A Hardware-Oriented IME Algorithm for HEVC and Its Hardware Implementation

Yibo Fan, Leilei Huang, Bei Hao, and Xiaoyang Zeng, *Member, IEEE*

***Abstract*— High Efficiency Video Coding (HEVC), the latest video coding standard, aims to provide coding performance that is much superior to that of its predecessor, H.264, especially for high definition video. To fulfill this goal, the inter-prediction unit (PU) partitions of HEVC are more complex, and the search range of motion estimation (ME) is much larger. As a result, ME becomes a bottleneck in the design of the HEVC inter predictor. In response to this challenge, we developed a hardware-oriented integer ME algorithm and the related hardware implementation. Our proposed algorithm led to a decrease in terms of the Bjontegaard Delta rate when compared with the HEVC test model 15.0. The corresponding hardware solution benefitted from 2-D data reuse supported by horizontal and vertical reference SRAMs, on-chip memory reduction supported by 4 × 4 block compression, and a low-power sum of absolute difference (SAD) tree supported by PU-level chip selection. When adopting a 32 × 32 SAD tree, the minimum and maximum required working frequency for 4K × 2K at 30 frames/s videos was [375, 500] MHz. These results demonstrated that our proposed solution offered desirable improvement in both coding speed and coding performance.**

***Index Terms*— 2-D data reuse, coding performance, coding speed, data compression, hardware implementation, HEVC, high efficiency video coding, IME, integer motion estimation, inter prediction, SAD tree.**

I. Introduction

HIGH Efficiency Video Coding (HEVC) is the latest video coding standard. HEVC aims to provide a much superior coding performance compared to that of its predecessor H.264, especially for high definition (HD) video. Several new concepts have been introduced in HEVC to improve coding performance, including notation of the coding unit (CU), prediction unit (PU), and transform unit (TU). However, the future design and implementation of HEVC faces challenges. For this paper, we examined difficulties concerning motion estimation (ME) and offer a proposed solution.

The Coding Tree Unit (CTU), which can be set to 64 × 64, is the root node of the CU quad-tree, while the CU is the root node of both the PU and TU quad-trees. In inter prediction, the PU can vary from 4 × 4 to 64 × 64, including the newly introduced asymmetric motion partitions (AMP). As shown in Fig. 1, the 64 × 64 PUs can split into 64 × 32, 32 × 64, 64 × 16, 64 × 48, 16 × 64, and 48 × 64. The 32 × 32 PUs can split further into 32 × 16, 16 × 32, 32 × 08, 32 × 24, 08 × 32, and 24 × 32. The 16 × 16 PUs can split further into 16 × 08, 08 × 16, 16 × 04, 16 × 12, 04 × 16, and 12 × 16.

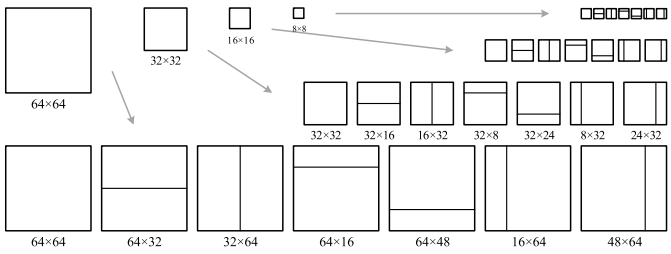


Fig. 1. Inter prediction partitions in HEVC.

In addition to the numerous combinations of possible partitions, the motion estimation (ME) search range in HEVC also tends to be larger than that in H.264. For example, the default search range of HEVC test model (HM) 15.0 is [−64, 64], and the search range used in the experimental part of related papers [1]–[3] was usually set to [-64, 64] as well. As a result, the implementation of ME has been challenging for the design of HEVC encoders.

Several papers have proposed highly productive ideas, architectures, or implementations on this topic for both software and hardware. In the software area, Li *et al.* [4] proposed a context-adaptive fast ME technique that chose the appropriate search method according to the motion intensity measured. Van *et al.* [5] proposed a fast ME algorithm for the closed-loop HEVC trans-rating that changed the start point and search range according to the input and output motion vectors (MVs). Nguyen *et al.* [6] proposed asymmetric diamond search patterns for ME that adopted the wide diamond pattern or the rot-w-diamond pattern to reduce the encoding time.

In terms of hardware, Nalluri *et al.* [7] proposed a high-speed sum of absolute difference (SAD) tree for variable block size that supported blocks from 4 × 4 to 64 × 64, including the AMP blocks. Medhat *et al.* [8] proposed another implementation for ME that could process 1920 × 1080@30fps with a search range of [−27, + 27]. Ye *et al.* [9] presented a hardware-oriented IME implementation that shared the search process among different PUs according to the clustering feature of MVs.

In view of the challenges to HEVC development described above and in light of the preceding research, we proposed a hardware-oriented integer ME (IME) algorithm and

Manuscript received October 11, 2016; revised February 27, 2017; accepted April 25, 2017. Date of publication May 8, 2017; date of current version August 3, 2018. This work was supported in part by the National Natural Science Foundation of China under Grant 61674041, in part by the STCSM under Grant 16XD1400300, and in part by the State Key Laboratory of ASIC and System under Grant 2015MS006. This paper was recommended by Associate Editor S. Shirani. *(Corresponding authors: Yibo Fan; Xiaoyang Zeng.)*

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

Digital Object Identifier 10.1109/TCSVT.2017.2702194

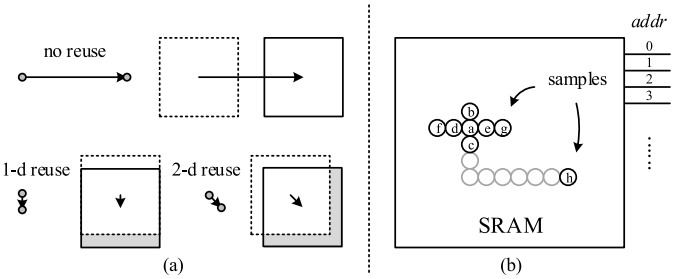


Fig. 2. (a) Data reuse level. (b) SRAM access limitation.

its hardware implementation. The remainder of this paper explains our procedures and findings. Section II presents the background, motivations, and contributions of this paper. Section III introduces the proposed IME algorithm, while Section IV gives the corresponding hardware implementation. Comparisons are provided in Section V. Finally, Section VI concludes this paper.

II. Background, Motivations & Contributions

In this section, we provide the notation used in this paper for the data reuse level and cost reuse level, followed by the motivations and contributions of this paper.

A. Background

1) Data Reuse Level: For hardware implementations, reference and current pixels are usually stored in two register arrays that are connected to a SAD tree to calculate the cost, as shown in Fig. 4(a). Three possible data reuse levels could be adopted during the update of the reference register array: "no reuse," "1-D reuse," and "2-D reuse," as shown in Fig. 2(a).

"No reuse" signifies that there is no data reuse between two search points. Generally, this level is adopted by software, with the result that there is an almost linear relationship between the time cost of search and the number of search points for software.

"1-D reuse" means that data between two vertical (horizontal) adjacent points can be reused efficiently, if reference pixels are located horizontally (vertically) in the reference SRAM. This level is adopted by almost all hardware designs.

"2-D reuse" signifies that data between any two adjacent points can be reused efficiently, which cannot be adopted in general structure due to the access limitation of SRAMs. This limitation will be analyzed in Section IV B.

2) Cost Reuse Level: For software algorithms, SAD costs of different PUs usually are tested in separate searches. For example, in HM, 64 $\times$ 64 PUs and their splits are searched first, then the four 32 $\times$ 32 PUs within it and their splits are searched. This process is executed recursively until the tests towards 8 $\times$ 8 PUs and their splits are finished, unless some early termination conditions are met.

For hardware implementations, thanks to the adoption of SAD trees, SAD costs can be reused. In other words, costs of all PU levels for the same MV are gathered in the same search. However, this approach could introduce some unnecessary power consumption if only one specific PU (and its splits) needs to be tested.

B. Motivations

In the software area, several fast ME algorithms have been proposed to reduce the total search points while keeping the same coding performance. However, it is still unlikely for them to finish the search task for HD applications in real time. Nevertheless, the contributions of these works are remarkable, such as the diamond search [11], the hexagonal search, and the square search, all of which would be of great assistance if they could be imported efficiently to hardware designs.

In the hardware area, several papers proposed hardware-friendly search algorithms [12]–[14]. However, these methods are inefficient because their data reuse levels are not satisfying. As a result, full search or block-based search and general SAD trees are still favored, which will unavoidably introduce much unnecessary power consumption.

Given the above state of affairs, it is quite an attractive challenge to work on the development of algorithms suitable for implementation on hardware platforms. Accordingly, for this paper we proposed a hardware-oriented IME algorithm and succeeded in providing a feasible implementation supporting real-time encoding.

C. Contributions

The contributions of this paper are given as follows:

1) We analyzed the limiting factors when implementing IME on a hardware platform.
2) We proposed a hardware-oriented algorithm based on the above analysis which achieved a decrease in BD rate.
3) We developed the corresponding hardware implementation that could support 4 K $\times$ 2 K@30fps video coding.

III. The Hardware-Oriented IME Algorithm

A. Limiting Factors

Three main limiting factors could exist during the hardware implementation of IME.

1) Search Pattern is Limited: During hardware implementations, it is important to pay great attention to the data reuse level because it has a deep influence on efficiency and power consumption.

For example, Medhat *et al.* [14] proposed a search pattern called FCSA that searches "from the center point at every row of search window, then, it takes, consecutively, one candidate left to the center point and one candidate to the right till it reaches search window's edges" [14]. This kind of search pattern may be efficient for software, but it is not efficient for hardware. The reason is quite obvious: the farther the search point is from the center, the less the data can be reused. As a result, full search or block-based search is still favored, but these patterns lack flexibility and efficiency.

2) Starting Point is Limited: In the HM, the IME is traversed from the largest CU (LCU) to the smallest CU (SCU) recursively, as mentioned previously. For this reason, the starting point for each PU could benefit from MV predictions (MVPs).

However, for hardware solutions, the search process can be paralleled between PUs, thanks to the adoption of SAD trees.

This adoption, in fact, is also an obvious necessity to fulfill real time encoding. Assuming that 8000 cycles (corresponding to 4K×2K @30fps @500MHz) is allowed for the IME towards one coding tree unit (CTU), if each PU is searched individually, the average cycle margin allowed for each would be only $8000 \div (1 + 4 + 16 + 64) \div 7 = 13$. It is almost impossible to find a good enough result in a search range of $[-64, 64]$ with such a small cycle margin. As a result, the starting point for each PU cannot be as flexible as it is in software.

3) Real Time Problem: Assuming that the search range is $[-64, 64]$, then $129 \times 129 = 16641$ search points must be tested for every PU partition. Considering the hardware cost of register arrays and SAD calculation logic, a 64×64 SAD tree would be too costly. Instead, we adopted a 32×32 SAD tree for the purposes of this paper. Even by taking full advantage of a SAD tree by utilizing parallel searching, the calculations would still take at least $16641 \times 4 = 66564$ cycles.

Of course, the search range could be reduced to keep the cycle cost within a reasonable range, such as $[-23, 23]$, which is $47 \times 47 \times 4 = 8836$ cycles. However, this approach would lead to a relatively high BD rate increase for videos with highly dynamic motions. For example, the BD rate of the sequence "BasketballDrive" increased 1.7% according to the test using HM 15.0.

B. The Detailed Algorithm

Based on the analysis made above, we proposed a hardware-oriented algorithm as presented in this paper.

In order to utilize fully the potential possibility of data reuse, the proposed algorithm searched from the center, then spread to the surroundings in diamond pattern. All search points in the same diamond pattern could benefit from 2-D data reuse, which made the search efficient and power-saving.

In order to balance the flexibility of starting points and the parallelism of search, the proposed algorithm took two types of steps: coarse steps and fine steps. In a coarse step, all PU partitions were searched in parallel from the same starting point (0, 0) without using MVPs. In a fine step, the best PU partition decided in the coarse step was refined further by searching around its own starting points if necessary.

Details of these steps are given as follows.

1) Coarse Step: In this step, the following points were tested and the best MV of each block was updated based on SAD cost only, instead of on the Rate-Distortion (RD) cost. The search worked as follows:

- Starting point (0, 0);
- each point on the diamond pattern with a stride of 1, then stride 2, 3, 4 ... 10;
- each point on the diamond pattern with a stride of 12, then stride 16, 20 and 24;
- each point on the diamond pattern with a stride of 32, then stride 40, 48, 56 and 64.

These searches are described by the solid lines in Fig. 3. After the above searches, the decision to PU partition was made. In order to get a more accurate partition results, both SAD costs and rate costs of MV differences (MVDs) are considered. However, the final MVs of the neighboring PUs

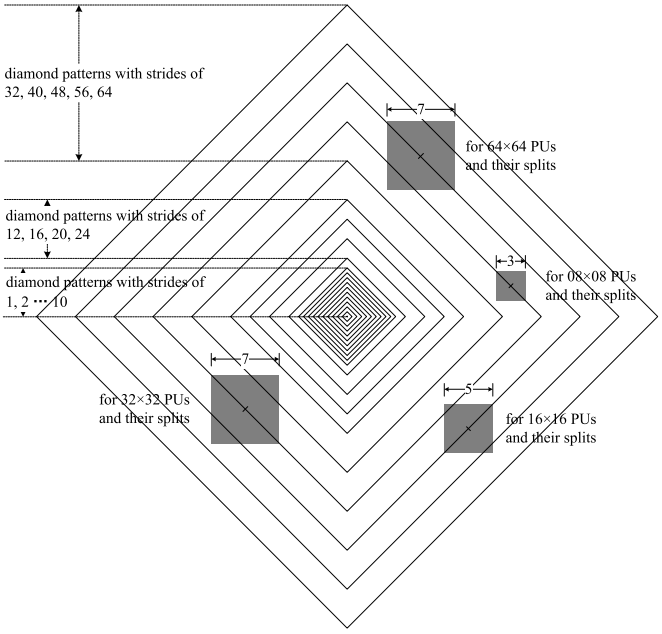


Fig. 3. The proposed algorithm.

are yet to be decided in the fine step. As a result, MVP of the current PU is generated based on the best MVs of the neighboring PUs got in the coarse steps.

2) Fine Step: After the PU partition decision was finished, MVs of the best PU partition were refined further if necessary. To be more specific, the following refinements were done if the best MV found in the coarse step was located in the diamond pattern whose stride was larger than 10.

- If the current block belonged to an 8×8 PU, a search range of $[-3, 3]$ from the corresponding MV was searched.
- If the current block belonged to a 16×16 PU, a search range of $[-5, 5]$ from the corresponding MV was searched.
- If the current block belonged to a 32×32 or 64×64 PU, a search range of $[-7, 7]$ from the corresponding MV was searched.

The above search process is described by the gray blocks.

IV. The Proposed Implementation

A. Overall Architecture

The overall architecture is shown in Fig. 4 (a). We proposed several novel ideas to solve problems in throughput, bandwidth, area, and power consumption, including:

- Horizontal and vertical (H-V) reference SRAM to provide 2-D data reuse, which is not supported by general reference SRAM structures;
- Data compression based on 4×4 blocks to reduce the area cost caused by on-chip reference SRAMs;
- PU-level chip selection to avoid unnecessary power consumption during the fine step and during the update of reference registers.

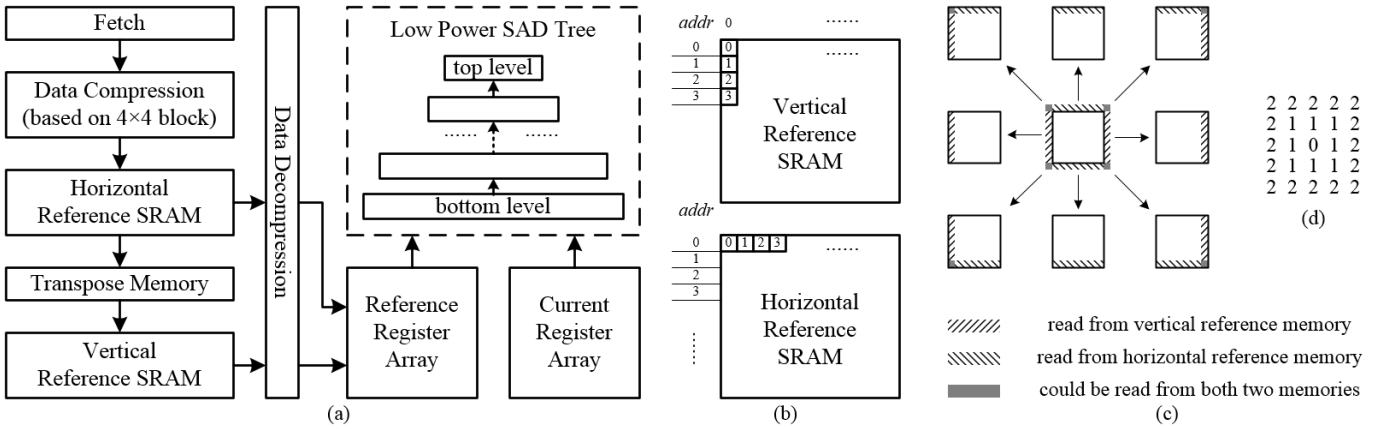


Fig. 4. (a) The proposed hardware architecture (b) data mapping in H-V reference SRAMs (c) update of register array (d) "checker board" distance.

B. Throughput and the Proposed H-V Reference SRAM

The efficiency of the proposed algorithm is based on an assumption that 2-D data reuse could be utilized. However, this data reuse level could not be adopted in general reference SRAM structures because of the access limitation of SRAMs, as mentioned in Section II A.

This kind of access limitation can be explained by Fig. 2 (b), in which the reference pixels are buffered by a large SRAM. Pixels in the zeroth row can be accessed with address 0; pixels in the first row can be accessed with address 1; and so on. In order to search one point for a 64 × 64 PU — for example, the 64 × 64 block of point **a**— 64 cycles are needed to load the 64 × 64 block stored in SRAM. However, after this block is loaded, by taking the advantage of 1-D reuse, the update of point **b**or TABLE I point **c** can be done in only one cycle because the missing pixels are located in just one address. In contrast, for the other adjacent points **d**, **e**, **f**, and **g**, or for the individual point **h**, missing pixels are located in different addresses. All of them require another 64 cycles to be updated. By using some extra registers [10], the cycle cost to prepare blocks for point **d** or **e** could be reduced to only 1 cycle. Unfortunately, the use of extra registers would be too costly to be applied to points **f** and **g** or points in even further positions such as point **h**. For this reason, 2-D reuse is not supported by existing designs.

In order to support 2-D reuse, we proposed using H-V reference SRAM, which adopted a horizontal SRAM and a vertical SRAM. As shown in Fig. 4 (b), the horizontal SRAM stored the reference pixels in horizontal order, and the vertical SRAM stored the reference pixels in vertical order. In other words, with address 0, the zeroth row could be accessed from the horizontal SRAM, and the zeroth column could be accessed from the vertical SRAM. Thus, if the search point moved vertically, missing pixels could be fetched from the horizontal reference SRAM in one cycle, which is similar to existing designs. If the search point moved horizontally, missing pixels could be fetched from the vertical SRAM in just one cycle, too. Even for diagonal directions, missing pixels could be fetched from two SRAMs in one cycle as well. The cycle cost to update the register array could be concluded by (1), which uses the "checker board" distance shown in Fig. 4 (c).

$$Cyc = \min(64,\ \max(|x_c - x_d|, |y_c - y_d|)) \tag{1}$$

To demonstrate the efficiency of H-V reference SRAM, we mapped separately an example of diamond search and an example of hexagonal search on the proposed architecture. As shown in Fig. 5, only 60 points were loaded into the register array to realize the diamond search with stride length of 8, which took 63 + 60 = 123 cycles. Only 82 points were loaded into the register array to realize the hexagonal search with the same stride length, which took 63 + 82 = 145 cycles. Here, black arrows indicate the search points loaded, while gray circles stand for the points calculated.

Attention should be paid to the following factors. First, the total cycle cost could be reduced further by rearranging the search order. Second, extra points loaded could either be sent to the SAD tree to improve the search results, or they could be bypassed to reduce the power consumption.

C. Bandwidth and Transpose Memory

From the above, it was easy to see the efficiency of the H-V reference SRAM. However, one critical problem remained to be solved: how to transfer the off-chip reference pixels to the vertical reference SRAM efficiently. Transferring pixels to the vertical reference buffer directly would do great harm to the bandwidth, considering that pixels belonging to the same column are stored discontinuously in the off-chip memories.

To solve this problem, a memory structure called transpose memory was adopted, to accomplish transposition towards the coefficients in the discrete cosine transform (DCT). As in the other designs, first the reference pixels were loaded into the horizontal reference SRAMs. Then they were transferred again from the horizontal reference SRAMs to the vertical reference SRAMs by the transpose memory.

It is obvious that the former transfer occupied the same bandwidth as the other designs, while the latter did not occupy any bandwidth. In fact, the latter transfer could be executed concurrently with other operations, for example, with loading

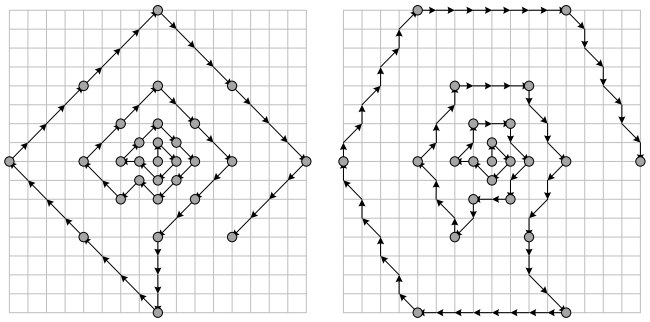


Fig. 5. Map of diamond search and hexagonal search.

current pixels or dumping reconstruction pixels. Thus, extra cycle costs were removed also.

Shang *et al.* [15] proposed an SRAM-based transpose memory, which could provide a throughput of 32 pixels/cycle with an extremely low hardware cost. By using such a memory, this design was able to prepare the reference pixels in the vertical reference SRAM easily without any extra bandwidth occupation or cycle cost.

D. Area Cost and the Proposed Block Compression Technique

Although H-V reference SRAM provided efficient 2-D data reuse, it doubled the area cost of on-chip reference buffers. As mentioned above, a commonly-adopted search range is $[-64, 64]$. In other words, $192 \times 192 = 36864$ pixels must be stored on chip, which is already a heavy burden.

Some reference compression algorithms [16], [17] were proposed to reduce the bandwidth to transfer reference pixels, but the area cost was not reduced. Ivanov and Moloney [19] proposed a 2×2 block-based compression technique which had a fixed data compression rate of 50%. In other words, their method could reduce both bandwidth and area cost to 50%. To further reduce the area cost, we proposed a data compression technique based on 4×4 blocks, which had a fixed data compression rate of 25%. According to Table I, our method led to a negligible increase in terms of BD rate. The detailed algorithm is given as follows.

Each 4×4 block, namely, $4 \times 4 \times 8 = 128$ bits, was compressed to 32 bits, 4 of which were used to store compression mode, and 28 of which were used to store truncated pixels. In other words, a total of $2\text{^}4 = 16$ modes were supported in this technique. Modes and the corresponding decompressed block are described in Fig. 6, where:

- Pixels in each 4×4 block are denoted by numbers 0∼f.
- The truncated pixels are marked by a suffixed single quote, which are obtained by discarding the last bit, as described by x' = x & 8'hfe in Fig. 6.
- The average operation towards a vertical line, horizontal line, or 2×2 block is marked by a prefixed "v", "h" or "b".
- "2×2 block" refers to pixel 0, 1, 4, 5, pixel 2, 3, 6, 7, pixel 8, 9, c, d or pixel a, b, e, f.

For decompression:

- In mode $0 \sim 7$, decompression was made based on truncated pixels in a horizontal or a vertical line.
- In mode $8 \sim 12$, decompression was made on truncated pixels in different 2×2 blocks.
- In mode $13 \sim 15$, decompression was made on the average of each horizontal line, vertical line, or 2×2 block.

TABLE I
BD RATE INCREASE OF THE PROPOSED DATA COMPRESSION

Class	Sequence	Y	U	V	Ave.
A	Traffic	0.1%	0.1%	0.1%	0.1%
	PeopleOnStreet	0.1%	0.3%	0.2%	0.2%
	Nebuta	0.1%	-0.3%	-0.1%	0.0%
	SteamLocomotive	0.1%	0.1%	0.4%	0.1%
B	Kimono	0.0%	-0.1%	0.2%	0.0%
	ParkScene	0.0%	0.0%	0.0%	0.0%
	Cactus	0.0%	0.1%	-0.1%	0.0%
	BasketballDrive	0.3%	0.2%	0.3%	0.3%
	BQTerrace	0.2%	0.2%	0.5%	0.2%
C	BasketballDrill	0.2%	-0.2%	-0.3%	0.1%
	BQMall	0.6%	0.3%	0.6%	0.5%
	PartyScene	0.2%	-0.1%	0.1%	0.1%
	RaceHorses	0.4%	-0.1%	0.4%	0.3%
D	BasketballPass	0.5%	0.1%	0.3%	0.4%
	BQSquare	0.3%	0.2%	0.4%	0.3%
	BlowingBubbles	0.4%	0.2%	0.5%	0.4%
	RaceHorses	0.4%	-0.1%	0.4%	0.3%
E	FourPeople	-0.2%	-0.1%	0.0%	-0.2%
	Johnny	-0.4%	-0.7%	-0.4%	-0.4%
	KristenAndSara	0.7%	0.7%	0.8%	0.7%
A	Average A	0.1%	0.1%	0.1%	0.1%
All	Average All	0.2%	0.1%	0.2%	0.2%

Anchor is HM-15.0 with default setting of encoder_lowdelay_P_main.cfg

Decision of these modes was made based on the SAD cost between the original and decompressed pixels. Regardless of the mode, decompression was always made based on just 4 truncated pixels. As shown in Fig. 7, in mode 0, the truncated value of pixel 0, 1, 2, 3 was used to do decompression, so the higher 7 bits of these data needed to be stored, which took $4 \times 7 = 28$ bits. The situation was quite similar for the other modes.

In addition, reference SRAMs that adopted this technique could be taken as normal SRAMs but with more write and read latency. This ability benefitted cooperation with H-V SRAMs, so combining this technique with H-V SRAMs proved to be straightforward. As shown in Fig. 8, to write original data, every 4 neighboring lines were compressed and stored together. In this way, the data width of SRAM remained unchanged, but the depth was reduced to one quarter. To read decoded data, for example, line x + 1, the corresponding line in reference SRAM was fetched and decompressed to recover line x + 1.

E. Power Consumption and the Proposed Low-Power SAD

For general SAD trees, costs of all PU levels are always calculated, which is suitable for full search and block-based search applications. However, for our proposed algorithm and other fast ME algorithms, some PUs could be bypassed to save unnecessary power consumption.

In addition, even by taking advantage of 2-D reuse, several cycles would still be needed to update the reference register array if the current search point and the next search point were relatively far away from each other. During this process, the entire SAD tree could be bypassed to save some power.

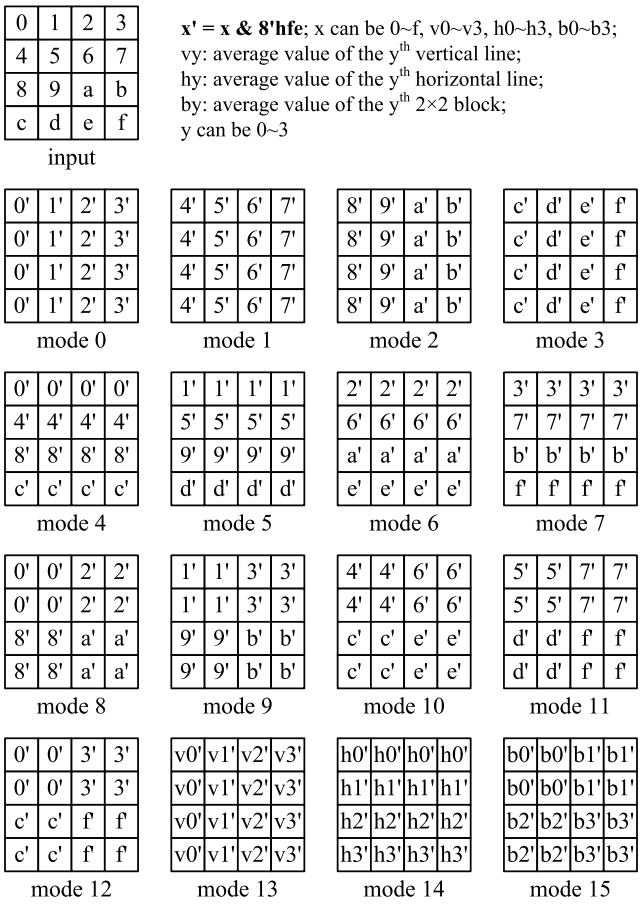


Fig. 6. The proposed data compression technique based on 4 × 4 block.

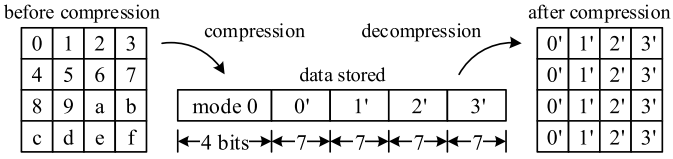


Fig. 7. Example of the proposed data compression technique.

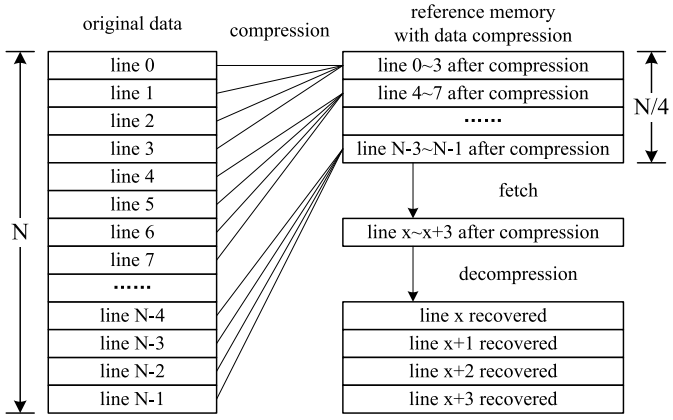


Fig. 8. Reference SRAM with the proposed data compression technique.

Based on the above two reasons, for our research we proposed a SAD tree with PU-level chip selection. An implementation of the 32 × 32 tree is presented in Fig. 9 as an example. Three pipeline stages were adopted in this tree,

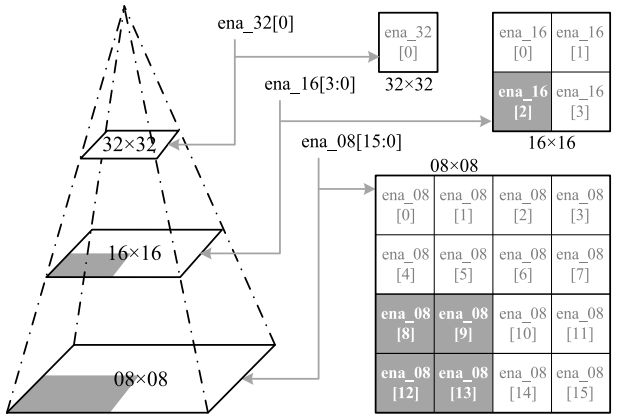


Fig. 9. Tree with PU-level chip selection: side view (left) and top view (right).

which were costs for 08 × 08 PUs and their splits, costs for 16 × 16 PUs and their splits, and costs for 32 × 32 PUs and their splits. To support the PU-level chip selection, each PU and its splits was given an individual enable signal, i.e., for PUs of 32 × 32 level, 16 × 16 level, and 08 × 08 level, 1, 4 and 16 enable bits were provided separately. For example, if ena_16[2] is valid, then only the gray parts in Fig. 9 are enabled.

V. Results and Comparisons

A. Software Performance

The proposed algorithm was realized and tested using HM 15.0 software by replacing the default IME search algorithm with the proposed one. As shown in Table II, the proposed algorithm achieved better BD rate performance than even HM 15.0, which may need some explanation here.

First, the anchor was HM 15.0 with the default setting, which adopts a fast search algorithm, not full search.

Second, by taking advantage of the H-V reference SRAM and SAD tree, the proposed solution searched much more densely around the shared starting point (0, 0), which is an advantage for a scene with relatively static motion. As for scenes with strong dynamic motions, the proposed solution would refine the search result for each PU if the coarse result were located on a diamond pattern with a stride of more than 10.

B. Hardware Performance

The corresponding hardware implementation was realized in RTL, tested by simulation, and verified by FPGA (TR4 development board), but not proved by silicon yet. Detailed performance results are provided below.

1) Throughput: The main contribution of the proposed H-V reference SRAM is that it can support 2-D reuse, fully utilizing the reusable data between neighboring samples in all of the eight directions. Table III lists the throughput of searching single points, along lines and within blocks.

The overall throughput of IME was calculated in the following way. Thanks to the H-V reference SRAM, the coarse step took only (31 + 1 + 0 + 4 + 0 + 8 + 0 + 12 + 0 +

TABLE II
BD RATE INCREASE OF THE PROPOSED ALGORITHM

Class	Sequence	Y	U	V	Average	Ye [9]	Medhat [14]	Jou [20]
A	Traffic	-0.8%	-0.7%	-0.8%	-0.8%	/	/	/
	PeopleOnStreet	-0.5%	-3.4%	-1.1%	-0.5%	/	/	/
	Nebuta	-0.1%	-1.3%	-0.3%	-0.1%	/	/	/
	SteamLocomotive	0.2%	0.7%	0.3%	0.2%	/	/	/
B	Kimono	-0.3%	-0.2%	-0.3%	-0.3%	/	/	/
	ParkScene	-0.7%	-0.5%	-0.7%	-0.7%	/	/	/
	Cactus	-0.6%	-1.3%	-0.7%	-0.6%	/	/	/
	BasketballDrive	0.4%	-0.4%	0.3%	0.4%	/	/	/
	BQTerrace	-1.1%	-3.1%	-1.6%	-1.1%	/	/	/
C	BasketballDrill	-0.5%	-0.4%	-0.5%	-0.5%	/	/	/
	BQMall	-0.8%	-0.4%	-1.3%	-0.8%	/	/	/
	PartyScene	0.0%	-0.3%	-0.1%	0.0%	/	/	/
	RaceHorsesC	0.6%	0.5%	0.2%	0.5%	/	/	/
D	BasketballPass	-0.3%	-0.2%	-0.3%	-0.3%	/	/	/
	BQSquare	-0.2%	-0.5%	-1.5%	-0.4%	/	/	/
	BlowingBubbles	-0.6%	0.1%	-0.5%	-0.6%	/	/	/
	RaceHorses	-0.2%	-1.0%	-1.2%	-0.5%	/	/	/
E	FourPeople	-0.2%	0.2%	-0.2%	-0.2%	/	/	/
	Johnny	-1.3%	-1.1%	-1.2%	-1.2%	/	/	/
	KristenAndSara	-1.2%	-1.0%	-1.1%	-1.2%	/	/	/
A	Average A	-0.2977%	-1.1832%	-0.7913%	-0.4701%	/	/	/
All	Average All	-0.4146%	-0.7027%	-0.7959%	-0.4983%	1.9%	0.92%	4.04%

Anchor of the proposed solution is HM-15.0 with default setting of encoder_lowdelay_P_main.cfg, but only one reference frame is allowed for each frame; Anchor of Ye[9] is TZ search in HM-11.0; Anchor of Medhat [14] is full search in x265; Anchor of Jou [20] is EPZS search in HM-5.0;

TABLE III
AVERAGE THROUGHPUT OF SEARCH (SAMPLE/CYCLE)

REUSE LEVEL (STRUCTURE)	POINT	LINE	BLOCK
no reuse (software method)	1/64	1/64	1/64
1-D reuse (general hardware)	64	1 or 1/64*	1 or <<1**
2-D reuse (proposed structure)	1/64	1	1

* "1" when searching along vertical lines, "1/64" when searching along other directions; ** "1" when using Byun *et al.*'s method [10], "<<1" when not using it.

$16 + 0 + 20 + 0 + 24 + 0 + 28 + 0 + 32 + 0 + 36 + 0 + 40 + 1 + 48 + 3 + 64 + 3 + 80 + 3 + 96 + 7 + 128 + 7 + 160 + 7 + 192 + 7 + 224 + 7 + 256) \times 4 = 6180$ cycles, where:

- 31 is the initial cycle for register arrays;
- $1 + 0$ is the cycle to search MV (0, 0) and the checker board distance between MV (0, 0) to a diamond pattern with a stride of 1 minus 1;
- $4 + 0$ is the cycle to search the diamond pattern with a stride of 1 and the checker board distance between it and the next diamond pattern of minus 1, and so on;
- $\times 4$ is needed because a 32×32 SAD tree was adopted instead of a 64×64 SAD tree to keep the hardware costs within a reasonable range. Since the smaller (32×32) calculation engine was used, more cycles were needed to get the results.
- Thanks to the simplicity of the proposed compression technique, the decompression operation here occupies no extra cycle.

As to the fine step, the cycle cost depended on the results achieved in the coarse step. In the hypothetical best case, no extra cycles were needed. In the worst case, all of the MVs garnered in the coarse step needed to be refined. If only 8×8 PUs existed, cycle cost was no more than $(7 + 7 \times (7 \div 7)) \times 64 \times 2 = 1792$, where:

- the first 7 is the maximum number of initial cycles for each block in an 8×8 PU;
- $7 \times (7 \div 7)$ is the cycle cost to search the window of $[-3, 3]$ with seven 8×8 engines working in parallel;
- 64×2 is the maximum MVs amount.

If only 16×16 PUs existed, cycle cost was no more than $(15 + 11 \times (11 \div 4)) \times 16 \times 2 = 1536$, where:

- 15 is the maximum number of initial cycles for each block in a 16×16 PU;
- $11 \times (11 \div 4)$ is the cycle to search the window of $[-5, 5]$ with four 16×16 engines working in parallel;
- 16×2 is the maximum MVs amount.

In a similar way, if only 32×32 PUs or a 64×64 PU existed, the cycle cost was no more than $(31 + 15 \times 15) \times 4 \times 2 = 2048$. As a result, the cycle cost of the fine step was no more than 2048. In other words, the maximum required working frequency for 4K $\times$ 2K@30fps videos was $3840 \times 2160 \div 64 \div 64 \times 30 \times (6180 + 2048) = 500$ MHz, and the minimum frequency was 375 MHz.

2) Bandwidth: Thanks to the transposed memories adopted and the 4×4 block compression technique proposed, bandwidth of the proposed design was reduced to 25%.

3) Area Cost: With the adoption of H-V reference SRAM, an extra vertical reference buffer was needed in this design. Fortunately, thanks to the data compression method proposed, the overall area cost of reference memories was even reduced.

Using ARM's memory compiler *sram_sp_hdc_svt_rvt_hvt*, the regular reference SRAM structure needed 12 SRAMs whose word number was 192 and word bit was 128. However,

TABLE IV
AREA COST OF THE PROPOSE ARCHITECTURES

STRUCTURE	GATE COUNT	FREQ
general reference SRAM	378.5 K	500 M
H-V reference SRAM	189.3 K	500 M
general SAD tree (32 × 32)	152.7 K	500 M
low-power SAD tree (32 × 32)	162.8 K	500 M
main control (including MVP generator)	3.3 K	500 M
data compression and decompression	6.7 K	500 M
reference register array	63.6 K	500 M
current register array	63.7 K	500 M
overall structure	489.4 K	500 M

TABLE V
POWER AND CYCLE COST OF MEMORY STRUCTURES

STRUCTURE	CYCLE	POWER
general reference memory structure	1856	133.5 mW
proposed reference memory structure	123	30.9 mW

the H-V reference SRAM structure with the proposed data compression technique needed only 6 SRAMs. Detailed gate counts are listed in Table IV, which were achieved under TSMC65 technology with Design Compiler.

Since the Least Significant Bit (LSB) of each pixel was discarded in the proposed data compression technique, the same modification was applied to the SAD tree as well.

4) Power Consumption: Compared with area cost, power consumption may have a higher priority in some embedded systems. In order to provide a quantitative comparison, our experiment implemented the diamond search with a stride length of 8 on the H-V reference memory structure and a general reference memory structure. According to Table V, the power consumption was reduced to 23.1%.

Table VI lists the power consumption of the SAD tree. If the entire tree was bypassed, power consumption was reduced to 18.8%. All results were obtained under TSMC65 technology with Prime Power.

C. Overall Comparison

Overall comparisons are given in Table VII. In terms of BD rate increase, the proposed design achieved the best performance among them.

In terms of area cost, the size of reference SRAM was greatly reduced thanks to the data compression technique proposed, while the gate count of the SAD tree was relatively larger because it supported all the block sizes in HEVC. (The size of reference SRAM was estimated by the search range. For example, the required size for a search range of [−16, 16] was $(16 \times 2 + 64)^2 = 9.216$ KB.)

In terms of throughput, only our proposed design considered the cycle cost to update the reference register array. The design also provided 2-D data reuse thanks to the H-V reference SRAM proposed, which makes the "maximum resolution" more meaningful. Comparison analyses to other works are given below.

TABLE VI
POWER CONSUMPTION OF SAD STRUCTURES (32 × 32)

STRUCTURE	POWER
general structure	96.4 mW
proposed SAD tree (all partitions enabled)	97.6 mW
proposed SAD tree (one 32 × 32 PU and its splits enabled)	90.8 mW
proposed SAD tree (one 16 × 16 PU and its splits enabled)	29.5 mW
proposed SAD tree (one 08 × 08 PU and its splits enabled)	20.5 mW
proposed SAD tree (entire tree disabled)	18.3 mW

By adopting a technique called PCTS, Ye *et al.* [9] claimed that their design succeeded in encoding 4K × 2K@30fps videos with a working frequency of only 200 MHz. However, this result may have been inaccurate because the cycle cost to update the reference pixels was omitted. According to their calculation, "only one cycle is required for every search candidate," but the checker board distances of most search points in their search steps were larger than 1. Of course, storing them by registers could provide the wanted throughput, but obviously it was unwise to store a search window of [−64, 64] with expensive registers.

By adopting a technique called FCSA, Medhat *et al.*'s [14] design succeeded in encoding 4K × 2K@30fps videos with a working frequency of 550MHz. However, the search range supported by them was only [-16, 16]. As mentioned previously, such a narrow search range would lead inevitably to a relatively high BD rate increase for videos with dynamic motion even if the full search were adopted.

Beyond this problem, Medhat el al. [14] also omitted a data reuse strategy in their paper. If the search window had been stored in SRAMs, the cycle cost to update the search point from one side to another would have increased dramatically with the distance from the center. If the search window had been stored in registers, the hardware cost would have increased dramatically with the search range. In other words, if their solution were to be implemented by hardware, the application scope might be limited to those with narrow search ranges.

Jou *et al.*'s [20] algorithm would "select the most probable search direction and steps through a statistical analysis to reduce the number of search points" and managed to encode 4 K × 2 K@60 fps videos. This approach seems attractive, but the following two points need to be considered.

First, their BD rate increase was as high as 4.04%, which meant the coding speed of the algorithm was excellent but the coding performance was poor. According to their paper, on the one hand, the search points were over-reduced by adopting a search method called predictive enhanced predictive zonal search (PEPZS). On the other hand, block sizes supported were over-simplified as well to reduce the calculation burden, which together led to a very high BD rate increase.

Second, their coding speed might have been exaggerated as well because the cycle costs to update the reference register array were not considered. If 2-D data reuse is not supported, their search method, PEPZS, would be quite inefficient because this method needs to take 8 different search directions.

TABLE VII
AREA COST OF THE PROPOSE ARCHITECTURES

DESIGN FEATURE	JSSC'14 [21]	JSTSP'13 [22]	VCIPC'14 [9]	ICECS'14 [14]	TCSVT'15 [20]	THIS WORK
Encoding standard	H264/AVC	HEVC	HEVC	HEVC	HEVC	HEVC
Technology	40 nm	65 nm	FPGA	FPGA	65 nm	65 nm
Working frequency	210 MHz	200 MHz	200 MHz	550 MHz	270 MHz	500 MHz
Supported resolution	8K × 4K P48	4K × 2K P60	4K × 2K P30	4K × 2K P30	4K × 2K P60	4K × 2K P30
Cycle cost	534 cyc/LCU	1628 cyc/LCU	3255 cyc/LCU	8952 cyc/LCU	2197 cyc/LCU	8228 cyc/LCU
Search method	/	/	PCTS	FCSA	PEPZS	the proposed search method
Supported PU sizes	8 × 8 to 16 × 16	16 × 16 to 64 × 64	/	8 × 8 to 32 × 32	8 × 8 to 64 × 64 (all in HEVC)	8 × 8 to 64 × 64 (all in HEVC)
Supported block sizes	/	/	12 sizes	/	8 sizes	28 sizes (all in HEVC)
Update of reference register array considered	×	×	×	×	×	√
Maximum search range	[-107,107] H [-56,56] V	[-64,64]	/	[-16,16]	[-64,64]	[-64,64]
Size of reference SRAM	/	36.9 KB**	/	9.2 KB**	36.9 KB**	18.4 KB
References SRAM data compression	×	×	×	×	×	√
2-D data reuse supported	×	×	×	×	×	√
Data reuse direction supported	2	2	2	2	2	8
Size of SAD tree	16 × 16	/	32 × 32	32 × 32	16 × 16	32 × 32
Normalized gate count of SAD tree	/	/	/	/	27 K	37 K *
BD rate increase	**/**	**12%**	**1.9%**	**0.92%**	**4.04%**	**-0.50%**

Anchor of the proposed solution is HM-15.0 with default setting of encoder_lowdelay_P_main.cfg, but only one reference frame is allowed for each frame; Anchor of Ye[9] is TZ search in HM-11.0; Anchor of Medhat [14] is full search in x265; Anchor of Jou [20] is EPZS search in HM-5.0; Anchor of Sinangil [22] is fast search in HM-3.0; * one 16 × 16 SAD tree is implemented to help the comparison. ** they are estimated data, estimated by (search range + 64)^2

As a matter of fact, Jou *et al.*'s [20] algorithm could be imported easily to our structure, since all the 8 directions needed would be provided efficiently by the H-V reference SRAM we have proposed. In other words, Jou *et al.*'s [20] design could be optimized by adopting the proposed H-V reference SRAM, as well as the proposed low-power SAD tree and 4 × 4 block based compression technique.

Fig. 8. Reference SRAM with the proposed data compression technique

Fig. 9. Tree with PU-level chip selection: side view (left) and top view (right)

VI. CONCLUSION

There are great challenges to implementing IME under the HEVC standard. This paper has presented a hardware-oriented IME algorithm that achieved a decrease in BD rate as compared to HM 15.0. Our corresponding hardware solution succeeded in supporting 4K × 2K@ 30 fps videos with a working frequency of no more than 500 MHz. These results demonstrate that our proposed solution offered desirable improvement in both coding speed and coding performance. For future study, we will investigate other possible algorithms further and test them on the proposed hardware platform.

REFERENCES

[1] N. Parmar and M. H. Sunwoo, "Enhanced Test Zone search motion estimation algorithm for HEVC," in *Proc. Int. SoC Design Conf. (ISOCC)*, Nov. 2014, pp. 260–261.

[2] S.-H. Yang, J.-Z. Jiang, and H.-J. Yang, "Fast motion estimation for HEVC with directional search," *Electron. Lett.*, vol. 50, no. 9, pp. 673–675, Apr. 2014.

[3] H. Kibeya, F. Belghith, H. Loukil, M. A. B. Ayed, and N. Masmoudi, "TZSearch pattern search improvement for HEVC motion estimation modules," in *Proc. 1st Int. Conf. Adv. Technol. Signal Image Process. (ATSIP)*, Mar. 2014, pp. 95–99.

[4] X. Li, R. Wang, X. Cui, and W. Wang, "Context-adaptive fast motion estimation of HEVC," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2015, pp. 2784–2787.

[5] L. P. Van, J. D. Cock, A. J. Diaz-Honrubia, G. V. Wallendael, S. V. Leuven, and R. V. D. Walle, "Fast motion estimation for closed-loop HEVC transrating," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 2492–2496.

[6] P. Nguyen *et al.*, "Asymmetric diamond search pattern for motion estimation in HEVC," in *Proc. IEEE 5th Int. Conf. Commun. Electron. (ICCE)*, Jul./Aug. 2014, pp. 434–439.

[7] P. Nalluri, L. N. Alves, and A. Navarro, "High speed SAD architectures for variable block size motion estimation in HEVC video coding," in *Proc. IEEE Int. Conf. Image Process. (ICIP)*, Oct. 2014, pp. 1233–1237.

[8] A. Medhat, A. Shalaby, and M. S. Sayed, "High-throughput hardware implementation for motion estimation in HEVC encoder," in *Proc. IEEE 58th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2015, pp. 1–4.

[9] X. Ye, D. Ding, and L. Yu, "A hardware-oriented IME algorithm and its implementation for HEVC," in *Proc. IEEE Vis. Commun. Image Process. Conf.*, Dec. 2014, pp. 205–208.

[10] J. Byun, Y. Jung, and J. Kim, "Design of integer motion estimator of HEVC for asymmetric motion-partitioning mode and 4K-UHD," *Electron. Lett.*, vol. 49, no. 18, pp. 1142–1143, Aug. 2013.

[11] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," in *Proc. Int. Conf. Inf., Commun. Signal Process. (ICICS)*, vol. 1. Sep. 1997, pp. 292–296.

[12] M. E. Sinangil, A. P. Chandrakasan, V. Sze, and M. Zhou, "Hardware-aware motion estimation search algorithm development for High-Efficiency Video Coding (HEVC) standard," in *Proc. 19th IEEE Int. Conf. Image Process. (ICIP)*, Sep./Oct. 2012, pp. 1529–1532.

[13] G. Sanchez, B. Zatt, M. Porto, and L. Agostini, "ES&IS: Enhanced spread and iterative search hardware-friendly motion estimation algorithm for the HEVC Standard," in *Proc. IEEE 20th Int. Conf. Electron., Circuits, Syst. (ICECS)*, Dec. 2013, pp. 941–944.

[14] A. Medhat, A. Shalaby, M. S. Sayed, M. Elsabrouty, and F. Mehdipour, "Fast center search algorithm with hardware implementation for motion estimation in HEVC encoder," in *Proc. 21st IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2014, pp. 155–158.
[15] Q. Shang, Y. Fan, W. Shen, S. Shen, and X. Zeng, "Single-port SRAM-based transpose memory with diagonal data mapping for large size 2-D DCT/IDCT," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 11, pp. 2423–2427, Nov. 2014.
[16] L. Guo, D. Zhou, and S. Goto, "A new reference frame recompression algorithm and its VLSI architecture for UHDTV video codec," *IEEE Trans. Multimedia*, vol. 16, no. 8, pp. 2323–2332, Dec. 2014.
[17] Y. Fan, Q. Shang, and X. Zeng, "In-block prediction-based mixed lossy and lossless reference frame recompression for next-generation video encoding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 1, pp. 112–124, Jan. 2015.
[18] X. Lian, Z. Liu, W. Zhou, and Z. Duan, "Lossless frame memory compression using pixel-grain prediction and dynamic order entropy coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 26, no. 1, pp. 223–235, Jan. 2016.
[19] Y. V. Ivanov and D. Moloney, "Reference frame compression using embedded reconstruction patterns for H.264/AVC decoder," in *Proc. 3rd Int. Conf. Digit. Telecommun. (ICDT)*, Bucharest, Romania, 2008, pp. 168–173.
[20] S. Y. Jou, S. J. Chang, and T. S. Chang, "Fast motion estimation algorithm and design for real time QFHD High Efficiency Video Coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 9, pp. 1533–1544, Sep. 2015.
[21] D. Zhou, J. Zhou, G. He, and S. Goto, "A 1.59 Gpixel/s motion estimation processor with -211 to $+211$ search range for UHDTV video encoder," *IEEE J. Solid-State Circuits*, vol. 49, no. 4, pp. 827–837, Apr. 2014.
[22] M. E. Sinangil, V. Sze, M. Zhou, and A. P. Chandrakasan, "Cost and coding efficient motion estimation design considerations for High Efficiency Video Coding (HEVC) standard," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, pp. 1017–1028, Dec. 2013.

Yibo Fan received the B.E. degree in electronics and engineering from Zhejiang University, Hangzhou, China, in 2003, the M.S. degree in microelectronics from Fudan University, Shanghai, China, in 2006, and the Ph.D. degree in engineering from Waseda University, Tokyo, Japan, in 2009.

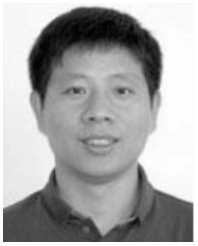
He was an Assistant Professor with Shanghai Jiao Tong University, Shanghai, China, from 2009 to 2010. He is currently an Associate Professor with the College of Microelectronics, Fudan University. His research interests include image processing, video coding, and associated VLSI architecture.

Leilei Huang received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2014, where he is currently pursuing the M.S. degree in microelectronics.

His research interests include VLSI design, algorithms and VLSI architectures for multimedia signal processing.

Bei Hao received the B.S. degree in microelectronics from Fudan University, Shanghai, China, in 2015, where she is currently pursuing the M.S. degree in microelectronics.

Her research interests include VLSI design, algorithms and VLSI architectures for multimedia signal processing.



Xiaoyang Zeng (M'05) received the B.S. degree from Xiangtan University, Xiangtan, China, in 1992, and the Ph.D. degree from Changchun Institute of Optics, Fine Mechanics, and Physics, Chinese Academy of Sciences, Changchun, China, in 2001. From 2001 to 2003, he was a Post-Doctoral Researcher with Fudan University, Shanghai, China. He joined the State Key Laboratory of ASIC and System, Fudan University, as an Associate Professor, and he is currently a Full Professor and the Director. His research interests include information security chip design, system-on-chip platforms, and VLSI implementation of digital signal processing and communication systems.