An Efficient Binary Motion Estimation Algorithm and its Architecture for MPEG-4 Shape Encoding

Esam A. Al_Qaralleh, Tian-Sheuan Chang, *Member, IEEE*, and Kun-Bin Lee, *Member, IEEE*

***Abstract*—This paper presents a fast binary motion estimation (BME) algorithm and its architecture for MPEG-4 shape encoding. The proposed algorithm explores the property of the binary-value in BME to quickly skip the unnecessary sum of absolute differences (SAD) computation. When comparing with the full search algorithm, simulation results show that it can efficiently save in the search positions to an average −99.58% of that in the full search algorithm with the same PSNR quality. Due to the algorithm's simplicity and regularity, the resulting hardware implementation also exhibits simple and regular control and data flow. It can achieve real-time encoding with only 11582 gate count.**

***Index Terms*—Binary motion estimation(BME), MPEG-4, shape coding, video object plane (VOP).**

I. Introduction

MPEG-4 is an object-based video standard that allows the transmission of arbitrarily shaped video objects [1]. The purpose of using shape is to achieve better subjective picture quality, increased coding efficiency as well as the possibilities for user interaction. In the MPEG-4 shape coding, the binary motion estimation (BME) has been adopted [1] to exploit the temporal redundancies inherent within image frames, and thus gain more compression ratio. However, due to its high computation complexity and huge memory bandwidth, it has been shown that BME occupies about 91% of computational complexity in MPEG-4 shape encoder, up to 4GOPS [2], [3] and far from real-time requirement [2]. Therefore, optimization on BME is essential to remove the bottleneck to achieve real-time shape encoding. Hence, various fast algorithms and hardware design were proposed [2]–[6] to reduce the computational complexity.

The fast algorithm approaches use various skipping techniques to speed up the BME. In [3] and [4], they skipped those search positions which are depart from the contour line of the object. In [5] it skipped computations of boundary alpha blocks (BABs) by testing if the motion compensation error for that BAB is less than a predefined threshold value. In [6], it generated a mask for the points close enough to the object boundary, and limited the search process only to those points. These fast algorithms are usually implemented by software. However, software implementation of BME on processors is not efficient since processors are more efficient at the word level instead of the bit-level operation as in BME. For hardware implementation, the data in BME is just a one-bit binary value (0/1), which can be easily represented by hardware, and achieve computation speedup by bit parallelism. The hardware design in [2] presented BME architecture by employing bit parallelism technique using 1-D systolic array to perform a full search BME. In all the above approaches, none have explored the property of binary-value in the BME for the algorithm and architecture design.

TABLE I
16-Classes Classification, Showing the Number of "1" in Each Class, and the Range of "1" in Every BAB That Matches With Each Class

classes number	# of "1" included in each class	# of "1" included in the matched BAB
class 1	16	1~16
class 2	32	17~32
…	…	…
class 15	240	225~240
class 16	256	241~255

In this paper, we propose a fast BME algorithm and its hardware design that significantly reduces the number of search positions for the block matching. The proposed algorithm explores the binary value property in BME to efficiently skip the highly unlikely search positions. The motivation of our approach is that, BME only deals with binary values (1 or 0) instead of the 8-bit pixel values in the texture ME. Hence, the block matching of BME can be regarded as a comparison of number of "1" contained in each candidate block with that of the current block. Thus, the proposed algorithm classifies each candidate block according to the number of "1" it contains, and only performs the block matching between those blocks belonging to the same class. Furthermore, we also present a hardware design for the proposed algorithm. Hardware design can make the binary bit-level processing much easier and faster than the software approach since traditional processor only deals with word-level processing. The simplicity and regularity of the proposed algorithm leads to a simple and regular hardware design.

This paper is organized as following. In Section II, an introduction to BME will be presented. Section III will describe the proposed algorithm. In Section IV, we will present the simulation results of the proposed algorithm. Then, the architecture for the proposed algorithm will be shown in Section V. Finally conclusions will be made in Section VI.

II. Introduction to BME

BME can remove temporal redundancy by searching in the reference video object plane (VOP) for a candidate BAB that

Manuscript received May 11, 2005; revised November 1, 2005. This work is supported by the National Science Council, Taiwan , R.O.C., under Grant NSC-93-2200-E-009-028. This paper was recommended by Associate Editor K. Aizawa.

The authors are with the Electronics Department, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C. (e-mail: esam@twins.ee.nctu.edu.tw; tschang@twins.ee.nctu.edu.tw; kblee@twins.ee.nctu.edu.tw).

Digital Object Identifier 10.1109/TCSVT.2006.878148

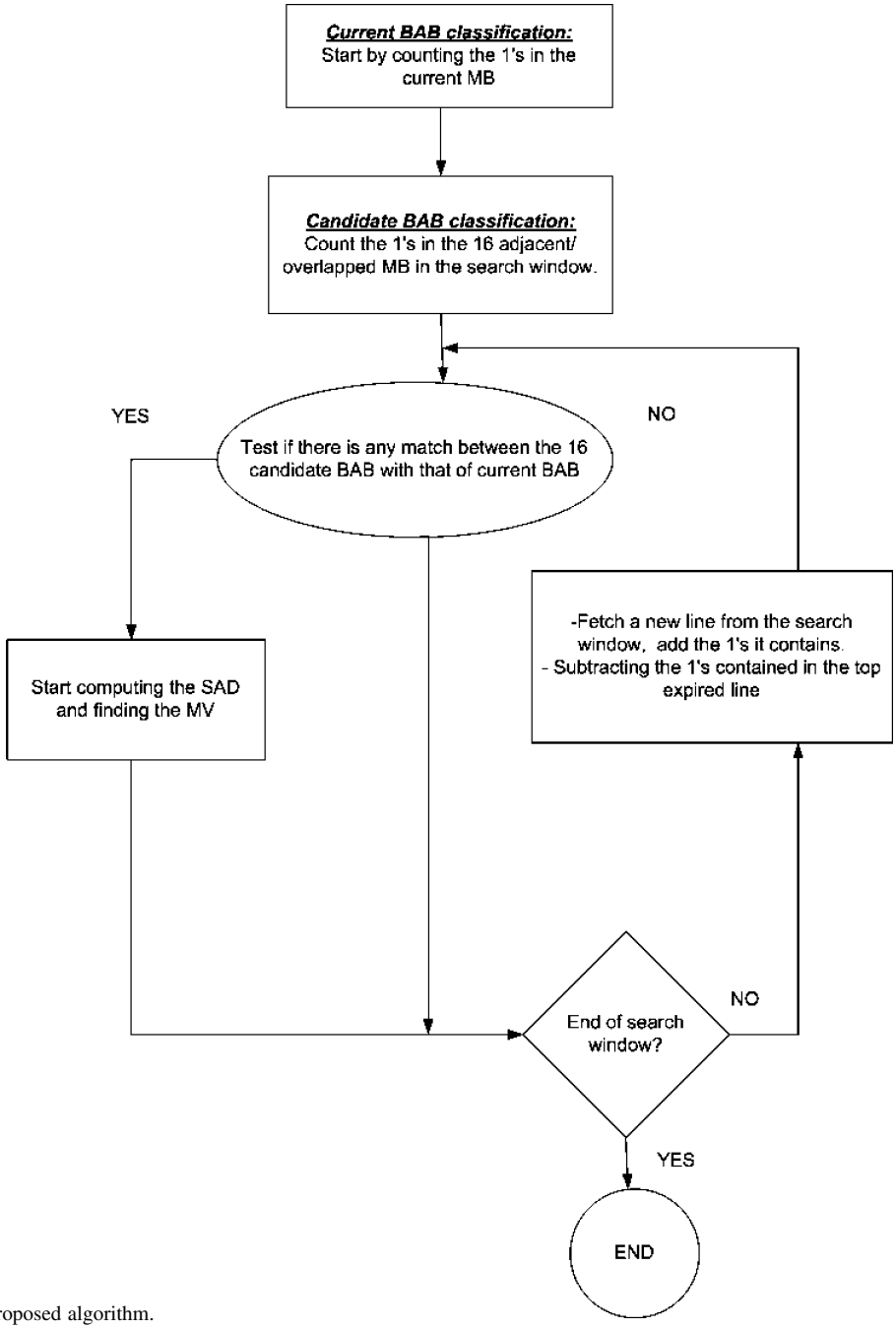


Fig. 1. Flowchart for the proposed algorithm.

is most similar to current BAB. Based on the assumption that the movement of an object is homogeneous, motion vector of neighbor BAB or texture block is used as the motion vector predictor for shape (MVPs). The block matching is performed around the MVPs to compute the sum of absolute differences (SAD) by comparing the BAB indicated by the motion vector and the current BAB. The SAD between the current BAB located at (x, y) in the current VOP I_c and a reference BAB located at a displacement of (v_x, v_y) relative to current BAB in the reference VOP I_r is given as

$$\mathrm{SAD}(v_x, v_y) = \sum_{m=0}^{15}\sum_{n=0}^{15} |I_c(x+m, y+m) - I_r(x+v_x+m, y+v_y+n)|. \quad (1)$$

The motion vector that minimizes the SAD is taken as MVS and this is further interpreted as motion vector difference for shape (MVDs), i.e., $\mathrm{MVDs} = \mathrm{MVS} - \mathrm{MVPs}$.

III. Proposed Algorithm

Fig. 1 shows the proposed algorithm flowchart. First, we classify the current BAB according to the number of "1" it contains. Then, for each search position in the search window, we also classify the candidate BAB using the same method (counting "1" contained in the BAB). If both the current BAB and the candidate BAB are in the same class (called a match), we start computing SAD for that position. Otherwise, we skip that position and start testing the next search position. The main concept behind this algorithm is that BME only deals with binary values.

TABLE II
PERFORMANCE OF THE PROPOSED ALGORITHM AND THE FULL SEARCH METHOD WHEN SEARCH WINDOW IS ±16 AND 255 CLASSES WITHOUT OVERLAPPING

	Bits/shape			# of MB processed (search positions)		
test sequences	FS	Proposed	CHG_BIT	FS	Proposed	CHG_SP
Foreman	311720	328329	5.33	7810280	25918	-99.67
Stefan	236964	248268	4.77	6141676	21812	-99.64
singer-247	239246	275191	15.02	6125008	20941	-99.66
News	298696	336933	12.80	5863564	16956	-99.71
dancer-247	440828	481913	9.32	9623385	36769	-99.62
coastguard	176666	188283	6.58	2062352	11885	-99.42
coastguard_obj_0	401875	433491	7.87	6066320	41951	-99.31
coastguard_obj_1	256330	283021	10.41	4134816	19069	-99.54
coastguard_obj_2	119925	122755	2.36	2036772	11714	-99.42
coastguard_obj_3	171030	181109	5.89	1949582	11382	-99.42
Total	2653280	2879293	8.52	51813755	218397	-99.58

TABLE III
PERFORMANCE OF THE PROPOSED ALGORITHM AND THE FULL SEARCH METHOD WHEN SEARCH WINDOW IS ±16 AND 255 CLASSES WITH 6 CLASSES OVERLAPPING

	Bits/shape			# of MB processed (search positions)		
test sequences	FS	Proposed	CHG_BIT	FS	Proposed	CHG_SP
Foreman	311720	314979	1.05	7810280	272905	-96.51
Stefan	236964	238255	0.54	6141676	259932	-95.77
singer-247	239246	239865	0.26	6125008	250416	-95.91
News	298696	298912	0.07	5863564	185674	-96.83
dancer-247	440828	444623	0.86	9623385	431812	-95.51
coastguard	176666	177752	0.61	2062352	94341	-95.43
coastguard_obj_0	401875	405486	0.90	6066320	364255	-94.00
coastguard_obj_1	256330	258102	0.69	4134816	207529	-94.98
coastguard_obj_2	119925	120353	0.36	2036772	80475	-96.05
coastguard_obj_3	171030	172147	0.65	1949582	92980	-95.23
Total	2653280	2670474	0.65	51813755	2240319	-95.68

Thus, we can use the number of "1" contained by the BAB to approximate the BAB's data and classify it into different classes. Hence, we can quickly skip SAD computation between different classes and only compute the SAD for the same class.

Table I represents an example for classifying each BAB according to the number of "1" it contains, i.e., if a BAB contains 20 pixels marked as "1", it will be classified as class 2. Fig. 2(a) shows a BAB, where the shadowed pixels represent "1" in the BAB. This BAB is classified as class 3 according to the classification in Table I (contains 34 pixels representing "1"). Fig. 2(b) represents the search window that shows two BABs: one with almost the same number of "1", classified as class 3 (contains 35 pixels representing "1"), while the other classified as class 12 (contains 189 pixels representing "1"). It is clear that the BAB with almost the same number of "1" is more likely to be a match to the current BAB rather than the other one with larger number of "1".

Classification and matching rule will severely affect the quality of searching results. The matching rule can be generalized from the same-class matching to the adjacent-classes matching. Thus, a matching could be hold for those belonging to the same class or adjacent classes. This feature (overlapping between one or more adjacent classes) gives us the ability to refine the MVS to be more accurate, which will be presented later.

IV. SOFTWARE SIMULATION RESULTS AND ANALYSIS

In the following two subsections, we will show the efficiency of the proposed algorithm by integrating it into the MPEG-4 verification model V18.0 [7]. All the following test sequences are in CIF format with 300 VOP and one Video Object (VO). Then we will show how to explore the flexibility of our algorithm (the classification and the matching rule) to control both the search positions and the bit rate to get more refined MVS.

A. Simulation Results

Tables II and III summarize the results compared with the full search algorithm, where CHG_BIT denotes the change of bits in percentage, and CHG_SP denotes the change of search position. Table II assumes ±16 search window with nonoverlapping 255 classes (every two adjacent classes differ only by one pixel value). Due to the strict nonoverlapping class partitioning, the search positions saving (CHG_SP) is −99.58% (the negative sign indicates saving, in other words the percentage of

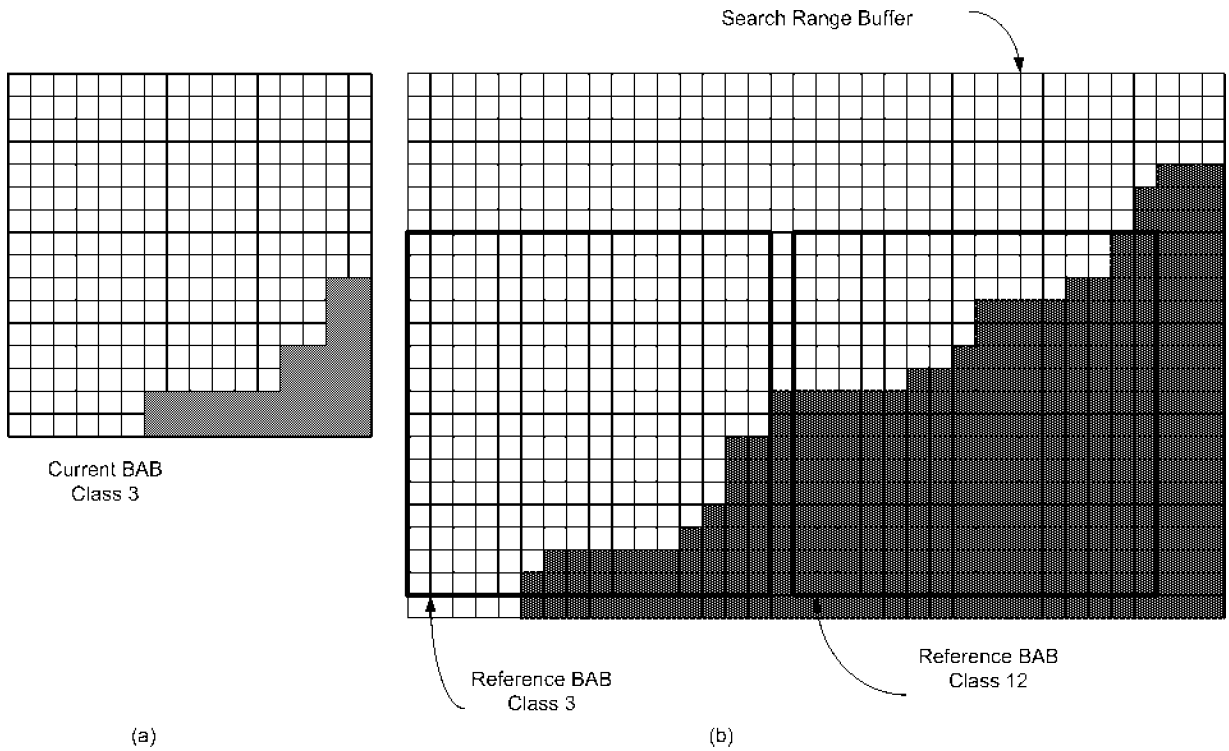


Fig. 2. (a) Current BAB contains certain number of "1" bits and belongs to class 3. (b) Search window showing two BABs, each one from different classes. The SAD operation will hold only for the left BAB in the search window, since it is more likely to be a match to the Current BAB, and hence belongs to the same class.

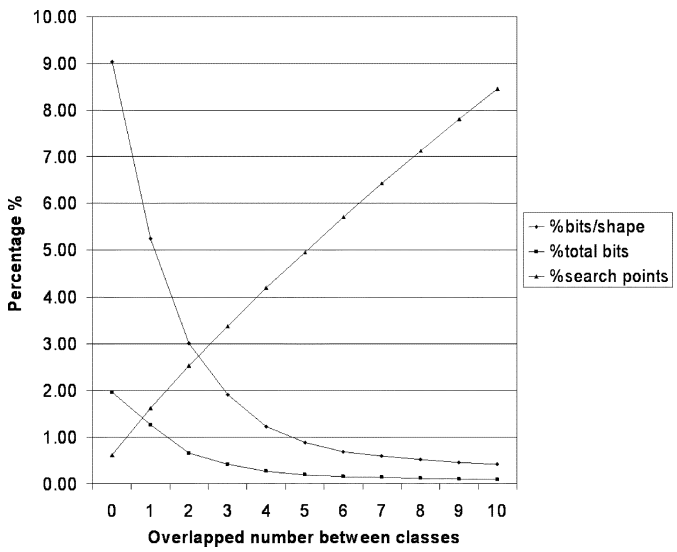


Fig. 3. Impact of classes overlapping on bits/shape, total encoded bit-stream, and the number of search positions.

search positions to that of the full search is 0.42%) of that in the full search algorithm. Such reduction comes with the cost of average 8.52% increase in the encoded shape bit rate (bits/shape).

Table III shows the effect of overlapping classes with the same ± 16 search window. With class overlapping, the increase in the bits/shape is significantly reduced to 0.65%, but it also reduces the CHG_SP to -95.68% compared with the nonoverlapping case. This is because class overlapping will enable more class matching for BABs with slight difference in number of "1". Thus, the increase in the encoded bit stream will be smaller than the nonoverlapping case at the cost of more search positions. Without class overlapping, we may skip possible search positions due to small difference. Fig. 3 shows the effect of overlapping on bit rate and search positions, it shows that, the extra bit rate increase reduced significantly with just two or three pixel overlapping. On the other hand, the required search positions are linearly increased as the number of classes overlapped is increased.

Table IV shows the effect of class partitioning. As the number of classes decreases (that is, count number of "1" in each class increases), the increase in encoded bit stream will be lower with the cost of increased search positions. This is because more BABs in the search window will be classified to be a match. This increases the required search positions but also reduces the bit rate due to more accurate motion matching, as shown in Fig. 3.

B. Consideration for the Classification and Matching Methods

The optimum classification of classes and class-overlapping are highly content dependent. For some test sequences, the probability of classes are not uniformly distributed (BABs belonging to a certain class are more probable than others). Fig. 4 shows the probability distribution for the 255-classes of the test sequence "container_2_obj". It is clear that, BABs in which the number of "1" in the range 1–83 are more probable, among which the range 57–81 has higher occurrence. Thus, we can divide the intervals for each class according to the probability density such that, 1–56 to 8-classes (each class differs by 7 bits of "1" from the neighbor classes), 57–81 to 25-classes (each class differs by one bit "1" from the neighbor classes), and the remaining into 4-classes. We ran three tests for the same test sequence,

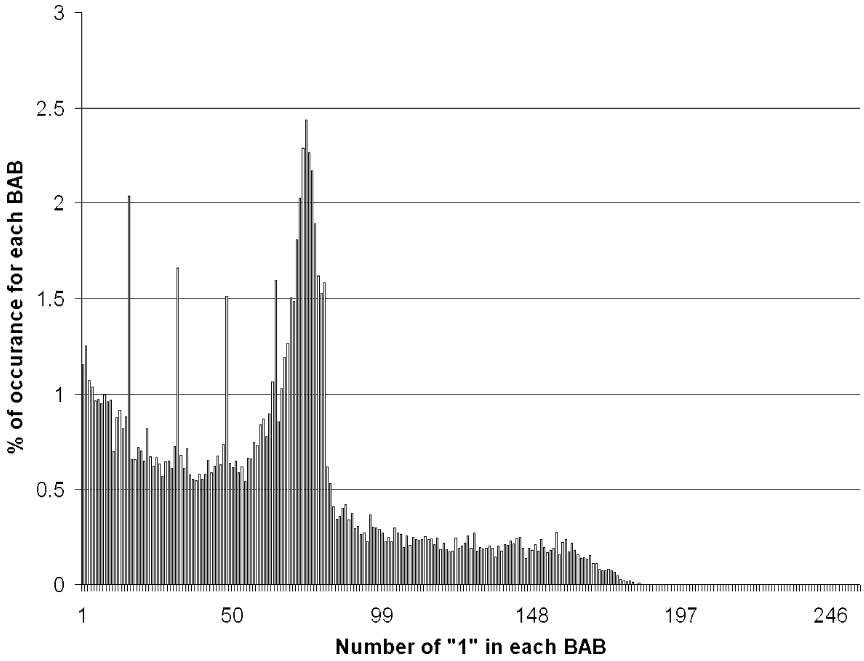


Fig. 4. Probability distribution for the 256 classes in the container_2_obj.

TABLE IV
COMPARISON BETWEEN DIFFERENT CLASSES

Class type	Bits in each class	CHG_BIT for shape	CHG_BIT for total bits	CHG_SP
16 class	16	1.17	0.24	-93.18
32 class	8	2.52	0.53	-96.25
64 class	4	4.57	0.99	-98.04
256 class	1	9.03	1.96	-99.38

TABLE V
RESULTS FOR DIVIDING CLASSES ACCORDING TO THE PROBABILITY OF CONTAINER_2 TEST SEQUENCE

test	CHG_BIT for shape	CHG_BIT for total bits	CHG_SP
case-1	16.45	5.31	-95.4
case-2	1.8	0.63	-81.7
case-3	3.14	1.09	-90.3

as shown in Table V. For the case-1 without overlapping between classes, we got a smaller number of search positions but larger bits/shape. In the case-2, we ran the same test with uniform overlapping between classes, and as expected this resulted in a higher number of search positions with fewer bits/shape. The case-3 compromises between reduction in search positions and bits/shape by overlapping only in the range of high probability, in the range of 57–81. We got lower search positions than the case-2, and lower bits/shape compared to case-1. By applying overlapping to those classes with more probable ones will refine the MVS with a little increase in search positions.

Statistics in Fig. 4 can be calculated at the run time, by accumulating the occurrence of every class, overlapping those of high probability, and joining more than one class for those with less probability. The statistics can be made according to a "frame window," such that, for a predefined number of frames (e.g., ten frames window) we count the statistics and consider the results for the coming frames. This will be explored in a future work by dynamic class assignment and overlapping.

V. ARCHITECTURE DESIGN

A. Architecture Design

Fig. 5 shows the block diagram of BME architecture. Due to the simplicity and regularity of the proposed fast algorithm, the whole architecture is similar to the full search architecture presented in [6]. However, instead of full search, we adopt the fast algorithm but maintain the regularity of full search. The extra hardware needed is the modification to the accumulator to support addition and subtraction, and also extra registers to save the accumulated count for "1" in every BAB. The addressing and control unit is simple due to regular data flow. In Fig. 5, the search window buffer (SR buffer) stores partial search window data that can be reused by PE array to reduce data transfer from off-chip frame memory. The PE array contains 16 processing elements, and each can compute the SAD of one candidate BAB. Another function of the PE is to count the "1" within every candidate BAB. Thus, a MUX (Multiplexer) will be used to select the operation for the PE between counting "1" and computing the SAD. A compare-and-select (CAS) module compares results of PE and selects the motion vector of minimal SAD. Control/Address generation (AG) module generates address for accessing SR buffer and control signals to other modules. Registers are used to store the count of "1" for each BAB in the SR, in which each register is 8-bit (enough to count up to 255).

B. PE Design

Fig. 6 shows the architecture of a single PE, it consists of an XOR circuit followed by an adder tree, and ended with an accumulator. The accumulator supports both addition and subtraction. For SAD computation, one row of current BAB and one row of candidate BAB are compared by bit-wise XOR. The

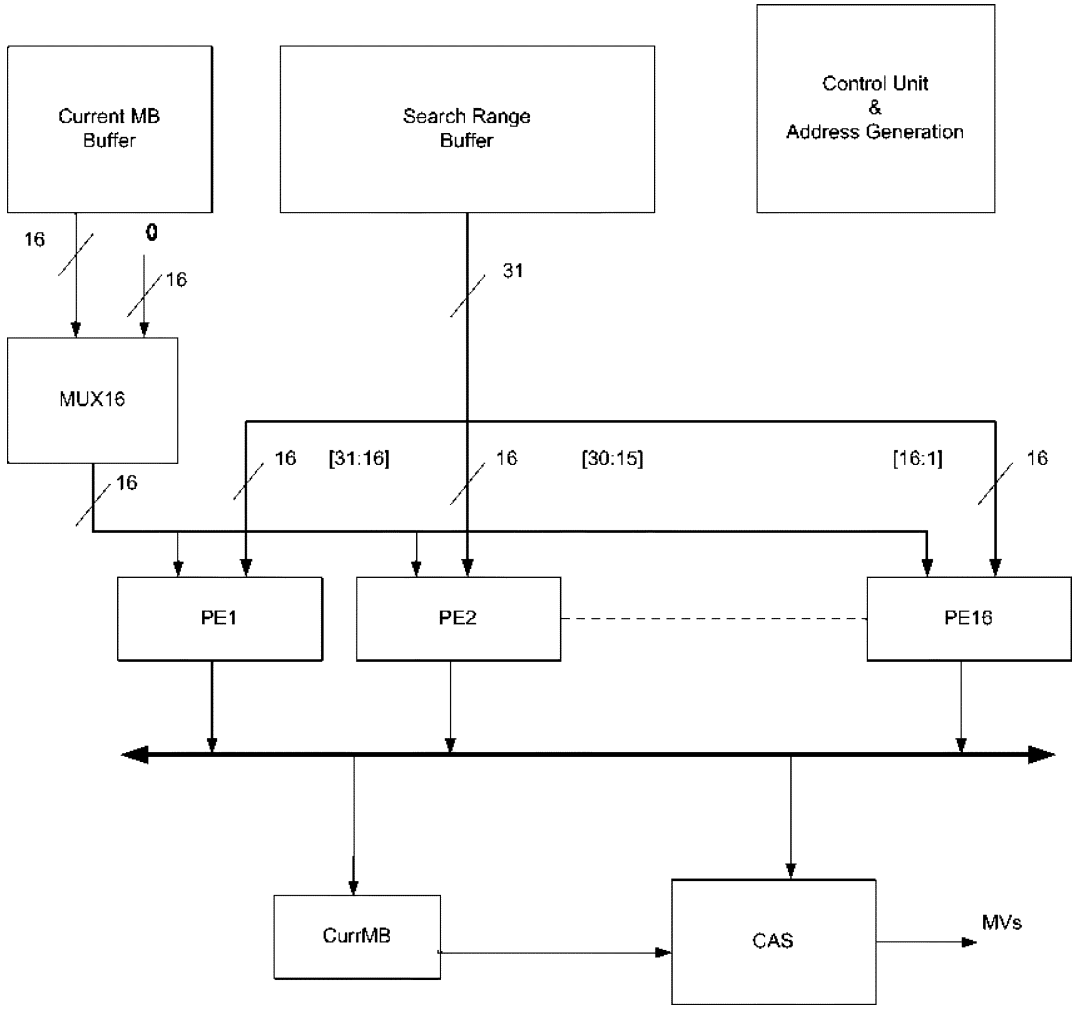


Fig. 5. Architecture block diagram of the proposed algorithm.

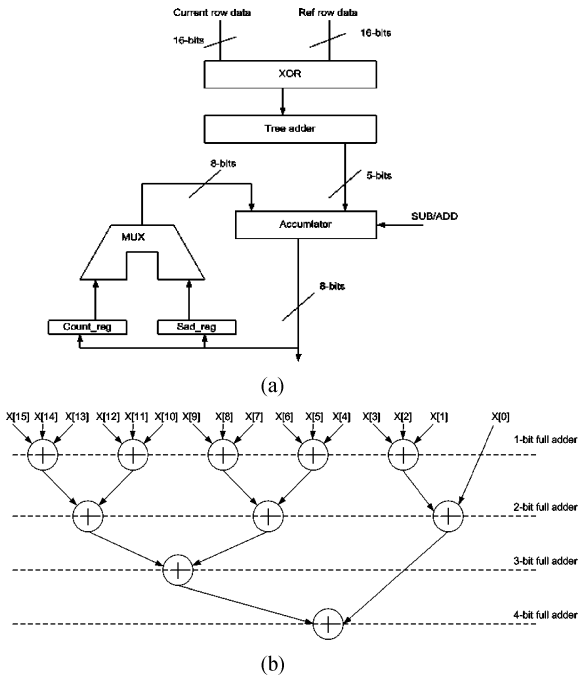


Fig. 6. (a) The SAD architecture, and (b) the tree adder.

resulted row of binary data represents the difference values between pixels of these two rows. The adder tree will sum up those binary data as partial SAD. The accumulator sums up 16 rows of partial SAD to obtain the SAD of one candidate position in SR (the SAD of one candidate BAB is produced every 16 cycles). In each PE there are two registers, one to save the partial SAD for later use as final SAD for that search position (sad_reg). The other register will hold the count of "1" pixels of candidate BAB (count_reg).

C. Data Reuse and Data Flow

Data reuse concept should be explored while reading from the search window buffer. Data redundancy exists in both directions, horizontally and vertically. The horizontal data redundancy could be due to computing SAD for more than one adjacent candidate BABs in the search window buffer. The vertical data redundancy could be due to counting "1" of adjacent candidate BABs. Since PEs in the PE array take responsibility for adjacent candidate BABs, the input data from search window for every PE have large redundancy. To reduce the data redundancy, we use the sliding widow scanning along with data dispatching to obtain and distribute the data. With data dispatching, we can achieve better data reuse utilization. As shown in Fig. 5,

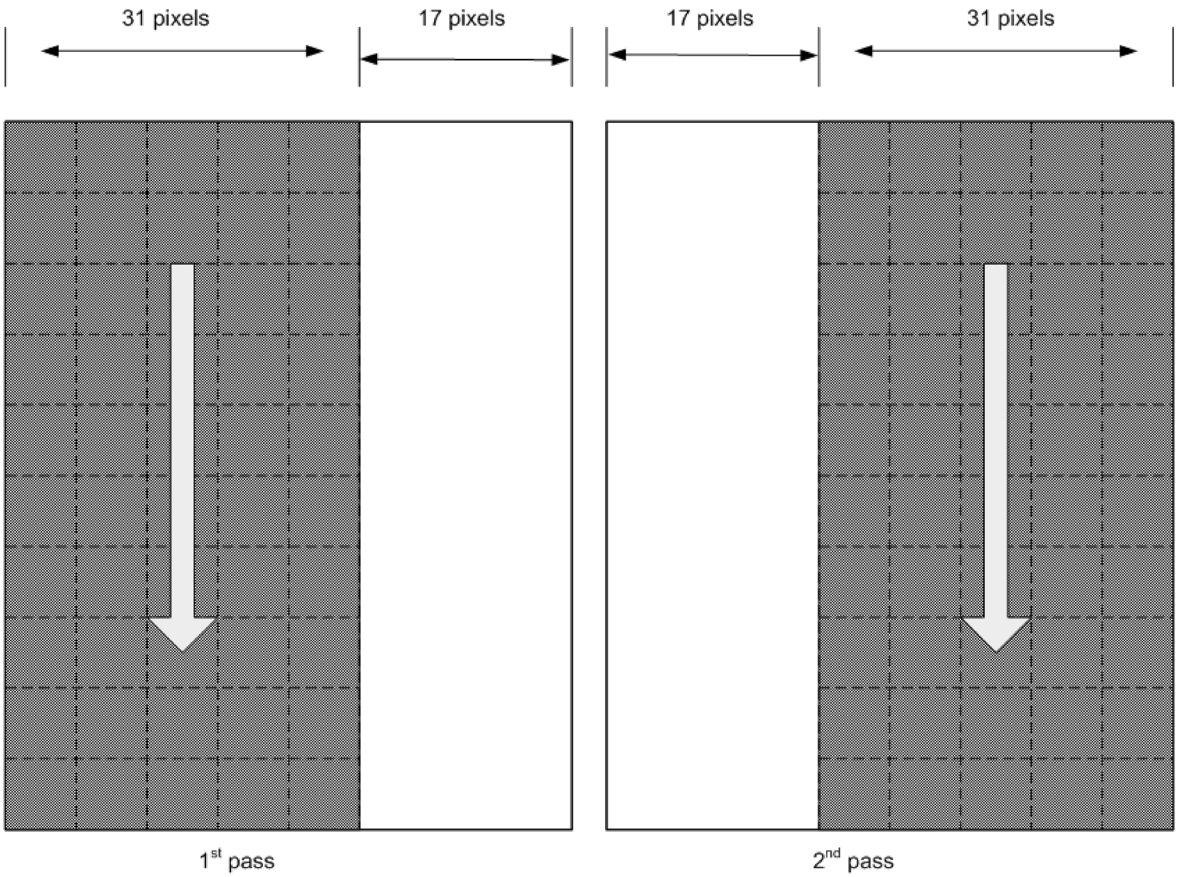


Fig. 7. Two passes needed to cover the search window.

data dispatch is implemented by hardwiring the desired reference data into each PE. Data [31:16] are dispatched to PE1, and data [30:15] are dispatched to PE2 and so on. The search window width is 48 pixels [for search range $(-16, +15)$], while the data fed into 16 PE are 31 bits, which leaves 17 bits to be scanned. Thus, only two passes will be needed to cover the entire search window, as shown in Fig. 7. Each pass will cover part of the search window, supplying the 16 PE with proper data to do the match. To facilitate the counting of "1" and SAD computation, the proposed design adopts the sliding window approach to read the pixel from the search window by sliding vertically, as shown in Fig. 8. Sliding down in the search window will keep tracking of the "1" counting for every adjacent candidate BAB by adding a new row, and subtracting the top expired row. As we slide down by one row, we still make use of the remaining 15 rows (the area marked by crossed bars). This will just add an overhead of 2-clock cycles to keep tracking of the "1" count for every BAB (2-clock cycles rather than 16-clock cycles to perform full SAD computation). We can summarize the procedure of the architecture operation as follows.

1) *Current BAB classification*: Count the number of "1" in the current BAB, and store the result into "CurrMB" register (this step needs16-clock cycles).
2) *Candidate BAB classification*: Start counting the number of "1" for 16 adjacent BABs within the search window, and store the results into each specified register ($\mathrm{Reg1} \sim \mathrm{Reg16}$). Each register located inside the PE, as shown in Fig. 6(a) (this step needs 16 clock cycles).
3) *Class match and SAD computation*: The comparison circuit will classify the results stored in the registers and determine which one matches the current BAB class. If a match occurs, start calculating the SAD for that position only (16-clock cycles when a match occurs to compute the SAD).
4) *Proceeding to new data*: If there is no match, we proceed to the next row by the sliding window approach (2- clock cycles overhead to count the number of "1" in this way).
5) Repeat steps (4) and (3) to the end of the search window.

D. Experiments Results

Since the architecture consists of 16-PEs working simultaneously, it is highly probable that more than one match could occur (two or more adjacent BAB belong to the same class, as shown in Fig. 9), and hence performs the SAD computation for more than one match at the same time. This will save the processing time since more than one match to be processed in one time slot needed to do one match. The whole design has been implemented in Verilog code and synthesized by Synopsis Design Compiler. The synthesized gate count for the architecture is 11582 for the total design, using 0.18-μm cell library.

The required cycle count is quite low due to our simple scheme to skip unlikely search positions. The cycle count to perform one full search can be calculated as follows; as an initial step we need to count "1" for the current BAB (16 clock cycles) and for the first 16 rows in the search window buffer (16 clock cycles). Then proceeding by adding new row and

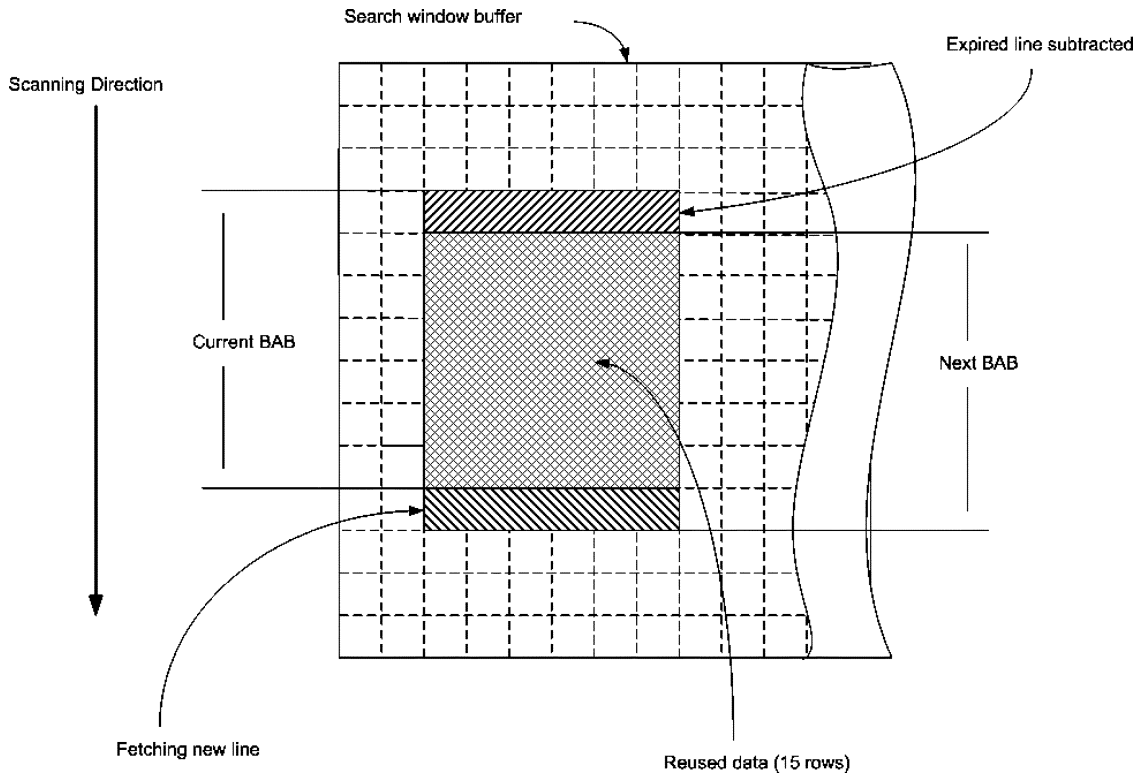


Fig. 8. Sliding window approach: counting the "1" in BABs.

subtracting the top expired row (Fig. 8), this will consume two clock cycles. Since we can scan the search window in two passes, so

$$(2\ \text{clocks}) \times (2\ \text{passes}) \times (32\ \text{rows for each pass}) = 128\ \text{clock cycles}$$

are needed to scan the search window. When a match occurs, 16 clock cycles are needed to compute SAD for that match (i.e., we express the number of matches or search positions by #SP). The total clock cycles will be $(16+16+128+(\#\,\mathrm{SP})\times 16)$ cycles. These cycles are needed to find one BAB MV.

Table VI gives the average clock cycles consumed to scan one search window for different classes overlapping. From Table VI, the average clock cycles to complete one frame in case of 32 classes overlapping (worst case) is 563. From which we can calculate the total clock cycles to complete one frame. Assuming the percentage of BABs to be 50% of the total alpha blocks (e.g., for CIF 352×288, the BABs will be 198), we need $563 \times 198 = 111{,}474$ clock cycle to complete one frame. From the above calculations, the overhead of our algorithm will be as follows: 16 clock cycles to count the "1" of the current BAB, another 16 clock cycles to count "1" for the first search position in the search window, the latter will be repeated twice, since we scan the search window twice, 2 clock cycles for every search position to add and subtract one row of pixels. Theoretically, the clock cycles needed to perform one full search window using one PE would be $(31 \times 31 \times 16 = 15376)$, and for 16-PE would be $(2 \times 31 \times 16 = 992)$ clock cycles, while the worst case in our design is 563 clock cycles.

TABLE VI
HARDWARE SIMULATION RESULTS FOR DIFFERENT CLASSES OVERLAPPING

	32 classes overlapping	16 classes overlapping	no overlapping
test sequences	AVE clk *	AVE clk *	AVE clk *
foreman	506	443	276
Stefan	635	536	302
singer-247	626	497	258
news	543	474	269
dancer-247	595	506	276
coastguard_obj_0	410	354	250
coastguard_obj_1	555	483	289
container_obj_0	540	452	284
container_obj_1	553	503	331
container_obj_2	662	578	302
container_obj_4	570	475	245
Average	563	482	282

* Average clock cycles needed to finish one search window.

E. Comparison

Tables VII and VIII show the comparison results between our proposed algorithm, and other fast algorithms. The proposed algorithm can achieve lower search positions and still has lower bit rate increase. Moreover, the flexibility of our proposed algorithm which lies in the classification of classes, and overlapping between classes, according to run time statistics, will benefit in tradeoff between reduction in search points and bits/shape. The comparisons are based on bits/shape and reduction in search positions. Bits/shape presented by [4] is based on WSAD (weighted SAD) which gives lower bit rate and different

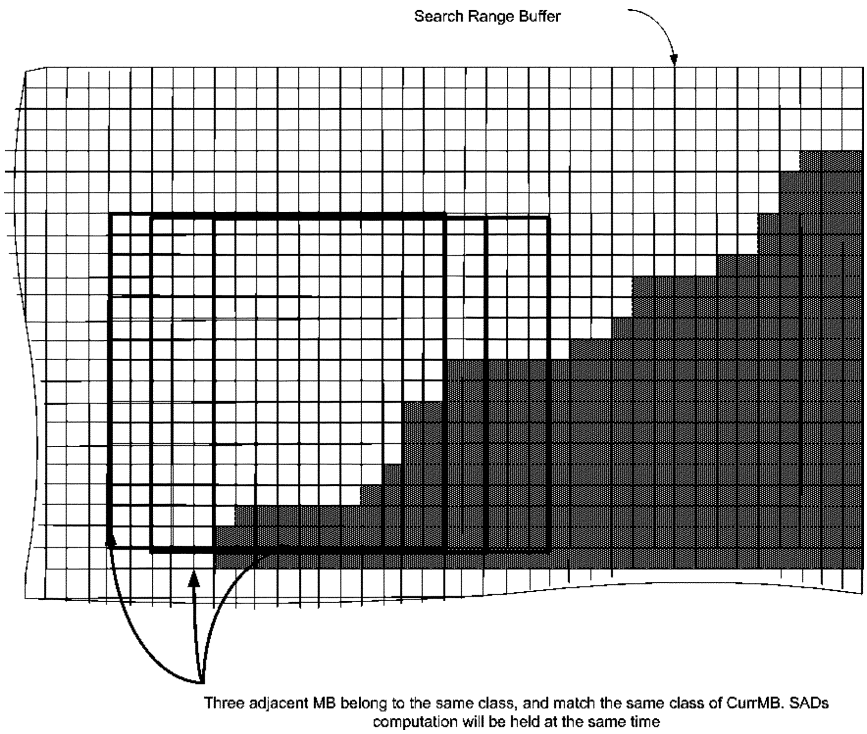


Fig. 9. Adjacent BABs belong to the same class, and match the same class of CurrMB. SAD calculation will be held at the same time in hardware implementation.

TABLE VII
CHG_SP FOR VARIOUS SEARCH ALGORITHMS RELATIVE TO THE FULL SEARCH ALGORITHM

Sequence	Ref[6]	Ref. [5]	Ref. [4]	proposed	
				±16 with 6 overlapping	±16 without overlapping
news	- 46.04%	- 99.12%	- 96.74%	-96.69%	-99.71%
foreman	- 43.94%	- 82.78%	- 96.85%	-96.49%	-99.67%

TABLE VIII
AVERAGE BIT-RATE FOR VARIOUS SEARCH ALGORITHMS. ALL ARE RELATIVE TO THE FULL SEARCH ALGORITHM

Sequence	Ref[6]	Ref. [5]	Ref. [4]*	proposed	
				±16 with 6 overlapping	±16 without overlapping
news	100.00%	99.26%	100.19%	100.07%	112.8%
foreman	100.00%	100.47%	99.65%	101.05%	105.33%

* Weighted SAD

values than normal SAD implemented by MPEG-4 VM, even for full search algorithm. Besides, they employed the diamond search algorithm to minimize the number of search positions that is not regular and is not suitable for hardware design. The average reduction in search positions achieved by our proposed algorithm is larger compared to others. The average search position reduction in [5] is −90.95%, and that for [4] is −96.78%, while it varies from −96.59% to −99.69% for our proposed algorithm. The minor increase in bits/shape produced by the proposed algorithm is not much deviating apart from other fast algorithms. The software implementation of the proposed algorithm has comparable performance to the algorithm presented in [5] which is a software approach. Thus, our proposed algorithm is suitable for software and hardware implementation.

For hardware design comparison, the proposed algorithm is simple to be implemented in hardware and similar to the full search scheme. No special computation circuitry is needed, which can make it switched to a full search without disabling any extra hardware. Control circuit and address generator is simple. For portable devices where the power is critical, when a match occurs, we can disable those PEs which are not a match to save power. BME architecture presented in [2] employs a full search algorithm, and needs a gate count of 9666 while operated at 7.29 MHz for core profile at level two. In comparison, our implementation needs slightly larger gate count of 11582 but needs fewer cycle count and lower frequency, only 3.34 MHz.

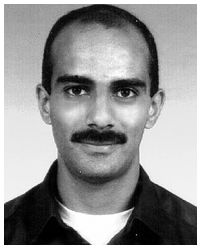
VI. CONCLUSION

In this paper, fast binary motion estimation for shape coding is proposed to save the required search positions to −99.58% of that in the full search algorithm. Due to its simplicity and regularity, we also propose a hardware implementation that only requires 11582 gate-counts with low computational cycle. The flexibility of this algorithm could be further explored through choosing the number of classes, and overlapping between classes to compromise between computational complexity and bit stream length.

REFERENCES

[1] N2502a, Generic Coding of Audio-Visual Objects: Visual 14496-2, Final Draft IS ISO AEC JTC1/SC29/WG11, Atlantic City, NJ, 1998.
[2] Y.-C. Wang, H.-C. Chang, W.-M. Chao, and L.-G. Chen, "An efficient architecture of binary motion estimation for MPEG-4 shape coding," in *Visual Commun. Image Process.*, San Jose, CA, Jan. 2001.

[3] T.-H. Tsai and C.-P. Chen, "An efficient binary motion estimation algorithm and its architecture for MPEG-4 shape coding," in *Proc. ISCAS*, May 2003, vol. 2, pp. 496–499.
[4] ——, "A fast binary motion estimation algorithm for MPEG-4 shape coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 14, no. 6, pp. 908–913, Jun. 2004.
[5] D. Yu, S.-K. Jang, and J.-B. Ra, "Fast motion estimation for shape coding in MPEG-4," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 4, pp. 358–363, Apr. 2003.
[6] K. Panusopone and X. Chen, "A fast motion estimation method for MPEG-4 arbitrarily shaped objects," in *Proc. IEE Int. Conf. Image Processing*, Sep. 2000, vol. 3, pp. 624–627.
[7] MPEG-4 Video Verification Model Version 18.0 ISO/IEC JTC1/SC29/WG11 N3908, 2001.



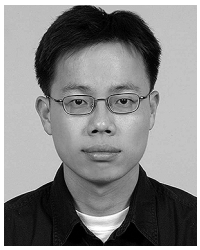
Esam A. Al_Qaralleh was born in Karak, Jordan, in 1972. He received the B.S. degree in electrical engineering in 1995 from the Jordan University, Amman, Jordan. He is currently working toward the Ph.D. degree in the Institute of Electronic Engineering Department, National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

His research interests are video compression, motion estimation.

Tian-Sheuan Chang (S'95-M'99) received the B.S., M.S., and Ph.D. degrees in electronics engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1993, 1995, and 1999, respectively.

During 2000 to 2004, he was with Global Unichip Corporation, Hsinchu, Taiwan, R.O.C. He is currently with Department of Electronics Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., as an Assistant Professor.

His research interest includes IP and SOC design, VLSI signal processing, and computer architecture.



Kun-Bin Lee (S'02–M'04) received the B.S. degree in electrical engineering from National Sun Yat-Sen University in Taiwan, R.O.C., in 1996, and the M.S. and Ph.D. degrees in electronics engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 1998 and 2003, respectively.

He is currently with MediaTek, Inc., Hsinchu, Taiwan, R.O.C. His current research interests include processor architecture, digital signal processing, and system-level exploration with focus on data transfer optimization and memory management for image and video applications.

In 2004, Dr. Lee received the Long-Term Paper Award from Acer, Professor Wen-Zen Shen Thesis Award from Taiwan IC Design Society, and Outstanding Design Award of University LSI Design Contest from ASP-DAC. In 2005, he received Outstanding Paper Award from MediaTek. He is a member of Phi Tan Phi.