doi:10.3772/j.issn.1006-6748.2019.01.004

Deeply pipelined interpolation architecture for full ultra-HD H.265/HEVC video encoding[①]

Ding Dandan(丁丹丹)*, Gui Kai*, Ye Xin**, Liu Fuchang*, Pan Zhigeng[②]*
(* Digital Media & Interaction Research Center, Hangzhou Normal University, Hangzhou 311121, P. R. China)
(** Institute of Information and Communication Engineering, Zhejiang University, Hangzhou 310058, P. R. China)

Abstract

Fractional motion estimation (FME) improves the video encoding efficiency significantly. However, its high computational complexity limits the real-time processing capability. Therefore, it is a key problem to reduce the implementation complexity of FME, especially in hardware design. This paper presents a novel deeply pipelined interpolation architecture of FME for the real-time realization of H.265/HEVC full Ultra-HD video encoder. First, a pipelined interpolation architecture together with an elegant processing order is proposed to deal with different search positions in parallel without pipeline stall and data conflict. Second, interpolation results sharing strategies are exploited among search positions to reduce the memory cost. Finally, the structure of the interpolation filter is further optimized for an area efficient implementation. As a result, the proposed design costs 41 917 slice LUTs on the Xilinx Kintex-7 FPGA platform with a 308 MHz working frequency. The measured throughput reaches a record of 1.238 Gpixels/s, which is sufficient for the real-time encoding of 8192 × 4320@ 30fps video.

Key words: fractional motion estimation (FME), interpolation, pipeline, H.265/HEVC, full Ultra-HD

0 Introduction

The latest video compression standard called high efficiency video coding (H.265/HEVC)[1,2] is approved by both moving picture experts group (MPEG) and ITU-T video coding experts group (VCEG) in January 2013. Compared with previous standard -H.264/AVC[3], H.265/HEVC offers about double of the data compression ratio at the same level of video quality[4]. Particularly, H.265/HEVC can process much higher video resolutions, such as ultra high definition televisions (Ultra-HD) at 7 680 × 4 320 and full Ultra-HD videos at 8 192 × 4 320 resolution. However, the high compression efficiency and high throughput of H.265/HEVC bring in heavy computation load. Now, it is emergent to design high efficiency and high throughput encoder for H.265/HEVC with acceptable complexity.

As the most critical component of video encoding, motion estimation is employed to handle high temporal redundancy between successive video frames, which is the most time-consuming part in the encoding process. Typically, motion estimation consists of two steps: integer motion estimation (IME) and fractional motion estimation (FME), in which IME is used to find the motion vector (MV) by referring to the integer pixels, and then FME is employed to further refine the MV by fractional pixels around the integer pixels. Compared with H.264/AVC, around 4.0% bitrate reduction is achieved by FME in H.265/HEVC[5]. Meanwhile, the computation complexity is increased. It is reported that FME costs almost 49% of the total encoding time due to the complex interpolation and fractional search process[6,7]. As a result, it is a big challenge to design high throughput and low complexity FME architecture for H.265/HEVC encoders.

Recently, many designs of FME interpolation have been developed. Some studies have been proposed to optimize the tap coefficients of the interpolation filter. For example, in Refs[8,9], different tap coefficients were adopted for hardware area reduction and working frequency improvement. As a result, 0.8833% and 1.8833% loss on BD_RATE was introduced compared with H.265/HEVC reference software HM16, respec-

① Supported by the Zhejiang Provincial Natural Science Foundation of China (No. LQ15F010001, LY16F020029), and the General Research Project of Zhejiang Provincial Education Department (No. Y201430479).
② To whom correspondence should be addressed. E-mail: zgpan@hznu.edu.cn
Received on Feb. 12, 2018

tively. In Ref. [10], an adder tree structured interpolator with modified coefficients was developed to replace the multiplier. The design could support 7 680 × 4 320 @ 30fps at a 280 MHz working frequency with 103.6K gate count, but the performance degradation was not mentioned. There are also some work on reducing the number of interpolation positions. For instance, as reported in Ref. [7], a new FME search pattern, which was based on the bilinear quarter pixel approximation (BQA) scheme, was employed to reduce the computation complexity. The results showed that the FME search candidates were reduced from 25 to 12 with 0.03 dB BD_PSNR degradation. Ref. [11] applied interpolation-free FME at the edges of moving objects with 5% BD_RATE increment. An 8 × 8 block based interpolation unit was adopted in Ref. [12]. However, interpolation of sub-blocks, such as 4 × 8, 4 × 16 and 12 × 16 PU, was skipped for simplification, which introduced 0.0438dB BD_PSNR degradation and 1.10% BD_RATE loss. There are also many other work on the interpolation of encoders, which use the same algorithm as H. 265/HEVC standard. In Ref. [13], the fractional positions were grouped into three different sets for hardware simplification and reuse. By increasing the number of pipeline stages in the filters, it is possible to increase the operational frequency and the number of frames processed per second. A two-dimensional continuous interpolation of 9 × 9 blocks with reconfigurable and dedicated filter cores was proposed in Ref. [14]. The processing capability was improved by four times with a relatively small hardware area increase since it covered four overlapping 8 × 8 blocks. Finally, the architecture proposed by Ref. [14] could encode 3840 × 2160 @ 30fps video in real-time. These previous work have explored the interpolation architecture extensively. However, most of these studies could not satisfy the real-time encoding of full Ultra-HD video.

This paper presents a novel interpolation architecture of FME for full Ultra-HD H.265/HEVC video encoding. The essence of this design aims at striking a balance among the area complexity, processing throughput, as well as video quality. Specifically, this design can support a full Ultra-HD resolution (8192 × 4320) up to 35 frames per second. In addition, this work applies the same algorithm with H. 265/HEVC standard so that there is no BD_RATE increment or BD _PSNR degradation. The highlights of the proposed FME architecture are three-fold: (1) 4-stage pipelined interpolation architecture with an elegant processing order is employed for parallel processing of different search positions, thus to avoid pipeline interruption and data conflicts. (2) Interpolation results are shared among different search positions to reduce on-chip memory cost. (3) Interpolation filters are optimized for further reduction in resource consumption.

The rest of the paper is organized as follows. In Section 1, the interpolation algorithm of H.265/HEVC is reviewed. The hardwired interpolation architecture for H.265/HEVC is proposed in Section 2. In Section 3, the implementation results are given. Section 4 draws the conclusion.

1 H.265/HEVC interpolation algorithm

In H. 265/HEVC, the accuracy of an interpolation filter has been improved noticeably over H. 264/AVC. For filter calculations, it employs more taps and higher operation precision[15]. In H.264/AVC, a two-stage cascaded filtering process is employed, where half-pixel samples are first obtained from integer pixels by a 6-tap filter, prior to using those to obtain the quarter-pixel samples. But H.265/HEVC interpolation filter directly computes the quarter-pixels with a 7-tap filter, which significantly reduces the rounding error to 1/128. What's more, for the luma component, one-fourth of pixel accuracy applied in H.265/HEVC can bring 4.0% coding efficiency gain on average compared with H.264/AVC[16].

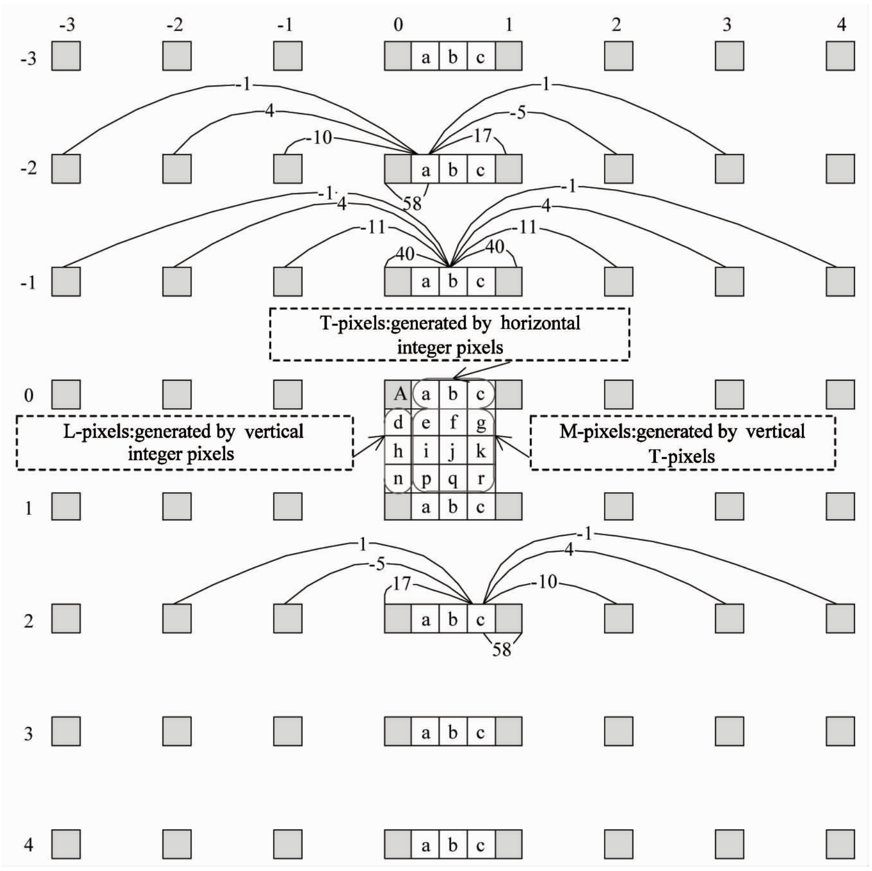
In order to capture the continuous motion more accurately, the fractional pixel accuracy in FME is more important than the integer accuracy in IME. In H.265/HEVC, the interpolation process employs an 8-tap filter for half-pixel position and two 7-tap filters for quarter-pixel positions, as shown in Eqs(1) – (3). Fig. 1 shows the integer, half and quarter positions of luma component, where the capital letters, such as $A_{-1,0}$, $A_{0,0}$, $A_{1,0}$, $A_{2,0}$, ⋯, etc., represent the integer pixels, and the small letters, such as $a_{0,0}$, $b_{0,0}$, $c_{0,0}$, ⋯, etc., denote the interpolated fractional pixels. Integer pixels are directly output without calculation. Fractional pixels like a, b and c are calculated by applying Eqs(1) – (3) to the nearest integer pixels in the horizontal direction, respectively. Fractional pixels like d, h and n are calculated by applying the corresponding equation in the vertical direction. The rest fractional pixels are calculated by applying equations to the unrounded intermediate of d, h and n.

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

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

$$c_{0,0} = (-A_{-3,0} - 5 \times A_{-2,0} + 17 \times A_{-1,0} + 58 \times$$

$A_{0,0} - 10 \times A_{1,0} + 4 \times A_{2,0} - A_{3,0}) >> 6$ (3)



-3
-2
-1
0
1
2
3
4
a
b
c
A
d
e
f
g
h
i
j
k
n
p
q
r
-1
4
-10
58
17
-5
-11
40
T-pixels:generated by horizontal integer pixels
L-pixels:generated by vertical integer pixels
M-pixels:generated by vertical T-pixels

Fig. 1 The integer, half and quarter positions of luma component. (T-pixels can be directly produced by the horizontal integer pixels and L-pixels can be directly produced by the vertical integer pixels and the generation of M-pixels relies on T-pixels.)

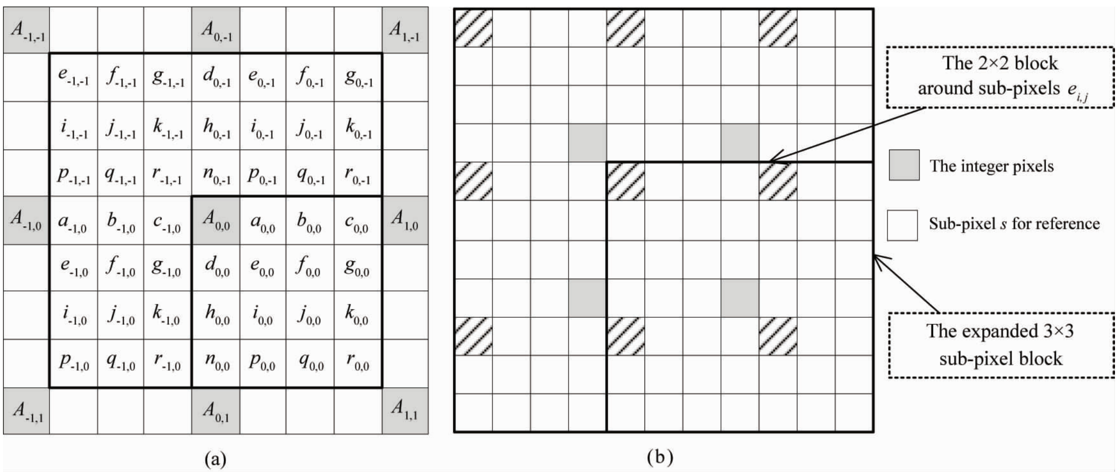
It can be seen that the interpolation calculation involves in a large number of pixels. For example, for 8192 ×4320@30fps video, the amount of luma data in one second is 8192 ×4320 ×30 =126.56 MB pixels. If the full interpolation algorithm is used, up to 126.56 MB × 16 = 2 025 MB interpolation positions should be processed in one second in case that only one reference frame is employed in the encoding procedure. Therefore, it is quite necessary to design a high throughput interpolation accelerator for the full Ultra-HD encoding.

2 Proposed pipeline architecture for H.265/HEVC interpolation

2.1 Results sharing among search positions

In the proposed work, an 8 ×8 block in parallel is interpolated. By using the full search algorithm in IME, the fractional pixel positions to be searched are limited to a 7 ×7 region around the best integer pixel position. The fractional pixel search area is shown in Fig.2(a), which covers a total of 48 fractional pixel positions in addition to the optimal integer position $A_{0,0}$. It can be found that after 1/4 precision interpolation, each integer pixel is corresponded to a 4 ×4 region of fractional pixels. Since the fractional pixels on the same position are calculated with the same formula, the fractional pixels corresponding to the 4 different integer pixels can be processed in parallel. As a consequence, it only requires 15 calculations to complete the processing of all 48 fractional pixel positions in the 7 × 7 region.

To avoid repeated calculation during interpolation, one more row and column of sub-pixels are calculated in each 8 ×8 block of the proposed design, thus an expanded fractional pixel block is obtained. For example, as shown in Fig.2(b), the 2 ×2 block around sub-pixels $e_{i,j}$ is finally expanded to a 3 ×3 sub-pixel block. Thus, for the 8 ×8 block processing unit here, an expanded 9 ×9 block is actually interpolated. The 9 ×9 block contains four overlapping 8 × 8 blocks and there are 7 ×7 samples in the common area. It can be seen that the horizontal or vertical MV difference of the four 8 × 8 blocks is 1. As a result, it is possible to check four fractional MVs around an integer MV simultaneously. In this way, the processing parallelism is increased by four times at a relatively small increase of hardware cost.



(a)
(b)
The 2×2 block around sub-pixels $e_{i,j}$
The integer pixels
Sub-pixel s for reference
The expanded 3×3 sub-pixel block

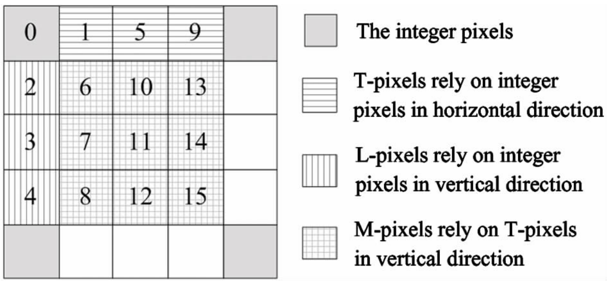
Fig. 2 Proposed results sharing strategies among different search positions: (a) the fractional pixel positions to be searched are limited to a 7 × 7 region around the best integer pixel position, and the fractional pixels corresponding to the 4 different integer pixels can be processed in parallel; (b) Expand the 2 × 2 block to a 3 × 3 sub-pixel reference block

2.2 Proposed 4-stage pipeline and processing order

From Section 1, it can be discovered that there exists data dependency among different fractional pixel positions. If the processing order of these positions was randomly arranged, the pipeline would be interrupted by data conflicts, and then the system processing speed would be seriously affected. Here, a detailed analysis on the data dependency among different fractional pixel positions is given, and then a continuous pipeline architecture for the fractional positions is proposed.

The data dependency among different positions to be interpolated is shown in Fig. 1. It is obvious that there is no data dependency on position a, b and c, which are all in the same row with the integer pixels, and their interpolation results can be directly derived from the integer pixels. It is also the same case with position d, h and n. However, calculation of the rest positions depends on the fractional positions in the same row or column, e. g., e, i and p depend on a; f, j and q depend on b and g, k and r depend on c. For these positions, the calculation cannot be started until the related positions are obtained.

Based on the analysis above, an elegant processing order is designed, as shown in Fig. 3, where the number represents the processing order of positions. It is worth noting that the fractional pixels on the same position of an 8 × 8 processing unit are processed at the same time. The interpolation of each position takes up 4 clock cycles including data-in, interp_0, interp_1 and data-out, which will be further described later.



0
1
5
9
2
6
10
13
3
7
11
14
4
8
12
15
The integer pixels
T-pixels rely on integer pixels in horizontal direction
L-pixels rely on integer pixels in vertical direction
M-pixels rely on T-pixels in vertical direction

Fig. 3 The proposed processing sequence of interpolation

The interpolations of position 1, 2, 3, 4, 5 and 9 are independent on the other fractional pixels, thus their results can be obtained directly from the corresponding integer pixels. Pixels at position 6, 7 and 8 depend on fractional position 1, thus they have to wait until position 1 is completed. Since the interpolation calculation of a fractional pixel takes up 4 cycles, position 2, 3 and 4 should be processed before position 6 (Fig. 4). After buffering positions 2, 3 and 4, the fractional pixel of position 1 has been derived and can be cached for subsequent use. Position 5 is processed after position 4 because position 10, 11 and 12 rely on position 5 and sufficient cycles should be left for them. After completing the interpolation of position 8, there are no other pixels depending on position 1 anymore. At this point, position 5 can be interpolated and the cache occupied by position 1 can be updated. Similarly, the processing order for the subsequent fractional pixels can be arranged.

In summary, the proposed interpolation pipeline architecture is illustrated in Fig. 4. Interp_0 and Interp_1 fulfill the splitting, multiplexing and recombining of interpolation filter coefficients. More specifically, Interp_0 conducts the shift, selection and the first two steps of addition operation, and Interp_1 completes the last two steps of addition operation. It can be seen that the pixels on the same positions can be processed in parallel. With the proposed processing order, the pipeline can process interpolation continuously without data conflicts. Because the 8-tap filter requires additional 4 columns of pixels on the left and right side of current processing unit as overhead, thus for an 8 ×8 processing unit, 9 × (4 + 9 + 4) = 153 pixels (about 1.2 kbit) need to be stored in SRAM on-chip.

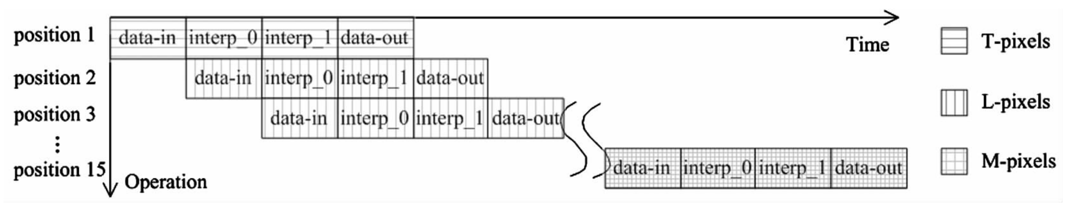


Fig. 4 Proposed 4-stage pipeline architecture for H.265/HEVC interpolation

2.3 Interpolation filter optimization

Utilizing longer tap filters in H.265/HEVC (7/8 tap filter for luma component) increases the amount of data that need to be fetched from the reference memory. At worst, the memory bandwidth of H.265/HEVC interpolation filter is approximately 51% higher than that of H.264/AVC[16]. Here, an optimization scheme is proposed on coefficients of interpolation filters to decrease the resource cost.

H.265/HEVC employs three types of interpolation filters: two kinds of 7-tap interpolation filters for quarter-pixel positions and an 8-tap interpolation filter for half-pixel positions. The tap coefficients of these interpolation filters are shown in Table 1 (a). The corresponding formulas are shown in Eqs(1) – (3) of Section 1, where the capital letters $A_{i,j}$ ($-3 <= i < 4$, $j = 0$) represent the integer pixels, and $a_{0,0}$, $b_{0,0}$ and $c_{0,0}$ are the outputs of three types of interpolation filters. Generally, in digital circuit design, the time delay of shift and add is much shorter than that of multiple operation, and the hardware implementation of shift and add is simpler. Hence, it is proposed to split the coefficients of all interpolation filters into the sum of 2 power exponents. After decomposition, tap coefficients of the three types of filters are rewritten in Table 1(b), and the corresponding formula is shown in Eqs(4) – (6).

Table 1 Proposed optimization on splitting, multiplexing and recombining of interpolation filter coefficients

Coefficient *i*(*j*=0)		*i*=-3	*i*=-2	*i*=-1		*i*=0						*i*=1		*i*=2			*i*=3
I	(a)	-1	4	-10		58						17		-5			1
	(b)	-1	4	-8	-2	-2	-4	64				16	1	-1	-4		1
	(c)	-1	4	-8	-2	-2	-4	16	16	16	16	16	1	-1	-2	-2	1
Coefficient *i*(*j*=0)		*i*=-3	*i*=-2	*i*=-1			*i*=0			*i*=1			*i*=2			*i*=3	*i*=4
II	(a)	-1	4	-11			40			40			-11			4	1
	(b)	-1	4	-8	-2	-1	8	32		32		8	-1	-2	-8	4	1
	(c)	-1	4	-8	-2	-1	8	16	16	16	16	8	-1	-2	-8	4	1
Coefficient *i*(*j*=0)		*i*=-2	*i*=-1			*i*=0		*i*=1						*i*=2		*i*=3	*i*=4
III	(a)	1	-5			17		58						-10		4	-1
	(b)	1	-4		-1	1	16	64				-4	-2	-2	-8	4	-1
	(c)	1	-2	-2	-1	1	16	16	16	16	16	-4	-2	-2	-8	4	-1

$$a_{0,0} = [-A_{-3,0} + 4 \times A_{-2,0} - (8+2) \times A_{-1,0} + (64-4-2) \times A_{0,0} + (16+1) \times A_{1,0} - (4+1) \times A_{2,0} + A_{3,0}] >> 6 \quad (4)$$

$$b_{0,0} = [-A_{-3,0} + 4 \times A_{-2,0} - (8+2+1) \times A_{-1,0} + (32+8) \times A_{0,0} + (32+8) \times A_{1,0} - (8+2+1) \times A_{2,0} + 4 \times A_{3,0} - A_{4,0}] >> 6 \quad (5)$$

$$c_{0,0} = [-A_{-3,0} - (4+1) \times A_{-2,0} + (16+1) \times A_{-1,0} + (64-4-2) \times A_{0,0} - (8+2) \times A_{1,0} + 4 \times A_{2,0} - A_{3,0}] >> 6 \quad (6)$$

Though the three types of interpolation filters have different coefficients, the same power exponent of 2 can be reused after decomposing their coefficients into the sum of 2 power exponents. The specific methods for splitting and multiplexing are shown in Table 1(c). In order to share the shift operation between neighboring coefficients, the '64' of tap coefficient 0 and tap coefficient 1 is further broken into four coefficients of '16'. As a result, the three types of interpolation filters are integrated into one computation unit, as shown in Fig. 5. Compared with the shift-addition scheme, after splitting and multiplexing, the required number of adders is reduced from 37 to 15, as shown in Table 2. Although the number of selectors is increased to 11, the overhead of the proposed architecture is still much lower than the one without optimization.

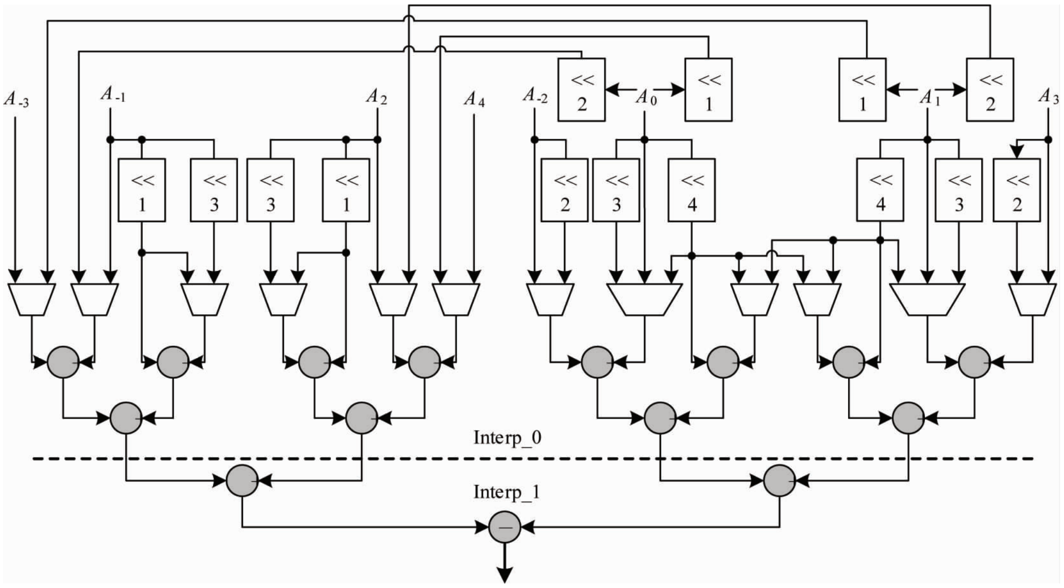


Fig. 5 Proposed structure of the interpolation filter (the Interp_0 and Interp_1 are corresponding to the second and third pipeline stage in Fig. 4, and here the selector with three input lines is actually implemented by a four-input selector in hardware, but only three of the four inputs are valid.)

Table 2 Comparison of computational resources cost between the shift-addition scheme and the proposed scheme

Scheme	Adder	Subtractor	Two-input selector	Four-input selector
Shift-addition	34	3	0	1
Proposed optimization	14	1	10	2

3 Implementation results

The proposed interpolation architecture is implemented in Verilog-HDL and synthesized on Xilinx Kintex-7 (xc7k325t-2ffg900) FPGA platform. It costs 41 917 slice lookup tables (LUTs) and 14 009 registers with a working frequency at 308 MHz. As a result, it is sufficient to process 8 192 × 4 320@ 30fps in real-time when only one reference frame is employed.

Table 3 summarizes the comparison among the proposed and previously published designs. Note that[13] can process 3 840 × 2 160@ 60fps resolutions in real time, using bi-directional prediction with two reference frames. When a single reference frame is employed, it can reach up to 3 840 × 2 160@ 120fps. It can be found that the proposed design delivers a maximum throughput of 1.238 G pixels/s for 8 192 × 4 320 video application, which is the highest throughput of all the reported designs. Compared with Refs[7-12] which employed new interpolation algorithms or coefficients, this design adopts the same interpolation algorithm with H.265/HEVC standard so that there is no BD_RATE increasing or BD_PSNR degradation. Although the previous work in Refs[13] and [14] also employed the same interpolation algorithm as H.265/HEVC standard, the real-time encoding of video resolution 8 192 × 4 320 could not be supported.

Table 3 Comparisons of area and throughput with prior arts

	The proposed	Ref. [7]	Ref. [9]	Ref. [13]	Ref. [14]
Technology	Kintex-7	65 nm	TSMC 65 nm	Stratix III	TSMC 90 nm
Frequency	308 MHz	188 MHz	384 MHz	353.8 MHz	400 MHz
Throughput	8 192 × 4 320@30fps	7 680 × 4 320@30fps	7 680 × 4 320@90fps	3 840 × 2 160@120fps	3 840 × 2 160@30fps
Area Results	41.9 K LUTs	1183 K	89.84 K	7701 ALUTs	240 K

4 Conclusion

In this paper, a deeply pipelined interpolation architecture for full Ultra-HD H.265/HEVC video encoding is presented. In the proposed architecture, processing parallelism on 8 × 8 block is adopted. A 4-stage pipeline architecture together with specific processing order of fractional positions is employed, thus to avoid data conflict during interpolation. Furthermore, the coefficients of interpolation filters are optimized to reduce the hardware cost. Finally, the proposed architecture costs 41.9 k LUTs and 1.2 k memory on-chip on Xilinx Kintex-7 platform with a working frequency at 308 MHz. It is capable of supporting the real-time encoding of full Ultra-HD (8192 × 4320) video at 30 frames per second.

References

[1] 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

[2] Dan G, Marpe D, Mulayoff A, et al. Performance comparison of H.265/MPEG-HEVC, VP9, and H.264/MPEG-AVC encoders[C]. In: Proceedings of the 30th Picture Coding Symposium, San Jose, USA, 2013. 394-397

[3] Wiegand T, Sullivan G J, Bjontegaard G, et al. Overview of the H.264/AVC video coding standard[J]. *IEEE Transactions on Circuits & Systems for Video Technology*, 2003, 13(7):560-576

[4] Vanne J, Viitanen M, Hamalainen T D, et al. Comparative rate-distortion-complexity analysis of HEVC and AVC video codecs[J]. *IEEE Transactions on Circuits & Systems for Video Technology*, 2012, 22(12):1885-1898

[5] Ugur K, Alshin A, Alshina E, et al. Motion compensated prediction and interpolation filter design in H.265/HEVC [J]. *IEEE Journal of Selected Topics in Signal Processing*, 2013, 7(6):946-956

[6] Li H, Zhang Y H, Chao H Y. An optimally scalable and cost-effective fractional-pixel motion estimation algorithm for HEVC[C]. In: Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing, Vancouver, Canada, 2013. 1399-1403

[7] He G, Zhou D J, Li Y S, et al. High-throughput power-efficient VLSI architecture of fractional motion estimation for Ultra-HD HEVC video encoding[J]. *IEEE Transactions on Very Large Scale Integration Systems*, 2015, 23 (12):3138-3142

[8] Diefy A, Shalaby A, Sayed M S. Efficient architectures for HEVC luma interpolation filter[C]. In: Proceedings of the 28th International Conference on Microelectronics, Giza, Egypt, 2016. 9-12

[9] Diefy A, Shalaby A, Sayed M S. Low cost luma interpolation filter for motion compensation in HEVC[C]. In: Proceedings of the 60th International Midwest Symposium on Circuits and Systems, Boston, USA, 2017. 1-4

[10] Wang S H, Zhou D H, Goto S. Motion compensation architecture for 8K UHDTV HEVC decoder[C]. In: Proceedings of the ICME 2014: IEEE International Conference on Multimedia and Expo, Chengdu, China, 2014. 1-6

[11] 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 & Systems for Video Technology*, 2015, 25(9):1533-1544

[12] Lian X C, Zhou W, Duan Z M, et al. An efficient interpolation filter VLSI architecture for HEVC standard[C]. In: Proceedings of the IEEE China Summit & International Conference on Signal and Information Processing (ChinaSIP), Xi'an, China, 2014. 384-388

[13] Maich H, Afonso V, Franco D, et al. High throughput hardware design for the HEVC fractional motion estimation interpolation unit[C]. In: Proceedings of the 20th IEEE International Conference on Electronics, Circuits, and Systems, Abu Dhabi, UAE, 2013. 161-164

[14] Pastuszak G, Trochimiuk M. Algorithm and architecture design of the motion estimation for the H.265/HEVC 4K-UHD encoder[J]. *Journal of Real-Time Image Processing*, 2016, 12(2):517-529

[15] Ugur K, Alshin A, Alshina E, et al. Interpolation filter design in HEVC and its coding efficiency-complexity analysis[C]. In: Proceedings of the 38th IEEE International Conference on Acoustics, Speech, and Signal Processing, Vancouver, Canada, 2013. 1704-1708

[16] Sze V, Budagavi M, Sullivan G J. High efficiency video coding (HEVC): algorithms and architectures[M]. Heidelberg/ New York/Dordrecht/London: Springer Publishing Company, Incorporated, 2014. 129-137

Ding Dandan, born in 1983. She received her B.S. and Ph.D degrees in communication and information system from Zhejiang University, Hangzhou, China, in 2006 and 2011, respectively. Her research interests include very large scale integration system design, FPGA design, video coding algorithm optimization, and image processing.