposed design are implemented by MCM approach. The coefficients used in Multi0 and Multi1 are defined by Eqs. (6) and (10). The internal design of Adder Tree block 0/1 is the same as the one used in [15].

But the number of constant multipliers in a MCM block will increase exponentially as the transform size increases. A direct VLSI implementation of such MCM block will become unaffordable in case of large size transform.

The MCM based multiplication block Multi2 is shown in Fig. 4(a) as an example. There are 8 input data (E2, E3) of Multi2. Each input data will be multiplied with 8 different transform coefficients (c02, c06, c10, c14, c18, c22, c26, c30). So the number of multiplications in MCM block Multi2 is 64.

When the 1D transform is done in multiple clock cycles, a hardware sharing technique can be used to reduce the number of constant multipliers. The more clock cycles are used, the fewer multipliers and adder trees are needed. Such approach is adopted in [15] to reduce the hardware cost. The multiplication block Multi2 and Multi3 in [15] are implemented with regular multipliers as shown in Fig. 4(b). The multiplicator of each multiplier is the corresponding transform coefficient and the multiplicand is the input signal. The transform coefficients are stored in SRAM. The transform coefficients will be read out from SRAM when the transform is in progress. Compared with MCM approach in Fig. 4(a), the number of multipliers use in Fig. 4(b) is reduced from 64 to 16.

A novel architecture named input-muxed constant multiplier (ICM) is proposed in this paper to further reduce the hardware cost of multiplication block. The diagram of this architecture is shown in Fig. 4(c). The number of multipliers used in ICM approach is still 16. But each multiplier is a constant multiplier instead of a regular multiplier. At each clock cycle, one input signal among the eight input signals (d8-d15) is selected and then multiplied by the corresponding transform coefficients. The transform coefficients used in ICM can be either positive or negative, which is decided by the control logic of the multiplication block.

Here we denote the 8 constant multipliers (C02, C06, C10, C14, C18, C22, C26, C30) as one set of constant multipliers, which is marked with dot line in Fig. 4(c). If the multiplication in Multi2 is done in one cycle, 8 sets of con-

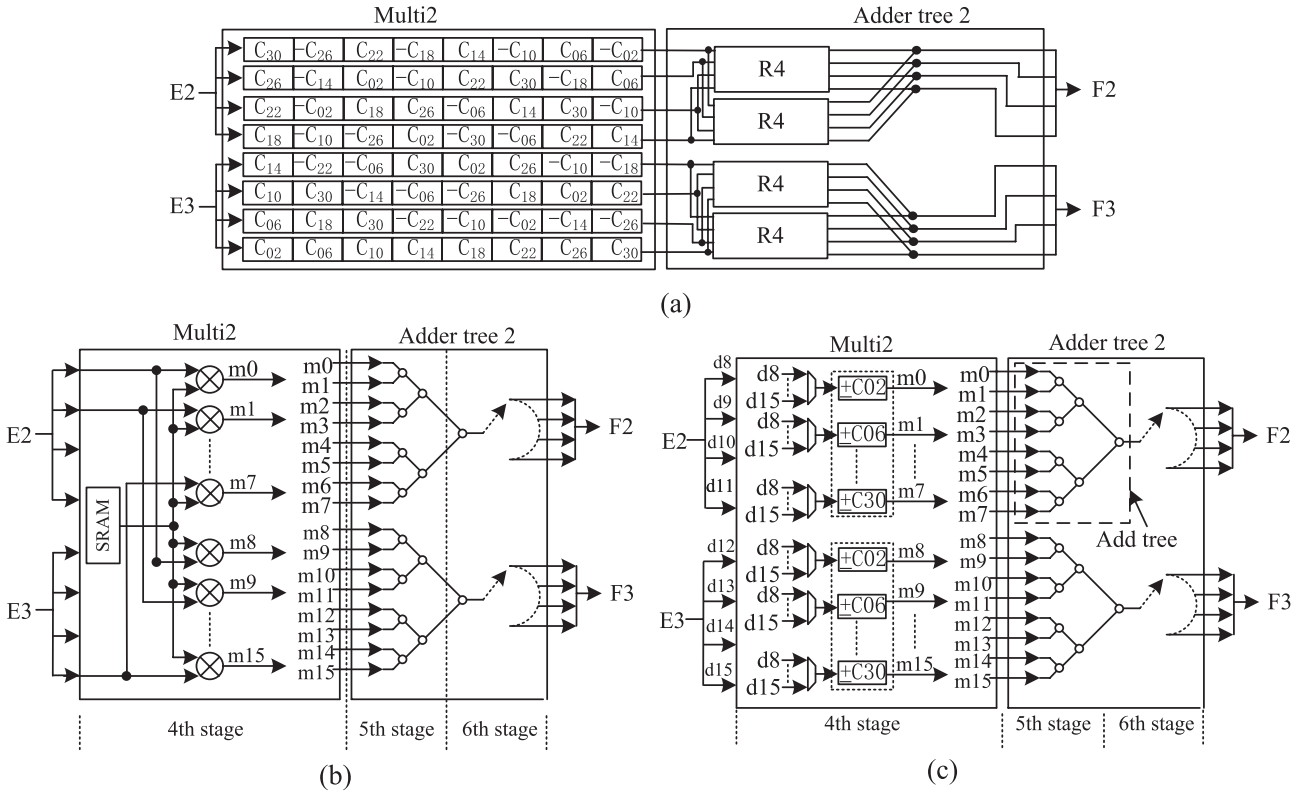


Fig. 4 Three different approach to implement the multiplication block Multi2 and adder tree block. (a) MCM based approach (b) Regular multiplier based approach (c) Proposed ICM based approach.

stant multipliers are needed. If the multiplication is done in 8 cycles, only one set of constant multipliers is needed. The more cycle used for multiplication, the less multipliers are required.

The minimum throughput of this design is set as 4-pixel per cycle. So the 16-point DCT needs to be done in 4 cycles. Two sets of constant multipliers are used in the Multi2 block which is shown in Fig. 4(c). Two outputs of adder tree 2 can be generated at each cycle. All 8 outputs of adder tree 2 can be generated in 4 cycles. The number of adders in adder tree 2 of Fig. 4(a) is 48. The number of adders is reduced from 48 to 14 for the adder tree 2 block in Fig. 4(b) or Fig. 4(c).

The same hardware sharing technique can also be applied to Multi3. As shown in Fig. 5, two sets of constant multipliers are used in Multi3 and each set consists of 16 constant multipliers (C01, C03,. . . , C29, C31). All 16 outputs of add tree 3 can be generated in 8 cycles. Four add tree blocks are used in adder tree 3 and the internal detail of each add tree is the same as what is marked with dash line in Fig. 4(c).

4. Experimental Results

Verilog HDL is used to implement the proposed design. Synthesized with SMIC 0.13 μm standard cell library, this design can work at 191 MHz and the gate count at this working frequency is 54.1 K. The hardware cost of each transform is also shown in Table 3.

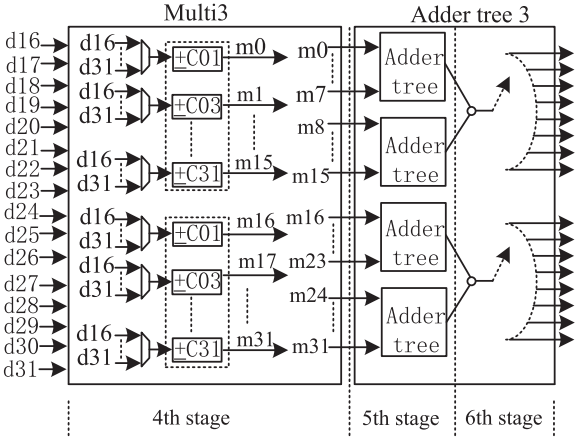


Fig. 5 ICM based multiplication block Multi3 and adder tree 3.

Table 3 Hardware area for each transform block.

	4-point	8-point	16-point	32-point
Gate Count (K)	7.2	19.6	31.1	54.1

The numbers of pipeline stages used for 4/8/16/32-point 1D transform are 2/3/5/6. The unused pipeline stages are bypassed. The latency between row and column trans-

form of 4-point 1D transform is 6 (4+2) cycles. The latency of 8/16/32-point transform is 11 (8+3) /69 (64+5) /262 (256+6) cycles respectively.

Three different approaches mentioned in Fig. 4 are used to implement the multiplication blocks named Multi0, Multi1, Mulit2 and Multi3. The gate count of each block is shown in Table 4. ICM is not applied to Multi0 and Multi1 due to the design requirement. So the gate count of Multi0 or multi1 with ICM approach is not available in Table 4. It can be seen from Table 4 that the ICM based multiplication block is much more area efficient than MCM or the regular multiplier based approach. Compared with the regular multiplier based design in [15], more than 50% silicon area is reduced in this proposed architecture for Multi2 or Multi3.

The precision of each input signals can be described as N-bit. Each 8-point butterfly operation will increase the precision by one bit. So the CBA block in Fig. 1 will increase the precision by 3 bits. According to Table 1 and Table 2, the maximum transform coefficients used in Multi0/Multi1/ Multi2/ Multi3 are 473, 502, 90 and 90. The bit precisions increased by these four multiplication blocks are 9/9/7/7. The bit precisions increased by adder tree block0/1/2/3 are 2/2/3/4. The width of the bus signal F0 in Fig. 1 can be up to $N + 14$ ($14 = 3+ 9+2$) bit. The width of F1/F2-F3/F4-F7 is $(N + 14)/(N + 13)/(N + 14)$ respectively. So the maximum bit depth of output bus signal Y can be up to $N + 14$.

In this proposed design, the width of input signal is set as 16-bit. The transform results of this design can be up to 30-bit (30=16+14). Some video coding standards such as AVS, VC-1, H.264 and HEVC use integer transforms. The bit precision of the 1D integer transform result is limited to be no more than 16-bit. Before the column transform starts, the results of row transform should be clipped to 16-bit. The clipping module is quite simple and straightforward. So it is not shown in Fig. 1. IEEE standard 1180-1990 also defines the required bit precision for floating point 8×8 transform. 16-bit input signals and 10-bit transform coefficients in this proposed design can meet the constraint of IEEE standard 1180-1990.

Table 4 Gate count of multiplication block.

	Multi0	Multi1	Multi2	Multi3
MCM	2.5K	7K	30K	110K
RM	3.9K	7.9K	15.6K	31K
ICM	-	-	5.8K	12.8K

The throughput of 8-point DCT is 8 pixels/cycle and 4 pixels/cycle for all other types of transforms. 4/8-point 1D transform can be done in one cycle. 16/32-point 1D DCT can be done in 4/8 cycles respectively. A 32×32 block can be processed in 128 ($32 \times 32/8$) cycles for 8-point DCT and 256 ($32 \times 32/4$) cycles for all other types of transforms.

If two proposed 1D transform architectures are used to perform row and column transform in a pipelined fashion, the working frequency needed to support a specific video sequence can be calculated by Eq. (13):

$$Freq = \frac{W \times H \times Format \times fps}{32 \times 32} \times (L + 256) \tag{13}$$

where $W \times H$ is the resolution of video sequence. Format is set as 1.5 for 4:2:0 video and 3 for 4:4:4 video. fps is the video frame rate. L is the latency between the row transform and column transform. (L+256) is the number of cycles needed to process a 32×32 block. The maximum latency is 262 cycles in this design. Our proposed work can easily support the high video resolution of $4\,\mathrm{K} \times 2\,\mathrm{K}$ (4096×2048) @30 fps with 4:2:0 format at the working frequency of 191 MHz.

The comparison between previous work and this work is summarized in Table 5. The designs proposed in [13], [14], [21], [22] can only support 4 or 8-point transform. The work in [17] can only support 16-point HEVC DCT and the work in [15] can support only support 4/8/16/32-point IDCT. This proposed design can support both the forward and inverse 4/8/16/32-point transforms in a unified architecture. Compared with the work in [15], this work is still 51% more efficient than [15] in terms of silicon area. Additional on-chip SRAM is used in [15] while this proposed design does not require any on-chip SRAM. The hardware cost is further reduced.

5. Conclusion

To the best of the authors' knowledge, this is the first published work to support both forward and inverse 4/8/16/32-point integer transform for HEVC standard in a unified architecture. It can support multiple video standards such as MPEG-2/4, VC-1, H.264, AVS and HEVC. A novel design named configurable butterfly array (CBA) can be configured to support either forward or inverse transform. The CBA block can also be configured to support different transform size. The hardware for smaller size integer

Table 5 Comparison with previous works.

Design	Gate count	Working freq (MHz)	Supported transform type								
			MPG-2/4	VC-1		H.264		AVS	HEVC		Hadamard
				4p	8p	4p	8p		DCT	IDCT	
Proposed	54.1K	191	√	√	√	√	√	√	√	√	√
Wang[13]	15.49K	100	√	√	√	√	√	√	×	×	×
Fan[21]	15.03K	100	×	√	√	×	×	×	×	×	×
Kim[22]	23.72K	80	√	×	×	×	×	×	×	×	×
Shen[15]	109.2K	350	√	√	√	√	√	√	×	√	×

transform can be recursively reused for larger size integer transform. Hadamard transform can also share the hardware resource with DCT/IDCT. In order to reduce the hardware cost, the multiplication of 16/32-point transform is implemented by ICM. This 6-stage pipelined architecture can support $4\,K \times 2\,K$ @30 fps video (4:2:0 YUV format) at 191 MHz working frequency. The gate count under this working frequency is 54.1 K. Our design is 51% more efficient than previous work in [15].

Acknowledgments

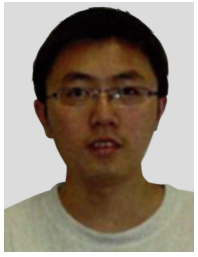
This paper is supported by National High Technology Research and Development Program (863, 2012AA012001), State Key Lab of ASIC & System Project (11MS004), Specialized Research Fund for the Doctoral Program of Higher Education (SRFDP, 20120071120021).

References

[1] ISO/IEC 11172-2 (MPEG-1), "Video coding standard information technology — Coding of moving pictures and associated audio for digital storage media at up to about 1.5 Mbit/s — Part 2: Video," 1993.
[2] ISO/IEC 13818-2 (MPEG-2), "Video coding standard, information technology — Generic coding of moving pictures and associated audio information: Video," 1995.
[3] ISO/IEC 14496-2 (MPEG-4), "Video coding standard, information technology — Coding of audio- visual objects — Part 2: Visual," 2004.
[4] ITU-T rec.H.264/ISO/IEC 14496-10, "Advanced video coding for generic audiovisual services," 2005.
[5] Chinese National Standard Committee, "Information technology — Advanced coding of audio and video — Part 2:Video, GB/T 200090.2-2006," 2006.
[6] SMPTE Standard: SMPTE 421M, "VC-1 Compressed Video Bitstream Format and Decoding Process," 2006.
[7] B. Bross, W.-J. Han, G.J. Sullivan, J.-R. Ohm, and T. Wiegand, "High Efficiency Video Coding (HEVC) text specification draft 7," JCT-VC-I1103.doc, May 2012.
[8] G. Wallendael, S. Leuven, J. Cock, F. Bruls, and R. Walle, "3D video compression based on high efficiency video coding," IEEE Trans. Consum. Electron., vol.58, no.1, pp.137–145, Feb. 2012.
[9] T. Tziortzios and S. Dokouzyannis, "A novel architecture for fast 2D IDCT decoders with reduced number of multiplications," IEEE Trans. Consum. Electron., vol.58, no.1, pp.1384–1389, Aug. 2011.
[10] C.P. Fan, "Fast 2-dimensional 4×4 forward integer transform implementation for H.264/AVC," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.53, no.3, pp.174–177, March 2006.
[11] G.A. Su and C.P. Fan, "Low-cost hardware-sharing architecture of fast 1-D inverse transforms for H.264/AVC and AVS applications," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.55, no.12, pp.1249–1253, Dec. 2008.
[12] C.P. Fan, C.H. Fang, and C.W. Chang, "Fast multiple inverse transforms with low-cost hardware sharing design for multistandard video decoding," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.58, no.812, pp.517–521, Aug. 2011.
[13] K. Wang, J. Chen, W. Cao, Y. Wang, L. Wang, and J. Tong, "A reconfigurable multi-transform VLSI architecture supporting video codec design," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.58, no.7, pp.432–436, July 2011.
[14] H. Qi, Q. Huang, and W. Gao, "A low-cost very large scale integration architecture for multistandard inverse transform," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.57, no.7, pp.551–555, July 2010.
[15] S. Shen, W. Shen, Y. Fan, and X. Zeng, "A unified 4/8/16/32-point integer IDCT architecture for multiple video coding standards," Proc. IEEE International Conf. on Multimedia and Expo, pp.788–793, July 2012.
[16] J.S Park, W.J Nam, S.M Han, and S.S Lee, "2-D large inverse transform (16×16, 32×32) for HEVC," J. Semiconductor Tech. & Science, vol.12, no.2, pp.203–211, June 2012.
[17] R. Jeske, J.C. Wrege, R. Conceicao, M. Grellert, J. Mattos, and L. Agostini, "Low cost and high throughput multiplierless design of a 16 point 1-D DCT of the New HEVC video coding standard," VIII Southern Conf. on Programmable Logic (SPL), pp.1–6, 2012.
[18] W.H. Chen, C.H. Smith, and S.C. Fralick, "A fast computational algorithm for the discrete cosine transform," IEEE Trans. Commun., vol.COM-25, no.9, pp.1004–1009, Sept. 1977.
[19] Y. Arai, T. Agui, and M. Nakajima, "A fast DCT-SQ scheme for Images," IEICE Trans., vol.71, no.11, pp.1095–1097, Nov. 1988.
[20] C. Loeffler, A. Ligtenberg, and G.S. Moschytz, "Practical fast 1-D DCT algorithms with 11 multiplications," Int. Conf. Acoustic, Speech and Signal Processing, pp.988–991, 1989.
[21] C.P. Fan and G.A. Su, "Fast algorithm and low-cost hardware-sharing design of multiple integer transforms for VC-1," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol.56, no.10, pp.788–792, Oct. 2009.
[22] K. Kim and J.S. Koh, "An area efficient DCT architecture for MPEG-2 video encoder," IEEE Trans. Consum. Electron., vol.45, no.1, pp.62–67, Feb. 1999.

Sha Shen obtained the B.E. degree in electronics and engineering from Fudan University, Shanghai, China in 2001, M.S. degree in Microelectronics from Fudan University, Shanghai, China in 2004. From 2004 to 2010, he worked as a senior engineer on digital circuit design in Trident Multimedia Technology (Shanghai) Co. Ltd. Currently he is working at the State Key Lab of ASIC and System in Fudan University and pursuing his Ph.D. degree. His research interests include high performance multimedia systems, video coding and image processing algorithm and the related VLSI design.

Weiwei Shen received the B.S. degrees in Microelectronics Department from Fudan University in 2010. He now is working at the State Key Lab. of ASIC and System in Fudan University, Shanghai, China. His research interests include video and image processing, digital signal processing, and VLSI design.



Yibo Fan received the B.E. degree in electronics and engineering from Zhejiang University, China in 2003, M.S. degree in Micro electronics from Fudan University, China in 2006, and Ph.D. degree in engineering from Waseda University, Japan in 2009. From 2009 to 2010, he worked as an Assistant Professor in Shanghai Jiaotong University, And currently, he is the Assistant Professor in Department of Microelectronics of Fudan University. His research interesting includes information security, video coding and associated VLSI architecture.

Xiaoyang Zeng received the B.S. degree from Xiangtan University, China in 1992, and the Ph.D. degree from Changchun Institute of Optics and Fine Mechanics, Chinese Academy of Sciences in 2001. From 2001 to 2003, he worked as a post-doctor researcher at the State Key Lab of ASIC & System, Fudan University, P.R. China. Then he joined the faculty of Department of Micro-electronics at Fudan University as an professor. His research interests in signal processing, and communication systems. Prof. Zeng is the Chair of Design-Contest of ASP-DAC 2004 and 2005, also the TPC member of several international conferences such as ASCON 2005 and A-SSCC 2006, etc.