An Efficient Architecture of Binary Motion Estimation for MPEG-4 Shape Coding

Yi-Chu Wang, Hao-Chieh Chang, Wei-Ming Chao and Liang-Gee Chen

DSP/IC Design Lab, Department of Electrical Engineering
National Taiwan University, Taipei, Taiwan, R.O.C.
{chuu, howard, hydra, lgchen}@video.ee.ntu.edu.tw

ABSTRACT

This paper presents an efficient architecture of binary motion estimation (BME) for MPEG-4 shape coding. This architecture, called DDBME, mainly consists of a data dispatch based 1-D systolic array and a 16×32 bit search range buffer. In DDBME, bit parallelism technique is applied on the SAD calculation of block matching algorithm. In order to support efficiently bit-data parallel processing, bit addressing should be taken into consideration. The data dispatch technique is applied on 1-D array by the hardwired data flow routing such that the bit addressing operations can be efficiently reduced. The DDBME operating at 7.29 MHz can handle the real-time requirement for encoding MPEG-4 shape sequence at core profile level 2, i.e. 2 VOs with CIF format, 30 fps and assuming each frame contains 30% boundary macroblocks in average. For the same real-time specification, the optimized software running on RISC (Ultra Sparc, 300MHz) can only achieve 1/20 performance.

Keywords: motion estimation, shape coding, MPEG-4, architecture

1. INTRODUCTION

Object-oriented presentation is one of the key functionality of MPEG-4 video standard[1]. It provides the capability to directly access the video contents (objects) in video scenes at today's multimedia communication systems. In order to support this functionality, the shape information (including position and shape) of an arbitrarily shaped video object has to be transmitted in addition to video texture information. In general, the shape information of a video object is represented by a binary alpha plane, which consists of binary pixels that indicate which part of the video scenes belongs to an object. Under the constrained transmission bandwidth, the compression of shape information is also required.

MPEG-4 video standard adopts macroblock (MB, 16x16)-based shape coding. The basic coding unit of binary alpha plane is also called a binary alpha block (BAB). Fig. 1 shows the block diagram of the binary shape coding system. For the coding the opaque or transparent BAB, only block type has to be encoded and transmitted. While for boundary BAB, context-based arithmetic coding (CAE) is applied. The binary motion estimation (BME) is performed to obtain best-matched BAB for inter-mode CAE so as to achieve higher coding efficiency. Each boundary BAB can be sub-sampled before intra-mode or inter-mode CAE for the purpose of rate control.

The run-time profiling of binary shape coding system implemented on Ultra Sparc (300 MHz) shows that it takes 320 seconds for the encoding of *child* sequence (100 CIF VOPs). The throughput is 0.25 VOPs per second. Clearly, the real-time coding requirement cannot be achieved. Among the shape coding system, 90% computing power is spent on BME. Optimization on BME is essential to remove the bottleneck in achieving real time shape coding. The value of binary shape could be transparent or opaque, hence they can be represented as one bit per pixel. These bits can be packed, stored in memory and loaded to processor so that they are processed in parallel. However, the data for block matching algorithm[2] may not align the packing and storage boundary. Extra bit addressing operations are therefore required to access certain of bits in one storage entry. These operations consume more computation power than other arithmetic operations in BME algorithm. Some BME architectures[3,4] for traditional motion estimation that involves transforming pixels of eight-bit resolution into one bit representation and applying the block matching on this bit plane are proposed. In those architectures, the bit-packing storage and access are not taken into consideration, therefore a larger search range buffer and logic gates for bit addressing become inevitable. It is inefficient to directly adopt those BME architectures for shape coding. By exploiting the computation features, we propose an architecture to remove the huge computation load of BME and efficiently solve the bit addressing problem.

The organization of this paper is shown as below. Section 2 briefly introduces BME for MPEG-4 shape coding. Section 3 analyzes the computation complexity and throughput for BME algorithm. Section 4 describes the proposed architecture. The implementation results are summarized in Section 5.

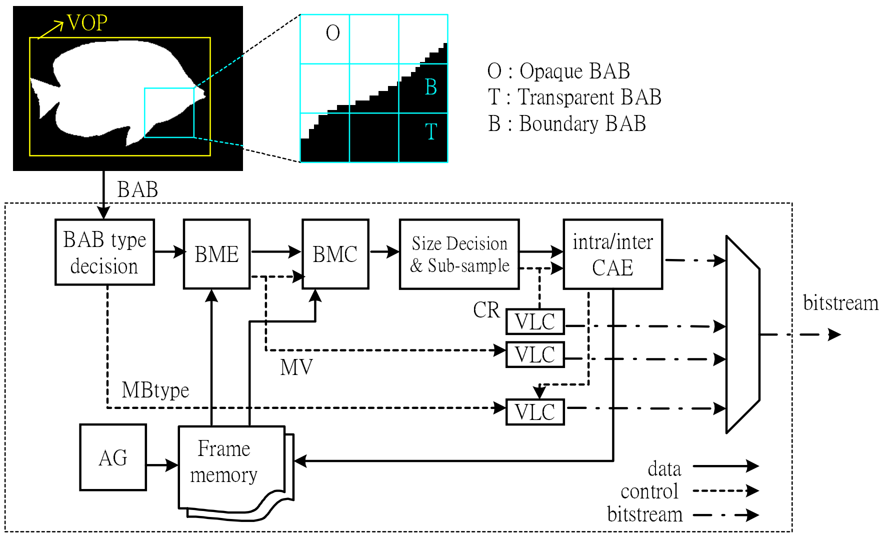


Fig. 1 Block diagram of shape coding system.

2. BINARY MOTION ESTIMATION FOR MPEG-4 SHAPE CODING

Motion estimation is used to remove temporal redundancy in video sequence and improve coding efficiency. The block-matching algorithm[2] is widely used in many video standards, and is also recommended in MPEG-4 shape coding. The matching criterion of BME is to find the minimal sum of absolute difference (SAD) between current BAB and candidate BAB in search range.

Predictive motion estimation is adopted in MPEG-4 shape coding in order to get better block matching results. The adoption priority of motion vector predictor for shape (MVPs) is shown in Fig. 2. By traversing the order of {MVs1, MVs2, MVs3, MV1, MV2, MV3}, MVPs is determined as the first encounter motion vector in the list that is defined. If no neighboring BAB or block has motion vector, MVPs is set to 0. When the number of different pixel between reference BAB indicated by MVPs and current BAB is less than a predefined threshold, the MVPs is directly employed as motion vector of shape (MVs). Otherwise, full-search binary motion estimation is performed around the MVPs with search range from –16 to 15.

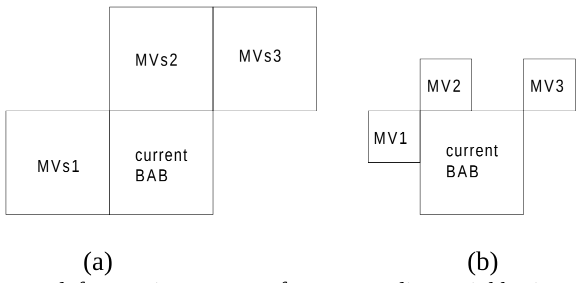


Fig. 2 MVs1~MVs3 and MV1~MV3 stand for motion vector of corresponding neighboring shape BAB (a) and texture block (b) respectively.

3. ANALYSIS

3.1 Bit Level Operation

Table 1 lists the pseudo code of BME. *VSR* and *HSR* indicate vertical and horizontal search range respectively. *C*[ ] and *P*[ ] represent pixel(s) in current BAB and reference search range respectively. Line 7 in Table 1 expresses the comparison of pixels in the corresponding positions of two BABs. Line 8 expresses the accumulation of the difference between two BABs. Since each pixel can be represented in one single bit, the comparison and accumulation of BME are bit level operations. However, the data format of RISC is usually byte (8 bits) or word (16 bits) or even larger. Directly implement these bit level operations on RISC is not efficient. By exploitng the fact that the operations in macroblock loop (line 5 ~ 9) has no dependence, we can pack continuous bits into a word so that a word loaded to RISC consists of bits each represent one pixel. These pixels are processed in parallel by one instruction. This bit parallelism technique is similar to sub-word parallelism technique used in modern DSPs/GPPs.

The "comparison" operation in direct implementation version consumes 256 exclusive-OR (XOR) operations for one search position. Applying the bit parallelism technique, *P*[*i, j, k, l*] and *C*[*i, j, k, l*] , *l=0,1,…15,* are packed to *P*[*i, j, k*] and *C*[*i, j, k*] respectively. Then the comparison for one search position consumes only 16 bit-wised XOR operations. After XOR, the number of different pixels is obtained by 256 addition operations for one searching position in direct implementation version. These operations can be replaced by 16 table look-up operations and 16 addition operations in optimized version.

Table 1: pseudo code of "direct implementation" and "optimized" BME.

Direct implementation	Optimized	Technique
1. MIN_SAD=MAXVALUE ;	1. MIN_SAD=MAXVALUE ;	
2. for(j = -VSR; j <VSR; j ++){	2. for(j = -VSR; j <VSR; j ++){	
3. for(i = -HSR; i <HSR; i ++){	3. for(i = -HSR; i <HSR; i ++){	
4. SAD(i, j) = 0;	4. SAD(i, j) = 0;	
5. for(k = 0; k <16; k ++){	5. for(k = 0; k <16; k ++){	
6. for(l = 0; l<16; l ++){	6.	
7. DIFF = P[i, j, k, l] xor C[k, l] ;	7. DIFF = P[i, j, k] xor C[k] ;	Bit parallel
8. SAD[i, j] += DIFF ;	8. SAD[i, j] += Table [DIFF] ;	Look-up table
9. }}	9. }	
10. UPDATE_MIN_SAD(i,j,SAD[i, j]) ;	10. UPDATE_MIN_SAD(i,j,SAD[i, j]) ;	
11. }}	11. }}	
// arithmetic operations :	// arithmetic operations :	
// xor: (2×VSR)×(2×HSR) ×16×16	// xor: (2×VSR)×(2×HSR)×16	
// add: (2×VSR)×(2×HSR) ×16×16	// add: (2×VSR)×(2×HSR)×16	
	// table lookup: (2×VSR)×(2×HSR)×16	

In order to support bit parallelism, pixels in each row of BAB are packed and then stored in one memory entry. However, the search position may not align the BAB boundary. As illustrated in Fig. 3, the dotted box indicates one candidate BAB in search range. The data of each row in that BAB come from two memory entries. These entries have to be read out even though only certain of bits are desired. Moreover, extra bit addressing operations such as SHIFT and PACK are required to form the desired data.

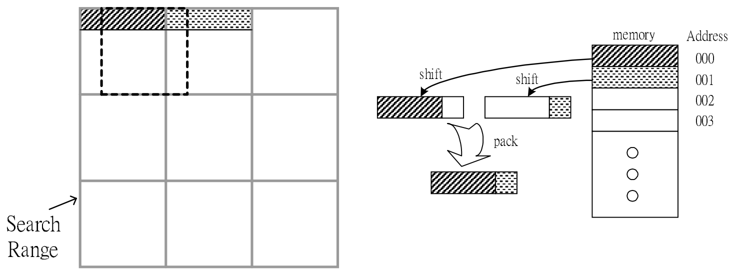


Fig. 3 The line in light gray indicates search range, and the dotted box indicates one candidate BAB. To produce one row of that candidate BAB, two memory entries have to be read out, shifted and packed.

3.2 Computation Complexity Summary

Take a typical MPEG-4 core profile level 2 application (2 VOs, CIF format, 30fps and assume 30% are boundary BAB among all) as the test condition. By the static computation model[5], it is analyzed that totally 3729.83 MOPS are required for the BME computation of real time MPEG-4 shape coding. After bit parallelism and table look-up techniques are applied, the arithmetic operations are reduced to 343.06 MOPS. However, bit addressing operations resulting from bit parallelism processing are up to 583.93 MOPS. Table 2 summarizes the computation load and memory transfer amount for BME algorithm. The run time simulation results are lists in Table 3. The experiment is taken on Ultra Sparc 300MHz, test sequence contain 1 VO only and are all 352×240. Simulation results show that it takes 350.39 seconds (worst case) to encode 100 VOPs. By bit parallelism and table look-up techniques, 10~13 times speedup can be reached. However, the optimized throughput is 3 ~ 9 fps and far from real time requirement.

Table 2: Computation load and data transfer amount for BME.

	Direct implementation	Optimization	
		Bit parallelism	Bit parallelism & table look-up
Arithmetic (MOPS)	3729.83	1978.05	343.06
Bit-addressing (MOPS)	---	583.93	583.93
Memory access (Mbyte/s)	3737.12	467.14	467.14

Table 3: Run-time for BME comparison of original and data-type optimized. (unit: sec)

Sequence	Ave. # of boundary BAB	Original	Optimized	Speedup
Children	70.79	350.39	33.99	10.3
Bream	53.93	273.67	25.73	10.6
News	57.16	145.98	10.99	13.3
Weather	37.56	146.30	11.49	12.7

4. ARCHITECTURE DESIGN

Many researches[6,7] have reported that systolic array can deal with motion estimation efficiently. In this section, the proposed 1-D systolic array based architecture for BME is described. Fig. 4 shows the block diagram of BME architecture. The PE array contains 16 processing elements, and each can calculate the SAD of one candidate BAB. Compare and select (CAS) module compares results of PE and selects the motion vector of minimal SAD. Search range buffer (SR buffer) stores partial search range data that can be reused by PE array so as to reduce data transfer from off-chip frame memory. If the MVPs is not a multiple of 16, the real search range will not align the BAB boundary, hence Shift and pack (SAP) module is required to produce the desired data. Address generation (AG) module generates address for accessing SR buffer and control signal for SAP. Since the block size of MB is 16x16, the word length of frame memory and SR buffer are 16 bits. An entry consists of 16 continuous pixels. Proper data flow is derived to maintain fully utilization of processing elements in systolic array and reduce memory access.

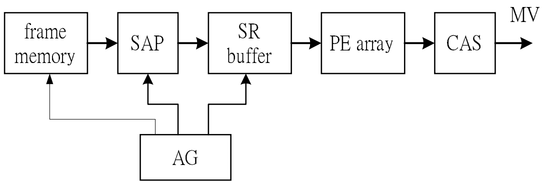
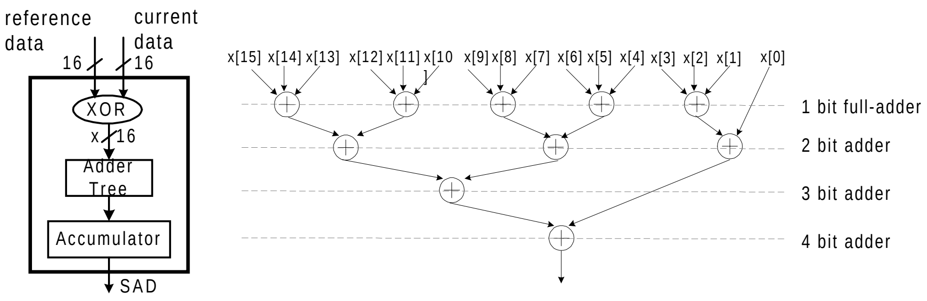


Fig. 4 Block diagram of BME.

4.1 PE Design

Fig. 5 shows the architecture of PE. The bit-wise XOR operation is performed on two rows from current BAB and candidate BAB in search range respectively. This operation will result in a row of binary data, which represents the error values between pixels of these two rows. After that, the number of different pixels should be calculated, i.e. counting the number of bit 1 in such row. Though we can use a look-up table in software implementation, it is not proper in hardware design due to the large storage element (65535×5bit ROM per PE). Optimal hardware solution to overcome this problem is the adder tree architecture. Fig. 5 (b) shows the adder tree structure that is adopted to obtain the number of different pixels between two rows. Finally, an accumulator sums up the 16 comparison results of each row. The SAD of one candidate MB is produced every 16 cycles.



(a) (b)

Fig. 5 (a) Architecture of PE. (b) Adder tree in PE

4.2 Data flow

Since PEs in PE array take responsibility for adjacent candidate search positions, the input data from search range for every PE have large redundancy. With carefully arranging the data flow, maximally data reuse utilization can be achieved. Fig. 6 shows the pixels distribution for two adjacent candidate search positions in horizontal and vertical direction. In both cases, the overlapped region of two positions contains 16×15 pixels. Undoubtedly, these overlapped pixels can be shared between PEs. When pixels of a row are packed and stored, the non-overlapped and overlapped pixels of two horizontally adjacent search positions come from the same storage entries. Hence the data read out from SR buffer for left candidate BAB can also be used for right candidate BAB. While in the case of vertically adjacent search positions, the data for lower position come from not only memory entries that overlapped pixels located but also two other entries. Comparing to the case of horizontally adjacent, two extra memory entries are required for every next adjacent position in vertical direction. Since we store and retrieve data in row basis, each PE in PE array should take responsibility for horizontally adjacent candidate position so that higher data utilization ratio of each read-out entry is achieved. Also notice that, when the number of PE in PE array equals to the number of pixels in one entry (this number is 16 in our design), every bits in the read-out entry is calculated by PE array. If we choose to pack bits of a column into the storage word, then vertically adjacent candidate MB should be processed in parallel by PE array. Table 4 list the memory access amount and execution cycles by using 16 PEs with data flow that is “identical” or “cross” to pixel-packing direction. In both data flow direction, the execution cycles are the same. This means that the data flow doesn’t affect the PE utilization ratio. For the test condition that is described in section 3.2, the memory access amount for “identical” data flow and packing direction is about 1.5 times less than “vertical” case, and 16 times less than RISC.

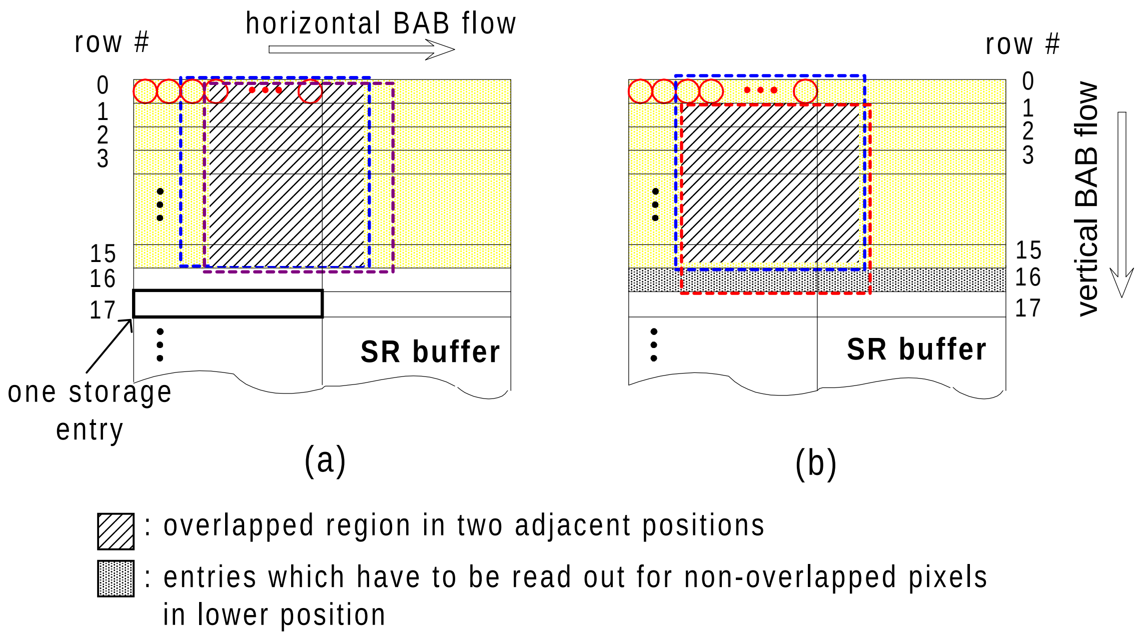

row #
horizontal BAB flow
0
1
2
3
15
16
17
SR buffer
one storage entry
(a)
row #
vertical BAB flow
0
1
2
3
15
16
17
SR buffer
(b)
▨ : overlapped region in two adjacent positions
▩ : entries which have to be read out for non-overlapped pixels in lower position

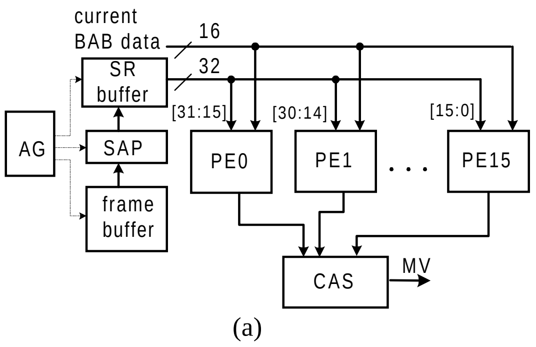
Fig. 6 Pixel distribution and location in memory entries for two adjacent searching positions in horizontal direction (a) and vertical direction (b)

Table 4: Memory access amount for different data flow direction

candidate BAB processing flow and pixels packing direction	Identical	Cross	RISC
Cycles(/s)	7.29M	7.29M	926.99M
Memory access(byte/s)	29.19M	41.71M	467.14M

4.3 Data Dispatch Based BME Architecture

Based the derived data flow, a special data dispatch technique, which is realized by hardwired routing from SR buffer to PE array, is used to efficiently reduce the bit addressing operation. Fig. 7 shows the data dispatch based BME (DDBME) architecture. The number of PEs equals to block size (16). In the beginning, 32 bits (corresponds to the first two adjacent rows in search range) are read from SR buffer (denotes as SR[31:0]). Every 16 continuous bits are dispatched to each PE in the order that bits SR[31:16] are connected to PE0, bits SR[30:15] are connected to PE1 and so on. The 16 LSBs, i.e. bits SR[15:0], are connected to PE15. At next cycle, second rows of search range are read out from SR buffer and dispatched to PE arrays in the same way. Every bit read out from SR buffer is useful and no extra SAP module is required. By this arrangement, each PE calculates one candidate BAB. As a result, all PEs will count out the SAD of horizontally adjacent candidate BABs every 16 cycles. The CAS module has 16 SAD buffers to store the sixteen SADs. The outputs of SAD buffer are sent to a comparator one by one to find the optimal motion vector. During the comparison, the PE array continuously calculates the next vertically adjacent candidate MBs. The processing flow in search range is shown in Fig. 7 (b). Candidate potions at (i,j) $0 \le i \le 15,\ 0 \le j \le 31,$ covered in region (I) and (II), are processed first, and then candidate positions at $16 \le i \le 31,\ 0 \le j \le 31,$ coverd in region (II) and (III), are proposed.



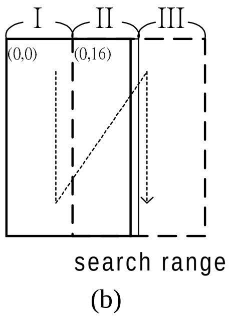


Fig. 7 (a) Architecture of data dispatch based BME (DDBME). (b) Processing flow of DDBME. Candidate positions in (I) (II) are calculated first, then candidate positions in (II) (III) are calculated.

4.4 Architectural Analysis

Some BME architectures[3,4] are proposed for traditional texture motion estimation that uses one bit for representing eight-bit-depth pixel and for block matching. In those architectures, the optimization for bit-level processing is performed only on PE, and the data flow is not taken into consideration. Fig. 8 is Natarajan's architecture[4] that is modified based on Yang's 1-D systolic array[8] for BME. It adopts vertically candidate BAB flow. Two SAPs are required for retrieving desired data from SR buffer. Because data are not reused properly, data transfer amount is huge, and half of data read out from SR buffer are actually not processed by PE. To obtain one motion vector, the data transfer from SR buffer to PE array is 4096 bytes for DDBME and 5828 bytes for Natarajan's architecture. To keep the data flow smooth, the size of SR buffer of Natarajan's architecture and DDBME is 47×32 bits and 16×32 bits respectively. In Natarajan's architecture, current BAB data are stored in 16 pipeline registers and propagated to PEs. These pipeline registers are connected in circular way. While each PE in DDBME compares the same row of current BAB concurrently, no pipeline registers is required. Current BAB is stored in a 16×16 RAM instead. They are accessed from row 0 to row 15 sequentially and repetitively until all candidates in search range are compared. Table 5 lists the detail comparison between DDBME and Natarajan's architecture. It reveals that although we can implement BME by modified existed texture ME architecture, without optimized data flow the redundant bit is read and extra hardware cost is paid for bit retrieval.

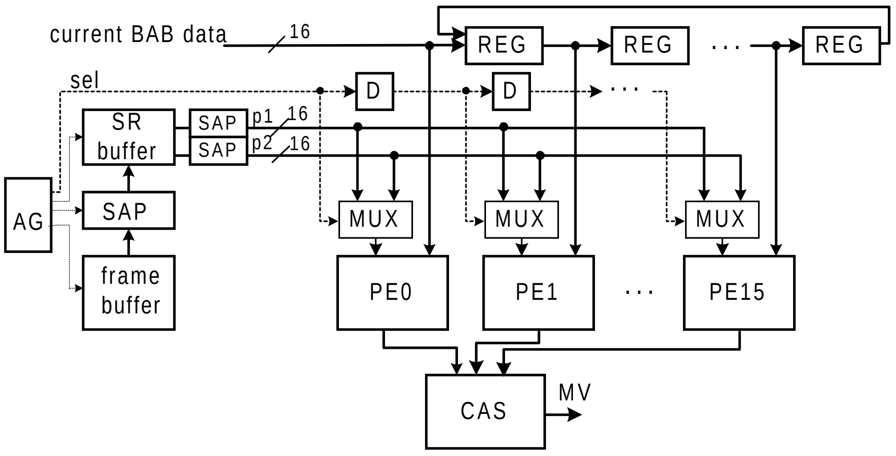


Fig. 8 Natarajan's architecture[4].

Table 5: Architectural analysis and comparison for DDBME and Natarajan's architecture. Search range is [–16, 15]

	DDBME	Natarajan's architecture[4]
Current BAB source	16×(16 bits) RAM	16×(16 bits) registers (in circular link)
Access from SR buffer to obtain one MV (bytes)	16×(2×VSR)×2×4 = 4096	(3×VSR-1)×2×2 + (3×VSR-1)×4×(2×HSR-2) = 5828
Latency of obtaining one MV (cycles)	32×16×2+15 = 1039	32×32+15 = 1039
SR buffer size	16×32 bits	47×32 bits
Delay registers	0	1×15
SAP	32 bit barrel shift×1	32 bit barrel shift×3
AG	Regular	Regular

4.5 IMPLEMENTATION RESULTS

The DDBME module is implemented and verified at gate level. It was synthesized with 0.35um CMOS standard-cell library using Synopsys. Table 6 lists gate count of each module in BME.

Table 6: Implementation gate count of DDBME sub-modules.

Module	Gate count
PE_Array	4668
SAP	1560
CAS	2763
AG	389
Total	9666

5. CONCLUSION

In this paper, we present an efficient binary motion estimation architecture design for MPEG-4 shape coding. First we carefully examine the computational behavior of BME operation. We realize that a large amount of bit-level processing is demanded for BME operations. Thus, bit parallelism technique is applied for SAD calculation in block matching algorithm in order to improve the speed of bit-level processing. However, the bit parallelism leads to the heavy bit-addressing operations. To remove the bit addressing operation, a data dispatch based 1-D systolic array and a special data flow is designed. The proposed architecture is verified at gate level simulation. Total gate counts of BME are 9666. The proposed architecture operating at 7.29 Mhz can handle real time MPEG-4 video application at core profile level 2.

6. REFERENCE

1. ISO/IEC JTC1/SC29/WG11, N2502a, Generic Coding of Audio-Visual Objects: Visual 14496-2, Final Draft IS, Atlantic City, Dec. 1998.
2. J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding", *IEEE Trans. Commun.* Vol. COM-29 pp. 1799-1808, Dec. 1981.
3. Y. K. Ko, H. G. Kim, J. W. Lee, Y. R. Kim, H. C. Oh and S. J. Ko, "New Motion Estimation Algorithm Based on Bit-Plane Matching and Its VLSI Implementation", Proceedings of the IEEE Region 10 Conference, Vol. 2, pp. 848-851, 1999
4. B. Natarajan, V. Bhaskaran and K. Konstantinides, "Low-Complexity Block-Based Motion Estimation via One-Bit Transforms", *IEEE Trans. on Circuits and Systems for Video Tech.*, Vol.7, No.4, August 1997.

5. H. C. Chang, L. G. Chen, M. Y. Hsu and Y. C. Chang, “Performance Analysis and Architecture Evaluation of MPEG-4 Video Codec System”, in Proc. of *International Symposium on Circuits and Systems (ISCAS'2000)*, Geneva, Swiss, May 2000.
6. Thomas Komarek and Peter Pirch, “Array Architectures for Block Matching Algorithms”, *IEEE Trans. on Circuit and Systems.* vol. 36 No. 10, pp. 1301-1308, Oct. 1989.
7. C. H. Hsieh and T. P. Lin, “VLSI Architecture for Block-Matching Motion Estimation Algorithm”, *IEEE Trans. on Circuit and Systems for Video Tech.*, vol. 2 No. 2, pp. 169-175, Jun. 1992.
8. K.-M Yang, M.-T Sun, and L. Wu, “A Family of VLSI Design for the Motion Compensation Block-Matching Algorithm”, *IEEE Trans. on Circuits and Systems for Video Tech.*, Vol.36, No.10, October 1989.