Circuits Syst Signal Process (2015) 34:1931–1945
DOI 10.1007/s00034-014-9933-z

# Flexible Architecture Design for H.265/HEVC Inverse Transform

**Grzegorz Pastuszak**

Received: 26 February 2014 / Revised: 28 October 2014 / Accepted: 29 October 2014 /
Published online: 13 November 2014
© The Author(s) 2014. This article is published with open access at Springerlink.com

**Abstract** Highly-efficient video coding involves a great amount of computations. Hardware encoders apply different parallelization techniques to satisfy real-time requirements. This paper describes a novel design methodology for the 2D inverse transform used in the H.265/HEVC hardware decoder and encoder. To support different transform sizes, matrix multiplications are decomposed into some steps based on the division of transform blocks into fixed-size subblocks. The assumed order of processed subblocks along with separate transform cores assigned to both dimensions allows a significant reduction of the size of the transposition buffer, which in turn decreases the resource consumption of the whole architecture. The decomposition enables different hardware configurations of the architectures. Particularly, configuration parameters enable the tradeoff between resources and throughput, the interface adaptation to desired horizontal and vertical sizes, and the availability of particular transform sizes. Two versions of the architecture are developed for FPGA and ASIC technologies. Synthesis results show that they can operate at 200 and 400 MHz when implemented in FPGA Arria II and TSMC 90 nm, respectively.

**Keywords** H.265/HEVC · Video coding · Discrete cosine transform (DCT) · FPGA · VLSI

## 1 Introduction

The latest research and standardization efforts in video coding have lead to a new specification, called H.265/High Efficiency Video Coding (HEVC) [6,8]. It provides

G. Pastuszak (✉)
Institute of Radioelectronics, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: G.Pastuszak@ire.pw.edu.pl

a significant rate-distortion improvement over its predecessor H.264/AVC [7] at the price of the increased computational complexity. Although the general structure of encoder and decoder remains the same, there are many changes in the algorithm of each module. In the case of the transformation, H.265/HEVC specifies larger block sizes. Apart from $4 \times 4$ and $8 \times 8$ blocks, the transformation can operate on $16 \times 16$ and $32 \times 32$ blocks. The larger blocks involve the increased size of the transposition buffer between vertical and horizontal stages of the 2D transform. As a consequence, the hardware architecture computing such a transform consumes a significant amount of resources. Another improvement in H.265/HEVC is the use of better integer basis functions (included in transform matrices) which approximate original Discrete Cosine Transform (DCT). However, transform matrix elements have the extended range of values (from $-90$ to 90), which involves multiplications. In the case of $4 \times 4$ intra blocks, H.265/HEVC imposes the use of the integer approximation of Discrete Sine Transform (DST).

In the literature, there are some architectures proposed for the computation of DCT/IDCT for H.265/HEVC. Some of them do not support all transform sizes [5, 9, 10, 12]. All the designs embed (or assume [2]) the full-size transposition buffer ($32 \times 32$ samples) implemented either as a register matrix [4,5,11,15] or memory modules [1,13–15]. For the register versions, the reported logic area is equivalent to 183k [12], 125k [14], or 126k [15] basic gates. As the designs are dedicated for specific throughputs, they cannot trade resources for the throughput. Implementations embedding regular multipliers and the full-size transposition buffer involve a large amount of hardware resources. To save them, some designs implement multiplications with constant multipliers [4,5,9,14]. This approach is area-efficient when the output contains the number of samples/coefficients equal to the 1D transform size (i.e. 32) [13]. However, this output format is inconvenient and requires additional registers for the adaptation to smaller sizes.

This work presents the novel design methodology for hardware-configurable IDCT architectures. The methodology uses regular multiplication units and applies the decomposition of matrix multiplication into steps performed in successive clock cycles on fixed-size subblocks distinguished in the input and output domains. The assumed order of subblocks computed in the pipeline at both transform stages (vertical and horizontal) allows the decreased size of the transposition buffer. Depending on the configuration, the buffer resources are reduced by 54–88 %. As a consequence, the architecture outperforms other designs in terms of the area-speed efficiency. Moreover, the achieved latencies are the lowest. The configurability enables the tradeoff between logic resources and throughput, the input/output interface adaptation, and the selection of supported transform sizes/types.

## 2 IDCT Basics

In compression algorithms, DCT is often used to compact the signal energy in a limited number of coefficients. On the other hand, IDCT restores the signal from the coefficients. 1D transforms are computed by the multiplication of two matrices. One of these matrices include input data included in the $N \times N$ transform block X. The second matrix specifies basis functions constant for a given transform type. The 1D vertical inverse transform on the $N \times N$ block X can be written as follows:

$$Y(i,j) = \sum_{k=0}^{N-1} C(k,i) * X(k,j) \tag{1}$$

where C and Y are the transform matrix and the result, respectively. This operation involves $N^3$ multiplications. In IDCT, even and odd rows in the transform matrix C are symmetric and anti-symmetric, respectively (see $16 \times 16$ matrix below). This property can be utilized to decompose the transform formula into two cases having the same partial sums:

$$Y(i,j) = \sum_{k=0}^{N/2-1} C(2k,i) * X(2k,j) + C(2k+1,i) * X(2k+1,j) \tag{2}$$

$$Y(N-i-1,j) = \sum_{k=0}^{N/2-1} C(2k,i) * X(2k,j) - C(2k+1,i) * X(2k+1,j) \tag{3}$$

where $i$ = 0,1,..., N/2 1.

This type of the decomposition (using butterfly structures) reduces the number of multiplications by half. In the case of the original DCT/IDCT, the transform can be recursively decomposed according to Chen's [3] scheme to reduce the number of multiplications. This scheme cannot be applied in H.265/HEVC since the standard specifies the integer approximation of the DCT matrix. Moreover, $4 \times 4$ intra blocks are processed using the integer approximation of DST. The H.265/HEVC standard specifies the $32 \times 32$ matrix which is directly applied to compute 32-point DCT and IDCT. Smaller transforms are computed with subsampled versions of the matrix. The $16 \times 16$ matrix provided below is derived by the selection of every second row and first 16 entries from each row. The 2D IDCT is obtained by applying the 1D IDCT vertically and horizontally. The vertical transform is equivalent to the horizontal one in terms of operations performed. Due to the increased size of the transforms, the dynamic range after both stages is limited by downshifting.

$$C = \begin{bmatrix}
64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 & 64 \\
90 & 87 & 80 & 70 & 57 & 43 & 25 & 9 & -9 & -25 & -43 & -57 & -70 & -80 & -87 & 90 \\
89 & 75 & 50 & 18 & -18 & -50 & -75 & -89 & -89 & -75 & -50 & -18 & 18 & 50 & 75 & 89 \\
87 & 57 & 9 & -43 & -80 & -90 & -70 & -25 & 25 & 70 & 90 & 80 & 43 & -9 & -57 & -87 \\
83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 & 83 & 36 & -36 & -83 & -83 & -36 & 36 & 83 \\
80 & 9 & -70 & -87 & -25 & 57 & 90 & 43 & -43 & -90 & -57 & 25 & 87 & 70 & -9 & -80 \\
75 & -18 & -89 & -50 & 50 & 89 & 18 & -75 & -75 & 18 & 89 & 50 & -50 & -89 & -18 & 75 \\
70 & -43 & -87 & 9 & 90 & 25 & -80 & -57 & 57 & 80 & -25 & -90 & -9 & 87 & 43 & -70 \\
64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 & 64 & -64 & -64 & 64 \\
57 & -80 & -25 & 90 & -9 & -87 & 43 & 70 & -70 & -43 & 87 & 9 & -90 & 25 & 80 & -57 \\
50 & -89 & 18 & 75 & -75 & -18 & 89 & -50 & -50 & 89 & -18 & -75 & 75 & 18 & -89 & 50 \\
43 & -90 & 57 & 25 & -87 & 70 & 9 & -80 & 80 & -9 & -70 & 87 & -25 & -57 & 90 & -43 \\
36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 & 36 & -83 & 83 & -36 & -36 & 83 & -83 & 36 \\
25 & -70 & 90 & -80 & 43 & 9 & -57 & 87 & -87 & 57 & -9 & -43 & 80 & -90 & 70 & -25 \\
18 & -50 & 75 & -89 & 89 & -75 & 50 & -18 & -18 & 50 & -75 & 89 & -89 & 75 & -50 & 18 \\
9 & -25 & 43 & -57 & 70 & -80 & 87 & -90 & 90 & -87 & 80 & -70 & 57 & -43 & 25 & -9
\end{bmatrix} . \tag{4}$$

## 3 Methodology

The proposed methodology consists of some key elements described in successive subsections.

### 3.1 Decomposition

The computation of the 1D transform according to Eq. 1 is susceptible to parallelize in three ways. Firstly, a part of input coefficient/sample contributions can be accumulated in one step. Secondly, separate accumulators can be assigned to each output sample/coefficient. Thirdly, some rows/columns can be processed simultaneously due to their independence. By combining the three methods, a number of multiplications and accumulations can be executed in parallel. The three types of the parallelism are orthogonal and specify a data cube processed in one step/phase. Thus, the matrix multiplication can be executed with a number of steps. Since the number of operations performed within the data cube is configurable, they can be efficiently mapped to available computational units. The decomposition of the 1D vertical transform into steps/phases is depicted in Fig. 1. In the figure, W, Hi, and Ho are the subblock width, the input-subblock height, and the output-subblock height, respectively. Loops indexed by variables $i$, $j$, and $k$ correspond to computations performed within one step/phase as described below.

The decomposition divides input and output domains into subblocks as shown in Fig. 2. Particularly, one input subblock is accessed to compute the partial sum for one output subblock in each step/phase. Partial sums are accumulated in some successive steps. In the vertical transform, each output subblock is computed from input subblocks located in the corresponding column. For example, the output subblock indexed by ($X$,2,3) in Fig. 2b is computed by the accumulation of contributions from four input subblocks with indices (0,2,$X$), (1,2,$X$), (2,2,$X$), and (3,2,$X$) in Fig. 2a. Actually, each column of the output subblock is computed from corresponding columns of input subblocks, and several columns are processed simultaneously. Within each column, all input coefficients contribute to each output one. If the height of input subblocks is small, more steps are required. The selection of the size of output subblocks is straightforward as it is sufficient to take into account coordinates within the selected subblock ($i$/$j$ indices in Eqs. 2 and 3). However, the widths of the output and input subblocks must be selected jointly since they are equal to the number of columns processed simultaneously. The total number of multiplications in the vertical transform is equal to W $\times$ Hi $\times$ Ho.

### 3.2 Order Control

The computation of each 1D transform is controlled by three nested loops (see Figs. 1, 2). In the case of the vertical (horizontal) engine, the inner loop determines vertical (horizontal) coordinates of accessed input subblocks (see Fig. 2a). Horizontal and vertical coordinates of output subblocks are determined in the middle and outer loop, respectively (see Fig. 2b). In the vertical processing, the middle loop also controls the

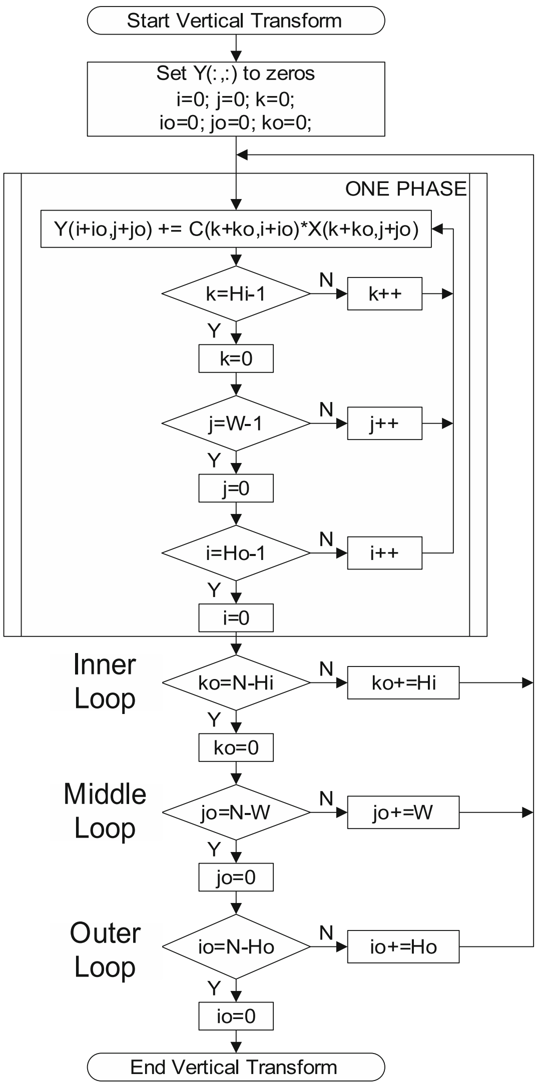

**Fig. 1** Decomposition of the vertical transform into computation phases. Hi, Ho, and W constants are powers of two not greater than N

horizontal position of the input subblock. The order determined by the loops enables processing of successive rows of output subblocks.

The example of the IDCT computation for the $32 \times 32$ transform block is depicted in Fig. 3. The processing is divided into two stages assigned to the vertical and the horizontal transform. In each step of the vertical processing, one $8 \times 8$ input subblock distinguished in the 2D-transform domain is accessed and used to compute the contribution to one $8 \times 4$ output subblock distinguished in the 1D-transform domain. Particularly, each eight-coefficient input column contributes to the corresponding four-coefficient output column. The data cube has the size of $8 \times 8 \times 4$, i.e., 256 multiplications are performed in one step. To compute one $8 \times 4$ output subblock of the vertical

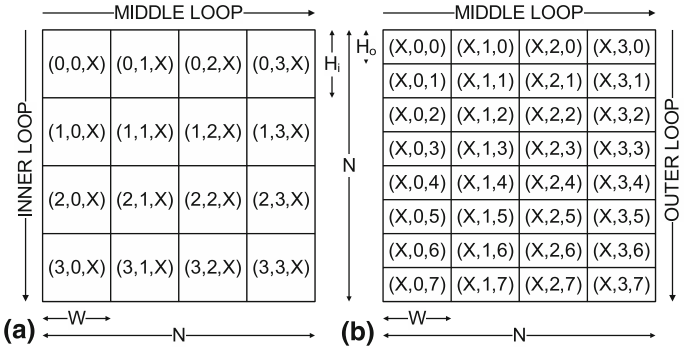
MIDDLE LOOP
INNER LOOP
OUTER LOOP
(0,0,X) (0,1,X) (0,2,X) (0,3,X)
(1,0,X) (1,1,X) (1,2,X) (1,3,X)
(2,0,X) (2,1,X) (2,2,X) (2,3,X)
(3,0,X) (3,1,X) (3,2,X) (3,3,X)
(X,0,0) (X,1,0) (X,2,0) (X,3,0)
(X,0,1) (X,1,1) (X,2,1) (X,3,1)
(X,0,2) (X,1,2) (X,2,2) (X,3,2)
(X,0,3) (X,1,3) (X,2,3) (X,3,3)
(X,0,4) (X,1,4) (X,2,4) (X,3,4)
(X,0,5) (X,1,5) (X,2,5) (X,3,5)
(X,0,6) (X,1,6) (X,2,6) (X,3,6)
(X,0,7) (X,1,7) (X,2,7) (X,3,7)
Hi
Ho
N
W
(a)
(b)

**Fig. 2** Example of the partitioning of the input (**a**) and output (**b**) domains into subblocks. *Numbers* point to the loop iteration in which a subblock is accessed/computed. The first, the second, and the third index are controlled by the inner, the middle, and the outer loop, respectively

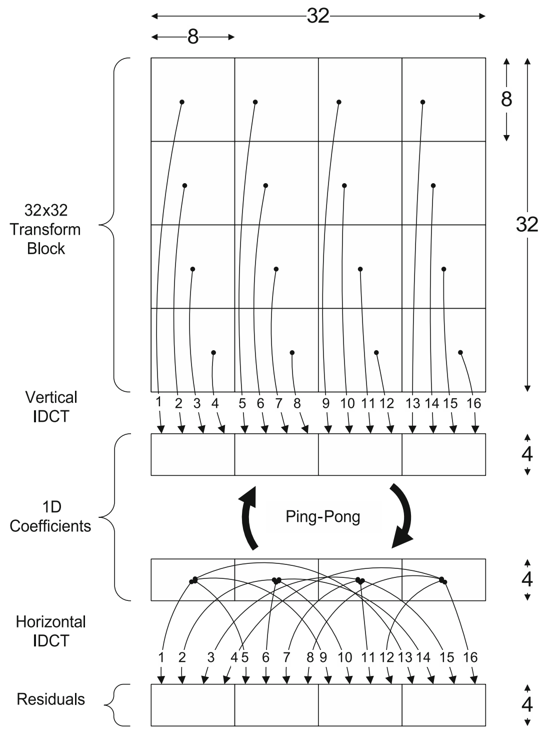

**Fig. 3** Row-oriented computation of the 2D 32 × 32 transform. The input domain is divided into 8 × 8 subblocks read subsequently. *Arrows* with *numbers* indicate the order of subblock contributions. The output subblock size at both stages is set to 8 × 4

stage, contributions from four 8 × 8 input subblocks located in the same column are accumulated in successive steps. Horizontally-successive 8 × 4 subblocks are computed in the same way with the order specified by numbers (1–16) labeling arrows in Fig. 3. When all these subblocks are computed, the resulting row of four subblocks (32 × 4 coefficients) is forwarded to the horizontal stage. This operation can be performed with the ping-pong intermediate buffer to allow the stage parallelism. At the horizontal stage, one $8 \times 4$ residual output subblock is computed by the accumulation of contributions from four $8 \times 4$ subblocks stored in the buffer. In particular, each eight-1D-coefficient row contributes to the collocated eight-residual output row. Each of four output subblocks is obtained every fourth step (16 steps required for each row). The processing order is indicated by numbers labeling arrows in Fig. 3. Similarly to the vertical stage, each step of the horizontal stage requires 256 multiplications. The $16 \times 16$ IDCT can be computed in similar way. The main difference is that each $8 \times 4$ intermediate (output) subblock is computed by the accumulation of partial sums obtained from two rather than four input (intermediate) subblocks.

The number of iterations of each loop is proportional to the transform size N and inversely proportional to the width or height of input/output subblocks (Wi, Wo, Hi, or Ho). For the vertical and horizontal transform, the total number of iterations can be expressed as $N^3/(W \times Hi \times Ho)$ and $N^3/(H \times Wi \times Wo)$, respectively. $H$ denotes the height of the input and output subblocks. In reality, the number of iterations must not be less than one, i.e., at least one iteration is indispensable to perform the transformation. However, the expression can give the result smaller than one if the transform size is small and subblock sizes are large. It means that resources are not utilized fully for computations. To achieve a better utilization, one larger (e.g. $8 \times 8$) output block composed of four smaller (e.g. $4 \times 4$) transform blocks should be computed. For example, four $4 \times 4$ transforms can be computed in parallel if the subblock size specified at the output interface is $8 \times 8$. Generally, different transform sizes can be computed on fixed-size subblocks by changing multiplication coefficients.

## 3.3 Transposition Buffer Reduction

The proposed methodology assumes the use of separate pipelined engines for both transform dimensions. The output subblock size used at the horizontal stage is the same as at the vertical one. This feature enables the balance between throughputs of both stages and facilitates the management of the transposition buffer, as explained later. The computation of successive rows of output subblocks by the vertical transform and the use of separate pipelined engines for both transform dimensions allow the decreased height of the transposition buffer. Particularly, each row of subblocks can be forwarded to horizontal processing immediately after the vertical engine finishes the computation of the last subblock in this row (see Fig. 3). With the assumption that vertical and horizontal engines operate on separate parts of the buffer in parallel, the buffer should keep two rows of subblocks. Hence, its height can be decreased to 2Ho. If Ho after the horizontal stage was greater than that after the vertical stage, the buffer size would have to be increased. If the heights are different, additional multiplexing is indispensable to switch between coefficient rows. Provided that the subblock sizes are powers of two, the proposed methodology has the advantage for Ho not greater than N/4. For Ho equal to N/2, the transposition buffer has the size the same as in the traditional approach. Also, the full-size transposition buffer would be indispensible if the 1D transform stages did not compute subblock rows.

### 3.4 Interface Adaptation

In the decoder proposed by Tikekar et al. [14], the accumulation following the adder tree enables only the adaptation of number of input samples/coefficients. The proposed methodology is more flexible as it enables the adaptation to the output interface. In particular, configuration constants at the HDL level determine the width and the height of subblocks computed/released by the transform module. The size of the input interface is dependent directly on the output one as the width constant is common for them. To obtain the efficient fitting to DSP units in FPGA, the height of the input matrix (vertical Hi) is fixed to eight in the implemented architecture. However, the proposed design methodology is not limited by this choice in general.

### 3.5 Butterfly Restrictions

As described in the previous section, the number of multiplications can be decreased by half with one butterfly decomposition. To benefit from this optimization, the height of the output subblock should be greater than 1. As a consequence of the butterfly operation, output subblocks after the 1D transform consist of two parts symmetric to the center of the $N \times N$ transform result. The parts correspond to Eqs. 2 and 3 (opposite indexing of Y rows). The same rule refers to column pairs after the horizontal transformation. Thus, except for particular cases, each output subblock after 2D transform is composed of four not-adjacent rectangular parts. In order to reconstruct the data coherence, the parts must be placed to relevant locations within the $N \times N$ area kept in the output buffer.

## 4 Architecture

The general IDCT architecture applying the proposed methodology and its timing diagram are depicted in Figs. 4 and 5, respectively. Although the timing diagram is shown for the ASIC implementation, its FPGA version is similar. In particular, single delay stages used before multiplication registers shift right the timing one and two clock cycle for the vertical and horizontal stages, respectively. The diagram illustrates computational dependencies between registers/stages distinguished in Fig. 4. Most transactions are based on the pipeline dependencies. However, the data in the second queue are transferred in the ring unless the queue is not reloaded. Details concerning computations are described in the following paragraph.

The IDCT communicates with two external buffers (storing dequantized coefficients and reconstructed residuals) to allow the flexibility in communication with the entropy decoder at the input and the reconstruction module at the output. The architecture consists of three main submodules: the vertical transformation, the transposition buffer, and the horizontal transformation. Transformation engines are pipelined to maximize the clock rate. The first stage embeds multiplications, whereas the second includes the adder tree with the accumulator. The third stage performs the butterfly operation followed by rounding. The separate rounding adder is not present in the ASIC version. Apart from the arithmetic operations, the transformation submodules

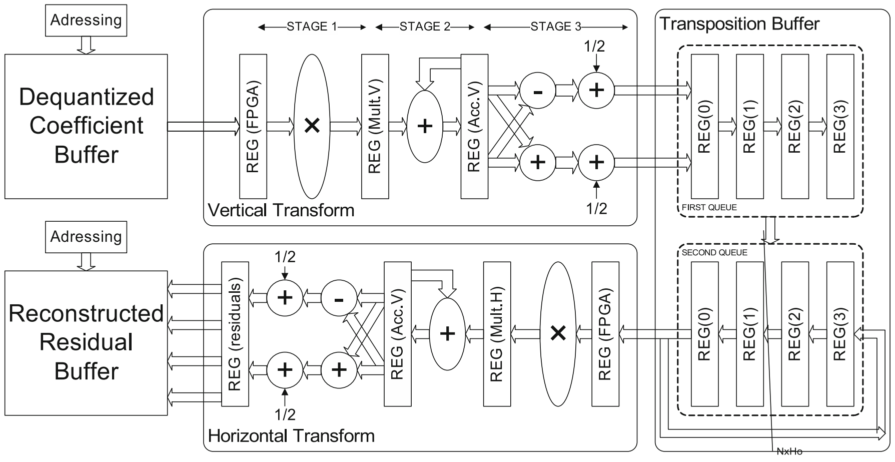

**Fig. 4** General IDCT architecture. Registers distinguished as FPGA are implemented only in FPGA

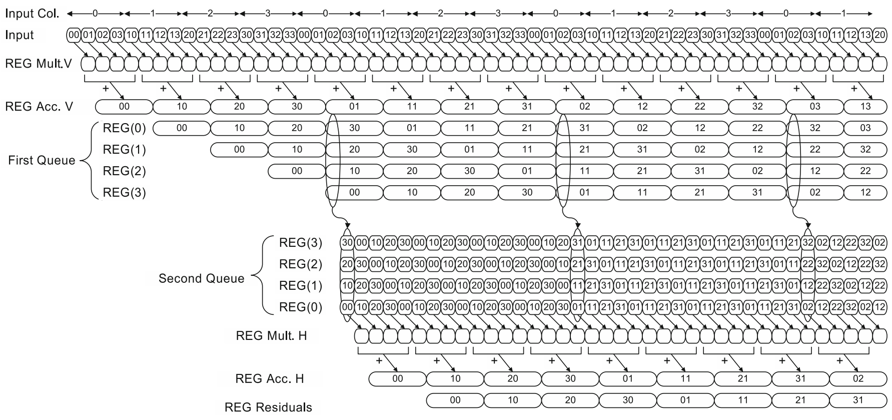

**Fig. 5** Timing diagram of the 2D 32 × 32 inverse transform (ASIC version). Subblocks are indexed by *horizontal* and *vertical* coordinates. Input, queue, and residual indices corresponds to input, 1D-transformed, and residual subblocks, respectively. For clarity, indices identify only top-left subblocks formed by *butterfly* operations

embed the control logic. It provides coordinates for input/output subblocks according to the order determined by three nested loops, as described in the previous section. The vertical transform engine reads dequantized coefficients as $W \times Hi$ subblocks from the external memory buffer. Vertically-transformed $W \times Ho$ subblocks are written to the first register queue in the transposition buffer. The number of writes for one row is equal N/W. Figure 4 shows the case for four writes. Both register queues have the height of Ho. When the processing of one row of subblocks ($N \times Ho$) is finished and the horizontal transform engine is ready, the content of the first queue is written to the second one. Next, the vertical transformation starts to write data from the following row of subblocks to the same queue, whereas the horizontal transformation reads the second queue. Since the same data have to be accessed for horizontally-consecutive

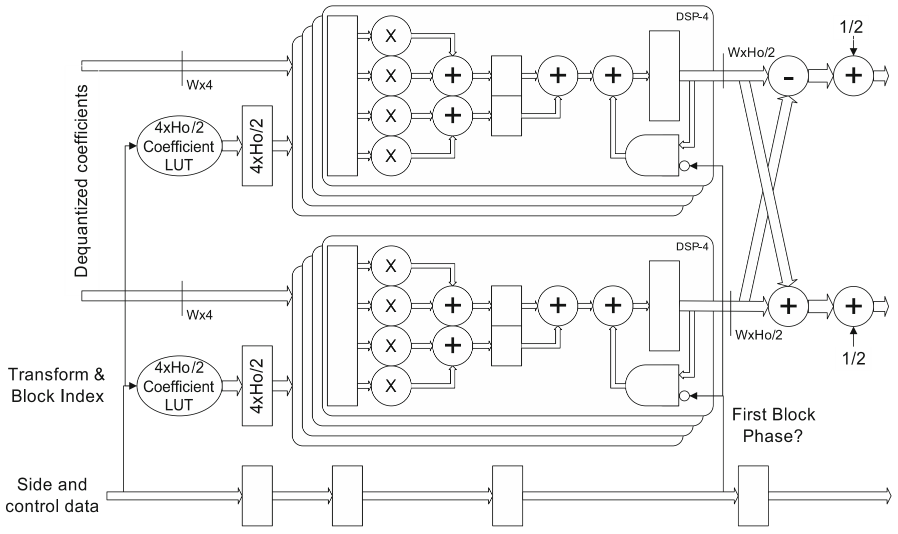

**Fig. 6** FPGA architecture of the 1D (*vertical*) transformation. *Rectangles* correspond to registers

output subblocks, the second queue is designed as a ring buffer including four register sections. As a consequence, four read accesses are needed to compute one residual subblock ($W \times Ho$) for $32 \times 32$ transform. For smaller transform sizes, only part of registers are used, and the number of reads is decreased. Reconstructed residuals produced by the horizontal transformation are stored to the buffer. As the output subblock is composed of four not-adjacent rectangular parts, the 2D coherency should be restored in the reconstruction module. The restoration can be performed by the selection of one part at the read port. Although the maximal throughput is decreased four times, it can match the throughput of transform modules for the largest transform size ($32 \times 32$). To increase reading rate, the buffer for reconstructed residuals can be divided into parts.

In FPGA technology, there are modules dedicated for typical DSP operations such as multiplications and accumulations. They allow designs to achieve higher performance in terms of speed and resources. Particularly, the modules can be pipelined to increase clock frequencies. Since the transformation involves a great amount of multiplications and accumulations, it is beneficial to utilize DSP modules when implementing the design in FPGA devices. However, the architecture optimized for ASIC technology is not always suitable to obtain the best performance in FPGA, and vice versa. Moreover, the same rule can apply to FPGA families from different vendors.

The architecture of the 1D transformation using Altera DSP modules is depicted in Fig. 6. Although Hi/2 is fixed to four to fit FPGA resources, it can be changed if needed. The 1D engine incorporates $W \times Ho$ DSP-4 units, each of which embeds four multiplications, three adders, and one accumulator. The operations are performed at two pipeline stages. There are three main differences with respect to the ASIC-optimized design: the presence of input registers (labeled as FPGA), moving some adders to the first stage (just after multiplications), and the use of separate adders for the final rounding. These modifications enable fitting to the structure of DSP modules available in Altera devices. The use of input registers (embedded in DSP) is indispensible to eliminate the impact of signal delay inferred from routing. In ASIC-oriented design, the rounding is implemented at the accumulation stage by providing an appropriate argument to the adder in the first cycle (phase). Totally, the 2D transform implemented in FPGA embeds $8 \times W \times Ho$ multipliers and $12 \times W \times Ho$ adders ($10 \times W \times Ho$ adders in ASIC).

## 5 Implementation Results

Two versions of the IDCT architecture are specified in VHDL and verified with the HM 13.0 reference model [6]. The first version is dedicated for FPGA devices as they embed DSP units. Each block includes pipelined multiplications and adder trees. The second is optimized for ASIC designs. The synthesis is performed for FPGA and ASIC technologies using the Altera Quartus II software and Synopsys Design Compiler, respectively. Particularly, FPGA synthesis is performed for Arria II GX FPGA devices, whereas TSMC 90 nm is selected as the ASIC technology. In the second case, the design takes advantage of clock gating to reduce power consumption. Achieved frequencies are 200 and 400 MHz for the FPGA and ASIC technologies, respectively.

Table 1 shows the resource consumption for different sizes of output subblocks and corresponding throughputs for different transform sizes ($8 \times 8/16 \times 16/32 \times 32$). The maximal throughput is achieved in the case of the $8 \times 8$ subblock size and the $8 \times 8$ transform. Particularly, only one iteration of all loops is required to compute result, i.e., one $8 \times 8$ transform block can be processed in each clock cycle. Throughputs for particular configurations show that the design allows the decoder to support different video resolutions. Provided the most-computationally-complex case (i.e., only $32 \times 32$ luma transforms and $16 \times 16$ chroma transforms), the $2 \times 2$ and $8 \times 8$ configurations have the throughput of 0.8 and 12.8 pixels per clock cycle, respectively. If the first (slowest)

**Table 1** Resource consumption for different configurations of the output interface

| | | TSMC 90 nm [gate] | | Arria II GX [ALUT] | | |
|---|---|---|---|---|---|---|
| Subblocks size | Througput | Total | Buffer | Total | Buffer | [DSP] |
| 8x8 | 64/32/16 | 689,972 | 58,698 | 11,023 | 8,465 | 256 |
| 8x4 | 32/16/8 | 346,590 | 29,568 | 5,834 | 4,271 | 128 |
| 4x8 | 32/16/8 | 379,542 | 58,648 | 9,853 | 8,232 | 128 |
| 4x4 | 16/8/4 | 190,012 | 29,488 | 5,240 | 4,130 | 64 |
| 8x2 | 16/8/4 | 175,042 | 14,764 | 3,053 | 2,073 | 64 |
| 2x8 | 16/8/4 | 221,812 | 58,693 | 9,215 | 8,224 | 64 |
| 4x2 | 8/4/2 | 96,575 | 14,755 | 2,810 | 2,067 | 32 |
| 2x4 | 8/4/2 | 111,180 | 29,499 | 4,846 | 4,111 | 32 |
| 2x2 | 4/2/1 | 57,410 | 14,762 | 2,531 | 2,057 | 16 |

**Table 2** Resource consumption for different configurations of supported transform sizes

| Transform size | | | | Arria II GX | TSMC 90 nm |
|---|---|---|---|---|---|
| $4 \times 4$ | $8 \times 8$ | $16 \times 16$ | $32 \times 32$ | IDCT $8 \times 2$ | IDCT $8 \times 2$ |
| × | × | × | × | 3,075 | 175 042 |
| | × | × | × | 3,079 | 174 933 |
| | | × | × | 3,075 | 174,915 |
| | | | × | 2,980 | 174,061 |
| × | × | × | | 1,946 | 156,315 |
| × | × | | | 1,353 | 130,759 |
| | × | × | | 1,958 | 152,425 |

configuration operates at 200 MHz (i.e. FPGA), the achieved throughput of 160M pixels/sec enables the support for 1080p@60fps videos (required 125M pixels/sec). The fastest $8 \times 8$ configuration clocked at 400 MHz (i.e. ASIC) has the throughput of 5120M pixels/sec allowing the support for 7680p@120fps videos (required 3982M pixels/sec).

The synthesis results included in Table 1 show that the dependence between throughput and resource consumption is almost proportional. The proportionality stems from the fact that the number of DSP-4 units (main contribution) is equal to the doubled size of the output subblock. At the same throughput, the design requires less resource when decreasing the subblock height. This stems from the fact that the size of the transposition buffer depends on the output-subblock height rather than the width. The impact of the buffer size on the resource consumption is strong for FPGA, as each flip-flop must be mapped into a separate logic element (ALUT). Thus, the full-size transposition buffer would consume 16k logic elements, which is much more than the whole IDCT architecture with the $8 \times 8$ output. If the output subblock height Ho is set to two, the buffer needs about 2k logic elements, i.e. the reduction of 87 % is achieved. In the case of AISC technologies, such a buffer consists of 16k flip-flops and corresponding two-input multiplexers. For the TSMC 90 nm general-purpose nominal-threshold-voltage library, the smallest area of one pair is equivalent to eight basic gates. Therefore, the buffer consumes resources equivalent to 128k gates. Buffer sizes reported in Table 1 are significantly smaller, i.e., the buffer is decreased by about 54–88 %.

Table 2 shows the resource consumption for different configurations of supported transform sizes. The results are provided for the output subblock size of $8 \times 2$ samples/coefficients. Note that the number of DSP units in the FPGA implementation is constant and equal to 64 (128 multiplications). If the 32-point transform is supported, the resource consumption is not changed significantly regardless of the configuration for the remaining ones. This observation stems from the fact that look-up tables for transform coefficients take relatively a small amount of resources. Therefore, the extended support for other symmetric transform matrices should slightly affect the hardware complexity provided that the coefficient accuracy and transform sizes are kept constant. On the other hand, removing the support for large transforms

**Table 3** Comparison of different IDCT architectures

| Design | Zhu [15] | Budagavi [2] | Shen [13] | Park [12] |
|---|---|---|---|---|
| Technology | TSMC 90 nm | 45 nm | SMIC 0.13 m | 0.18 m |
| Gate count (k) | 412.3/320 | 130 | 109.2 | 287 |
| Memory (bit) | 0/16,384 | – | 18,944 | – |
| Clock Frequency (MHz) | 311 | 250 | 350 | 300 |
| Throughput 2D (samples/clock) | 4/8/16/32 | 2/4/8/16 | 2 | 2.1289 |
| Latency (clock) | 38 | > 32 | 19/68/261 | 481 |
| Transposition buffer | registers/ memory | not included | memory | $32 \times 32$ registers |
| Power (mW) | 61.5/89.4 | n.a. | n.a. | n.a. |
| Functionality | IDCT, DCT, & Hadamard | IDCT | Multistandard IDCT | $16 \times 16$ and $32 \times 32$ IDCT |
| **Design** | **Tikekar [14]** | **Chiang [4]** | **This work** | **This work** |
| Technology | TSMC 40 nm | 90 nm | TSMC 90 nm | TSMC 90 nm |
| Gate count (k) | 121.1 | 63.8(core)+70 | 175.0 | 96.6 |
| Memory (bit) | 16,384 | – | – | – |
| Clock Frequency (MHz) | 200 | 270 | 400 | 400 |
| Throughput 2D (samples/clock) | 2 | 2/2.67/2/1.39 | 16/16/8/4 | 8/8/4/2 |
| Latency (clock) | 256 | 8/16/72/416 | 7/7/10/22 | 8/8/14/38 |
| Transposition buffer | memory + 36 registers | $32 \times 32$ | registers | $32 \times 4$ registers |
| Power (mW) | n.a. | n.a. | 40.8 | 22.2 |
| Functionality | IDCT and IDST$-4 \times 4$ | IDCT | IDCT and IDST$-4 \times 4$ | |

significantly reduces the amount of resources. The main reason is the decrease of the transposition buffer.

Table 3 provides the comparison of different IDCT/DCT architectures described in literature. If applicable, throughputs are provided for different transform sizes ($4 \times 4/8 \times 8/16 \times 16/32 \times 32$). Two versions of the proposed architecture having throughputs close to those of referred designs are provided (subblock size equal to $8 \times 2$ and $4 \times 2$). The proposed designs can operate at the highest frequencies. The proposed methodology allows the lowest latencies. Owing to the decreased transposition buffer, the proposed architectures consume less resource at similar or higher throughputs. Although some architectures incorporating memories needs less gates, their total resource cost should be increased significantly. In particular, the area of the 16 kb on-chip memory divided into 32 modules is equivalent to 108.1k gates [15]. For lower throughputs, the number of memories can be traded for wider data widths [13] or additional registers used before writing [14]. When four memory modules are employed, their gate equivalent can be reduced about three times. If the single-port memory is employed, the reduction is a little more. Although memory-based designs [13,14] require a similar amount of resources compared to the proposed architecture with the subblock size set to $4 \times 2$, the throughput of the latter is significantly higher.

As IDCT process data in different order than neighboring modules (e.g. entropy decoder and prediction/reconstruction), buffers are indispensible to form the processing path [14]. Architectures based on the 32-point core require the adaptation of interfaces to the subblock processing order. In the proposed methodology, the adaptation is natural since the subblock width and height are configurable. On the other hand, the input buffer needs to be continuously accessed.

## 6 Conclusion

This paper proposes the flexible architecture design for the inverse transform used in H.265/HEVC. The design methodology can also be applied in the development of the forward transform. The methodology enables the reduction of the transposition matrix, the resource-throughput tradeoff, and the interface adaptation to desired subblock sizes. Although regular multiplication units are used, the methodology allows the design to consume less resource compared to other approaches. Furthermore, architectures can be easily adapted to support other transform sizes and types. If the additional transform matrix is symmetric, the change would only affect the LUT providing matrix coefficients.

**Acknowledgments** This work is supported by the Foundation for the Development of Radiocommunications and Multimedia Technologies.

**Open Access** This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

## References

1. A. Ahmed, M.U. Shahid, A.U. Rehman, N point DCT VLSI architecture for emerging HEVC standard. VLSI Des. **2012**, 6 (2012)
2. M. Budagavi, V. Sze, Unified forward+inverse transform architecture for HEVC. in Proceedings IEEE International Conference on Image Processing (ICIP), (2012), pp. 209–212
3. W.H. Chen, C.H. Smith, S.C. Fralick, A fast computational algorithm for the discrete cosine transform. IEEE Trans. Commun. **25**(9), 10041009 (1977)
4. P.-T. Chiang, T. S. Chang, A reconfigurable inverse transform architecture design for HEVC decoder. in Proceedings IEEE International Symposium on Circuits and Systems (ISCAS 2013), (2013), pp. 1006–1009
5. A. Edirisuriya, A. Madanayake, R.J. Cintra, F.M. Bayer, A multiplication-free digital architecture for 16×16 2-D DCT/DST transform for HEVC. in Proceedings IEEE 27th Convention of Electrical & Electronics Engineers in Israel, (2012)
6. HEVC software repository - HM-13.0 reference model
7. ITU-T Recommendation H.264 and ISO/IEC 14496–10 MPEG-4 Part 10, Advanced Video Coding (AVC), (2003)
8. ITU-T Recommendation H.265 and ISO/IEC 23008–2 MPEG-H Part 2, High Efficiency Video Coding (HEVC), (2013)

9. R. Jeske, J. C. de Souza, G. Wrege, R. Conceiao, M. Grellert, J. Mattos, L. Agostini, Low cost and high throughput multiplierless design of a 16 point 1-D DCT of the new HEVC video coding standard. in Proceedings VIII Southern Conference on Programmable Logic, (2012)
10. M. Martuza, K. Wahid, A cost effective implementation of $8 \times 8$ transform of HEVC from H.264/AVC. in Proceedings IEEE Canadian Conference on Electrical & Computer Engineering, (2012)
11. P.K. Meher, S.Y. Park, B.K. Mohanty, K.S. Lim, C. Yeo, Efficient integer DCT architectures for HEVC. IEEE Trans. Circuits Syst. Video Technol. **24**(1), 168178 (2014)
12. J.-S. Park, W.-J. Nam, S.-M. Han, S. Lee, 2-D large inverse transform (16x16, 32x32) for HEVC (High efficiency video coding). J. Semicond. Technol. Sci. **12**(2), 203–211 (2012)
13. S. Shen, W. Shen, Y. Fan, X. Zeng, A unified 4/8/16/32-point integer IDCT architecture for multiple video coding standards. in Proceedings IEEE International Conference on Multimedia and Expo, (2012) pp. 788–793
14. M. Tikekar, C.-T. Huang, C. Juvekar, V. Sze, A.P. Chandrakasan, A 249-Mpixel/s HEVC video-decoder chip for 4K ultra-HD applications. IEEE J. Solid State Circuits **49**(1), 61–72 (2014)
15. J. Zhu, Z. Liu, D. Wang, Fully pipelined DCT/IDCT/Hadamard unified transform architecture for HEVC codec. in Proceedings IEEE International Symposium on Circuits and Systems (ISCAS 2013), (2013) pp. 677–681