An Area-Efficient VLSI Architecture for Variable Block Size Motion Estimation of H.264/AVC

Hoai-Huong Nguyen Le' and Jongwoo Bae[1]

[1] Department of Information and Communication Engineering, Myongji University San 38-2 Namdong, Cheoin-Gu, Yongin-Si, Gyeonggi-Do 449-728, Korea. {hoaihuong, jwbae}@mju.ac.kr

Abstract. In this paper, a highly efficient architecture for H.264/AVC variable block size motion estimation is proposed. The proposed architecture is based on the Propagate Partial Sum of Absolute Difference (PPSAD) architecture. A pipelined architecture is used to optimize the circuit of Processing Element (PE) and Row Adder Tree. The new architecture improves the system latency and hardware cost when removing small modes (4x4 and 4x8 modes) without affecting the coding quality. The proposed architecture is implemented in TSMC LVT 90 process, showing 26.47% improvement in the hardware cost and 18.3% in the frequency, compared to the previous PPSAD architecture.

Keywords: Motion estimation, H.264/AVC, variable block size, VLSI design.

1 Introduction

The H.264/Advance Video Coding (AVC) is an industry standard for video compression. The H.264/AVC standard was first published in 2003 [1], demonstrating the better compression efficiency and higher flexibility in video compression. Variable block size motion estimation (VBSME) which improves the coding efficiency by varying the size of blocks is adopted in H264/AVC. Generally, a video frame is divided into macroblocks (MBs) of size 16x16. Then each macroblock is segmented into subblocks of size 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4 as shown in Fig. 1. Therefore, there are 41 subblocks in a macroblock. Although the VBSME achieves good quality, its computational complexity is high compared with other algorithms. Various fast algorithms such as three-step search, four-step search, diamond search and hexagonal search have been proposed to reduce the computational complexity in motion estimation technique. Some of the improved fast motion estimation algorithms were adopted by H.264/AVC JM Reference Software. The hardware architecture for VBSME widely used in H.264/AVC is PPSAD architecture. We show a new architecture for PPSAD architecture in this paper.

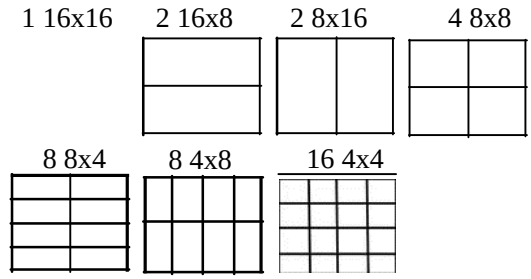


Fig. 1. Block sizes supported by H.264 for ME

This paper is organized as follows. In Section II, the background for the PPSAD architecture is explained first and then the proposed architecture is explained in detail. In Section **III,** experimental results are shown. Finally, Section IV concludes the paper.

2 Proposed Method

2.1 Background

Motion estimation is the key technique of video compression to reduce the temporal redundancies of image sequences for higher compression efficiency. Video encoding algorithms typically process one macroblock at a time. During the encoding process, the goal of motion estimation is to find the best match in the current frame from a reference frame by minimizing the cost function. Various functions have been proposed and analyzed in the literatures such as the mean absolute difference (MAD), mean square error (MSE) and sum of absolute difference (SAD). Due to its good performance and ease of hardware implementation, SAD is chosen for video coding as shown in Eq. **(1).**

$$SAD(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} |R(m+i, n+j) - C(m,n)| \quad (1)$$

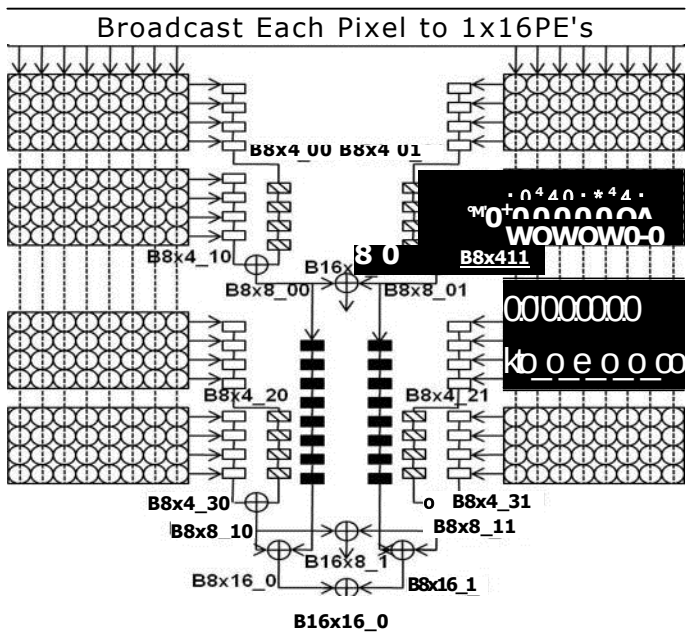
In Eq. (1), MxN is the size of the current block, (i, j) represents the motion vector. R(m,n) and C(m,n) are the pixels of the reference and current block, respectively.

The PPSAD architecture speeds up the computing process of VBSME algorithms by simultaneously calculating SAD values, and provides an efficient datapath. The original PPSAD architecture is first designed in [2] and the Parallel Improved PPSAD (PI-PPSAD) architecture is proposed in [3]. In fact, the dataflow of the PI-PPSAD is similar to that of the original PPSAD architecture. The current pixels are stored in the PE array. The 16x1 reference pixels are broadcasted to the PE array in every clock cycle. Each PE row

computes the SAD value of one row in a macroblock. One Row Adder Tree is accumulated with the partial SAD propagated in and then the result is propagated to the next stage vertically.

2.2 Proposed Architecture

In H.264 encoding process, more than 50% of the computation power is consumed by VBSME 7 modes algorithm. Many studies and researches have been performed on the VLSI design of VBSME. Especially in the case of HDTV applications, the large resolution and frame rate increase the system latency and hardware cost. By employing the mode elimination technique [4], the small modes can be ignored without affecting the coding quality. Since the 4x4 mode negligibly affects the coding quality, it is eliminated in our architecture. The availability of the 4x8 or 8x4 mode has enough ability to handle fine motion and results in good quality. Our proposed PPSAD architecture employs 5 modes: 16x16, 16x8, 8x16, 8x8 and 8x4 are shown in Fig. 2.



0 Processing Element f\N Shift Register for 8x4 Partial SAD
___ Row Adder Tree Shift Register for 8x8 Partial SAD

Fig. 2. Our proposed PPSAD architecture

In our PPSAD architecture, each PE contains an 8-bit register to store one pixel of the current macroblock. The reference pixels are broadcasted to every 1x16 PEs array in

every cycle. The PE is used to calculate the difference between the stored current pixel and the reference pixel, which is chosen from the reference memory through the input bus. Every Row Adder Tree is accumulated with the partial SAD values and then the result is propagated to the next stage in the vertical direction. Two SAD values of B8x4_00 and B8x4_01 are generated in the first four cycles. For the next four cycles, B8x4 10 and B8x4_11 are generated. Then, these SAD values are delivered to the below adder to form the large SAD values: B8x8_00, B8x8_01 and B16x8_0. Similarly, in the next eight cycles, SAD values of B8x4_20, B8x4_21, B8x4_30, B8x4_31, B8x8_10, B8x8_11, B 16x8_1, B8x16_0, B8x16_1 and B16x16_0 are generated.

To reduce the system latency and hardware cost, the pipeline operation is applied in order to optimize the PE and the adder tree in the PE row. The hardware implementation of the absolute difference operation is described in [5]. The last adder in every PE can be merged in the Carry Save Adder tree (CSA) of the Row Adder Tree. With the last adder eliminated in every PE, the number of remaining PE is 256, which leads to significant reduction in the hardware cost. The detailed design of Row Adder Tree for the Horizontal PPSAD architecture is shown in Fig. 3.

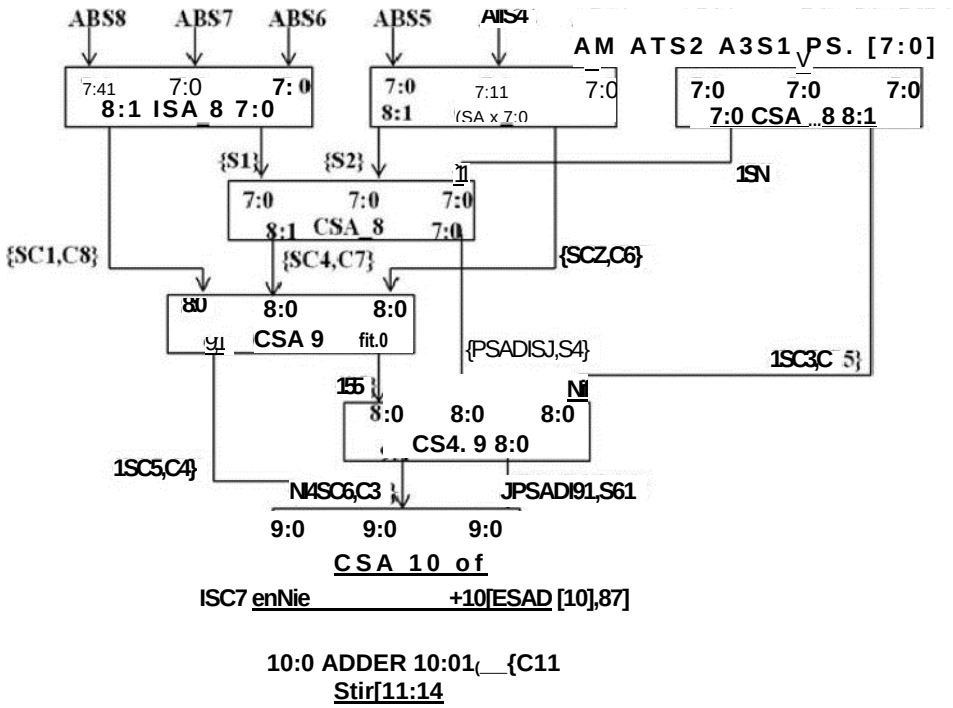


Fig. 3. Our proposed Row Adder Tree for our PPSAD Architecture

3 Experimental results

Our proposed architecture has been implemented in TSMC LVT90 process. The experimental results demonstrate the improvement in the system latency and the hardware cost compared to the previous PPSAD designs. The original PPSAD architecture is shown in [3]. All 16 partial SAD values of 4x4 blocks are propagated through the shift registers which becomes the critical path. The SAD values are accumulated to generate other SAD values resulting in longer system latency. The PI-PPSAD is designed in [2], which optimized the shift register for 4x4 blocks to reduce the hardware cost and system latency. Besides, this architecture shows the hardware of PE array and Row Adder Tree to save the adders in every PE array.

By employing the mode reduction technique, our architecture eliminates the trivial modes without affecting the coding quality. In addition, the improvement of the Row Adder Tree architecture reduces 256 adders in the whole system. The compared performances between our architecture and two previous PPSAD architectures are shown in Table 1. The PPSAD architecture can operate at 157.35 MHz. The hardware cost of the Horizontal PPSAD is improved by 33.35% when compared with the design in [3]. Compared to the architecture cost in [2], we reduce the hardware cost by 26.47% and increase the operating clock frequency by 18.3%.

4 Conclusion

In this paper, a highly efficient PPSAD and Row Adder Tree architecture are presented to improve the hardware cost and the operating clock frequency for VBSME of H.264/AVC. By eliminating 4x4 and 4x8 modes without affecting the coding quality, we reduce the hardware cost by 26.47% and increase the frequency by 18.3 %. Various architectures are compared with the proposed one. The experimental results show that the proposed architecture is the best choice considering the frequency and size. The proposed architecture can achieve the real-time encoding capability for high resolution applications such as HDTV and Ultra HDTV.

Table 1. The comparison with the previous architectures.

Design	Propagate Partial SAD [3]	PI-PPSAD [2]	Our **PP** SAD
The number of PE	256	256	256
Clock (MHz)	66.67	133	157.35
PE array & Cur. MB (gates)	79k	71.6k	52.65k

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (grant number 2011-0026386).

References

1. Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T Rec. H.264/ISO/IEC 14496-10 AVC). In Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-G050 (2003)
2. Y. W. Huang, T. C. Wang, B. Y. Hsieh, L. G. Chen: Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264. In Proceedings of ISCAS 2003, volume 2, pages 796-799 (2003)
3. Zhenyu Liu, Yiqing Huang, Yang Song, Satoshi Goto, Takeshi Ikenaga: Hardware Efficient Propagate Partial SAD Architecture for Variable Block Size Motion Estimation in H.264/AVC. In GLSVLSI'07, pp.160-163 (2007)
4. Yiqing Huang, Zhenyu Liu, Yang Song, Satoshi Goto, Takeshi Ikenaga: Parallel Improved HDTV720p Targeted Propagate Partial SAD Architecture for Variable Block Size Motion Estimation in H.264/AVC. In IEICE Trans. Fundamentals, vol.E91-A, no.4 (2008)
5. J.Vanne, E. Aho, T. D. Hamalainen, K. Kuusilinna: A high-performance sum of absolute difference implementation for motion estimation. In IEEE Transactions on Circuits and Systems for Video Technology, 16(7):876-883 (2006)