2012 4th International Conference on Intelligent and Advanced Systems (ICIAS)

UMHexagonS Based Motion Estimation Architecture Comparison

Arief Affendi Juri
School of Microelectronic Engineering
UniMAP
Perlis, Malaysia
s0930110441@studentmail.unimap.edu.my

Asral Bahari Jambek
School of Microelectronic Engineering
UniMAP
Perlis, Malaysia
asral@unimap.edu.my

***Abstract*— This paper is a presentation of the existing design and review of hardware for UMHexagonS based motion estimation for H.264/AVC video compression. Four existing motion estimation architectures that implement UMHexagonS are presented, analysed, and compared to show the area needing future improvements. The presented architectures are also compared against our proposed architecture. The results are compared in terms of gate count and throughput with emphasis on throughput as it shows the speed of the architecture. The proposed architecture gives a promising result compared with some of the reviewed architecture. It shows an improvement of 91% compared to the full search architecture and 20% compared to a basic UMHexagonS architecture.**

Keywords-component; UMHexagonS; Motion Estimation; architecture; H.246/AVC

I. Introduction

H.264 has been approved and finalized as International Standard 14496-10 (MPEG-4 Part 10) Advanced Video Coding (AVC) by ISO/IEC and as Recommendation H.264 by ITU-T on May 2003 [1]. It became the main standard used for good video quality especially for high-resolution video. It has been used in the latest video recording equipment, portable video recording devices, and smartphones. The high popularity of the H.264 standard is because of its ability to produce better video quality compared to the older video compression standards while minimizing the bitrate with a small increase in computational effort. Bitrate saving of 25% to 45% can be achieved compared to MPEG-4 Advanced Simple Profile (ASP) and a saving of 50% to 70% compared with MPEG-2 [2].

For the H.264/AVC encoder, motion estimation (ME) plays a crucial part in determining the quality of the compression in terms of picture quality, bitrate, or power consumption. For an H.264 encoder, 74.29% (234 GIPS) of computation and 77.49% (365 GByte/s) of memory bandwidth are required for ME purposed as discussed in [3]. From an experimental result in [4], it shows that from the whole H.264 encoding time, 60% is consumed by ME for one reference frame and approximately 80% for five reference frames.

Several architectures for H.264 have been proposed in the past to implement the capabilities of the H.264 encoder fully. In this paper, the existing architecture will be reviewed and compared to analyse the best H.264 architecture implementation.

II. Literature Review

In this section, four ME architectures will be discussed. The selections of the architectures are based on reported implementation of UMHexagonS architectures.

The architecture in [5] makes use of its ability to process five reference frames in real time. The architecture is shown in Figure 1. It contains six processing units (PU), five buffers 16x15 bytes each, one reference block memory of 16x16 bytes, one 4x16 reference block buffer and a comparing unit. It requires 32 bytes of memory bandwidth. The reference memory block cells are made of shift registers so that the values can be passed to different directions, depending on the horizontal/vertical (H/V) switch. The PU in Figure 2 contains 16 processing elements (PE), which are divided into 4 groups, 12 delay registers, and 37 adders. The outputs of each group of PEs are then summed using an adder tree, which gives an output of SAD values that are sent to the comparing unit for calculating the motion vectors. The comparing unit contains 41 comparing elements to compute all 41 SAD outputs. The buffers consist of 16x15 byte shift registers used to hold the overlap values of the successive candidate blocks that are processed by the same PU. This reduces the external memory access.

The architecture for UMHexagonS proposed in [6] focuses

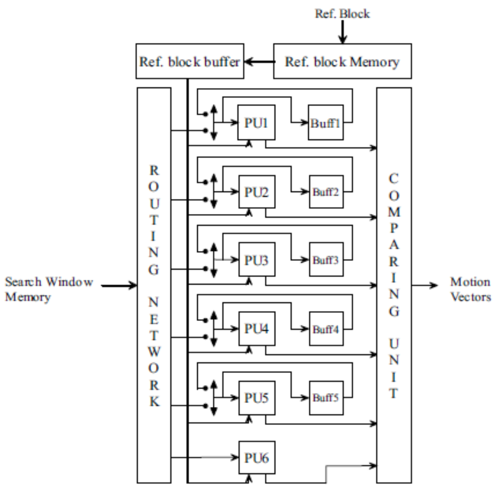

Figure 1: Past architecture in [5]

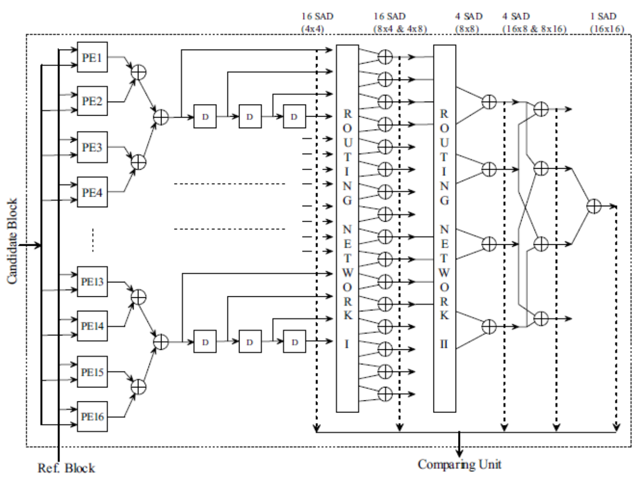

Figure 2: Process Unit of the past architecture in [5]

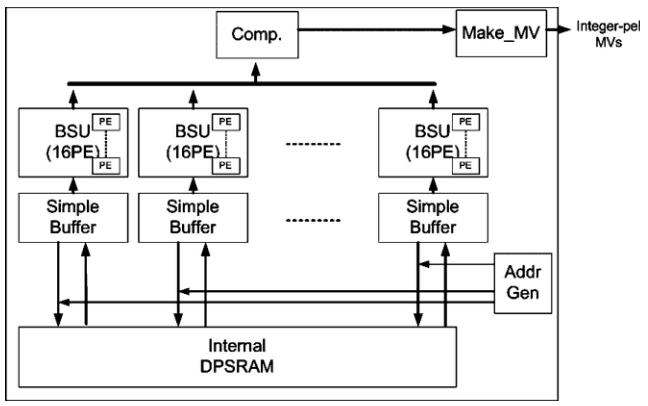

Figure 3: Past architecture in [6]

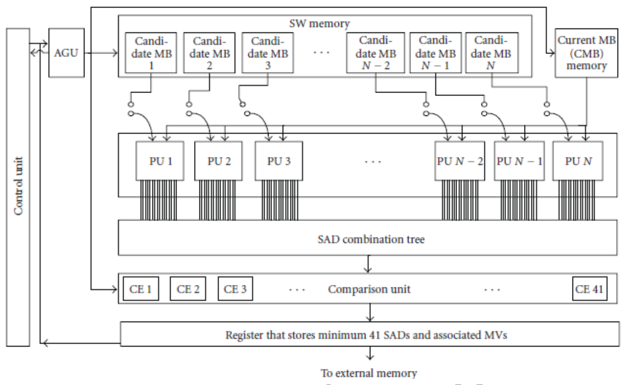

Figure 4: Past architecture in [7]

on a good tradeoff between small gate counts and high throughput. Figure 3 shows the block diagram of the said hardware architecture. It consists of Dual Port SRAM modules (DPSRAM), 16 simple buffers, 16 basic search units (BSU), a comparator, and a make motion vector (make_mv). The DPSRAM is used to store the current and previous frame. A BSU contains 16 (4x4) PE to calculate the SAD of one 4x4 block at a time to manage the variable block matching algorithm in UMHexagonS.

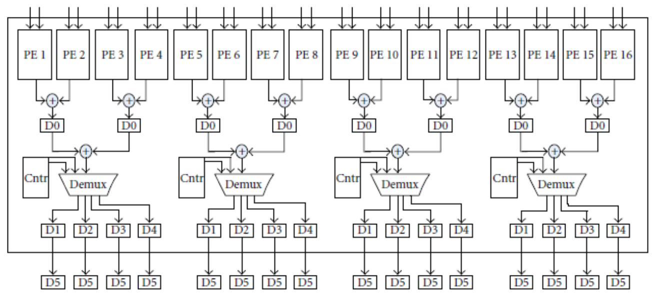

Figure 5: Process Unit of the past architecture in [7]

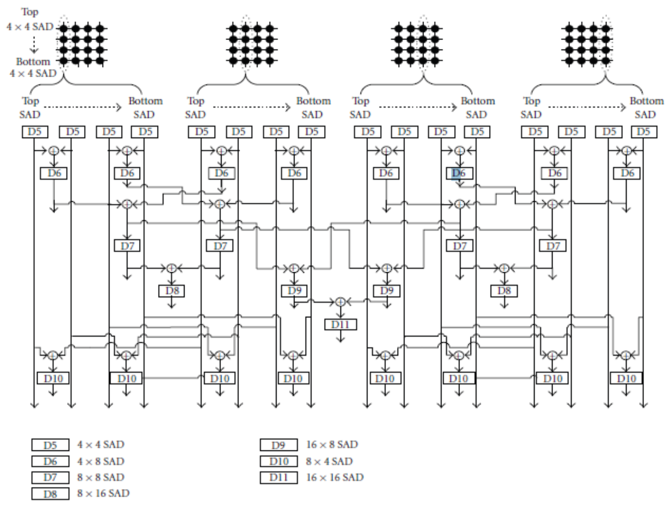

Figure 6: SAD Combination Tree of the past architecture in [7]

The proposed architecture in [7] uses a modified UMHexagonS algorithm known as Simplified Unified Multi Hexagon (SUMH) [8] to outperform the UMHexagonS. The top-level of the architecture is shown in Figure 4. The architecture is composed of search window (SW) memory, current MB (CMB) memory, an address generation unit (AGU), a control unit, a block of processing units (PUs), a SAD combination tree, a comparison unit, and a register for storing the 41 minimum SADs and their associated motion vectors. There are two internal memories in this architecture, the SW memory and the CMB memory where it uses dual port block RAMS (BRAMS) to store the search window and current MB, respectively. The SW memory consists of N 16x16 BRAMS for N numbers of MB candidates, where N depends on the desired search range. The PU contains 16 PE each as shown in Figure 5. Each PU will calculate 16 4x4 SADs for one candidate of MB. All 16 4x4 SADs will then further combine in the SAD combination tree to yield 41 SADs for each MB candidate as shown in Figure 6.

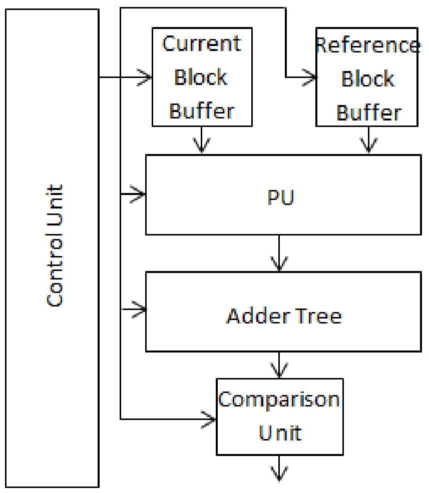

Figure 7: Proposed architecture

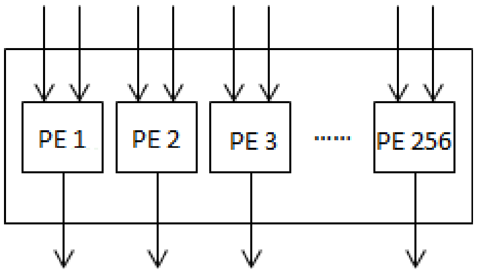

Figure 8: The Diagram of PU

III. PROPOSED ARCHITECTURE

In this section, the proposed architecture will be discussed. The proposed architecture is intended to implement efficient ME architecture that has higher flexibility for future improvement. The architecture utilises a direct implementation of the UMHexagonS to the architecture but using a parallel architecture. The top level of the proposed architecture is shown in Figure 7. It contains six main parts; a Control Unit, a Current Block Buffer (CBB), a Reference Block Buffer (RBB), a Processing Unit (PU), an Adder Tree, and a Comparison Unit. The control unit controls the entire unit to execute the algorithm. Details of the module operation will be explained in the following paragraph.

Both the Current Block Buffer and the Reference Block Buffer have the same architecture except for a different size. The CBB uses a 1024 byte SRAM to store a 32x32 search windows area while the RBB uses a 256 byte SRAM to store a 16x16 reference block. The data is stored in the related buffer before it is sent to the Processing Unit (PU).

The PU contains 256 Process Elements (PE), as in Figure 8. Each PE calculates a single absolute difference between the current and reference block. The PU will produce 256 absolute differences between a 16x16 area current block and a 16x16 area reference block. All 256 outputs of the PU will then be sent to the Adder Tree to calculate the sum of absolute differences (SAD).

Architecture	No. of PEs	Throughput (blocks/cycle)	Gate count
Full Search [6]	256	1/16384	350k
UMHexagonS[5]	96	1/490	NA
UMHexagonS[6]	256	1/1792	390k
SUMH[7]	256(*N*x16)	1/405	388k
This paper	256	1/1429	NA

Table 1: Comparison hardware architecture

The Adder Tree will calculate the SAD depending on the location and the size of the macroblock. It will generate 41 SAD outputs for 6 different macroblock sizes. By using all 256 absolute differences from the PU, the Adder Tree will calculate a total 16 4x4 macroblock SADs. With the SAD of the 4x4 macroblock, the Adder Tree will then calculate the sum for 2 more sizes, 8x4 and 4x8 macroblock with 8 SADs each. A total of 4 8x8 macroblock SADs will then be obtained from the SAD of an 8x4 or 4x8 macroblock. The 4 8x8 macroblock SADs will be used to calculate the sum of a 16x8 and 8x16 macroblock with 2 SADs each. Finally, the biggest macroblock SAD is calculated using either a 16x8 or an 8x16 macroblock, which makes 41 SAD outputs. The architecture of the Adder Tree is shown in Figure 9. The architecture for the adder tree is designed so that it will be able to provide all 41 possible outputs of different sizes and addresses.

Each of 41 SAD outputs will then be compared to the previous SAD value. The comparison unit is not only used to compare the value, but the lowest SAD value will also be kept temporarily for the next comparison. At the same time, the lowest SAD address will be kept as important data during motion vector (MV) calculation.

IV. RESULT AND DISCUSSION

Table 1 shows the comparison result of the hardware architecture. The result is compared in terms of gate count and throughput (block/cycles). The full search is given to show difference between full search and fast search (UMHexagonS). From the table, full search yields the lowest throughput (1/16384) as it searches all locations instead of only possible candidates. UMHexagonS architecture in [5] is designed to implement five reference frames while other architectures apply a single reference. To make it easier to compare, all of the results in this section will be based on a single reference frame. With single frame implementation, UMHexagonS [5] would give a throughput of 1/490. UMHexagonS [6] is designed mainly to describe the efficiency of VLSI architecture for UMHexagonS. Therefore, it can be a good reference for UMHexagonS architecture. It gives a throughput of 1/1792. As for SUMH [7], it is designed based on modified UMHexagonS and it yields a throughput of 1/405.

The proposed architecture would take three cycles to obtain the minimum SAD for the given point. The UMHexagonS algorithm has approximately 136 search points in a complete algorithm with extended hexagon search done 3 times. So 136x3=405 cycles per complete block including the initiation of the buffer unit, which takes 1024 cycles, the overall cycle needed will be 405+1024=1429.

The overall result is shown in Table 1. From the results, we can see that the full search [6] has the lowest gate count compared to other algorithms. However, the most important criterion for ME architecture is the throughput. This is represented by the number of cycles needed to complete a block. In terms of throughput, the SUMH [7] gives the best result. The proposed architecture shows a better result compared to the full search [6] by 91%. In term of other UMHexagonS architecture, our proposed architecture outperforms UMHexagonS [6] by 20%.

V. Conclusion

In this paper, the existing UMHexagonS architecture has been reviewed and compared. The SUMH architecture shows the best result compared to all reviewed architectures. The proposed architecture shows promising results where it outperforms the full search architecture throughput by 91% and UMHexagonS [6] by 20%. In the next works, we will improve the proposed architecture, especially in the buffer module, as this is the part that consumes most of the clock cycles in the proposed architecture.

References

[1] "Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITUT Rec. H.264 | ISO/IEC 14496-10 AVC)," in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVTG050r1, May 2003.

[2] T. Wiegand, H. Schwarz, A. Joch, F. Kossentini, and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," IEEE Trans. Circuits Syst. Video Technol., vol. 13, no. 7, pp. 688–703, Jul. 2003.

[3] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen, "Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder," IEEE Trans. Circuits Syst. Video Technol., vol. 16, no. 6, pp. 673–688, Jun. 2006.

[4] "Fast integer pel and fractional pel motion estimation for AVC," in Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, JVT-F016, December 2002.

[5] Choudhury A. Rahman and Wael Badawy, "UMHexagonS Algorithm Based Motion Estimation Architecture For H.264/AVC," Proceedings of the 9th International Database Engineering & Application Symposium (IDEAS), 2005

[6] Myung-Suk Byeon, Yil-Mi Shin, and Yong-Beom Cho, "Hardware Architecture for Fast Motion Estimation in H.264/AVC Video Coding," IEICE Trans. Fundam. Electron. Commun. Comput. Sci., vol. E89-A, no. 6,pp. 1744-1745,June 2006

[7] Obianuju Ndili and Tokunbo Ogunfunmi, "FPSoC-Based Architecture for a Fast Motion Estimation Algorithm in H.264/AVC," EURASIP Journal on Embedded Systems, vol. 2009, Article ID 893897, 16 pages, 2009.

[8] X. Yi, J. Zhang, N. Ling, and W. Shang, "Improved and simplified fastmotion estimation for JM," in Proceedings of the 16th Meeting of the Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Posnan, Poland, July 2005, JVT-P021.doc.

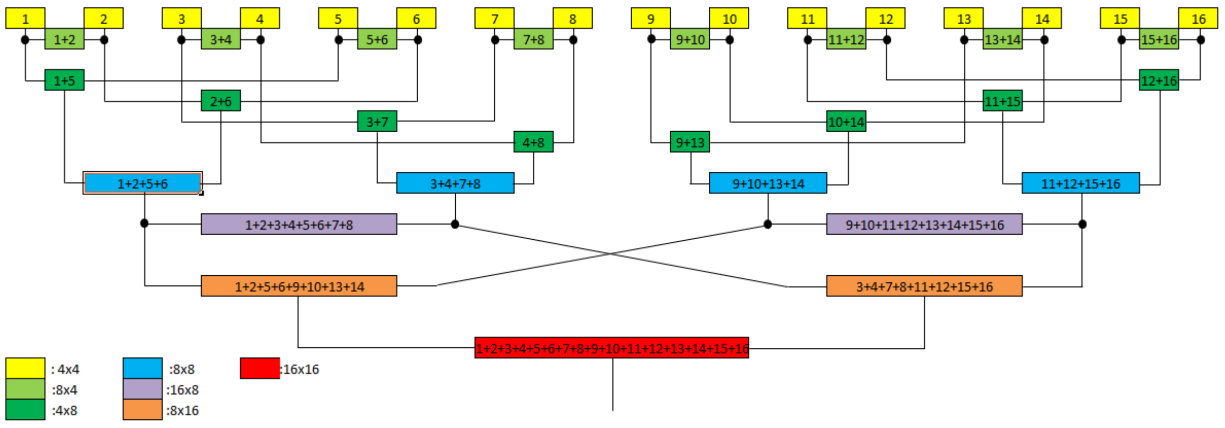

Figure 9: Adder Tree Diagram