Journal of Real-Time Image Processing (2024) 21:25
https://doi.org/10.1007/s11554-023-01402-8

RESEARCH

An efficient versatile video coding motion estimation hardware

Waqar Ahmad[1,2] · Hossein Mahdavi[1] · Ilker Hamzaoglu[1,3]

Received: 11 November 2023 / Accepted: 14 December 2023

Abstract
Versatile Video Coding (VVC) is the latest video coding standard. It provides higher compression efficiency than the previous video coding standards at the cost of significant increase in computational complexity. Motion estimation (ME) is the most time consuming and memory intensive module in VVC encoder. Therefore, in this paper, we propose an efficient VVC ME hardware. It is the first VVC ME hardware in the literature. It has real time performance with small hardware area. This efficiency is achieved by using a 64 × 64 systolic processing element array to support maximum coding tree unit (CTU) size of 128 × 128 and by using a novel memory-based sum of absolute differences (SAD) adder tree to calculate SADs of 128 × 128 CTUs. The proposed VVC ME hardware reduces memory accesses significantly by using an efficient data reuse method. It can process up to 30 4 K (3840 × 2160) video frames per second.

Keywords Video compression · VVC · Motion estimation · Hardware

1 Introduction

As the amount of video data is increasing significantly, more efficient video compression is needed to transmit and store this video data with limited available bandwidth and storage space [1]. Therefore, Joint Video Experts Team (JVET) of ITU-T and ISO standardization organizations developed Versatile Video Coding (VVC) standard in 2020 [2]. VVC provides 50% higher compression efficiency than its predecessor High Efficiency Video Coding (HEVC) standard developed in 2013 [3, 4]. VVC is designed to encode diverse video content such as high dynamic range, 360° video and virtual reality [5].

VVC uses several new encoding tools to achieve better compression than HEVC such as new block partitioning structure called quadtree plus multi-type tree (QTMT), affine motion estimation and multiple transforms [6]. VVC divides a video frame into blocks called coding tree units (CTUs) and encodes each CTU separately. Each CTU can be further divided into coding units (CUs) using QTMT. QTMT allows more partitions than simple quadtree (QT) partitioning used in HEVC. The maximum CTU size in VVC is 128 × 128. The maximum CTU size in HEVC is 64 × 64.

VVC achieves higher compression efficiency than HEVC at the cost of significant increase in computational complexity. VVC encoder is 5 times and 31 times more complex than HEVC encoder under Low-Delay and All-Intra configurations, respectively [7]. The encoding time of VVC reference software encoder (VTM) is about 10 times more than the encoding time of HEVC reference software encoder (HM) [8]. Therefore, dedicated hardware implementations are needed for processing high resolution videos in real-time [9].

Successive frames in a video sequence have temporal redundancy. Video coding standards remove this temporal redundancy by performing motion estimation (ME). ME is the most time consuming and memory intensive module in video encoding [10]. More than 50% of the encoding time of VVC encoder is spent for ME [7]. Up to 60% of the memory accesses of VVC encoder comes from ME module [11].

There are several HEVC ME hardware in the literature [12–20]. Several sum of absolute differences (SAD) hardware that can be used for ME are proposed in the literature [21, 22]. There are several VVC intra prediction, fractional

✉ Ilker Hamzaoglu
ilker.hamzaoglu@ozyegin.edu.tr

Waqar Ahmad
waqar.ahmad@giki.edu.pk

Hossein Mahdavi
hmahdavi@sabanciuniv.edu

1 Sabanci University, Tuzla, 34956 Istanbul, Turkey

2 Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Topi 23460, Pakistan

3 Ozyegin University, Çekmeköy, 34794 Istanbul, Turkey

Published online: 29 January 2024

interpolation and transform hardware in the literature [23–26]. However, to the best of our knowledge, there is no VVC ME hardware in the literature.

VVC ME has higher computational complexity than HEVC ME because of using a larger maximum CTU size and a more complex block partitioning structure called QTMT. Two types of SAD adder trees are proposed in the literature for HEVC ME hardware. Fully parallel SAD adder tree processes all the pixels in the largest CU in parallel. Sequential SAD adder tree divides the largest CU into smaller blocks and processes each block in successive clock cycles. Using a fully parallel SAD adder tree in VVC ME hardware results in very large hardware area. Using a sequential SAD adder tree in VVC ME hardware results in low data reuse and low throughput. Therefore, the methods proposed in the literature for designing HEVC ME hardware are inefficient for designing VVC ME hardware.

In this paper, we propose the first VVC ME hardware in the literature. The proposed hardware uses the Full Search ME algorithm with the SAD metric to find the best motion vector for a wide range of QTMT partition sizes, ranging from 8×4 (4×8) to 128×128, within a CTU. The proposed hardware calculates SADs of 128×128 CTU using a 64×64 systolic processing element array and a novel memory-based SAD adder tree to achieve real-time performance with small hardware area. It reduces memory accesses significantly by using an efficient data reuse method.

The proposed novel memory-based SAD adder tree combines the features of fully parallel and sequential SAD adder trees. It is highly efficient as it achieves high data reuse and high throughput, and it uses smaller hardware area than a fully parallel 128×128 SAD adder tree.

The proposed VVC ME hardware is implemented using Verilog HDL. It works at 253 MHz on a Xilinx Virtex 7 FPGA, and it can process up to 30 4K (3840×2160) video frames per second (fps).

The rest of the paper is organized as follows. In Sect. 2, VVC ME is explained. Section 3 describes the proposed VVC ME hardware. Its implementation results and comparison with HEVC ME hardware in the literature are given in Sect. 4. Finally, Section 5 concludes the paper.

2 VVC motion estimation

VVC uses block matching for translational ME. In block matching, current video frame is divided into blocks. As shown in Fig. 1, for each block in the current frame, the best matching block in a search window (SW) in the reference frame and the corresponding motion vector (MV) are determined. SAD metric is typically used to determine the best matching block. SAD between blocks A and B is calculated as shown in Eq. (1), where W×H is the block size, A(i, j) and B(i, j) are pixels in ith row and jth column of A and B, respectively.

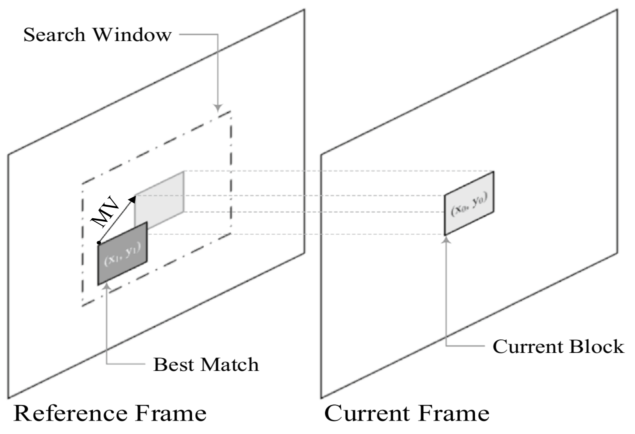

Fig. 1 Block matching motion estimation

$$SAD = \sum_{i=0}^{W-1} \sum_{j=0}^{H-1} |A(i,j) - B(i,j)| \tag{1}$$

Video coding standards perform variable block size ME. Large block sizes achieve higher compression for the smooth areas in video frames, whereas small block sizes achieve higher compression for the areas with more details. Because large block sizes can find good matches for the smooth areas using less MVs than small block sizes. However, for the areas with more details, using large block sizes results in large residue. For such areas, small block sizes find better matches resulting in small residue and better compression despite using more MVs than large block sizes.

In variable block size ME, for each block in the current frame, all sub-block sizes are used to determine the best match in its search window in the previous frame, and the sub-block size achieving the best compression is used for that block. Therefore, variable block size ME achieves higher compression than fixed block size ME.

Both HEVC and VVC divide a video frame into blocks called CTU. In HEVC, the maximum CTU size is 64×64. A CTU can be recursively partitioned into square-shaped CUs using QT. The size of a CU can be from 8×8 to 64×64. A CU can be partitioned only once into square, rectangular and asymmetric partitions called prediction unit (PU). The PU size can be from 4×8 or 8×4 to the CU size for motion estimation.

In VVC, the maximum CTU size is 128×128. A CTU can be recursively partitioned into CUs using QTMT [27]. QTMT achieves higher compression than QT used in HEVC by allowing more partitions than QT.

QTMT is a tree in which a node can be split using QT, binary tree (BT) or ternary tree (TT). A BT splits a node into

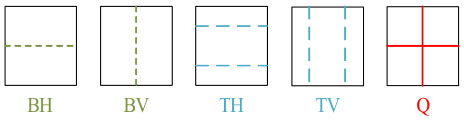

Fig. 2 Allowed partitions in VVC

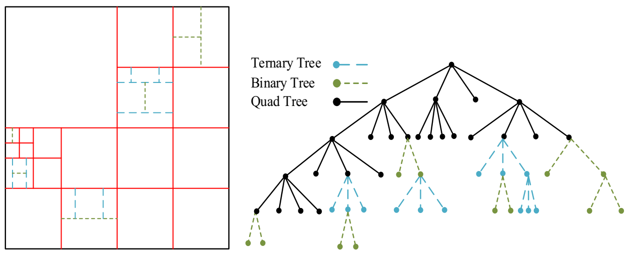

Fig. 3 An example QTMT partitioning of 128 × 128 CTU and its decision tree

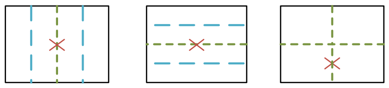

Fig. 4 Examples of redundant partitions in VVC

two rectangular blocks. A TT splits a node into three rectangular blocks, two of which have the same size. BT and TT splits can be applied in horizontal or vertical direction. In Fig. 2, five possible QTMT partitions are shown; binary horizontal (BH), binary vertical (BV), ternary horizontal (TH), ternary vertical (TV), quad (Q).

There are some restrictions in QTMT partitioning [28]. If a node is split with QT, it can be further split with any of the five QTMT partitions. However, if a node is split with either BT or TT, it can no longer be further split with QT. An example of QTMT partitioning of 128×128 CTU is shown in Fig. 3.

As shown in Fig. 4, the same partitions can be achieved with different splitting patterns. If the central partition of a TT split is further split with BT in the same direction, it achieves the same partitions with BT split followed by BT split in the same direction. Similarly, QT split achieves the same partitions with BT split in one direction followed by BT split in the other direction. VVC does not allow these redundant partitions [28].

In addition to translational ME, VVC uses affine motion estimation (AME) to predict more complex motion such as rotation or scaling. Turning off AME in VVC video encoding causes 3% loss in compression efficiency but provides 20% encoding time reduction [29]. It is reported in [30] that translational motion estimation is used for more than 90% cases in VVC video encoding. Therefore, in this paper, we propose an efficient ME hardware for VVC translational ME.

3 Proposed VVC motion estimation hardware

VVC defines the following control parameters to adjust the computational complexity of ME by restricting the number of partitions. *MaxCUWidth* and *MaxCUHeight* define the maximum allowed width and height of a CU, respectively. *MinQTSize* defines the minimum node size that can be reached with QT split. *MaxBtSize* and *MaxTtSize* define the maximum node size to which BT and TT split can be applied, respectively. *MaxMttDepth* defines the maximum allowed depth of multi-type tree splitting after QT split.

In the proposed VVC ME hardware, *MaxCUWidth* and *MaxCUHeight* are set to 128. Therefore, the largest CU size is 128×128. *MinQTSize* is set to 8. Therefore, an 8×8 CU can only be further split with BT. *MaxBtSize* and *MaxTtSize* are set to 32. *MaxMttDepth* is set to 2. Therefore, multi-type tree split is not applied to CU sizes larger than 32×32. The maximum depth of multi-type tree split is 2, i.e., multi-type tree split can be applied at most twice.

The number of possible partitions in a 64×64 CU with these parameter values are shown in Table 1. Let X and Y represent one of the four possible multi-tree type partitions shown in Fig. 2, then the partition type X_Y in Table 1 represents the case where first X type split then Y type split are applied after QT split. For example, BH_BH partition type represents the case where first binary horizontal split is applied after QT split, then binary horizontal split is applied to the 2 new partitions resulting in 4 partitions.

The number of unique MVs is less than the number of partitions for some split types. For example, a TH split of a QT node results in 3 partitions. The top and bottom partitions of TH split are the same as top and bottom partitions of BH_BH split. Therefore, there is no need to calculate MVs for top and bottom partitions of TH split and only 1 of the 3 TH partitions require a unique motion vector.

Redundant partitions, which are not allowed in VVC, are not shown in Table 1. For example, BH_BV split achieves the same partitions with QT split. Therefore, it is not allowed in VVC. In addition, some partitions are not allowed since they result in a partition size with height or width smaller than the minimum allowed CU size. These partitions are also not shown in Table 1. For example, when a 16×16 block is split with ternary tree, its further split with ternary tree will result in a partition size of 8×2 or smaller. This is smaller than the minimum allowed CU size. Therefore, this is not allowed.

The proposed VVC ME hardware is shown in Fig. 5. For a 64×64 CU, it only processes the partitions shown in Table 1. It does not process the partitions which are not allowed by VVC. It consists of on-chip memory to store search window pixels and next block of current frame, a

Table 1 Number of possible partitions and unique MVs in a 64×64 CU

Block size	Partition type	Partitions	Unique MVs
64×64	No Partition	1	1
	Q	4	4
32×32	Q	16	16
	BH	8	8
	BV	8	8
	TH	12	4
	TV	12	4
	BH_BH	16	16
	BH_TH	24	24
	BH_TV	24	8
	BV_BV	16	16
	BV_TH	24	8
	BV_TV	24	24
	TH_BH	24	16
	TH_TH	20	20
	TV_BV	24	16
	TV_TV	20	20
16×16	Q	64	64
	BH	32	32
	BV	32	32
	TH	48	16
	TV	48	16
	BH_BH	64	64
	BH_TV	96	32
	BV_BV	64	64
	BV_TH	96	32
8×8	BH	128	128
	BV	128	128
Total		1077	821

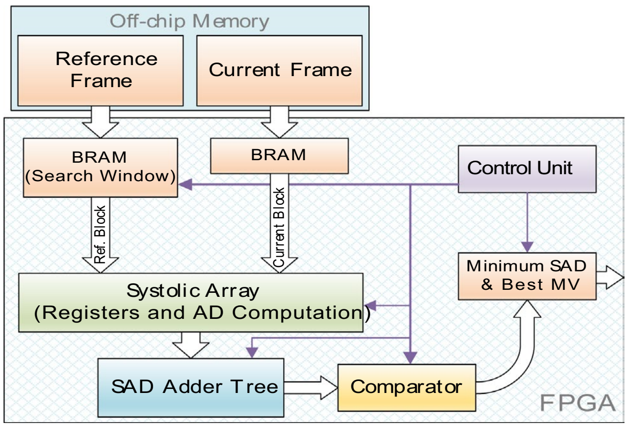

Fig. 5 Proposed VVC motion estimation hardware

systolic array of processing elements (PEs) to store current and reference block pixels and calculate their absolute differences, an SAD adder tree to calculate SADs for all the supported CU sizes, a comparator unit to determine the minimum SAD and its corresponding MV for each CU size, and a control unit to perform control operations.

To achieve real-time performance with small hardware area, the proposed VVC ME hardware divides 128×128 CTU into four 64×64 CUs. It uses a 64×64 systolic PE array and 64×64 SAD adder tree to determine the best 821 unique MVs for each of these 64×64 CUs sequentially. First, the best 821 unique MVs for the first 64×64 CU are determined. Then, the remaining three 64×64 CUs are processed one by one.

The proposed VVC ME hardware uses an efficient data reuse method to significantly reduce the memory accesses required for processing each 64×64 CU and its partitions. It uses a novel memory-based SAD adder tree to determine the best MV for 128×128 CU. The best MV for 128×128 CU is determined together with the best MVs of last 64×64 CU.

Since the proposed VVC ME hardware divides 128×128 CTU into four 64×64 CUs, it uses a 64×64 systolic PE array which is 4 times smaller than a 128×128 systolic PE array. Since it stores search window pixels of 64×64 CU in on-chip memory instead of storing search window pixels of 128×128 CTU in on-chip memory, it uses less on-chip memory as well. Although this significantly reduces area of the proposed VVC ME hardware, it has real time performance. This efficiency is achieved by using the efficient data reuse method based on vertical snake scan of the search window and the novel memory-based SAD adder tree.

3.1 On-chip memory and systolic array

FPGAs have fast dedicated on-chip memories called Block RAMs (BRAMs). In the proposed hardware, the current 64×64 CU and its corresponding search window are read from off-chip memories and stored in the on-chip BRAMs. A ±64 search range centered around the current 64×64 CU defines a 128×128 search window. This requires storing 192×192 reference frame pixels in the BRAMs. In the proposed hardware, the first and last 64 columns are stored in 8 BRAMs of size 384×64 bits each. The middle 64 columns are stored in 8 BRAMs of size 192×64 bits each. The 64×64 pixels of current frame are stored in 8 BRAMs.

The proposed hardware has a 64×64 systolic PE array as shown in Fig. 6. The systolic array also contains 64 registers to store an additional column of the search window. As shown in Fig. 7, a PE consists of two registers which store a current block pixel and a reference block pixel, an absolute difference (AD) hardware, and an output register. AD hardware subtracts the reference pixel from the current pixel.

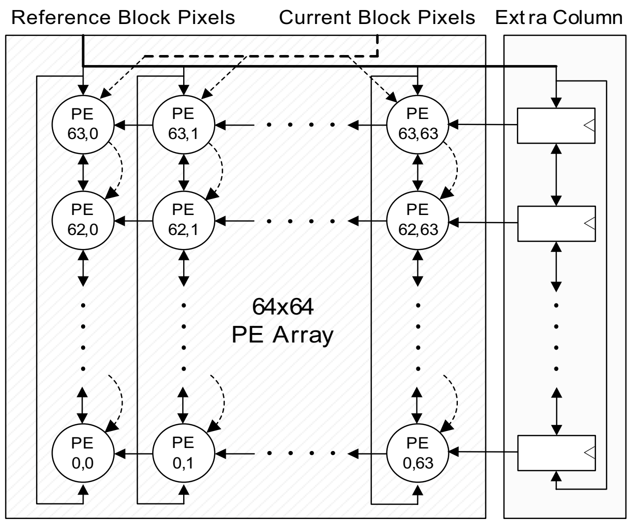

Fig. 6 Systolic processing element (PE) array and registers

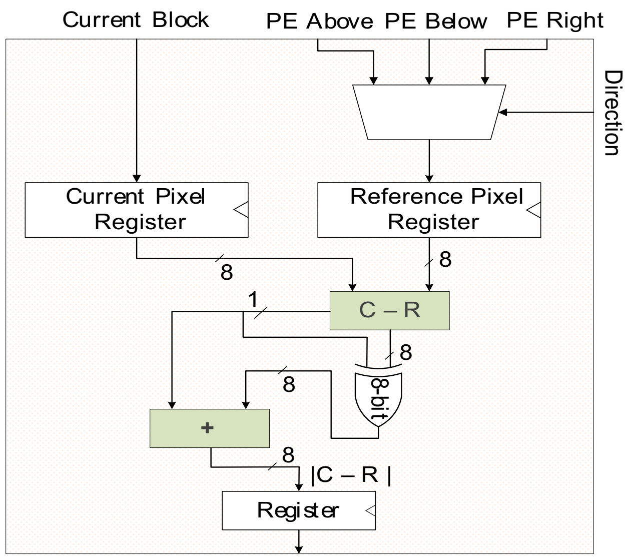

Fig. 7 Processing element (PE)

If the subtraction result is negative, i.e., its sign bit is 1, it takes its 2’s complement to calculate the absolute difference.

The systolic array receives a new row of 64 reference block pixels from BRAMs in every clock cycle. It takes 64 clock cycles to fill the systolic array with the 64 × 64 reference block of the first search location. At the same time, the 64 × 64 current block pixels are also received from BRAMs and stored in the systolic array row by row. After that systolic array calculates 64 × 64 absolute differences in one clock cycle and sends them to SAD adder tree which calculates the SADs for all the partitions of 64 × 64 CU.

The systolic array stores the same 64 × 64 current block until all the search locations in the search window are searched for that current block. It can search a new search location in the search window in every clock cycle, i.e., it can process 64 × 64 reference block of each search location in one clock cycle.

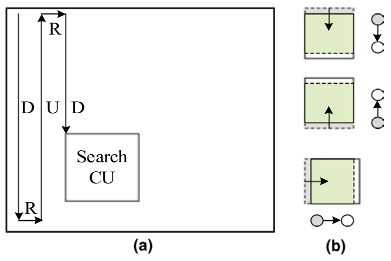

Fig. 8 **a** Vertical snake scan order, **b** Data reuse in downward, upward, and right directions

To achieve high data reuse, the proposed hardware uses the vertical snake scan order shown in Fig. 8a. The search starts from the top-left corner of the search window and moves downward until all the search locations in the first column are searched. Then, the search locations in the second column are searched in the upward direction. Then, the search locations in the third column are searched in the downward direction. This continues until all the search locations in the search window are searched for the current block.

To achieve high data reuse, each PE can shift its reference pixel up, down, or left. After a search location in a column, which is searched in the downward direction, is searched, all the PEs shift their reference pixels up, and a new row of 64 reference block pixels is read from search window memory and stored in the last row of systolic array as shown in Fig. 8b. This continues until all the search locations in that column are searched.

In Fig. 8b, green area represents the reused reference block pixels in the systolic array, white area represents the new row of 64 reference block pixels, and grey area represents the discarded row of 64 reference block pixels in the previous reference block.

After searching a location in a column, which is searched in the upward direction, all the PEs shift their reference pixels down. Then, a new row of 64 reference block pixels is read from search window memory and stored in the first row of systolic array as shown in Fig. 8b. This continues until all the search locations in that column are searched.

Once all the search locations in a column are searched, all the PEs shift their reference pixels left, and a new column of 64 reference block pixels should be stored in the last column of systolic array. Since row aligned BRAMs are used in the proposed hardware, it would take 64 clock cycles to read a new column of 64 reference block pixels from BRAMs.

Therefore, an extra column of 64 registers is used in the systolic array. In every clock cycle, instead of 64, a new row of 65 reference block pixels is read from search window memory and stored in the systolic array. Therefore, after all the search locations in a column are searched, all the PEs shift their reference pixels left, and the PEs in the last column of systolic array receive their new reference pixels from the extra column of 64 registers. This takes only one clock cycle.

The 192 × 192 search window pixels are divided into three blocks of 192 × 64 pixels. The first and third blocks are stored together in 8 BRAMs. The middle block is stored in other 8 BRAMs. To access 65 reference block pixels, 128 pixels (64 pixels each from the first and second block, or 64 pixels each from the second and third block) are read from these 16 BRAMs.

The sequence in which search window pixels are read from the BRAMs remains constant. However, the arrangement of pixels needed by the PE array and temporary registers changes based on the location of the column being processed. Therefore, a horizontal data rotator hardware is used to reorder the 128 pixels such that the required 65 pixels are always in the MSB positions. These 65 pixels are sent to systolic PE array and the extra column of registers.

In the proposed hardware, BRAMs are configured as true dual port memories. After the current 64 × 64 CU is stored in the systolic array, the next 64 × 64 CU of the current frame is read into the BRAMs from off-chip memory. Similarly, the search window BRAMs are also updated dynamically with the search window of the next 64 × 64 CU of the current frame from off-chip memory.

3.2 SAD adder tree

In HEVC, the maximum CTU size is 64 × 64, and 593 unique MVs should be calculated for a 64 × 64 CU [15]. In VVC, the maximum CTU size is 128 × 128, and 821 unique MVs should be calculated for a 64 × 64 CU.

In addition, in VVC, there are more complex asymmetric partitions which are not used in HEVC. Therefore, SAD adder tree in VVC ME hardware is more complex than SAD adder tree in HEVC ME hardware.

In the proposed hardware, the SAD adder tree calculates the SADs of all the 821 unique partitions of a 64 × 64 CU by reusing the SADs of smaller partitions to calculate the SADs of larger partitions.

After the SAD adder tree receives 64 × 64 ADs for the first search location from the systolic array, it receives and processes 64 × 64 ADs of a new search location in every clock cycle. For each 64 × 64 ADs, the corresponding 256 4 × 4 SADs are calculated in four clock cycles. One 4 × 4 SAD calculation including the AD calculation in PEs is shown in Fig. 9. The red dotted lines in the figure indicate the pipeline registers.

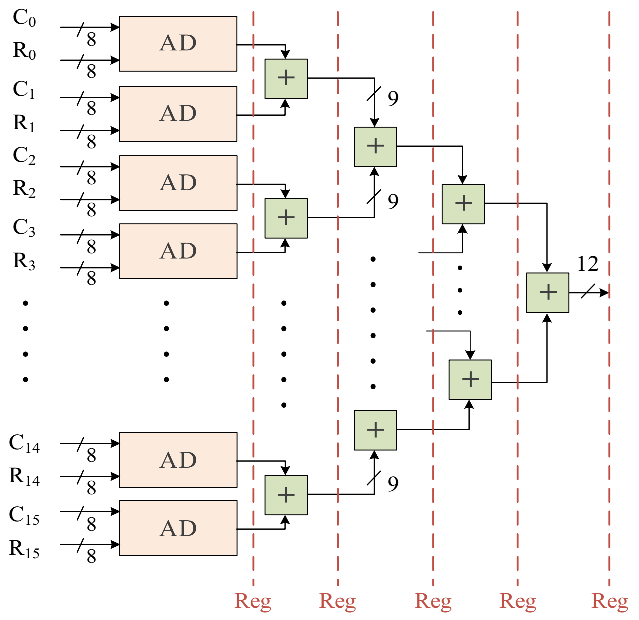

Fig. 9 4 × 4 SAD calculation

These 4 × 4 SADs are then used to calculate SADs of larger partitions in a hierarchical manner. For example, 4 × 4 SADs are used to calculate SADs of binary partitions (BV, BH) of 8 × 8 CUs. Then, the SADs of BV partitions of 8 × 8 CUs are used to calculate 64 SADs of 8 × 8 CUs. SADs of binary and quad partitions of 16 × 16, 32 × 32, and 64 × 64 CUs are calculated similarly as shown in Fig. 10a.

Similarly, the SADs of BV and BH partitions of 8 × 8 CUs are used to calculate SADs of BH_BH, BV_BV, BV_TH and BH_TV partitions of 16 × 16 CUs. Then, the SADs of BH_BH and BV_BV partitions are used to calculate SADs of TH and TV partitions of 16 × 16 CUs. SADs of the same shaped partitions of 32 × 32 CUs are calculated similarly using BV and BH partitions of 16 × 16 CUs as shown in Fig. 10b.

SADs of TH_TH, TV_TV, TH_BH, TV_BV, BH_TH and BV_TV partitions of 32 × 32 CUs are calculated using BH_BH and BV_BV partitions of 16 × 16 CUs as shown in Fig. 10c.

The proposed hardware calculates the SADs of all the 821 unique partitions of a 64 × 64 CU for the first search location in the search window in 13 clock cycles and sends them to the comparator. After that, 821 new SADs are calculated in every clock cycle and sent to the comparator.

To achieve real-time performance with small hardware area, the proposed hardware divides 128 × 128 CU into four 64 × 64 CUs, processes them one by one and calculates SAD of 128 × 128 CU using the novel memory-based accumulator hardware shown in Fig. 11. In the BRAM, 4096 SAD values, each 22 bits, are stored.

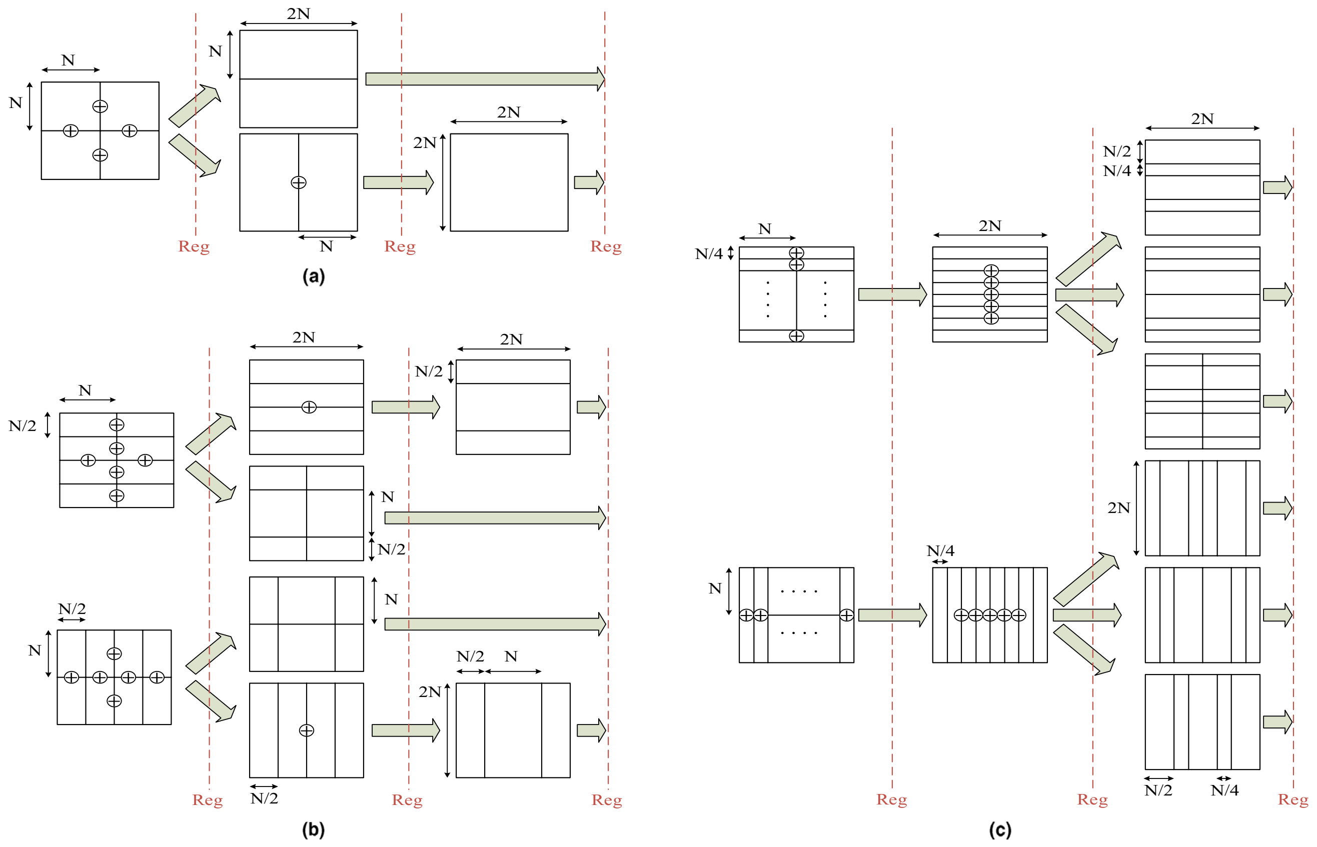

Fig. 10 SAD adder tree **a** SADs of BH, BV, Q partitions N = 4, 8, 16, 32 **b** SADs of BH_BH, BV_TH, BH_TV, BV_BV, TH, TV partitions N = 8, 16 **c** SADs of TH_TH, TH_BH, BH_TH, BV_TV, TV_BV, TV_TV partitions N = 16

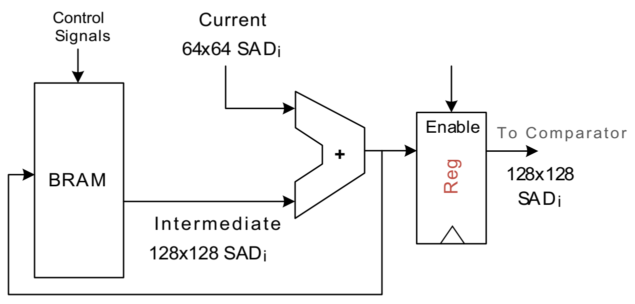

Fig. 11 128 × 128 SAD calculation

The top left 64 × 64 CU is processed first. For every ith search location in the search window, the SAD calculated for this 64 × 64 CU is sent to both the comparator and the memory-based accumulator where it is added to the content of ith location of BRAM and the result is written back to ith location of BRAM. The contents of the BRAM are initially set to 0. Therefore, the SADs of the top left 64 × 64 CU are stored in the BRAM.

Then, the top right 64 × 64 CU is processed. Therefore, the ith SAD of the top right 64 × 64 CU is added to the ith SAD of the top left 64 × 64 CU, and the result is written back to ith location of BRAM. Then, the bottom left 64 × 64 CU is processed similarly. Finally, the bottom right 64 × 64 CU is processed similarly.

When the first SAD of the bottom right 64 × 64 CU is added to the content of the first location of BRAM, the adder output is the first SAD of the 128 × 128 CU. Therefore, the output register in Fig. 11 is enabled, and the first SAD of the 128 × 128 CU is sent to the comparator. After that, a new SAD of the 128 × 128 CU is calculated in every clock cycle and sent to the comparator. When the last SAD of the bottom right 64 × 64 CU is calculated, the last SAD of the 128 × 128 CU is also calculated after one clock cycle and sent to the comparator.

3.3 Comparator

The comparator unit determines the minimum SAD and its corresponding best MV for each CU size. It consists of one comparator for each of the 821 unique partitions of 64 × 64 CU and one additional comparator for the 128 × 128 CU. The sizes of these comparators vary from 13-bits for the smallest CU to 22-bits for the 128 × 128 CU. The latency of the comparator unit is one clock cycle. In every clock cycle, it compares all the SADs it receives from the SAD adder tree with the previous minimum SADs of the corresponding partitions and determines the minimum SAD and its corresponding best MV for each partition. The cost of MV bits is not considered in determining the best MV.

4 Implementation results

The proposed VVC ME hardware is implemented using Verilog HDL. The Verilog RTL codes are implemented on a Xilinx Virtex 7 FPGA using Xilinx Vivado 2017.4 with the *area-optimized_high* synthesis strategy and the *performance_explore* implementation strategy. The FPGA implementation is verified with post-implementation timing simulations.

The proposed hardware has 14 stages pipeline from AD calculation to comparator output. The latency for processing a 128 × 128 CTU can be calculated as (64 + 14 + Search Locations) × 4. The systolic array is filled in 64 clock cycles. It takes 14 clock cycles to calculate the SADs of all the CUs for the first search location in the search window and compare them. After that, all the CUs for a new search location are processed in every clock cycle. Multiplication by 4 is necessary since a 64 × 64 SAD adder tree is used and a 128 × 128 CTU has four 64 × 64 CUs.

We implemented and verified the proposed VVC ME hardware in two different configurations for three different search ranges. One configuration supports 128 × 128 largest CTU size using a 64 × 64 systolic array and 64 × 64 SAD adder tree. The other configuration supports 64 × 64 largest CTU size using a 32 × 32 systolic array and 32 × 32 SAD adder tree. In each configuration, the size of the search range is set to the largest CTU size, 75% of the largest CTU size, and half of the largest CTU size.

The search range is centered around the top left pixel of current CTU. A search range of 128 × 128 means that the first pixel of first reference CTU is located at position (-64, -64) left of the first pixel of current CTU in the search window. Similarly, the first pixel of last reference CTU is located at position (+ 64, + 64) right of the first pixel of current CTU in the search window.

Performance of the proposed VVC ME hardware for different configurations are shown in Table 2. The clock frequency (MHz), the number of clock cycles required to process a current CTU, and the throughput in frames per second (fps) for three different video resolutions (full HD (FHD), 2K, 4K) are shown in the table. The throughput in fps is calculated as shown in Eq. (2).

Table 2 Performance of the proposed VVC ME hardware for different configurations

CTU size	128 × 128			64 × 64		
Search Range	128 × 128	96 × 96	64 × 64	64 × 64	48 × 48	32 × 32
Frequency (MHz)	253	253	253	306	306	306
CTU Latency	65,848	37,176	16,696	16,560	9,392	4,272
FPS at 1080p	30	53	120	36	64	141
FPS at 2 K	28	50	112	34	60	132
FPS at 4 K	7	13	30	9	16	35

$$fps = \frac{1}{CTUlatency \times CTUsperframe \times Clockperiod} \quad (2)$$

For 128 × 128 largest CTU size with 128 × 128 search range, 30 fps throughput is achieved for full HD video resolution. If the search range is reduced to 64 × 64, 30 fps throughput is achieved for 4K video resolution. For 64 × 64 largest CTU size with 32 × 32 search range, 35 fps throughput is achieved for 4K video resolution.

The proposed VVC ME hardware does not implement multi-type tree splitting for CU sizes larger than 32 × 32. The default configuration of VVC reference software (VTM) allows several multi-type tree splits larger than 32 × 32. The proposed hardware implements multi-type tree depth of 2. The default multi-type tree depth in VTM is 3.

We assessed the impact of the proposed hardware on VVC coding efficiency. Table 3 presents the rate-distortion performance for six different configurations of the proposed hardware with varying CTU sizes and search ranges for four full HD (1920 × 1080) videos using VTM v22.1. VTM is used in default configuration except the search algorithm, which is changed to Full Search. The VTM evaluations do not consider the impact of merge modes. It is assumed that they are checked in the mode decision module.

The FPGA resource usages of the proposed VVC ME hardware for 128 × 128 largest CTU size configuration with 128 × 128 search range and for 64 × 64 largest CTU size configuration with 64 × 64 search range are shown in Tables 4 and 5, respectively. The resource usage of 64 × 64 largest CTU size configuration is almost 4 times less than the resource usage of 128 × 128 largest CTU size configuration.

The systolic array uses the most FPGA resources. It uses 54% of the total flip-flops and 38% of the total LUTs used by the 128 × 128 largest CTU size configuration. The current pixel registers, reference pixel registers, and output registers in the systolic array justify the amount of flip-flop usage.

The SAD adder tree uses the second most FPGA resources. It uses 31% of the total flip-flops and 28% of the total LUTs used by the 128 × 128 largest CTU size configuration. Since the comparator unit uses registers to store the minimum SADs and corresponding best MVs, its flip-flop usage is higher than its LUT usage.

The proposed VVC ME hardware is the first VVC ME hardware in the literature [31]. The proposed VVC ME hardware implementation is compared with the HEVC ME hardware implementations in the literature in Table 6. Although VVC ME has larger maximum CTU size and it is more computationally complex than HEVC ME, the proposed VVC ME hardware has smaller area and higher throughput than some of these HEVC ME hardware.

The HEVC ME hardware proposed in [13] and [14] use full search ME algorithm. The hardware proposed in [13] does not support the asymmetric partitions in HEVC ME.

Table 3 Impact of the proposed hardware on VVC coding efficiency

CTU size	128 × 128						64 × 64					
Search range	128 × 128		96 × 96		64 × 64		64 × 64		48 × 48		32 × 32	
	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)	BD-Rate (%)	BD-PSNR (dB)
Kimono	1.609	– 0.056	1.503	– 0.053	1.517	– 0.053	1.517	– 0.053	1.714	– 0.060	1.467	– 0.052
Tennis	3.476	– 0.102	3.612	– 0.105	3.550	– 0.105	3.550	– 0.105	3.653	– 0.107	3.361	– 0.098
Cactus	0.644	– 0.016	0.610	– 0.016	0.476	– 0.012	0.476	– 0.012	0.549	– 0.013	0.755	– 0.017
Park Scene	0.561	– 0.019	0.527	– 0.018	0.467	– 0.016	0.467	– 0.016	0.536	– 0.018	0.631	– 0.021

Table 4 Resource usage for 128 × 128 CTU size

	LUTs	Flip-Flops	BRAMs
Systolic Array	56,321	98,816	–
SAD Adder Tree	40,970	57,312	4
Control Unit	36,582	2806	–
Comparator	11,308	21,343	–
Memory	425	2050	24
Total	145,606	182,327	28

Table 5 Resource usage for 64 × 64 CTU size

	LUTs	Flip-Flops	BRAMs
Systolic Array	16,390	24,832	–
SAD Adder Tree	11,022	14,474	1
Control Unit	9954	1,348	–
Comparator	2818	4,948	–
Memory	210	1,026	12
Total	40,394	46,628	13

These HEVC ME hardware use more LUTs and have lower throughput than our VVC ME hardware.

A sequential and a parallel HEVC ME hardware implementing diamond search algorithm are proposed in [17]. Since the parallel hardware has higher performance than the sequential hardware, we compare our VVC ME hardware with the parallel HEVC ME hardware. The HEVC ME hardware has smaller area and lower throughput than our VVC ME hardware.

The HEVC ME hardware proposed in [18] uses full search ME algorithm. It uses more LUTs, less FFs, and has the same throughput as our VVC ME hardware. VVC supports a larger number of partitions compared to HEVC. For a 64 × 64 CU size, our VVC ME hardware finds 821 unique MVs for 1077 potential partitions, whereas the HEVC ME hardware in [18] finds 593 unique MVs for 677 potential partitions. Since our VVC ME hardware finds more MVs and highly pipelined, it uses more FFs than the HEVC ME hardware in [18]. Since we used area optimized synthesis strategy, our VVC ME hardware uses less LUTs than the HEVC ME hardware in [18].

The HEVC ME hardware proposed in [19] uses a fast hybrid pattern search algorithm. It has higher throughput and a much larger area than our VVC ME hardware.

In summary, the proposed VVC ME hardware is more efficient than HEVC ME hardware in the literature. The proposed VVC ME hardware and the HEVC ME hardware proposed in [13, 14, 18] use full search algorithm. The proposed VVC ME hardware can process larger (128 × 128 CTU size as opposed to 64 × 64 CTU size) and more complex coding block structure of VVC while achieving higher throughput and using smaller area than them. Since the HEVC ME hardware proposed in [17] and [19] use fast search algorithms, they may not find the best MVs. The proposed VVC ME hardware has higher throughput than the HEVC ME hardware proposed in [17]. Although the proposed VVC ME hardware processes much larger CTU size of 128 × 128 as opposed to CTU size of 32 × 32 processed by the HEVC ME hardware proposed in [19], it has much smaller area.

5 Conclusions

In this paper, we proposed the first VVC ME hardware in the literature. It supports up to 4309 CU partitions and computes 3285 unique motion vectors for a 128 × 128 CTU. It uses a novel memory-based SAD adder tree, and an efficient data reuse method. It is implemented on a Xilinx Virtex 7 FPGA. It can process up to 30 4K (3840 × 2160) video frames per second.

Table 6 Comparison with HEVC ME hardware

	[13]	[14]	[17]	[18]	[19]	Proposed
Standard	HEVC	HEVC	HEVC	HEVC	HEVC	VVC
FPGA	Xilinx Virtex 5	Xilinx Virtex 5	Xilinx Virtex 7	Xilinx Virtex 7	Xilinx Virtex 7	Xilinx Virtex 7
CTU Size	64 × 64	64 × 64	64 × 64	64 × 64	32 × 32	128 × 128
Search Range	64 × 64	64 × 64	144 × 144	64 × 64	64 × 64	64 × 64
Frequency (MHz)	125	84.96	198.73	247	162	253
Throughput	4 K @ 13 fps	4 K @ 9 fps	FHD @ 30 fps	4 K @ 30 fps	8 K @ 78 fps	4 K @ 30 fps
LUTs	209,434	153,314	49,258	188,664	485,760	145,606
Flip-Flops	199,066	36,368	13,351	144,302	607,200	182,327

Acknowledgements This research was supported in part by the Scientific and Technological Research Council of Turkey (TUBITAK) under the contract 118E134.

Author contributions WA: Conceptualization, investigation, hardware, validation, writing-original draft. HM: software, validation, writing-review & editing. IH: methodology, writing-review & editing, supervision.

Funding Open access funding provided by the Scientific and Technological Research Council of Türkiye (TÜBİTAK).

Declarations

Conflict of interest The authors declare no competing interests.

References

1. Cisco, V.: Cisco visual networking index: Forecast and trends, 2017–2022. In: Cisco Systems White Paper. 1(1), (2018)
2. ITU-T and ISO/IEC JTC: Versatile video coding. Recommendation ITU-T H.266 and ISO/IEC 23090–3 (2020)
3. Bross, B., Wang, Y., Ye, Y., Liu, S., Chen, J., Sullivan, G.J., Ohm, J.: Overview of the Versatile Video Coding (VVC) Standard and its Applications. IEEE Trans. Circ. Syst. Video Tech. **31**(10), 3736–3764 (2021). https://doi.org/10.1109/TCSVT.2021.3101953
4. Menasri, W., Skoudarli, A.: Performance comparison of throughput between AVC, HEVC and VVC hardware CABAC decoder. Journal of Real-Time Image Proc. **20**(2), 26 (2023). https://doi.org/10.1007/s11554-023-01266-y
5. Bross, B., Chen, J., Ohm, J.R., Sullivan, G.J., Wang, Y.K: Developments in international video coding standardization after AVC, with an overview of versatile video coding (VVC). In: Proceedings of the IEEE 109(9), 1463–1493 (2021). https://doi.org/10.1109/JPROC.2020.3043399
6. Park, S.H., Kang, J.W.: Fast affine motion estimation for versatile video coding (VVC) encoding. IEEE Access **7**, 158075–158084 (2019). https://doi.org/10.1109/ACCESS.2019.2950388
7. Pakdaman, F., Adelimanesh, M.A., Gabbouj, M., Hashemi, M.R.: Complexity Analysis of Next-Generation VVC Encoding And Decoding. In Proc. IEEE Int. Conf. on Image Proc. (ICIP), Abu Dhabi, UAE, (2020). https://doi.org/10.1109/ICIP40778.2020.9190983
8. Brandenburg, J., Wieckowski, A., Hinz, T., Henkel, A., George, V., Zupancic, I., Stoffers, C., Bross, B., Schwarz, H., Marpe, D.: Towards fast and efficient VVC encoding. In Proc. IEEE Int. Workshop on Multimedia Signal Proc. (MMSP), Tampere, Finland, (2020). https://doi.org/10.1109/MMSP48831.2020.9287093
9. Saldanha, M., Corrêa, M.C., Palomino, D., Porto, M., Zatt, B., Agostini, L.: An overview of dedicated hardware designs for state-of-the-art AV1 and H.266/VVC video codecs. In Proc. IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS), Glasgow, UK, (2020). https://doi.org/10.1109/ICECS49266.2020.9294862
10. Aksehir, Y., Erdayandi, K., Ozcan, T.Z., Hamzaoglu, I.: A low energy adaptive motion estimation hardware for H.264 multiview video coding. J. Real-Time Image Proc. **15**, 3–12 (2018). https://doi.org/10.1007/s11554-013-0383-9
11. Cerveira, A., Agostini, L., Zatt, B., Sampaio, F.: Memory profiling of H.266 versatile video coding standard. In Proc. IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS), Glasgow, UK, (2020). https://doi.org/10.1109/ICECS49266.2020.9294952
12. Kalaycioglu, C., Ulusel, O.C., Hamzaoglu, I.: Low power techniques for Motion Estimation hardware. In Proc. Int. Conf. on Field Programmable Logic and Applications, Prague (2009). https://doi.org/10.1109/FPL.2009.5272508
13. D'huys, T., Momcilovic, S., Pratas, F., Sousa, L.: Reconfigurable data flow engine for HEVC motion estimation. In Proc. IEEE Int. Conf. on Image Processing (ICIP), Paris, France, (2014). https://doi.org/10.1109/ICIP.2014.7025244
14. Vayalil, N.C., Kong, Y.: VLSI architecture of full search variable block size motion estimation for HEVC video. IET Circ. Devices Syst. **11**(6), 543–548 (2017). https://doi.org/10.1049/iet-cds.2016.0267
15. Ahmad, W., Ayrancioglu, B., Hamzaoglu, I.: Comparison of approximate circuits for H.264 and HEVC motion estimation. In Proc. Euromicro Conf. on Digital System Design (DSD), Kranj, Slovenia (2020), https://doi.org/10.1109/DSD51259.2020.00036
16. Pastuszak, G., Trochimiuk, M.: Algorithm and architecture design of the motion estimation for the H265/HEVC 4K-UHD encoder. J. Real-Time Image Proc. **12**, 517–529 (2016). https://doi.org/10.1007/s11554-015-0516-4
17. Khemiri, R., Kibeya, H., Loukil, H., Sayadi, F.E., Atri, M., Masmoudi, N.: Real-time motion estimation diamond search algorithm for the new high efficiency video coding on FPGA. Analog Integr. Circ. Sig. Proc. **94**, 259–276 (2018). https://doi.org/10.1007/s10470-017-1072-6
18. Alcocer, E., Gutierrez, R., Lopez-Granado, O., Malumbres, M.P.: Design and implementation of an efficient hardware integer motion estimator for an HEVC video encoder. J. Real-Time Image Proc. **16**, 547–557 (2019). https://doi.org/10.1007/s11554-016-0572-4
19. Gogoi, S., Peesapati, R.: A hybrid hardware oriented motion estimation algorithm for HEVC/H.265. J. Real-Time Image Proc. **18**, 953–966 (2021). https://doi.org/10.1007/s11554-020-01056-w
20. Fan, Y., Huang, L., Hao, B., Zeng, X.: A Hardware-Oriented IME Algorithm for HEVC and Its Hardware Implementation. IEEE Trans. Circ. Syst. Video Technol. **28**(8), 2048–2057 (2018). https://doi.org/10.1109/TCSVT.2017.2702194
21. Silveira, B., Paim, G., Abreu, B., Grellert, M., Diniz, C.M., Costa, E.A.C., Bampi, S.: Power-efficient sum of absolute differences hardware architecture using adder compressors for integer motion estimation design. IEEE Trans Circ. Syst. I Regul. Papers **64**(12), 3126–3137 (2017). https://doi.org/10.1109/TCSI.2017.2728802
22. Ahmad W., Hamzaoglu, I.: An efficient approximate sum of absolute differences hardware for FPGAs. In Proc. IEEE Int. Conf. on Consumer Electronics (ICCE), Las Vegas, (2021), https://doi.org/10.1109/ICCE50685.2021.9427756
23. Azgin, H., Kalali, E., Hamzaoglu, I.: An efficient FPGA implementation of versatile video coding intra prediction. In Proc.

Euromicro Conf. on Digital System Design (DSD), Kallithea, Greece, (2019) https://doi.org/10.1109/DSD.2019.00037
24. Azgin, H., Mert, A.C., Kalali, E., Hamzaoglu, I.: A reconfigurable fractional interpolation hardware for VVC motion compensation. In Proc. Euromicro Conf. on Digital System Design (DSD), Prague, Czech Republic, (2018) https://doi.org/10.1109/DSD.2018.00030
25. Mahdavi, H., Azgin, H., Hamzaoglu, I.: Approximate versatile video coding fractional interpolation filters and their hardware implementations. IEEE Trans. Consum. Electron. **67**(3), 186–194 (2021). https://doi.org/10.1109/TCE.2021.3107460
26. Ben Jdidia, S., Belghith, F., Masmoudi, N.: A high-performance two-dimensional transform architecture of variable block sizes for the VVC standard. J. Real-Time Image Proc. **19**, 1081–1090 (2022). https://doi.org/10.1007/s11554-022-01250-y
27. Fan, Y., Chen, J., Sun, H., Katto, J., Jing, M.: A fast QTMT partition decision strategy for VVC intra prediction. IEEE Access **8**, 107900–107911 (2020). https://doi.org/10.1109/ACCESS.2020.3000565
28. Huang, Y.W., An, J., Huang, H., Li, X., Hsiang, S.T., Zhang, K., Gao, H., Ma, J., Chubach, O.: Block partitioning structure in the VVC standard. IEEE Trans. Circ. Syst. Video Technol. **31**(10), 3818–3833 (2021). https://doi.org/10.1109/TCSVT.2021.3088134
29. Chien, W.J., Boyce, J., Chen, W., Chernyak, R., Choi, K., Hashimoto, R., Huang, Y.W., Jang, H., Liu, S., Luo, D.: JVET AHG report: Tool reporting procedure (AHG13). Document JVET-S0013, (2020)
30. Jung, S., Jun, D.: Context-based inter mode decision method for fast affine prediction in versatile video coding. Electronics **10**(11), 1243 (2021). https://doi.org/10.3390/electronics10111243
31. Ahmad, W.: Efficient HEVC and VVC motion estimation hardware. Ph.D. Dissertation, Faculty of Engineering and Natural Sciences, Sabanci University, Istanbul, Turkey, (2021)

Waqar Ahmad received PhD degree in Electronics Engineering from Sabanci University, Istanbul, Turkey in 2021. He is currently an Assistant Professor at Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Pakistan. His research interests include energy-efficient digital hardware design for video coding and approximate computing.

Hossein Mahdavi received B.S. degree in Electrical and Electronics Engineering from Azad University, Iran, in 2006, and M.S. degree in Digital Electronics Engineering from Shahid Beheshti University, Tehran, Iran, in 2017. He is currently a Ph.D. student in Electronics Engineering at Sabanci University, Istanbul, Turkey. His research interests include digital VLSI design for video processing and coding.

Ilker Hamzaoglu received B.S. and M.S. degrees in Computer Engineering from Bogazici University, Istanbul, Turkey in 1991 and 1993, respectively. He received Ph.D. degree in Computer Science from University of Illinois at Urbana-Champaign, IL, USA in 1999. He worked as Senior and Principle Staff Engineer at Motorola Inc., IL, USA between August 1999 and August 2003. He worked as Assistant and Associate Professor at Sabanci University, Istanbul, Turkey between September 2003 and January 2023. He is working as Professor at Ozyegin University, Istanbul, Turkey since February 2023. His research interests include digital hardware design for video processing and compression.