Cost-Effective VLSI Architectures and Buffer Size Optimization for Full-Search Block Matching Algorithms

Yuan-Hau Yeh and Chen-Yi Lee, *Member, IEEE*

***Abstract*— This paper presents two efficient very large scale integration (VLSI) architectures and buffer size optimization for full-search block matching algorithms. Starting from an overlapped data flow of search area, both systolic- and semisystolic-array architectural solutions are derived. By means of exploiting stream memory banks, not only input/output (I/O) bandwidth can be minimized, but also processor element efficiency can be improved. In addition, the controller structure for both solutions are very straightforward, making them very suitable for VLSI implementation to meet computational requirements. Moreover, by exploring the dependency graph, we focus on the problem of reducing the internal buffer size under minimal I/O bandwidth constraint to derive guidelines on reducing redundant internal buffer as well as to achieve area-efficient VLSI architectures. Simulation results show that, for $N = P = 16$ (N is the reference block size and P is the search range), I/O bandwidth can be reduced by 2.4 times, while buffer size increases less than 38%. Two prototype chips for $N = P = 16$ have been designed and fabricated. Test results show that clock rate can be up to 90 MHz, implying that more than 87.9-K motion vectors per second can be achieved to meet real-time requirements specified in MPEG-2 MP@ML coding standard.**

***Index Terms*— Block matching, computing architecture, full search, integrated circuits, optimization.**

I. Introduction

BLOCK matching algorithms (BMA's) are often found in image and video applications for pattern analysis, motion detection, data compression, etc. Inherent computational complexities in these algorithms often demand special hardware to meet real-time performance. Due to algorithm regularity and modularity, such algorithms are very suitable for very large scale integration (VLSI) implementation. However, for practical system design considerations, not only computational requirements have to be coped with, but also memory bandwidth has to be minimized to reduce input/output (I/O) pin count and, hence, realization cost. In other words, the desired hardware has to provide sufficient computational power to meet algorithm complexity. Moreover, the large volumes of image/video data have to be managed carefully to enhance on-chip data reusability. From these viewpoints, a good (or so-called cost-effective) VLSI architecture for a BMA is often judged by the following issues:

1) internal storage space, which determines on-chip memory cost and bandwidth;
2) I/O pin count, which determines I/O bandwidth and packaging cost;
3) processor element (PE) efficiency, which determines PE structure and its utilization ratio for a given period.

These issues will be used as platform for performance comparison in the following sections. Motion estimation (ME) based on full-search BMA (FSBMA), one of the key techniques in video coding to remove redundancy, is a good example for illustration of such design complexity. Many research results on VLSI architectures for FSBMA were proposed in the past [1]–[11], [14], where research efforts have moved from meeting computational requirements to reducing memory bandwidth, as well as providing multifunctions specified in standard [17]. This trend results from the consideration that, as VLSI technology advances, more devices can be integrated on-chip to cope with computational requirement, while cost of chip I/O still remains very high. To cope with computational complexity, systolic array (SA) architecture [12], [13] is often exploited. The number of PE's is dependent on the size of reference block and search area. To reduce I/O bandwidth, search data are often organized in a snakelike form to run through a PE array, as found in [5]–[7], and [15], or in an interlacing style, as in [8]. Although I/O pin count is reduced, the time to initialize both reference and search data (i.e., before a PE array can be fully operated) is too long and, hence, overall performance becomes degraded. For example, for 16×16 reference block and $[-16, +15]$ search area, it may take more than half of the total cycle count for data initialization [6], [7]. In addition, for block matching at search boundary, some PE's may become idle because of pipeline filling [5], which also leads to degraded system performance. Yeo and Hu [10] have proven when the search range (P) is set to half of the macroblock size (N), significant reduction in I/O bandwidth can be achieved without sacrificing performance. However, in the case of $N = P$, four cascadable chips are needed, resulting in increasing the implementation cost for a large search area.

To cope with those above-mentioned problems, we have already proposed a semisystolic array (SSA) architecture with double stream memory banks [16]. Based on an overlapped search data flow, we have reached 100% efficiency of a PE

Manuscript received June 26, 1997; revised February 4, 1998 and July 19, 1998. This work was supported by the National Science Council under Grant NSC86-2221-E-009-016.

The authors are with the Department of Electronics Engineering, National Chiao Tung University, Hsinchu 300, Taiwan, R.O.C.

Publisher Item Identifier S 1063-8210(99)02611-6.

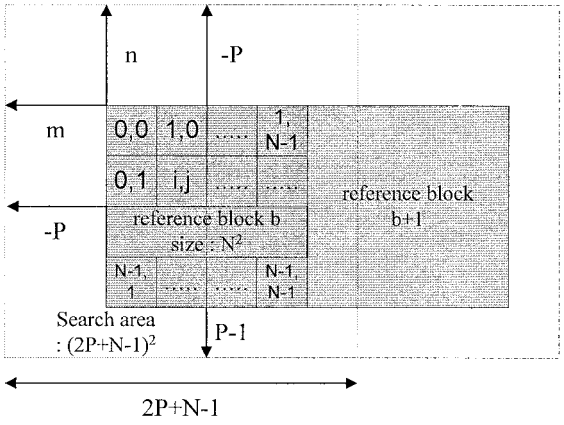


Fig. 1. Image frame structure.

```
for(b=0; b<N_h*N_v; b++)
{
        MV(b) = 0;
        Dtotal(b) = MAX_VALUE;
        for(l=0; l<(2P)^2; l++)
        {
                MAD(l)=0;
                for(k=0; k<N^2; k++)
                {
                        MAD(l) = MAD(l)+|S(l,k)-R(k)|;
                }
                if (Dtotal(b) > MAD(l))
                {
                        Dtotal(b) = MAD(l);
                        MV(b) = l;
                }
        }
}
```

Fig. 2. Three-level loop algorithm for FBMA.

array with a minimum number of I/O pin count. The broadcasting signals in search area requests a special memory design [15] to speed up the motion vector (MV) search. Recently, we have mapped the same data flow onto a pure SA architecture where the merits obtained in SSA architecture still remain the same. The only difference between these two architectures lies in both stream memory size and PE structure. In this paper, we will show the mapping process of these two architectures to see how a common data flow can be mapped onto different architectures. In addition, the proposed two architectures can be scaled for large search area applications, making them also suitable for scalable designs. Comparisons between them and among the solutions found in the literature will also be highlighted to show the benefits of the proposed architectures.

This paper is organized as follows. In Section II, we first briefly discuss the algorithm fundamentals of FSBMA operations derived from dependency graph (DG). In Section III, we present two efficient VLSI architectures to optimally reach 100% PE efficiency. Emphasis will be on the mapping procedure, stream memory bank, PE structure, and controller complexity. In Section IV, we present a scalable solution based on the proposed architectures, where the different size of reference data and search area can be handled. In Section V, we present optimal buffer size estimation from DG analysis. Finally, we compare the results of these two proposed solutions with other available solutions found in the literature.

II. Data Flow and Algorithm Mapping

The basic concept of full-search ME is illustrated in Fig. 1. Usually, a picture frame is divided into many macroblocks, each macroblock containing N^2 pixels. Assuming N_h is the frame's width and N_v is the frame's height, a frame is arranged as $N_h/N \times N_v/N$ macroblocks; (h, v) are the horizontal and vertical macroblock indexes in one frame. For one macroblock, there are $(2P)^2$ candidates inside its search area; (m, n) are the horizontal and vertical candidate indexes for one macroblock. Looking inside the macroblock, (i, j) are the horizontal and vertical pixels indexes in one macroblock. To find the so-called minimum distortion, very often mean absolute difference (MAD) is exploited as the matching criterion because of simple operation. However, for typical applications where 16×16 reference block and $[-16,+15]$ search area are considered, at least 2^{18} MAD and accumulation operations are needed for each reference block. Considering a video sequence consisting of 30 frames, each of which has several thousands of 16×16 reference blocks, more than 10 billion MAD and accumulation operations are needed. To meet such a computational requirement, a two-dimensional (2-D) PE array is often requested. However, how to organize reference and search data in order to minimize memory bandwidth and cycle count for MV search becomes an important issue in enhancing performance of ME processor designs.

The matching criterion of the FBMA can be expressed as

$$\mathrm{MAD}(m,n) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |R(i,j) - S(i+m, j+n)| \quad (1)$$

where R is a "reference block" and S is a "search area." According to the formulas derived in [10],

$$\begin{cases} b = v \cdot N_n + h \\ 0 \le b < N_h \cdot N_v \\ h = b \bmod N_h \\ v = \left\lfloor \dfrac{b}{N_h} \right\rfloor \end{cases} \qquad \begin{cases} l = 2P(m+P) + n + P \\ 0 \le l < (2P)^2 \\ m = l \bmod 2P - P \\ n = \left\lfloor \dfrac{l}{2P} \right\rfloor - P \end{cases}$$

$$\begin{cases} k = i \cdot N + j \\ 0 \le k < N^2 \\ i = k \bmod N \\ j = \left\lfloor \dfrac{k}{N} \right\rfloor . \end{cases} \quad (2)$$

A three-level loop algorithm can then be formulated, as shown in Fig. 2. If we swap the direction of (m, n) and (i, j), we can change the scan order of parameters l and k, and we can then get four basic types of DG, as shown in Fig. 3.

Redrawing the DG of type II in Fig. 4, we set $N = 3$ and $P = 2$, respectively. The dashed lines present the schedule vectors.[1] If we assume that reference data are stored locally (within each PE) and search data are pumped into a PE

[1] In this example, we partition the DG into three parts. Each part has its own schedule vector.

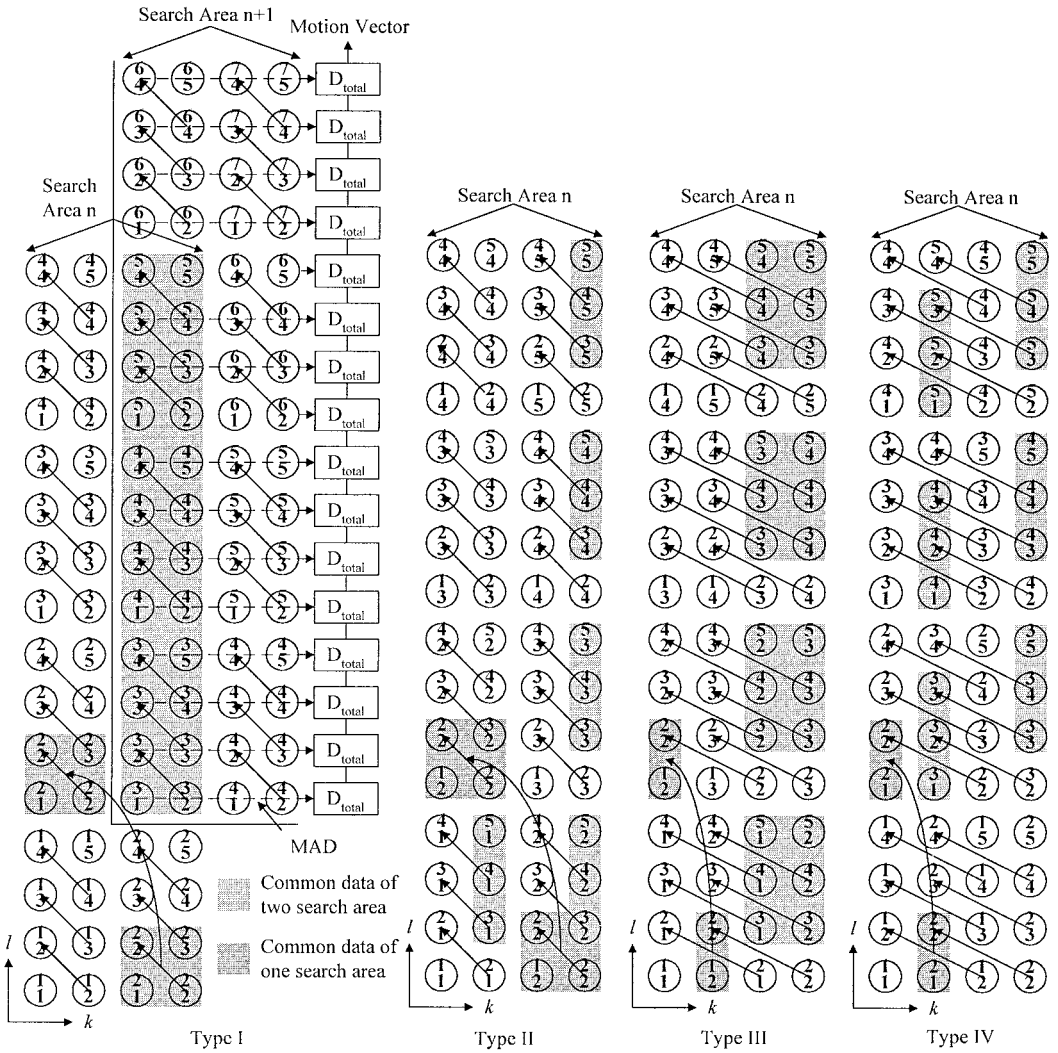


Fig. 3. Four basic types of DG, from type I to IV.

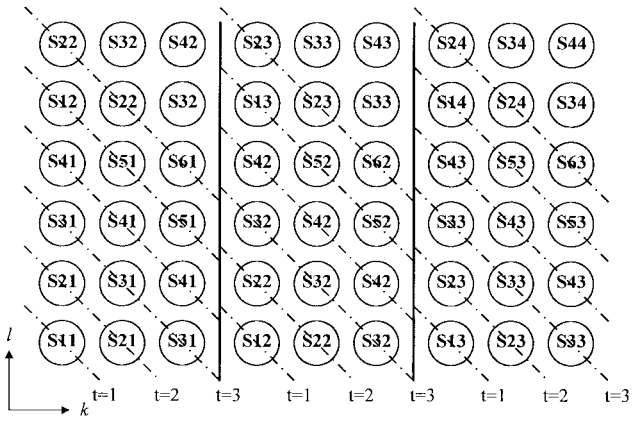


Fig. 4. Partial DG for type II in Fig 3, $N = 3$ and $P = 2$.

array in parallel, all PE's are working simultaneously to compute distortion. However, as the boundary pixels are under processing, part of the PE's becomes idle because boundary search data have to be pumped into a PE array for necessary computations of integer candidates. This process continues until the distortion computation of the boundary candidate is finished, as shown in Fig. 5(a). The problem lies in that *these boundary search data are not fully repeatedly used for N times.* Thus, if we organize search data in a partly overlapping style, as shown in Fig. 5(b), we can make sure that all PE's work on correct data items when an MV search is started. That is, without counting the delay in data initialization and latency, the distortion of each candidate can be obtained within one cycle. In other words, it takes $(2P)^2$ cycles to find an integer–pixel MV because there are $(2P)^2$ integer candidates.

To further illustrate how data are processed in the PE array, we use the example given in Fig. 6. It can be found that without data overlapping, hardware efficiency of the PE array is only about 50%. This low efficiency can be overcome by preloading data onto the next row to PE's when boundary is detected. As shown in Fig. 7(a), when the distortion calculation is done on the boundary, data of the next row should be pumped into a PE array at the next cycle. The mask region indicates that these data have to be simultaneously pumped into the PE array. Fig. 7(b) illustrates the process of distortion calculation at boundary. At cycle 4, boundary is detected. Summation of the distortion values of different candidates are performed respectively at $\mathrm{PE}_{i,1}, i = 1,$

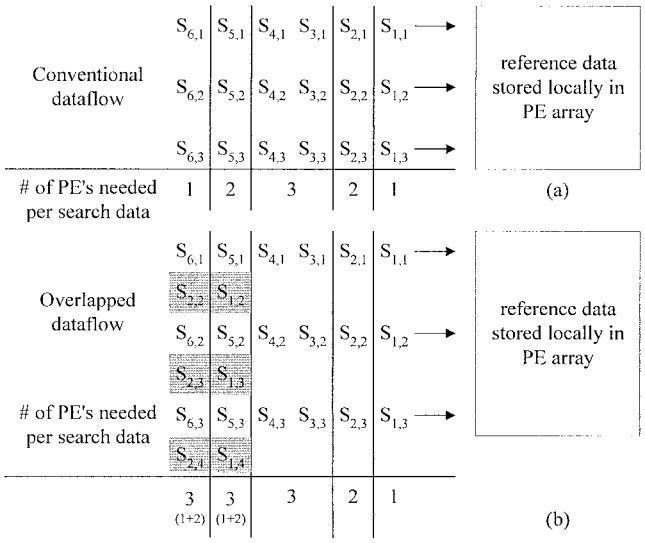


Fig. 5. Data flow and PE usage for FSBMA with reference data stored locally. (a) Conventional data flow. (b) Overlapped data flow.

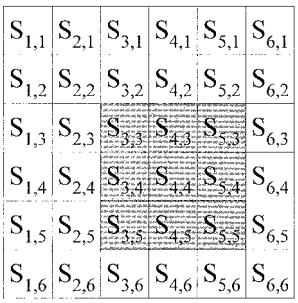


Fig. 6. Example of illustration for $N = 3$ and $P = 2$.

2, 3. At cycle 5, the boundary block still needs the boundary data $(S_{5,1})$, which should be pumped into $\mathrm{PE}_{2,1}$ and $\mathrm{PE}_{3,1}$. However, data of the next row $(S_{1,2})$ is now pumped into $\mathrm{PE}_{1,1}$. At cycle 6, data items ($S_{6,1}$ and $S_{2,2}$) are needed. Distortion calculation of the candidate on the boundary is achieved now in $\mathrm{PE}_{3,1}$. In the meantime, distortion of the candidates of the next row are also performed in $\mathrm{PE}_{2,1}$ and $\mathrm{PE}_{1,1}$. At cycle 7, only the data items from the same row are needed. In summary, the first pixel of each row is read out every $(k(2P)+1)$th[2] cycles with data overlapping of $N-1$ cycles, as shown in Fig. 7(a).

In Section III, we'll see how the overlapped search data is achieved by means of stream memory banks and how the FSMBA can be mapped onto both systolic and SSA structures.

III. The Proposed VLSI Architectures

To generate integer MV's, we use the block diagram given in Fig. 8, which mainly consists of the following four blocks:

1) stream memory bank;
2) 2-D PE array;
3) parallel adder tree;
4) compare–select unit.

[2]Here, $k = 0, \cdots, 2P-1$ for every $2P$ candidates.

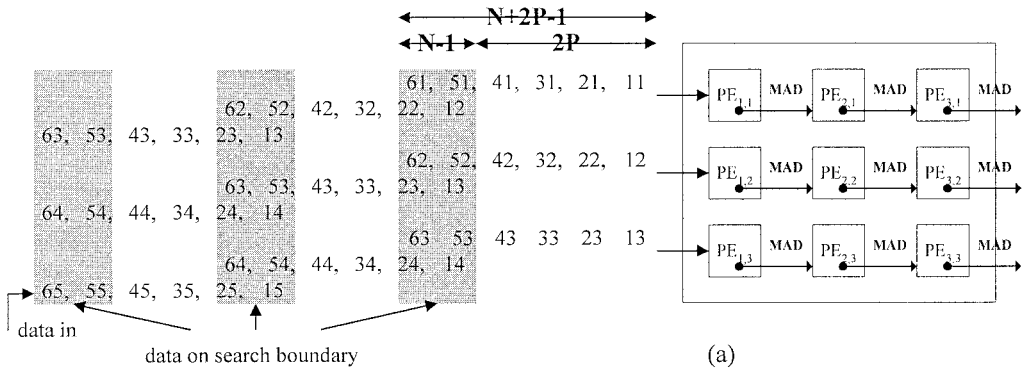
The stream memory is designed to: 1) buffer the search data to reduce I/O bandwidth and 2) to skew the search data to meet requested data flow, as mentioned above. The PE array is designed to provide sufficient computation capability for calculating and accumulating MAD. The parallel adder tree is designed to compute the total MAD for each candidate and then sends it to compare–select unit to generate the MV. Since stream memory bank and PE array are the two key modules in ME processor design, they will be discussed in more detail in the following subsections, which discuss systolic and SSA solutions, respectively.

A. SA VLSI Architecture

From the derived overlapped data flow, it can be found that search data from two different rows (or columns) are needed as a boundary candidate is detected. This implies that two input data buses are needed, where one is always busy and the other is needed only when a boundary candidate is detected. Therefore, how to schedule search data into the PE array so that MAD at each PE can always be calculated becomes a key issue in SA-based realization. This problem can easily be solved by taking into account data flow needed in the PE array. Fig. 9 shows the search data needed in each PE for the first six cycles. The marked data items indicate they are from a different row (or column) and, thus, two input buses are requested. If we can map search data into the PE array in that way, all PE's are working on correct data items, resulting in $(2P)^2$ cycles in obtaining an MV. This also implies that the search area should be partitioned into two regions, which are LS and RS, respectively as shown in Fig. 10. Both regions have the size of $(2P)*(2P+N-1)$ and $(N-1)*(2P+N-1)$, respectively. This partitioning method is very straightforward because it is based on the reference block size N only. Thus, all search data are only accessed once and then buffered in a stream memory bank for other necessary computations. This method is different from overlapped partitioned regions [7], which may increase I/O bandwidth. The stream memory used here can be either shifter register array or pointer addressable memory (PAM) [15]. The overall SA-based architecture is depicted in Fig. 11. Note that the required skew for search data is achieved by both steam memory and PE array since search data on the lower row is always one clock ahead of its upper row. In that case, partial MAD's of each candidate, which are obtained row by row, can be matched.

1) PE Design: To meet real-time performance, very often a 2-D array structure is requested for an ME processor. Based on the above description, we propose a double-port (LS and RS) structure, as shown in Fig. 12. Here, search data run horizontally and MAD and control signal "*cmux*" run vertically. In the initialization phase, all internal registers, except MAD accumulation register "*acc*," are defined appropriately through two data buses. The control register "*sel*" is used to select either "LS" or "RS," which should be accessed during MAD computation.

2) Controller Design: It can be found that from Fig. 9, the control signals needed for each PE are a stair-like waveform, which can easily be obtained by a ring counter, as



at cycle:	search data flow in first PE row	output from each PE (only the 1st row is shown)			candidate
	$PE_{1,1}$ $PE_{2,1}$ $PE_{3,1}$	$PE_{1,1}$	$PE_{2,1}$	$PE_{3,1}$	
4	41 41 41	(41-11),	(41-21)+(31-11)	(41-31)+(31-21)+(21-11)	(-2, -1)
5	12 51 51	(12-11),	(51-21)+(41-11)	(51-31)+(41-21)+(31-11)	(-2, 0)
6	22 22 61	(22-11),	(22-21)+(12-11)	(61-31)+(51-21)+(41-11)	(-2, 1)
7	32 32 32	(32-11),	(32-21)+(22-11)	(32-31)+(22-21)+(12-11)	(-1, -2)

(b)

Fig. 7. An example, with $N = 3$ and $P = 2$ to demonstrate how overlapping strategy is exploited to enhance throughput. (a) Overlapped data flow. (b) Data items within a PE array.

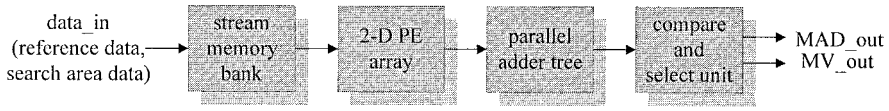


Fig. 8. Block diagram of the ME engine for integer MV search.

cycle	1	2	3	4	5	6
requested search data in PE array	X, X, 11	X, 11, 21	11, 21, 31	21, 31, 41	31, 41, 51	41, 51, 61
	X, 12, 22	12, 22, 32	22, 32, 42	32, 42, 52	42, 52, 62	13, 23, 33
	13, 12, 22	23, 33, 43	33, 43, 53	43, 53, 63	14, 24, 34	24, 34, 44
candidate distortion available at cycle#					(-2, -2)	(-2, -1)

2 cycles delay *

* 2 cycles delay at Parallel adder tree, compare and select

Fig. 9. An overlapped search data flow is developed to allow each PE handling correct data items based on SA architecture.

shown in Fig. 13. As an MV search is activated, this ring counter generates the required control signals, which are then connected to $(N-1)$ PE columns. Note that this ring counter is reset every $2P$ cycles.

3) Performance Evaluation: Here, we summarize the performance of the SA-based ME architecture as follows, where reference size of $N \times N$ and search range of $[-P, P-1]$ are assumed.

1) *Initialization phase*: It takes $2P * (N-1)$ cycles to preload both reference data and search data.
2) *Execution phase*: It takes $(2P)^2$ cycles to find one MV, thus, the PE array efficiency[3] becomes 68% for $N = P = 16$.

[3]The efficiency can achieve 100% if double buffering is exploited, i.e., use another buffer to preload next reference and search data. However, this needs to revise PE structure and makes the controller more complex.

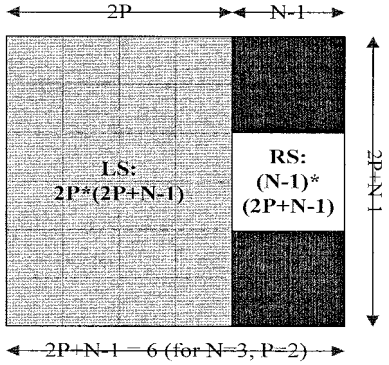


Fig. 10. Search area partitioned into two regions, namely LS and RS, which contain the sizes of $(2P + N - 1) \times 2P$ and $(2P + N - 1)$ $((N - 1)$, respectively.

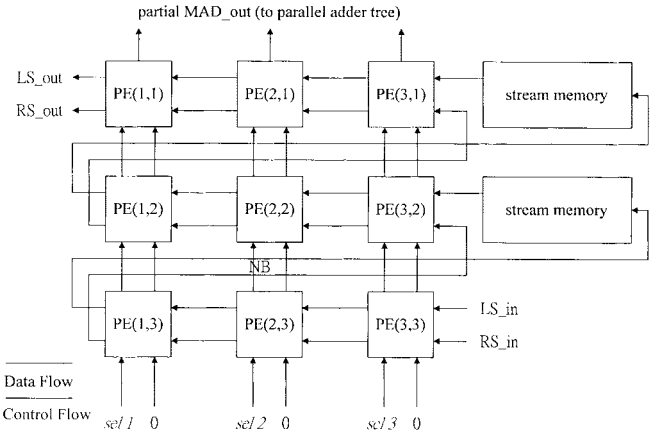


Fig. 11. SA architecture for the FSBMA.

3) *Stream memory size*: It only needs $(N-1)\times(2P-N+1)$ storage space to buffer part of the LS search area.[4]
4) *Pipelinable design*: A pipeline register can be directly inserted to enhance clock rate.

B. SSA VLSI Architecture

Again, based on the overlapped data flow given in Fig. 2(b), we can develop another VLSI architecture where search data are broadcast to a PE array on the same row (or column). The overall ME architecture is shown in Fig. 14. The search data needed in the PE array (for $N = 3$, $P = 2$) is shown in Fig. 7(b). It can be found that PE's on the same column are computing partial MAD's of the same candidate. These partial MAD's are then moved right for accumulation. This is different from that of the SA version where PE's of the same row compute partial MAD's of the same candidate. As a boundary candidate is detected, we need to provide two search data to the PE array to avoid idle operations. Thus, two global buses are running through a PE array where each PE selects the correct search data item. Since search data are now stored in the stream memory, the required space becomes $(2P+N-1)*(N-1)$. To reduce memory area, a dedicated PAM module was developed [15].

[4]Part of the RS and LS search area is buffered in the PE array.

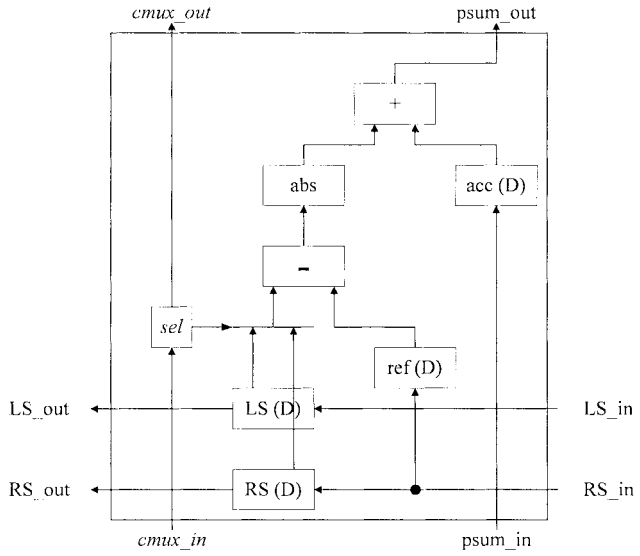


Fig. 12. The structure of PE for SA-based architecture.

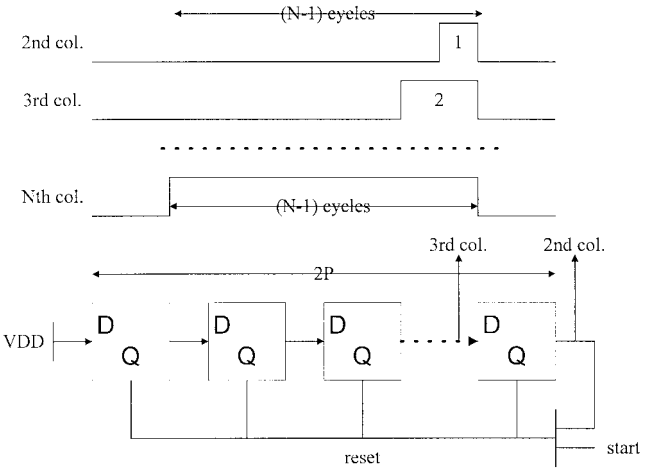


Fig. 13. Ring-counter-based controller for the PE array of SA architecture. The size of this ring counter is equal to $\mathrm{MAX}(2P, N)$.

1) SSA-Based PE Design: The design of the PE is aimed to select search data from two alternative buses (namely, LS and RS) for computing partial MAD. Like SA-based PE design, we have to select either LS or RS search data, which are now from the two global buses. The selected data is stored inside PE to compute local MAD, which is then accumulated with the partial MAD obtained from previous PE's. Thus, the structure of the SSA-based PE, as shown in Fig. 15, is very similar to that of SA-based PE, except the following: 1) global buses are running through the PE, i.e., LS, RS, and *cmux*, and 2) accumulation of the partial MAD is running in the same direction as that of search data.

2) SSA-Based Control Generation: The required control signals are very similar to those used SA-based architecture. As the boundary candidate is detected, which occurs every $2P$ cycles, control signals for boundary search data are activated and exists in $(N-1)$ cycles. The same control signals are needed in the PE array to select the correct search data to compute partial MAD. Since the required control signals are the same as that discussed in SA-based architecture, a very simple ring counter can be exploited here.

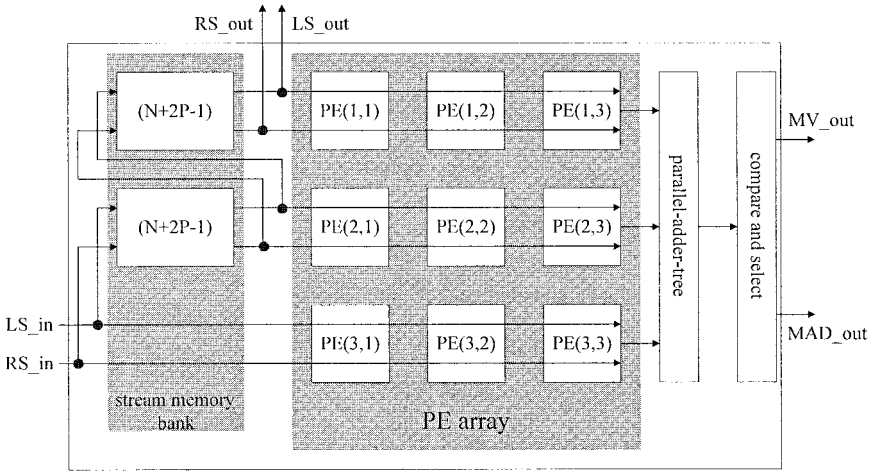


Fig. 14. SSA-based architecture for an FSBMA ME processor with $N = 3, P = 2$.

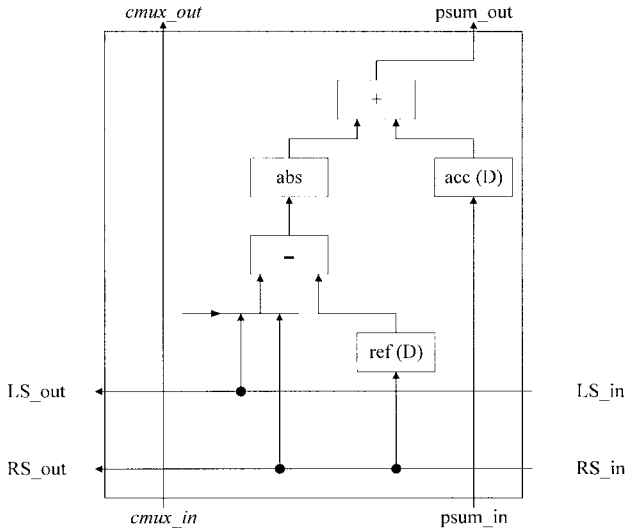


Fig. 15. The structure of PE for the SSA-based architecture.

3) Performance and Requirements: Here, we summarize the features and hardware requirements for the SSA-based architecture for an FSBMA ME processor.

1) *Initialization phase*: It takes $2P * (N - 1)$ cycles to preload both search data and reference block through two input buses.
2) *Execution phase*: It takes $(2P)^2$ cycles to obtain one MV.
3) *Stream memory size*: It needs $(2P + N - 1) * (N - 1)$ storage space to buffer search data so that I/O bandwidth can be reduced.
4) *Double buffering to improve performance*: The initialization phase takes more than 32% (for $N = P = 16$) of the total cycle count. However, with another buffer, this problem can be solved. In that case, we do not have to modify the PE structure, except that input ports have to be increased to meet the need of requested data rate.[5]

[5]This is not always true because for neighboring reference blocks, their search area are partly overlapped. Thus, part of the desired search area for initializing the next reference block can be directly obtained and, hence, the I/O bandwidth can be reduced.

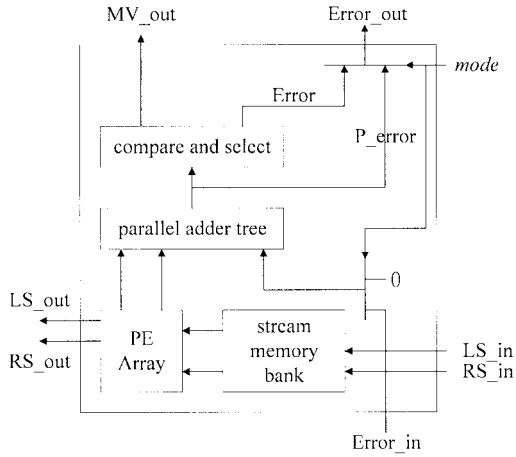


Fig. 16. Illustration of the scalable design based on the proposed architectures.

5) *Pipelining capability*: The PE array can still be pipelined to enhance clock rate if the access time of stream memory can be matched.

IV. Scalable VLSI Architectures

In this section, we present a scalable design based on the two ME architectures proposed above. Since both of them have same I/O terminals, we will use an abstract view to illustrate how they can be cascaded (or combined) for a large reference block and search area. We will directly use some examples to show the mapping procedure. However, before doing that, we will first modify the available ME structure so that scalability is allowed.

A. Revising Overall ME Structure

We first assume that each ME processor can handle a reference block of $N \times N$ and search range of $[-P, P-1]$. To provide scalable capability, we have to take into account both data communications and I/O pin count into account. For example, both MAD and MV should be sent out when search range increases; for large reference block size, partial MAD has to be sent out. Based on these two considerations, we have modified the structure of the parallel adder tree, as shown in Fig. 16. The partial MAD from other ME processors

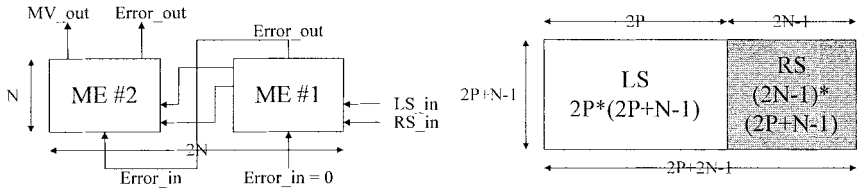


Fig. 17. Scalable ME solutions for large reference block size $(2N \times 2N)$, where each MV can be obtained within $(2P)^2$ cycles. (a) Two-processor configuration. (b) Corresponding search area partition.

can be accumulated by the parallel adder tree and then sent out for further accumulation. This ME processor can also be used stand alone to find the MV and corresponding MAD by simply setting the *mode* option.

B. Case I: Increasing Reference Block Size

We assume that each ME processor can handle $N \times N$ reference block, search range of $[-P, P-1]$ and produce one MV every $(2P)^2$ cycles. If, for example, the reference block becomes $2N \times N$ and search range remains the same, we need two ME processors to produce one MV every $(2P)^2$ cycles. Fig. 17(a) shows the scalable design, where each ME processor computes partial MAD for each $N \times N$ block. The partial MAD from PROC1 is then sent to PROC2 to find total MAD of each candidate. MV and corresponding MAD are then obtained from the compare–select unit in PROC2 every $(2P)^2$ cycles without counting latency. Note that to reach the performance, search data has to be arranged as shown in Fig. 17(b), which is again partitioned into RS and LS regions. Once data are preloaded in $2*(N-1)*(2P)$ cycles, MV's can be obtained every $(2P)^2$ cycles.

C. Case II: Increasing Search Range

We assume that the search arrange is extended to $[-2P, 2P-1]$ and MV still has to be located in $(2P)^2$ cycles. Without using multiple ME processors, it takes $(4P)^2$ cycles to find one MV. We thus partition the search area into four regions, each of which will be handled by one ME processor. For example, the search range from $[-2P, -1]$ is handled by ME PROC1, as shown in Fig. 18(a). The search area is partitioned into two regions with overlapped $(N-1)$ rows and columns, as shown in Fig. 18(b). These four ME processors are executing in parallel to compute local minimal MAD's, which are then further processed to find the final MV.

D. Memory Bandwidth Requirement

The required bandwidth[6] for off-chip memory can be calculated as follows:

$$((N+2P-1)^2+N^2)/(2P)^2. \tag{3}$$

Here, search area, reference block, and available cycle count are given for $N \times N$ reference block and $[-P, P-1]$ search range. If each input sample is kbit wide, then the required

[6]Here, we assume that input rate from off-chip memory is equal to on-chip clock rate.

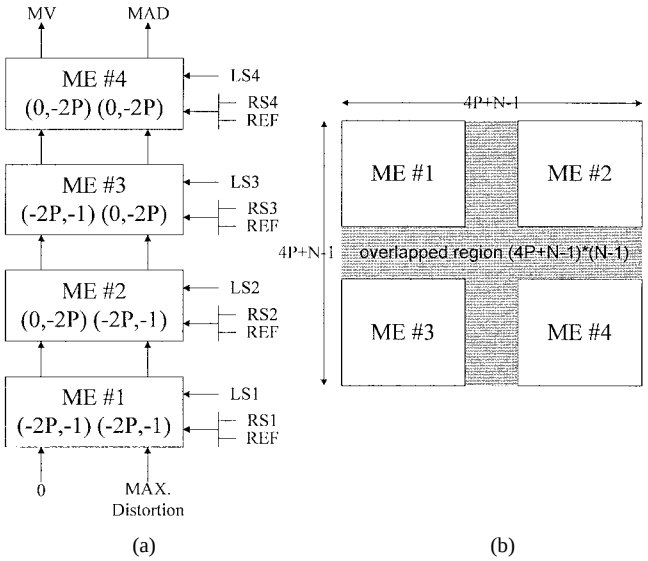


Fig. 18. Scalable ME solutions for large search range $[-2P, 2P-1]$, where each MV can be obtained within $(2P)^2$ cycles. (a) Cascadable structure. (b) Partitioning of the search area.

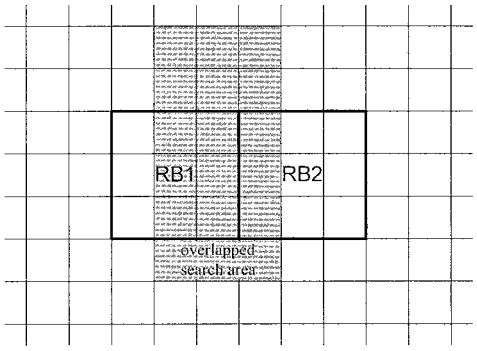


Fig. 19. Illustration of an overlapped search area for neighboring reference blocks.

bandwidth becomes

$$k \times ((N+2P-1)^2+N^2)/(2P)^2. \tag{4}$$

If the initialization delay of loading the search area is to be avoided, i.e., one MV can be obtained every $(2P)^2$ cycles, the required memory bandwidth can still remain the same. However, another buffer for both search data and reference block should be provided. Note that for neighboring reference blocks, the required memory bandwidth can be reduced because of the overlapped search area, as shown in Fig. 19. Therefore, part of the search area for the next reference block can be directly obtained from the available

TABLE I
DEPENDENCY ARCS DERIVED FROM FIG. 3

Dependency Arc	Type I	Type II	Type III	Type IV
R, D_{total}, MV	[1,0]	[1,0]	[1,0]	[1,0]
MAD	[0,1]	[0,1]	[0,1]	[0,1]
S_{within_block} (S1)	[1,-1] or [-1,1]	[1,-1] or [-1,1]	[1,-N] or [-N,1]	[1,-N] or [-N,1]
S_{within_SA}* (S2)	[2P,-N]	[2P,-N]	[2P,-1]	[2P,-1]
$S_{between\_SA}$** (S3)	[-2P,-N(N-1)] or [-2NP,0]	[-1, -(N-1)]	[-1,-N]	[-2P,-1]

* Data dependency in the same search area.
** Data dependency between two adjacent search areas.

search area. The required average memory bandwidth thus becomes

$$k \times \{(N+2P-1)^2 + N^2 + (\lceil S_h/N \rceil - 1) \times (N \times (N+2P-1) + N^2)\} \times \lceil S_h/N \rceil \times (2P)^2. \quad (5)$$

For $N = P = 16$ and $S_h = 720$, the required average bandwidth is now reduced from 2.41 to 1.02 K.

V. OPTIMAL BUFFER SIZE ESTIMATION FROM DG ANALYSIS

Analyzing the DG's shown in Fig. 3, we can extract the dependency $\vec{b}$ arc of all variables. Table I shows the dependency arcs which are necessary to evaluate buffer size for a given mapping strategy. R is the reference block data, D_{total} is the distortion value of a candidate block, MV is the MV of one macroblock, and S is the search area data. Data dependency in the search area are more complicated than other variables. In Fig. 3, observing the DG of type I, there are three types of relations in S-variables which will be discussed in detail in the following subsections. From this table, we know that, for different mapping directions, we can choose freely various dependency arcs unless they violate the mapping constraints. With this table, we can evaluate optimized memory size for different mapping directions under I/O bandwidth constraint. Note that we withdraw types III and IV from our discussion because they cannot be used to generate cost-effective architecture mappings. Dependency arcs can be mapped to PE array and a set of dependency edges and delays are acquired in the following equation, which is described in [12]:

$$\begin{bmatrix} D(\vec{e}) \\ \vec{e} \end{bmatrix} = \begin{bmatrix} \vec{S}^T \\ P^T \end{bmatrix} \cdot [\vec{b}] \quad (6)$$

is the delays on edge $\vec{e}$ in the PE array, $\vec{S}$ is the schedule vector, and P is the processor basis. According to delays of each dependency edge $\vec{e}$, we can derive total memory size for a giving mapping.

Memory elements can be classified as: 1) registers in each PE and 2) memory banks outside the PE array. Our target is to find the solution for the following equation:

$$\text{Minimize} \quad C_1\vec{S}^T \cdot \vec{b}_R + C_2\vec{S}^T \cdot \vec{b}_{\text{MAD}} + C_3\vec{S}^T \cdot \vec{b}_{\text{total}} + C_4\vec{S}^T \cdot \vec{b}_{S1} + C_5\vec{S}^T \cdot \vec{b}_{S2} + C_6\vec{S}^T \cdot \vec{b}_{S3}. \quad (7)$$

Under the condition of minimum I/O bandwidth. $(C_1 \cdots C_6)$ is the edge count of data dependency $\vec{b}$. From Fig. 3, we know most existent architectures for FSBMA set their projection direction $\vec{d}$ to [1,0] or [0,1], in other word, most 2-D PE arrays contain N^2 or $(2P)^2$ PE's. In Table II, we know for the four conditions $D(\vec{e}_R)$, $D(\vec{e}_{\text{total}})$, and $D(\vec{e}_{\text{MAD}})$ are all equal to one. These cause one or zero delay in each PE depending on the projection direction $\vec{d}$. For example, in Type I(a), $D(\vec{e}_{\text{total}})$ does not result in any delay in each PE, while $D(\vec{e}_R)$ causes one delay in each PE. Those delays mentioned above are inevitable and, hence, the buffer minimization problem falls on exploiting $D(\vec{e}_{S2})$ and $D(\vec{e}_{S3})$. In following subsections, we describe this problem of the four conditions, respectively, focusing on the condition $N = P$.

TABLE II
DELAYS ON DEPENDENCE EDGES

	Type I (a)	Type I (b)	Type II (a)	Type II (b)
$\vec{d}$	[1,0]	[0,1]	[1,0]	[0,1]
$D(\vec{e}_R)$	1	1	1	1
$D(\vec{e}_{MAD})$	1	1	1	1
$D(\vec{e}_{S1})$	0	0	0	0
$D(\vec{e}_{S2})$	2P-N	2P-N	2P-N	2P-N
$D(\vec{e}_{S3})$	-2NP+$(2P)^2$ *	N-2P **	$(2P)^2$-N	N^2-N
PE-count	N^2	$(2P)^2$	N^2	$(2P)^2$

* $(2P)^2$ term indicates the effective latency of one search area.
** Particular negative delay value will be described lately.

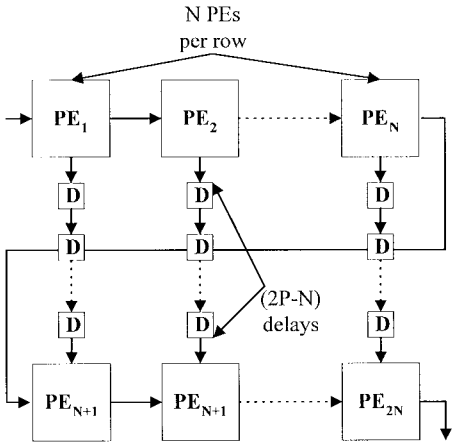


Fig. 20. Delay elements for dependency edge $\vec{e}_{S2}$.

A. Type I(a)—$\vec{S} = [1,1]$, $\vec{d} = [0,1]$, $P = [0,1]$

First, from Table II, we know $D(\vec{e}_{S2}) = 2P - N$ and $\vec{e}_{S2} = -N$. We can arrange N^2 PE's as snakelike, as shown in Fig. 20. Applying the mapping information shown in Table II,

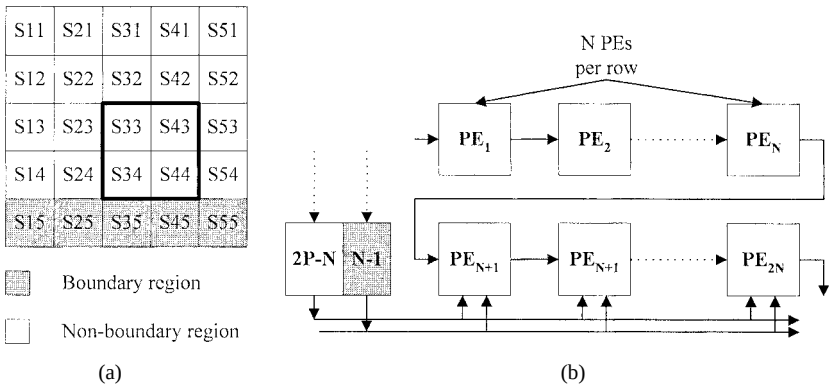


Fig. 21. (a) "Nonboundary" and "boundary" regions of one search area. (b) Buffer size reduction after "merging" procedure.

$N * (2P - N)$ registers are needed between two adjacent rows. However, there exists data dependency among the data in $N * (2P - N)$ registers, which can be easily read from examining Fig. 3. We then apply a procedure called "merging" in order to greatly reduce the register count. From the DG, we know sometimes "nonboundary" and "boundary" data must be passed to the same row, like $(S21, S15)$ or $(S31, S25)$, etc. If we assume $N = P$, the two following inequations are acquired:

$$\begin{aligned} \text{delay} &= 2P - N > N - 1 \\ \text{delay} &= 2P - N < N + P \end{aligned} \tag{8}$$

$(2P-N) > (N-1)$ means only $(N-1)$ registers are sufficient to buffer "boundary" data and $(2P - N) < (N + P)$ means at least $2P - N$ (this value is right equal to $D(\vec{e}_{S2})$, this is the lower bound size to buffer "nonboundary" data) registers are needed to buffer "nonboundary" data. After applying the "merging" procedure, we can reduce the buffer size from $N * (2P - N)$ to $(2P - 1)$, as shown in Fig. 21(b).

Next, another key point to reduce the I/O bandwidth is to consider dependency arc $S3$; $S3$ also means the common data of two adjacent search areas depicted in Fig. 3. Fig. 22 shows three conditions if we choose different $S3$. One idea to reduce the buffer size is: *the reusable data is passed to the buffer for finding the next MV as late as possible.* Thus, condition III is a good choice. Thus, we get $S3 = [-2NP, 0]$ and $D(\vec{e})_{S3} = 2P * (2P - N)$. Observing the condition III in Fig. 22, from $(t + 1)$ to $(t + 2P)$ in the time axis, $(N + 2P - 1)$ registers are needed to buffer the data. Thus, for $D(\vec{e}_{S3}) = -2NP + (2P)^2$, dependency edge $\vec{e}_{S3}$ results in $(2P - N) * (N + 2P - 1)$ buffer size. According to the above discussion, a formula for the buffer size estimation of DG type I(a) is given as

$$\begin{aligned} N^2 \cdot (\vec{S}^T \cdot \vec{b}_R + \vec{S}^T \cdot \vec{b}_{\text{MAD}} + \vec{S}^T \cdot \vec{b}_{S1}) + (N-1) \cdot (2P-1) \\ + (2P-N) \cdot (N+2P-1). \end{aligned} \tag{9}$$

B. Type I(b)—$\vec{S} = [1, 1]$, $\vec{d} = [0, 1]$, $P = [1, 0]$

This case causes $D(\vec{e}_{S3}) = N - 2P < 0$. Although $D(\vec{e}_{S3})$ violates the constraint $\vec{S}^T \cdot \vec{d} \geq 0$, appending two adjacent search areas together, as shown in Fig. 23, is a special case when $N = P$. $D(\vec{e}_{S3})$ does not cost any buffer to hold data.

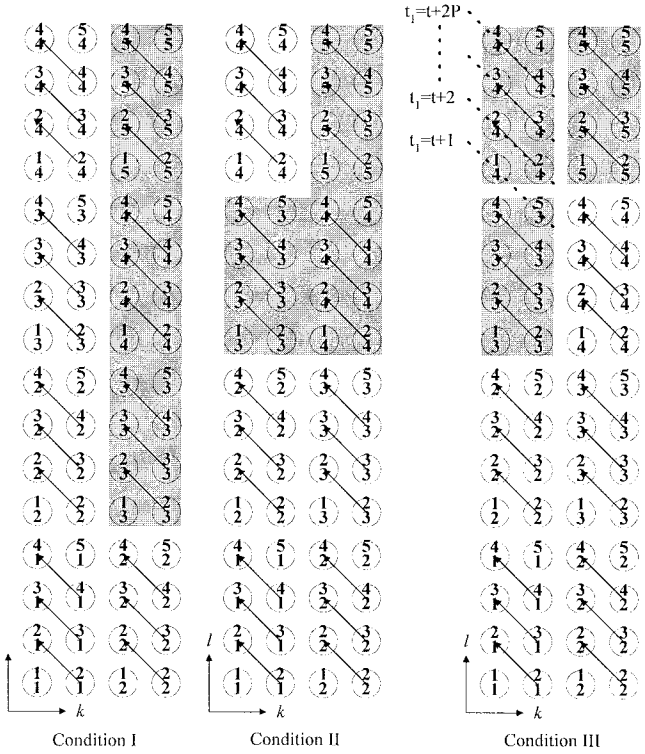


Fig. 22. Three conditions for dependency arc $S3$.

Let us just consider $(\vec{e}_{S2}) = 2P - N$ and $\vec{e}_{S2} = 2P$. The $(2P)^2$ PE's can be arranged as a snakelike form, as shown in Fig. 24. At time t, three pixels ($S35$, $S43$, and $S51$) are sent to row n concurrently. After the "merging" procedure, we can reduce the buffer size from $2P * (2P - N)$ to $(4P - N - 1)$ per row. The total buffer is given as

$$\begin{aligned} (2P)^2 \cdot (\vec{S}^T \cdot \vec{b}_R + \vec{S}^T \cdot \vec{b}_{D_{\text{total}}} + \vec{S}^T \cdot \vec{b}_{\text{MAD}} + \vec{S}^T \cdot \vec{b}_{S1}) \\ + (2P-1) \cdot (4P - N - 1). \end{aligned} \tag{10}$$

C. Type II(a)—$\vec{S} = [1, 1]$, $\vec{d} = [1, 0]$, $P = [0, 1]$

In this case, the buffer size for $\vec{e}_{S2}$ is the same as DG type I(a). From Table II, we know $D(\vec{e}_{S3}) = (2P)^2 - N, \vec{e}_{S3} = -(N - 1)$. Applying the concept described in Section V-A, the whole common data between two adjacent search areas should be buffered, thus, the buffer size for $\vec{e}_{S3}$ is

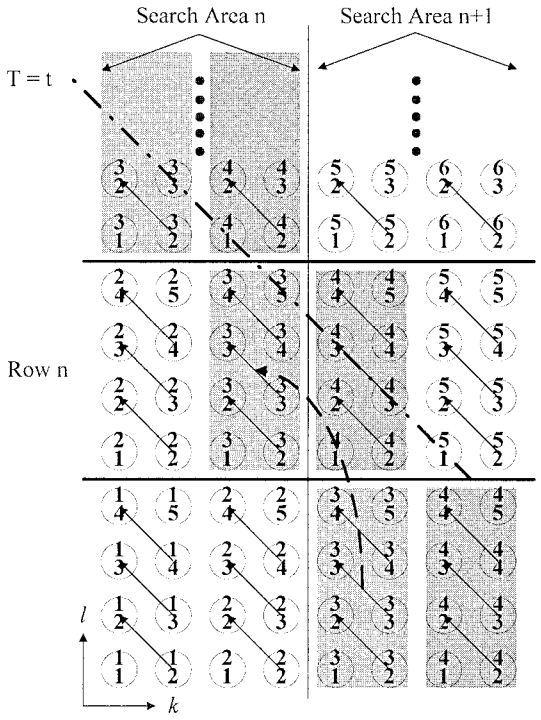


Fig. 23. Partial DG between two search areas of Type I(a).

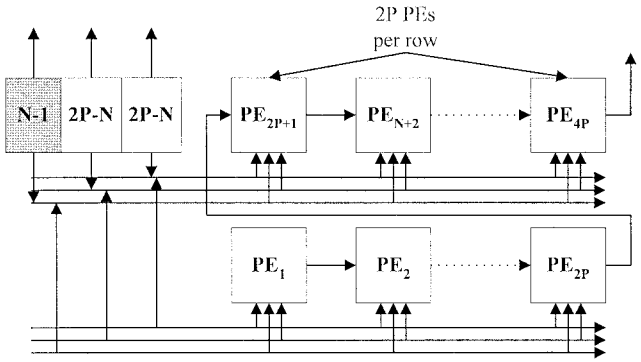


Fig. 24. PE array and buffer structure of DG type I(b).

$(2P-1)*(N+2P-1)$. The total buffer for DG type II(a) is

$$N^2 \cdot (\vec{S}^T \cdot \vec{b}_R + \vec{S}^T \cdot \vec{b}_{\text{MAD}} + \vec{S}^T \cdot \vec{b}_{S1}) + (N-1) \times (N+P-1) + (2P-1) \cdot (N+2P-1). \quad (11)$$

D. Type II(b)—$\vec{S} = [1,1]$, $\vec{d} = [0,1]$, $P = [1,0]$

In this case, the buffer size for $\vec{e}_{S2}$ is the same as DG type I(b), which is described in Section V-B. Consider the illustrated example shown in Fig. 25. At time $t+1$ to $t+N+1$, the common data (i.e., $S31$, $S41$, and $S51$, the size is $2P-1$) between two adjacent search areas should be stored in the buffer every N cycles. For $D(\vec{e}_{S3}) = N^2 - N$, the buffer size for $\vec{e}_{S3}$ is $(N^2-N)/N \times (2P-1) = (N-1) \times (2P-1)$. Thus, the total buffer size is given as

$$(2P)^2 \cdot (\vec{S}^T \cdot \vec{b}_R + \vec{S}^T \cdot \vec{b}_{D_{\text{total}}} + \vec{S}^T \cdot \vec{b}_{\text{MAD}} + \vec{S}^T \cdot \vec{b}_{S1}) + (2P-1) \cdot (N+2P-1) + (N-1) \times (2P-1). \quad (12)$$

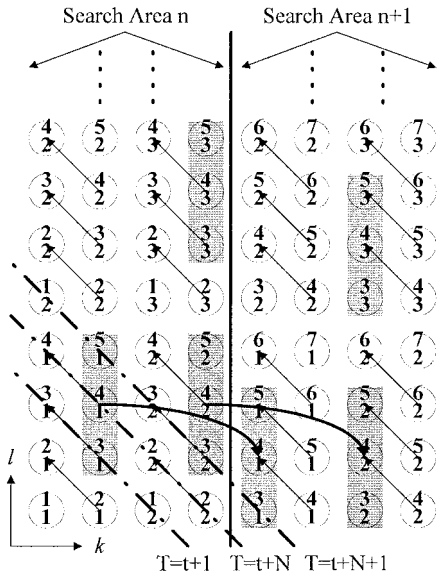


Fig. 25. Partial DG for Type II(b).

E. Mapped ME Architectures

According to the discussion in Section V-A, beside registers in each PE, two extra buffers are used to buffer the data for $\vec{e}_{S2}$ and $\vec{e}_{S3}$; theses two buffers are illustrated in Fig. 26 as buffers I and II, respectively. The buffers can be implemented as pointer address memory described in [15]. There are two buses passing each row of the PE array, supplying "boundary" and "nonboundary" data concurrently. In order to reduce I/O bandwidth, the penalty paid is complex routing, and the data broadcasting problem during changing to the next search area should be carefully handled.

Also, according to the discussion in Section V-B, the mapped architecture is shown in Fig. 27 and the size of the PE array is $(2P)^2$. There are three buses passing each row of the PE array. The reason is in the worst case, as three pixels should be consumed at the same time.

VI. Comparison and Discussion

In this section, we first present a systematic procedure for the design of cost-effective ME processors based on the two proposed architectures described in Section III. We then compare the achieved performances with available ME solutions found in the literature.

A. A Systematic Design Procedure for FSBMA ME Processor

From the description of algorithm mapping and architecture design, it can be found that good data partitioning may result in reducing memory bandwidth and, hence, I/O pin count can be minimized without degrading overall system performance. Idle operations can be avoided if both initialization delay and boundary search data are handled appropriately. Below, we summarize the key procedures for designing scalable ME processors based on FSBMA.

1) *Search area partitioning*: The first step is to partition search area such that all PE's work all the time once an MV search is activated. This is often found in the boundary candidate whose search data are reused less

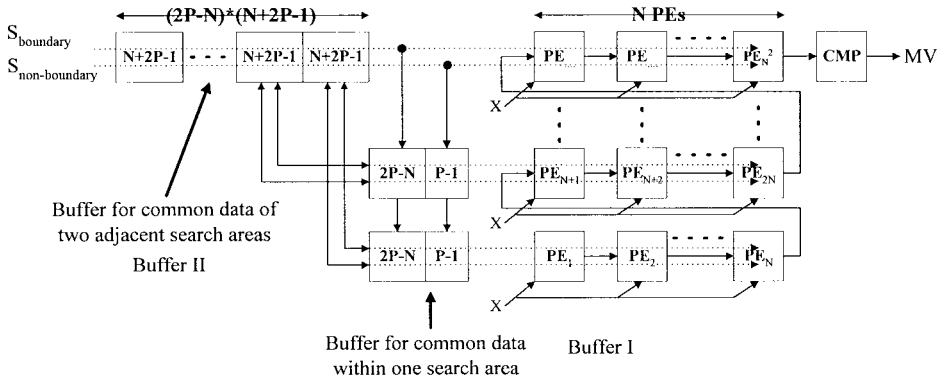


Fig. 26. Mapped ME architecture for Type I(a).

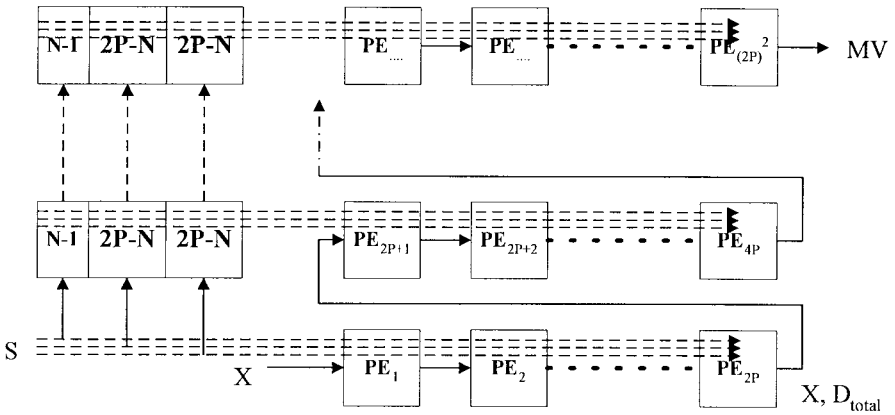


Fig. 27. Mapped ME architecture for Type I(b).

than N times. Therefore, these search data have to be overlapped with other search data to ensure that all PE's are working on correct data items. In principle, search area is partitioned into two regions whose size are $(2P+N-1)*2P$ and $(2P+N-1)*(N-1)$, respectively, as shown in Fig. 10.

2) *ME processor structure*: The ME processor mainly consists of four blocks, namely, stream memory bank, PE array, parallel adder tree, and compare–select, as shown in Fig. 8. If search data flow and partial MAD accumulation is running in the same direction, the SSA architecture solution is proposed. On the other hand, if they run orthogonally, the SA architecture is proposed. Total storage space[7] needed for both the search area and the reference block is $4N^2+(N-1)*(2P-N+1)$. Without counting the initialization delay, each MV can be obtained every $(2P)^2$ cycles. By setting mode lines, we can use multiple ME processors to handle the large reference block and search range where any MV can still be obtained within $(2P)^2$ cycles. As for the controller, a ring-counter-based structure is proposed.
3) *Data initialization phase*: In practical applications, we have to take the initialization delay into account because only when both search and reference data are placed appropriately can an MV search be activated. If the initialization phase takes too many cycles, the overall performance becomes degraded. In our proposals, both architectures need $(N-1)*(2P)$ cycles to preload search and reference data through two input buses. Thus, it totally takes $(2P)^2+(N-1)*(2P)$ cycles to find one MV. For $N=P=8$, the PE array becomes idle about one-third of the total cycle count. To solve this initialization delay problem, a double buffering scheme is suggested. Although this results in the need of doubling storage space and three input buses, i.e., one for the reference block and the other two for the

[7]The storage space includes registers within PE array and the stream memory bank.

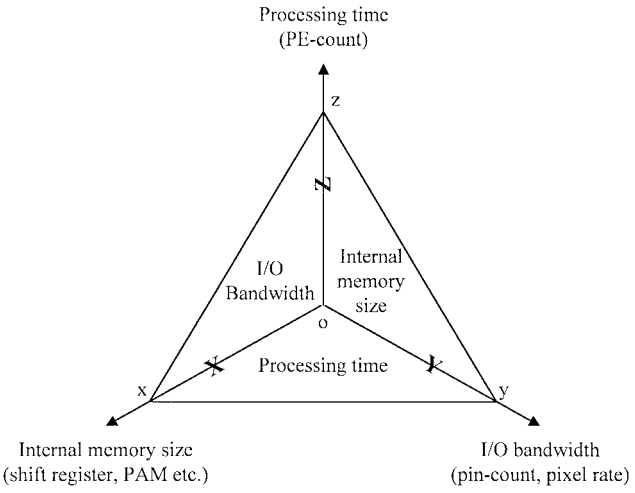


Fig. 28. Tradeoff diagram of processing time, internal memory size, and I/O bandwidth.

TABLE III
PERFORMANCE COMPARISON BASED ON $N = 16$, $P = 8$, $S_h = 720$, $S_v = 480$, $F_r = 30$

Design issues	MV count per second	cycle-count per MV	internal memory size (bytes)	I/O bandwidth (Mbits/sec)	PE-count	I/O pin-count	PE's complexity & word-length
Hsieh et. al[5]	126.6K	961	1249	49.3	256	16	2 adders, 12-bit
Wang et. al[8]	312.5K	256	1904	31.1	256	24	2 adders + 1 comparator, 16-bit
Our approach I	351.6K	256	1054	49.3	256	24	2 adders + 1 mux, 12-bit
Our approach II	351.6K	256	1442	49.3	256	24	2 adders + 1 mux, 12-bit

TABLE IV
PERFORMANCE COMPARISON BASED ON $N = 16$, $P = 16$, $S_h = 720$, $S_v = 480$, $F_r = 30$

Design issues	MV count per second	cycle-count needed per MV	Internal memory size (bytes)	I/O bandwidth (Mbits/sec)	PE-count	I/O pin-count
Hsieh et. al[5]	55.1K	2209	1489	99.8	256	16
Baek et. al[7]	40.9K	1760	1730	120	256	16
Wang et. Al[8]	312.5K	256	7616	52.2	1024	48
Yeo& Hu[10]	156.3K	256	3072	93.3	1024	72
Our approach I-1	351.6K	256	4472	128	1024	56
Our approach I-2	87.9K	1024	1278	99.8	256	24
Our approach I-1	351.6K	256	6024	128	1024	56
Our approach II-2	87.9K	1024	1726	99.8	256	24
Type I(a)	87.9K	1024	1729	42.1	256	24
Type I(b)	351.6K	256	4529	42.1	1024	32
Type II(a)	87.9K	1024	2434	42.1	256	24
Type II(b)	351.6K	256	4994	42.1	1024	32

search area, the cycle count needed to obtain one MV still remains $(2P)^2$ cycles.

4) *Tradeoff*: From the above discussion, it can be found that three factors may have influence on the ME processor design. They are: 1) PE count; 2) internal memory size; and 3) I/O bandwidth. We use Fig. 28 to illustrate the relationship among these factors. Moving[8] along axis Z, we can decrease/increase the processing time (cycles/per MV), moving along axis X, we can decrease/increase the internal memory size, and moving along axis Y, we can decrease/increase the I/O bandwidth. The area enclosed by $\overrightarrow{oz}$, $\overrightarrow{ox}$, and $\overrightarrow{zx}$ indicates the I/O bandwidth, the area enclosed by $\overrightarrow{oz}$, $\overrightarrow{oy}$, and $\overrightarrow{zy}$ indicates the internal memory size, and the area enclosed by $\overrightarrow{ox}$, $\overrightarrow{oy}$, and $\overrightarrow{xy}$ indicates the processing time. For example, if we put our effect on reducing the processing time (one way is to add the PE count), we may increase the internal memory size or increase I/O bandwidth or both. If the I/O bandwidth (implying the pin count or the loading of the memory management unit) is critical, we can increase (moving x toward the original point) the internal memory size or increase the processing time (one way is to decrease PE count), if the requirement is satisfied. The final item is to decrease the internal memory size, we can increase the I/O bandwidth, or increase the processing time.

B. Performance Comparison

Here, we make some comparisons among the proposed architectures and other solutions found in the literature. The comparisons are based on: 1) memory bandwidth; 2) I/O port for both reference and search data; 3) internal storage space; and 4) cycle count needed to produce one MV when data are initialized appropriately. Note that for internal storage space, we count both registers in the PE array and stream memory in order to make the comparison more fair. We first define the following parameters:

- S_h horizontal size of a frame;
- S_v vertical size of a frame;
- F_r frame size.

The architecture proposed by Hsieh and Lin [5] requires the storage space of $4N^2 + (N-1)*(2P-1)$. However, it takes $(2P+N-1)^2$ cycles to find one MV. The architecture proposed by Baek *et al.* [7] needs storage space of $5N^2 + 2(N-1)*(2P-N+1)$. It needs $N^2+(N-1)*2P+(2P)^2$ cycles to obtain one MV. In addition, memory bandwidth is increased due to overlapped window size of $(2P-N)$. The architecture proposed by Wang *et al.* [8] requires storage space of $(N+1)*2P+6*(2P)^2+12P$. It takes N^2 cycles to obtain one MV. However, it needs three I/O ports. In addition, data flow for the search area becomes more complex. Our approach I (SA-based) requires storage space of $4N^2+2*(N-1)*(2P-N+1)$. Our approach II (SSA-based) requires storage space of $2N^2+2*(N-1)*(2P+N-1)$. Both takes $(2P)^2$ cycles to compute one MV.

Table III shows the performance analysis of available architectures found in the literature. All data are based on the case of $S_h = 720$, $S_v = 480$, $F_r = 30$, $N = 16$, and $P = 8$ with a pipeline register inside each PE. If we consider the above parameters for scalability, it can be found that control becomes more complex in Wang *et al.* [8], which needs $2P$ counters. As for the architecture proposed by Yeo

[8]Moving outward direction indicates decrement.

and Hu [10], there is a comparator in the PE. Both above architectures need $(8 + \log_2 N^2)$-bit wordlength in each PE. In our PE design, only $(8 + \log_2 N)$ bits in the worst case are needed. If $N = 16$, our design can reduce 25% wordlength of each PE compared to [8] and [10], making our design more cost-effective in VLSI implementation. The data in Table IV are based on the case of $S_h = 720$, $S_v = 480$, $F_r = 30$, $N = 16$, and $P = 16$. Our approach can provide both scalable (our approach I-1, etc.) and single chip (our approach I-2, etc.) solutions. On the other hand, applying the buffer size optimization procedure described in Section V, we can derive optimal buffer size under minimum I/O bandwidth, as shown in Table IV, Type I $\sim$ Type II.

Two prototype chips for $N = P = 16$ were designed and fabricated using 0.8- and 0.6-μm complimentary metal–oxide–semiconductor (CMOS) technology, respectively. Test results show that clock rate up to 90 MHz can be achieved. This implies that the performance reaches 87.9K MV's per second and 90M MAD operations, which is sufficient to meet real-time requirements specified in MPEG-2 MP@ML video coding standard. (Here, the following are assumed: frame size is 720 pixels by 480 lines, frame rate is 30 frames per second, and the search range is $\sim$+15.5–16.5. In the worst case, we assume that all predicted pictures are B-pictures, and the minimum clock rate is $[(720 * 480)/(16 * 16) * 30 * 1024] * 2 = 82.944$ MHz.

VII. Conclusion

In this paper, we have presented two cost-effective VLSI architectures for ME processors. They both are derived from a common data flow with differences in both PE structure and stream memory bank. Results show that PE array efficiency up to 100% can be achieved, where storage space and I/O pin count can still compete with those solutions found in the literature. In addition, the proposed architectures also allow scalability for different sizes of reference block (N) and search range (P). Thus, they are very suitable for high-definition video coding systems. Moreover, a guideline has also been derived for the optimization of internal buffer size of an FSBMA ME processor. Optimized buffer size helps greatly reducing I/O bandwidth with little penalty of implementation cost in application-specific integrated circuit (ASIC) design. We are currently investigating the possibility of mapping a hierarchical search algorithm onto these proposed architectures for large search area applications.

Acknowledgment

The authors would like to thank M. C. Lu, G. L. Tzeng, and H. Z. Huang for many helpful discussions in preparing this manuscript. The multiple project chip (MPC) support from NSC/CIC for two prototyping chips is also acknowledged.

References

[1] L. K. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1301–1308, Oct. 1989.

[2] R. C. Kim and S. U. Lee, "A VLSI architecture for a pel recursive motion estimation algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1291–1300, Oct. 1989.

[3] L. De Vos and M. Stegherr, "Parameterizable VLSI architectures for the full-search block matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1309–1316, Oct. 1989.

[4] K. M. Yang, M. T. Sun, and L. Wu, "A family of VLSI designs for the motion compensated block-matching algorithm," *IEEE Trans. Circuits Syst.*, vol. 36, pp. 1317–1325, Oct. 1989.

[5] C. H. Hsieh and T. P. Lin, "VLSI architecture for block-matching motion estimation algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 2, pp. 169–175, June 1992.

[6] S. I. Uramoto, A. Takabatake, M. Suzuki, H. Sakurai, and M. Yoshimoto, "A half-pel precision motion estimation processor for NTSC-resolution video," in *IEEE Custom Integrated Circuits Conf.*, San Diego, CA, May 9–12, 1993, pp. 11.2.1–11.2.4.

[7] J. Baek, S. Nam, M. Lee, C. Oh, and K. Hwang, "A fast array architecture for block matching algorithm," in *Proc. ISCAS'94*, London, U.K., May 30–June 2, 1994, pp. 4.211–4.214.

[8] C. L. Wang, K. M. Chen, and J. M. Hsiung, "A high-throughput and flexible VLSI architecture for motion estimation," in *Proc. ICASSP'95*, Detroit, MI, May 8–12, 1995, pp. 3295–3298.

[9] S. Chang, J. H. Hwang, and C. W. Jen, "Scalable array architecture design for full search block matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 332–343, Aug. 1995.

[10] H. Yeo and Y. H. Hu, "A novel modular systolic array architecture for full-search block matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 407–416, Oct. 1995.

[11] L. De Vos and M. Schobinger, "VLSI architecture for a flexible block matching processor," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 417–428, Oct. 1995.

[12] H. T. Kung, "Why systolic architectures," *IEEE Trans. Comput.*, vol. C-15, pp. 37–46, Jan. 1982.

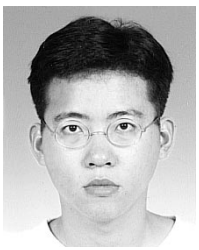
[13] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[14] K. Ishihara, S. Masuda, S. Hattori, H. Nishikawa, Y. Ajioka, T. Yamada, H. Amishiro, and M. Yoshimoto, "A half-pel precision MPEG2 motion estimation processor with concurrent three-vector search," in *Proc. ISSCC'95*, Feb. 1995, pp. 288–289.

[15] G. L. Tzeng and C. Y. Lee, "An efficient memory architecture for motion estimation processor design," in *Proc. ISCAS'95*, Seattle, WA, Apr. 29–May 3, 1995.

[16] M. C. Lu and C. Y. Lee, "Semi-systolic array based motion estimation processor design," in *Proc. ICASSP'95*, Detroit, MI, May 8–12, 1995.

[17] *International Organization for Standardization Coding of Moving Pictures and Associated Audio*, ISO/IEC JTC1/SC29/WG11/N702, 1994.



Yuan-Hau Yeh received the B.S. and M.S. degrees in electrical engineering from National Chiao Tung University, Hsinchu Taiwan, R.O.C., in 1993 and 1990, respectively, and is currently working toward the Ph.D. degree in electrical engineering at the same university.

His research interests include VLSI algorithms and architectures for multimedia systems (videos and graphics), and memory optimization for multimedia chips.

Chen-Yi Lee (S'89–M'90) received the B.S. degree from National Chiao Tung University, Hsinchu Taiwan, R.O.C., in 1982, and the M.S. and Ph.D. degrees from Katholieke University Leuven (KUL), Leuven, Belgium, in 1986 and 1990, respectively, all in electrical engineering.

From 1986 to 1990, he was with IMEC/VSDM, where he worked in the area of architecture synthesis for digital signal processing (DSP). In February 1991, he joined the faculty of the Electronics Engineering Department, National Chiao Tung University, where he is currently a Professor. His research interests mainly include VLSI algorithms and architectures for high-throughput DSP applications. He is also active in various aspects of very low bit coding, multimedia signal processing, high-speed networking, and system-on-chip design technology.