A full-pipelined 2-D IDCT/ IDST VLSI architecture with adaptive block-size for HEVC standard

Hong Liang[a), **He Weifeng**[b), **Zhu Hui, and Mao Zhigang**

School of Microelectronics, Shanghai Jiao Tong University

a) *liang.h.hong@gmail.com*

b) *hewf@sjtu.edu.cn*

Abstract: High Efficiency Video Coding (HEVC) is the currently developing video coding standard beyond H.264/AVC. In this paper, a full pipelined 2-D IDCT/IDST VLSI architecture compatible with HEVC standard is presented for the first time. The proposed architecture supports adaptive block size IDCT from 4×4 to 32×32 pixels as well as IDST while keeping nearly 100% hardware utilization. Using SMIC 65 nm 1P9M technology, the synthesis results show that the architecture achieves the maximum work frequency at 480 MHz and the hardware cost is about 115.8 K Gates. Experimental results show that the proposed architecture is able to deal with real-time HEVC IDCT/IDST of $4\,\mathrm{K} \times 2\,\mathrm{K}$ (4096×2048)@30 fps video sequence at 171 MHz in average. In consequence, it offers a cost-effective solution for the future UHDTV applications.

Keywords: IDCT, IDST, HEVC, Video Coding, VLSI architecture

Classification: Integrated circuits

References

[1] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC Video Coding Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, July 2003.

[2] G. J. Sullivan, J.-R. Ohm, W.-J. Han, and T. Wiegand, "Overview of the High Efficiency Video Coding (HEVC) Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[3] M. T. Pourazad, C. Doutre, M. Azimi, and P. Nasiopoulos, "HEVC: The new gold standard for video compression: How does HEVC compare with H.264/ AVC?," *IEEE Consumer Electron. Mag.*, vol. 1, no. 3, pp. 36–46, July 2012.

[4] M. Martuza and K. Wahid, "A cost effective implementation of 8×8 transform of HEVC from H.264/AVC," *25th IEEE Canadian Conference of Electrical and Computer Engineering*, Montreal, Canada, pp. 1–4, May 2012.

[5] R. Jeske, J. C. de Souza, G. Wrege, R. Conceição, 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," *2012 VIII Southern Conference on Programmable Logic (SPL)*, Bento Gonçalves, Brazil, pp. 1–6, March 2012.

DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

[6] A. Edirisuriya, A. Madanayake, R. J. Cintra, and F. M. Bayer, "A Multiplication-free Digital Architecture for 16×16 2-D DCT/DST Transformation for HEVC," *2012 27th IEEE Convention of Electrical and Electronics Engineers in Israel*, Eilat, Israel, pp. 1–5, Nov. 2012.

[7] J. S. Park, W. J. Nam, S. M. Han, and S. S. Lee, "2-D Large Inverse Transform (16×16, 32×32) for HEVC (High Efficiency Video Coding)," *J. Semiconductor Technology and Science*, vol. 12, no. 2, pp. 203–211, July 2012.

[8] S. Shen, W. Shen, Y. Fan, and X. Zeng, "A Unified 4/8/16/32-point integer IDCT architecture for multiple video coding standards," *Multimedia and Expo (ICME)*, Melbourne, Australia, pp. 788–793, July 2012.

[9] I. K. Kim, J. Min, T. Lee, W. J. Han, and J. H. Park, "Block Partitioning Structure in the HEVC Standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1697–1706, Dec. 2012.

1 Introduction

The Discrete Cosine Transform (DCT) and its Inverse Discrete Cosine Transform (IDCT) are widely employed in video and image compression, being a major component in contemporary media standards. Many high performance video codec standards, such as H.261, H.263, MPEG-1, MPEG-2, MPEG-4 and H.264/AVC [1] as well as the popular image compression standard JPEG, employ 4×4 or 8×8 DCT/ IDCT as their transform domain tools.

High Efficiency Video Coding (HEVC) standard [2] is an emerging video coding standard which is jointly developed by the MPEG and ITU. HEVC provides 50% more bit-rate reduction and high degree of parallelism when compared to H.264/AVC by adopting a variety of efficient coding tools.

Integer DCT/IDCT is the preferred transform technique in the developing HEVC standard. Unlike its predecessors, the DCT/IDCT tool of HEVC supports several block sizes: 4×4, 8×8, 16×16 and 32×32 pixels. Moreover, HEVC also supports the use of the Discrete Sine Transform (DST) and its inverse transform (IDST) in certain intraprediction modes to obtain higher coding gains on residuals that tend to be larger in value for pixels away from the boundaries [3].

Additionally, the computational complexity of DCT/IDCT also increases linearly along with the increase of the transform block size. For example, when the block size enlarges from 4×4 pixel to 32×32 pixel, the computational complexity of IDCT also increases 8 times to the same video sequence. Moreover, some new features such as recursive block partitioning, variable block size, the emergence of beyond-HD formats (e.g., 4 k×2 k or 8 k×4 k resolution) and different transform mode of the HEVC, bring great challenges to IDCT/IDST VLSI architecture design.

Currently, researches on DCT/IDCT architecture design are focused on 4-point or 8-point DCT/IDCT operations. Few papers begin to pay attention to HEVC DCT/IDCT architecture design since 2012.

In 2012, M. Martuza and K. Wahid presented a shared architecture which can compute 8×8 IDCT for HEVC as well as H.264/AVC by using a

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

new mapping technique [4]. R. Jeske also presented a 16-point 1-D DCT multiplierless hardware architecture for HEVC in 2012 [5]. Based on approximation DCT algorithm, A. Edirisuriya proposed a multiplication-free architecture for 16 × 16 point approximation HEVC 2-D DCT/DST transform in 2012 [6]. To support large size IDCT, J.S. Park presented a 16 × 16 and 32 × 32 inverse transform architecture for HEVC standard in 2012 [7]. To support different block size IDCT, S. Shen and W. Shen presented a unified 4/8/16/32-point integer IDCT architecture for HEVC standard [8].

All in all, all these architectures are focused on DCT/IDCT VLSI design for the developing HEVC standard. However, none of them is compatible with HEVC standard fully. Furthermore, some important issues of real HEVC video sequences are not considered by them. For example, According to our statistical results by Using HM-6.0 codec model and PeopleOnStreet 3840 × 2160@30 fps video sequence, the percentage of 16 × 16 block size used for IDCT is 9.1%, 32 × 32 block size used for IDCT almost 1.3%. As a result, high throughput parallel array architecture with huge hardware overhead and access bandwidth for large block size IDCT is not necessary when considering practical HEVC video sequences.

In this paper, we propose a novel full-pipelined 2-D IDCT/IDST VLSI architecture with adaptive block size for HEVC standard for the first time. To improve hardware utilization and to reduce I/O peak access bandwidth with different block size transform, the recursive and regular butterfly computation structure is unrolled. As a result, the architecture can deal with variable block size IDCT from 4 × 4 to 32 × 32 pixels as well as IDST in parallel or in pipeline dynamically with a nearly 100% hardware utilization rate. In consequence, the proposed architecture is a promising solution for HEVC IDCT/IDST with the future UHDTV applications.

The rest of this paper is organized as follows. In Section 2, the HEVC IDCT/IDST algorithm is introduced in brief. Section 3 presents our proposed 2-D IDCT/IDST architecture. The VLSI implementation result and performance analysis are shown in Section 4. Finally, Section 5 gives some concluding remarks.

2 The HEVC IDCT/IDST

The basic processing unit in HEVC is a Coding Tree Unit (CTU) [9] which is a generalization of the H.264/AVC concept of a Macroblock. The CTU is further partitioned into multiple Coding Units (CU) to adapt to various local characteristics. Furthermore, The TU (Transform Unit) specified by the CU is a basic representative block having residual or transform coefficients for applying the integer transform and quantization.

Fig. 1 shows 4 kinds of TU with block size 4 × 4, 8 × 8, 16 × 16 and 32 × 32 pixels partitioned by a CU. The DCT/IDCT algorithm of HEVC

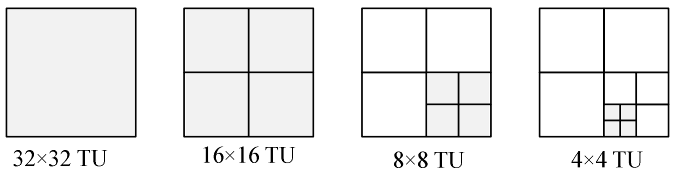

Fig. 1. Illustration of TU splitting types in HEVC

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

supports all these kinds of block size TU but the DST/IDST algorithm only supports 4×4 pixel TU. According to the texture patterns of the image, the HEVC codec picks the best block size and transform mode (DCT or DST) dynamically by illustrating all kinds of block size and transform mode exhaustively to improve the coding efficiency.

Fig. 2 shows a butterfly computation structure of HEVC 16-point 1-D DCT/IDCT. If the data flow is from left to right, the butterfly structure describes DCT operations. If the data flow is from right to left, it describes the inverse transform operations. According to the Figure, 16-point input data is issued to calculate its DCT/IDCT results in parallel for the maximum reuse of internal results. The butterfly computation structure for 8/4-point integer DCT/IDCT is also shown in the figure. 4-point DCT/IDCT is marked with dot line and 8-point DCT/IDCT with dash line. It can be clearly seen that the result of small block size DCT/IDCT can be recursively used for larger block size DCT/IDCT. In consequence, the hardware circuit for 4×4 IDCT may be reused to deal with all larger block size IDCT operations.

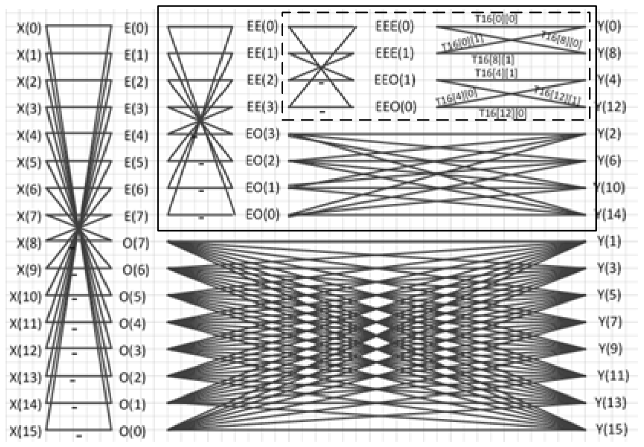

Fig. 2. The butterfly computation structure of 16 point integer 1-D DCT/IDCT

3 The proposed 2-D IDCT/IDST architecture

3.1 System architecture of our proposed HEVC 2-D IDCT/IDST

According to the Fig. 2, the regular butterfly computation structure need input one row/column data in each cycle to calculate its row/column IDCT results in parallel. This is not feasible when the block size is large. To change the granularity of IDCT computation in each cycle, we unroll the butterfly computation structure to remove internal computational correlation first. We illustrate this process with a 4-point HEVC IDCT example as shown in Fig. 3.

Fig. 3 (a) gives an example of input and output pixel data of a transform image with each pixel data represented by a variable. Fig. 3 (b) shows the butterfly computation structure of 4×4 1-D IDCT. Unrolling the butterfly computation structure illustrated in Fig. 3 (b), the output pixel variables of the first row IDCT according to the HEVC standard are given as below:

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

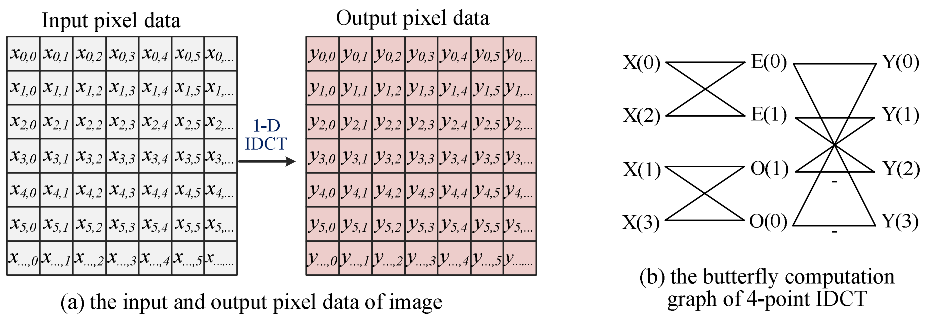
Input pixel data
Output pixel data
1-D IDCT
(a) the input and output pixel data of image
X(0)
X(2)
X(1)
X(3)
E(0)
E(1)
O(1)
O(0)
Y(0)
Y(1)
Y(2)
Y(3)
(b) the butterfly computation graph of 4-point IDCT

Fig. 3. Illustration of the 4 point 1-D IDCT

$$y_{0,0} = 64x_{0,0} + 83x_{0,1} + 64x_{0,2} + 36x_{0,3} \tag{1}$$

$$y_{0,1} = 64x_{0,0} + 36x_{0,1} - 64x_{0,2} - 83x_{0,3} \tag{2}$$

$$y_{0,2} = 64x_{0,0} - 36x_{0,1} - 64x_{0,2} + 83x_{0,3} \tag{3}$$

$$y_{0,3} = 64x_{0,0} - 83x_{0,1} + 64x_{0,2} - 36x_{0,3} \tag{4}$$

Equations (1)∼(4) are a mathematic description of the first row 4-point IDCT results, and internal results such as *E(0)*, *E(1)*, *O(0)* and *O(1)* are unrolled. According to these equations, $y_{0,0} \sim y_{0,3}$ can be calculated in parallel. As a result, the parallelism of 4-point IDCT computation is flexible and changeable to meet real performance requirements.

Using this method, the 4, 8, 16 and 32 point butterfly computation structures of HEVC are unrolled to reduce the hardware cost and to improve the hardware utilization of our IDCT/IDST architecture.

The proposed novel full-pipelined 2-D IDCT/IDST VLSI architecture with adaptive block-size for HEVC standard is shown in Fig. 4.

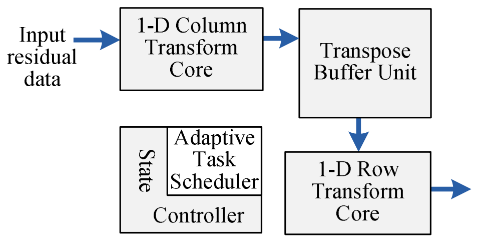

Fig. 4. The proposed 2-D IDCT architecture

Our architecture adopts traditional row-column decomposition approach which includes a 1-D Column Transform Core, a 1-D Row Transform Core, a Transpose Buffer Unit and a State Controller. The two Transform Cores have similar hardware architectures except their data width. These two cores are employed to deal with 1-D row or column IDCT/IDST operations with different block size dynamically. The Transpose Buffer Unit is employed for column-to-row data buffering and transform. The Transpose Buffer Unit includes a 16 bit-to-64 bit Data Packing Unit, a 64 bit Dual Port Memory and a 64 bit-to-16 bit Data Unpacking Unit. Thereinto, the key element is Dual Port Memory which addresses 4 pixel data (each pixel is described as a 16 bit data) at each cycle. The State Controller includes a Adaptive Task Scheduler to monitor the transform block size, transform mode and the state of the Transpose Buffer

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

Unit to issue the transform task to the Cores.

3.2 The architecture of the 1-D transform core

Based on the unrolling approach mentioned above, we develop the architecture of 1-D Column/Row Transform Core. The 1-D Transform Core is based on a 1-D hierarchical linear array architecture. Fig. 5 shows the top level structure of the Transform Core.

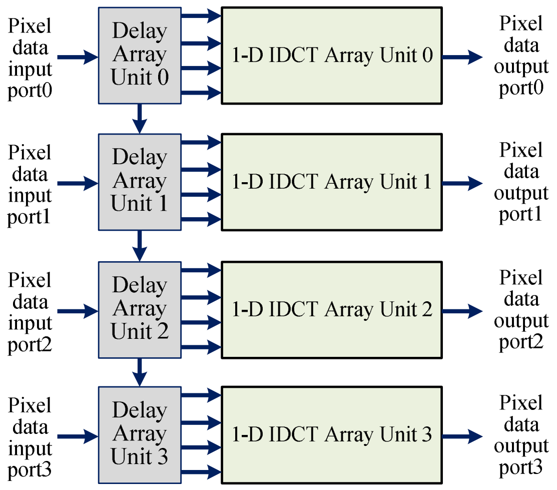

Fig. 5. The architecture of 1-D Column/Row Transform Core

At top level, the Transform Core includes 4 Delay Array Units and 4 1-D IDCT Array Units. There are four input ports to deliver the residual pixel data into the Core. When current block is a 4×4 or 8×8 pixel block, the 1-D Core is divided into four parts and each part includes a Delay Array Unit and a 1-D IDCT Array Unit. The vertical links among Delay Array Units are disabled. In this mode, one column/row data of the pixel block is sent to its Delay Array Unit and 1-D IDCT Array Unit to generate transform results separately. As a result, each 1-D IDCT Array Unit conduct one column/row transform operation individually and the Core supports 4 column/row IDCT/IDST operations in parallel.

If the current block is a 16×16 pixel block, the Core supports 2 column/row IDCT operations in parallel. In this mode, the linear array structure is divided into two sub-arrays and each sub-array include two Delay Array Units and two 1-D IDCT Array Units to conduct one row/column 16-point IDCT individually. Moreover, the Pixel data input port1, the Pixel data input port3 and the vertical link between Delay Array Unit1 with Delay Array Unit2 are disabled.

If the current block is a 32×32 pixel block, the Pixel data input port1, the Pixel data input port2 and the Pixel data input port3 are disabled. In this mode, the Core is a linear systolic array structure to conduct IDCT operation in pipeline.

The 4 Delay Array Units and 4 1-D IDCT Array Units are scheduled to conduct adaptive block size IDCT/IDST dynamically in parallel or in pipeline by the Adaptive Task Scheduler.

At bottom level, the Delay Array Unit and 1-D IDCT Array Unit are also a 1-D linear array structure. Fig. 6 Shows the detailed structure of the Delay Array Unit and 1-D IDCT Array Unit. Each Delay Array Unit use

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

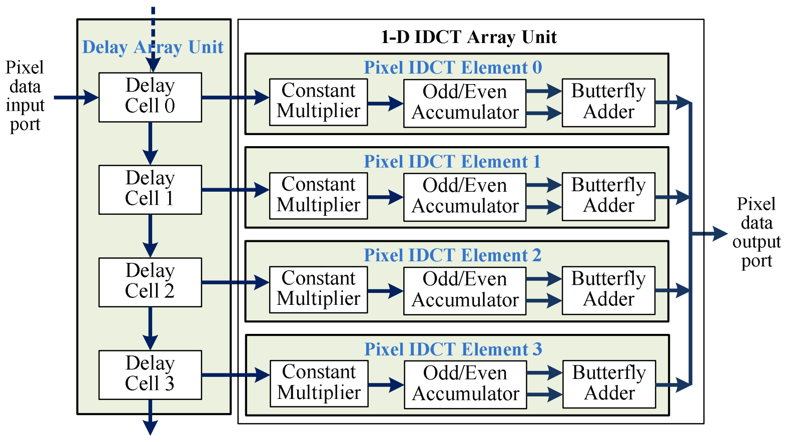

Fig. 6. The detailed structure of Delay Array Unit and 1-D IDCT Array Unit

one of two input ports to receive residual pixel data. Then, the input data is buffered and sent to the 1-D IDCT Array Unit in order by a Delay Cell chain. Each 1-D IDCT Array Unit includes 4 Pixel IDCT Elements and each Pixel IDCT Element is used to calculate one pixel's IDCT/IDST result individually.

The Pixel IDCT Element is the basic function module to calculate a pixel's transform result individually. Each Pixel IDCT Element includes a Constant Multiplier, an Odd/Even Accumulator and a Butterfly Adder as shown in Fig. 6. The Constant Multiplier employs multiplierless structure to deal with constant multiplication operation by shift and add operations. There are 29 constants of HEVC IDCT with different block size and 4 constants of IDST. In Constant Multiplier, all constant multiplications are produced by using shift and add operations.

According to pervious equations (1)∼(4), we can find that the r^{th} output pixel $y_{(r)}$ and the $(N-r-1)^{th}$ (where N is the block size) output pixel $y_{(N-r-1)}$ of the same row share all constant multiplication results except the sign of even number of the results. For example, $y_{0,1}$ (described by equation (1)) accumulates all constant multiplication results of $64x_{0,0}$, $36x_{0,1}$, $-64x_{0,2}$ and $-83x_{0,3}$ while $y_{0,2}$ (described by equation (2)) accumulates the constant multiplication results of $64x_{0,0}$, $-36x_{0,1}$, $-64x_{0,2}$ and $83x_{0,3}$. In consequence, the constant multiplication results of $y_{(r)}$ can be reused to calculate the $y_{(N-r-1)}$.

The Odd/Even Accumulator is employed to accumulate the constant multiplication results of the Constant Multiplier. The detailed structures of the Odd/Even Accumulator and the Butterfly Adder are shown in Fig. 7. In Odd/Even Accumulator, there are two internal data accumulation paths, the Odd pixel Acc path and the Even pixel Acc path to accumulate the

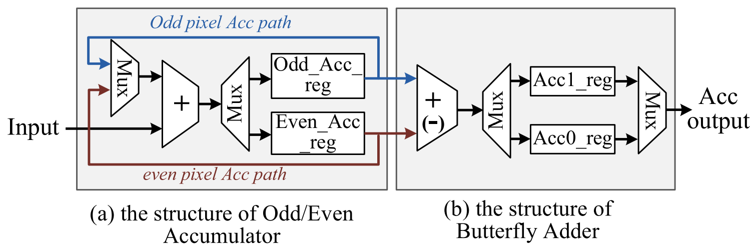

Fig. 7. The detailed structure of Odd/Even Accumulator and Butterfly Adder

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

even and odd number of constant multiplication results separately. For example, if the current Pixel IDCT Element is employed to calculate the pixel point $y_{0,0}=64x_{0,0}+89x_{0,1}+83x_{0,2}+75x_{0,3}+64x_{0,4}+50x_{0,5}+36x_{0,6}+18x_{0,7}$ (8×8 IDCT), the final accumulation result of the $Even\_Acc\_reg=64x_{0,0}+83x_{0,2}+64x_{0,4}+36x_{0,6}$, and the result of the $Odd\_Acc\_reg=89x_{0,1}+75x_{0,3}+50x_{0,5}+18x_{0,7}$.

The Butterfly Adder employs an adder (subtracter), a Acc1_reg register and a Acc0_reg register to generate the final value of output pixel. If the current block mode is 8×8, 16×16 or 32×32 pixel block, the output results $y_{(r)}$ *and* $y_{(N-r-1)}$ *are* generated in pipeline by the Butterfly Adder. In this mode, the output result $y_{(r)}=Odd\_Acc\_reg+Even\_Acc\_reg$ is stored in Acc1_reg and the result $y_{(N-r-1)}=Odd\_Acc\_reg-Even\_Acc\_reg$ is stored in Acc0_reg. By using this accumulation method, the Pixel IDCT Element employs only one constant multiplier to calculate $y_{(r)}$ and $y_{(N-r-1)}$ at the same time. If the current block mode is 4×4 pixel IDCT/IDST block, only $y_{(r)}=Odd\_Acc\_reg+Even\_Acc\_reg$ is calculated by the Adder.

3.3 The adaptive block scheduling scheme for different block size IDCT

The proposed 1-D Column/Row Transform Core is able to deal with different block size IDCT/IDST dynamically in parallel or in pipeline. Fig. 8 illustrates a data flow of the 1-D Column Transform Core with 4×4 IDCT/IDST, 8×8 IDCT and 16×16 IDCT following the pixel data of Fig. 3 (a).

	1-D 4×4 column IDCT/IDST																1-D 8×8 column IDCT								1-D 16×16 IDCT			
	1-D Transformation core				1-D IDCT Array Unit 0								1-D Transformation core				1-D Transformation core				1-D Transformation core				Pixel data input port		Pixel data output port	
	Pixel data input port				IDCT Element input				IDCT element output				Pixel data output port				Pixel data input port				Pixel data output port							
Cycle	port0	port1	port2	port3	port0	port1	port2	port3	port0	port1	port2	port3	port0	port1	port2	port3	port0	port1	port2	port3	port0	port1	port2	port3	port0	port2	port0	port2
0	$x_{0,0}$	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	$x_{0,0}$	0	0	0	0	0	0	0	0	0	0	0	$x_{0,0}$	$x_{0,1}$	$x_{0,2}$	$x_{0,3}$	0	0	0	0	$x_{0,0}$	$x_{0,1}$	0	0
1	$x_{1,0}$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	$x_{1,0}$	$x_{0,0}$	0	0	0	0	0	0	0	0	0	0	$x_{1,0}$	$x_{1,1}$	$x_{1,2}$	$x_{1,3}$	0	0	0	0	$x_{1,0}$	$x_{1,1}$	0	0
2	$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	$x_{2,0}$	$x_{1,0}$	$x_{0,0}$	0	0	0	0	0	0	0	0	0	$x_{2,0}$	$x_{2,1}$	$x_{2,2}$	$x_{2,3}$	0	0	0	0	$x_{2,0}$	$x_{2,1}$	0	0
3	$x_{3,0}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	$x_{3,0}$	$x_{2,0}$	$x_{1,0}$	$x_{0,0}$	0	0	0	0	0	0	0	0	$x_{3,0}$	$x_{3,1}$	$x_{3,2}$	$x_{3,3}$	0	0	0	0	$x_{3,0}$	$x_{3,1}$	0	0
4	$x_{0,4}$	$x_{0,5}$	$x_{0,6}$	$x_{0,7}$	$x_{0,4}$	$x_{3,0}$	$x_{2,0}$	$x_{1,0}$	$y_{0,0}$	0	0	0	$y_{0,0}$	$y_{0,1}$	$y_{0,2}$	$y_{0,3}$	$x_{4,0}$	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	0	0	0	0	$x_{4,0}$	$x_{4,0}$	0	0
5	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$	$x_{1,7}$	$x_{1,4}$	$x_{0,4}$	$x_{3,0}$	$x_{2,0}$	0	$y_{1,0}$	0	0	$y_{1,0}$	$y_{1,1}$	$y_{1,2}$	$y_{1,3}$	$x_{5,0}$	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	0	0	0	0	$x_{5,0}$	$x_{5,1}$	0	0
6	$x_{2,4}$	$x_{2,5}$	$x_{2,6}$	$x_{2,7}$	$x_{2,4}$	$x_{1,4}$	$x_{0,4}$	$x_{3,0}$	0	0	$y_{2,0}$	0	$y_{2,0}$	$y_{2,1}$	$y_{2,2}$	$y_{2,3}$	$x_{6,0}$	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	0	0	0	0	$x_{6,0}$	$x_{6,1}$	0	0
7	$x_{3,4}$	$x_{3,5}$	$x_{3,6}$	$x_{3,7}$	$x_{3,4}$	$x_{2,4}$	$x_{1,4}$	$x_{0,4}$	0	0	0	$y_{3,0}$	$y_{3,0}$	$y_{3,1}$	$y_{3,2}$	$y_{3,3}$	$x_{7,0}$	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	0	0	0	0	$x_{7,0}$	$x_{7,1}$	0	0
8	$x_{4,0}$	$x_{4,1}$	$x_{4,2}$	$x_{4,3}$	$x_{4,0}$	$x_{3,4}$	$x_{2,4}$	$x_{1,4}$	$y_{0,4}$	0	0	0	$y_{0,4}$	$y_{0,5}$	$y_{0,6}$	$y_{0,7}$	$x_{0,4}$	$x_{0,5}$	$x_{0,6}$	$x_{0,7}$	$y_{0,0}$	$y_{0,1}$	$y_{0,2}$	$y_{0,3}$	$x_{8,0}$	$x_{8,1}$	0	0
9	$x_{5,0}$	$x_{5,1}$	$x_{5,2}$	$x_{5,3}$	$x_{5,0}$	$x_{4,0}$	$x_{3,4}$	$x_{2,4}$	0	$y_{1,4}$	0	0	$y_{1,4}$	$y_{1,5}$	$y_{1,6}$	$y_{1,7}$	$x_{1,4}$	$x_{1,5}$	$x_{1,6}$	$x_{1,7}$	$y_{1,0}$	$y_{1,1}$	$y_{1,2}$	$y_{1,3}$	$x_{9,0}$	$x_{9,1}$	0	0
10	$x_{6,0}$	$x_{6,1}$	$x_{6,2}$	$x_{6,3}$	$x_{6,0}$	$x_{5,0}$	$x_{4,0}$	$x_{3,4}$	0	0	$y_{2,4}$	0	$y_{2,4}$	$y_{2,5}$	$y_{2,6}$	$y_{2,7}$	$x_{2,4}$	$x_{2,5}$	$x_{2,6}$	$x_{2,7}$	$y_{2,0}$	$y_{2,1}$	$y_{2,2}$	$y_{2,3}$	$x_{10,0}$	$x_{10,1}$	0	0
11	$x_{7,0}$	$x_{7,1}$	$x_{7,2}$	$x_{7,3}$	$x_{7,0}$	$x_{6,0}$	$x_{5,0}$	$x_{4,0}$	0	0	0	$y_{3,4}$	$y_{3,4}$	$y_{3,5}$	$y_{3,6}$	$y_{3,7}$	$x_{3,4}$	$x_{3,5}$	$x_{3,6}$	$x_{3,7}$	$y_{3,0}$	$y_{3,1}$	$y_{3,2}$	$y_{3,3}$	$x_{11,0}$	$x_{11,1}$	0	0
12	$x_{4,4}$	$x_{4,5}$	$x_{4,6}$	$x_{4,7}$	$x_{4,4}$	$x_{7,0}$	$x_{6,0}$	$x_{5,0}$	$y_{4,0}$	0	0	0	$y_{4,0}$	$y_{4,1}$	$y_{4,2}$	$y_{4,3}$	$x_{4,4}$	$x_{4,5}$	$x_{4,6}$	$x_{4,7}$	$y_{4,0}$	$y_{4,1}$	$y_{4,2}$	$y_{4,3}$	$x_{12,0}$	$x_{12,1}$	0	0
13	$x_{5,4}$	$x_{5,5}$	$x_{5,6}$	$x_{5,7}$	$x_{5,4}$	$x_{4,4}$	$x_{7,0}$	$x_{6,0}$	0	$y_{5,0}$	0	0	$y_{5,0}$	$y_{5,1}$	$y_{5,2}$	$y_{5,3}$	$x_{5,4}$	$x_{5,5}$	$x_{5,6}$	$x_{5,7}$	$y_{5,0}$	$y_{5,1}$	$y_{5,2}$	$y_{5,3}$	$x_{13,0}$	$x_{13,1}$	0	0
14	$x_{6,4}$	$x_{6,5}$	$x_{6,6}$	$x_{6,7}$	$x_{6,4}$	$x_{5,4}$	$x_{4,4}$	$x_{7,0}$	0	0	$y_{6,0}$	0	$y_{6,0}$	$y_{6,1}$	$y_{6,2}$	$y_{6,3}$	$x_{6,4}$	$x_{6,5}$	$x_{6,6}$	$x_{6,7}$	$y_{6,0}$	$y_{6,1}$	$y_{6,2}$	$y_{6,3}$	$x_{14,0}$	$x_{14,1}$	0	0
15	$x_{7,4}$	$x_{7,5}$	$x_{7,6}$	$x_{7,7}$	$x_{7,4}$	$x_{6,4}$	$x_{5,4}$	$x_{4,4}$	0	0	0	$y_{7,0}$	$y_{7,0}$	$y_{7,1}$	$y_{7,2}$	$y_{7,3}$	$x_{7,4}$	$x_{7,5}$	$x_{7,6}$	$x_{7,7}$	$y_{7,0}$	$y_{7,1}$	$y_{7,2}$	$y_{7,3}$	$x_{15,0}$	$x_{15,1}$	0	0
16	$x_{0,8}$	$x_{0,9}$	$x_{0,10}$	$x_{0,11}$	$x_{0,8}$	$x_{7,4}$	$x_{6,4}$	$x_{5,4}$	$y_{4,4}$	0	0	0	$y_{4,4}$	$y_{4,5}$	$y_{4,6}$	$y_{4,7}$	$x_{0,8}$	$x_{0,9}$	$x_{0,10}$	$x_{0,11}$	$y_{0,4}$	$y_{0,5}$	$y_{0,6}$	$y_{0,7}$	$x_{0,2}$	$x_{0,3}$	$y_{0,0}$	$y_{0,1}$
17	$x_{1,8}$	$x_{1,9}$	$x_{1,10}$	$x_{1,11}$	$x_{1,8}$	$x_{0,0}$	$x_{7,4}$	$x_{6,4}$	0	$y_{5,4}$	0	0	$y_{5,4}$	$y_{5,5}$	$y_{5,6}$	$y_{5,7}$	$x_{1,8}$	$x_{1,9}$	$x_{1,10}$	$x_{1,11}$	$y_{1,4}$	$y_{1,5}$	$y_{1,6}$	$y_{1,7}$	$x_{1,2}$	$x_{1,3}$	$y_{1,0}$	$y_{1,1}$
18	$x_{2,8}$	$x_{2,9}$	$x_{2,10}$	$x_{2,11}$	$x_{2,8}$	$x_{1,8}$	$x_{0,0}$	$x_{7,4}$	0	0	$y_{6,4}$	0	$y_{6,4}$	$y_{6,5}$	$y_{6,6}$	$y_{6,7}$	$x_{2,8}$	$x_{2,9}$	$x_{2,10}$	$x_{2,11}$	$y_{2,4}$	$y_{2,5}$	$y_{2,6}$	$y_{2,7}$	$x_{2,2}$	$x_{2,3}$	$y_{2,0}$	$y_{2,1}$
19	$x_{3,8}$	$x_{3,9}$	$x_{3,10}$	$x_{3,11}$	$x_{3,8}$	$x_{2,8}$	$x_{1,8}$	$x_{0,0}$	0	0	0	$y_{7,4}$	$y_{7,4}$	$y_{7,5}$	$y_{7,6}$	$y_{7,7}$	$x_{3,8}$	$x_{3,9}$	$x_{3,10}$	$x_{3,11}$	$y_{3,4}$	$y_{3,5}$	$y_{3,6}$	$y_{3,7}$	$x_{3,2}$	$x_{3,3}$	$y_{3,0}$	$y_{3,1}$

Fig. 8. The data flow of the proposed 1-D Column Transform Core with different Block Size

To 1-D 4×4 column IDCT/IDST, the left part of Fig. 8 shows the Transform Core's and its internal 1-D IDCT Array Unit0's data flow. In this mode, the 1-D Transform Core is scheduled to deal with 4 column IDCT/IDST operations in parallel. According to the figure, four Pixel IDCT Elements of the 1-D IDCT Array Unit0 receive their first column data in sequence. Since there is a cycle delay between input data of the two adjacent Pixel IDCT Elements, these four Elements are scheduled to calculate their IDCT results in pipeline to avoid peak output data access at the same cycle.

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

After 4 cycles of the first input data, the Pixel IDCT Element0 generates its first pixel value $y_{0,0}$. After that, $y_{1,0}$, $y_{2,0}$ *and* $y_{3,0}$ are generated cycle by cycle. By using this scheduling scheme, each Pixel IDCT Element generates one transform result in four cycles and each 1-D IDCT Array Unit generates one transform result in every cycle. The port0 of Pixel data output port shows the final output data flow of the 1-D IDCT Array Unit0. Moreover, the IDCT mode and IDST mode have the same data flow but different coefficients of constant multiplication.

To 1-D 8×8 column IDCT, the 1-D Transform Core also deals with 4 column IDCTs in parallel. To keep a regular data flow, the Acc0_reg (the value of $y_{(N-r-1)}$) of the Butterfly Adder is controlled by the scheduler to reorder the sequence of the 1-D IDCT Array Unit's output data. As a result, the transform Core's data flow of 1-D 8×8 column IDCT is similar to the data flow of 1-D 4×4 Transform. The Pixel input port0～port3 deliver four column input pixel data into the Delay Array Unit0～Unit3 in parallel. Since the Pixel IDCT Element need 8 cycles to accumulate its output value, the output result is generated after 8 cycles.

To deal with 16×16 transform, the upper sub-array, the Delay Array Unit0 connected with the Delay Array Unit1, the 1-D IDCT Array Unit0 and the 1-D IDCT Array Unit1 are employed to conduct one column 16-point transform in pipeline. The bottom sub-array of the circuit is employed to conduct another column transform in pipeline. The data flow of 1-D 16×16 column IDCT is also shown in the right part of the figure. According to the figure, The Pixel input port0 and Pixel input port2 deliver two column 16-point input pixel data into the Delay Array Unit0 and the Delay Array Unit2 in parallel. 16 cycles later, the 1-D IDCT Array Units begin to output the results.

When the IDCT block is 32×32 block, only Pixel data input port0 is used to deliver residual data to all 1-D IDCT Array Units through the linear Delay Cell chain in order. In this mode, the 1-D Transform Core receives one input pixel data in each cycle and generates its output value after 32 cycles.

Ignoring switch time of different block size and the access conflict of the transpose buffer, the hardware utilization rate the 1-D Transform Core is nearly 100% by using the above adaptive block scheduling approach.

4 Experimental results and performance analysis

Using SMIC 65 nm 1P9M technology, the proposed HEVC 2-D IDCT/IDST architecture is synthesized. The maximum work frequency is about 480 MHz and the total power is 56.36 mw. The hardware overhead of 1-D Column Transform Core is 60.5 K NAND2 gates and the 1-D Row Transform Core is about 55.3 K NAND2 gates. Using 4 K×2 K People-OnStreet and Traffic video sequences recommended by HEVC, the proposed architecture is able to deal with real-time HEVC IDCT/IDST of 4096×2048@30 fps video sequence at 171 MHz in average.

Table I shows the hardware overhead comparisons of our proposed 1-D Transformation core with other 1-D HEVC DCT/IDCT architectures. Thereinto, reference [6] and [5]'s architectures are implemented with FPGA, and the gate counts are their LUT numbers.

To evaluate the hardware utilization, we defined hardware utilization

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

Table I. Comparisons of 1-D IDCT/DCT Architectures

Researchers			[8]	[6]	[4]	[5]	[7]	Ours
Algorithm			IDCT	DCT*	IDCT	IDCT	IDCT	IDCT/IDST
Block size		4×4 IDST						√
	DCT/IDCT	32×32	√				√	√
		16×16	√	√		√	√	√
		8×8	√		√			√
		4×4	√					√
Technology			0.13um	N/A	0.18um	N/A	0.18um	65nm
Input throughput (pixels/cycle)			4	16	1	16	16	4
Max Speed (MHz)			350	180	211.4	87.6	300	480
Transpose buffer bandwidth (pixels/cycle)			32	16	N/A	N/A	16	4
Gate counts			109.2K	5925	12.3K	5618	52.3K	60.5K
Hardware utilization			<20%	≈100%	≈12.5%	≈100%	N/A	≈100%

rate as:

$$\gamma = \frac{execution\_cycles \times occupied\_hardware\_resources}{(execution\_cycles + idle\_cycles) \times total\_hardware\_resources} \times 100\% \tag{5}$$

The equation (5) reflects the hardware utilization of the circuit. To a given computation task, if all modules and blocks of the circuit are occupied with no idle cycles, the hardware utilization rate will be 100%. There is no accurate hardware utilization result with the reference [7]. According to the paper, its hardware utilization rate maybe less than 80%. Since the reference [8]'s architecture is based on 32-point IDCT butterfly computation structure, most blocks of it will be idle during the variable block size IDCT process. According to previous statistical results, the percentage of 16×16 block size used for IDCT is 9.1%, 32×32 block size used for IDCT almost 1.3%. As a result, blocks of large hardware overhead, such as *Mutlit3, R16 and B32,* are unoccupied in most of time. Moreover, when the block size is 32×32, 16×16 or 8×8 pixel block, the butterfly architecture will keep in idle state for 7, 6 and 4 cycles in each 8 cycles respectively. According our evaluation, its hardware utilization is lower than 20%. According to the table, the hardware utilization rates of reference [6], reference [5] and reference [7] are much higher than others. However, the performance of these parallel architectures with huge throughput far exceed their practical requirements. For example, the 16×16 IDCT only occupies nearly 10% of IDCT block type. According to the input throughput of these architectures, the work frequency with 4096×2048@30 fps realtime application is as low as 4.19 MHz.

Compared with [8], our architecture can save about 45% hardware overhead and only need 1/8 transpose memory peak bandwidth. Compared with [7], our architecture only need 25% input access bandwidth to deal with all kinds of block size and block mode of IDCT/IDST with higher hardware utilization rate while only increasing 15.7% hardware cost. In summary, our proposed 1-D Transform Core is the first HEVC IDCT/IDST

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013

architecture compatible with all transform mode. The architecture achieves the minimum input access bandwidth and transpose memory bandwidth while keeping nearly 100% hardware utilization.

5 Conclusion

HEVC is the developing video coding standard over H.264 to fulfill the demand for future ultra-high resolution videos. In this paper, we proposed a novel full-pipelined 2-D IDCT/IDST VLSI architecture with block-size adaptive for the developing HEVC standard. In summary, our proposed architecture is the first adaptive IDCT/IDST architecture supporting transform calculation with variable block size from 4×4 to 32×32 pixels and IDCT/IDST mode architecture compatible with HEVC standard fully. In consequence, it offers a cost-effective solution for the future UHDTV applications.

Acknowledgments

This work was supported by the Hi-Tech Research and Development Program (863) of China under Grant No. 2012AA012701.

© IEICE 2013
DOI: 10.1587/elex.10.20130210
Received March 29, 2013
Accepted April 09, 2013
Published May 10, 2013