Hardware Implementation of Motion Estimation Technology Using High Level Synthesis and Investigations into Techniques for Improvements

Shota Nagai, Takashi Kambe†, Gen Fujita‡
Graduate School of Science and Engineering, Kindai University
3-4-1 Kowakae, Higashi-Osaka City, Osaka, Japan
†Depart. of Electric and Electronic Engineering, Kindai University
‡Faculty of Information and Communication Engineering,
Osaka Electro-Communication University, Osaka, Japan

Abstract— Due to the trend to increasingly large screen sizes and resolutions, the volume of data required by recent video images is enormous. Video coding technology improves every year, but a pressing issue is that processing times are also increasing. With this as a motivation, the current study targeted the motion estimation technology that is a key part of the H.264/AVC (Advanced Video Coding) standard, implemented it as hardware using high-level synthesis technology, and investigated improvements. An EPZS (Enhanced Predictive Zonal Search) algorithm was implemented instead of a Full Search algorithm, and the results evaluated to understand the effectiveness of the high-level synthesis technology and of the speedup techniques that were adopted.

I. Introduction

The volume of data required by recent video images is enormous due to the advent of larger screen sizes and high quality 4K and 8K television technology. Video coding technology improves every year, but an issue is that processing times are also increasing because of the amount of data. To cope with this, various types of hardware are designed, but RT level design often takes too long to meet time-to-market requirements. In this paper, we propose a hardware implementation methodology for the motion estimation process in H.264/AVC [1]. We use the EPZS algorithm to speed up the motion vector search process. Bach C high level synthesis technology [2] is used for circuit synthesis and the effectiveness of each design technique is evaluated.

II. H.264/AVC motion estimation technology

Motion estimation technology can effectively eliminate redundancy that exists within image sequences. Block Based Motion Estimation techniques are the most widely used due to their relative implementation simplicity and efficiency [1, 3]. The current frame is divided into square *blocks* of pixels. For each of these blocks, the algorithm tries to find a best-matched block in a previous frame using a predefined distortion measure. The displacement of the reference and best-matched block defines a *motion vector (MV)*. This best match is then used as a *predictor* for the block in the current frame. The encoder then only needs to send the MV and a residue block in order to recover the original reference block. The distortion measure most commonly used is *the sum of absolute differences (SAD)* because of its simplicity. The algorithm that examines all these locations is called *Full Search (FS)* and requires a significant part of the computational power of an encoder.

III. design methods based on high level synthesis

In this section, a C-based design methodology for memory access, loop processing, data structure, and parallel processing is described.

A. memory access

For many applications, memory access is often a major bottleneck. The simplest way to access large amounts of data at high speed is to store all the data in on-chip memory, but this increases chip size and is expensive.

A popular search method is spiral search, but this often reads the same pixels multiple times while searching out spirally from the center. To reduce these duplicated memory accesses, an array register is provided that stores sixteen lines. The register is accessed from top-left to bottom-right. After the SAD calculation, each line is shifted up and only one new line is added to the bottom of the array. Although the array register causes an increase in circuit size, it improves processing time for spiral searches.

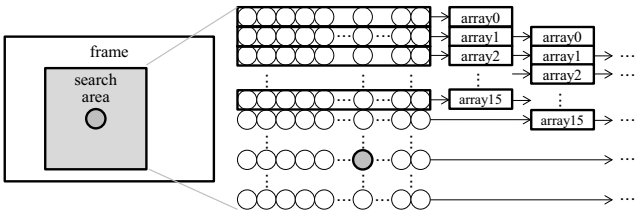

Fig. 1. Memory access

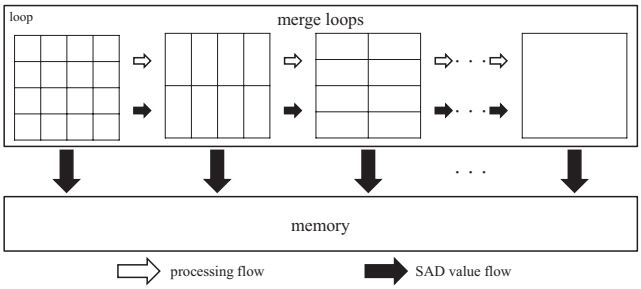

Fig. 3. After merging

B. *loop processing*

In motion estimation technology, data memory for each pixel is used when the SAD value is calculated incrementally for each block size from 4×4 to 16×16. SAD calculation is done for larger blocks using the results from the smaller blocks. This calculation accesses the data memory repeatedly in the SAD calculation for FS.

Fig. 2 shows loop processing and memory access in each block size from 4x4 to 16x16. This loop processing is to calculate the cost for each block type. Nested loop processing requires large amounts of memory access and is a major bottleneck in the motion estimation calculation. However, because each loop has the same count, loop fusion can applied to the nested loops as shown in Fig. 3. This loop fusion is able to remove all memory reads in each loop.

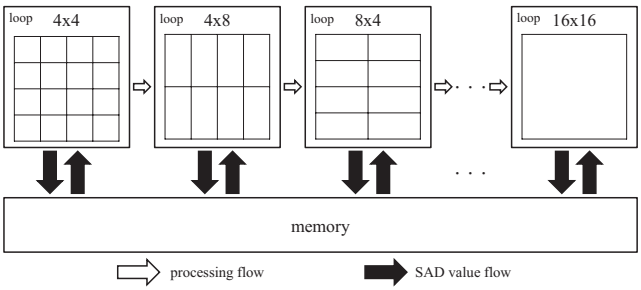

Fig. 2. Before merging

C. *circuit size*

To achieve smaller circuit size while maintaining high performance, the number of bits for each variable is optimized by verifying its correctness using simulation.

In cases where the functionality of code blocks is the same, high level synthesis is able to optimize circuit reusability. However, if there is a small difference in the C description between two loop structures such as, for example, in the cost calculation for the luma prediction in the current frame, then the loops cannot be optimized. These loops are therefore merged using if-branches to deal with the differences between them and thus reduce gate size.

D. *parallel processing and pipelining*

For data processing of large volumes of data, parallel data processing is often effective. Because processing of each 4×4 block in the SAD calculation is independent, sixteen processes can be executed in parallel by dividing a 16×16 pixel block into sixteen 4×4 pixel blocks as in shown in Fig. 4. Although it is possible to apply parallel processing to the 4×4 pixel blocks and execute 256 parallel processes, this has a large circuit overhead and does not actually achieve faster processing. Pipeline processing sometimes achieves better speed improvement with only a small increase in circuit size, so we decided to apply it to the 4×4 pixel block processing.

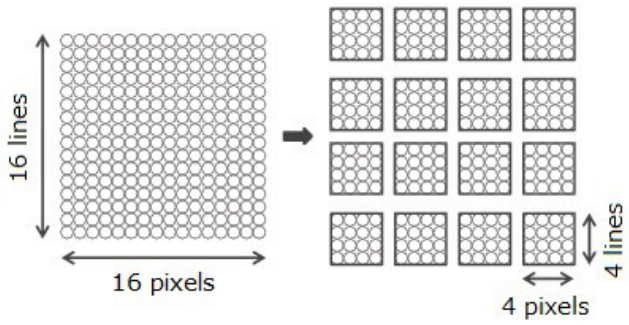

Fig. 4. 16 parallel processes

E. *composite data structure*

Composite data structures are often used in C programming for design efficiency. However, when these are used for data transfer among different functions, then high level synthesis generates many redundant wires and registers. The H.264 reference software has many composite data structures for the video, input parameters and so on. For smaller circuit size and better performance, these should

be replaced with only the variables needed for each function.

IV. Implementation of the Enhanced Predictive Zonal Search (EPZS) Algorithm

Recently, various kinds of software algorithm have been proposed to speed up motion estimation. Of these, we chose the EPZS [4] Algorithm to implement as hardware. The reason for selecting this method is that, due to improvements in the MV search method, it has better performance than other algorithms such as Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) [5] and Advanced Predictive Diamond Zonal Search (APDZS) [6].

A. EPZS algorithm

Because the EPZS algorithm considers several predictors in the predictor selection phase, the pattern of the search can be considerably simplified. The algorithm also adopts a more robust and efficient adaptive thresholding calculation due to the high efficiency of the prediction stage. The predictor candidate set of MVs consist of three subsets A, B and C. Subset A has the median value of the motion vectors of the three adjacent blocks, the blocks on the left, top, and top-right from the current position. Subset B adds the MV at the (0,0) position to that of subset A. And subset C has MVs which have increasing or decreasing acceleration information of the same spatial position in the reference frame (Fig. 5). Subset C also has MVs at neighbor blocks (top, bottom, left and right positions) in the reference frame (Fig. 6). By using these subsets, EPZS can predict MVs quickly and accurately.

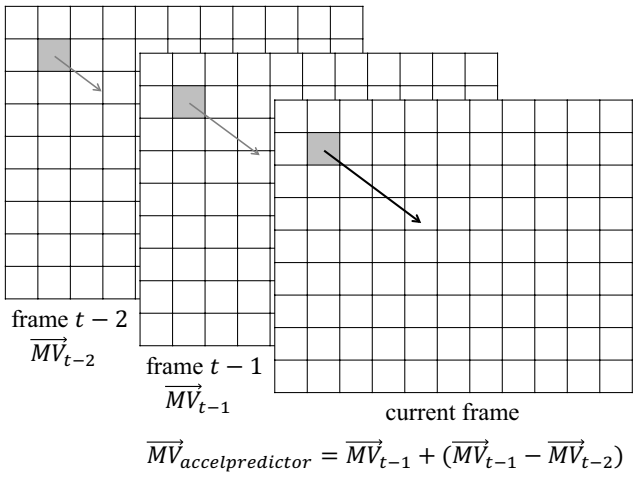

Fig. 5. Subset C

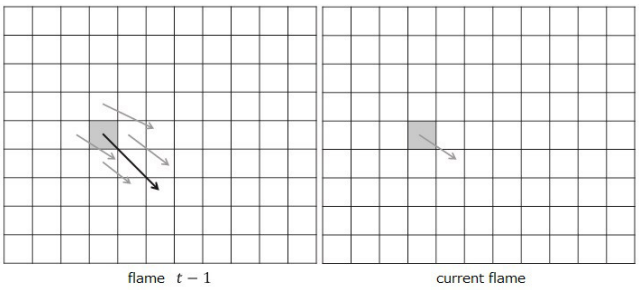

Fig. 6. Additional subset C

The threshold value is calculated using the minimum SAD value of three adjacent blocks and of the collocated block in the previous frame to avoid early the search termination or errors that occur in conventional methods. The threshold value is formulated as follows:

$$T_k = a_k \times min(MSAD_1, MSAD_2, ..., MSAD_n) + b_k \quad (1)$$

where a_k and b_k are fixed parameters and $MSAD$ represents the SAD value for each subset.

EPZS is also better than other methods in its choice of search pattern. Because combinations of several search patterns often produce better results, EPZS uses several types of small area search to reduce the processing time (Fig. 7).

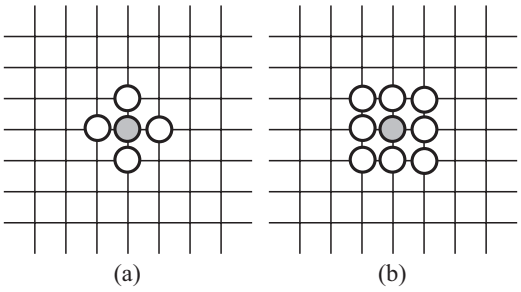

Fig. 7. Small area search

B. hardware implementation of EPZS

Subset C has two kinds of MV. One kind are MVs that are in spatially the same position across several reference frames and the other kind are MVs that are in neighbor blocks in reference frames. However MV information needs to be kept in memory and frequent memory accesses are required. To reduce this overhead, we use several types of fixed point patterns in the spiral search search area (Fig. 8).

In this paper, to maintain prediction accuracy, we added an MV at the left-top position block to subset B

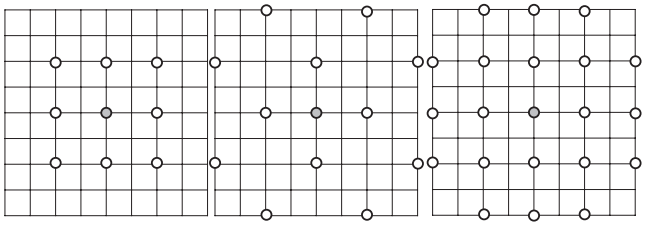

Fig. 8. Fixed points for subset C for 8×8 block

which has already been calculated. Furthermore, to speed up EPZS, the threshold value is fixed for each block size and its permissible range is set to ±2 pixels. When the costs of all MVs calculated using subsets A,B and C are larger than the threshold, the MV that has the minimum cost is chosen.

V. design results and their evaluation

Before the proposed EPZS algorithm was implemented in hardware, its performance was evaluated as shown in Table I. Three kinds of software algorithm were compared using the BD-BR metric (average bit rate difference in % over the whole range of peak signal-to-noise ratio, [7]). The first software algorithm is EPZS of the JM 18.0 reference software. The second is only subsets A and B of our EPZS implementation. And the third is three fixed points searches (8, 12, and 20) for subsets A, B and C in our EPZS implementation. Two test video data, "foreman.cif" and "football.cif" were used. These were to test "low activity" and "high activity" respectively.

The comparison showed that the BD-BR of the proposed algorithms are about 3% to 6% worse than JM 18.0. One reason may be a threshold control issue. The BD-BR of the "high activity" data is better than the "low activity" data when the number of fixed points are increased, but the effect of subset C is still small compared with JM 18.0. We continue to improve our EPZS implementation.

TABLE I
Comparison of software algorithms

	BD-BR[%]	
content	foreman.cif	football.cif
EPZS of JM18.0	0.278	3.072
Subset A, B	2.934	6.317
Subset A, B, C (8 points)	4.789	5.329
Subset A, B, C (12 points)	4.997	5.510
Subset A, B, C (20 points)	5.784	4.538

In this paper, we choose "Subset A, B, C (8 points) " of TABLE I for hardware implementation. Bach C system version 3.6 was used to synthesize to a VHDL RT level description and the gate level circuits were synthesized from RT level HDL using Design Compiler 2009 provided by VDEC and mapped to Hitachi 0.18 micron CMOS library cells. The clock frequency of all circuits is 100 MHz and the memory access time is 5 ns. The simulation time is measured for searching MVs for one macro block.

Table II compares processing times and circuit sizes for the proposed design methods. (a) is simply the C to Bach C manual translation result. (b) - (h) are the results of incrementally applying the methods described in Section III. (b) - (c) and (e) - (g) are for speed up of FS. (i) is the application our EPZS implementation (number of fixed points 8) to design (d) and (h) without (b),(c) and (e) - (g). The circuit processing time for (h) is about 4.8 times faster than that for (a) and its circuit size is about 45% smaller than (a). Moreover, the processing time for (j) is about 32 times faster than that for (a) and its circuit size is about 43% smaller than (a). The integer motion estimation portion of this circuit is about 50% slower than [8]. We continue to improve the EPZS implementation to get better both performance and BD-BR.

TABLE II
Processing time and circuit area results

	processing time [ms]	circuit area [gates]
(a) sequential	179.388	1,282,345
(b) memory access	111.288	1,535,288
(c) loop processing	108.032	1,562,476
(d) reduce circuit size	151.963	763,208
(e) 16-parallel	44.184	816,998
(f) 256-parallel	38.325	1,908,852
(g) pipeline	39.270	790,866
(h) data structure	37.328	705,673
(i) EPZS	6.253	502,069
(j) EPZS with pipeline	5.665	551,976

VI. Conclusion

In this paper, we proposed a C-based design method and applied it to the MV algorithm of H.264/AVC. We also designed a new EPZS algorithm by changing subsets and threshold control. This obtained higher speed processing with only small deterioration in picture quality.

We are continuing to improve both the BD-BR and the performance of the proposed algorithm and plan to apply our methods to the H265/HEVC standard [3].

Acknowledgements

The authors would like to thank the Bach system development group at SHARP Corporation, Electronic Components and Devices Development Group, for their help

in hardware design using the Bach system. This work is supported by the VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys Corporation.

References

[1] I. E. Richardson: "The H.264 Advanced Video Compression Standard," Publisher: Wiley; 2 edition (August 9, 2010).

[2] K.Okada, et al. : "Hardware Algorithm Optimization Using Bach C," IEICE Trans. Fundamentals vol.E85-A, No.4, pp835-841, 2002.

[3] Sullivan, G. J., et al.: "Overview of the High Efficiency Video Coding (HEVC) Standard." Circuits and Systems for Video Technology, IEEE Transactions on 22(12), pp.1649-1668, 2012.

[4] A.M.Tourapis : " Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation, in proceedings of Visual Communications and Image Processing, pp.1069-1079 (2002).

[5] A.M. Tourapis, O.C. Au, and M.L. Liou : "Predictive Motion Vector Field Adaptive Search Technique (PMVFAST) -Enhancing Block Based Motion Estimation, in proceedings of Visual Communications and Image Processing 2001 (VCIP-2001), pp.883-892 (2001).

[6] A.M. Tourapis, O.C. Au, and M.L. Liou : "New Results on Zonal Based Motion Estimation Algorithms - Advanced Predictive Diamond Zonal Search, in proceedings of 2001 IEEE International Symposium on Circuits and Systems (ISCAS-2001), vol.5, pp.183-186 (2001).

[7] G. Bjontegaard: Calculation of average PSNR difference between RD-curves, VCEG-M33, Austin, Texas, USA, April 2001.

[8] Y. Ningmei, et al.: "A high-performance configurable VLSI architecture for integer motion estimation in H.264", 13th International Symposium on Integrated Circuits (ISIC), 2011.