AN EFFICIENT INTERPOLATION FILTER VLSI ARCHITECTURE FOR HEVC

[1*]*Wei Zhou,* [2]*Xin Zhou,* [1]*Xiaocong Lian*

[1]School of Electronics and Information, [2]School of Automation,
Northwestern Polytechnical University, Xi'an 710072, China (zhouwei@nwpu.edu.cn)

ABSTRACT

Firstly, an implementation-friendly interpolation filter algorithm is proposed in this paper. It can save 19.6% processing time on average with negligible coding quality degradation. Then based on the proposed algorithm, an optimized interpolation filter VLSI architecture, composed of the reused data path of interpolation, efficient memory organization and the pipeline interpolation filter engine is presented to reduce the implement hardware area. The resulting design can achieve 240 MHz with only 37.2K gate count and support real-time interpolation filter operation of 3840×2160@47fps video application by using 90nm CMOS technology.

Index Terms—HEVC, interpolation filter, VLSI

1. INTRODUCTION

High Efficiency Video Coding (HEVC) is a new video coding standard currently being developed jointly by Video Coding Experts Group (VCEG) and Moving Picture Experts Group (MPEG) in the joint collaborative team on video coding (JCT-VC)[1][2]. It provides a significant rate-distortion improvement over its predecessor H.264/AVC and can save 40%-50% bitrates compared to H.264/AVC, especially for ultra-high video resolutions [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 is the key factor for efficient video compression. Compensation for motion with fractional-pel accuracy requires interpolation of reference pixels. In HEVC, three different 8-tap or 7-tap filters are used for the interpolation of half-pixel and quarter-pixel positions, respectively. Sub-pixel interpolation is one of the most computationally intensive parts of HEVC. Compared with the 6-tap filter used in H.264/AVC [4], the 7-tap and 8-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 dedicated hardware architecture for interpolation filter to realize the real-time processing for high resolutions video.

There are some previous works focusing on designing efficient architecture for HEVC interpolations [5-8]. Huang proposed a high-throughput interpolation filter architecture and a unified filter combining the 8-tap luma and 4-tap chroma filters to optimize area [5]. In [6], a dedicated hardware accelerator for interpolation was presented. Although it could read eight input samples and produce 64 output samples at each clock cycle, its area cost was huge. A sub-pixel interpolation hardware only for 4×4 PU size and a 2-D filter reuse scheme for sub-block 4×4 were proposed in [7]. But the hardware had restricted reconfigurability. In this paper, a fast and implementation-friendly interpolation filter algorithm is proposed. Then based on the proposed algorithm, an efficient interpolation filter VLSI architecture with the reused data path and efficient memory organization is presented.

The rest of this paper is organized as follows. The implementation-friendly interpolation filter algorithm is proposed in Section 2. The proposed optimized interpolation filter VLSI architecture is presented in Section 3 in details. Section 4 shows the implementation results. Finally, this paper is concluded in Section 5.

2. PROPOSED INTERPOLATION FILTER ALGORITHM

2.1 The implementation-friendly interpolation filter algorithm

Like H.264/AVC, mode decisions with motion estimation remain 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×2N and N×N) and two symmetric motion partition (SMP) modes (2N×N and N×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×nU, 2N×nD, nR×2N and nL×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. All possible prediction modes are traversed. And the one having the minimum R-D cost will be used.

According to 8 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 4-pixel interpolation unit include 4×8, 4×16, 8×4, 16×4, 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 1. The size ranges from 64× to 4×. 64× size includes 64×32 and 64×64 modes. 32× size includes 32×8, 32×16, 32×24, 24×32 and 32×32 modes. 16× size includes 16×8, 16×16, and 16×32 modes. 8× size includes 8×8, 8×16, and 8×32 modes. 4× size includes 4×8, 4×16, 8×4, 16×4, 12×16 and 16×12 modes. It can be observed from Table 1 that, although the splitting modules for a 4-pixel interpolation unit (4× size) are only about 3.52% of all possible splittings of PUs, they cost more hardware areas and clock cycles in the hardware implementation.

Table 1. The inter-prediction splitting modes

	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×	19654	8994	2307	876	2193
8×	28408	9261	4232	1409	2098
4×	2582	502	426	150	97
Total	65950	26771	8356	2778	7320

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. 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 are skipped in the interpolation. Based on the proposed fast interpolation algorithm, we re-arrange the classification of PU splitting modules, as shown in Table 2. 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 2. The new splitting module in interpolation filtering

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

2.2 Experiment results

In order to evaluate the performance of the proposed interpolation algorithm, the algorithm is implemented by using the recent HEVC reference software (HM 13.0) and is compared with the original algorithm of HEVC in low complexity configuration. The performance of the proposed algorithm is shown in Table 3.

The proposed algorithm is evaluated with QPs 22, 27, 32 and 37 using 14 typical sequences recommended by the JCT-VC in five resolutions [9]. Computational complexity is measured by the consumed coding time. BDPSNR (dB) and BDBR (%) are used to represent the average PSNR and bit rate difference [10]. "Time save (%)" is used to represent the coding time change in percentage. The positive and negative values represent increments and decrements, respectively.

Table 3. Results of the proposed algorithm compared to HEVC

Class	Sequence	BD_PSNR [dB]	BD_Rate [%]	Time save [%]
A	PeopleOnStreet	-0.0529	1.84	21.9
	Traffic	-0.0273	0.79	19.4
B	BasketballDrive	-0.0136	0.60	19.9
	BQTerrace	-0.0138	0.83	18.9
	Cactus	-0.0299	1.35	19.6
C	PartyScene	-0.0611	2.84	20.2
	Flowervase	-0.0961	1.36	17.1
	BasketballDrill	-0.0275	0.77	18.9
D	BasketballPass	-0.1071	2.46	21.0
	BlowingBubbles	-0.1131	2.81	21.0
	Keiba	-0.1109	2.31	21.4
E	Johnny	-0.0314	1.08	17.4
	KristenAndSara	-0.0392	1.23	17.2
	Vidyo4	-0.0173	0.71	18.0
Average		-0.0516	1.41	19.6

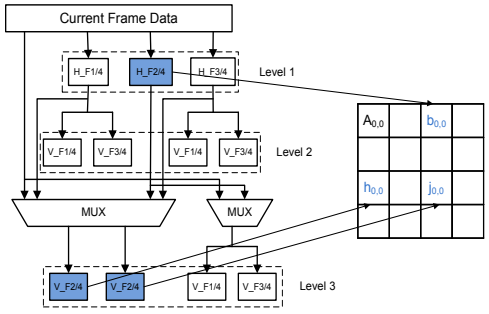
Table 3 shows the performance of the proposed fast interpolation algorithm as compared to the original algorithm in HEVC. The proposed algorithm can reduce the coding time by 19.6% in average. It can also reduce the coding time by about 10% in average compared to our previous algorithm which only the interpolation process of 4×8, 4×16, and 12×16 blocks are skipped [11]. The gain of our algorithm is high because unnecessary small CU size decision has been skipped. On the other hand, coding efficiency loss is negligible, where the average coding efficiency loss in term of PSNR is about 0.05 dB. Therefore, the proposed algorithm can efficiently reduce the coding time while keeping nearly the same RD performance as with the original algorithm in HEVC. What's more, it can also reduce the implementation area cost in the VLSI design.

3. THE OPTIMIZED INTERPOLATION FILTER VLSI ARCHITECTURE

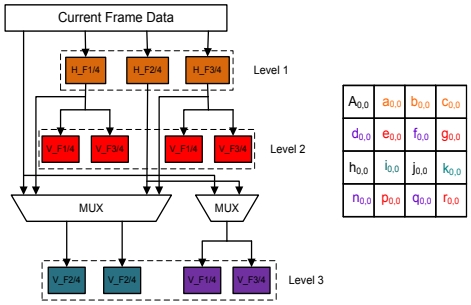
3.1 The reused data path of interpolation

For the interpolation process of a 64×64 CU, 2×(64+1)×(64+8)×(8+6)= 131040 bits RAM is required in total. The area cost will be huge for hardware implementation. Therefore, a reused three-level architecture for fractional pixel

interpolation is proposed to reduce the area cost for about 131040 bits RAM. Figure 1 shows the data path of the three-level reused architecture.



(a) First round: half-pixel interpolation



(b) Second round: quarter-pixel interpolation

Fig.1. The reused data path of interpolation

There are three horizontal filters in the first level. For the half-pixel interpolation as shown in Fig.1 (a), the filter H_F2/4 is open for the interpolation of half-pixel $b_{0,0}$ and the other two (H_F1/4 and H_F3/4) are close in the first round. For the quarter-pixel interpolation in the second round as shown in Fig.1 (b), the half-pixels $a_{0,0}$, $b_{0,0}$ and $c_{0,0}$ will be interpolated by the three filters in Level 1 from $A_{0,0}$. Level 2 contains four vertical filters. They work at the second round of quarter-pixel interpolation process. The quarter-pixels $e_{0,0}$, $p_{0,0}$, $g_{0,0}$ and $r_{0,0}$ are interpolated by V_F1/4 and V_F3/4 respectively from the half-pixel $a_{0,0}$ and $c_{0,0}$ in the vertical direction. Level 3 also contains four vertical filters. The half-pixels $h_{0,0}$ and $j_{0,0}$ are interpolated by the two vertical filters V_F2/4 from $A_{0,0}$ and $b_{0,0}$ at the first round. During the second round, the quarter-pixels $i_{0,0}$ and $k_{0,0}$ will be interpolated by the same two vertical filters V_F2/4. The quarter-pixels $d_{0,0}$ and $n_{0,0}$ will be interpolated by the other two vertical filters V_F1/4 and V_F3/4 from $A_{0,0}$ when the horizontal component of MV is equal to zero; Otherwise the quarter-pixels $f_{0,0}$ and $q_{0,0}$ will be interpolated by the same vertical filters V_F1/4 and V_F3/4 from the half pixel $b_{0,0}$.

Therefore all the horizontal and vertical filters in the process of half-pixel interpolation can be reused in quarter-pixel interpolation and some filter units can be reused for different quarter-pixel positions. This reused architecture will greatly reduce the area cost in hardware implementation.

3.2 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.

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 there are eight SRAMs in order to realize the storage of a 72×72 pixel matrix. As the width of processing unit is from 8 to 64, the 72×72 pixel matrix is stored in terms of nine pixel width separately as shown in Fig. 2. The depth of every SRAM is 9×8 bit = 72 bits and every bit is the data address of each line. Based on this organization, only SRAM0 and SRAM1 are open for 8× processing unit while the others are close with no data access. When the width of processing unit is 64, all the SRAMs will be used to store and read the input pixels.

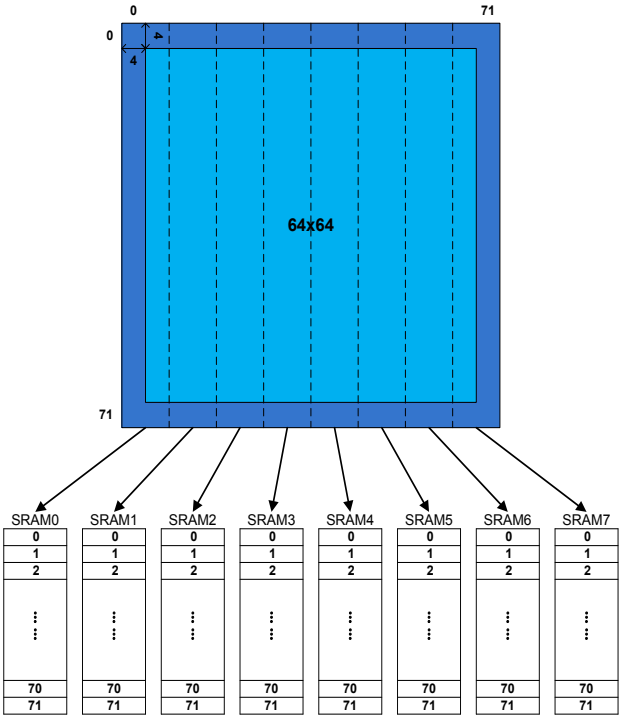


Fig.2. Memory organization

3.3 The pipeline interpolation filter engine

The proposed pipeline filter engine is shown in Fig. 3 where the 8× block module is the basic reused block. As shown in Fig.3, h_f and v_f are the 8-tap horizontal and vertical interpolation filters. 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.

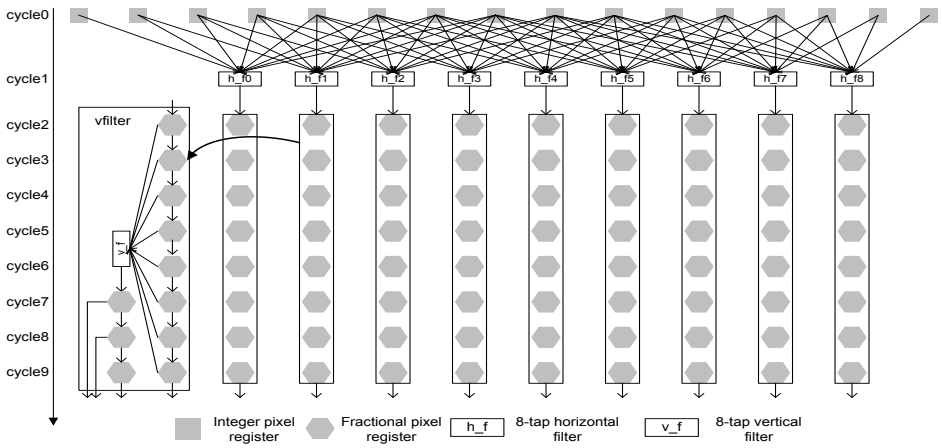


Fig.3. The pipeline interpolation filter engine

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

4. IMPLEMENTATION RESULTS

The proposed interpolation filter architecture is implemented in Verilog HDL and synthesized using SMIC 90nm cell library. Table 4 shows the implementation result of the proposed architecture, including the comparison with the previous works [4][5][6][7] as well as our previous architecture[11]. When synthesized with 90nm CMOS standard library, the total gate count of this design is 37.2k for supporting 3840 × 2160@47fps videos and real time processing with a working clock speed of 240MHz.

In terms of hardware resources, the proposed architecture can reduce about 52% hardware area compared to the original HEVC interpolation architecture and 18% area is reduced compared to the works in [5]. The optimized hardware architecture proposed in this paper can also reduce about 42% hardware area compared to our previous work [11]. Although the works in [6] has eight times greater parallelism and can work at higher frequencies than the design in this paper, the amount of logic resources is also 6 times greater. The proposed architecture also allows for the use of a reduced input buffer so that the memory cost can be reduced by 131040 bits.

In terms of performance, the throughput of the proposed architecture is 0.84 pixel/cycle, which is 15% higher than the works in [7] with 0.73 pixel/cycle, 6% lower than the works in [4] with 0.89 pixel/cycle for H.264/AVC. Consequently, the hardware implementation cost of our architecture is comparable to H.264/AVC.

5. CONCLUSION

High performance VLSI architecture for interpolation in HEVC is proposed in this paper based on an implementation-friendly interpolation filter algorithm. The optimized interpolation filter VLSI architecture only requires 37.2k gates in a standard 90nm CMOS technology at an operating frequency of 240MHz. It can support real-time interpolation filter operation of 3840 × 2160@47fps video application.

6. ACKNOWLEDGEMENT

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).

Table 4. Result comparison of previous works and proposed architecture

	[4]	[5]	[6]	[7]	[11]	Proposed
Standard	H.264, AVS	HEVC	HEVC	HEVC	HEVC	HEVC
Technology	0.18um	40nm	90nm	90nm	90nm	90nm
Parallelism	4×	8×	64×	4×	8×	8×
Gates Count (K Gates)	26.3	45.2	211.693	32.496	64.5	37.2
Memory(byte)	N/A	N/A	N/A	N/A	0	0
Interpolation execution time(pixel/cycle)	0.89	N/A	N/A	0.73	0.84	0.84
Working Frequency	100MHz	200MHz	400MHz	200MHz	193MHz	240MHz
Application Target	1080p@30fps	QFHD@30fps	1080p@30fps	QFHD@30fps	QFHD@47fps	QFHD@47fps

7. REFERENCES

[1] Sullivan G J, Ohm J R, Han W J, et al, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[2] J. Ohm and G.J Sullivan, "High efficiency video coding: the next frontier in video compression [Standards in a Nutshell] ," *Signal Processing Magazine, IEEE*, 2013, 30(1), 152-158.

[3] J.-R. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards - including high efficiency video coding (HEVC)," *IEEE Transaction on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012.

[4] D. Zhou and P. Liu, "A Hardware-Efficient Dual-Standard VLSI Architecture for MC Interpolation in AVS and H.264," in *IEEE International Symposium on Circuits and Systems*, pp. 2910-2913, May. 2007.

[5] Chao-Tsung Huang, Chiraag Juvekar, Mehul Tikekar, Anantha P. Chandrakasan, "HEVC Interpolation Filter Architecture for Quad Full HD Decoding," in *Visual Communications and Image Processing* (*VCIP*), pp. 1-5, Nov. 2013.

[6] G. Pastuszak, M. Trochimiuk, "Architecture Design and Efficiency Evaluation for the High-Throughput Interpolation in the HEVC Encoder," in *2013 Euromicro Conference on Digital System Design*, pp. 423-428, Sep. 2013.

[7] Guo Z, Zhou D, Guto S, "An optimized MC interpolation architecture for HEVC," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 1117–1120, Mar. 2012.

[8] Ercan Kalali, Yusuf Adibelli, Ilker Hamzaoglu, "A Reconfigurable HEVC Sub-pixel Interpolation Hardware," in *IEEE Third International Conference on Consumer Electronics - Berlin*, pp.125-128, Sept. 2013.

[9] F. Bossen, "Common test conditions and software reference configurations," *Joint Collaborative Team on Video Coding (JCTVC) of ITU-T SG16 WP3 and ISO/IEC JTC1/SC29/WG11*, Document: JCTVC-B300, 2nd Meeting: Geneva, CH, 21-28 July, 2010

[10] G. Bjontegaard, "Calculation of average PSNR difference between RD-curves," *13th VCEG-M33 Meeting*, Austin, TX, Apr. 2-4, 2001.

[11] Xiaocong Lian, Wei Zhou, Zhemin Duan, Rong Li, "An efficient interpolation filter VLSI architecture for HEVC standard," in *2nd IEEE China Summit and International Conference on Signal and Information Processing (ChinaSIP 2014), Xi'an, China.*, pp. 384–388, July. 2014.