2023 Volume 1, Issue 2
Signal and Information Processing
ISSN: 2630-4805

Optimization of High Efficiency Video Coding Integer Motion Estimation Algorithm Based on Hardware Implementation

Yuxin Nie, Longzhao Shi*, Lin Huang
College of Physics and Information Engineering, Fuzhou University, Fuzhou 350108, Fujian, China

***Corresponding author:** Longzhao Shi, slz@fzu.edu.cn

Abstract:

In order to reduce the number of cycles of the motion estimation module and improve hardware efficiency, optimizes the small diamond search algorithm from the perspective of hardware implementation. First, the iteration and processing sequence of PU and CU were adjusted to solve the problem of pipeline stagnation in the process of processing. At the same time, the parallel computing method for small PU blocks can further improve the processing speed. This article first uses MATLAB to implement the search algorithm, uses Verilog language to describe the hardware circuit, the two versions use the same excitation file on the data, and compare the intermediate values of each module for functional verification. Through the test of multiple sets of sequences, the hardware circuit of this article needs an average consumption of 5,800 clk for AVMP and IME processing on a 64 px × 64 px CTU. The Arria10AX115N3F40E2SG development board is selected on the QuartusII platform, the main frequency can reach 186 MHz, and the overall performance of the whole pixel motion estimation module can reach 1080p @ 61 fs^{-1}.

Keywords:

High-efficiency video coding
Integer-pixel motion estimation
Video coding standard
Hardware implementation
Small diamond search algorithm

Online publication: December 29, 2023

1. Introduction

With the rise of short videos and the widespread adoption of 5G technology, there is a growing demand for ultra-high-definition videos. Consequently, the High Efficiency Video Coding standard (H.265/HEVC) has emerged to meet this need. Compared to its predecessor, H.264, H.265/HEVC adopts a similar hybrid coding framework, including modules such as transformation, quantization, entropy coding, intra-frame prediction, inter-frame prediction, and loop filtering [1]. However, it differs in its ability to perform adaptive quadtree partitioning of coding units and improves the accuracy of intra-frame prediction. The inter-frame mode also introduces advanced motion vector prediction technology (AMVP) [2]. As a result, HEVC achieves a bitstream size that is only half of H.264 for the same video quality. Inter-frame prediction

accounts for approximately 70% of the entire encoding time in the HEVC encoding process, making optimization of inter-frame prediction crucial for improving encoder performance [3].

Currently, common search algorithms for HEVC encoders include Full-search, TZ-Search, Two-dimensional logarithmic search [4], Three-step search [5], and Four-step search [6]. Different algorithms strike a balance between speed and performance. Shen *et al.* (2013) reported that, a fast CU size decision algorithm is proposed, which skips motion estimation for unnecessary CU sizes through three early termination methods based on motion uniformity, rate-distortion cost (RD-COST), and SKIP mode [7]. However, it is less efficient for encoding images with vigorous motion.

Many researchers have also proposed hardware-friendly motion estimation algorithms. Fan *et al.* (2018) presents an improved diamond search algorithm that performs fine selection in the central region and coarse selection in the peripheral region, while using a SAD tree to record the cost data of all PUs simultaneously [8]. However, it incurs a higher hardware resource overhead. Estefania *et al.* (2019) conducts an S-shaped full search on a 128px × 128px area, using shift registers to store reference pixels [9]. It discards one column of pixels when searching left or right, enabling reference pixel switching within one clock cycle. However, this approach results in a longer overall clock cycle for the module. Cheng *et al.* (2016) merges the CTU depths of the frame with the minimum QP value in the GOP and the same position in the previous frame to determine the maximum depth for current CTU calculation [10]. This reduces the CTU depth decision path from four depths to three, effectively reducing computational complexity, but the performance loss is not ideal. Li *et al.* (2014) proposes a fast algorithm with a variable search range, which estimates the search range of sub-blocks (PUs) using empirical formulas based on the MV of the LCU [11]. This effectively reduces the number of search points during motion estimation but may not perform well for certain specific sequences.

This study adjusts the algorithm based on the small diamond search algorithm within a hardware implementation framework. The study employs pipelining and parallel processing techniques to pipeline the AMVP and IME modules, and adjust the processing order of PUs and CUs. This addresses stalls in the pipeline caused by the mutual referencing of AMVP and IME. The hardware circuit processes small-size PUs in parallel, further reducing the overall module cycle count.

2. Brief description of the algorithm

2.1. Small diamond search algorithm

In a frame, the motion vector (MV) of each PU is closely related to its adjacent PUs. Therefore, HEVC introduces the AMVP algorithm to determine the search starting point and speed up the search process. The search process of the small diamond search algorithm is shown in **Figure 1**, where the numbers represent the search order, and the search starting point is determined by the result of AMVP. Each search requires calculating the SAD (Sum of Absolute Differences) cost corresponding to a total of 5 points, including the center point and its upper, lower, left, and right points. After a search, cost comparison is performed to determine whether iteration is needed or if the search for the next PU can be started. The criterion for judgment is whether the point with the minimum cost is the center point. If not, iterative searching is performed with the point of minimum cost as the center point until the point with the minimum cost is the center point or the maximum number of iterations is reached (the maximum number of iterations in this study is set to 64). Compared with the TZ-Search algorithm in the video coding standard code HM16.7, the small diamond search algorithm only increases the BD rate by 0.5% on average while reducing the number of search points by 72.9% on average [12]. Therefore, the small diamond search algorithm can significantly reduce the amount of search data and improve hardware processing speed.

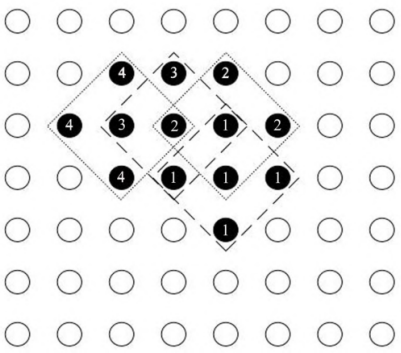

Figure 1. Small diamond search order.

On the HM16.7 platform, the proportion of asymmetric mode partitions for CUs is only 6.43% [13]. Performance testing was conducted on HM16.7 with the asymmetric partition mode for CUs disabled, and the results are shown in **Table 1**. As can be seen from the table, disabling the asymmetric mode reduces the overall encoding time by an average of 13.2% and increases the BD rate by an average of 0.687%. Not all PUs in HM execute the asymmetric mode; only CUs with sizes of 32px × 32px and 16px × 16px activate the asymmetric mode [15]. Different partition methods correspond to different asymmetric modes. PUs of 2N px × N px only undergo two types of asymmetric partitions: 2N px × nD px and 2N px × nU px, while PUs of N px × 2N px only undergo nL px × 2N px and nR px × 2N px asymmetric partitions. PUs of 2N px × 2N px require all four asymmetric partitions mentioned above. Therefore, disabling the asymmetric mode only reduces complexity by 13.2%. However, for hardware implementation, as long as there are PUs that require asymmetric processing, corresponding hardware circuits need to be prepared. Since most PUs do not require asymmetric mode, this results in low utilization of hardware circuits and wastage of hardware resources. Additionally, disabling the asymmetric mode reduces the number of PUs that a CU needs to process from 13 to 5. For a CTU with a size of 64px × 64px, it can be divided into 85 CUs through quadtree partitioning. Thus, processing a CTU reduces the number of PUs traversed from 1,105 to 425, effectively reducing processing time with minimal performance loss. In summary, the motion estimation module only performs symmetric partitioning for CUs.

Chen *et al.* (2018) present a hardware implementation based on the small diamond search algorithm [12]. The IME process for PUs adopts serial computation, where the iteration judgment result of the current PU determines whether the next processing is for a new PU block or an iteration of the current PU block. This processing method is logically clear and simple to implement. However, due to waiting for iteration judgment results, the data processing within each hardware module is intermittent, leading to inefficient hardware operation. Additionally, Chen *et al.* (2018) adopts a layer-by-layer search for CUs, with each layer executed sequentially in a Z-scan manner [12]. The five PUs within a CU are also executed sequentially. Since adjacent CUs refer to each other's IME results, this can also cause stalls in hardware processing.

For high frame rate and high-definition videos, it is not feasible to adopt the hardware processing architecture from Chen *et al.* (2018) to achieve real-time encoding [12]. In the study of Chen *et al.* (2018), the processed CTU size is 16px × 16px, which includes a total of 25 PUs, resulting in a relatively small overall data volume [12]. This study aims to process CTUs with a size of 64px × 64px, containing 425 PUs. Therefore, adopting the aforementioned method cannot achieve the desired throughput target. Based on the small diamond search algorithm, this study optimizes the processing methods for PUs and CUs, adjusts the processing order of CUs and PUs, and effectively improves hardware processing efficiency.

Table 1. Performance comparison after turning off AMP mode (%)

Sequence	ΔBR	Δ*t*
BlowingBubbles (384 px ×192 px)	0.689	-13.79
Foreman (320 px × 256 px)	1.602	-12.88
BasketballDrill (832 px × 448 px)	0.291	-11.02
RaceHorsesCtype (832 px × 448 px)	0.524	-15.85
Kimono1 (1920 px × 1024 px)	0.329	-12.48
Average value	0.687	-13.20

Note: ΔBR represents the BD change rate; Δt represents the encoding time change rate.

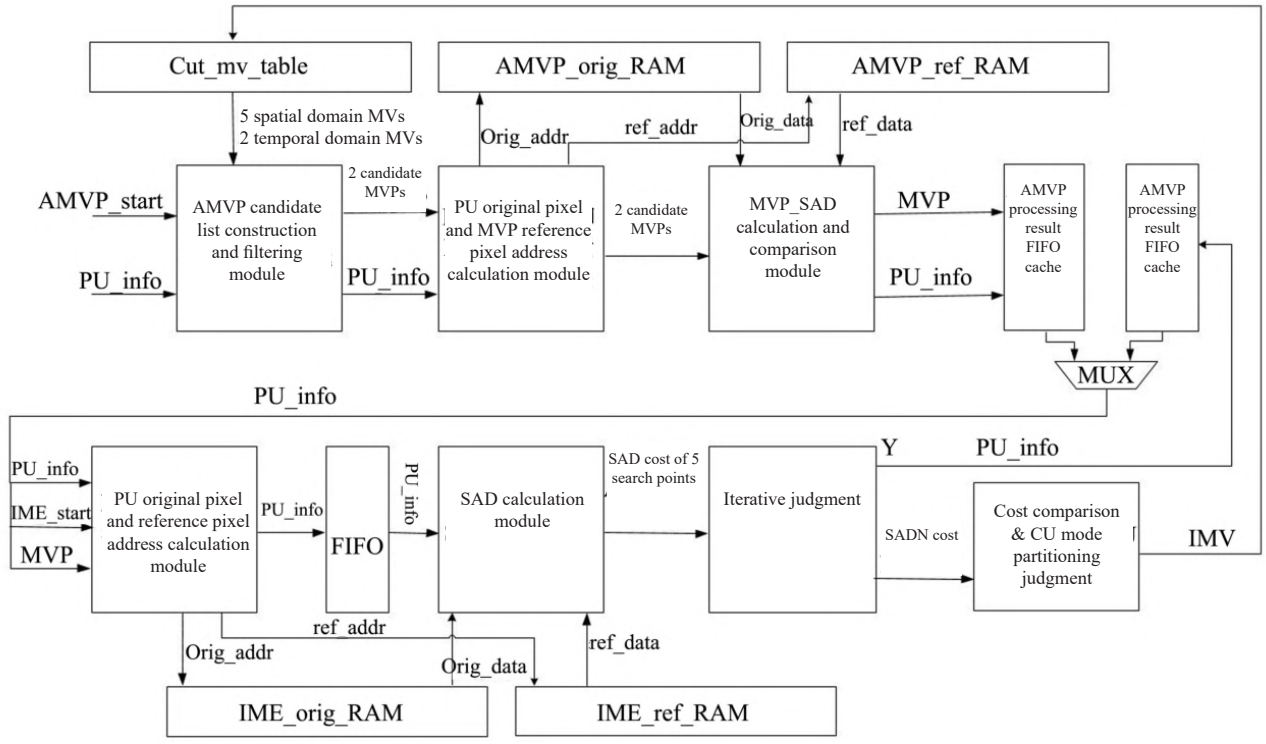

Figure 2. Hardware architecture diagram.

2.2. Hardware architecture design based on small diamond search

The hardware framework structure proposed in this study is shown in **Figure 2**. The processing of a PU is mainly divided into two major processes: AMVP processing and IME search.

The sentence you provided is quite technical and detailed, focusing on the processing order and optimization of computational units (PU) and coding units (CU) in video encoding. Here's the translation:

"FIFO is used for data caching between two processes to achieve parallel computing. The AMVP process mainly includes the establishment of a candidate list and a cost calculation comparison module. The IME process primarily consists of calculating the addresses of original pixels and reference pixels corresponding to the PU, the SAD cost calculation module, and the iterative judgment module."

2.3. Processing order of PU

As shown in **Figure 2**, the AMVP process is performed first for a PU, followed by the IME search using the results of AMVP. To efficiently process a PU, the AMVP and IME processes are modified to pipelined processing, and the processing order of the PU is adjusted. For a CU containing five PUs under three partitioning methods (as shown in **Figure 3**), the AMVP processing of the PU block with serial number 3 depends on the IME result of the PU block with serial number 2 (similarly for PU blocks with serial numbers 4 and 5). Therefore, the processing order of the PU blocks is adjusted as follows:

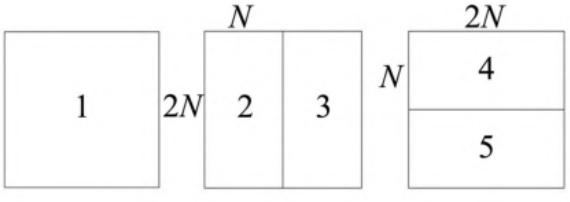

Figure 3. PU block.

The five PUs under the CU are divided into three groups with no mutual reference relationship in MV: group 1 contains block 1, group 2 contains blocks 2 and 4, and group 3 contains blocks 3 and 5. The processing order of the PU blocks is adjusted to 2, 4, 1, 3, 5. Since there is no reference relationship between the AMVP of blocks 2 and 4, the AMVP processing of block 4 can

be performed simultaneously with the IME processing of block 2. Similarly, the proposed PU processing order inserts two unrelated blocks between two blocks with reference relationships (e.g., blocks 2 and 3), effectively solving the issue of stalled AMVP processing due to waiting for IME results and reducing the overall clock cycles of the module.

2.4. Processing order of CU

The CU processing order in Chen *et al.* (2018) is hierarchical, with each layer processed in a Z-scan order [12]. This approach results in the next CU having to wait for the previous CU to finish before starting. The post-processing of a CU includes cost comparison between different partitioning methods and updating the IME to the CTU_MV_TABLE_RAM, which can cause stalls in the IME processing module during CU switching. Therefore, a new CU processing order is proposed, as shown in **Figure 4**, utilizing the rule that MVs between different layers of CUs do not reference each other. The numbers in **Figure 4** represent the processing order of the CUs, with CUs from different layers or non-adjacent positions inserted between the processing of two adjacent CUs. This approach ensures smooth processing between adjacent CUs. From subsequent simulation waveforms, it can be seen that the proposed PU and CU processing orders enable the SAD module in IME to achieve a fully pipelined state.

2.5. Alternating calculation and pipelined processing of IME module for PU

IME processing involves three main steps: (1) calculating the addresses of corresponding original pixels and reference pixels based on PU information; (2) calculating the SAD cost after cropping the row-input original pixels and reference pixels; (3) comparing the costs of five search points corresponding to the PU during a search process and determining whether to end the current IME process or perform iterative searching.

Due to the existence of an iterative mechanism, it is not possible to determine whether the next process to be handled is an iterative search of the current PU or a search of the next PU until the iteration process is complete. Therefore, serial processing of PUs can lead to increased processing time and intermittent data flow within various sub-modules (as shown in **Figure 5**).

To address this issue, the PU processing approach is modified to a three-stage pipelined process. To reduce clock cycle waste caused by judging whether to iterate and ensure continuous data calculation within the module,

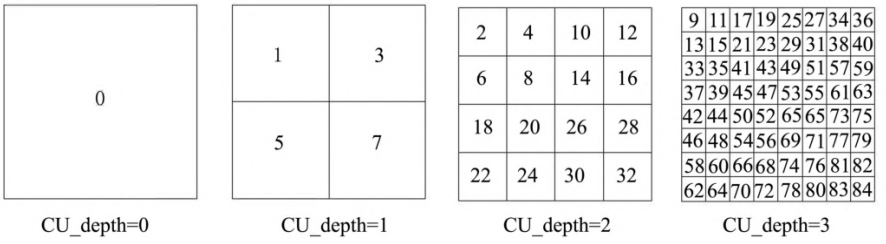

Figure 4. Processing order of CU in CTU.

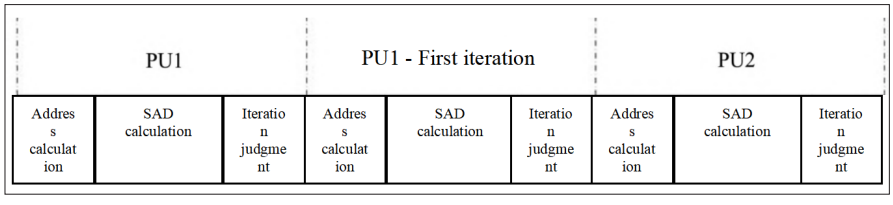

Figure 5. Traditional serial processing method for PU.

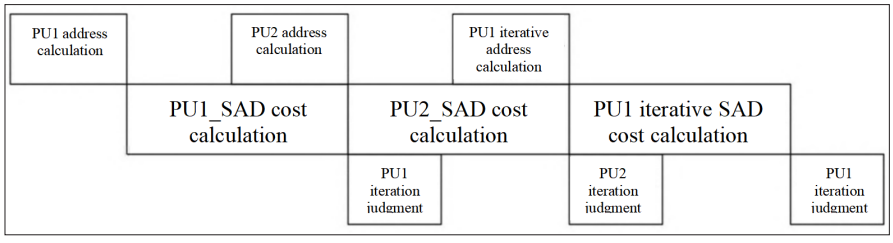

Figure 6. IME module pipeline processing.

the calculation of the next PU is inserted between two iterations (as shown in **Figure 6**). During the SAD calculation of PU1, the address calculation for PU2 is performed simultaneously. When the iterative processing of PU1 ends, if iteration is required, the information of PU1 is stored in the iteration FIFO. Thus, the IME has two data sources: one is the FIFO storing the next PU's information, and the other is the FIFO storing the iterative PU's information. The iterative FIFO has a higher priority. Before each IME starts, the data storage status of the two FIFOs is checked to determine which FIFO to retrieve data from."

2.6. IME_SAD cost calculation

After performing IME processing on the five PUs contained in the CU, the costs of three different partitions are compared to determine the final partition method for the CU. The formula for calculating the cost is J = D + λR + cost. Where: The residual D is represented by SAD during the IME process; λ is the Lagrangian factor, which is related to the external configuration parameter QP value and is obtained by looking up a table in the hardware of this study; R is the number of encoding bits for the difference between the current MV and MVP (MVD); cost represents the cost.

The formula for calculating SAD is $SAD = \sum_{i=1}^{M} \sum_{j=1}^{N} |orij(i,j) - ref(i,j)|$. Since the CU is divided into two PUs, the amount of information that needs to be encoded increases. Therefore, an additional header bit cost related to the partition method is added. The calculation method is $cost = \sqrt{\lambda(\text{pu\_num} + 4)}$.Here, pu_num represents the number of PUs contained after CU partitioning.

The CTU (Coding Tree Unit) of 64px × 64px contains a total of 425 PUs (Prediction Units), among which there are only 3 PUs with a width of 64px. To avoid resource wastage caused by incomplete utilization of logical resources when processing small PUs, it is chosen to simultaneously calculate 32-pixel operators per clock cycle. For PU blocks with a width of 64px, cost calculation is performed twice. Therefore, in the AMVP (Advanced Motion Vector Prediction) module, PU information and MVP (Motion Vector Predictor) for 64px width are stored twice consecutively in the FIFO (First In, First Out) buffers of AMVP and IME (Integer Motion Estimation). However, for PUs of 16px × 16px and 8px × 8px, more than half of the processing units will be idle. To improve the utilization of hardware resources, the 32-point operators are divided into two independent operators. Each operator can process PU blocks with a width of 16px or can be combined to process 32-point PUs. Thus, for CUs (Coding Units) with widths of 16px and 8px, PU blocks of 2N px × 2N px and 2N px × N px above can be processed simultaneously.

2.7. IME storage unit design

To achieve parallelism in computing units, the primary task is to ensure that the storage unit can provide the required data on time. Firstly, let's explain the requirements of the storage unit. The original pixel RAM temporarily stores the original pixel data of a CTU in rows, with a width of (64 × 8) bits and a depth of 64. The search box in this study is set to extend 32 pixels around the CTU, and considering sub-pixel interpolation requires an additional extension of 4 pixels, the adopted search box size is 136px × 136px. The reference pixel RAM has a width of (136 × 8) bits and a depth of 136.

The IME address calculation module calculates the coordinates of the current processing PU block in the CTU and search box. Based on the y-coordinate, it outputs the address to the reference pixel and original pixel RAMs. The reference pixels and original pixels input to the SAD are 136px and 64px respectively, filtered by the x-coordinate value for the pixels input in rows. Then, the two are subtracted, and an absolute value calculation is performed. The appropriate SAD value is selected based on the PU width and added to obtain the final SAD value for a PU block. To improve the processing speed of a PU block, the hardware simultaneously calculates the SAD costs for 5 search points. Since the positional relationship between the 5 search points is fixed, the reference pixels and original pixels can be read once to calculate the costs for all 5 points, as shown in **Figure 7**. The black dots in the figure represent the 5 search points that need to be searched simultaneously, and the solid and dashed boxes indicate the PU blocks corresponding to each search point. The top-left pixel coordinate of the search center

is used as the starting address. In the first clock cycle, the absolute sum of the differences between the first row of reference pixels and original pixels gives the SAD cost for the top search point's first row. In the second clock cycle, the absolute sum of the differences between the second row of reference pixels and the delayed first row of original pixels yields the SAD values for the first row of the three middle search points, which only require different filtering of the reference pixels. Similarly, the absolute sum of differences between the second row of reference pixels and original pixels gives the SAD cost for the second row of the top search point. Following this calculation method, the costs for the last row of the 5 search points are obtained with only a two-clock cycle difference, and the reference pixels and original pixels are read only once, improving data interaction efficiency.

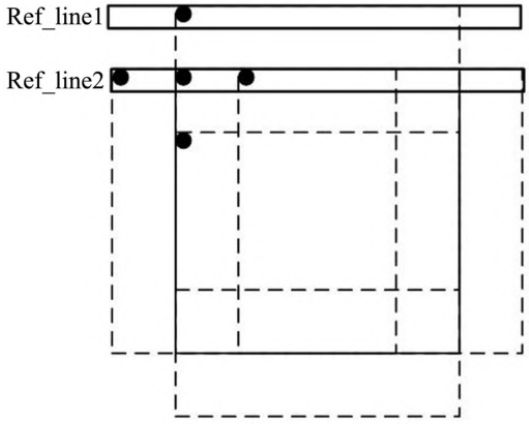

Figure 7. Schematic diagram of parallel SAD (Sum of Absolute Differences) cost calculation for 5 search points in IME.

2.8. Five-stage pipeline processing of AMVP and IME

According to the previous arrangement, the SAD calculation process of the IME module can achieve continuous full pipelining. Therefore, the AMVP module also needs to achieve this rate to match the processing speed of the IME module. To this end, the AMVP module should prepare the data as early as possible and store it in the FIFO for the IME module to read. The pipeline structure is shown in Fig 8. Each PU's processing needs to go through five stages, where AMVP is divided into two stages: candidate list establishment and cost comparison. IME is divided into three stages: IME address calculation, SAD cost calculation, cost comparison, and iterative judgment. Overall, the processing time of AMVP is less than that of IME. Therefore, this study redesigned the processing order of PUs so that there is no MV reference relationship between two adjacent processing PUs. The AMVP module can calculate the next PU without waiting for the results of IME and then store information such as MVP in the FIFO. After the IME module finishes, it updates the obtained IMV to the CTU_MV_TABLE and judges whether the cache FIFO is empty. If there is data in the FIFO, the IME processing of the next PU can be self-started. FIFOs are also used for data caching between each stage of pipelining within AMVP and IME.

3. Comprehensive results of hardware circuits

The hardware framework was designed using the Verilog language. Under the Linux system, the RTL code was simulated and compiled using Synopsys VCS software, and timing adjustments were made to the waveforms using Verdi software. Firstly, MATLAB was used to print out the excitation data required by the hardware, including reference pixels, original pixels, and reference MVs from the upper and left CTUs. Testbench was utilized to store the excitation data from text files in RAM and then send a start signal. The correctness of the circuit was verified by comparing the IME cost values and CU partitioning methods in the hardware circuit with the data

Figure 8. Schematic diagram of the processing pipeline for AMVP and IME.

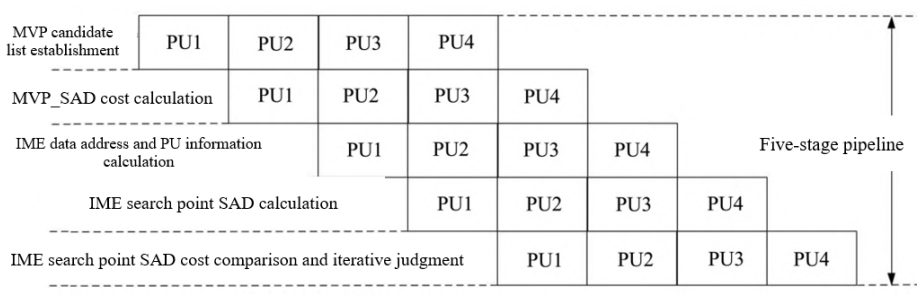

Table 2. Resources compared with this study and other articles

Architecture	$n_{Flip\text{-}Flops}$ / each	n_{LUTs} / each	Memory / kB	n_{Cycles} / each	$f_{Main\ frequency}$ / MHz	Throughput / fs^{-1}
This study	11.5×10^3	59.0×10^3	27.5	5 800	186	1080p@61
Estefania *et al.* (2019)	140.9×10^3	184.2×10^3	36.0	16 462	247	1080p@32
Ye *et al.* (2014)	246.0×10^3	44.0×10^3	373.5	3 186	200	1080p@120
Gogoi *et al.* (2021)	132.9×10^3	80.9×10^3	8.32	2724	353	4K@60

from MATLAB. Afterward, using QuartusII software and selecting the Arria10AX115N3F40E2SG development board, the module resource test results are detailed in **Table 2**.

Estefania *et al.* (2019) reported a full search method was adopted for the search area, using shift registers to store data [9]. When the search point moves up and down, a row of pixels is discarded, and when it moves left and right, each shift register moves one bit. This allows for reference pixel switching in just one clock cycle. However, the data repetition rate between two points in the snake-shaped full search is high. Nevertheless, full search has high computational complexity, requiring a massive amount of data processing, leading to significant hardware resource overhead and a large number of processing cycles. Ye *et al.* (2014) employed a layered search combined with parallel processing for searching [14]. Parallel computing greatly improved processing speed but incurred significant hardware resource overhead. Gogoi *et al.* (2021) first performed a rough selection, and the step size obtained through this rough selection determined the second search method, significantly reducing the number of search points but requiring more storage and hardware resources [16]. As can be seen from **Table 2**, the main frequency of this study is slightly lower than that of Estefania *et al.* (2019) and Ye *et al.* (2014), but the hardware resource overhead is significantly smaller [9,14]. This study achieves a higher cost-effectiveness in terms of throughput and hardware resource expenditure compared to the three papers mentioned.

The number of cycles in the table represents the clock cycles consumed for processing one CTU. In this study, the average number of cycles is 5,800 clock cycles. To visually demonstrate the effectiveness of the proposed method, we also tested the average number of cycles consumed for processing a 64px × 64px CTU without changing the processing order of CU and PU, which resulted in 9,918 clock cycles. This is because not changing the PU/CU processing order leads to an MV reference relationship between two adjacently processed PU/CU blocks, requiring the IME calculation of the previous PU block to finish before performing AMVP on the current PU block and waiting for the processing results of the AMVP module. Therefore, compared to the normal sequential approach, the design scheme in this study reduces the average number of clock cycles by 41.5% for processing one CTU.

4. Conclusion

In video coding, inter-frame motion estimation accounts for most of the encoding time. To further improve the processing efficiency of the encoder, optimization of motion estimation is necessary in terms of search methods or hardware implementation. The point with the minimum cost in the search box must exist. The ultimate goal of researchers is to quickly locate it using search algorithms and efficiently calculate the cost of each point through hardware design. In this study, the small diamond search algorithm was optimized on the hardware architecture, and the processing order of CU and PU was redesigned. By adopting alternating PU processing and pipelining, the IME module was able to achieve full pipelining. Additionally, parallel processing of small-sized PUs further reduced the cycle count of the module. Through synthesis and compilation of RTL code using the QuartusII platform, the proposed hardware architecture consumed an average of 5,800 clock cycles for processing a 64px × 64px CTU, with a main frequency of up to 186 MHz and comprehensive performance reaching 1080P@61 fs^{-1}.

Funding

2020 Science and Technology Innovation Team Funding Project of Fujian Higher Education Institutions

Disclosure statement

The authors declare no conflict of interest.

References

[1] Sullivan GJ, Ohm JR, Han WJ, et al., 2013, Overview of the High Efficiency Video Coding (HEVC) Standard. IEEE Transactions on Circuits and Systems for Video Technology, 22(12): 1649–1668.

[2] Wan S, Yang F, 2014, New Generation High-Efficiency Video Coding H.265/HEVC: Principles, Standards, and Implementation. Publishing House of Electronics Industry, Beijing.

[3] Song B, Chang Y, Zhou N, 2006, A Fast Algorithm Based on H.264 Inter-frame Prediction. Acta Electronica Sinica, 34(1): 31–34.

[4] Jain JR, Jain AK, 1981, Displacement Measurement and Its Application in Interframe Image Coding. IEEE Transactions on Communications, 29(12): 1799–1808.

[5] Koga T, 1981, Motion Compensated Interframe Coding for Video Conferencing. IEEE Proceedings of the National Telecommunication Conference, New Orleans, 1981: 531–535.

[6] Po LM, Ma WC, 1996, A Novel Four-Step Search Algorithm for Fast Block Motion Estimation. IEEE Transactions on Circuits and Systems for Video Technology, 6(3): 313–317.

[7] Shen LQ, Liu Z, Zhang XP, et al., 2013, An Effective CU Size Decision Method for HEVC Encoders. IEEE Transactions on Multimedia, 15(2): 465–470.

[8] Fan YB, Huang LL, Hao B, et al., 2018, A Hardware-Oriented IME Algorithm for HEVC and Its Hardware Implementation. IEEE Transactions on Circuits and Systems for Video Technology, 28(8): 2048–2057.

[9] Estefania A, Roberto G, Otoniel ML G, et al., 2019, Design and Implementation of an Efficient Hardware Integer Motion Estimator for an HEVC Video Encoder. Journal of Real-Time Image Processing, 16(2): 547–557.

[10] Cheng X, Liu ZY, Tetsunori K, et al., 2016, Multi-Feature Based Fast Depth Decision in HEVC Inter Prediction for VLSI Implementation. The 9th International Congress on Image and Signal Processing, Biomedical Engineering, and Informatics, Datong, IEEE, 2016: 124–128.

[11] Li GL, Wang CC, Chiang KH, 2014, An Efficient Motion Vector Prediction Method for Avoiding AMVP Data Dependency for HEVC. IEEE International Conference on Acoustics, Speech, and Signal Processing, Florence, IEEE, 2014: 7363–7366.

[12] Chen Q, Shi L, Huang B, et al., 2018, A Fast Algorithm and Hardware Architecture for Motion Estimation in HEVC. Journal of Fuzhou University (Natural Science Edition), 46(5): 636–643.

[13] Gao X, 2018, Research on Fast Algorithm and Hardware Implementation of Inter-Prediction Mode Selection in HEVC, thesis, Fuzhou University.

[14] Ye X, Ding DD, Yu L, 2014, A Hardware-Oriented IME Algorithm and Its Implementation for HEVC. IEEE Visual Communications and Image Processing Conference, Valletta, IEEE, 2014: 205–208.

[15] Zhang Y, Li Q, 2018, A Fast Decision Algorithm for HEVC Inter-frame Mode Based on Motion Characteristics. Computer Engineering and Applications, 54(23): 195–202.

[16] Gogoi S, Peesapati R, 2021, Design and Implementation of an Efficient Multi-Pattern Motion Estimation Search Algorithm

for HEVC/H.265. IEEE Transactions on Consumer Electronics, 67(4): 319–328.