Proceedings of the 4th International Conference on Signal Processing and Machine Learning
DOI: 10.54254/2755-2721/49/20241073

Hardware motion estimation methods survey and example real-time architecture design

Wentian Zhong

College of Electronic Engineering (College of Artificial intelligence), South China Agricultural University, Wushan Street five road No. 483, China

c148490101@163.com

Abstract. Motion estimation is essential in video processing, computer vision, and other related fields due to its applications in improving data compression, enhancing image/video quality, increasing computer vision task accuracy, and conserving computational resources. Typical motion estimation methods include FS (Full Search) and DS (Diamond Search). Traditional motion estimation methods will face the challenge of high complexity in practical real-time applications. Therefore, this paper focuses on fast motion estimation algorithms and hardware architecture design for real-time applications survey and research. Aiming at the need for 1920x1080@60fps processing speed, this paper explores the key ideas of FS and DS hardware structure design based on the Vivado HLS high-level synthesis tool and provides a feasible hardware scheme to meet the real-time requirements.

Keywords: Motion Estimation, Real-time, Fast Algorithm, Hardware Design, Vivado HLS.

1. Introduction

The frames of video images are highly correlated and similar to each other. Objects will move between video images. Through motion estimation (ME) technology, the direction and degree of movement of regions or objects can be determined. In the motion estimation, the image sequence is divided into several blocks, and the position of the block in the reference frame is searched. The relative displacement between the position of the block in the reference frame and the current block is regarded as the motion vector. Motion estimation is widely used in video coding, video processing, and temporal filtering.

During motion estimation, it is critical to find the best search point with the minimum difference in a certain search range by an effective search algorithm. Typical motion estimation algorithms include full search, diamond search, hexagonal search, and three-step search. FS finds all points in a certain range, takes the search center as the starting point, and calculates the Sum of Absolute Difference (SAD) of each point, which searches accurately but takes a long time. Diamond search takes the search starting point as the center and searches the center point and the four points around it to calculate their SAD. DS is the fastest of the four methods. Three templates are used for the hexagonal search, which are a hexagonal template, a small diamond template, and a small square template. Three-step search compares the search center with eight points around it and selects the smallest SAD as the center for the next search. This method is more accurate and faster than FS.

Typical optimization methods of motion estimation algorithms can be achieved towards faster search or more accurate search. A fast motion estimation algorithm based on different size prediction modes

DOI: 10.54254/2755-2721/49/20241073

was proposed, which reduced the search time by 20% [1]. Jun Luo et al. proposed a fast algorithm based on the adaptive pattern and search priority, which reduced the computational complexity and improved motion estimation search performance [2]. An efficient hierarchical diamond search was proposed, which reduced the complexity. What's more, the quality of the algorithm was equivalent to that of FS [3]. Pakdaman F et al. optimized the Test Zone (TZ) search and reduced the search time by 35%-85% compared with the traditional TZ search [4]. An optimized DS algorithm was proposed, which can save 10.81 search points [5].

In addition to optimizing the motion estimation algorithms with software for faster implementation, in practice, there are many hardware real-time implementation motion estimation requirements. It requires efficient motion estimation design methods along with hardware architecture designs, which are different a lot from the above software-oriented fast motion estimation methods. A method combining algorithm and architecture optimization was proposed to reduce the search points by 90.5% and the interpolation filtering by 62.4% [6]. Seda Yavuz utilized the architecture of simultaneous binarization and matching to reduce the complexity of motion estimation [7]. An efficient integer motion estimation hardware approach was proposed with a low latency reduction of 80% [8]. Eianca Silveira adopted a structure for the Sum of Absolute Transform Differences (SATD), which consumes 50.85pJ for a single SATD [9].

This paper is to study the typical motion estimation optimization methods in recent years, and deeply analyze the hardware architecture design of each typical method. What's more, we compare the hardware performance of several classical motion methods through experimental analysis by a high-level synthesis method, to provide a basis for selecting the appropriate hardware motion estimation method.

The remaining organizations of this paper are composed of four parts. Section II will introduce VIVADO HLS, which is a popular software tool to simulate hardware designs. Besides these, several typical hardware motion methods are presented in detail. Section III shows the hardware structure designs of several classical motion estimations. Section IV gives the experimental results, and the conclusion is given in Section V.

2. Related Work

2.1. Motion Estimation

Typically, motion estimation can be implemented either pixel-based or block-based. Considering the high complexity of pixel-based motion estimation, block-based motion estimation is commonly used at present.

The key elements of block-level motion estimation methods include search range, search method, and matching criterion. As shown in figure 1, N×N blocks are searched with (x, y) as the starting point in the figure, and the search range is the dashed line range in the reference frame. The dashed box is searched in the search range, and the motion vector is (u, v).

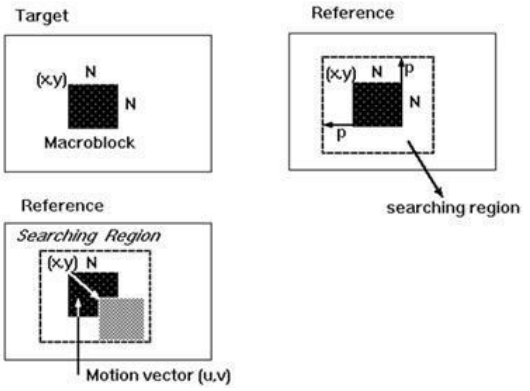

Figure 1. Motion estimation search.

DOI: 10.54254/2755-2721/49/20241073

To obtain accurate motion estimation results, efficient search methods, including FS and DS, are needed. In these search methods, appropriate matching criteria are needed to determine whether the optimal search block is obtained. For example, SD, SAD, SATD. Among them, SAD is often used for motion estimation search, as shown in equation (1),

$$\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\left|\begin{matrix}Orgvalue(x,y)\\-Predvalue(x,y)\end{matrix}\right| \tag{1}$$

where $Orgvalue$ is the original frame and $Predvalue$ is the predicted frame.

Traditional search methods including the FS and DS with high complexity, which is a challenge for real-time applications. Therefore, it is necessary to study efficient fast-motion estimation methods and architectures for hardware designs.

2.2. Typical Optimal Hardware Motion Estimation Methods

An efficient block matching algorithm and a three-level memory organization were proposed [10]. A large search range does not necessarily improve the coding performance, while a small search range can reduce the search complexity without significantly reducing the search accuracy. Based on this fact, the author proposed the block matching algorithm and introduced SADcenter into this method.

On the one hand, the search center is predicted and the SADcenter is the SAD calculated from the current block and the search block. Tn is the dynamic threshold, if SADcenter is smaller than Tn, the search center coordinates are output as motion vector (MV). If SADcenter is greater than Tn, two parallel searches, Limited-search and Hierarchy-search, are performed. Hierarchy-search includes Coarse-search and Refining-search. Coarse-search performs 4×4 block matching within a search range of [-32,+32), and the lowest SAD cost and second lowest SAD cost are considered candidates. In Refining-search, all variable block sizes are available, and variable block size full search is performed on candidates. The flowchart of the algorithm is shown in figure 2.

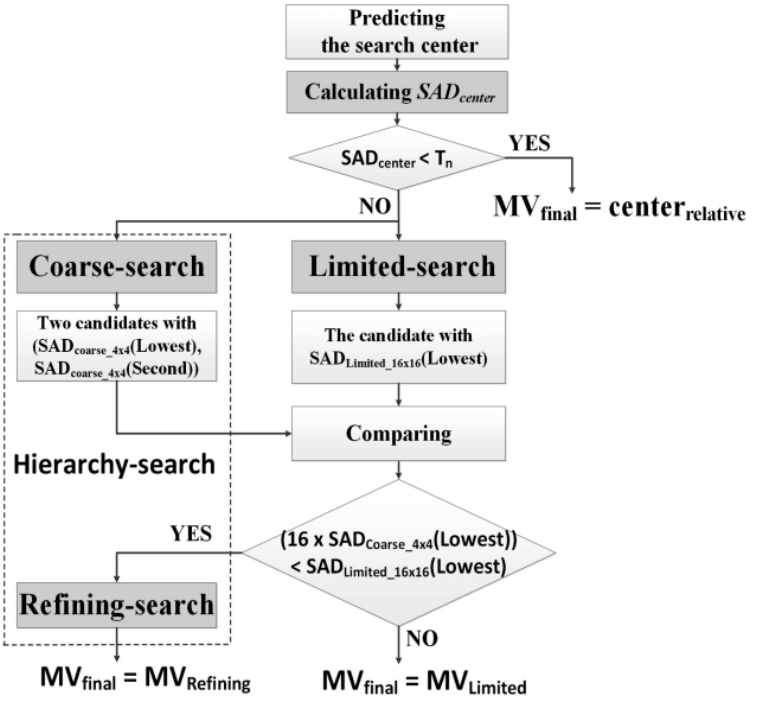

Figure 2. Flow diagram of the proposed ME algorithm.

DOI: 10.54254/2755-2721/49/20241073

On the other hand, the external memory is used to store the reference frame pixels, and the on-chip SRAM is used to load the sampled data of the common search window. Figure 3 is the three-level memory organization, which consists of external memory, on-chip SRAM and reg array.

Experimental results show that the hardware efficiency is improved by 1.8 to 3.7 times, and the on-chip storage space is reduced by 96.5%.

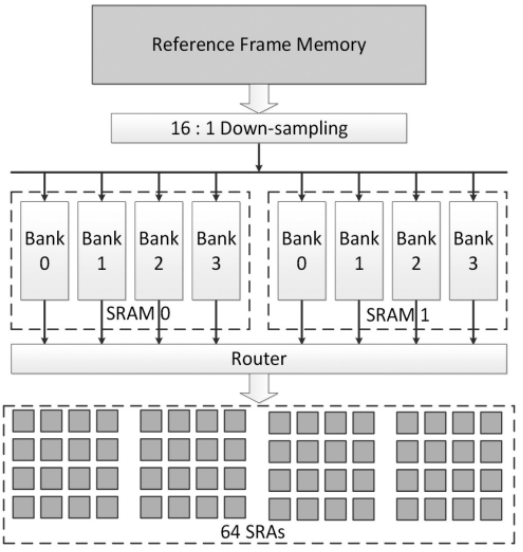

Figure 3. Three-level memory organization for the coarse-search.

K.R. Sarath Chandran and Premanand Venkatesh Chandramani proposed an architecture to accelerate SAD [11]. Besides, they also reduced power consumption and improved resource utilization. In SAD, macroblocks of 4×4 are used in the paper and videos are divided into three types according to the motion level (lm), as shown in equation (2),

$$lm = \frac{1}{A}\sum_{i=0}^{251}\sum_{j=0}^{187}\left|Y1_{ij} - Y2_{ij}\right| \tag{2}$$

where i, j are the position of the block within the search range, A is the total number of pixels in the subframe, and $Y1_{ij}$, $Y2_{ij}$ are the two selected subframes in $Y1$, $Y2$.

The motion level (lm) can be categorized as low motion videos, medium motion videos and high motion videos. Lm <10 is low motion videos, lm at (10, 20) is medium motion videos, and lm >20 is high motion videos. The block diagram of the motion analyzer is shown in figure 4.

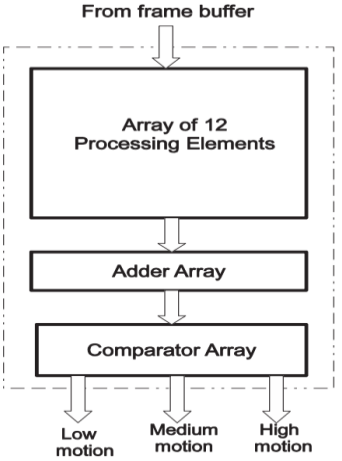

Figure 4. Structure of motion analyzer.

DOI: 10.54254/2755-2721/49/20241073

Motion analyzer is based on the design of processing element. Processing elements are selected by means of SAD registers. When lm is located in three different intervals, the value of SAD is used to find the minimum SAD through the comparator array and then find the matching block. The SAD module is shown in figure 5, where PE is the processing element. The results show a 4.6x speedup with this architecture.

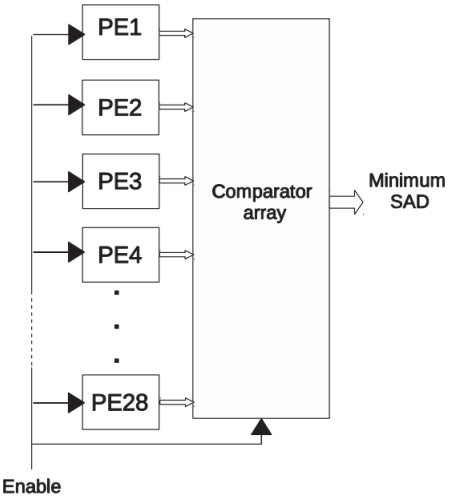

Figure 5. Structure of SAD module.

2.3. VIVADO HLS

Vivado High-Level Synthesis (HLS) is a software developed by Xilinx. It can translate a C or C++ design into a Register (reg) Transfer Level (RTL) implementation. Users can use its IP core to implement basic mathematical operations and signal processing.

The traditional RTL design flow uses Hardware Description Language to describe the algorithm and test bench to perform RTL simulation. The next step is Synthesis and Implementation. The final process is to perform System Debug. C design flow uses C/C++ to describe the algorithm and test bench to implement C Simulation. We then do C Synthesis and RTL Synthesis Implementation. In C Synthesis, the conversion of C to RTL is done. Finally, we execute System Debug. The C design flow is shown in figure 6.

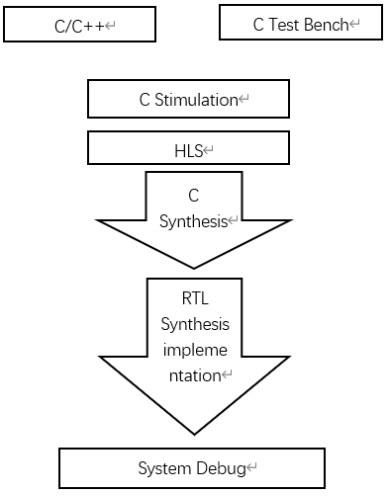

Figure 6. C design flow.

DOI: 10.54254/2755-2721/49/20241073

Besides, Vivado HLS provides a variety of optimization strategies, including pipelining, parallelization, resource sharing, etc., to help developers optimize the performance, power consumption, and resource utilization of the design. Compared with RTL development, Vivado HLS can implement the design faster, reduce the development cycle, improve the development efficiency, and achieve speedup from days to hours.

In general, Vivado HLS has higher development efficiency, faster design speed, richer optimization strategies and more flexible design choices than RTL. Therefore, this software will be used for motion estimation hardware design in this paper.

3. Hardware Designs for Tpical Motion Estimation

The analysis data depends on key computational bottlenecks, such as FS being computationally intensive, DS having few complexities but irregular computational points, etc. The purpose of hardware structure design for FS and DS is to explore the efficiency of Vivado HLS on ME, and the efficiency provides a reference for further use of FS and DS.

The hardware structure of this paper is designed for images with a resolution of 1920×1080 at 60 frames per second. In blocks of 8×8, for each block, its blk_cycle is given by the equation (3) under the 150M clock frequency as,

$$blk\_cycle = \frac{150\times10^6\times8^2}{1920\times1080\times60} = 77 \tag{3}$$

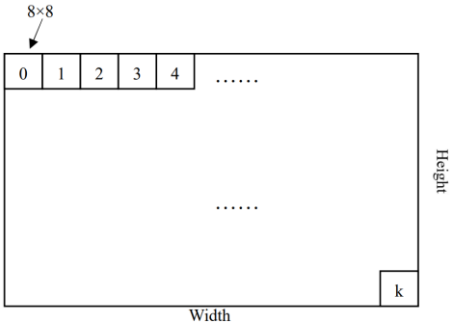

Figure 7. Image partitioning process.

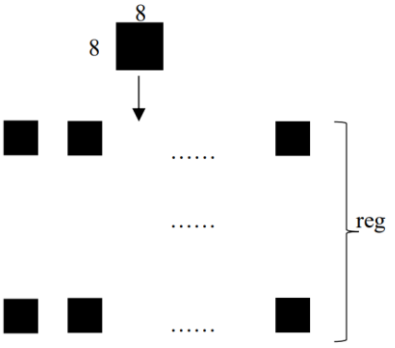

Figure 8. Fetch process of each pixel paralleled in registers.

DOI: 10.54254/2755-2721/49/20241073

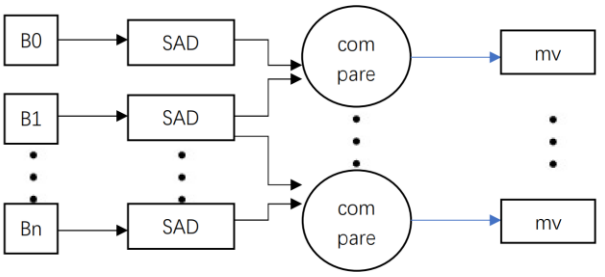

Figure 9. SAD computation parallel process.

Therefore, our target is to design efficient hardware architectures to achieve this goal for each 8×8 block. The hardware design is presented by several processes as follows. Firstly, the image is divided into several blocks in 8×8 units, as shown in figure 7.

Then, to achieve faster implementation speed, we should consider the parallelism strategy. Hence, each 8×8 block is partitioned totally and stored in registers separately above all. It will help to access each pixel simultaneously by read and write operation. The pixel data organization is shown in figure 8.

Based on these pixels, the SAD computation can be unrolled into register level parallelism, which means each candidate SAD can be obtained at the same time for current blocks with its candidate reference blocks at the pixel level, which is shown in figure 9, where Bn stands for one block.

To compute the SAD, subtract the original pixel from the reference pixel, which are denoted as Pnorg and Pnref for the nth pixel, then the difference is sent to the Absolute (ABS) value operator module, and finally, all the values are taken as sum magnitude for the current block. The calculation process of SAD is shown in figure 10.

In order to reduce the latency, pipelining operation is used in the hardware design in this paper. As shown in figure 11, we can pipeline each block in figure 7 to achieve a speedup., where F stands for Fetch, CA is Calculate Difference and ABS operation, and Cp is Compare cost.

Based on the above structure design, this paper uses Vivado HLS optimization instruction to realize PIPELINE processing, as shown in **Pseudocode I**, where #pragma HLS PIPELINE stands for pipelining the corresponding logic. It should be noted that in this paper, pixel-wise pipelining is pipelined, so almost fully parallel operation can be achieved.

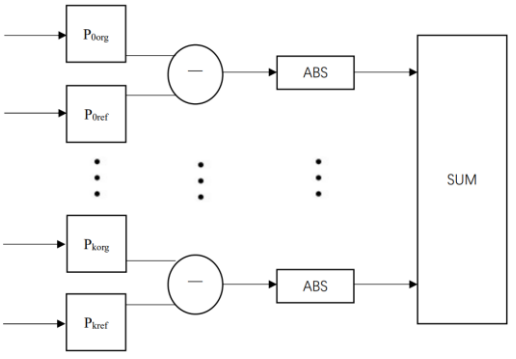

Figure 10. SAD calculation kernel process.

DOI: 10.54254/2755-2721/49/20241073

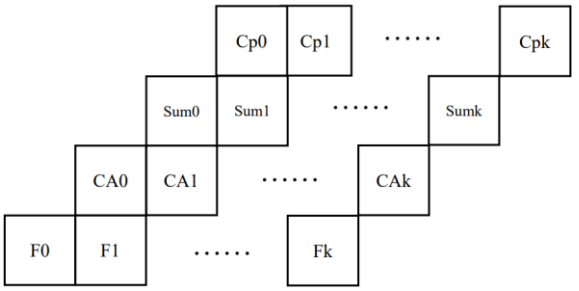

Figure 11. Pipelined SAD computation design.

Pseudocode. I

```
func FS_HLS()
{
for (int j = 0; j < 2*MAX_MOTION; j++)
{
#pragma HLS PIPELINE
for (int i = 0; i< 2*MAX_MOTION; i++)
{
#pragma HLS PIPELINE
fetch_ref();
cal_difference_and_abs();
sad = get_sum();
compare_cost();
}
}
}
```

4. Experiment Results

To verify the hardware structure design performance of the proposed FS and DS, this paper uses the Vivado HLS 2019.2 version for experiments. As mentioned before, the experiments were analyzed target to achieve 1080P@60fps based on an 8x8 block size process unit cycle of no more than 77 cycles.

On one hand, to show the algorithm performance difference between the FS and DS methods, the PSNR(Peak Signal Noise Ratio) and implementation time are provided as the comparison in table 1. From the results, FS has higher search accuracy, but it takes more time than DS.

Table 1. Motion results and time comparison.

	DS	FS
Frame	0-29	0-29
Average PSNR	27.35dB	27.56dB
Time	0.364s	0.375s

DOI: 10.54254/2755-2721/49/20241073

Table 2. Hardware implementation cycle for different motion estimation methods.

	FS	DS
Clock	ap_clk	ap_clk
Target	10.00ns	10.00ns
Estimated	8.354ns	8.750 ns
Uncertainty	1.25ns	1.25ns
Cycle	67	24

Table 3. Hardware implementation resources for different motion estimation methods.

	FS		DS	
Name	FF	LUT	FF	LUT
DPS	-	-		
Expressiono	0	41	0	145
FIFO	-	-		
Instance	33	93005	0	92955
Memory	-	-	10	4
Multiplexer	-	1185	-	159
Register	2747	-	5335	160
Total	2780	94231	5345	93423
Utilization (%)	~0	3	~0	3
Utilization SLR (%)	~0	11	~0	11

On the other hand, the experiments were synthesized at 10ns. Timing and Latency are given for the hardware implementation cycle for different motion estimation methods as listed in table 2. The hardware implementation resource utilization results are given in table 3. From the results, it can be seen that FS uses 67 cycles and DS uses 24 cycles. In terms of utilization estimates, DS has expensed more utilization of FF and LUT than FS, which can help to achieve higher throughput than FS.

In a word, as to the software implementation for the two motion estimation methods, FS has relatively high complexity, more accurate search and implantation time. While the DS has lower complexity, weak search accuracy and less time. As to the hardware implementation of them, they can both achieve real-time processing ability as the goal for a video with 1920×1080 at 60 frames per second. It should be noted that the hardware design for the DS method can obtain more throughout with only 24 cycles as compared to the FS, which means it can achieve more than 120fps for 1920x1080. Therefore, the DS can save several hardware resources when we slow down its throughout.

5. Conclusions

This article investigates fast motion estimation algorithms and hardware structure design for real-time applications. Specifically, aiming at the processing speed of 1920x1080@60fps, this article explores a feasible hardware scheme based on the Vivado HLS high-level synthesis tool, which meets the real-time requirements for the typical full search and diamond search motion estimation methods. From the experimental results, FS has higher search accuracy and more search time. DS has expensed more utilization of FF and LUT than FS and can save more hardware resources. In the future, based on high-level synthesis tools, we will continue to further explore hardware design and achieve a more optimized motion estimation hardware structure.

References

[1] Pan Z, Lei J, Zhang Y, et al. Fast motion estimation based on content property for low-complexity H. 265/HEVC encoder[J]. *IEEE Transactions on Broadcasting*, 2016, 62(3): **675-84**.

[2] Luo J, Yang X, Liu L. A fast motion estimation algorithm based on adaptive pattern and search priority[J]. *Multimedia Tools and Applications*, 2015, 74: **11821-36**.

DOI: 10.54254/2755-2721/49/20241073

[3] Agha S, Khan M, Jan F. Efficient fast motion estimation algorithm for real-time applications[J]. *Journal of Real-Time Image Processing*, **2022: 1-11**.

[4] Pakdaman F, Hashemi M R, Ghanbari M. A low complexity and computationally scalable fast motion estimation algorithm for HEVC[J]. *Multimedia Tools and Applications*, 2020, 79: **11639-66**.

[5] Pan Z, Zhang R, Ku W, et al. Adaptive pattern selection strategy for diamond search algorithm in fast motion estimation[J]. *Multimedia Tools and Applications*, 2019, 78: **2447-64**.

[6] Jou S Y, Chang S J, Chang T S. Fast motion estimation algorithm and design for real-time QFHD high efficiency video coding[J]. *IEEE Transactions on Circuits and Systems for Video Technology*, 2015, 25(9): **1533-44**.

[7] Yavuz S, ÇelebE A T, ÇelebE A, et al. Local binary pattern method and its hardware architecture for low-complexity motion estimation[C]//*2017 25th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2017: **1-4**.

[8] Yahi A, Toumi S, Bourennane E, et al. A speed FPGA hardware accelerator based FSBMA-VBSME used in H. 264/AVC[J]. *Evolving Systems*, 2016, 7: **233-41**.

[9] Silveira E, Diniz C, Fonseca M B, et al. SATD hardware architecture based on 8× 8 Hadamard Transform for HEVC encoder[C]//*2015 IEEE International Conference on Electronics, Circuits, and Systems (ICECS)*. IEEE, 2015: **576-9**.

[10] Zheng J, Lu C, Guo J, et al. A hardware-efficient block matching algorithm and its hardware design for variable block size motion estimation in ultra-high-definition video encoding[J]. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2019, 24(2): **1-21**.

[11] Chandran K R S, Chandramani P V. Hardware-software co-design framework for sum of absolute difference based block matching in motion estimation[J]. *Microprocessors and Microsystems*, 2020, 74: **103012**.