Design and Analysis of Fast Search Motion Estimation Architecture for Video Compression

Asral Bahari Jambek[1], Yoong Yee Lai[2], and Arief Affendi Juri[3]
School of Microelectronic, University Malaysia Perlis
[1]asral@unimap.edu.my, [2]lyyee_1019@yahoo.com, [3]arief.affendi@gmail.com

***Abstract* – In the 21st century, video communication plays an important role in various multimedia communications, from internet streaming to HDTV broadcasting. A video usually consumes a great deal of storage space. Hence, video compression is needed. Motion estimation plays a vital role in video compression. This paper discusses the design and analysis of motion estimation using the unsymmetrical multi-hexagon search (UMHexagonS) fast search algorithm. The architecture consists of pixel buffers, processing elements, adder tree, comparator unit and control unit. The result shows that the architecture is able to perform the complete motion estimation effectively with a minimum clock cycle of 32612.**

I. INTRODUCTION

Video coding is important to compress large video data efficiently. The H.264 video coding standard, (also known as MPEG-4 AVC), has been developed for a broad range of applications from low bit-rate internet streaming applications to HDTV broadcasting with nearly lossless coding [1]. Motion estimation plays a vital role in video compression to remove the temporal redundancy between successive frames of video sequences [2].

Motion estimation can be implemented in several methods. However, the block-matching algorithm is the most famous and has been adopted by various video coding standards due to its simplicity and effectiveness in finding the best motion vector. The block matching algorithm finds the best-matched macroblock in a reference frame within a certain search window. Among various algorithms for block matching, a full search algorithm gives effective and optimal performance [3]. This algorithm, evaluates all of the search locations, which results in an optimal solution. However, due to the evaluation of many search locations, it has a high computational load. This high computational complexity makes the algorithm unsuitable for use in real-time applications.

In order to reduce the computational load, many fast search algorithms have been proposed and developed, such as 2D-logarithm search (2DLOGS) [4], three-step search (3SS) [5], new three-step search (N3SS) [6], four-step search (4SS) [7], block-based gradient descent search (BBGDS) [8], and diamond search (DS) [9, 10]. These fast search algorithms reduce the search points, hence reducing the computational complexity.

The latest H.264 reference software implements an unsymmetrical cross-shaped multi-level hexagonal grid search (UMHexagonS) algorithm as the fast search to determine the motion vector in its encoder [11]. This algorithm performs several search modes that improve the effectiveness and robustness in its motion prediction. The algorithm accelerates the search speed and without loss of accuracy.

This paper discusses the design and implementation of the fast search algorithm for motion estimation using an Unsymmetrical Multi-Hexagon Search (UMHexagonS). This paper is organised as follows. Section II reviews the existing technique of motion estimation. Section III is a discussion of the methodology utilized in this work. Section IV discusses the results obtained from the experiments. Finally, Section V concludes the paper.

II. UMHexagonS Algorithm

UMHexagonS has excellent performance as well as accuracy. This algorithm uses a multiple search pattern and utilises the Sum of Absolute (SAD) as the matching criteria. Aside from this, to reduce search time, if a minimal SAD is found early on, it can terminate the search. It provides good accuracy comparable to the full search algorithm but with better performance against time. This makes the algorithm suitable to be used for motion estimation. If an early termination is not implemented, there will be 124 candidate blocks for comparison in a search window. The UMHexagonS search flow diagram is shown in Figure 1. These comparisons are done in four step, consisting of unsymmetrical cross search, uneven multi-hexagon-grid search, extended hexagon-based search, and small diamond search, as shown in Figure 2 [12].

First, the unsymmetrical cross search is performed. Before implementing this search pattern, a centre point must be initialized. In this work, the top left corner of the candidate block for a current macroblock will be used as the initial location for the search centre. With the centre point initialized, the search is performed on the search area defined in the reference frame. A minimum SAD value is used to determine the best match for each search point.

Next, the uneven multi-hexagon-grid search will be performed. As with the unsymmetrical cross search, the location of the centre point search method in the search area is located at the top left corner of the candidate block. The SAD values are calculated and compared for each search point in order to find the minimum SAD values for

 Digital Object Identifier: 10.1109/ICIAS.2012.6306215

this search pattern. This minimum SAD value will then be compared with the minimum SAD value of the unsymmetrical cross search. The smallest among these values will be used as the search centre for the next step.

The extended hexagon-based search is performed in the third step. If there is a minimum SAD value among the 6 search points, the search location is this minimum SAD value will be used as the new search centre point. Then the hexagon-based search will be repeated until the minimum SAD value remains at the centre of the previous search centre. The last step in this algorithm is the small diamond search. Each search checks four search points. As in the extended hexagon search, this search is iterate until the best search is located at the same point as the previous search centre.

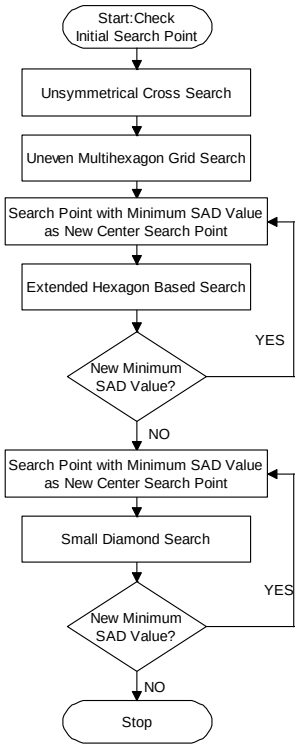

Figure 1: Detailed flow chart for UMHexagonS algorithm.

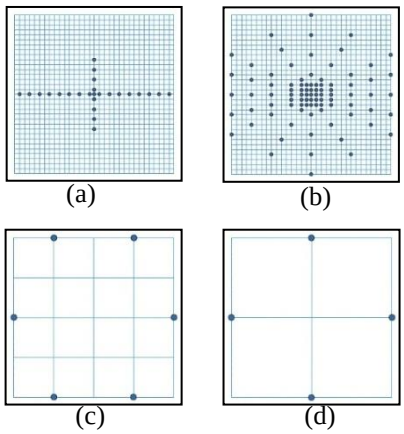

Figure 2: The search methods used in the UMHexagonS algorithm are: (a) unsymmetrical cross search, (b) uneven multi-hexagon-grid search, (c) extended hexagon-based search and (d) small diamond search.

III. METHODOLOGY

This section discusses the methodology taken to design the UMHexagonS architecture. The current macroblock consists of 16x16 pixels located in the current frame as shown in Figure 3. The current macroblock is processed in row-by-row fashion as shown in Figure 3 until the entire macroblock in the current frame has been predicted. A search area is a region that consists of 48x48 pixels (search range, p=16) of data that cover the possible range where the current macroblock is moved and is defined in a previous frame. Motion estimation is performed by comparing the minimum SAD value for one macroblock in the current frame with the search area defined in the reference frame. After all of the search locations have been evaluated for every current macroblock, the best matched candidate (i.e., the candidate that has the minimum SAD value) location is stored in the memory. These locations are then used to extract the 16x16 pixels and reconstructed as a predicted frame.

For each current macroblock, a search pattern, as shown in Figure 4, is implemented to find the smallest SAD value. Figure 5 shows the overall architecture to implement the UMHexagonS algorithm. This architecture consists of search area memory, current macroblock memory, a processing element, an adder tree, a comparator and a control unit. First, pixels for the candidate macroblock are stored in the current macroblock memory, whereas the search area pixels are stored in the search area memory. This current macroblock memory stores 16x16 pixels of data, whereas 48x48 search area pixels are stored in the search area memory. The output for the search area memory and current macroblock memory are connected to the processing element.

The processing element is one of the major parts for the architecture. It calculates the absolute difference value between the current pixel and the reference pixel. This design uses 256 processing elements (PE) to perform the computation. Figure 6 shows one of these PEs. Pixels that are stored in the current macroblock and search area memories are input into the current pixel register (CPR) and reference pixel register (RPR). These data are then used to perform the absolute difference between the current macroblock and search area pixels. This process is repeated until all current macroblock and reference macroblock pixels are evaluated. The results of the 256 absolute differences are input to the adder tree.

The adder tree, as shown in Figure 7, performs summation of the 256 absolute differences obtained from the processing elements. The sum of absolute different (SAD) values will be used in the next stage, which is a comparator module as shown in Figure 8. The comparator unit determines the minimum SAD value for each of the corresponding motion vectors. Register 1 in the comparator is first initialised to a maximum SAD value. The SAD value from adder tree will be compared to the value in Register 1. The smaller value will be stored in

Register 1. At the same time, the minimum SAD value will be output through the value stored in Register 2.

The control unit will determine the operation of all modules in Figure 5 according to the UMHexagonS algorithm. Once the minimum SAD is found for each best-matched macroblock in the search area, the SAD value will be stored in memory together with the best-matched location. The location of the search point is determined from the address decoder obtained in the control unit.

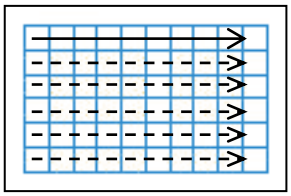

Figure 3 An example of video frame devided into 60 macroblocks; motion estimation is performed row-by-row.

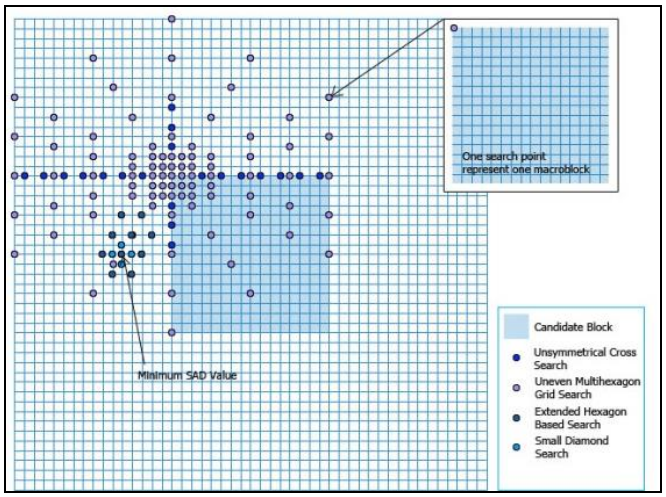

Figure 4 UMHexagonS algorithm search point in the search area.

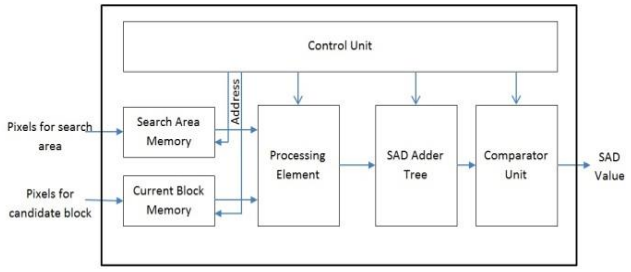

Figure 5 The proposed UMHexagonS architecture.

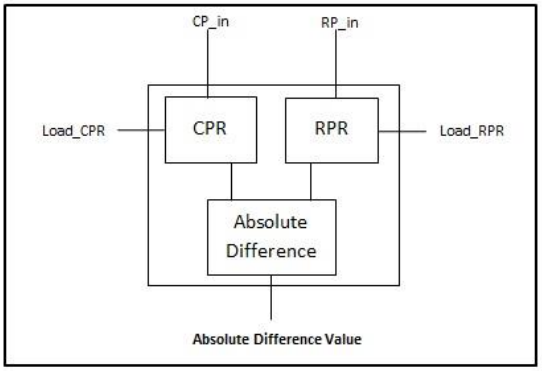

Figure 6 One of the processing element block diagram.

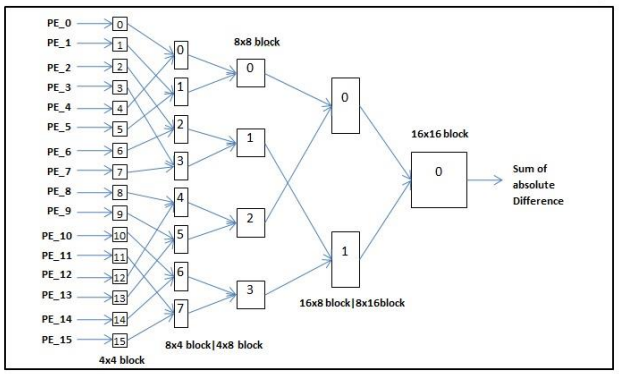

Figure 7 Adder tree architecture for the UMHexagonS.

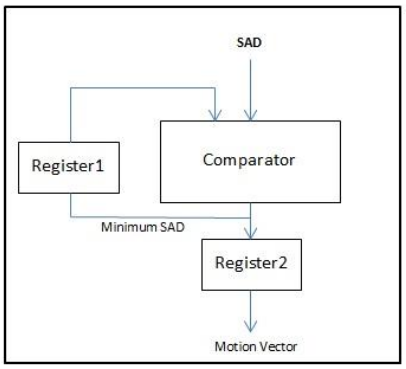

Figure 8 Comparator block diagram.

IV. RESULTS AND DISCUSSION

In this work, the hardware has been modelled using Verilog hardware description language (HDL) and simulated using Model-Sim software. Foreman video sequence (QCIF format) is used as a test bench during hardware simulation. Pixel data of these frames is stored in the current memory block and reference memory block respectively. A video frame is divided into several macroblocks. A macroblock represents 16x16 pixels of data (i.e., 256 pixels). Hence, there are 99 macroblocks for a resolution of 176x144.

Table 1 tabulates the total clock cycles used in a search point. In this architecture, one search point consumes 263 clock cycles to find a SAD value. This total

clock cycles are constructed by 1 clock cycle from the memory block, 257 clock cycles from a 256-bit buffer, 1 clock cycle from a processing element, 2 clock cycles from the adder tree and 2 clock cycles from the comparator.

The search pattern is composed of 4 searches. An unsymmetrical cross search contains 25 search points. Therefore, 25x263 = 6575 clock cycles are needed. An uneven multi-hexagon-grid search contains 89 search points. Therefore, 89x263 = 23407 clock cycles are needed. As extended hexagon-based search contains 6 search points. If there is no iterate loop, it requires 6x263 = 1578 clock cycles. Otherwise, if it needs ***x*** times the iterate loop, then ***x***x6x263 = 1578***x*** clock cycles are needed. A small diamond search contains 4 search points. If there is no iterate loop, it requires 4x263 = 1052 clock cycles. Otherwise, if it need ***x*** times the iterate loop, then ***x***x4x263 = 1052***x*** clock cycles are needed.

Figure 9 shows the current frame, reference frame and predicted frame results from performing motion estimation using the designed UMHexagonS. The output from the motion estimation is used to reconstruct the predicted frame. Currently, the hardware supports a 16x16 macroblock. In the future, this work will enhance the capability of the motion estimation architecture to include variable block size as recommended by the H.264/MPEG-4 AVC standard.

Table 1: Total clock cycles used in the UMHexagonS (a) clock cycle needed to load the data into each sub module (b) clock cycle needed during performing each search pattern.

Processing Modules	Clock Cycle
Memory Block	1
256 bits Buffer	257
Processing Element	1
Adder Tree	2
Comparator	2
Total Clock Cycles	***263***

(a)

Search Pattern	Clock Cycle
Unsymmetrical Cross Search	6575
Uneven Multi-hexagon-grid Search	23407
Extended Hexagon Based Search	1578
Small Diamond Search	1052
Minimum Total Clock Cycles	***32612***

(b)

(a) (b) (c)

Figure 9: (a) Current frame, (b) Reference frame and (c) Predicted frame from best-matched macroblock.

V. CONCLUSION

Motion estimation is an important part in a video compression process. There are many algorithms that can be used to apply motion estimation, such as fast search algorithm and full search algorithm. A fast search algorithm is preferable than a full search in term of computational time. In this project, UMHexagonS fast search algorithm is chose. This algorithm is divided into 4 steps in sequence, which is unsymmetrical cross search, uneven multi-hexagon-grid search, extended hexagon based and small diamond search. After completing this search pattern, motion vector is determined and this information will be used in the video compression process. The result shows that the motion estimation architecture can perform 16x16 block search effectively with a minimum of 32612 clock cycles. The next part of the work will enhance the capability of the architecture to include variable block size motion estimation as recommended by the H.264/MPEG-4 AVC standard.

VI. REFERENCES

[1] T. Wiegand, G. J. Sullivan, G. Bjntegaard, and A. Luthra,"Overview of the H.264/AVC video coding standard,"*IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 7, pp. 560-576, July 2003.

[2] Yair Moshe and Hagit Hel-Or,"Video block motion estimation based on Gray-code kernels," IEEE Trans. on Image Processing, vol.18, pp.2243-2254, oct.2009

[3] Tih-Chuan Lin and Shen-Chuan Tai, "Fast Full-Search Block-Matching Algorithm for Motion-Compensated Video Compression," IEEE Trans. Commun., vol.45, pp.527-531, may 1997

[4] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COM-29, pp. 1799–1806, Dec. 1981.

[5] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, Nov. 29–Dec. 3 1981, pp. G5.3.1–G5.3.5.

[6] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*,vol. 4, pp. 438–442, Aug. 1994.

[7] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*,vol. 6, pp. 313–317, June 1996.

[8] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for block motion estimation in video coding," *IEEE Trans. Circuits Syst.Video Technol.*, vol. 6, pp. 419–423, Aug. 1996.

[9] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center biased diamond search algorithm for block motion estimation," *IEEE Trans. Circuits. Syst. Video Technol.*, vol. 8, pp.369–377, Aug. 1998. 85

[10] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast blockmatching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp.287–290, Feb. 2000.

[11] "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.

[12] C. Rahman and W. Badawy, "UMHexagonS Algorithm Based Motion Estimation Architecture for H.264/AVC" *In Proceedings, Fifth International Workshop on System-on-Chip for Real-Time Applications*, 2005.