A Parallel-Access Mapping Method for the Data Exchange Buffers Around DCT/IDCT in HEVC Encoders Based on Single-Port SRAMs

Yibo Fan, Leilei Huang, Yufeng Bai, and Xiaoyang Zeng, *Member, IEEE*

***Abstract*—In the High Efficiency Video Coding (HEVC) standard, a notation of the transform unit (TU) is introduced with four different sizes, i.e., 4 × 4, 8 × 8, 16 × 16, and 32 × 32, which results in at least two problems in the use of discrete cosine transform/inversed discrete cosine transform (DCT/IDCT). One is changeable input/output format presented by DCT/IDCT when it deals with TUs of different sizes, which intensifies the nonconformity during the data exchange with other modules. The other is the demand for high throughput to traverse the vast possible TU partitions to find the best one, which would be easily dragged by an inefficient data exchange method. To solve this problem, a parallel-access data mapping method based on single-port static random access memory devices (SRAMs) is proposed in this brief. It can be applied to the data exchange buffers around DCT/IDCT in HEVC encoders to fulfill a high-throughput data exchange. Here, parallel access means one row of 1 × 32 pixels, two rows of 1 × 16 pixels, four rows of 1 × 8 pixels, or four rows of 1 × 4 pixels could be accessed in one cycle depending on the specific size of the current TU.**

***Index Terms*—Buffer, data mapping method, discrete cosine transform (DCT), High Efficiency Video Coding (HEVC), inversed discrete cosine transform (IDCT), single-port static random access memory.**

I. Introduction

During the implementation of video coding with hardware, data exchange between discrete cosine transform/inversed discrete cosine transform (DCT/IDCT) and other modules is an annoying problem. The input/output (I/O) format of DCT/IDCT is presented in rows or columns and varies with the matrix size. This format originates from the realization mechanism adopted by most DCT/IDCT designs, which replaces a 2-D one with two 1-D transforms, namely, the row–column decomposition method (RCDM). However, due to the algorithm set by the encoding standard or out of convenience in real implementation, the I/O formats of other modules are usually in blocks, leading to a nonconformity in the data exchange between DCT/IDCT and other modules. This kind of nonconformity exists in every related data path, which would easily affect efficiency.

In High Efficiency Video Coding (HEVC) encoders, this situation becomes even worse due to the introduction of the notation of the transform unit (TU) with four different sizes, i.e., 4 × 4, 8 × 8, 16 × 16, and 32 × 32. On one hand, the I/O format can be changed because of the different TU sizes. On the other hand, the throughput demand is higher because of the traverse process. To solve the latter problem, some high-throughput DCT/IDCT designs are proposed; Meher *et al.* [1] proposed a fixed-throughput one. This design processes pixels at a fixed rate of 32 pixels/cycle, irrespective of TU sizes, which is very suitable for real-time encoding toward high-definition (HD) videos. However, without an efficient data exchange method, this design will be impractical to use.

Thus, in this brief, a parallel-access mapping method based on single-port static random access memory devices (SRAMs) is proposed to provide a considerable throughput for the data exchange around DCT/IDCT in HEVC encoders with a low-level cost in hardware resources. Here, parallel access means that this mapping method could provide one row of 1 × 32 pixels, two rows of 1 × 16 pixels, four rows of 1 × 8 pixels, or four rows of 1 × 4 pixels in one cycle, which can perfectly cooperate with the work of Meher *et al.* [1].

The rest of Section I would give detailed background about DCT/IDCT in HEVC and the data exchange around it, followed by motivations and challenges. In Section II, some naive methods will be introduced for comparison. In Section III, the proposed mapping method will be presented with figures, examples, and some necessary explanations. Comparisons to other solutions would be provided in Section IV with solid experimental data, as well as some illustrations. Finally, the conclusion is given in Section V.

A. DCT/IDCT in HEVC

According to the HEVC standard, the TU size can vary from 4, 8, 16, to 32, as shown in the left part of Fig. 1(a), which denotes these TUs by TU_04, TU_08, TU_16, and TU_32, respectively. This change introduces at least two problems that do not exist in the implementation of H.264 encoders. One problem is the I/O format, and the other is throughput.

The I/O format problem is that the DCT/IDCT module in HEVC would process pixels in different orders according to different TU sizes. For example, during the process of TU_04, DCT/IDCT would naturally process data in rows of 1 × 4 pixels (or columns of 4 × 1 pixels). However, when the object is TU_08, TU_16, or TU_32, the width of rows (or the length of

Manuscript received May 22, 2015; accepted August 9, 2015. Date of publication August 14, 2015; date of current version November 25, 2015. This work was supported in part by the National Natural Science Foundation of China under Grant 61306023 and in part by the State Key Laboratory of ASIC & System under Grant 2015MS006. This brief was recommended by Associate Editor J. P. de Gyvez.

The authors are with State Key Laboratory of ASIC & System, Fudan University, Shanghai 200433, China (e-mail: fanyibo@fudan.edu.cn; xyzeng@fudan.edu.cn).

Digital Object Identifier 10.1109/TCSII.2015.2468915

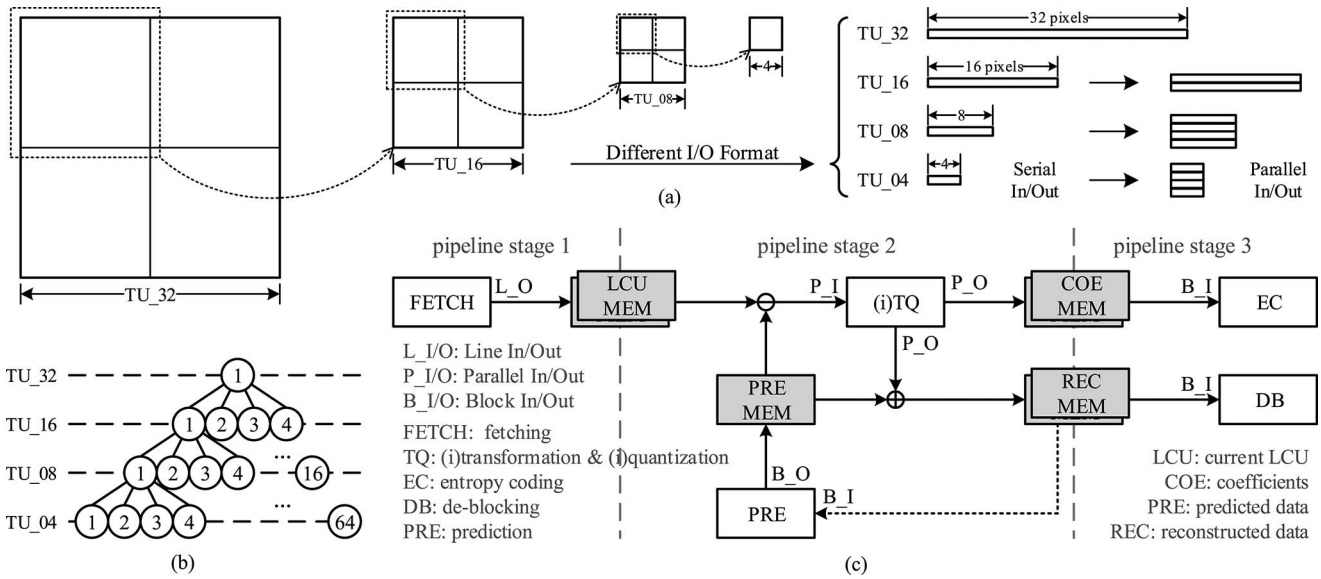

Fig. 1. (a) TU partitions in HEVC and the corresponding I/O format of DCT/IDCT. (b) Traverse toward possible TUs. (c) General hardware architecture for HEVC encoder. The application objects of the proposed mapping method are marked with gray color.

columns) of the I/O format would increase with the TU size, as described by the "Serial In/Out" part of Fig. 1(a). This format origins from the commonly used implementation method for DCT/IDCT, namely, RCDM, which is favored by many state-of-the-art designs [2]–[4]. By RCDM, 2-D DCT/IDCT is decomposed into two 1-D DCT/IDCTs; thus, pixels are naturally processed in rows or columns instead of blocks.

The throughput problem is that the DCT/IDCT module in HEVC encoders is heavily burdened, and therefore, its throughput has to be high enough to encode HD videos in real time. To be more specific, in order to find the best TU partition, DCT/IDCT is required to traverse the TU tree, as shown in Fig. 1(b); thus, 4 times of 32×32 transform, 16 times of 16×16 transform, 64 times of 8×8 transform, and 256 times of 4×4 transform would be executed during the transform process toward one large coding unit (LCU), which has a size of 64×64 (here, if the prediction mode is also determined by the results of DCT/IDCT, the execution time will be further increased). Of course, some fast algorithm can be adopted to narrow down the range of traversal. However, according to these state-of-the-art techniques [5]–[7], the remaining range is still wide, leading to a still huge amount of calculation, which gives rise to many high-throughput DCT designs [8]–[10].

B. Data Exchange Around DCT/IDCT

As mentioned earlier, due to the changeable I/O format of the DCT/IDCT module, data exchange around it is becoming an annoying problem in HEVC. A general architecture for HEVC encoders is presented in Fig. 1(c). It can be seen from this figure that DCT/IDCT (TQ) module could be taken as the core module of HEVC encoders in some sense because it has data exchange with all of the other modules, including pixel fetching, entropy coding, deblocking, and predicting modules, which are denoted by FETCH, EC, DB, and PRE in this brief.

For DCT/IDCT, the I/O format is already a tricky problem, as introduced earlier. However, in order to further raise the throughput, a fixed-throughput design is proposed by Meher *et al.* [1], which makes it even trickier. To be more specific, Meher *et al.*'s design could afford a throughput of 32 pixels/cycle, irrespective of TU sizes. For example, one row of 1×32 pixels per cycle for TU_32, two rows of 1×16 pixels per cycle for TU_16, or four rows of 1×8 pixels for TU_08, which are denoted by P_I/O in this brief and described by the "Parallel In/Out" in Fig. 1(a).

For other module, the I/O format is different from that of DCT/IDCT. Here, EC, namely CABAC, is set as an example; it is natural for the input data of CABAC to be provided in the format of 4×4 blocks because the basic processing unit of CABAC is one 4×4 block, according to the algorithm specified by the HEVC standard. In a similar way, PRE and DB also adopt this kind of I/O format, which is denoted by B_I/O. While for FETCH, in order to fully utilize the burst access characteristic of external memory controller and bus system, pixels are usually fetched in lines, which is denoted by L_O in this brief.

With the help of these abbreviations, it can be clearly shown in Fig. 1(c) that the format nonconformity exists in every related data path; thus, unavoidable buffers are needed between DCT/IDCT and other modules to buffer original, predicted, and reconstructed pixels and coefficients, which are denoted by CUR, PRE, REC, and COE in this brief. These buffers would be the application objects of the proposed mapping method.

C. Motivations and Challenges

Both the motivations and the challenges to propose such a mapping method can be simply concluded into two aspects: area and throughput.

When the area is considered first, it is worth noticing that the data amount to be buffered is quite huge. As shown in Fig. 1(c), a total of seven LCUs need to be buffered according to the arrangement of pipeline stages (in fact, due to the traverse operation to find the best TU partition, a total of three LCUs would be needed to store the current data, which is the best data until now and the data for next pipeline stage). For such a high amount of data, it is meaningful to save as more area as possible.

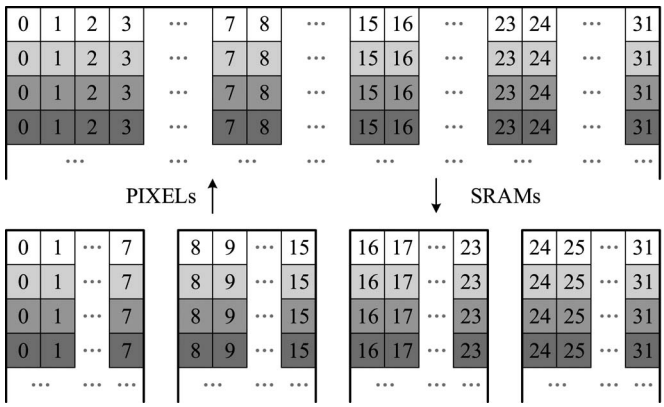

Fig. 2. Area-oriented straightforward mapping method.

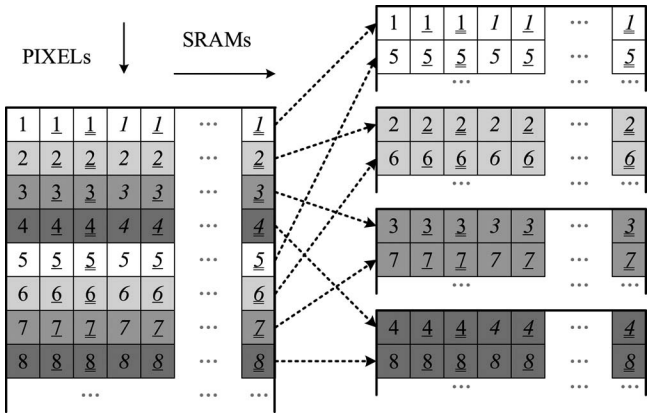

Fig. 3. Multiple memory devices. Here, numbers stands for the row number.

When the throughput is considered first, it is worth noticing that the throughput of individual module is already high enough, such as PRE designs [11]–[13], EC designs [14]–[16], DB designs [17]–[19], and the aforementioned DCT/IDCT designs. However, without an efficient data exchange method, those high-throughput designs may be wasted. Although, in the H.264 era, even a software method [20] would be enough to fulfill the need for a highly efficient encoder.

Thus, it is beneficial to find a solution that could balance both aspects, but such a solution is difficult to find. To be more specific, the main challenge is how to accomplish the same high throughput with different formats of P_I/O, B_I/O, and L_I/O without using area-cost registers, multiport memory devices, or something similar. Of course, the clock frequency and power consumption should also be taken into considerations.

II. Straightforward Solutions

A. Area Oriented

If area is given with a higher priority, SRAM would be a straightforward choice to save hardware cost. A simple solution is presented in Fig. 2, which puts pixels directly according to its physical position. Since the maximum size of TU is 32×32, it would be natural to take 32×32 blocks, namely QLCU, as the individual mapping object. Here, a total of four single-port SRAMs are adopted to store pixels data to accomplish the "Serial In/Out" throughput described in Fig. 1(a), which means four cycles are needed to access one TU_04. Of course, one SRAM with four times the data width could afford the same throughput, but in actual situation, such a wide data width is not supported by general SRAM compilers. Unnecessary pixels could be kept unread if four SRAMs are adopted, which would save some power.

B. Throughput Oriented

If throughput is given with higher priority, these solutions can be figured out, including registers, multiple memory devices, or multiport memory.

Of course, if these data are stored by registers, the throughput can be as high as possible, but the hardware cost is unbearable.

As to multiple SRAMs, it means using several memory devices to store the data belonging to one access separately. As shown in Fig. 3, every four lines of pixels are distributed into the same SRAM, and the neighboring lines are distributed into different SRAMs. Thus, this mapping method could fulfill the "Parallel In/Out" throughput described in Fig. 1(a). However, the depth of these SRAMs would be so small that it is meaningless to use SRAMs, considering the area cost of using a shallow-depth SRAM will be no much better than that of using registers.

Multiport memory solves the parallel problem in a similar way as using multiple memory devices. Thus, four pairs of READ and WRITE ports are needed to access one 4×4 block in one cycle, which is not supported by commonly used memory compilers.

III. Proposed Method

As shown in Fig. 4, the proposed data mapping method is based on four single-port SRAMs whose data width is eight pixels. As mentioned earlier, it would be natural to take QLCU as the individual mapping object. One QLCU could be divided into 16 8×8 blocks, which are marked with $0, 1, \ldots, f$, and one 8×8 block can be further divided into eight rows of 1×8 pixels, which are marked with nBlock-0, nBlock-1, …, nBlock-7. Here, nBlock stands for the corresponding 8×8 block that this row belongs to. For example, row 3-7 refers to the seventh 1×8 row belonging to the third 8×8 block of a certain QLCU, whereas row 6-4 refers to the fourth 1×8 row belonging to the sixth 8×8 blocks of a certain QLCU.

The proposed mapping method is explicitly expressed with the above numbering scheme. For example, row 0-0 is stored in address 0 of the zeroth SRAM, row 1-0 in address 0 of the second SRAM, row 2-0 in address 0 of the first SRAM, and row 3-0 is stored in address 0 of the third SRAM. Similarly, row 0-1 is stored in address 1 of the first SRAM, row 1-1 in address 1 of the third SRAM, row 2-1 in address 1 of the second SRAM, and row 3-1 in address 1 of the zeroth SRAM. The rest are done in the same manner, which could be concluded into the following pseudocode:

$$
\begin{aligned}
&\text{switch } (\text{nRow}\%4)\\
&\quad \text{case } 0: \quad \text{bank} = (\text{nBlock} + 0)\%4\\
&\quad \text{case } 1: \quad \text{bank} = (\text{nBlock} + 2)\%4\\
&\quad \text{case } 2: \quad \text{bank} = (\text{nBlock} + 1)\%4\\
&\quad \text{case } 3: \quad \text{bank} = (\text{nBlock} + 3)\%4\\
&\text{end}\\
&\text{addr} = 32 \times \text{nQLCU} + 8 \times \text{floor}(\text{nRow}/4) + \text{nBlock} \qquad (1)
\end{aligned}
$$

where nRow is the relative row number in the 8×8 block that the current 1×8 row belongs to, nBlock is the relative block

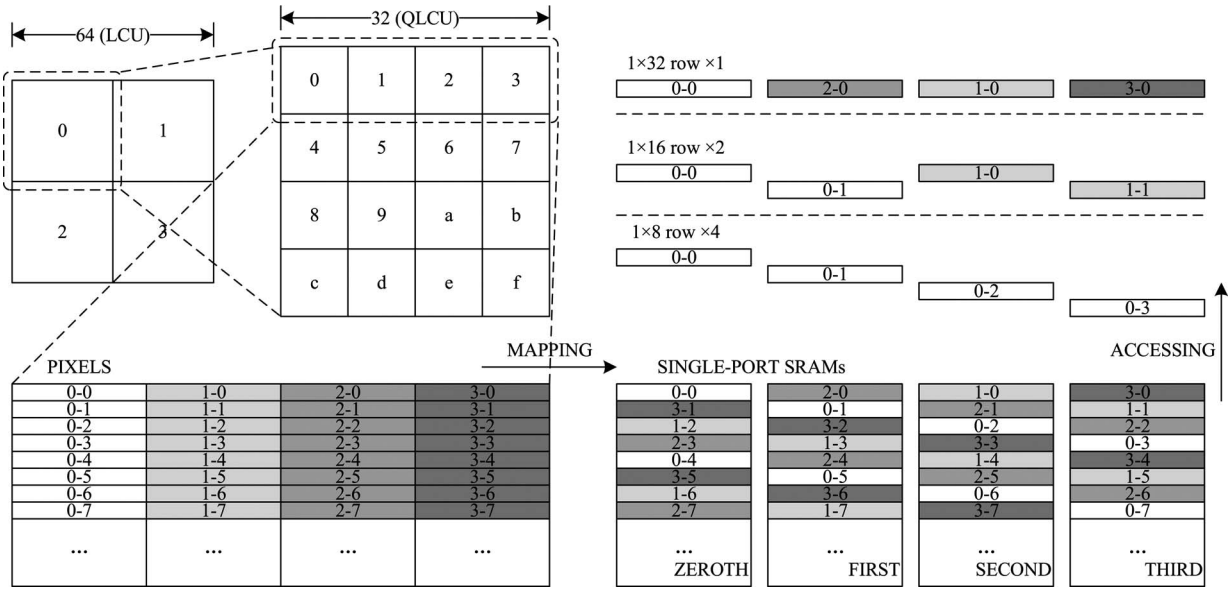

Fig. 4. Proposed mapping method.

number in the QLCU that the current 8×8 block belongs to, nQLCU is the number of QLCU, and bank and addr are the number and address to be access in SRAMs. Row 2-5 is set as an example. If it belongs to the zeroth QCLU, then this row would be allocated in address $32 \times 0 + 8 \times \text{floor}(2/4) + 5$, namely address 5, of SRAM $(5+1)\%4$, i.e., the second SRAM. According to (1), parallel access can be easily fulfilled based on single-port SRAMs.

It is easy to find out that all of the 1×8 rows belonging to one 1×32 row are always mapped to the same address of different SRAMs; therefore, there would be no collision during the access of 1×32, 1×16, 1×8, or 1×4 rows. During the access to a 1×16 row, only half of the throughput is occupied. In fact, the rest of the throughput could be filled by the next 1×16 row, such as rows 0-0, 0-1, 1-0, and 1-1, which have no access collision as shown in Fig. 4. Similarly, any four neighboring 1×8 rows can also be accessed in one cycle, such as rows 0-0, 0-1, 0-2, and 0-3, as also shown in Fig. 4.

Thus, no matter the P_I/O format, the B_I/O format of 4×4 blocks, or the L_I/O format of 1×32 rows, this mapping method could always provide in one cycle. Moreover, the throughput of the proposed method in fact could be expanded to 64 pixels. In such situation, eight single-port SRAMs or four two-port SRAMs will be needed. However, a throughput of 32 pixels/cycle will be enough for state-of-the-art designs such as that of Meher *et al.* [1], which balanced the speed and hardware cost quite well.

IV. Comparison

In this section, detailed comparisons to other solutions would be given with solid data and some explanations, followed by a conclusive results of the performances in area or throughput.

A. With Registers

A total of $64 \times 64 \times 8$-bit registers would be needed to buffer all of the predicted data of the whole LCU. According to the driving capacities, the hardware cost would be slightly different, as shown in Table I.

TABLE I
Hardware Cost of Registers

Cell Name	Gate Count	Total Gate Count
DFCNQD1	7	229K
DFCNQD2	8.5	279K
DFCNQD4	9.5	311K

These cells can be found in the TSMC general-purpose synthesis library (*tcbn65gplus*). Gate count shows the count of one single cell, while total gate count means the count of 32768 cells.

TABLE II
Hardware Cost of Multiple Memory Devices

Data Depth	Data Width	Gate Count	Total Gate Count
32	256	does not exist	/
32	128	12682	101K

These register files are complied with the high-speed single-port register file compiler (*rf_sp_hdd_svt_rvt_hvt*) from ARM. Gate count shows the count of one single register file, while total gate count means the total count of the needed register files.

B. With Multiple Memory

If the method of using multiple memory devices is adopted, four separate SRAMs are needed, which have a data width of 32×8 bits and a depth of 32 theoretically. However, in practice, the data width of 256 bits usually exceed the capacity of normal memory compilers; thus, one 256-bit register file may need to be realized by two separate 128-bit register files, which makes the final gate count larger than expected, as shown in Table II.

C. The Straightforward and Proposed Method

Both the straightforward method introduced in Section II-A and the proposed one need four separate SRAMs, which is similar to the method of using multiple memory devices, but these SRAMs would have a data width of 8×8 bit and a depth of 128 theoretically. In practice, such parameters can usually be satisfied, as shown in Table III.

TABLE III
HARDWARE COST OF THE PROPOSED METHOD

Data Depth	Data Width	Gate Count	Total Gate Count
128	64	10594	42.4K

This register file is complied with the same compiler listed in Table II. Gate count shows the count of one single register file, while total gate count means the total count of the needed register files.

TABLE IV
COMPARISONS (SAME AREA)

Method	Throughput	Power	Freq
straightforward method	Serial I/O	16.6mW	600M
proposed method	P_I/P_O, B_I/B_O L_I/O	18.8mW	600M

TABLE V
COMPARISONS (SAME THROUGHPUT)

Method	Gate Cnt	Saved	Wrapper	Power	Freq
registers	229K	81.66%	without	/	600M
multiple memories	101K	58.42%	without	/	600M
proposed method	42.4K	/	without	/	600M
registers	330K	86.21%	with	111mW	600M
multiple memories	102K	55.4%	with	73.4mW	600M
proposed method	45.5K	/	with	18.8mW	600M

D. Comparison

Detailed comparisons are shown in Tables IV and V. Designs in the former one have the same area, whereas designs in the latter one have the same throughput. These comparisons show that, if a same throughput is achieved, the proposed method could save 82% of the gate count when compared with using registers or 58% when compared with using multiple memory devices. If the same area cost is occupied, the proposed one can provide "Parallel In/Out" throughput, whereas the straightforward one could only fulfill "Serial In/Out" throughput. The supported working frequency and power assumption is also analyzed, obtained by DC and PP tools, respectively.

To conclude, the proposed design succeeds in providing a considerable throughput for the DCT/IDCT in HEVC encoders with a low-level cost in hardware resources.

V. CONCLUSION

In HEVC standard, a notation of TU is introduced with four different sizes, resulting in at least two problems in the use of DCT/IDCT. One is the changeable I/O format presented by DCT/IDCT when dealing with TUs of different sizes, which intensifies the nonconformity during the data exchange with other modules. The other is the demand for high throughput to traverse the vast possible TU partitions to find the best one, which would be easily dragged by an inefficient data exchange method. To solve this problem, a parallel-access data mapping method based on single-port SRAMs has been proposed in this brief. It could be applied to the data exchange buffers around DCT/IDCT in HEVC encoders to fulfill a high-throughput data exchange. Under TSMC 65-nm technology, experimental results shows that it occupies a gate count of 42.7 K only, which saves 82% of the gate count when compared with using registers or saves 58% when compared with multiple memories.

REFERENCES

[1] P. K. Meher, S. Y. Park, B. K. Mohanty, K. S. Lim, and C. Yeo, "Efficient integer DCT architectures for HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 1, pp. 168–178, Jan. 2014.

[2] H. Sun, D. Zhou, J. Zhu, S. Kimura, and S. Goto, "An area-efficient 4/8/16/32-point inverse DCT architecture for UHDTV HEVC decoder," in *Proc. IEEE Vis. Commun. Image Process. Conf.*, Dec. 7–10, 2014, pp. 197–200.

[3] H. Liang, H. Weifeng, Z. Hui, and M. Zhigang, "A cost effective 2-D adaptive block size IDCT architecture for HEVC standard," *Proc. IEEE 56th Int. MWSCAS*, Aug. 4–7, 2013, pp. 1290–1293.

[4] M. Martuza and K. Wahid, "A cost effective implementation of 8 × 8 transform of HEVC from H.264/AVC," in *Proc. IEEE CCECE*, Apr. 29–May 2, 2012, pp. 1–4.

[5] G. Tian and S. Goto, "Content adaptive prediction unit size decision algorithm for HEVC intra coding," in *Proc. PCS*, May 7–9, 2012, pp. 405–408.

[6] J. Xiong and H. Li, "Fast and efficient prediction unit size selection for HEVC intra prediction," in *Proc. ISPACS*, Nov. 4–7, 2012, pp. 366–369.

[7] T. Nishikori, T. Nakamura, T. Yoshitome, and K. Mishiba, "A fast CU decision using image variance in HEVC intra coding," in *Proc. IEEE ISIEA*, Sep. 22–25, 2013, pp. 52–56.

[8] E. Kalali, E. Ozcan, O. Yalcinkaya, and I. Hamzaoglu, "A low energy HEVC inverse transform hardware," *IEEE Trans. Consum. Electron.*, vol. 60, no. 4, pp. 754–761, Nov. 2014.

[9] J. Zhu, Z. Liu, and D. Wang, "Fully pipelined DCT/IDCT/Hadamard unified transform architecture for HEVC codec," in *Proc. IEEE ISCAS*, May 19–23, 2013, pp. 677–680.

[10] R. Jeske *et al.*, "Low cost and high throughput multiplierless design of a 16 point 1-D DCT of the new HEVC video coding standard," in *Proc. VII SPL*, Mar. 20–23, 2012, pp. 1–6.

[11] Z. Liu, D. Wang, H. Zhu, and X. Huang, "41.7BN-pixels/s reconfigurable intra prediction architecture for HEVC 2560 × 1600 encoder," in *Proc. IEEE ICASSP*, May 26–31, 2013, pp. 2634–2638.

[12] N. Zhou, D. Ding, and L. Yu, "On hardware architecture and processing order of HEVC intra prediction module," in *Proc. PCS*, Dec. 8–11, 2013, pp. 101–104.

[13] M. Kammoun, A. Ben Atitallah, and N. Masmoudi, "An optimized hardware architecture for intra prediction for HEVC," in *Proc. 1st IPAS*, Nov. 5–7, 2014, pp. 1–5.

[14] D. Zhou *et al.*, "A 4320p 60 fps H.264/AVC intra-frame encoder chip with 1.41 Gbins/s CABAC," in *Proc. Symp. VLSI Circuits*, Jun. 13–15, 2012, pp. 154–155.

[15] P. Jayakrishnan, P. V. A. Lincy, and R. M. Niyas, "A high speed real time multi-bin CABAC encoder for ultra high resolution video," in *Proc. ICE-CCN*, Mar. 25–26, 2013, pp. 207–210.

[16] D. Zhou, J. Zhou, W. Fei, and S. Goto, "Ultra-high-throughput VLSI architecture of H.265/HEVC CABAC encoder for UHDTV applications," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 3, pp. 497–507, Mar. 2015.

[17] M. Mody, N. Nandan, and T. Hideo, "High throughput VLSI architecture supporting HEVC loop filter for ultra HDTV," in *Proc. IEEE 3rd ICCE-Berlin*, Sep. 9–11, 2013, pp. 54–57.

[18] C. M. Diniz, M. Shafique, F. V. Dalcin, S. Bampi, and J. Henkel, "A deblocking filter hardware architecture for the high efficiency video coding standard," in *Proc. DATE*, Mar. 9–13, 2015, pp. 1509–1514.

[19] W. Shen, Q. Shang, S. Shen, Y. Fan, and X. Zeng, "A high-throughput VLSI architecture for deblocking filter in HEVC," in *Proc. IEEE ISCAS*, May 19–23, 2013, pp. 673–676.

[20] H. Javaid, M. Shafique, S. Parameswaran, and J. Henkel, "Low-power adaptive pipelined MPSoCs for multimedia: An H.264 video encoder case study," in *Proc. 48th ACM/EDAC/IEEE DAC*, Jun. 5–9, 2011, pp. 1032–1037.