A Reconfigurable Implementation of Motion Compensation in HEVC

Xiaoyan Xie*, Xiang Lei*, Jinna Zhou*, Yun Zhu*, Lin Jiang*
*Xi'an University of Posts & Telecommunications, Xi'an, China
E-mail: 18829290892@163.com Tel: +86-29-1882929082

***Abstract*—The motion compensation algorithm in High Efficiency Video Coding (HEVC) has a large number of interpolation calculations at the same time, and it is difficult to achieve flexible switching of different coding blocks, which puts higher requirements on its computational efficiency and control logic. In order to solve such problems, the data is divided according to the characteristics of the algorithm, and the motion compensation algorithm is mapped onto the reconfigurable array structure, so that the previous serial algorithm can be processed in parallel. According to the data overlapping relationship between the next reference block and the current reference block of the encoding process, the data multiplex idea is used to reduce the number of the pixels which were read from the external storage, thereby shortening the reading time of the next reference block data. At the same time, according to the reconfigurable structural features, flexible switching of the algorithm variable block mode is designed to improve flexibility. Finally, parallel processing is performed according to the data rule of motion compensation algorithm and a large number of interpolation characteristics, which improves the computational efficiency of the algorithm. In this paper, a 16×16 Processing Element (PE) is used to dynamically process a 4×4-64×64 block size. On the Virtex-6 FPGA attached to the BeeCube, the reference block update speed is increased by 39.9%; in the case of an array size of 16 PEs. In parallel, the degree of parallelism can reach 16, which has better flexibility while achieving higher execution efficiency.**

Keywords: High Efficiency Video Coding(HEVC), Motion compensation, Parallelization, Reconfigurable

I. INTRODUCTION

The new generation of multimedia video coding standard HEVC/H.265[1] proposed by the video coding joint group has significant compression performance, reducing 50% of the bit rate while providing coding mode similar to H.264[2]. Especially in inter prediction, the motion compensation algorithm uses an 8-tap luma interpolation filter and a 4-tap chroma interpolation filter as interpolation portions, which requires more reference frame information to be generated, resulting in a larger data manipulation amount. The entire HEVC interpolation filter accounts for 20~30% and 20~40% of the encoder and decoder execution time, respectively, which makes interpolation filter one of the most time-consuming coding parts in HEVC[3]. In order to improve the computational efficiency of motion compensation algorithms, scholars have proposed a variety of solutions.

In[4][5] designed a hardware accelerator for fractional motion estimation. The core of the algorithm is to accelerate the interpolation calculation and improve the coding and energy efficiency. However, the acceleration architecture only designs 8×8 and 16×16 block sizes.The flexibility is not high. In [6], a new half-pixel and quarter-pixel interpolation filter are proposed, which can reduce the computation time of the algorithm by half, but can only process 4×4 blocks, which makes the algorithm less flexible. Reference [7] uses 64 reconfigurable filters to satisfy different HEVC filter types. Although architecture can achieve high throughput, it is only suitable for 8×8 coded blocks and is less flexible. A simplified field-programmable gate array (FPGA) was proposed in [8] to implement a fractional motion estimation interpolation architecture that increased the rate by 13%, but only processing 8x8 coded blocks resulted in reduced flexibility. Low-energy HEVC sub-pixel interpolation hardware for all PU sizes is proposed in [9], which can process 30 frames of quadruple full HD (3840×2160) video per second, but its operating frequency is lower. Therefore, how to balance the flexibility of the algorithm and the computational efficiency of motion compensation has become a basic problem to be solved.

This paper uses a dynamic programmable reconfigurable array processor to achieve parallelization of motion compensation algorithms. This reconfigurable array structure have taken the advantages of the flexibility of a general-purpose processor and the efficiency of a dedicated circuit[10], so it is an effective way to achieve video codec with high computational complexity and variable block modes. Based on the characteristics of this structure, the idea of data multiplexing can be used to reconstruct the variable block mode switching of the motion compensation algorithm. The results show that the proposed reconfigurable scheme has a great improvement in both resource consumption and algorithm operation efficiency. The organization structure of this paper is as follows. The first part mainly analyzes the related concepts of motion compensation algorithm, and the second part analyzes the update process of reference block data. The third part is a reconfigurable implementation of the motion compensation algorithm. The fourth part verifies the feasibility of the reconfigurable implementation of the motion compensation algorithm and analyzes the verification results. Finally, the full text is summarized.

II. MOTIVATION

In the latest video codec standard HEVC, Fig. 1 is a schematic diagram of the position of a motion compensation algorithm luminance interpolation score precision sample. The position of the integer sample in the figure is represented by

uppercase letters, and the position of the decimal sample is represented by lowercase letters. When the motion vector points to the position of the integer sample, the interpolation operation is not performed, and the integer sample value is directly output as the final prediction result. When the motion vector points to a non-integer position (fractional position), ie 1/2 pixel precision or 1/4 pixel precision position, the motion compensated interpolation module will use the interpolation filter to perform samples on the integer around the non-integer sample prediction interpolation position.

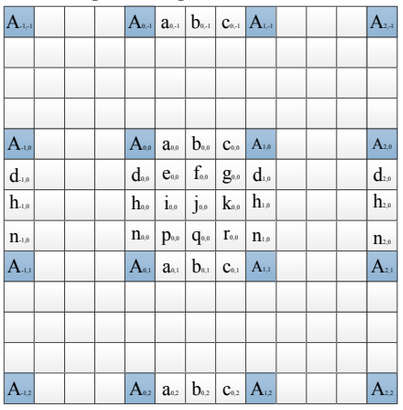


Fig. 1 Schematic diagram of the sample location for the fractional precision of the brightness interpolation

For the motion compensation algorithm, if only one pixel is processed at a time, it takes a long time, so that the calculation efficiency of the algorithm is also low. For example, to process a block size of 8x8, there is no data correlation between the interpolation calculation for each pixel and the interpolation calculation for the other 63 pixel values. Therefore, parallel ideas can be used to process multiple pixels simultaneously. Compared with H.264, HEVC inter mode achieves better coding efficiency at the expense of higher complexity, with prediction units ranging from 4 by 4 to 64 by 64[11]. The parallelization hardware architecture of the traditional interpolator is only processed by a single PU block[12]. If all the PU modes between frames are predicted, the hardware architecture is redesigned, which greatly wastes resources.

In order to improve the flexibility of the algorithm, reduce the use of hardware resources. This paper is based on a reconfigurable array processor to achieve flexible switching of different block sizes of motion compensation algorithms. The reconfigurable array processor includes a global instruction memory, an input memory, an output memory, an array processor, and a global controller. The global controller is used to store the operation instructions and call instructions of the array processor, and also includes broadcasting of different block size operation instructions, distribution of call instructions, and collection of computing resource information, and the input memory is responsible for loading corresponding input data from the external memory. A layered programming network is formed between the host interface and the array processor during processing, and a layered programming network is used to implement control and management of array computing resources. In order to facilitate the addressing, in the addressing process, the bit width is gradually decremented to ensure that each instruction arrives at the PE at the same time, and the different PEs execute the corresponding operation after the instruction arrives.

III. RECONFIGURABLE IMPLEMENTATION OF MOTION COMPENSATION ALGORITHM

The kernel of the dynamic reconfiguration mechanism is the command delivery network. This paper mainly utilizes instruction broadcast operations and instruction issued operations to form the instruction delivery network. The instruction broadcast operation is for the reconstruction of the size in the reconstruction of this document. The reconstruction means to store the instructions in advance in the instruction memory of each PE in the array structure, and then to turn on all or part of the PE at the same time through the instruction broadcast operation. The instruction issuing operation delivers the instruction in the global instruction memory to the designated PE through the instruction issuing operation. The scale reconstruction has 256 PEs (16 PEGs), 64 PEs (4 PEGs) and 16 PEs (1 PEGs) and several irregular block mode sizes, as shown in Fig.2.When performing a block size of 8x16-64x64, only size reconstruction is needed, and the block will be decomposed into a number of 8x8 block sizes. The local storage of PEG00-PEG33 simultaneously stores 8×8 block size instructions, and its size reconstruction is shown in Fig. 2. If a 64×64 block size is used, all the clusters (PEG00-PEG33) will work through the instruction broadcast operation instruction, but each time a 32×32 block size can be executed at most, and therefore, four commands should be issued by the issuing instruction. Only one 64×64 block can be processed. If a block size of 32x32 is made, all the 16 clusters of PEG00-PEG33 will be operated by the broadcast operation instruction. If the 32×16 block size is processed, the eight clusters of PEG00-PEG13 will be operated by the broadcast operation instruction, and the remaining clusters do not operate at the same scale. (Specifically shown in Fig.2).Each cluster has an 8x8 block size, and other block sizes are handled by combining cluster sizes. When executing a block size of 4×4-16×4, since only 8×8 block size instructions are stored in the local storage, it is necessary to transfer the network to PEG00, PEG01, PEG02, PEG03, PEG10, PEG11, PEG12 through the instruction transmission network. PEG13, PEG20, PEG21, PEG22, PEG23, PEG30, PEG31, PEG32, PEG33 issues 4×4 block size instructions. The block size in this range is divided into a number of 4×4 block sizes, and 4×4 instructions are issued through the instruction transmission network, as shown in Fig.3. When processing a block size of 16×12, PEG00-PEG03, PEG10-PEG13, and PEG20-PEG23 were operated by the broadcast manipulation instruction.

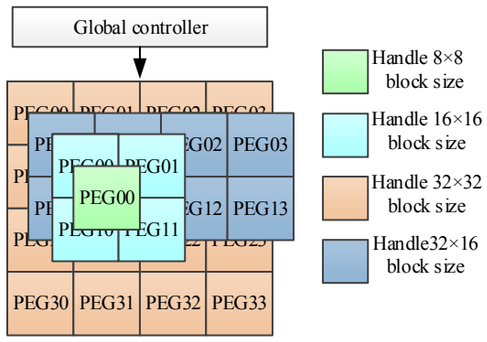


Fig. 2 8×16-64×64 motion compensation algorithm reconfigurable function map

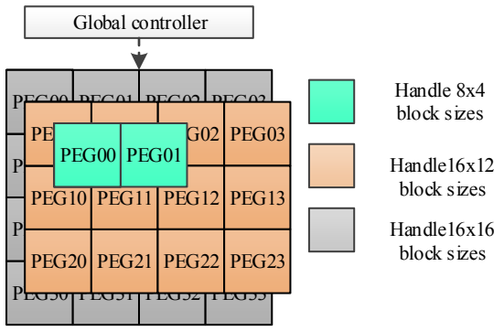


Fig.3 4×4-16×4 motion compensation algorithm reconfigurable functional map

The following is an example of how to switch a block size of 16x12 to a 32x32 block size as an example. The first block address of the encoded block should be stored as different resolution test sequences before loading. First, the YUV test sequence has to be converted to a decimal value, which can be identified by the array structure using MATLAB Lab software. The data is distributed in an array format and stored in DIM.

First, the 64x32 block size is processed, broken down into 32 8x8 blocks and processed in PEG00-PEG37, respectively. The 8x8 motion compensation instructions are stored in the local instruction memory of PEG00-PEG37. Processing a 16×12 block size requires decomposing it into 12 4×4 blocks for processing in PEG00-PEG03, PEG10-PEG13 and PEG20-PEG23, respectively, and transmitting 4×4 motion compensation instructions through the command transmission network. Go to the local instruction memory of each PE requiring it.

Next, if the block size is 32 × 32, it indicates that the instruction transmission network performs a broadcast (bit [31:30] = 10) operation, And at the same time, the execution operation of each PE is started. If the block size is 16 x 12, the command transmission network performs instruction transfer (bits [31:30] = 01) and issues a 4x4 motion compensation command to PEG00-PEG03, PEG10-PEG13 and PEG10-PEG13. In the local instruction memory of PEG20-PEG23, as shown in Fig. 3 and 4, solid arrows indicate that the execution operation of each PE is turned on, and dotted arrows indicate the issued instructions.

Finally, start execution the motion compensation code is started. The specific execution process of each cluster is the same.

Step1: Data loading; PE01 accesses the DOM through register R11 and reads the corresponding reference pixel value. PE00 accesses the DIM through register R11 and reads the corresponding original pixel value. When loading data, it is executed from left to right and top to bottom.

Step 2: The reference pixel value is then sent to the corresponding PE. The data transfer order is PE01 to PE00, PE01, PE02, PE03 and PE11. The, PE00 is sent to PE10, PE20 and PE30; PE11 is delivered to PE21 and PE31; PE02 is sent to PE12, PE22 and PE32; PE03 is sent to PE13, PE23 and PE33. In order to improve the efficiency, no other PEs will be issued until PE00 has been issued. Instead, the PE will transfer the data to the appropriate PE as soon as it receives the data.

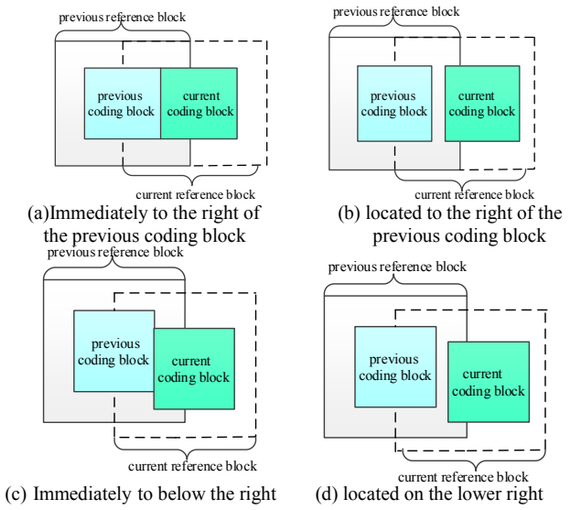
Step 3: After each PE receives the data, it starts to perform 1/2 or 1/4 interpolation calculation. After the calculation is completed, 4 predicted values are stored in each PE, and then each PE sequentially transmits 4 predicted values in its own memory to PE03 in sequence.

Step 4: Calculate the residual based on the predicted value and the original pixel. The process of processing the block size of 16x12 is similar to the above.

IV. REFERENCE BLOCK DATA UPDATE

The update of reference block is ensured based on the position of the current coding block in the previous coding block, There are four positional relationships, as shown in Fig4. The reference block design update can be divided into four cases. According to the four cases, the data amount of data overlap is different. According to the update direction, it can be divided into two ways. One is to update only the right data, and the other is to update. The data on the right side also updates the next row of data.

The current coding block is next to the previous coding block. As shown in Fig. 4(a), the process of updating the reference block is divided into two steps. Step one is to process the pixels of the reference block by column: First, delete the first 8 columns of pixel values of the 15x15 reference block in the processing element. At the same time the ninth column pixel of the 15×15 reference block pixel in the processing element is taken as the first column pixel of the reconstructed 15×15 reference block. Then, the 10th column pixel of the 15×15 reference block pixel in the processing element is regarded as the second column of pixels of the 15×15 reconstructed reference block. And so on, using the 15th column of the 15×15 reference block pixels in the processing element as the 7th column of pixels reconstructing the 15×15 reference block, this completes the reconstruction Refer to the first 7 columns of the block.



(a)Immediately to the right of the previous coding block (b) located to the right of the previous coding block

(c) Immediately to below the right side of the previous coding block (d) located on the lower right side of the previous coding block

Fig.4 Data overlap between adjacent reference blocks

Step two is to load the remaining reference block data from the external memory: since the first 7 columns of pixels have already existed, this step loads 8 columns of pixel data by row. The first 8 pixels are loaded from the external memory, updated to the pixel position in the 1st row of the 15×15 reference block. Then 8 pixels in the next row are updated to the pixel position of the 2nd row of the reconstructed 15×15, etc. Finally the eight pixels of the fifteenth line is loaded from the external memory and updated to the pixel position of the fifteenth line of the 15x15 reference block. The process of Fig4. (b) is essentially the same as the process to the right of the current coded block of the previously coded block.

The current coding block is immediately adjacent to the lower right side of the previous coding block. As shown in Fig4. (c), the process of updating the reference block is also divided into two steps: Step one is to process the reference block column by column: first, delete the pixel value of the first row and the first 8 columns of pixels of the 15x15 reference block in the processing element, and so did the ninth column pixel (only 14 pixels) of the 15 × 15 reference block pixels in the post-processing element is deleted as the first column pixel of the 15 × 15 reference block. Then the 15 × 15 reference block in the processing block is processed. The 10th column pixel of the pixel (only 14 pixels) is used as the 2nd column pixel of the 15×15 reference block. And so on, the 15th column pixel (only 14 pixels) of the 15×15 reference block pixel is finally processed. The 7th column of pixels of the 15x15 reference block is reconstructed, which first processes the first 7 column blocks of the reconstruction reference

Step two is to load the data line by line from the external memory: since the first 7 columns of pixels already exist, this step processes the 8 columns and the last row of pixels. Data is loaded the data line by line. The first 8 pixels are loaded from the external memory, placed at the position of the pixel in the 1st row of the 15x15 reference block. Then the 8 pixels in the next row are loaded from the external memory and placed in the reconstruction 15x15. Referring to the position of the pixel in row 2 of block 15. Loading 8 pixels of the 3rd row from the external memory, placing the pixel position of the 3rd row of the 15x15 reference block. And so on, up to 8 pixels of the 14th row Loaded from external memory and placed on the 14th row of pixels of the 15x15 reference block. Finally, 15 pixels are loaded from the external memory and placed on the 14th row of pixels in the 15x15 reference block. The process of Fig. 4(d) is essentially the same as the process immediately below the current coded block of the previously coded block.

V. IMPLEMENTATION AND RESULT

In order to verify the feasibility of the reconfigurable implementation of motion compensation algorithm, dynamic reconfigurable array structure is used for verification in this paper. The method is as follows: first, modify the configuration file of the test model HM10.0, obtain test data and block partition information, store it in off-chip memory. And then , use QuestaSim simulation to map the reconfigurable scheme to the dynamically reconfigurable array platform for verification. Among them, the motion compensation algorithm performs time statistics and comparison under serial execution and parallel execution conditions respectively. As shown in Fig. 5, parallel execution saves about 88% of coding time than serial execution. The reconfigurable coding block time results are shown in Table 1. After the simulation verification of the reconfigurable implementation of the motion compensation algorithm, the time for each block size and resource occupation is calculated. It can be seen that as the number of PE arrays increases, the size of blocks that can be processed is larger, the circuit scale is in an increased state, and the running time is also increasing. For different block sizes, a reconfigurable implementation can be implemented to enable flexible algorithm switching.

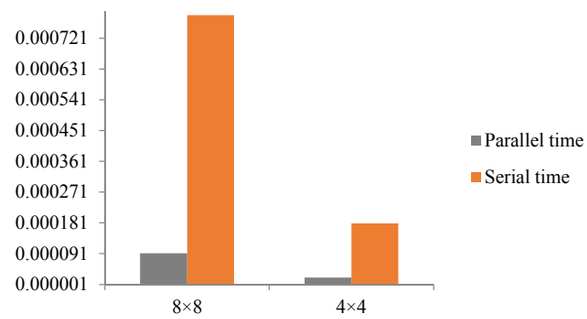


Fig.5 Serial time and parallel comparison

Table1 The Experimental Results of Motion Compensation

Array size	Processing block size	Time(s)	Frequency(MHZ)	Resources

4×4	4×4	2.153×10^{-5}	156	31200(LUT)
	8×8	9.254×10^{-5}		9800(Flip-flops)
8×4	8×4	4.291×10^{-5}	156	63521(LUT)
	16×8	1.943×10^{-4}		18654(Flip-flops)
16×8	16×8	1.943×10^{-4}	156	250354(LUT)
	32×16	7.362×10^{-4}		79053(Flip-flops)
16×12	16×12	2.206×10^{-4}	156	375232(LUT)
	32×24	1.088×10^{-3}		117549(Flip-flops)
16×16	16×16	3.445×10^{-4}	156	485789(LUT)
	32×32	1.481×10^{-3}		157320(Flip-flops)

The current coding block is adjacent to the right side of the previous coding block and has a type of a. The current coding block is located at the right b of the last coding block, and the current coding block is located at the lower right corner of the previous coding block as c. The coding block is located at the lower right corner of the last coding block, d. According to the following analysis, when the current coding block is immediately adjacent to the right side of the previous coding block, the time for updating the reference block is increased by 46.6%; when the current coding block is located on the right side of the previous coding block and the current coding block is immediately adjacent to the right of the previous coding block At the bottom, the update speed of the reference block is increased by 39.9%; when the current code block is located at the lower right of the previous block, the reference block update speed is increased by 33.4%. as shown in Table 2.

Table2 Comparison of Reference Block Update Time (Unit: Clock Period)

Parallel scheme	Traditional reference block update time	Reference block update time after data multiplexing	Increase the percentage of time
a	3059	1632	46.6%
b		1836	39.9%
c		1836	39.9%
d		2040	33.4%
average	3059	1836	39.9%

Compared with the ASIC implementation introduction of the parallel implementation of the HEVC motion compensation algorithm based on the reconfigurable video array processor in table 3. Literature [5] designed a parallel architecture of the interpolator, but its frequency is lower than this paper, and the resource occupancy is higher than this paper. The parallel structure proposed in [6] can also handle variable block size by extending the hardware architecture, but its parallelism can only reach half of the paper. [6] Although the hardware resources are smaller than the design in this paper, the working frequency and parallelism are smaller than this paper. The highly parallel pipeline design proposed in [11] can process 32 pixels simultaneously. The parallel structure proposed in this paper is equivalent to the resource consumption of 32 PE arrays. However the parallelism is slightly higher than it.

Table3 Comparison with other ASIC architectures

Compare items	Technology	Frequency	Resources	parallelism
[5]	0.13um	200(MHZ)	148.653(gate)	/
[6]	180nm	185(MHz)	41.97k(gate)	8
[11]	40nm	342(MHz)	297.3k(gate)	32
This work	90nm	357(MHz)	137.425k(gate)	16

Table 4 shows a comparison with the FPGA implementation. Although the resource occupancy in [5] is lower than this paper, its frequency is much lower than this paper. [7] is an FPGA implementation of interpolation filters, so their hardware area is slightly smaller than the design of this article. However, it only works with 8×8 coded blocks and has poor flexibility. In [13], the hardware frequency is slightly lower, and the hardware resources are more than four times that in this paper. The hardware structure of this article is highly scalable. 16 PE-level array structures can achieve 16 degrees of parallelism, and 32 PE array structures can achieve 32 degrees of parallelism.

Table4 Comparison with other FPGA architectures

Compare items	Technology	Frequency	Resources(LUT)	parallelism
[5]	Arria Ⅱ GX	100(MHz)	19.106	/
[7]	Arria Ⅱ GX	200(MHz)	28.757k	64
[13]	Zynq 7045	150(MHz)	126 k	/
This work	Virtex-6	156(MHz)	31.2k	16

VI. CONCLUSIONS

Based on the reconfigurable video array processor architecture, a new high-parallelism reconfigurable scheme of the HEVC variable block motion compensation algorithm is proposed. Firstly, the parallelization of the motion compensation algorithm is implemented according to the data correlation, and the reference block data is updated by the data multiplexing idea. Secondly, the size of the video array processor can be dynamically adjusted according to different coding block sizes, and reconfigurable implementations of different block sizes can be implemented. This design allows for flexible block size switching and maximizes the use of reconfigurable array processors. The experimental results show that for the implementation of the motion compensation algorithm, the structure can increase the average speed of the

reference block update by 39.9%. For the block mode switching of the motion compensation algorithm and the dynamic adjustment of the processor scale, the structure can be correctly scheduled and can be improved Parallelism. The most outstanding advantage of this structure is that the reconfigurable structure has better flexibility to facilitate different needs and is more suitable for market applications.

ACKNOWLEDGMENT

This paper is supported by the National Natural Science Foundation of China under Grant No.61802304, No.61772417, No.61834005, No.61602377, No.61874087 and No. 61634004, The Shaanxi Province Co-ordination Innovation Project of Science and Technology under Grant, and the National Science under Grant No.2016KTZDGY02-04-02, The Shaanxi Provincial key R&D plan under Grant No.2017GY-060 and Shaanxi International Science and Technology Cooperation Program No.2018KW-006.

REFERENCES

[1] Team J V. Draft ITU-T Recommendation and Final draft international standard of joint video specification.2013.

[2] Sullivan G J, Ohm J R, Han W J, et al. Overview of the High Efficiency Video Coding (HEVC) Standard[J]. IEEE Transactions on Circuits & Systems for Video Technology, 2012, 22(12):1649-1668.

[3] Ugur K, Alshin A, Alshina E, et al. Interpolation filter design in HEVC and its coding efficiency-complexity analysis[C]// IEEE International Conference on Acoustics, Speech and Signal Processing,2013:1704-1708.

[4] I. Seidel, V. Rodrigues Filho, L. Agostini et al.Coding- and Energy-Efficient FME Hardware Design[C]// 2018 IEEE International Symposium on Circuits and Systems (ISCAS), Florence, 2018:1-5.

[5] Pastuszak, Grzegorz et al. Optimization of the Adaptive Computationally-Scalable Motion Estimation and Compensation for the Hardware H.264/AVC Encoder[J]. Journal of Signal Processing Systems, 2016,82(3): 391-402.

[6] Kammoun M, Atitallah A B, Masmoudi N. An efficient hardware architecture for interpolation filter of HEVC decoder[C]// IEEE 12th International Multi-Conference on Systems, Signals & Devices(SSD15),2015:1-5.

[7] Pastuszak G, Trochimiuk M. Architecture design of the high-throughput compensator and interpolator for the H.265/HEVC encoder[J]. Journal of Real-Time Image Processing, 2016, 11(4): 663-673.

[8] Afonso V, Maich H, Agostini L, et al. Low cost and high throughput FME interpolation for the HEVC emerging video coding standard[C]// IEEE 4th Latin American Symposium on Circuits and Systems (LASCAS), 2013: 1-4.

[9] Kalali E, Hamzaoglu,et al. A low energy HEVC sub-pixel interpolation hardware[C]// IEEE International Conference on Image Processing (ICIP), 2014:1218-1222.

[10] Li Tao, Jiang Lin, Liu Zhenwei, Han Jungang, Du Huimin. A Novel Array Video Signal Processing Unit Structure [P], China, (National Property Rights Bureau of the People's Republic of China) 20012 China Public, 201110046537.

[11] Wang S, Zhou D, Zhou J, et al. VLSI Implementation of HEVC Motion Compensation With Distance Biased Direct Cache Mapping for 8K UHDTV Applications[J]. IEEE Transactions on Circuits and Systems for Video Technology, 2017, 27(2):380-393.

[12] J. S. León, C. S. Cárdenas and E. V. Castillo.A high parallel HEVC Fractional Motion Estimation architecture[C]//2016 IEEE ANDESCON, Arequipa, 2016:1-4.

[13] Abeydeera M, Karunaratne M, Karunaratne G, et al. 4K Real-Time HEVC Decoder on an FPGA[J]. IEEE Transactions on Circuits & Systems for Video Technology, 2015, 26(1):236-249.