Journal of Real-Time Image Processing, May 2014, Springer International Publishing

Low-Energy Motion Estimation Architecture Using Quadrant-Based Multi-Octagon (QBMO) Algorithm

Asral Bahari Jambek and Arief Affendi Juri

School of Microelectronic Engineering,Universiti Malaysia Perlis, Pauh Putra Campus, 02600 Pauh, Perlis, Malaysia.

Phone: +6-04-988 5570

Fax: +6-04-988 5510

asral@unimap.edu.my

URL: http://asral.unimap.edu.my

Abstract: The H.264 video-coding standard is a great improvement on its predecessor in that it is able to save 50% of the bit-rate while maintaining the same quality as MPEG-4. However, its high computational complexity means the standard consumes large amounts of energy to process a video sequence, especially during motion estimation (ME) searches. To overcome this problem, a low-energy ME architecture is proposed in this paper that utilizes a quadrant-based multi-octagon search algorithm (QBMO) as one of its fast-search motion-estimation techniques. The proposed architecture is able to reduce the clock cycle needed to perform the search by 42% compared to the original conventional algorithm. This clock cycle reduction reduces energy consumption by up to 43%.

Keywords: H.264, UMHexagonS, motion estimation, architecture, low energy.

Introduction

The H.264/MPEG-4 AVC video-coding standard was developed by a Joint Video Team (JVT) consisting of experts from the Moving Picture Experts Group (MPEG) of the ISO/IEC and the Video Coding Experts Group (VCEG) of the ITU-T. The standard is used worldwide because of its superior performance compared to the previous standard. With PSNR 2 dB and 3 dB higher than MPEG-4 and H.263, respectively, it has become a favourite video-coding standard. Furthermore, with the same video quality, H.264 can save 50% bit-rate compared to MPEG-4 [1-3].

However, these advantages come at the cost of an increase in computational complexity: the encoding and decoding times increase three and two times, respectively, compared to H.263. Energy consumption also increases. For H.264, motion estimation (ME) consumes 70% (one reference frame) to 90% (five reference frames) of the total encoding time of H.264 [4-6].

This paper discusses the hardware implementation of the improved motion estimation algorithm used in H.264 reference software, known as the unsymmetrical multi-hexagon search algorithm (UMHexagonS). In our proposed method, the UMHexagonS algorithm is modified by implementing a quadrant-based multi-octagon-grid search algorithm (QBMO) in the fourth stage of the UMHexagonS algorithm. This method is able to reduce the number of candidates, and hence also the energy consumption.

The rest of this paper is organized as follows. Section II discusses the existing motion estimation using the UMHexagonS algorithm. Section III outlines our proposed architecture. The result obtained from our design is discussed in Section IV. Section V concludes the paper.

Literature review

UMHexagonS is a fast-search motion-estimation algorithm that consists of several fast-search techniques. The steps are initial-search-point prediction, unsymmetrical cross search, small full search, uneven multi-hexagon-grid search, extended hexagon-based search, and small diamond search [7-9]. Several methods have been proposed for ME implementation in H.264, since this is crucial to determine the quality and speed of the encoder. One of the earliest UMHexagonS architectures was proposed in [10], as shown in Figure 1. It makes use of the ability of H.264 to process five reference frames in real time, and consists of six processing units (PU), five buffers of 16x15 bytes each, one reference block memory of 16x16 bytes, one 4x16 reference block buffer and a

comparison unit. The reference memory is made up of shift registers to allow data to be moved vertically or horizontally, depending on the required data direction. The PU architecture, as shown in Figure 2, contains 16 processing elements, 12 delay registers and 37 adders. The 16 processing elements are divided into four groups. Values of sum of absolute differences (SAD) are calculated by the adder tree using the output of each PU. Then, the output is sent to the comparison unit for motion vector (MV) calculation.

The method proposed in [11] focuses on the trade-off between small gate counts and high throughput. The block diagram of the architecture is shown in Figure 3. The architecture contains dual-port SRAM modules (DPSRAM), 16 simple buffers, 16 basic search units (BSU) and a comparator. The DPSRAM module is used as a buffer to store the current and reference frame. Each BSU contains 4x4 processing elements (PE) to calculate the SAD of a single 4x4 block at a time.

The proposed architecture in [12] uses a modified UMHexagonS algorithm known as simplified unified multi-hexagon (SUMH) [13] to outperform the UMHexagonS. As illustrated in the top-level diagram shown in Figure 4, the architecture consists of search window (SW) memory, current macroblock (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 MVs. Both internal memories in the architecture use dual-port block RAMS (BRAMS) to store the SW and CMB. The SW memory consists of *N* 16x16 BRAMS, where *N* represents the numbers of macroblock (MB) candidates. Figure 5 details the implementation of the processing unit. From Figure 5, it can be seen that each PU contains 16 PE, and each PE calculates a single 4x4 block.

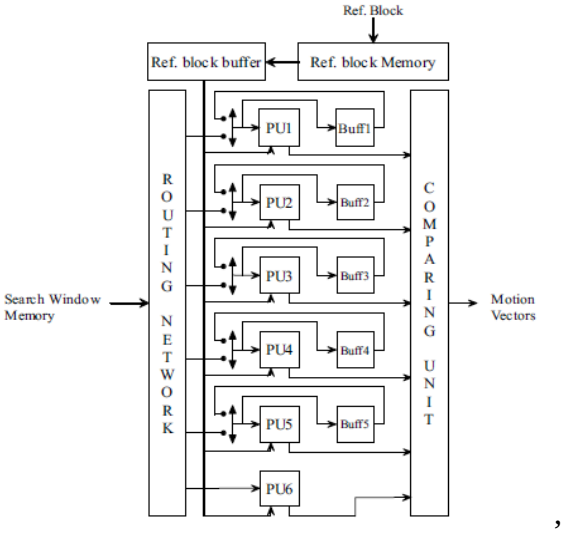


Figure 1: Architecture proposed in [10].

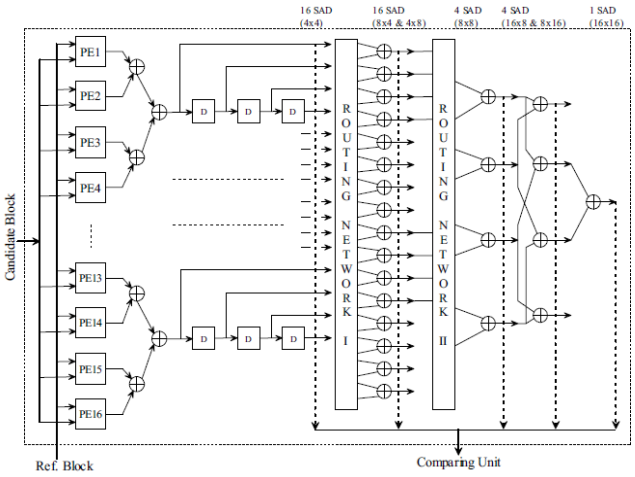


Figure 2: Processing Unit of architecture proposed in [10].

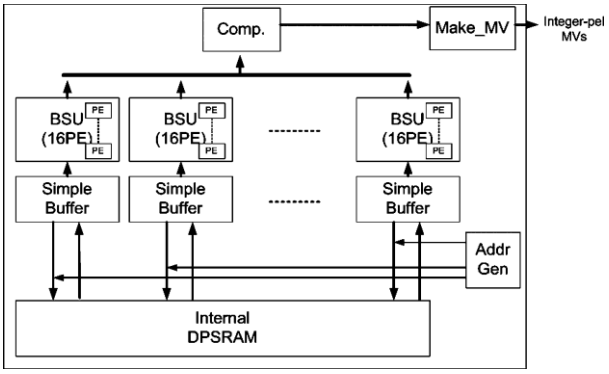


Figure 3: Architecture proposed in [11].

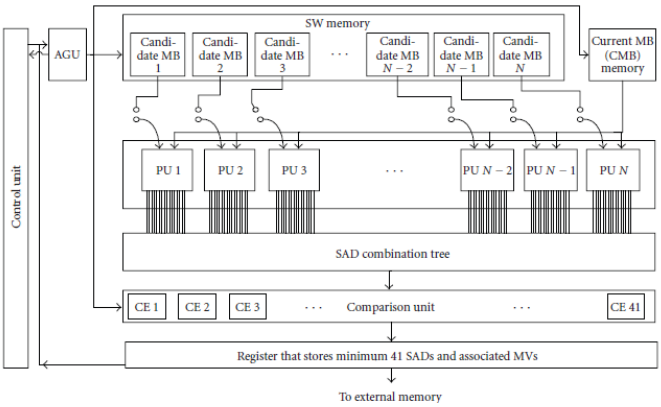


Figure 4: Top-level diagram proposed in [12].

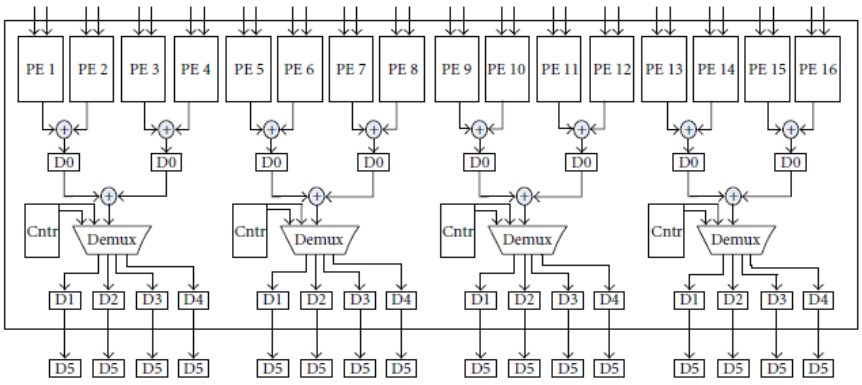


Figure 5: Processing Unit of the architecture in [12].

Proposed algorithm and architecture

This work aims to reduce the energy consumed by video compression. This is achieved by reducing the complexity of the ME by directly reducing the search point without sacrificing the video quality. As discussed in the previous section, the conventional UMHexagonS algorithm consists of six steps: (a) initial search point prediction, (b) unsymmetrical-cross search, (c) small full search, (d) uneven multi-hexagon-grid search, (e) extended hexagon-based search, and (f) small diamond search, with the number of search points for each step being 1, 24, 25, 64, 6 and 4 respectively [7-9].

In this work, the step with the highest search point will be focused upon since reducing these search locations will yield lower computational load and thus lowering the energy consumption. An uneven multi-hexagon-grid search has a very high search point of 64, constituting half of the total search points for UMHexagonS algorithm. Based on this observation, the uneven multi-hexagon-grid search will be focused upon here to reduce the overall search points.

The conventional UMHexagonS algorithm emphasizes the left and right sides of the search range to deal with horizontal video motion, especially when performing an uneven multi-hexagon-grid search. The result of this is further refined by performing an extended hexagon search and a small diamond search. Thus, the emphasis for horizontal video motion can be removed from the uneven multi-hexagon-grid search, since this task can be performed by an extended hexagon and a small diamond search.

Based on the above observation, to reduce the computational load for the conventional uneven multi-hexagon-grid search, this paper proposes to replace this search with a method called quadrant-based multi-octagon-grid search (QBMO). In this proposed method, the uneven multi-hexagon-grid search is first replaced by a multi-octagon-grid search as shown in Figure 6. Furthermore, instead of performing the search on all the multi-octagon candidates, this method performs the search on only one quarter of the multi-octagon-grid candidates. This is done by dividing the multi-octagon area into four quadrants, as shown in Figure 7. Only one quadrant will be used to look for the motion vector MV. The quadrant is chosen based on the location of the MV of the same block in the previous frame. For example, if the MV of the previous frame is located in the top left of the quadrant shown in Figure 8(a), that particular quadrant will be used for the search, as shown in Figure 8(b). By implementing the multi-octagon-grid search, the total candidates for the search are reduced from 64 to only eight during this step. This is equivalent to search point reduction of more than 80% compared to the original uneven multi-hexagon-grid algorithm.

To compare the complexity of the proposed algorithm, Table 1 summarizes the total number of search points for the conventional UMHexagonS (UMHS) and the modified UMHexagonS with QBMO replacing the multi-hexagon search (QBMO). From Table 1, it can be seen that the conventional UMHexagonS shows the highest number of search points at 123. QBMO shows a total number of search points of 67. This 45.53% reduction of search points is caused by implementation of QBMO at the fourth step of UMHexagonS.

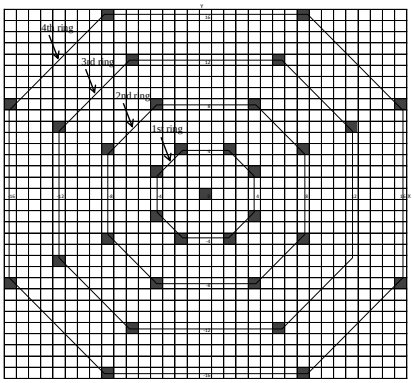


Figure 6: Multi-octagon-grid search.

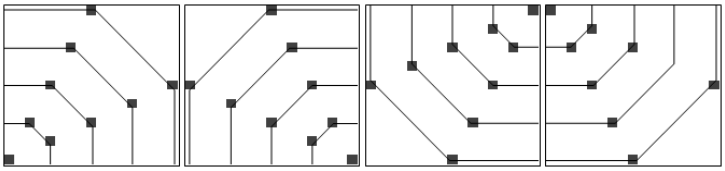
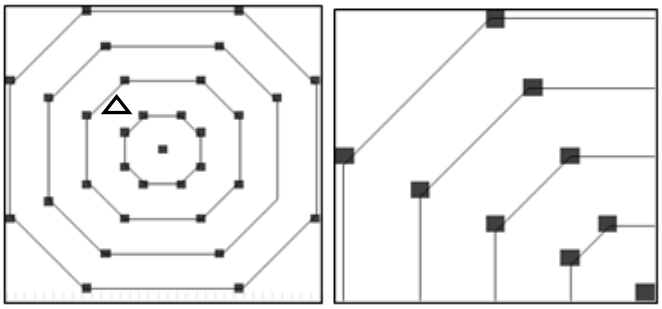


Figure 7: Four quadrants of the multi-octagon-grid search.



(a) (b)

Figure 8: Determining the multi-octagonal search quadrant: (a) the triangle mark is the MV of the block located in the previous frame, (b) the chosen quadrant for the current block.

Table 1: Summary of search points for each ME algorithm.

UMHexagonS Search Step	UMHS	QBMO
Unsymmetrical cross search	24	24
5x5 square full search	25	25
Uneven multi hexagon-grid search	64	8
Extended hexagon	6	6
Diamond search	4	4
Total	**123**	**67**

The proposed hardware implementation of the UMHexagonS algorithm with QBMO is illustrated in Figure 9. It consists of six main parts: a control unit, a current block buffer (CBB), a reference block buffer (RBB), a processing unit (PU), an adder tree (AT), a comparison unit (CU) and a control unit. The control unit plays a vital role in the architecture since it ensures the correct operation of the whole module in performing the proposed algorithm. This unit consists of a state machine as shown in Figure 10. In total, the state machine contains 12 states with five states controlling the top-level operation (s1, s2, s3, s4 and s5) and another five states controlling the PU sub-operation (st2, st3_1, st3_2, st4_1 and st4_2). The other two states are used for initialization and reset (srset and start).

For each MB, the process starts with a global initialize and reset state. Then, the buffer units (CBB and RBB) are initialized during states s1 and s2. Stages st2 to st4_2 are then executed. During the execution of these states, the state that performs PE, AT and CU is called up repeatedly until the final candidate has been evaluated. Then, it returns to the start state to be ready to process the next MB. This process is repeated until the last MB of the frame.

The RBB consists of a 2401-byte data buffer to store a 48x48 search window. The CBB consists of a 256-byte data buffer to store a 16x16 current block. The data is stored in the respective buffer before it is sent to the PU. Thus, the CBB and RBB require 256 and 2401 clock cycles, respectively, to store the data into the buffer. Both modules require 256 clock cycles to read out the current MB and reference MB data into PU.

The PU consists of 256 process elements (PE), as shown in Figure 11. Each PE calculates one absolute difference between the current and reference MB pixel. The PU will produce 256 absolute differences in one clock cycle. Once the absolute difference has been calculated, all 256 outputs of the PU will then be sent to the adder tree to calculate the SAD.

The adder tree will first calculate the 256 SAD before recombining the results to generate 41 final SAD, which represent the SAD of seven different block sizes, as shown in Figure 12. In the adder tree, the calculation is performed at five levels, where each level takes its input from the previous level. Each level generates different SAD for different block sizes. The first level generates SAD for the 4x4 block. The second level generates SAD for the 4x8 and 8x4 blocks. The third level generates the SAD for the 8x8 block. The fourth level generates SAD for the 8x16 and 16x8 blocks. Finally, the fifth level generates SAD for the 16x16 block. This adder tree requires four clock cycles to calculate all the 41 SAD.

The diagram of the comparison unit is shown in Figure 13. In this unit, each of the 41 SAD outputs will then be compared with the previously calculated SAD. The minimum SAD value will be kept temporarily for

the next comparison. At the same time, the lowest SAD block location will be kept, since it is used during the motion vector (MV) calculation. The comparison unit requires one clock cycle to perform the operation.

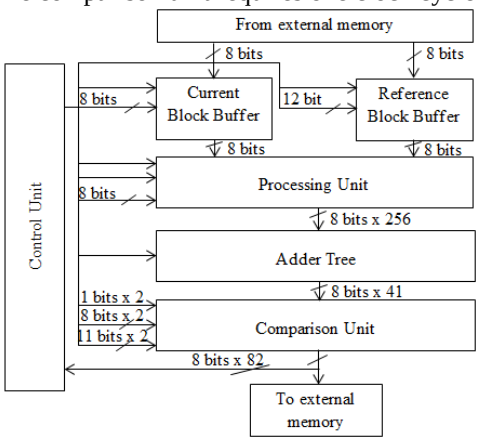


Figure 9: Top-level architecture of the proposed algorithm.

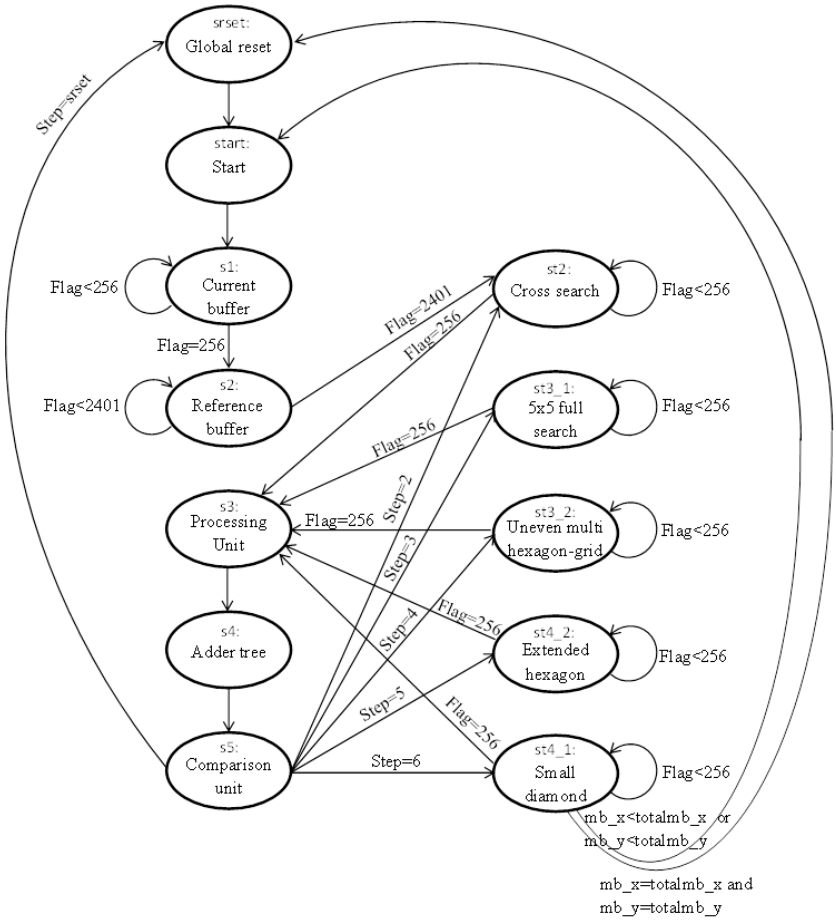


Figure 10: State machine flow for control unit

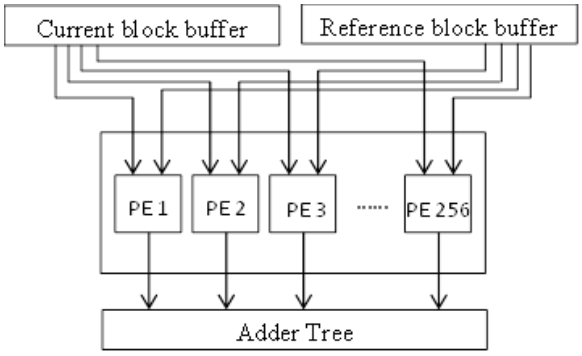


Figure 11: Processing unit (PU) diagram.

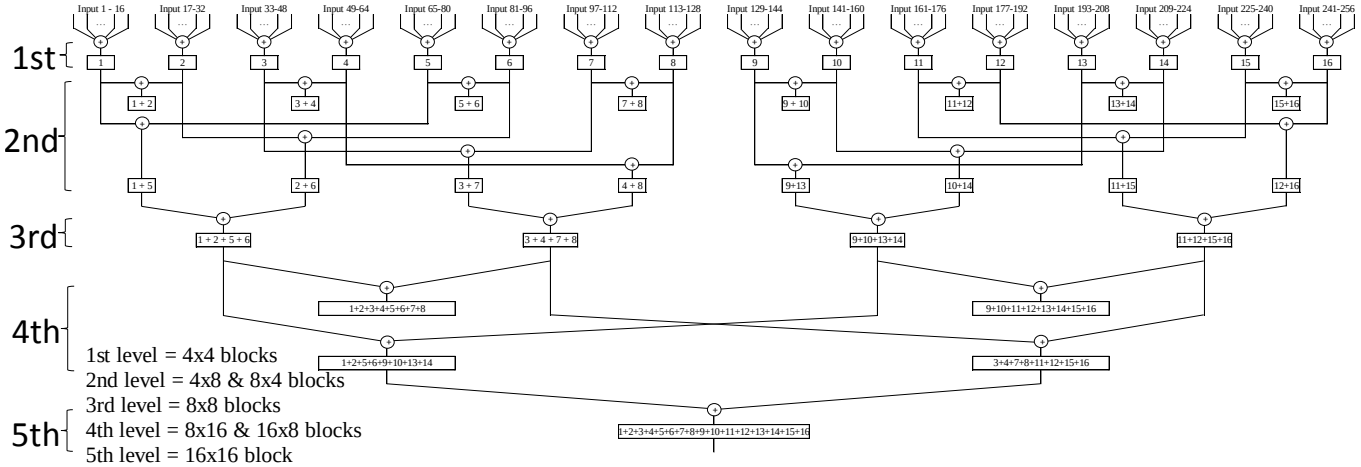


Figure 12: Adder tree diagram.

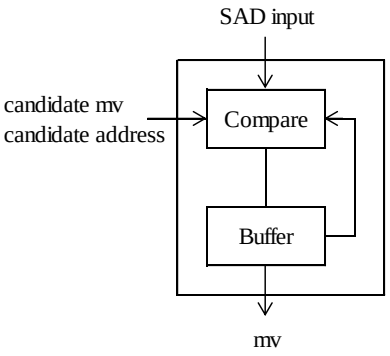


Figure 13: Comparison unit diagram.

Results and discussion

To implement the proposed QBMO algorithm, the algorithm is included in the reference software version JM17.2 by replacing the conventional uneven multi-hexagon-grid search. The simulation is performed using the standard encoder baseline profile, with the parameters as listed in Table 2. Six test video sequences (CIF format) categorized into three motion types are used. These are: aggressive motion (Bus and Football video), medium motion (Foreman and Silent video) and low motion (News and Hall video). Since there is no standard ways to measure the picture quality metrics, PSNR value is used as the measure throughout this work. This helps us in comparing our result with the existing works. Furthermore, in cases where accuracy is important, subjective quality will be used as the final comparison.

As discussed in the previous section, the computational load reduction for QBMO is achieved by reducing the number of search locations. However, this reduction could cause a drop in the prediction quality. To evaluate the prediction quality for the proposed QBMO algorithm, the reference software is simulated and the output is monitored in terms of PSNR, motion estimation simulation time and bit-rate. The result is tabulated as shown in Table 3.

From Table 3, the QBMO is able to achieve ME encoding time (MET) reduction without significantly sacrificing the quality of the video and the bit-rate. The average decrease in PSNR is less than only 0.01dB with a maximum PSNR drop of 0.02dB. These small changes in PSNR, however, will not be visible to the human eye [14].

Similarly, the reduction of the search candidate could affect the bit-error-rate optimization. From Table 3, on average of only a 0.5% bit-rate increase can be observed for the QBMO, as compared to the conventional UMHexagonS. The worst bit-rate increment for the QBMO is 1.18% or 18.72 kb/s.

To further evaluate the algorithm performance in real-world implementation, Figures 14 (a) to (c) show PSNR vs. bit-rate for three video sequences that represent each type of video motion. From Figure 14, the QBMO can be seen to closely match the conventional UMHexagonS algorithm. This shows that the QBMO is effective in reducing the computational load of the UMHexagonS without sacrificing the overall H.264 compression efficiency.

Based on the results shown in Table 3 and Figure 14, it is clear that the proposed QBMO is able to reduce the computational load without degrading the video output quality significantly. One of the reasons for this is the implementation of the multi-octagon search with the selective quadrant method. Since neighbouring videos have high similarity in terms of video motion, using the previous frame as the guide to determine the selected quadrant results in good prediction accuracy. This proposed method therefore does not degrade the picture quality and compression efficiency significantly.

In order to compare the algorithm performance with the standard method, the proposed QBMO algorithm is implemented in the hardware. The architectures are coded using Verilog hardware description language and simulated using VCS from Synopsys Inc. During the behavioural simulations, the architectures' operation and the timing of the simulated architecture are verified. Once synthesized, the gate-level simulation is performed to verify the timing and energy consumption of the architectures.

The motion estimation behaviour is simulated using a test bench that contains a sample video with known MV. The test bench is fed into the architecture and the simulation output is monitored. The calculated MV must match the known MV to ensure the architecture functions as intended.

Table 4 shows the clock cycle requirement for both the conventional and the proposed algorithm. In total, the conventional UMHexagonS algorithm requires 123 search points per MB. From the simulation results, the algorithm requires 34,632 clock cycles to calculate a single MB, and consumes 13,678,852 clock cycles to process one video frame.

The proposed QBMO algorithm has similar clock cycle requirements to the conventional architecture, except that it only evaluates 67 search locations per MB. Due to the lower number of candidates, as the simulation results show, the proposed algorithm requires 19,960 clock cycles to calculate a single MB and 7,883,412 clock cycles to process one frame sample video. Thus, the proposed algorithm results in 42% fewer clock cycles compared to the conventional UMHexagonS architecture. This proves that the proposed QBMO algorithm greatly saves on clock cycle requirements.

In order to evaluate the energy consumption of the proposed algorithm, the architectures are compiled and simulated using Power Compiler from Synopsys Inc. From the simulation result, the total energy consumption can be deduced. Table 5 shows the power result for each architecture module. From Table 5, it can be seen that the power consumption for both the conventional and the improved UMHexagonS are similar, with PU consuming the largest amount of power (40%), followed by control unit (23%) and RBB (19%).

In order to determine the total energy for the proposed algorithm, the results obtained from the simulated power are used to calculate the energy needed to process one frame. Table 6 shows the result of the energy consumption for the proposed algorithm. From Table 6, it is clear that the proposed architecture uses less energy (16.8 mJ) compared to the conventional UMHexagonS algorithm (29.7 mJ). The results show that the proposed architecture is able to save on energy by 43% compared to the conventional method. The reduction in energy is due to the proposed QBMO algorithm requiring fewer clock cycles to complete the same task as the conventional UMHexagonS.

Table 2: H.264 parameters for the algorithms' simulation.

Parameter	Value
Profile	Baseline
Level	4.0
Codec	JM17.2
Image format	CIF (352x288 pixels)
MV search range	32
Frame rate	30 fps
RD optimization	On
Total number of reference	5
Sequence type	IPPP
Entropy coding	CAVLC
Encoded frames	100
Motion Estimation for component	Y (luma)

Table 3: Motion estimation execution time result.

Algorithm	UMHS			QBMO			Performance Comparison		
	MET (s)	PSNR (dB)	Bitrate (kb/s)	MET (s)	PSNR (dB)	Bitrate (kb/s)	Reduce in MET (%)	Drop in PSNR (dB)	Increase in Bitrate (%)
Bus	463.58	34.93	1202.44	405.3	34.91	1214.82	12.57	0.02	1.02
Football	628.63	36.7	1566.99	513.87	36.71	1585.71	18.26	-0.01	1.18
Foreman	327.91	36.86	394.95	309.69	36.87	397.4	5.56	-0.01	0.62
Silent	244.72	36.16	247.21	237.9	36.17	247.54	2.79	-0.01	0.13
News	223.23	38.35	214.86	212.01	38.35	214.54	5.03	0.00	-0.15
Hall	206.28	37.82	241.31	207.09	37.82	241.25	-0.39	0.00	-0.02
Average	**349.06**	**36.80**	**644.63**	**314.31**	**36.81**	**650.21**	**7.30**	**0.00**	**0.46**

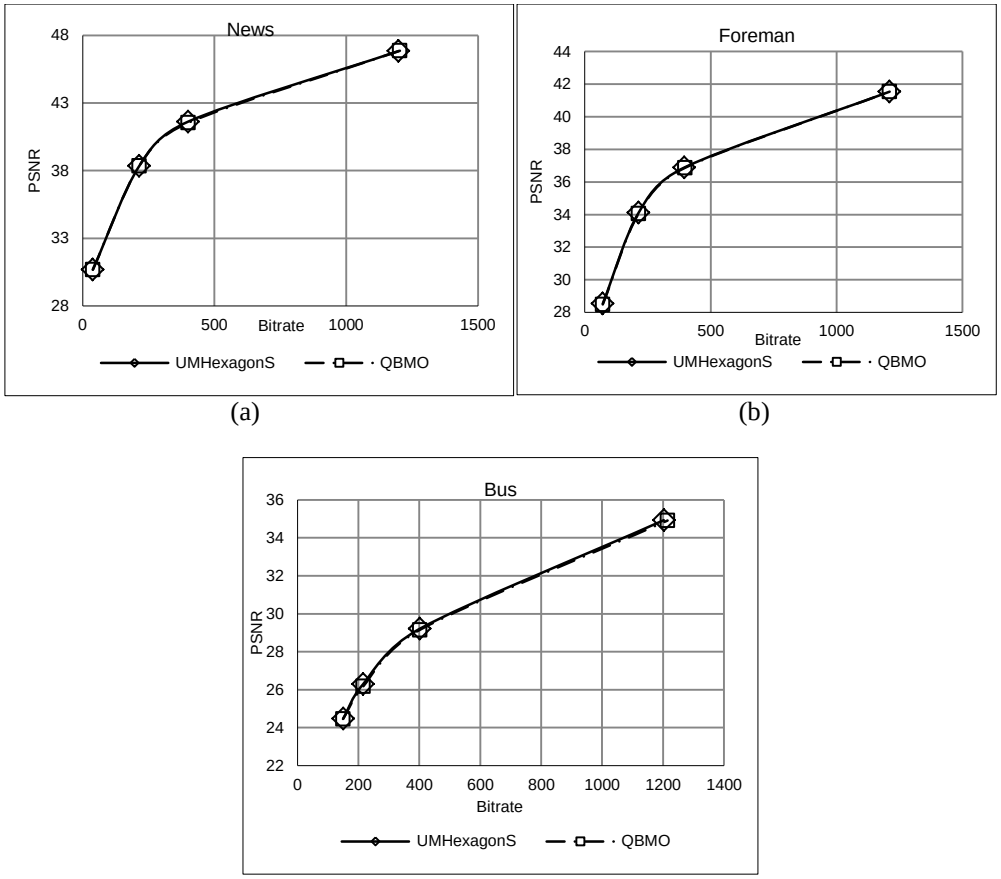


Figure 14: PSNR vs. bit-rate for various frame sequence (a) News (b) Foreman, and (c) Bus.

Table 4: Clock cycle to process one macroblock.

Moule	Conventional UMHexagonS Algorithm	Proposed Algorithm using QBMO
CBB (initialize)	256	256
RBB (initialize)	2401	2401
CBB (read)	256	256
RBB (read)	256	256
PU	1	1
AT	2	2
CU	1	1
Total search points	123	67
Total clock cycle (per MB)	34632	19960
Total clock cycle (per frame)	13678852	7883412

Table 5: Breakdown of power analysis for each architecture module.

Module	Conventional UMHexagonS Algorithm	Proposed Algorithm using QBMO
Control Unit	5.48 µW	4.99 µW
CBB	1.02 µW	1.02 µW
RBB	4.03 µW	4.02 µW
PU	8.39 µW	8.43 µW
Adder Tree	1.64 µW	1.57 µW
Comparison Unit	1.15 µW	1.26 µW
Total	21.71 µW	21.29 µW

Table 6: Result for energy analysis.

Architecture	Total dynamic (µW)	Clock cycle to process one frame	Time to process one frame (ms)	Energy required to process one frame (mJ)
Conventional UMHexagonS Algorithm	21.71	13678852	1.37	29.7
Proposed Algorithm using QBMO	21.29	7883412	0.79	16.8

Conclusion

This paper has discussed the design of low-energy motion estimation for H.264 video coding. An improved UMHexagonS algorithm has been presented to reduce the computational load. The algorithm implements quadrant-based multi-octagon-grid search (QBMO) in the fourth step of the UMHexagonS algorithm. Based on the experimental results, the proposed architecture is able to reduce energy consumption by 43% compared to the conventional algorithm, without affecting the prediction quality.

References

1. Jimenez-Moreno, A., Martinez-Enriquez, E., Diaz-de-Maria, F.: Mode Decision-Based Algorithm for Complexity Control in H.264/AVC. Multimedia, IEEE Transactions on, vol 15, Issue 5, 1094 - 1109 (2013).
2. Weiyao Lin, Panusopone, K., Baylon, D.M., Ming-Ting Sun, Zhenzhong Chen, Hongxiang Li: A Fast Sub-Pixel Motion Estimation Algorithm for H.264/AVC Video Coding. IEEE Transactions on Circuits and Systems for Video Technology, vol 21, No 2, 237 - 242 (2011)
3. Jianning Zhang, Yuwen He, Shiqiang Yang and Yuzhuo Zhong: Performance and Complexity Joint Optimization for H.264 Video Coding. Proceeding of International Symposium on Circuit and System, vol. 2, 888-891 (2003).
4. Zhou Wei, Zhou Xin: A fast hierarchical 1/4-pel fractional pixel motion estimation algorithm of H.264/AVC video coding. 8th IEEE Conference on Industrial Electronics and Applications (ICIEA), 891-895 (2013).
5. Ates, H.F., Altunbasak, Y.: Rate-Distortion and Complexity Optimized Motion Estimation for H.264 Video Coding: IEEE Transactions on Circuits and Systems for Video Technology, vol.18, no.2, 159-171 (2008).
6. Zhibo Chen, Yun He: Fast Integer and Fractional Pel Motion estimation. Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Geneva (2002).
7. Yufeng Li, Jufei Xiao, Wei Wu: Motion estimation based on H.264 video coding. 5th International Congress on Image and Signal Processing (CISP), 104-108 (2012).
8. Xie Lifen, Huang Chunqing, Chen Bihui: UMHexagonS search algorithm for fast motion estimation. 2011 3rd International Conference on Computer Research and Development (ICCRD), 483-487 (2011).

9. Peng Huang, Cui-Hua Li: Irregularity-cross multi-hexagon-grid search algorithm for fast motion estimation on H.264. 2nd International Conference on Computer Engineering and Technology, Vol. 3, 587-592 (2010).
10. Rahman, C.A., Badawy, W.: UMHexagonS Algorithm Based Motion Estimation Architecture for H.264/AVC. *International Workshop on System-on-Chip for Real-Time Applications*, 207 – 210 (2005).
11. Myung-Suk Byeon, Yil-Mi Shin and Yong-Beom Cho: Hardware Architecture for Fast Motion Estimation in H.264/AVC. Journal IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E89-A, Issue 6, 1744-1745 (2006).
12. Obianuju, N., Ogunfunmi, T.: FPSoC-Based Architecture for a Fast Motion Estimation Algorithm in H.264/AVC. EURASIP Journal on Embedded Systems (2009).
13. Xiaoquan, Y., Jun, Z., Nam, L., and Weijia, S.: Improved and simplified fast motion estimation for JM. 16th Meeting of the Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG. Poznan, Poland (2005).
14. Thomos, N., Boulgouris, N., & Strintzis, M.: Optimized transition of JPEG2000 streams over wireless channels. IEEE Trans. Image Processing, vol 15(1), 54 – 67 (2006).