Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

Highly parallel implementation of sub-pixel interpolation for AVS HDTV decoder[*]

Wan-yi LI[†], Lu YU
(*Institute of Information and Communication Engineering, Zhejiang University, Hangzhou 310027, China*)
[†]E-mail: leeswane8621@hotmail.com

Received Feb. 18, 2008; revision accepted Apr. 8, 2008; CrossCheck deposited Nov. 10, 2008

Abstract: In this paper, we propose an effective VLSI architecture of sub-pixel interpolation for motion compensation in the AVS HDTV decoder. To utilize the similar arithmetical operations of 15 luma sub-pixel positions, three types of interpolation filters are proposed. A simplified multiplier is presented due to the limited range of input in the chroma interpolation process. To improve the processing throughput, a parallel and pipelined computing architecture is adopted. The simulation results show that the proposed hardware implementation can satisfy the real-time constraint for the AVS HDTV (1 920×1 088) 30 fps decoder by operating at 108 MHz with 38.18k logic gates. Meanwhile, it costs only 216 cycles to accomplish one macroblock, which means the B frame sub-pixel interpolation can be realized by using only one set of the proposed architecture under real-time constraints.

Key words: VLSI architecture, Interpolation, AVS HDTV
doi:10.1631/jzus.A0820112 **Document code:** A **CLC number:** TN919.8; TP37

INTRODUCTION

AVS (Audio Video coding Standard) is a digital audio and video coding standard established by the AVS Workshop of China (AVS, 2003). It has been adopted as the national standard of China because of its excellent performance. Some key compression technologies, such as in-loop deblocking, quarter-pixel interpolation, 8×8 integer transform, are used in AVS. Compared with ISO/IEC MPEG-4 AVC/ITU H.264, AVS adopts different four-tap filters for half- and quarter-pixel interpolation. This method can save 11% memory bandwidth in similar computation complexity with respect to that of H.264 (Wang *et al*., 2004).

By using quarter-pixel interpolation, the image reconstruction accuracy can be improved; however, it brings a sharp increase in computation and time complexity (Horowitz *et al*., 2003). The computational load of sub-pixel interpolation processing occupies about 25% of the whole AVS decoder (Deng *et al*., 2004). Therefore, the VLSI implementation of sub-pixel interpolation is the most important part of the design of the AVS HDTV decoder.

Usually, under real-time constraints the interpolation should be finished in limited cycles. Low throughput makes most conventional designs replicate the interpolation module to decode B frames (Wang *et al*., 2005; Zheng *et al*., 2006; Li *et al*., 2007), which raises the hardware cost in all. In this paper, a high throughput hardware design of interpolation is introduced, by which the processing of a luma block can be accomplished within 28 cycles even in the worst case. The high speed means the two-directional interpolation can be finished with only one set of architecture. Meanwhile, to reduce the logic gate cost, the interpolation filters are simplified into three types according to the similar arithmetical operations of different luma positions interpolation, and in the chroma interpolation process a small lookup table replaces multipliers as the key computational unit.

[*] Project (No. 20051321B01) supported by the Science and Technology Development Project of Hangzhou, China

The rest of the paper is arranged as follows. Section 2 gives a brief introduction of the AVS interpolation algorithm. Section 3 presents the framework of the proposed interpolation architecture, and in Section 4, the pipelined workflow of sub-pixel position generation is described. The implementation results and comparisons are shown in Section 5. Finally, concluding remarks are given in Section 6.

INTERPOLATION ALGORITHM IN AVS

In the interpolation process of AVS video decoding, 15 luma sub-pixel positions might be reconstructed, which are depicted in Fig.1 (positions *a*~*k*, *n*, *p*~*r*).

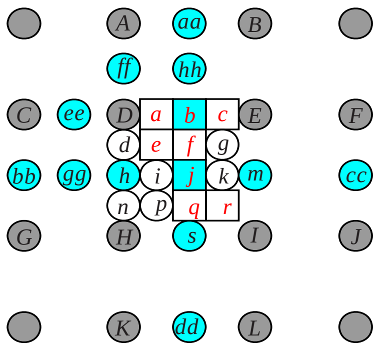

Fig.1 Luma interpolation positions in AVS

The horizontal or vertical half-samples are calculated directly by the four adjacent integer-samples. The intermediate values are calculated as follows:

$$b' = -C + 5D + 5E - F,$$
$$h' = -A + 5D + 5H - K,$$
$$j' = -aa' + 5b' + 5s' - dd'.$$

The final half-samples can be obtained as

$$B = Clip1((b' + 4) >> 3),$$
$$H = Clip1((h' + 4) >> 3),$$
$$J = Clip1((j' + 32) >> 6).$$

Quarter-samples at positions *a*, *c*, *d* and *n* are generated by their adjacent half-samples and integer-samples in the same way. Quarter-samples at positions *f*, *q*, *i* and *k* are calculated in the same way. Corresponding equations are illustrated as follows:

$$a' = Clip1((ee' + 7D' + 7b' + E' + 64) >> 7),$$
$$f' = Clip1((hh' + 7b'' + 7j' + s'' + 512) >> 10).$$

Quarter-samples at positions *e*, *g*, *p* and *r* are calculated by diagonal samples (as shown in Fig.1):

$$E = Clip1((D'' + j' + 64) >> 7).$$

Chroma interpolation in AVS is based on the 4×4 sub-blocks. The calculation formula is

$$predMatrix[x,y] = [(8 - dx)\cdot(8 - dy)\cdot A + dx\cdot(8 - dy)\cdot B + (8 - dx)\cdot dy\cdot C + dx\cdot dy\cdot D + 32] >> 6,$$

where d*x*, d*y*, *A*, *B*, *C* and *D* are shown in Fig.2.

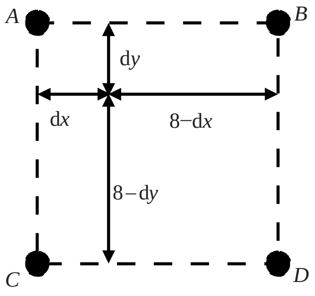

Fig.2 Chroma interpolation positions

PROPOSED ARCHITECTURE

To design an AVS HDTV (1920×1088) 30 fps video decoder working at 108 MHz, means to decode a macroblock within 441 cycles, i.e.,

$$\left\lfloor \frac{108\times10^6}{1920\times1088\times30/(16\times16)} \right\rfloor = 441.$$

If there are B frames in the bitstream, the average value of two-directional prediction should be taken as the final result, which needs a high throughput interpolation module. In our design, parallel computational units are used to meet the need.

Memory bandwidth is a key issue of concern in the design of an HDTV decoder (Ling and Wang, 2003; Tsai *et al.*, 2005; Mizosoe *et al.*, 2007). Since the smallest block size of motion compensation in AVS is 8×8, the 13×13 reference data (The original 8×8 block size plus 2-pixel extension to the left and top plus 3-pixel extension to the right and bottom) are always sufficient for such a block, no matter where the target interpolation position is. Similarly, the reference data window is chosen as 5×5 for a sub-block interpolation of chroma. Therefore, for every macroblock we should load 876-byte integer-samples from off-chip memory. The required memory bandwidth is about 215 MB/s. It is acceptable under our objective specification. With a 32-bit system bus, memory access of interpolation module occupies about 50% of memory bandwidth in the AVS decoder.

Fig.3 shows the whole framework of the proposed design including interpolation modules and their adjacent modules, which contain a system controller for giving control signals and two RAMs for storing data.

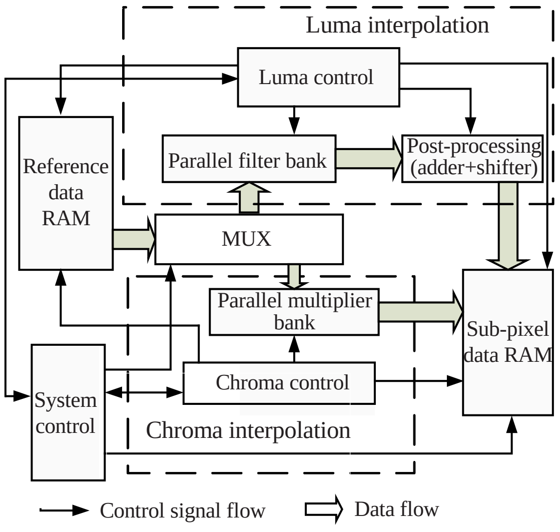

Fig.3 Framework of the proposed design

The system controller gives the control signals to every module in the whole system of the decoder, where it provides information to interpolation, such as the start/finish signal and the interpolation type information (luma or chroma). Two RAMs actually are the interface between the interpolation module and other computational modules. They store the reference data received from off-chip memory and the final result of interpolation, respectively.

The interpolation module consists of luma component interpolation and chroma component interpolation.

Luma component interpolation

In the luma component interpolation algorithm of AVS, the horizontal position and its corresponding vertical position are calculated using the same method (such as *a* vs *d*, *b* vs *h*). To sufficiently reuse the hardware architecture, the 15 positions are divided into two classes and the positions in each class share the same hardware structure. In Fig.1, the positions with rectangles belong to Class 1, and the other positions are Class 2.

Meanwhile, the reference data RAM stores 13×13 integer-samples, which have been arranged according to the position information given by the system controller in advance. If the luma interpolation position belongs to Class 1, the reference data are stored by row, otherwise by column. Fig.4 shows the data organizations in RAM. For every two cycles, the reference data in an address (such as 00h) are loaded from RAM and the inputs of every filter are updated.

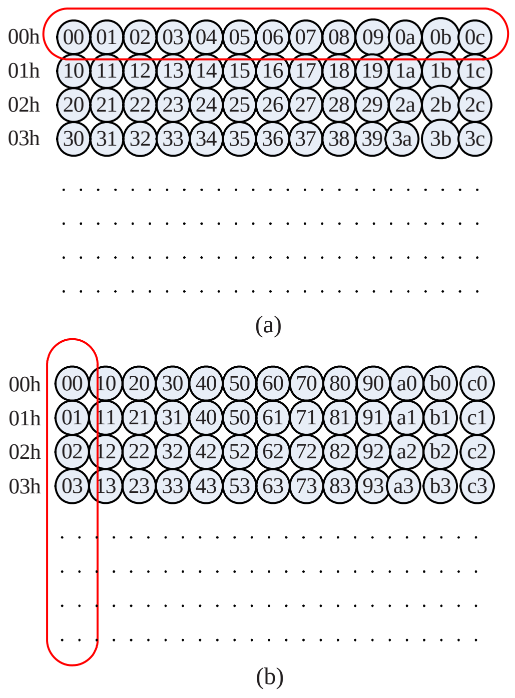
00h 00 01 02 03 04 05 06 07 08 09 0a 0b 0c
01h 10 11 12 13 14 15 16 17 18 19 1a 1b 1c
02h 20 21 22 23 24 25 26 27 28 29 2a 2b 2c
03h 30 31 32 33 34 35 36 37 38 39 3a 3b 3c
(a)
00h 00 10 20 30 40 50 60 70 80 90 a0 b0 c0
01h 01 11 21 31 40 50 61 71 81 91 a1 b1 c1
02h 02 12 22 32 42 52 62 72 82 92 a2 b2 c2
03h 03 13 23 33 43 53 63 73 83 93 a3 b3 c3
(b)

Fig.4 Reference data RAM. (a) Class 1; (b) Class 2

To reduce the area cost, the same structure is used to realize more filters and arrange the filters in the pipelined process. As described in Section 2, the interpolation of all luma sub-pixel positions is composed of only two kinds of calculations: $-A+5B+5C-D$ and $A+7B+7C+D$. All quarter-samples are calculated after the adjacent half-samples are obtained. Half-samples *j* are calculated after half-samples *b* are available. So the whole sub-pixel interpolation process can be divided into three steps. Correspondingly three types of filters are used to realize all operations: the half-pixel interpolation filters F1, F2, and the quarter-pixel interpolation filter F3 (Table 1).

Table 1 The three types of filters

Filter	Input bit width (bit)	Output bit width (bit)	Calculation	Output data
F1	8	13	$-A+5B+5C-D$	Half-samples *b*
F2	13	17	$-A+5B+5C-D$	Half-samples *j*
F3	17	22	$A+7B+7C+D$	Quarter samples

In Fig.1, when the quarter-pixel positions are *e*, *p*, *g* and *r*, the calculation is $A+D$. We can still realize it using F3 by resetting the inputs *B* and *C* to 0.

The core of luma interpolation is a parallel filter bank, the structure of which is shown in Fig.5. In each cycle, all the filters pull corresponding data in and push results out for the next computational units. Especially, the input data of F3 can be selected according to the quarter-pixel positions, and the output data are sent to the post-processing module to obtain the final result.

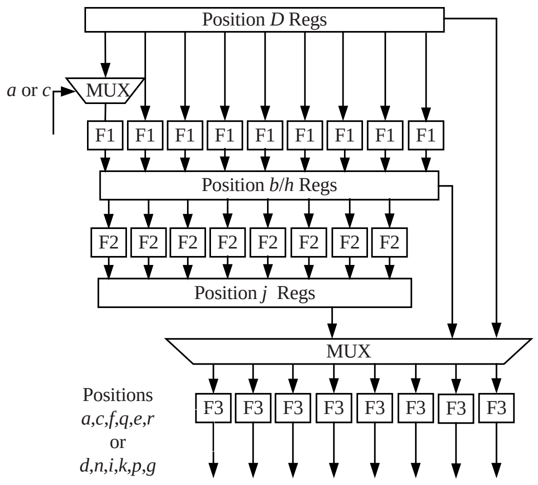

Fig.5 Structure of a parallel filter bank

Chroma component interpolation

Parallel computational units of chroma interpolation contain eight basic units in total. Fig.6 shows the structure of one unit. Obviously, multipliers are the key part of this unit. Suppose *Mul* and *X* are the inputs of the multipliers. Since the possible values of *Mul* are $8-dx$, d*x*, d*y*, $8-dy$, the range of which is [0, 8], a lookup table (Table 2) can be used to realize the multiplication. With the same 0.18-μm technology, the proposed lookup table needs only 3 093 gates while the multipliers require 3 103 gates.

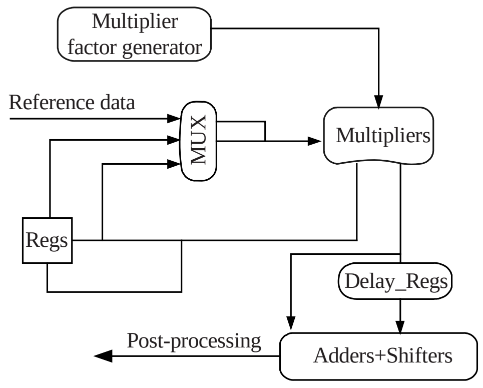

Fig.6 Chroma interpolation unit

Table 2 Multiplier realization

Multiplier	*Mul*·*X*	Multiplier	*Mul*·*X*
0	0	5	(X<<2)+X
1	X	6	(X<<2)+(X<<1)
2	X<<1	7	(X<<2)+(X<<1)+X
3	(X<<1)+X	8	X<<3
4	X<<2		

To speed up the calculation, the intermediate results can be used. For example, in Fig.2, points *C*, *D* will be in positions *A*, *B* respectively when the reference data are updated. The results $(8-dx)\cdot C$ and $dx\cdot D$ are saved for the interpolation of the next sample. Therefore, we can accomplish the chroma interpolation of a 4×4 sub-block within 2+3+3+3+2=13 cycles.

INTERPOLATION WORKFLOW

Pipelined process

As analyzed in Section 3, one B macroblock should be interpolated within 441 cycles. It contains eight luma blocks and four chroma blocks for the 4:2:0 format. Table 3 is the pipeline of the luma computational process. Symbol “s” represents saving the result into registers; “w” represents writing the result into RAM. The original data are updated in every two cycles. Take position *b* as an example. It is unnecessary to write the result to RAM until the reference data of the third row come, but the results of the first two rows should be saved for the interpolation of position *j*.

Table 3 Pipeline process of luma interpolation

Row	Step	*b*	*j*	*a,c,e,r,f*	*q*
1	1				
	2	s			
2	1				
	2	s			
3	1				
	2	s w			
4	1				
	2	s w			
5	1		s		
	2	s w			
6	1		s w		
	2	s w		s w	
7	1		s w		
	2	s w		s w	s w
8	1		s w		
	2	s w		s w	s w
9	1		s w		
	2	s w		s w	s w
10	1		s w		
	2	s w		s w	s w
11	1		s w		
	2	s		s w	s w
12	1		s w		
	2	s		s w	s w
13	1		s w		
	2	s		s w	s w
14	1		s		
	2				s w

It is clear from Table 3 that even for the most complicated position *q*, the interpolation of one block can be finished within 28 cycles. Since one chroma sub-block needs 13 cycles, 13×8+28×4=216 cycles are needed to finish one macroblock interpolation. The two-directional interpolation of one B macroblock can be finished within 432 (<441) cycles.

Extra registers

Apart from the computing units, the need for extra registers should be considered. Due to the high throughput required for the HDTV decoder, it is expected that more pixels be interpolated within one cycle. Not only parallel filters but also more registers are used simultaneously.

In Fig.7, position *f* is taken as an example. Position *f* requires the value of position *j*; position *j* requires the value of position *b* in the latest four rows (Reg *b* data d3, Reg *b* data d2, Reg *b* data d1, Reg *b* data d0). Every two cycles the registers are refreshed. Therefore two registers saving position *j* and four registers saving position *b* need to be interpolated. Other situations in the pipelined process are analyzed carefully. The required numbers of additional registers are listed in Table 4.

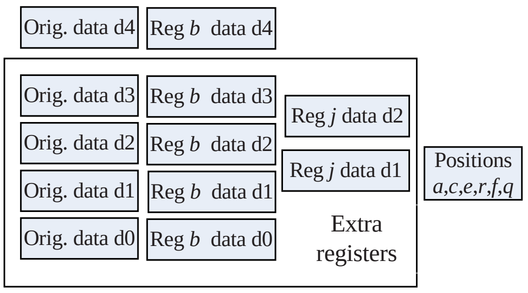

Fig.7 Arrangement of extra registers

Table 4 Required numbers of additional registers

Data in register	Register bit width (bit)	Number of registers
Original data	8	9×4=36
Position *b*	13	9×4=36
Position *j*	17	8×2=16

IMPLEMENTATION RESULTS

The proposed interpolation architecture is designed with Verilog HDL and synthesized with 0.18-μm CMOS standard-cell library by Synopsys Design Compiler. The overall hardware cost is 38.18k gates, at the clock frequency 108 MHz.

In Section 1, it is mentioned that the computational complexity of interpolation in AVS is almost the same as that in H.264/AVC. Table 5 gives a comparison of the proposed architecture with other designs for H.264 (Because reference design should be replicated to process B frames interpolation simultaneously, the total gate cost of interpolation module in the decoder system should be twice).

The comparison between existing interpolation schemes of AVS is shown in Table 6.

Obviously, parallel filter bank and parallel multiplier bank can speed up the interpolation process. Although it leads to more registers and filters, the total area decreases because the B frame sub-pixel interpolation can be realized using only one set of the proposed architecture. So we can find the proposed design has obvious advantages both in less area and in less execution time.

Table 5 Comparison with other interpolation modules in H.264 HDTV decoder

Source	Clock rate (MHz)	Technology	Logic gates	On-chip memory (B)	Macroblock (cycles)
Chen *et al*., 2004	100	0.18 μm	23.87k×2=47.74k	NA	–
Song *et al*., 2005	274	0.18 μm	24.33k×2=48.66k	238×2=476	–
Wang *et al*., 2005	100	0.18 μm	20.68k×2=41.36k	NA	560
Tsai *et al*., 2005	125	0.18 μm	21.51k×2=43.02k	NA	488
Li *et al*., 2007	100	0.18 μm	13.02k×2=26.04k	2K×2=4K	304
This paper	108	0.18 μm	38.18k	169	216

Table 6 Comparison with other interpolation architectures of AVS

Source	Technology	Luma area (μm^2)	Clock rate (MHz)	Total gate cost	Block (cycles)	Macroblock (cycles)
Deng *et al*., 2004	0.25 μm	379 515×2=759 030	150		71	
Zheng *et al*., 2006	0.18 μm		148.5	100k		580
This paper	0.18 μm	295 527	108	38.18k	28	216

CONCLUSION

In this paper, we propose an effective VLSI hardware architecture of interpolation for the AVS HDTV decoder. The pipelined and parallel design highlights the reduction in the operating cycles. One set of the proposed architecture can realize the function of two sets of conventional parallel design under real-time constraints. To reduce the logic gate cost, three types of filters and a small lookup table instead of multipliers are adopted in the interpolation process. The results show that the proposed design can support the HDTV (1 920×1 088) 30 fps decoder with a higher speed and less area while working at a lower frequency.

References

AVS (Audio Video Coding Standard Workgroup of China), 2003. Video Coding Standard FCD1.0.

Chen, T.C., Huang, Y.W., Chen, L.G., 2004. Fully Utilized and Reusable Architecture for Fractional Motion Estimation of H.264/AVC. Proc. IEEE ICASSP, **5**:9-12.

Deng, L., Gao, W., Hu, M.Z., Ji, Z.Z., 2004. An Efficient VLSI Implementation for MC Interpolation of AVS Standard. Advances in Multimedia Information Processing, p.200-206.

Horowitz, M., Joch, A., Kossentini, F., Hallapuro, A., 2003. H.264/AVC baseline profile decoder complexity analysis. *IEEE Trans. on Circuits Syst. Video Technol.*, **13**(7):704-716. [doi:10.1109/TCSVT.2003.814967]

Li, Y., Qu, Y.M., He, Y., 2007. Memory Cache Based Motion Compensation Architecture for HDTV H.264/AVC Decoder. IEEE Int. Symp. on Circuits and Systems, p.2906-2909. [doi:10.1109/ISCAS.2007.377857]

Ling, N., Wang, N.T., 2003. A real-time video decoder for digital HDTV. *J. VLSI Signal Processing*, **33**(3):295-306. [doi:10.1023/A:1022179914445]

Mizosoe, H., Yoshida, D., Nakamura, T., 2007. A single chip H.264/AVC HDTV encoder/decoder/transcoder system LSI. *IEEE Trans. on Consum. Electron.*, **53**(2):630-635. [doi:10.1109/TCE.2007.381739]

Song, Y., Liu, Z.Y., Goto, S., Ikenaga, T., 2005. A VLSI Architecture for Motion Compensation Interpolation in H.264/AVC. 6th Int. Conf. on ASIC, **1**:279-282.

Tsai, C.Y., Chen, T.C., Chen, T.W., Chen, L.G., 2005. Bandwidth Optimized Motion Compensation Hardware Design for H.264/AVC HDTV Decoder. 48th Midwest Symp. on Circuits and Systems, **2**:1199-1202. [doi:10.1109/MWSCAS.2005.1594322]

Wang, R.G., Huang, C., Li, J.T., Shen, Y.F., 2004. Sub-pixel Motion Compensation Interpolation Filter in AVS. IEEE Int. Conf. on Multimedia and Expo, **1**:93-96.

Wang, S.Z., Lin, T.A., Liu, T.M., Lee, C.Y., 2005. A New Motion Compensation Design for H.264/AVC Decoder. Proc. IEEE Int. Symp. on Circuits and Systems, **5**:4558-4561. [doi:10.1109/ISCAS.2005.1465646]

Zheng, J.H., Deng, L., Zhang, P., Xie, D., 2006. An efficient VLSI architecture for motion compensation of AVS HDTV decoder. *J. Comput. Sci. Technol.*, **21**(3):370-377. [doi:10.1007/s11390-006-0370-8]