Zhou et al. EURASIP Journal on Advances in Signal Processing (2015) 2015:95
DOI 10.1186/s13634-015-0284-0

EURASIP Journal on Advances in Signal Processing
a SpringerOpen Journal

RESEARCH Open Access

An efficient interpolation filter VLSI architecture for HEVC standard

Wei Zhou[1*], Xin Zhou[2], Xiaocong Lian[1], Zhenyu Liu[3] and Xiaoxiang Liu[4]

Abstract

The next-generation video coding standard of High-Efficiency Video Coding (HEVC) is especially efficient for coding high-resolution video such as 8K-ultra-high-definition (UHD) video. Fractional motion estimation in HEVC presents a significant challenge in clock latency and area cost as it consumes more than 40 % of the total encoding time and thus results in high computational complexity. With aims at supporting 8K-UHD video applications, an efficient interpolation filter VLSI architecture for HEVC is proposed in this paper. Firstly, a new interpolation filter algorithm based on the 8-pixel interpolation unit is proposed in this paper. It can save 19.7 % processing time on average with acceptable coding quality degradation. Based on the proposed algorithm, an efficient interpolation filter VLSI architecture, composed of a reused data path of interpolation, an efficient memory organization, and a reconfigurable pipeline interpolation filter engine, is presented to reduce the implement hardware area and achieve high throughput. The final VLSI implementation only requires 37.2k gates in a standard 90-nm CMOS technology at an operating frequency of 240 MHz. The proposed architecture can be reused for either half-pixel interpolation or quarter-pixel interpolation, which can reduce the area cost for about 131,040 bits RAM. The processing latency of our proposed VLSI architecture can support the real-time processing of 4:2:0 format 7680 × 4320@78fps video sequences.

Keywords: HEVC, Interpolation filter, VLSI

1 Introduction

Now, the Joint Collaborative Team on Video Coding (JCT-VC) is developing the next-generation video coding standard, called High-Efficiency Video Coding (HEVC) [1, 2]. It provides a significant rate-distortion improvement over its predecessor H.264/AVC and can save 40–50 % bit rates compared to H.264/AVC, especially for 4K (3840 × 2160)/8K (7680 × 4320)-ultra-high-definition (UHD) video applications [3]. A number of new algorithmic tools have been proposed, covering many aspects of video compression technology, such as larger coding units, new tools, and more complex prediction schemes.

Motion compensation (MC) is the key factor for efficient video compression. Compensation for motion with fractional-pel accuracy requires interpolation of reference pixels. Therefore, in order to increase the performance of integer pixel motion estimation, the sub-pixel (i.e., half and quarter) accurate variable block size motion estimation is applied in both H.264/AVC and HEVC. The H.264/AVC standard uses a six-tap finite impulse response (FIR) luma filtering at half-pixel positions followed by a linear interpolation at quarter-pixel positions. Chroma samples are computed by the weighed interpolation of four closest integer pixel samples. In HEVC standard, three different eight-tap or seven-tap FIR filters are used for the luma interpolation of half-pixel and quarter-pixel positions, respectively. Chroma samples are computed using four-tap filters. Sub-pixel interpolation is one of the most computationally intensive parts of HEVC video encoder and decoder. In the high-efficiency and low-complexity configurations of HEVC decoder, 37 and 50 % of the HEVC decoder complexity is caused by sub-pixel interpolation on average, respectively [4]. On the other hand, compared with the six-tap filters used in H.264/AVC standard, the seven-tap and eight-tap filters cost more area in hardware implementation and occupy 37~50 % of the total complexity for its DRAM access and filtering. Therefore, it is necessary to design a dedicated hardware

* Correspondence: zhouwei@nwpu.edu.cn
[1]School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710072Shaanxi, China
Full list of author information is available at the end of the article

architecture for MC interpolation filter to realize the real-time processing for high-resolution videos.

Some high-throughput interpolators have been proposed for H.264/AVC in literatures [5–10]. Usually, they are embedded in the fractional motion estimation pipeline stage that follows the integer-pel motion estimation. Their scheduling assumes two successive steps for half and quarter interpolations. A two-step approach is natural in terms of the specification of quarter-pel computations which refer to the results of half-pel computations. This data flow cannot be applied directly in HEVC since the quarter-pel samples are computed using separate filters. In particular, more filters are needed in the second step. Furthermore, the hardware cost increases due to a larger number of filter taps, and thus, much higher throughputs are required (i.e., more partitioning modes).

There have been many previous works focusing on designing efficient architecture for HEVC MC interpolations [11–18]. Huang proposed a high-throughput interpolation filter architecture with a prediction unit (PU)-adaptive filtering flow and a unified filter combining the eight-tap luma and four-tap chroma filters [11]. But its hardware area is larger than the hardware cost proposed in this paper. In [12], a dedicated hardware accelerator for interpolation was presented. Although it could read 8 input samples and produce 64 output samples at each clock cycle, its area cost was huge. An efficient VLSI design which is composed of a reconfigurable filter, an optimized pipeline engine organization, and a filter reuse scheme for HEVC interpolation was proposed in [13]. This hardware is slower than the architecture proposed in this paper because it has restricted reconfigurability for filter data paths. In [14], a simplified fractional motion estimation (FME) architecture for field-programmable gate arrays (FPGAs) is presented that processes only 8×8-sized blocks at the cost of a bit rate increase of 13 %. In [15], reconfigurable acceleration engines were developed in the interpolation filter hardware architecture to adapt to different filter types. In [16], a low-energy HEVC sub-pixel interpolation hardware for all PU sizes was proposed and Hcub multiplierless constant multiplication algorithm was used. But the focus of [14–16] is on developing FPGA-based reconfigurable hardware architecture, and block random access memories (BRAMs) are usually embedded in the FPGA platform.

To overcome the obstacles of the previous work, we proposed a fast interpolation filter algorithm and the corresponding hardware architecture in [18], which can save the encoding time and reduce the computational complexity of fractional motion estimation in HEVC. In the aspect of algorithm, we speed up the encoding process by skipping all the 4×8, 4×16, and 12×16 prediction units in the queue. Based on the algorithm, we designed the interpolation filter VLSI architecture with the reconfigurable configuration and the cell block reuse to reduce the implement hardware area.

In this paper, we extend our earlier work [18] in three ways. First, on the basic of our original algorithm, we skip the 4×8, 4×16, 8×4, 16×4, 16×12, and 12×16 sub-PU blocks in the interpolation process instead of only skipping the 4×8, 4×16, and 12×16 PU, to further save the encoding time and save a large area in hardware implementation by skipping 4× size with acceptable coding quality degradation. Second, an efficient memory organization method is proposed in this paper to reduce the data access of SRAM and save the power of VLSI architecture. Third, a five-step pipeline interpolation filter engine is proposed in this paper to shorten the critical path of the filter and improve the working speed.

Another obvious limitation of our earlier work [18] and the above progress is that they only target at the video applications with the resolutions up to HD or 4K. For higher throughput of 8K-UHD 7680×4320, more efficient hardware architecture is desirable. Although the power-efficient FME VLSI architecture proposed in [17] can realize 8K-UHD video real-time encoding, the focus of it is on the search pattern and hardware architecture of fractional search module. With aims at supporting 8K-UHD video applications, an efficient interpolation filter VLSI architecture is proposed in this paper.

The main contributions of this paper are summarized as follows:

(1) A fast and implementation-friendly interpolation algorithm is proposed, which skips the interpolation process of 4×8, 4×16, 8×4, 16×4, 16×12, and 12×16 sub-PU blocks to reduce the encoding time and hardware complexity.
(2) A reused three-level interpolation filter architecture is adopted for the half-pixel and quarter-pixel interpolations to store the intermediate result and thus can reduce the hardware cost.
(3) An efficient memory organization method is proposed in the paper to reduce the data access of SRAM and save the power of VLSI architecture.
(4) A five-step pipeline interpolation filter engine is proposed in the paper. It can shorten the critical path of the filter and improve the working speed.
(5) A reconfigurable interpolation unit is developed in the paper, and the two types of the filters can be carried out with the same hardware architecture by only reversing the order of input reference pixels. As a result, the proposed reconfigurable

filter can reduce the area of the whole architecture.

The rest of this paper is organized as follows. Section 2 presents the overview of the interpolation filter algorithm in the HEVC test model (HM). Section 3 describes the improved fast interpolation filter algorithm. The proposed efficient interpolation filter VLSI architecture is presented in Section 4 in details. The implementation results are analyzed in Section 5. Finally, Section 6 concludes this paper.

2 Overview of interpolation algorithm

In HEVC standard, three different eight-tap and seven-tap interpolation FIR filters are used for both half-pixel and quarter-pixel interpolations. These three FIR filters, i.e., type A, type B, and type C, are shown in (1), (2), and (3), respectively.

$$a_{0,0} = (-A_{-3,0} + 4 \times A_{-2,0} - 10 \times A_{-1,0} + 58 \times A_{0,0} + 17 \times A_{1,0} - 5 \times A_{2,0} + A_{3,0} + 32) >> 6 \quad (1)$$

$$b_{0,0} = (-A_{-3,0} + 4 \times A_{-2,0} - 11 \times A_{-1,0} + 40 \times A_{0,0} + 40 \times A_{1,0} - 11 \times A_{2,0} + 4 \times A_{3,0} - A_{4,0} + 32) >> 6 \quad (2)$$

$$c_{0,0} = (A_{-2,0} - 5 \times A_{-1,0} + 17 \times A_{0,0} + 58 \times A_{1,0} - 10 \times A_{2,0} + 4 \times A_{3,0} - A_{4,0} + 32) >> 6 \quad (3)$$

Integer pixels ($A_{x,y}$), half pixels ($b_{x,y}$, $h_{x,y}$, $j_{x,y}$), and quarter pixels ($a_{x,y}$, $c_{x,y}$, $d_{x,y}$, $e_{x,y}$, $f_{x,y}$, $g_{x,y}$, $i_{x,y}$, $k_{x,y}$, $p_{x,y}$, $q_{x,y}$, $r_{x,y}$, $n_{x,y}$) in a PU are shown in Fig. 1. The half pixels are interpolated from the nearest integer pixels in either horizontal direction or vertical direction. The quarter pixels are interpolated from the nearest half pixels in the horizontal direction and in the vertical direction, respectively, using type A, type B, or type C filter. According to which fractional pixel should be computed, one interpolation filter is chosen. As the position of a quarter pixel point is close to the integer pixel, we can choose a seven-tap interpolation filter. But for the farther half-pixel point, an eight-tap filter is required.

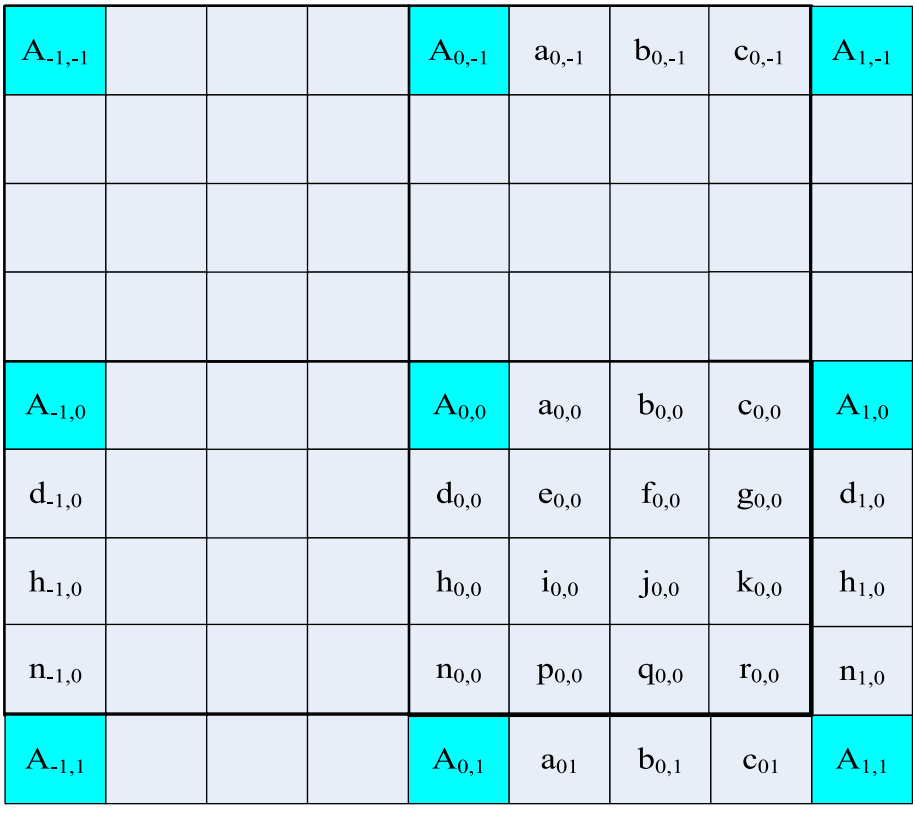

Fig. 1 Integer, half, and quarter pixels. The positions of integer pixels, half pixels, and quarter pixels in a PU. Variable ($A_{x,y}$) represents integer pixels. Variables ($b_{x,y}$, $h_{x,y}$, $j_{x,y}$) represent half pixels. Variables ($a_{x,y}$, $c_{x,y}$, $d_{x,y}$, $e_{x,y}$, $f_{x,y}$, $g_{x,y}$, $i_{x,y}$, $k_{x,y}$, $p_{x,y}$, $q_{x,y}$, $r_{x,y}$, $n_{x,y}$) represent quarter pixels

For the half-pixel point interpolation, $b_{0,0}$ can be calculated by applying (2) from $A_{0,0}$ in the horizontal direction. Then, $h_{0,0}$ and $j_{0,0}$ can be calculated from $A_{0,0}$ and $b_{0,0}$, respectively, in the vertical direction.

After that, the motion vector (MV) can be obtained by using the SATD algorithm [1] to calculate the values of the half-pixel points. For the quarter-pixel point interpolation, $a_{0,0}$ and $c_{0,0}$ can be calculated by applying (1) and (3) from $A_{0,0}$ in the horizontal direction. Then, $e_{0,0}$, $p_{0,0}$, $g_{0,0}$, and $r_{0,0}$ can be calculated from $a_{0,0}$ and $c_{0,0}$, respectively, in the vertical direction, while the other points should be filtered according to the MV.

If the horizontal component of MV is equal to zero, $d_{0,0}$ and $n_{0,0}$ can be calculated by applying (1) and (3) from $A_{0,0}$; otherwise, $f_{0,0}$ and $q_{0,0}$ will be calculated by applying (1) and (3) from $b_{0,0}$. If the vertical component of MV is equal to zero, $a_{0,0}$ and $c_{0,0}$ can be calculated from themselves; otherwise, $i_{0,0}$ and $k_{0,0}$ will be calculated by applying (2) from $a_{0,0}$ and $c_{0,0}$.

For 8×8 sub-block predictions, the reference pixel values of a 16×16 prediction block are required in the worst-case scenario. Compared to the six-tap interpolation filter in H.264/AVC, the interpolation filter in HEVC will cost a larger area. Therefore, it is very important to design an efficient luma interpolation VLSI architecture to realize the implementation of real-time video coding and to reduce the implementation area.

3 The fast interpolation filter algorithm

3.1 Fast interpolation filter algorithm

Like H.264/AVC, mode decisions with motion estimation (ME) remain to be among the most time-consuming computations in HEVC. In the initial HEVC design, there are four different possible partition modes for inter predictions: two square partition modes ($2N \times 2N$ and $N \times N$) and two symmetric motion partition (SMP) modes ($2N \times N$ and $N \times 2N$). As a complement to the square-shaped or non-square symmetrically partitioned prediction blocks, the asymmetric motion partition (AMP) is proposed in HEVC. AMP includes four partition modes: $2N \times nU$, $2N \times nD$, $nR \times 2N$, and $nL \times 2N$, which divide a coding block into two asymmetric prediction blocks along the horizontal or vertical direction. In HEVC, the size of the largest PU is 64×64. So, it can be split into a total of 21 different sizes of sub-PUs, as shown in Table 1 (there is no 4×4 mode in the interpolation filtering operation). All possible prediction modes are traversed. And the one having the minimum RD cost will be used.

Table 1 The inter-prediction splitting modes

Number	The size of sub-PU
4	8×4, 4×8, 4×4
8	16×8, 8×16, 8×8, 16×4, 16×12, 4×16, 12×16
16	32×16, 16×32, 16×16, 32×8, 32×24, 8×32, 24×32
32	64×64, 64×32, 32×64, 32×32

A total of 21 inter-prediction splitting modes of sub-PUs

In an inter-prediction mode decision, a full-search algorithm searches for every possible block size and refines the results from the integer-pel to quarter-pel resolution. Thus, a full-search algorithm guarantees the highest level of compression performance. However, the considerable computational complexity for a mode decision is critical for the encoding speed. Moreover, the main target resolution of HEVC is full HD (1920×1080) and beyond. For hardware implementation of an HEVC encoder, the area cost will be very high if the hardware structure executes interpolation filter for all possible prediction modes. In the VLSI architecture design, therefore, it is required to achieve the interpolation filtering operation of larger blocks by reusing the smallest unit.

According to eight different possible splittings of PUs, a 4-pixel interpolation unit and an 8-pixel interpolation unit are used in the proposed architecture. The splitting modules for the 4-pixel interpolation unit include 4×8, 4×16, 8×4, and 16×4 modes. Both 4-pixel interpolation unit and 8-pixel interpolation unit will be used for 12×16 and 16×12 modes. So the 4-pixel interpolation unit mentioned in this paper also includes 12×16 and 16×12 modes. The 4-pixel interpolation unit is capable of processing every sub-block in a coding unit (CU), but it will cost more hardware areas and clock cycles. So it is very difficult for a 4-pixel interpolation unit to achieve the real-time processing of interpolation filter with reasonable computing powers. The statistics of possible splittings of PUs in HM 13.0 with low delay configuration is shown in Table 2. The size ranges from 64× to 4×. The 64× size includes 64×32 and 64×64 modes. The 32× size includes 32×8, 32×16, 32×24, 24×32, and 32×32 modes. The 16× size includes 16×8, 16×16, and 16×32 modes. The 8× size includes

Table 2 The statistics of possible splitting of PUs

	Class A	Class B	Class C	Class D	Class E
Size	Traffic	Cactus	BasketballDrill	Keiba	Johnny
64×	5361	2663	454	88	1703
32×	9945	5751	937	255	1229
16×	19,654	8994	2307	876	2193
8×	28,408	9261	4232	1409	2098
4×	2582	502	426	150	97
Total	65,950	27,171	8356	2778	7320

The statistics of possible splitting of PUs in HM 13.0 with low delay configuration. 64×, 32×, 16×, 8×, and 4× represent the size of PUs. In terms of resolutions, the video sequences are classified into five classes, including class A (2560×1600), class B (1920×1080), class C (832×480), class D (416×240), and class E (1280×720). Traffic, Cactus, BasketballDrill, Keiba, and Johnny represent five video sequences

8 × 8, 8 × 16, and 8 × 32 modes. The 4× size includes 4 × 8, 4 × 16, 8 × 4, 16 × 4, 12 × 16, and 16 × 12 modes.

It can be observed from Table 2 that, the splitting modules for a 4-pixel interpolation unit (4× size) are only about 3.52 % of all possible splitting of PUs, so it will only cause a small decrease in image quality to skip them. In addition, it needs 4 + 8 = 12 pixels data width to process the 4-pixel interpolation unit, and the valid data percentage is only 33 %. If we can skip the 4× size, using 8-pixel interpolation unit as the basic unit, the valid data can account for 50 %. Therefore, it can save a large area and clock cycles in hardware implementation by skipping the 4× size.

Therefore, we propose a fast and implementation-friendly interpolation algorithm in which the interpolation processing with a 4-pixel interpolation unit will be skipped. If we use the 8-pixel interpolation unit, we will skip the 4× basic blocks' (i.e., 4 × 8, 4 × 16, 8 × 4, 16 × 4, 16 × 12, and 12 × 16) interpolation operation in HEVC. Figure 2 illustrates the top-level block diagram of our proposed fast interpolation algorithm. Compared to the original algorithm, the interpolation process of 4 × 8, 4 × 16, 8 × 4, 16 × 4, 16 × 12, and 12 × 16 sub-PU blocks is skipped in the interpolation.

Based on the proposed fast interpolation algorithm, we re-arrange the classification of PU splitting modules, as shown in Table 3. According to the new splitting modules and the proposed fast algorithm, we can put the minimum 8× PU modes together to realize the interpolation of larger blocks in the VLSI design.

Table 3 The new splitting module in interpolation filter

PU module	The size of sub-PU
8×	8 × 8, 8 × 16, 8 × 32
16×	16 × 8, 16 × 16, 16 × 32
24×	24 × 32
32×	32 × 8, 32 × 16, 32 × 24, 32 × 32, 32 × 64
64×	64 × 32, 64 × 64

Based on the proposed fast interpolation algorithm

3.2 Experiment results

In order to evaluate the performance of the proposed fast interpolation algorithm, we implement the algorithm using the recent HEVC reference software (HM 13.0). A PC with Inter Core™i5-2400K CPU @ 3.1GHz and 4-G RAM is used in the experiments. We compare the proposed algorithm and our previous work [18] in a low complexity configuration with the original algorithm in HM 13.0 encoder. The performance of the proposed algorithm is shown in Table 4.

A set of experiments are carried out for IPPP frame sequences in which CABAC is used as the entropy coder. The proposed algorithm is evaluated with QPs 22, 27, 32, and 37 using 13 typical sequences recommended by the JCT-VC in five resolutions [19]. In terms of resolutions, the video sequences are classified into five classes, including class A (2560 × 1600), class B (1920 × 1080), class C (832 × 480), class D (416 × 240), and class E (1280 × 720). Coding efficiency is measured in terms of peak signal-to-noise ratio (PSNR) and bit rate. Computational complexity is measured by the consumed coding time. Bjøntegaard delta PSNR (BDPSNR) (dB) and Bjøntegaard delta bit rate (BDBR) (%) are used to represent the average PSNR and bit rate difference [20]. "Time save (%)" is used to represent the coding time change of motion estimation in percentage. The positive and negative values represent increments and decrements, respectively.

Table 4 shows the comparison of our previous work [18] and the proposed new interpolation algorithm as compared to the original algorithm in HM 13.0 encoder. For the five classes (A, B, C, D, and E) of test sequences, the proposed new algorithm can greatly reduce the coding time of motion estimation. The proposed algorithm can achieve about 19.7 % motion estimation time reduction with a maximum of 21.9 % in "PeopleOnStreet (2560 × 1600)" and a minimum of 17.1 % in "Flowervase (832 × 480)". Our previous algorithm in [18] can only achieve about 10.0 % total coding time reduction with a maximum of 11.1 % in "Racehorses (832 × 480)" and a minimum of 9.0 % in "Traffic (2560 × 1600)". Compared with our previous algorithm in [18] which only the interpolation process of 4 × 8, 4 × 16, and 12 × 16 blocks are skipped, it can achieve a higher time saving. About 8.1–11.5 % encoding time can be further reduced, while

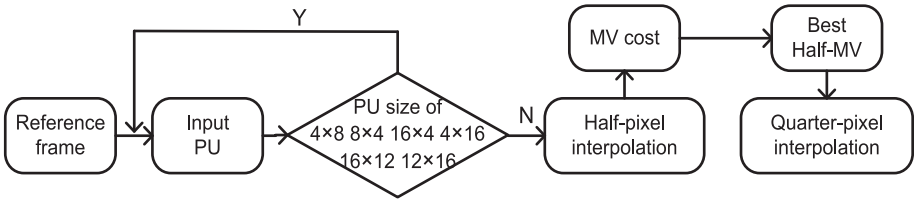

Fig. 2 The improved luma interpolation algorithm. The top-level block diagram of our proposed fast interpolation algorithm. If the size of input PU is 4 × 8, 4 × 16, 8 × 4, 16 × 4, 16 × 12, or 12 × 16, the interpolation process of this PU will be skipped. The interpolation process includes half-pixel interpolation, MV cost calculation, best half MV determination, and quarter-pixel interpolation

Table 4 Comparisons of our previous works [18] and proposed new algorithm compared to HEVC

Class	Sequence	BDPSNR (dB)		BDBR (%)		Time save (%)	
		[18]	New	[18]	New	[18]	New
A	PeopleOnStreet	−0.0458	−0.0492	1.14	1.26	10.4	21.9
	Traffic	−0.0165	−0.0211	0.58	0.62	9.0	19.4
B	BasketballDrive	−0.0046	−0.0136	0.16	0.30	10.2	19.9
	BQTerrace	−0.0035	−0.0132	0.21	0.53	9.6	18.9
	Cactus	−0.0156	−0.0299	0.69	1.35	10.0	19.6
C	Racehorses	−0.0866	−0.0936	2.36	2.43	11.1	21.3
	Flowervase	−0.0443	−0.0961	1.36	1.41	9.0	17.1
	BasketballDrill	−0.0275	−0.0326	0.77	0.91	10.6	18.9
D	BasketballPass	−0.1071	−0.1264	2.46	2.59	10.8	21.0
	BlowingBubbles	−0.0739	−0.1131	2.14	2.43	10.5	21.0
	Keiba	−0.0357	−0.1109	0.76	0.97	10.9	21.4
E	Johnny	−0.0272	−0.0314	1.08	1.22	9.1	17.4
	KristenAndSara	−0.0268	−0.0392	0.90	1.23	9.0	17.2
Average		−0.0396	−0.0594	1.12	1.38	10.0	19.7

BDPSNR (dB) and BDBR (%) are used to represent the average PSNR and bit rate difference. "Time save (%)" is used to represent the coding time change of motion estimation in percentage. The positive and negative values represent increments and decrements, respectively

the average coding efficiency loss is small, less than 0.26 % BDBR increase. For those sequences with higher resolutions (such as 1920 × 1080 and 2560 × 1600), the proposed algorithm shows impressive improvement with more than 19.9 % of coding time saved. Therefore, it is especially efficient for coding higher-resolution video. In [21], a simplified HEVC FME interpolation unit targeting a low-cost and high-throughput hardware design was proposed and it causes a bit rate loss of about 13.18 % and a quality loss of about 0.45 dB. Compared to the works in [21], our algorithm provides significant improvement in terms of PSNR and bit rate.

The gain of our algorithm is high because unnecessary small PU size decision has been skipped. For sequences with large smooth texture areas like "PeopleOnStreet", the algorithm saves more than 20 % coding time. On the other hand, coding efficiency loss is acceptable in Table 4, where the average BDBR increase is 1.38 % with the minimum of 0.3 % in "BasketballDrive" and the average BDPSNR decrease is 0.0594 dB with the minimum of 0.0132 dB in "BQTerrace". The above experimental results indicate that the proposed new algorithm is efficient for all types of video sequences and outperforms our previous algorithm [18] for HEVC encoders. Therefore, the proposed algorithm can efficiently reduce the coding time while keeping nearly the same RD performance as the original HM encoder. What is more, it can also reduce the implementation area cost in the VLSI design.

4 The efficient interpolation filter VLSI architecture

4.1 The reused data path of interpolation

Fractional motion estimation performs a half-pixel refinement about the integer search positions, and then a quarter-pixel one is performed around the best half-pixel position. In the interpolation algorithm described in Section 2, it is known that the quarter-pixel interpolation processor needs to filter the results of the half-pixel horizontal interpolation in a vertical direction. If carrying out the interpolation process of a 64 × 64 CU, 2 × (64 + 1) × (64 + 8) × (8 + 6) = 131,040 bits RAM is required in total. The area cost will be huge for hardware implementation. In our design, a reused three-level architecture is proposed for half-pixel and quarter-pixel interpolations. With this structure, we would not need to store the intermediate results and thus can reduce the area cost for about 131,040 bits RAM.

Figure 3 shows the data path of the interpolation processor. There are three horizontal filters (H_F1/4, H_F2/4, H_F3/4 in level 1) and eight vertical filters (V_F1/4, V_F2/4, V_F3/4 in level 2 and level 3) in the proposed three-level reused architecture.

There are three horizontal filters in the first level (level 1). For the half-pixel interpolation as shown in Fig. 3a, the horizontal filter H_F2/4 is open and the other two are close in the first round. The half-pixel $b_{0,0}$ (as seen in Fig. 1) is calculated by H_F2/4 from the integer pixel $A_{0,0}$ in the horizontal direction.

For the quarter-pixel interpolation in the second round as shown in Fig. 3b, the filtered results of pixels $a_{0,0}$, $b_{0,0}$, and $c_{0,0}$ are calculated by the three horizontal filters in level 1 from the integer position $A_{0,0}$.

The second level (level 2) contains four vertical filters. They work just at the second round of the quarter-pixel interpolation process. The quarter pixels $e_{0,0}$ and $p_{0,0}$ are interpolated by the filters V_F1/4 and V_F3/4, respectively, from the pixel $a_{0,0}$ in the vertical direction. Similarly, the quarter pixels $g_{0,0}$ and $r_{0,0}$ are interpolated, respectively, by the filters V_F1/4 and V_F3/4 from the pixel $c_{0,0}$ in the vertical direction.

The last level (level 3) also contains four vertical filters. The difference between the four vertical filters in level 2 and level 3 is that the data inputs of the vertical filters in level 3 are not fixed. The filtered results of the half pixels $h_{0,0}$ and $j_{0,0}$ are calculated by the two vertical filters V_F2/4 from the pixels $A_{0,0}$ and $b_{0,0}$ at the first round of the half-pixel interpolation process. During the second round, the quarter pixels $i_{0,0}$ and $k_{0,0}$ are interpolated by the same two vertical filters from the pixels $a_{0,0}$ and $c_{0,0}$ when the vertical

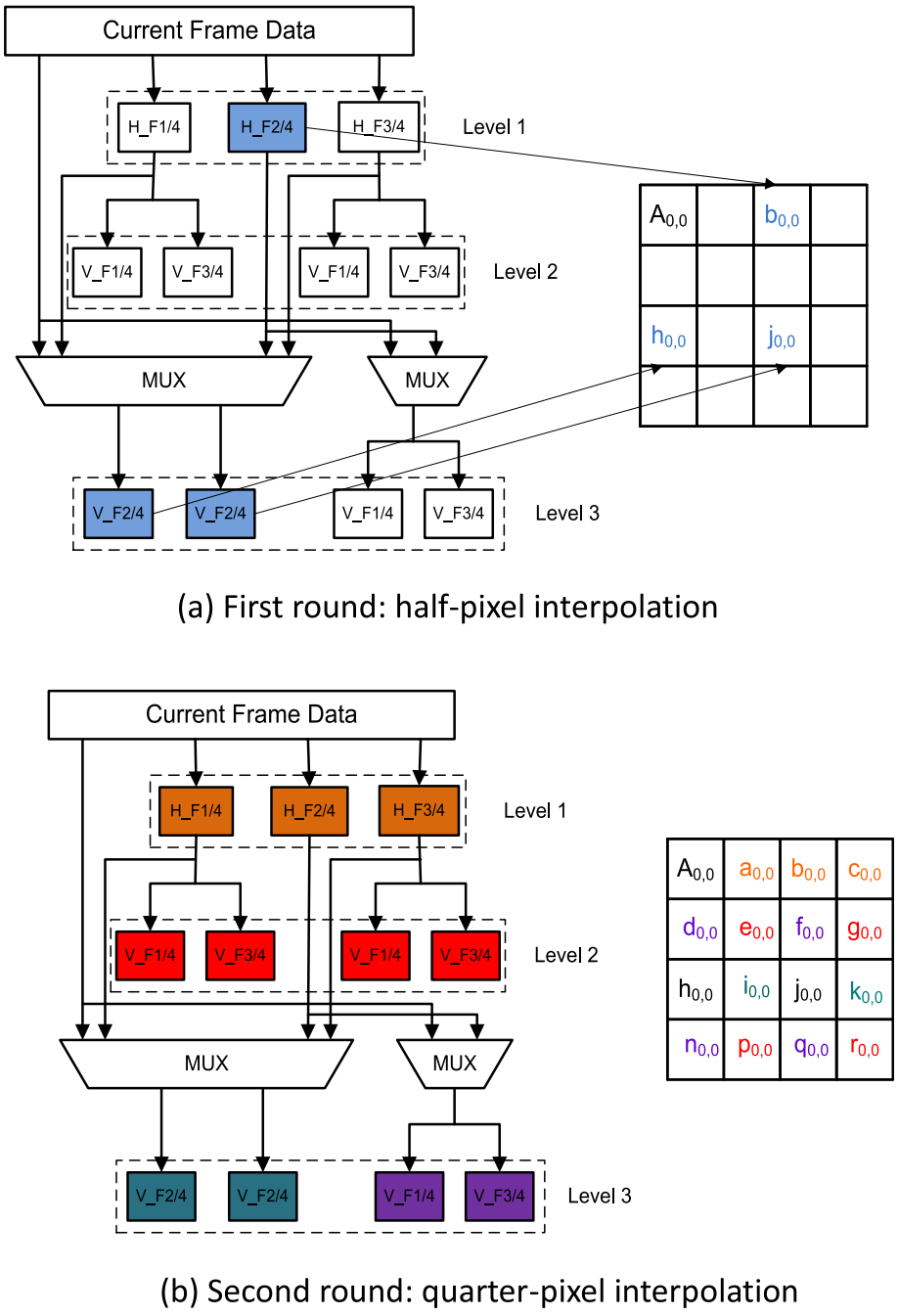

Fig. 3 The reused data path of interpolation filter. **a** First round: half-pixel interpolation. **b** Second round: quarter-pixel interpolation. The reused data path of the interpolation processor. **a** The data path of the first round of interpolation processor for half-pixel interpolation. **b** The data path of the second round of interpolation processor for quarter-pixel interpolation. H_F1/4, H_F2/4, and H_F3/4 in level 1 represent three horizontal filters. V_F1/4, V_F2/4, and V_F3/4 in level 2 and level 3 represent eight vertical filters. MUX represents multiplexor

component of the best half MV is not equal to zero. The interpolated results of quarter pixels $d_{0,0}$ and $n_{0,0}$ are calculated by the other two vertical filters V_F1/4 and V_F3/4 from the integer pixel $A_{0,0}$ when the horizontal component of MV is equal to zero; otherwise, the quarter pixels $f_{0,0}$ and $q_{0,0}$ are interpolated by the same vertical filters V_F1/4 and V_F3/4 from the half pixel $b_{0,0}$.

From the above data path of the proposed interpolation filter architecture, it can be seen that all the horizontal and vertical filters in the process of half-pixel interpolation can be reused in the process of quarter-pixel interpolation.

Moreover, some interpolation filter units can be reused for different quarter-pixel positions. This reused architecture will greatly reduce the area cost in hardware implementation.

4.2 Memory organization

In the VLSI design, an 8-pixel interpolation unit is applied to balance the processing time and the hardware efficiency. Because every PU can be split into multiple 8× blocks, the 8-pixel interpolation unit can deal with every sub-block in the processing unit of inter-prediction.

Extra four pixels around every 8 × 8 block will be used as the input pixels for the 8-tap interpolation filter. So the window of filter should be (4 + 8 + 4) × (4 + 8 + 4) = 16 × 16 and the width of the input data is 16 pixels. The scan order is vertical, and the adjacent 8 × 8 blocks adopt similar operations to reuse the interpolation data and to reduce the memory access.

The basic filter unit is 8× block which will be reused many times for 8 × 8 blocks and larger PU blocks. The reference data inputs should be reasonably stored before the process of sub-pixel interpolation to reduce the memory access. SRAM is used to store the input reference pixels. The maximum processing unit of LPU is 64 × 64 block, and there are also four extra reference pixels around the processing unit. So the actual reference pixel matrix is 72 × 72. As the width of processing unit is from 8 to 64, the 72 × 72 pixel matrix is stored in terms of 9-pixel width separately as shown in Fig. 4. The

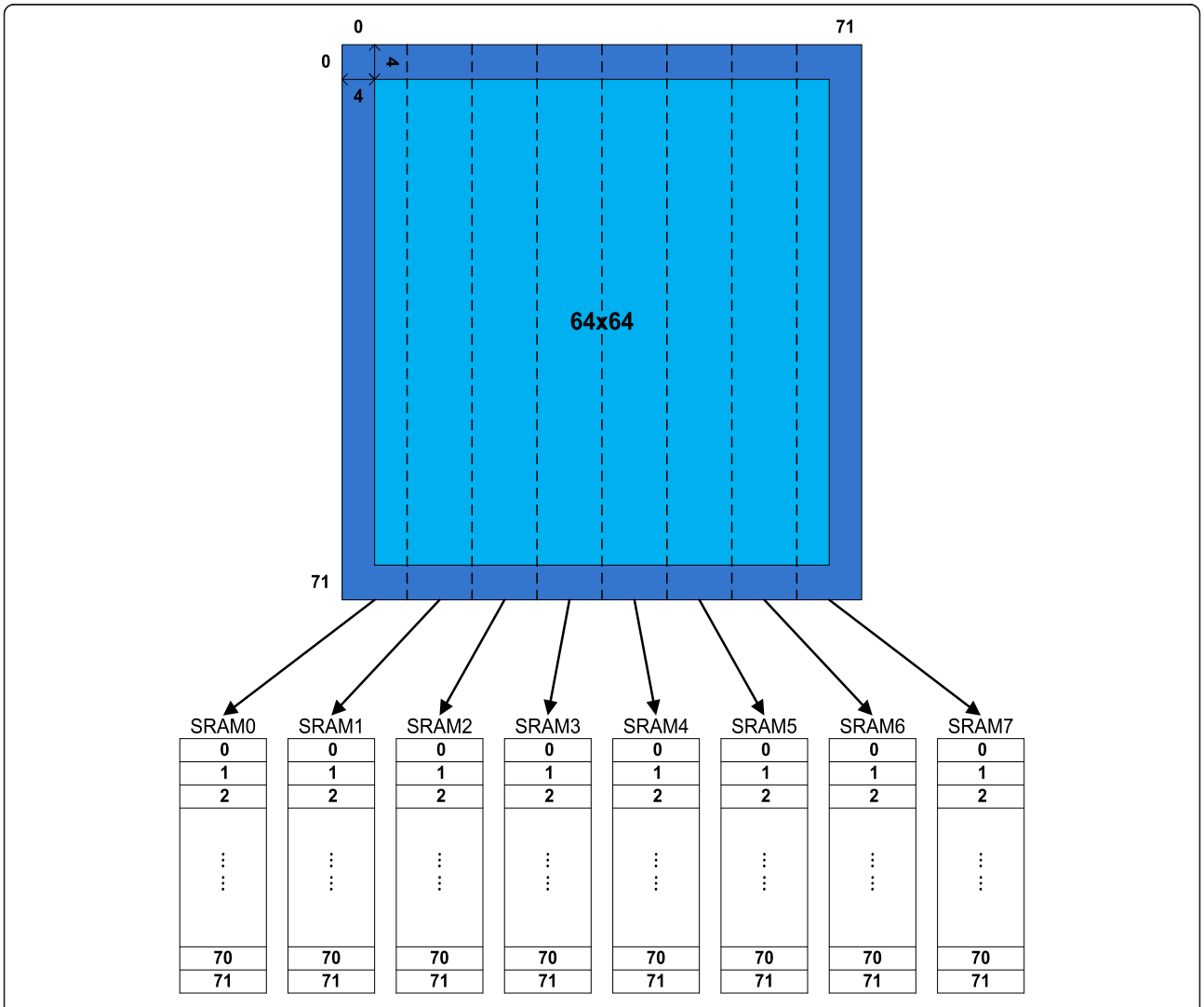

Fig. 4 Memory organization. Memory organization of the proposed architecture. The maximum processing unit of LPU is 64 × 64 block, and there are also four extra reference pixels around the processing unit. SRAM0–SRAM7 represent eight SRAMs in order to realize the storage of a 72 × 72 pixel matrix. The depth of every SRAM is 9 × 8 bit = 72 bits, and every bit is the data address of each line

depth of every SRAM is 9 × 8 bit = 72 bits, and every bit is the data address of each line. There are eight SRAMs in order to realize the storage of a 72 × 72 pixel matrix. Based on this organization, only SRAM0 and SRAM1 are open for 8× processing unit while the others are close with no data access. Only when the width of processing unit is 64, all the SRAMs will be used to store and read the input reference pixels.

4.3 The reconfigurable interpolation filter architecture

4.3.1 The pipeline interpolation filter engine

According to the analysis in Section 3, the 8-pixel interpolation unit is chosen as the basic unit in the proposed interpolation processor. The proposed pipeline interpolator architecture is shown in Fig. 5 where the 8× block module is the basic reused block. This interpolator can support 8-pixel interpolation, which can adapt to most of the variable block sizes. One 16 × 16 block is split into two 8 × 16 blocks, and a 16 × 8 block is split into two 8 × 8 blocks. For the interpolation process of a 64 × 64 CU, 8× block module can be reused by eight times.

As shown in Fig. 5, h_f is the 8-tap horizontal interpolation filter and v_f is the 8-tap vertical interpolation filter. The h_f can support 8-pixel interpolation. There are nine 8-tap horizontal interpolation filters (h_f0~h_f8), and only eight filtered results among them are selected as the predicted outputs according to the distribution of half pixels around the integer pixels. After a horizontal interpolation filtering, the vertical interpolation filter reads the horizontal outputs. There are eight shift registers in the vertical interpolation filter, and the output data from the horizontal filter are stored in these registers sequentially. When the eight registers are filled with the predicted outputs from the horizontal interpolation filter, the vertical interpolation filter starts to work.

There are five steps in the operation of the interpolation filter pipeline.

Step 1: The interpolation filter reads the reference integer pixels from the first line, and as a result, there are 16 reference data inputs from 0~15.

Step 2: The horizontal interpolation filter h_f0 reads the integer pixels 0~7 of line 1, and the filter h_f1 reads the integer pixels 1~8, and so on. These 16 pixels are interpolated by the corresponding horizontal interpolation filters.

Step 3: The filtered data from the horizontal interpolation filter of line 1 are written into the registers of the vertical interpolation filter v_f. By repeating the same operations as in step 1 and step 2, the filtered data of following lines are written into the registers.

Step 4: When the registers of v_f are filled with eight pixels, the 8-tap vertical interpolation starts to work and the filtered results of line 1 will be obtained.

Step 5: When v_f executes filtering from line 1 to line 8, the input reference pixels of line 9 are interpolated by the horizontal interpolation filter h_f simultaneously. After the filtered data of line 9 are written into the register of v_f and the filtered results of line 1 are released by the register, the vertical interpolation filter starts to

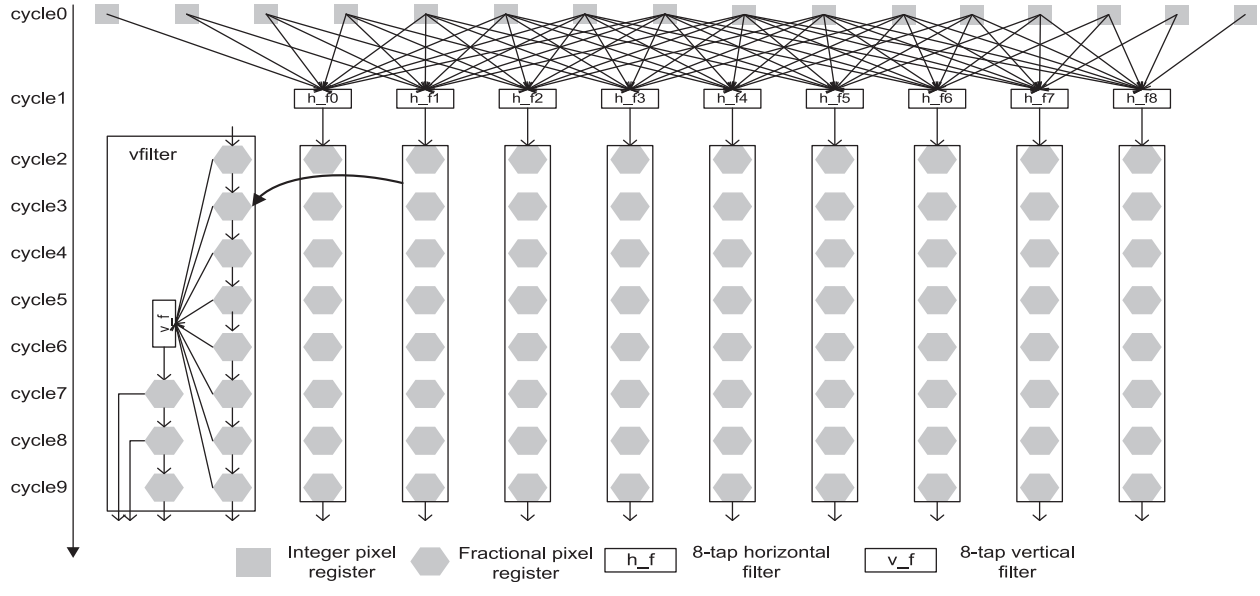

Fig. 5 The pipeline 8-pixel interpolation filter engine. The proposed pipeline interpolator architecture. *h_f* represents the 8-tap horizontal interpolation filter, and *v_f* represents the 8-tap vertical interpolation filter

execute the filtering operation on the reference pixels from line 2 to line 9.

According to the five steps above, the 8× block interpolation engine performs the pipeline filtering operations and the ultimate interpolation filtered result will be obtained after one clock cycle.

4.3.2 The reconfigurable interpolation unit

Table 5 shows the coefficients of an 8-tap filter. It can be observed that the coefficients of A and C type are symmetry. Therefore, the interpolation of A and C type filters can be carried out with the same hardware architecture by only reversing the order of input reference pixels.

The interpolation performed by the seven-tap or eight-tap interpolation filter (h_f or v_f in Fig. 5) is implemented by shifters and additions and subtraction operations. Based on (1) and (2), the 8-tap filter needs 33 adders (12 adders for A type, 9 adders for B type, and 12 adders for C type) and 14 shifters (5 shifters for type A, 4 shifters for B type, and 5 shifters for C type) in the hardware implementation.

The proposed optimal architecture of A and B filters are shown in Fig. 6 where A, B, C, D, E, F, G, and H are eight input reference pixels. The structures of horizontal and vertical filters are identical. Compared to the above 33 adders and 14 shifters, the proposed architecture of A and B type filters only needs 19 adders (10 adders for A type and 9 adders for B type) and 8 shifters (4 shifters for A type and 4 shifters for B type) to realize the hardware implementation. Since only one type of the three filters is used at one time, the interpolation of A and C type filters can be carried out with the same hardware architecture by only reversing the order of input reference pixels. As a result, the proposed reconfigurable filter can reduce the area of the whole architecture.

5 Implementation results

The proposed interpolation filter architecture is implemented in Verilog HDL and synthesized using SMIC 90-nm cell library. Table 6 shows the implementation comparison between the proposed and state-of-the-art designs, as well as our previous work [18]. When synthesized with 90-nm CMOS standard library, the total gate count of this design is 37.2k for supporting 7680 × 4320@78fps (4:2:0 format) videos and real-time processing with a working clock speed of 240 MHz.

Table 5 Three types of 8-tap filters

Type	Coefficient
A	[−1, 4, −10, 58, 17, −5, 1]
B	[−1, 4, −11, 40, 40, −11, 4, −1]
C	[1, −5, 17, 58, −10, 4, −1]

Shows the coefficients of A type, B type, and C type 8-tap filter

In terms of hardware resources, the proposed architecture can reduce about 18 % area compared to the works in [11]. Although the works in [12] has eight times greater parallelism and can work at higher frequencies than the design in this paper, the amount of logic resources is also six times greater. The proposed architecture also allows for the use of a reduced input buffer so that the memory cost can be reduced by 131,040 bits. Compared with the works in [13, 16], due to the different targeted video specification, our design consumes more hardware cost. Although hardware cost in [16] is only 28.5k, memories are not included in the gates' area. The reconfigurable hardware accelerator engines in [14, 15] are synthesized for the FPGA device and BRAMs are used, so it is difficult to make a fair hardware cost comparison with the works in [14, 15].

In terms of performance, the throughput of the proposed architecture is 13.4 pixels/cycle, which is almost 18 times larger than the works in [13] with 0.73 pixel/cycle, 6 times larger than the works in [11] with 2.58 pixels/cycle, and 2 times larger than the works in [15] with 8.5 pixels/cycle. So the hardware implementation can better adapt to the HEVC standard with larger LCU size. Consequently, the hardware implementation cost of our architecture is comparable to H.264/AVC.

At 0.9 V power supply, the processing of the proposed hardware for 7680 × 4320@78fps video processing dissipates only 4.7 mW when running at 240 MHz. The power consumptions shown in [14, 15] are measured by the FPGA-based evaluation system, and the power consumption shown in [17] also includes the energy consumption of fractional search module. It is quite unfair to make a power comparison with them. The power consumptions of interpolation hardware in [11–13] are not shown.

From Table 6, it also can be seen that only the proposed architecture and the works in [17] can support 8K-UHD video real-time encoding. The hardware costs of the works in [17] are 1183k gates, and fractional search module is also included. Therefore, it is difficult to make a fair comparison of hardware resources. For the processing speed of the design, the throughput of the proposed architecture is almost 2.5 times larger than the works in [17] with 5.26 pixels/cycle. The design delivers a maximum throughput of 2588 Mpixels/s for 7680 × 4320 78 frames/s video application, and the works in [17] can only achieve a maximum throughput of 995 Mpixels/s for 7680 × 4320 30 frames/s video coding. Compared to the

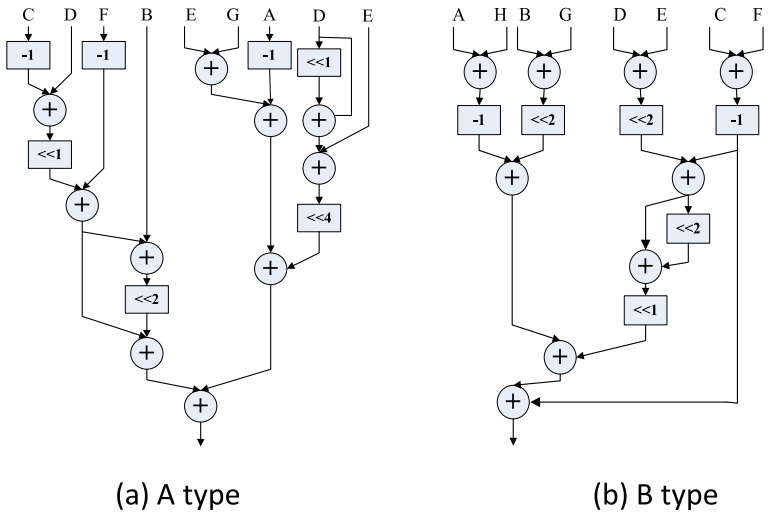

Fig. 6 The proposed architectures of A and B type filters. **a** A type. **b** B type. Shows the proposed optimal architecture of A and B type filters. **a** The architecture of A type filters. **b** The architecture of B type filters. *A*, *B*, *C*, *D*, *E*, *F*, *G*, and *H* represent eight input reference pixels. "<<1" and "<<2" represent shifters. "+" represents adder. "−1" represents "multiplied by minus one"

works in [17], our architecture provides significant speed improvement.

Table 6 also shows the implementation comparison between the proposed architecture and our previous works in [18]. Our previous works in [18] can only support 4K-UHD video real-time encoding with the frame rate of 47, and the proposed architecture can support 8K-UHD video real-time encoding with the frame rate of 78. The proposed architecture can also achieve about 42 % area reduction and 40 % power reduction compared to our previous works in [18]. These speed and cost improvement come from both the new fast interpolation filter algorithm and hardware efficiency.

6 Conclusions

In this paper, high-performance VLSI architecture for luma interpolation in HEVC is proposed and it is implemented with 37.2k gates at an operating frequency of 240 MHz. It can support 8K-UHD (7680 × 4320)@78fps (4:2:0 format) real-time video processing. Our proposed architecture can be reused for half-pixel interpolation and quarter-pixel interpolation, and it reduces the area cost about 131,040 bits RAM with the reused interpolation architecture. Our proposed architecture can achieve high throughput for real-time encoding of ultra high-resolution videos with reduced hardware resources and is especially suitable for 8K-UHD video real-time encoding.

Table 6 Comparisons between the proposed architecture and state-of-the-art designs

	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	Proposed architecture
Standard	HEVC	HEVC	HEVC	HEVC	HEVC	HEVC	HEVC	HEVC	HEVC
Technology (nm)	40	90	90	FPGA 65	FPGA 65	90	65	90	90
Parallelism	8×	64×	8×	8×	8×	8×	8×	8×	8×
Logic gate account	45.2k	211.693k	32.496k	5710 LUTs	5017 LUTs	28.5k[a]	1183k[b]	64.5k	37.2k
Power (mW)	N/A	N/A	N/A	379	89	N/A	198.6	7.9	4.7
Interpolation execution time (pixel/cycle)	2.58	N/A	0.73	N/A	8.5	N/A	5.26	0.84	13.4
Max operation frequency (MHz)	200	400	171	403	200	200	188	193	240
Throughput	QFHD @30fps	1080p @30fps	QFHD @60fps	QFHD @60fps	QFHD @30fps	QFHD @30fps	8K-UHD @30fps	QFHD @47fps	8K-UHD @78fps

N/A not available
[a]Excluding on-chip memories
[b]Fractional search module included

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This work was supported in part by the National Natural Science Foundation of China (60902101), New Century Excellent Talents in University of Ministry of Education of China (NCET-11-0824), and Fundamental Research Funds for the Central Universities (3102014JCQ01057).

Author details
[1]School of Electronics and Information, Northwestern Polytechnical University, Xi'an 710072Shaanxi, China. [2]School of Automation, Northwestern Polytechnical University, Xi'an 710072, China. [3]Research Institute of Information Technology, Tsinghua University, Beijing 100084, China. [4]Complex System Inc, Tsinghua University, Calgary, Alberta T2L2K7, Canada.

Received: 1 April 2015 Accepted: 10 November 2015
Published online: 17 November 2015

References

1. GJ Sullivan, JR Ohm, WJ Han et al., Overview of the high efficiency video coding (HEVC) standard. IEEE Trans. Circuits Syst. Video Technol. 22(12), 1649–1668 (2012)
2. J Ohm, GJ Sullivan, High efficiency video coding: the next frontier in video compression [standards in a nutshell]. Signal Process Magazine IEEE 30(1), 152–158 (2013)
3. J-R Ohm, GJ Sullivan, H Schwarz, TK Tan, T Wiegand, Comparison of the coding efficiency of video coding standards—including high efficiency video coding (HEVC). IEEE Trans. Circuits Syst. Video Technol. 22(12), 1669–1684 (2012)
4. Y. J. Ahn, W. J. Han, D. G. Sim, "Study of decoder complexity for HEVC and AVC standards based on tool-by-tool comparison," SPIE Appl. Digital Image Process. XXXV, 8499, 84990X-1-84990X-10 (2012)
5. T-C Chen, S-Y Chien, Y-W Huang, C-H Tsai, C-Y Chen, TW Chen, L-G Chen, Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder. IEEE Trans. Circuits Syst. Video Technol. 16(6), 673–688 (2006)
6. Z Liu, Y Song, M Shao, S Li, L Li, S Ishiwata, M Nakagawa, S Goto, T Ikenaga, HDTV1080p H.264/AVC encoder chip design and performance analysis. IEEE J. Solid-State Circuits 44(2), 594–608 (2009)
7. C Yang, S. Goto, T. Ikenaga, in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*. High performance VLSI architecture of fractional motion estimation in H.264 for HDTV (IEEE, Kos, Greece, 2006)
8. S. Oktem and I. Hamzaoglu, in *Proc.10th Euromicro Conference on Digital System Design*. An efficient hardware architecture for quarter-pixel accurate H.264 motion estimation (IEEE, Luebeck, Germany, 2007)
9. G Pastuszak, M Jakubowski, Adaptive computationally-scalable motion estimation for the hardware H.264/AVC encoder. IEEE Trans. Circuits Syst. Video Technol. 23(5), 802–812 (2013)
10. D. Zhou and P. Liu, in *Proc. IEEE International Symposium on Circuits and Systems*. A hardware-efficient dual-standard VLSI architecture for MC interpolation in AVS and H.264 (IEEE, New Orleans, Louisiana, 2007)
11. Chao-Tsung Huang, Chiraag Juvekar, Mehul Tikekar, Anantha P. Chandrakasan, in *Proc. IEEE Conference on Visual Communications and Image Processing (VCIP)*. HEVC interpolation filter architecture for quad full HD decoding (IEEE, Kuching, Sarawak, 2013)
12. G. Pastuszak, M. Trochimiuk, in *Proc. 16th Euromicro Conference on Digital System Design*. Architecture design and efficiency evaluation for the high-throughput interpolation in the HEVC encoder (IEEE, Santander, Spain, 2013)
13. Guo Z, Zhou D, Guto S, in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. An optimized MC interpolation architecture for HEVC (IEEE, Kyoto, Japan, 2012)
14. V. Afonso, H. Maich, L. Agostini, and D. Franco, in *Proc. IEEE Lat. Amer. Symp. Circuits Syst. (LASCAS)*. Low cost and high throughput FME interpolation for the HEVC emerging video coding standard (IEEE, Cusco, Peru, 2013)
15. CM Cláudio, M Shafique, S Bampi, J Henkel, A reconfigurable hardware architecture for fractional pixel interpolation in high efficiency video coding. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. 34(2), 238–251 (2015)
16. E. Kalali, I. Hamzaoglu, in *Proc. IEEE International Conference on Image Processing (ICIP)*. A low energy HEVC sub-pixel interpolation hardware (IEEE, Paris, French, 2014)
17. G. He, D. Zhou, Y. Li, Z. Chen, T. Zhang, and S. Goto, "High-throughput power-efficient VLSI architecture of fractional motion estimation for ultra-HD HEVC video encoding," IEEE Trans. Very Large Scale Integr. VLSI Syst. (2015) doi: 10.1109/TVLSI.2014.2386897.
18. X Lian, W Zhou, Z Duan, R Li, in *Proc. 2nd IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP)*. An efficient interpolation filter VLSI architecture for HEVC standard (IEEE, Xi'an, China, 2014)
19. F. Bossen, "Common test conditions and software reference configurations," document JCTVC-H1100, ITU-T/ISO/IEC Joint CollaborativeTeam on Video Coding (JCT-VC) (ITU-T/ISO/IEC, San Jose, USA, 2012)
20. G. Bjontegaard, *"Calculation of average PSNR difference between RD-curves," document VCEG-M33* (ITU-T, Austin, USA, 2001)
21. V Afonso, H Maich, L Agostini, D Franco, in *Proc. 2013 Data Compression Conference*. Simplified HEVC FME interpolation unit targeting a low cost and high throughput hardware design (Snowbird, Utah, 2013)