Global Elimination Algorithm and Architecture Design for Fast Block Matching Motion Estimation

Yu-Wen Huang, Shao-Yi Chien, Bing-Yu Hsieh, and Liang-Gee Chen

***Abstract*—This paper presents a new block matching motion estimation algorithm and its VLSI architecture design. The proposed global elimination algorithm (GEA) was derived from successive elimination algorithm (SEA), which can skip unnecessary sum of absolute difference (SAD) calculation by comparing minimum SAD with subsampled SAD (SSAD). Our basic idea is to separate the decision of early termination and SAD calculation for each candidate block to make data flow more regular and suitable for hardware. In short, we first compare the rough characteristics of all candidate blocks with the current block (SSAD). In turn, we select several best roughly matched candidate blocks to re-compare them with the current block by using detailed characteristics (SAD). Other features of GEA include fixed processing cycles, no initial guess, and high video quality (almost the same as full search). Unlike other fast algorithms, the mapping of GEA to hardware is very simple. We proposed an architecture that is composed of a systolic part to efficiently compute SSAD, an adder tree to support both SSAD and SAD calculations, and a comparator tree to avoid expensive sorting circuits. Simulation results show that our design is much more area efficient than many full-search architectures while maintaining high video quality and processing capability.**

***Index Terms*—Block matching, global elimination algorithm (GEA), motion estimation (ME), successive elimination algorithm (SEA).**

I. Introduction

MOTION-COMPENSATED transform coding has been adopted by all of the existing international standards related to video coding, such as the ISO MPEG series [1]–[3] and the ITU-T H.26x series [4]–[6]. Motion estimation (ME) removes temporal redundancy within frames and, thus, provides coding systems with high compression ratio. Since an ME module is usually the most computationally intensive part in a typical video encoder (50%–90% of the entire system), efficient implementation of ME is a must. A block matching approach is mostly selected as the ME module in video codecs and is adopted in the standards because of its simplicity and good performance. Among all the block matching algorithms, full-search block matching algorithm (FSBMA) is the most popular but demands the most computation. Despite the huge computation, e.g., 9.3 giga operations per second (GOPS) for CIF 30 Hz with $[-16, +15]$ search range, many parallel and pipelined VLSI architecture designs for FSBMA have been developed due to its regular data flow. In general, the processing capability of one-dimensional (1-D) systolic arrays is not high enough for real-time applications with large search range or large frame size, so the operating frequency has to be raised to hundreds of MHz. The processing capability of two-dimensional (2-D) systolic arrays is very high, but the number of processing elements (PE) is too large. As for the tree architecture, the main problem is the huge memory access. Thus, we decided to improve the ME design at the algorithmic level.

Many fast algorithms have been proposed to save the huge computation of FSBMA. One method is to reduce the search positions, such as three-step search [7], new three-step search [8], four-step search [9], 1-D full search [10], and diamond search [11]–[13]. Another way is to simplify the matching operations, such as pixel decimation [14], mini-max criterion [15], boundary match [16], and pixel truncation [17]. Nevertheless, these fast algorithms suffer for considerable peak signal-to-noise ratio (PSNR) degradation compared to FSBMA, especially when the motion field is large and complex.

Recently, fast full-search block matching algorithms, such as successive elimination algorithm (SEA) [18]–[20], partial distortion elimination (PDE) [21], and winner-update algorithm [22], have come into notice for their capabilities to reduce the heavy computation of FSBMA and to maintain the same results as FSBMA simultaneously. However, they were not developed with hardware considerations and are only suitable for software implementation. SEA avoids unnecessary sum of absolute difference (SAD) calculations by comparing an up-to-date minimum SAD with the absolute difference between the sum of pixels in current block and sum of pixels in candidate block, or namely subsampled SAD (SSAD). The total execution time will be reduced if many SAD calculations are skipped. Hence, it is critical for SEA to find good initial guesses of motion vectors (MVs) with smaller SADs in the early stage to increase the skipping ratio, but this is a difficult task for the regions where the motion field is not smooth. Moreover, the irregular data flow of SEA makes the systolic mapping to hardware a very tough task.

In this paper, we modified the SEA and proposed a new block matching algorithm called global elimination algorithm (GEA). In contrast to SEA, the conditional branch when a candidate block cannot satisfy the criterion of early termination is removed, the data flow is more regular, the processing cycles are fixed, and no initial guess is needed. Experimental results show that GEA can achieve almost the same results as FSBMA. Furthermore, the most important feature of GEA is

Manuscript received December 20, 2002; revised December 1, 2003. This paper was recommended by Associate Editor R. Chandramouli.

Y.-W. Huang and L.-G. Chen are with the DSP/IC Design Laboratory, Department of Electrical Engineering and Graduate Institute of Electronics Engineering, National Taiwan University, Taiwan, 10617 R.O.C (e-mail: yuwen@video.ee.ntu.edu.tw; lgchen@video.ee.ntu.edu.tw).

S.-Y. Chien was with the Graduate Institute of Electronics Engineering, National Taiwan University, Taiwan 10617, R.O.C. He is now with Quanta Computer Inc., Taoyuan, Taiwan, 10617 R.O.C. (e-mail: sherwin.chien@quantatw.com).

B.-Y. Hsieh is with MediaTek Inc., Hsinchu, Taiwan, 10617 R.O.C. (e-mail: BY_Hsieh@mtk.com.tw).

Digital Object Identifier 10.1109/TCSVT.2004.828321

that it is very easy to design. We also proposed an efficient VLSI architecture design for GEA. The core consists of a systolic part, a parallel adder tree, and a parallel comparator tree. Compared to many existing FSBMA architectures, our design is much more area-efficient.

The rest of the paper will be organized as follows. In Section II, we will review SEA and describe our GEA. Next, our GEA architecture will be presented in Section III. Section IV is the comparison of our design and FSBMA architectures. Finally, Section V gives a conclusion.

II. Algorithms

Usually, the block matching process is performed only on the luminance frame. Each luminance frame is divided into blocks of size $N \times N$, and each block in the current frame is matched with candidate blocks of size $N \times N$ within the search area in the reference frame. The best matched block has the lowest SAD among all of the candidate blocks. Instead of the original block, the displacement (motion vector) of the best matched block and the prediction residue will be transmitted to the decoder.

A. Full-Search Block Matching Algorithm

FSBMA can be described by the following equations:

$$\mathrm{SAD}(m,n) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |c(i,j) - s(i+m, j+n)| \quad (1)$$

$$\mathrm{MV} = \{(u,v) | \mathrm{SAD}(u,v) \le \mathrm{SAD}(m,n)\} \quad (2)$$

where $\mathrm{SAD}(m,n)$ denotes the SAD at search position $(m,n), (m,n)$ is in the range of $[-p, p-1], \{c(x,y) \,|\, 0 \le x \le N-1, 0 \le y \le N-1\}$ means current block data, $\{s(x,y) \,|\, -p \le x \le p+N-2, -p \le y \le p+N-2\}$ stands for search area data, the block size is $N \times N$, the search range is $[-p, p-1]$, and MV expresses the motion vector of current block with minimum SAD among $(2p)^2$ search positions.

B. SEA

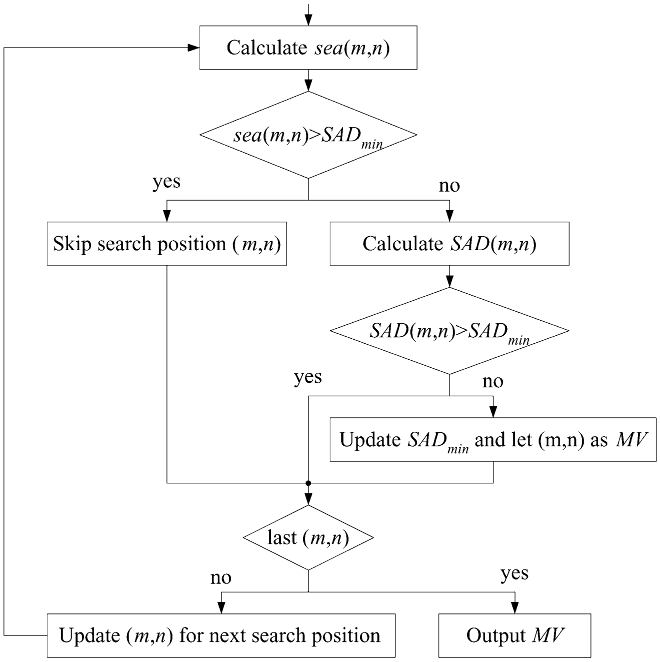

Calculate *sea*(*m*,*n*)
sea(*m*,*n*)>*SAD*$_{min}$
yes
no
Skip search position (*m*,*n*)
Calculate *SAD*(*m*,*n*)
SAD(*m*,*n*)>*SAD*$_{min}$
yes
no
Update *SAD*$_{min}$ and let (m,n) as *MV*
last (*m*,*n*)
no
yes
Update (*m*,*n*) for next search position
Output *MV*

Fig. 1. Flowchart of SEA.

The main idea of SEA [18] can be shown in the following inequality:

$$\begin{aligned}\mathrm{SAD}(m,n) &= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} |c(i,j) - s(i+m, j+n)| \\ &\ge \left| \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} c(i,j) - \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} s(i+m, j+n) \right| \\ &\equiv |K - \mathrm{SB}(m,n)| \\ &\equiv \mathrm{sea}(m,n)\end{aligned}$$

where $K \equiv \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} c(i,j)$, and $\mathrm{SB}(m,n) \equiv \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} s(i+m, j+n)$. To compute sea is much easier than to compute SAD due to the fact that K is the sum of pixels in current block and only has to be calculated one time, and the sum of pixels in candidate block $\mathrm{SB}(m,n)$ at search position (m,n) can be derived from the previous value of $\mathrm{SB}(m-1,n)$ as follows:

$$\begin{aligned}\mathrm{SB}(m,n) = \mathrm{SB}(m-1,n) &+ \sum_{a=0}^{N-1} s(m+N-1, n+a) \\ &- \sum_{b=0}^{N-1} s(m-1, n+b). \quad (3)\end{aligned}$$

If $\mathrm{sea}(m,n)$ is larger than current minimum SAD, denoted by $\mathrm{SAD}_{\mathrm{min}}$, it is guaranteed by inequality (3) that $\mathrm{SAD}(m,n)$ will be larger than $\mathrm{SAD}_{\mathrm{min}}$, and, thus, the search position (m,n) can be skipped. Otherwise, $\mathrm{SAD}(m,n)$ still needs to be calculated and compared with $\mathrm{SAD}_{\mathrm{min}}$. It is clear that a good initial guess of MV with small SAD is critical for SEA to increase the skipping ratio. Therefore, it is common to further speed up SEA with the use of MV predictors or spiral scan order of search positions. The flowchart of SEA is summarized in Fig. 1. A conditional branch exists after sea calculation for each search position, which makes the data flow irregular and unfavorable for hardware.

Another faster way to compute $\mathrm{SB}(m,n)$ is the frame-based method stated in [22]. An example with $N = 4$ is shown in Fig. 2. The first step is the horizontal scanning process on the original image, as shown in Fig. 2(a). A window of length N is placed at each position by raster scan order. The pixels in the window are summed, and the result is stored in the horizontally scanned image. Besides, efficient reuse of the previous result can be easily achieved, as shown in Fig. 2(b). The second step is the vertical scanning process on the horizontally scanned image, as shown in Fig. 2(c). The process is similar to the horizontal scanning, and the result is stored in the vertically scanned image.

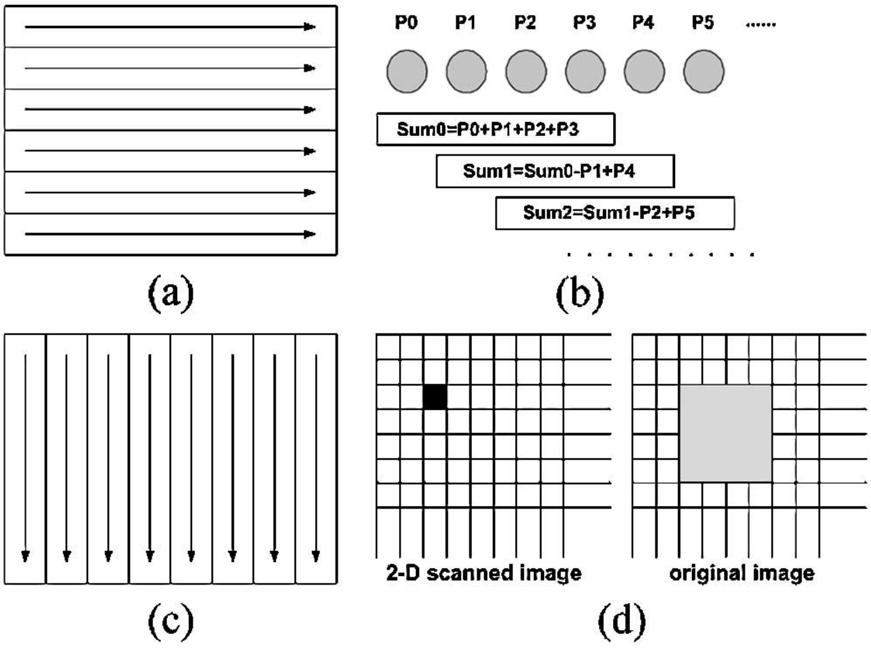


Fig. 2. Illustration of frame-based calculation for sum of 4 × 4 candidate block, $\mathrm{SB}(m,n)$. (a) Horizontal scanning process. (b) Example of data reuse for a row. (c) Vertical scanning process. (d) Marked pixel in the 2-D scanned image is equal to sum of pixels in the 4 × 4 block (painted in gray) in the original image.

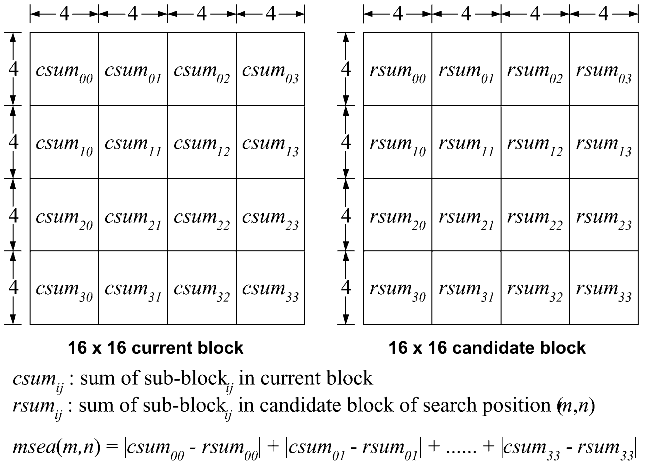
Eventually, each intensity value in the 2-D scanned image corresponds to the sum of pixels in a block of size $N \times N$, as illustrated in Fig. 2(d). Although the frame-based method is much faster, it requires a frame memory and is not suitable for hardware.

C. Multilevel SEA

Multilevel SEA (MSEA) [19], [20] changes the sea value in Fig. 1 to msea value. Let us modify inequality (3) to form the following inequality:

$$\begin{aligned}\mathrm{SAD}(m,n) &= \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}|c(i,j)-s(i+m,j+n)| \\ &\geq \sum_{q=0}^{L-1}|K_q-\mathrm{SB}_q(m,n)| \\ &\equiv m\,\mathrm{sea}(m,n) \\ &\geq \left|\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}c(i,j)-\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}s(i+m,j+n)\right| \\ &\equiv |K-\mathrm{SB}(m,n)| \\ &\equiv \mathrm{sea}(m,n). \end{aligned} \tag{4}$$

In inequality (4), a block of size $N \times N$ is divided into L nonoverlapping subblocks, K_q is the sum of the qth subblock in current block, and $\mathrm{SB}_q(m,n)$ is the sum of the qth subblock in candidate block at search position (m,n). For each search position, $m\,\mathrm{sea}(m,n)$ is calculated to decide if the SAD calculation can be skipped or not. When $L=1, m\,\mathrm{sea}$ is reduced to sea and is called $m\,\mathrm{sea}$ at level 1. If the subblocks are of the same size and $L=4,16,\ldots,$ it is called $m\,\mathrm{sea}$ at level $2,3,\ldots,$ respectively. Fig. 3 illustrates $m\,\mathrm{sea}$ at level 3 with $N=16$. In general, the skipping ratio of MSEA is higher than that of SEA, but to get the $m\,\mathrm{sea}$ value requires more computation than sea. In fact, both sea and msea can be regarded as SAD with reduced number of pixels. There is only one "macro-pixel" in a macroblock for SEA. For $N=16$ and MSEA at level 3, there are sixteen "macro-pixels" in a macroblock. In the rest of this paper, we denote sea and msea as SSAD (SSAD). For $N=16$, SSAD with Level $=1$ denotes sea, and SSAD with Level $=5$ is exactly the same as SAD.



$csum_{ij}$: sum of sub-block$_{ij}$ in current block
$rsum_{ij}$: sum of sub-block$_{ij}$ in candidate block of search position (m,n)

$msea(m,n) = |csum_{00} - rsum_{00}| + |csum_{01} - rsum_{01}| + \ldots\ldots + |csum_{33} - rsum_{33}|$

Fig. 3. Illustration of msea at level 3 with $N=16$.

D. GEA

The data flow of SEA or MSEA is not regular due to the conditional branches after SSAD calculation to determine the skipping process. This makes hardware design a tough task. Furthermore, when true MVs are beyond the search range, good initial guesses can never be found, and the skipping ratio could be so low that the processing time of SEA or MSEA is even longer than that of FSBMA. In

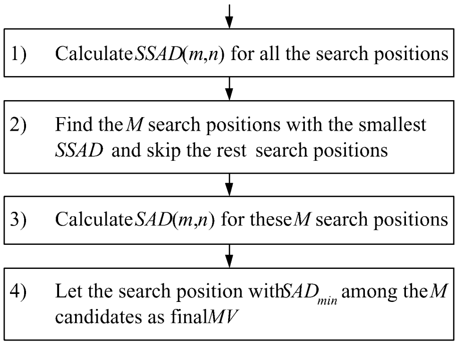

1) Calculate SSAD(m,n) for all the search positions
2) Find the M search positions with the smallest SSAD and skip the rest search positions
3) Calculate SAD(m,n) for these M search positions
4) Let the search position with SAD_{min} among the M candidates as final MV

Fig. 4. Flowchart of GEA.

our GEA, these problems can be solved. The steps of GEA are shown in Fig. 4 and can be described in (5). Note that $\{\mathrm{SSAD}_0, \mathrm{SSAD}_1, \mathrm{SSAD}_2, \ldots, \mathrm{SSAD}_M, \ldots, \mathrm{SSAD}_{4p^2-1}\}$ denotes the sorted result of $\mathrm{SSAD}(m,n)$.

$$\begin{aligned} \mathrm{MV} &= \{(u,v) \mid \mathrm{SAD}(u,v) \le \mathrm{SAD}(m_i,n_i), \quad 0 \le i \le M\}. \\ (m_i,n_i) &= \{(m,n) \mid \mathrm{SSAD}(m,n) < \mathrm{SSAD}_M\}. \\ \mathrm{SSAD}_0 &\le \mathrm{SSAD}_1 \le \mathrm{SSAD}_2 \le \cdots \le \mathrm{SSAD}_{4p^2-1}. \end{aligned} \tag{5}$$

Global elimination of search positions, which is described by step 2), is done after all SSAD values are calculated. No conditional branch is required within the calculation of SSAD at every search position. Consequently, the control of GEA is easier than SEA or MSEA. The result of GEA will be the same as FSBMA when the SSAD of the search position at which the smallest SAD is located is smaller than SSAD_M. Therefore, larger M makes the results more reliable while smaller M saves more computation. Generally speaking, the M parameter provides an easy tradeoff between speed and coding gain. No matter what M will be, the processing cycles can be fixed as long as M is determined, which is also a good feature for hardware scheduling. Moreover, the processing speed is independent of scan order. Raster scan, which is easier to implement than spiral scan, can be chosen for GEA without sacrifice of processing speed.

We use different values of M and Level under two typical cases. The first case has the following parameters: $N = 16$, QCIF frame size (176×144), $[-16, +15]$ search range. The parameters of the second case are $N = 16$, CIF frame size (352×288), $[-32, +31]$ search range. The results of Stefan are shown in Table I. As you can see, if we increase M and Level step by step, the video quality will become better and better. However, the improvement of video quality by increasing M and Level will stop when the video quality is already very close to full search. Further increase in M and Level may result in waste of computation. According to our experiments, we suggest to choose $M = 7$ and Level $= 3$ for QCIF and CIF size images.

TABLE I
AVERAGE PSNR OF COMPENSATED FRAMES IN DB AND STEFAN

Case I[a]				
	Level=1	*Level*=2	*Level*=3	*Level*=4
M=01	15.55	21.51	24.16	24.66
M=03	16.87	21.51	24.57	24.66
M=07	18.11	23.83	24.67	24.70
M=15	19.57	24.24	24.70	24.70
M=31	21.31	24.48	24.71	24.70
M=63	23.05	24.62	24.71	24.71

Case II[b]				
	Level=1	*Level*=2	*Level*=3	*Level*=4
M=01	14.95	21.33	25.21	25.72
M=03	16.31	21.33	25.60	25.72
M=07	17.34	24.60	25.71	25.73
M=15	18.41	25.13	25.75	25.73
M=31	19.73	25.44	25.75	25.73
M=63	21.43	25.62	25.74	25.73

[a] Case I: 16×16-block, QCIF, [-16, +15] search range.
[b] Case II: 16×16-block, CIF, [-32, +31] search range.

TABLE II
AVERAGE PSNR OF COMPENSATED FRAMES IN DB AND THE PERCENTAGE OF SAME MVS

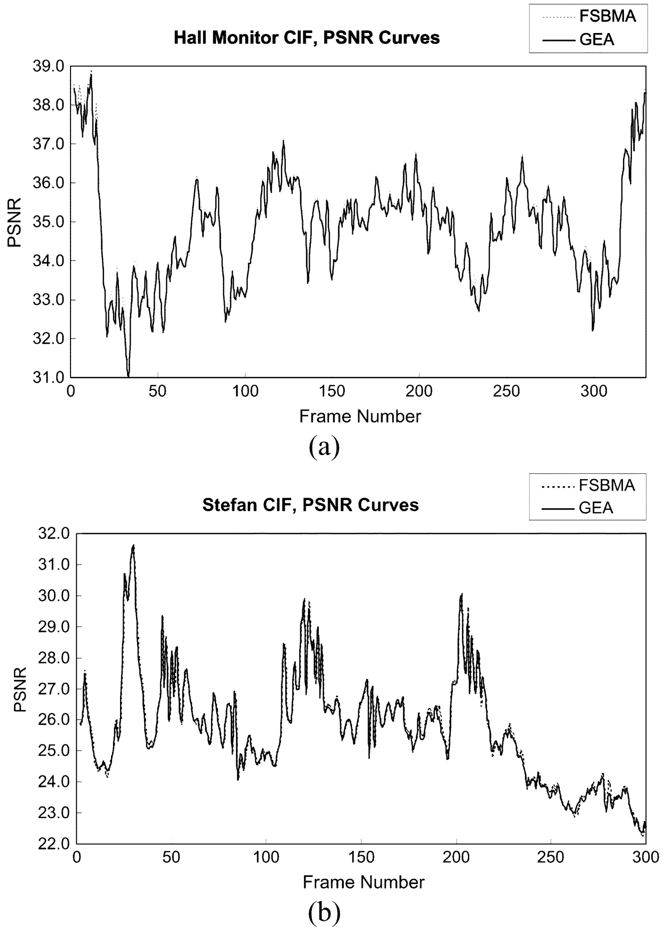
Case I[a]			
Video sequence	GEA	FSBMA	Same MV (%)
Coastguard	32.93	32.93	98.96
Container	43.11	43.11	98.49
Foreman	32.22	32.21	93.61
Hall monitor	32.97	32.98	94.62
Mobile calendar	26.15	26.15	99.59
Silent	35.16	35.14	96.92
Stefan	24.67	24.71	86.01
Table tennis	32.11	32.10	94.13
Weather	38.42	38.42	99.82

Case II[b]			
Video sequence	GEA	FSBMA	Same MV (%)
Coastguard	31.55	31.59	95.22
Container	38.53	38.53	93.83
Foreman	32.82	32.85	89.60
Hall monitor	34.82	34.90	73.96
Mobile calendar	25.16	25.20	95.91
Silent	36.11	36.12	93.36
Stefan	25.71	25.73	78.25
Table tennis	32.96	33.03	92.18
Weather	37.45	37.45	99.11

[a] Case I: 16×16-block, QCIF, [-16, +15] search range, *Level*=3, and *M*=7.
[b] Case II: 16×16-block, CIF, [-32, +31] search range, *Level*=3, and *M*=7.

E. Comparison and Analysis

After M and Level are selected, we compare GEA to FSBMA. Many standard sequences were tested. The results are shown in Table II. The average PSNR of motion compensated frames obtained by GEA is very close to FSBMA. Sometimes, GEA even results in a little higher PSNR values. Note that minimum SAD does not always lead to minimum mean square error, e.g., $1+9 < 5+6$, but $1^2+9^2 > 5^2+6^2$. The maximum PSNR drop is only 0.08 dB (Hall monitor in Case II). The percentages of the same MVs found by both GEA and FSBMA are also listed. It is shown that GEA and FSBMA usually produce the same MVs. For Hall monitor and Stefan in Case



(a)

(b)

Fig. 5. PSNR versus frame. (a) Hall monitor in Case II. (b) Stefan in Case II.

TABLE III
RunTime of Various ME Algorithms in Milliseconds

	Coastguard	Foreman	Stefan	Table	Avg.[a] per frame
FSBMA	83,537	112,385	83,548	84,157	280.58
SEA[b]	21,575	20,149	30,678	17,537	69.40
MSEA[c]	6,666	8,225	10,265	7,069	24.86
PDE[d]	16,903	24,532	24,723	17,565	64.60
WinUpdate[e]	13,270	16,671	15,406	13,226	45.20
GEA[f]	8,866	12,008	8,766	8,838	29.69
Diamond[g]	1,920	2,955	2,401	1,870	7.06

[a] QCIF frame size, *N*=16, *p*=16.
[b] Spiral scan and frame-based calculation of *SSAD*.
[c] Spiral scan, frame-based calculation of *SSAD*, and *Level*=3.
[d] Spiral scan and early stop of the *SAD* calculation. See ref. [21] for details.
[e] See ref. [22] for details.
[f] Raster scan, frame-based calculation of *SSAD*, *Level*=3, and *M*=7.
[g] See ref.[11] and [12] for details. May suffer considerable loss of quality.

II, the moving objects are small, and the motion is large and complex, which leads to lower percentages of the same MVs. Even so, the PSNR drop is still insignificant, as shown in Fig. 5. Therefore, we claim that although GEA does not guarantee the same results as FSBMA, GEA is very reliable.

Table III summarizes the runtime profiles of various ME algorithms. Except for GEA and Diamond, all the other fast algorithms are categorized as the fast full search, but note that GEA produces almost the same results while Diamond search suffers considerable PSNR degradation. The platform is a personal computer with 866-MHz CPU, and the program codes are written in C++. The speed of GEA is about the same as MSEA and is significantly faster than other fast full-search algorithms. Note that spiral scan of search positions is adopted for (M)SEA and PDE to achieve such fast speeds, while GEA uses raster scan. Besides, the speed of MSEA varies a lot for different scenes. For example, MSEA is faster than GEA on average. However, when the motion field is large and complex, e.g., in Stefan, the skipping ratio of MSEA is not high enough to defeat GEA. Diamond search, which is recommended by MPEG-4 due to its average good performance, is four times faster than GEA. However, it cannot guarantee the video quality. The PSNR drops of GEA compared to FSBMA are only 0, -0.01, 0.04, and -0.01 dB for the four listed sequences in Table III, respectively. As for Diamond search, the drops are 0.04, 0.49, 1.29, and 0.43 dB, which may cause up to 30% of increase in bit rate.

In sum, GEA regularizes the data flow of SEA by moving the SAD calculations behind all SSAD calculations. No initial guess or special scan order is required. The processing cycles are fixed. It is easy to make a tradeoff between the processing speed and the video quality by adjusting the value of M. GEA can produce almost the same results as FSBMA, and the speed of software implementation is about the same as spiral MSEA. Most important of all, the hardware mapping of GEA is very simple. We will show this in the next section.

III. Architecture Design Based on GEA

We will choose $N = 16, p = 16, \text{Level} = 3$, and $M = 7$ as an example to explain our architecture. The core of our architecture can be divided into three main parts: the systolic part, the parallel adder tree, and the parallel comparator tree.

A. Systolic Part

The data flow of the systolic part, which is to calculate the sum of pixels in a 4 × 4 subblock, is shown in Fig. 6. Note that $c_{l,k}$ and $s_{l,k}$ denotes $c(k,l)$ and $s(k,l)$ in (1), respectively, and the rectangles stand for shift registers. A column of block data is loaded in parallel at each clock cycle, denoted by t. When $t = 0 - 15, c$ data are loaded. The sum of pixels of each subblock in current block, depicted as $c\,\text{sum}_{00} - c\,\text{sum}_{33}$ in Fig. 3, is computed in parallel at $t = 15$ and stored into registers. Next, s data are loaded column by column. When $t = 16$–62, s data of search positions $(-16,-16)$–$(+15,-16)$ are loaded, the sum of pixels of each subblock in candidate block, denoted as $r\,\text{sum}_{00} - r\,\text{sum}_{33}$ in Fig. 3, will be available at $t = 31$–62 for search positions $(-16,-16)$–$(+15,-16)$, respectively. Then, columns of s data for search positions $(-16,-15)$–$(+15,-15)$ are loaded in parallel at $t = 63$–109 and vice versa. Thus, it takes $N + 2p(2p + N - 1)$ clock cycles for a macroblock to load all the data into the systolic part.

B. Parallel Adder Tree

The first purpose of the parallel adder tree is to compute SSAD. In Fig. 7, ADxx takes charge of the computa-

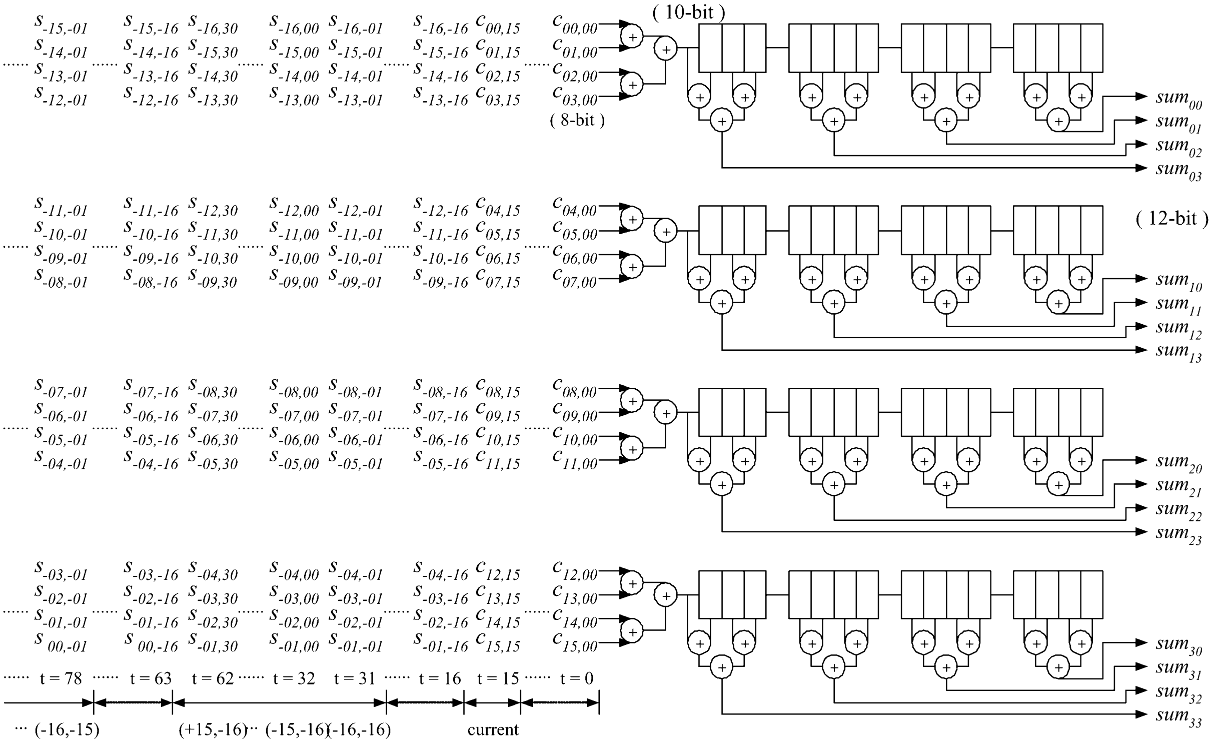

s-15,-01 s-15,-16 s-16,30 s-16,00 s-16,-01 s-16,-16 c00,15 c00,00
s-14,-01 s-14,-16 s-15,30 s-15,00 s-15,-01 s-15,-16 c01,15 c01,00
s-13,-01 s-13,-16 s-14,30 s-14,00 s-14,-01 s-14,-16 c02,15 c02,00
s-12,-01 s-12,-16 s-13,30 s-13,00 s-13,-01 s-13,-16 c03,15 c03,00
(10-bit)
(8-bit)
sum00
sum01
sum02
sum03
s-11,-01 s-11,-16 s-12,30 s-12,00 s-12,-01 s-12,-16 c04,15 c04,00
s-10,-01 s-10,-16 s-11,30 s-11,00 s-11,-01 s-11,-16 c05,15 c05,00
s-09,-01 s-09,-16 s-10,30 s-10,00 s-10,-01 s-10,-16 c06,15 c06,00
s-08,-01 s-08,-16 s-09,30 s-09,00 s-09,-01 s-09,-16 c07,15 c07,00
(12-bit)
sum10
sum11
sum12
sum13
s-07,-01 s-07,-16 s-08,30 s-08,00 s-08,-01 s-08,-16 c08,15 c08,00
s-06,-01 s-06,-16 s-07,30 s-07,00 s-07,-01 s-07,-16 c09,15 c09,00
s-05,-01 s-05,-16 s-06,30 s-06,00 s-06,-01 s-06,-16 c10,15 c10,00
s-04,-01 s-04,-16 s-05,30 s-05,00 s-05,-01 s-05,-16 c11,15 c11,00
sum20
sum21
sum22
sum23
s-03,-01 s-03,-16 s-04,30 s-04,00 s-04,-01 s-04,-16 c12,15 c12,00
s-02,-01 s-02,-16 s-03,30 s-03,00 s-03,-01 s-03,-16 c13,15 c13,00
s-01,-01 s-01,-16 s-02,30 s-02,00 s-02,-01 s-02,-16 c14,15 c14,00
s 00,-01 s 00,-16 s-01,30 s-01,00 s-01,-01 s-01,-16 c15,15 c15,00
sum30
sum31
sum32
sum33
t = 78 t = 63 t = 62 t = 32 t = 31 t = 16 t = 15 t = 0
(-16,-15) (+15,-16) (-15,-16) (-16,-16) current

Fig. 6. Data flow of systolic part. Columns of block data are loaded in parallel, and the sums of subblocks sum_{00}–sum_{33} are computed in parallel as well.

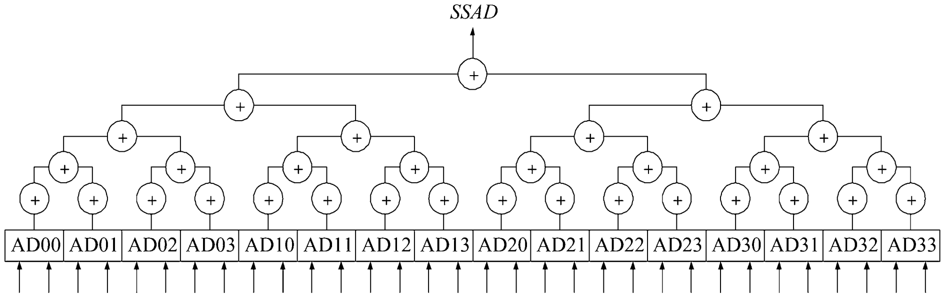

SSAD
AD00 AD01 AD02 AD03 AD10 AD11 AD12 AD13 AD20 AD21 AD22 AD23 AD30 AD31 AD32 AD33

Fig. 7. Parallel adder tree calculates the $\mathrm{SSAD(msea)}$ value shown in Fig. 3. It can be reused to compute SAD of the M candidate blocks having smallest SSADs.

tion of absolute difference between $c\,\mathrm{sum}_{xx}$ and $r\,\mathrm{sum}_{xx}$ denoted in Fig. 3. The parallel adder tree adds the results from AD00–AD33 and outputs SSAD value. The second purpose is to compute SAD after the M candidate blocks have been found by the parallel comparator tree, which will be described in the next subsection. SAD of 16 pixels can be calculated in one clock cycle, and the partial results are accumulated in another register. Thus, it requires N clock cycles to compute the SAD of a candidate block.

C. Parallel Comparator Tree

The objective of parallel comparator tree is to find the M search positions that have the smallest SSAD values. The idea is to save the up-to-date M smallest SSAD values and their corresponding MVs in the registers. In turn, we compare the new coming SSAD of the next search position with the M stored values and find the maximum among them. If the new coming SSAD is larger than the M stored values, nothing will happen. If the maximum is one of the stored SSAD values, the largest one and its corresponding MV will be replaced by the new SSAD and the new MV. If more than one of the stored SSAD values are equal to the maximum, only one of them and its MV will be replaced. In this way, we do not need sorting circuits, which is very expensive in chip area, to find the M smallest SSAD values.

The comparator tree is divided into a forward part, as shown in Fig. 8(a), and two feedback parts, as shown in Fig. 8(b) and (c). Note that the symbols with "_reg" are registers. In Fig. 8(a), SSAD1_reg-SSAD7_reg should be properly initialized as 0xFFFF before the first valid SSAD value comes from parallel adder tree to SSAD_in_reg. The maximum value among the eight SSAD values, denoted as $\mathrm{SSAD\_max}$, is computed by the forward part. In Fig. 8(b), EQUx compares whether SSADx_reg equals to $\mathrm{SSAD\_max}$ or not. If SSADx_reg equals to $\mathrm{SSAD\_max}$, EQUx will be activated; otherwise, it will be inactivated. The CHECK module functions as follows. If no EQU is active, no $\mathrm{replace}_x$ will be active. If only one EQUx is active, only the $\mathrm{replace}_x$ will be active. If more than one

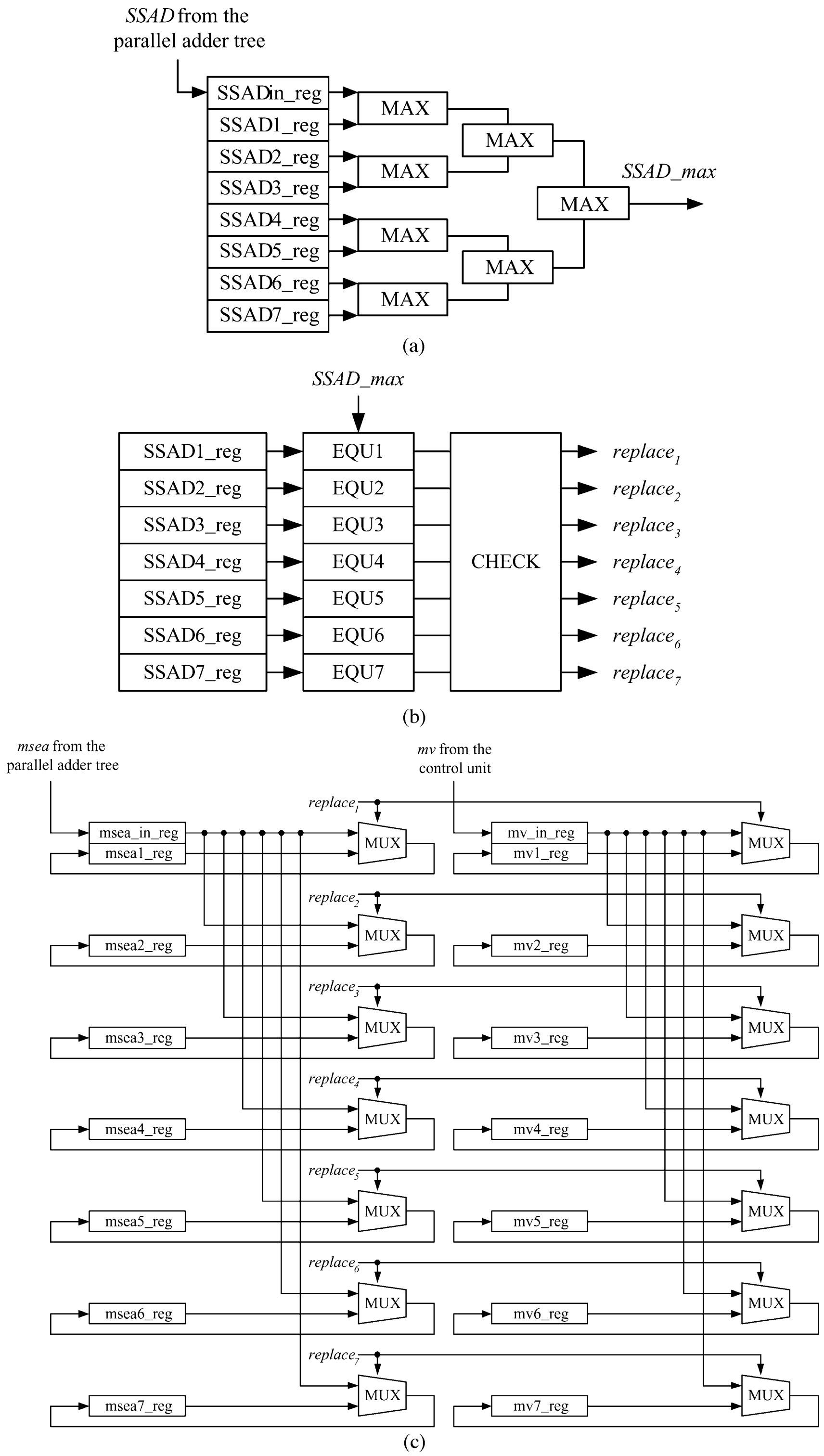


Fig. 8. Parallel comparator tree. (a) Forward part to compute $\mathrm{SSAD\_max}$. (b) One feedback part to find the target SSADx_reg that will be replaced. (c) The other feedback part to replace the SSADx_reg and mvx_reg with SSAD_in_reg and mv_in_reg, respectively.

EQU units are active, only one of the corresponding replace_x signal can be activated. For instance, when EQU1 and EQU2 are both active, only replace1 will be activated. In Fig. 8(c), if replace_x is active, SSADx_reg and mvx_reg will be replaced by the current values in SSAD_in_reg and mv_in_reg, respectively. Moreover, as shown in Fig. 6, invalid SSAD values are generated periodically during the first $N-1$ cycles of loading s data on every row of the search positions. At these cycles, the SSADs inputted to the parallel comparator tree must be replaced by 0xFFFF. In sum, the parallel comparator tree is to keep the M smallest SSAD values and their MVs in SSADx_reg and mvx_reg registers, respectively.

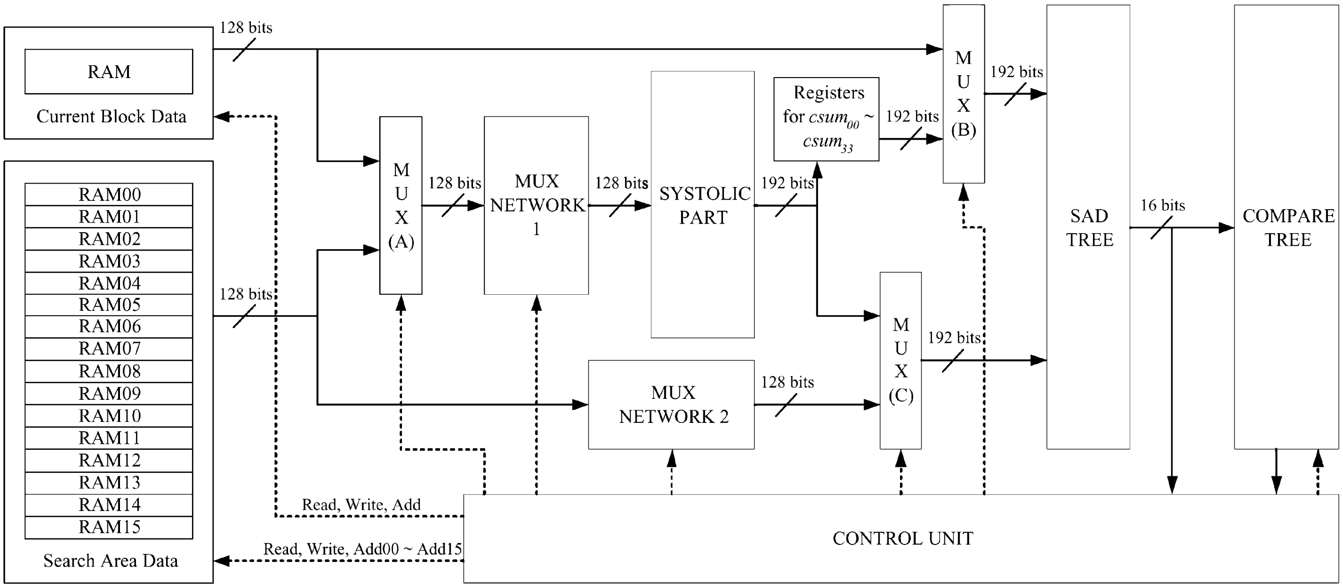


Fig. 9. Overall architecture design for GEA.

TABLE IV
COMPARISON OF VARIOUS ME ARCHITECTURES WITH $N = 16, p = 16, \text{Level} = 3$, and $M = 7$

Architecture[a]	Description	Memory width (bits)	Clock cycles (per *MV*)	Required freq.[b] (MHz)	Gate count (K gates)
[23] Yang	1-D semi-systolic	24	8,192	97.32	44.7
[24] AB1	1-D systolic	256	24,064	285.88	14.4
[24] AB2	2-D systolic	128	1,504	17.87	98.2
[25] Hsieh*	2-D systolic	8	2,209	26.24	100.1
[26] Tree	Tree structure	4,096	1,024	12.17	58.7
[27] Yeo	2-D semi-systolic	24	256	3.04	436.6
[28] Lai	1-D semi-systolic	24	256	3.04	384.8
[29] SA*	2-D systolic	16	1,024	12.17	127.0
[29] SSA*	2-D semi-systolic	16	1,024	12.17	110.6
Ours	Based on GEA	256	1,635	19.42	23.1

[a] For the architectures labeled with *, many shift registers are still needed and were not included in the implementation.
[b] Required frequency for CIF 30 Hz. The critical path constraint is set to just meet this target speed.

D. Overall Architecture Design

Fig. 9 presents the block diagram of the complete GEA architecture. Besides the three core modules mentioned above (SYSTOLIC PART, SAD TREE, and COMPARE TREE), there are still some control circuits (CONTROL UNIT, MUX, and MUX NETWORK $1/2$) and memory modules to store current block data and search area data. Note that the solid arrows denote the data signals, and the dotted arrows denote the control signals. In order to output a column of block data in parallel, 16 on-chip SRAM modules are required.

In brief, the architecture design of GEA is very simple. The proposed hardware requires $N + 2p(2p + N - 1)$ clock cycles to find the M search positions with the smallest SSAD values, and MN clock cycles are needed for the SAD calculation of the M candidate blocks. Therefore, our GEA architecture in total spends $N + 2p(2p + N - 1) + MN$ clock cycles to obtain the MV of a macroblock. If we want to eliminate the bubble cycles of SSAD calculations during the beginning $(N - 1)$ cycles at change of rows, we can use ping-pong mode systolic registers to store sums of 1×4 pixels. In this way, 32 on-chip SRAM modules are required to output the 32 pixels in parallel, and the number of total cycles required for a macroblock is reduced to $N + (2p)(2p) + MN$.

IV. ARCHITECTURE COMPARISON, DISCUSSIONS, AND CHIP IMPLEMENTATION

A. Comparison Between GEA Architecture and FSBMA Architectures

In this subsection, we compare our design with many architectures based on FSBMA. The results of two typical cases are shown in Tables IV and V. Only the PE arrays of the FSBMA architectures were implemented for comparison because the control unit only occupies a very small part of the whole design. In Tables IV and V, architectures labeled with "*" symbol need many additional shift registers to reuse data, but these registers were not implemented. The real gate counts of these designs would be higher than the reported results. As for our GEA architecture, all the parts in Fig. 9 except for RAM modules were included in the simulation. Note that all designs have to save current block data and search area data, so the required number of bits for on-chip SRAM is the same for all designs.

TABLE V
COMPARISON OF VARIOUS ME ARCHITECTURES WITH $N = 16, p = 32, \text{Level} = 3$, AND $M = 7$

Architecture[a]	Description	Memory width	Clock cycles	Required freq.[b]	Gate count
		(bits)	(per MV)	(MHz)	(K gates)
[23] Yang	1-D semi-systolic	24	16,384	194.64	107.6[c]
[24] AB1	1-D systolic	256	80,896	961.04	16.5[d]
[24] AB2	2-D systolic	128	5,056	60.07	107.9
[25] Hsieh*	2-D systolic	8	6,241	74.14	100.0
[26] Tree	Tree structure	4,096	4,096	48.66	58.4
[27] Yeo	2-D semi-systolic	24	256	3.04	1746.4
[28] Lai	1-D semi-systolic	24	256	3.04	1539.3
[29] SA*	2-D systolic	16	4,096	48.66	127.0
[29] SSA*	2-D semi-systolic	16	4,096	48.66	110.6
Ours	Based on GEA	256	5,187	61.62	33.2

[a] For the architectures labeled with *, many shift registers are still needed and were not included in the implementation.
[b] Required frequency for CIF 30 Hz. The critical path constraint is set to just meet this target speed.
[c] Only 119.33MHz can be achieved.
[d] Only 178.57MHz can be achieved.

The VLSI circuits were described in Verilog HDL and synthesized by SYNOPSYS Design Analyzer using AVANT! 0.35-mu m cell library. In order to be fair, the comparison was made under the same throughput of MVs (number of estimated MVs per second). Consequently, we first estimated the required operating frequency of each architecture for the real-time application of 30 CIF f/s. In turn, proper pipeline registers were added and the critical path constraints were set to just meet the target speeds.

In general, the processing capability of the 1-D arrays is not high enough. The operating frequency for applications with large frame size and large search range has to be substantially increased to an unacceptable extent. The processing capability of the 2-D arrays is very high, but the enormous gate count makes them expensive. The architecture in [28] belongs to 1-D array, but it adopts data interlacing and 2-D data reuse, which also requires too large of an area. The tree architecture has good performance in area and speed, but the required memory bit width is too large, which is not feasible. As for the proposed GEA architecture, the processing speed is only a little slower than 2-D arrays, but the gate count of GEA is much smaller. The 2-D array architecture in [25] is even slower than ours. The processing capabilities of the 1-D arrays are much lower than that of GEA, and the gate count of the architecture in [23] is even larger than that of GEA. Therefore, GEA has superior area–speed performance than FSBMA architectures.

B. Chip Implementation

The proposed GEA architecture was verified by the VLSI implementation. The target speed of the chip is real-time processing for 30 CIF or 120 QCIF f/s with block of size $16 \times 16, [-16, +15]$ search range, $\text{Level} = 3$, and $M = 7$. If the ME module constantly access the external memory of the video coding system, the system bus will become too busy. So we use on-chip SRAM to store current block data and search area data. The amount of the external memory access can be significantly reduced in this way, although the ME module needs some extra clock cycles to load the data. Data reuse of overlapped search area between horizontally adjacent macroblocks is also adopted to reduce the loading cycles. The required working frequency is 21.2016 MHz. The chip photo is shown in Fig. 10. We store search area data in 16 144×8-SRAM modules, as the RAM00–RAM15 organized in Fig. 9. Four 16×32-SRAM modules controlled by the same address generator are used to store current block data. According to testing results, the chip works well for all test patterns, and the maximum working frequency is 27.8 MHz, which is high enough to support MPEG-4 Simple Profile Level 3 (CIF 30 Hz). The other chip specifications are shown in Table VI. Because of the fixed processing cycles, low requirement of bandwidth on system bus, low gate count, high processing capability, and high video quality, our design is a good silicon IP core and can be easily integrated in video coding systems.

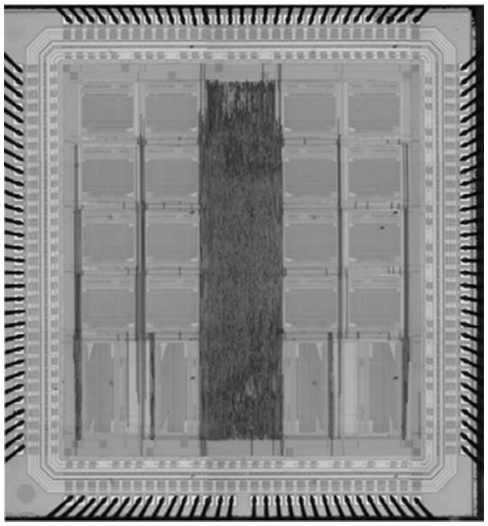


Fig. 10. Chip photo.

TABLE VI
CHIP SPECIFICATIONS

Process	TSMC 1P4M 0.35um
Area	3,679um×4,001um
Package	128 CQFP
Memory	24,080 bits
Transistor count	357,551
Maximum frequency	27.8 MHz
Power consumption	189mW @ 25.0 MHz
Search range	[-16, +15]
Processing capability	38 CIF or 152 QCIF fps

V. Conclusion

We proposed GEA and its architecture design for fast block matching ME. Our algorithm can solve several problems encountered by SEA. The main concept is to first roughly compare candidate blocks with current block, and then to precisely compare the best roughly matched candidate blocks with current block. Unlike other fast algorithms, GEA is hardware oriented. No initial guess or special scan order of search positions is required, the data flow is regular, the processing cycles are fixed, and the motion-compensated results are almost the same as those obtained by the FSBMA. We also proposed a VLSI architecture for GEA. It mainly includes a systolic part, a parallel adder tree, and a parallel comparator tree. For typical applications, the area–speed performance of our design is much better than many existing architectures based on the FSBMA.

Acknowledgment

The authors would like to thank Dr. T.-C. Wang and W.-M. Chao for discussion and many valuable suggestions.

References

[1] *Information Technology—Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to About 1.5 Mbit/s—Part II: Video*, ISO/IEC 11 172-2, 1993.

[2] *Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video*, ISO/IEC 13 818-2 and ITU-T Recommendation H.262, 1996.

[3] *Information Technology—Coding of Audio-Visual Objects—Part 2: Visual*, ISO/IEC 14 496-2, 1999.

[4] *Video Codec for Audiovisual Services at p × 64 Kbit/s*, ITU-T Recommendation H.261, Mar. 1993.

[5] *Video Coding for Low Bit Rate Communication*, ITU-T Recommendation H.263, Feb. 1998.

[6] *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification*, ITU-T Recommendation H.264 and ISO/IEC 14 496-10 AVC, Joint Video Team, May 2003.

[7] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *Proc. Nat. Telecommunications Conf.*, 1981, pp. C9.6.1–C9.6.5.

[8] R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 438–442, Aug. 1994.

[9] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, pp. 313–317, June 1996.

[10] M. J. Chen, L. G. Chen, and T. D. Chiueh, "One-dimensional full search motion estimation algorithm for video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 504–509, June 1994.

[11] S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp. 287–290, Feb. 2000.

[12] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 369–377, Aug. 1998.

[13] A. M. Tourapis, O. C. Au, M. L. Liou, G. Shen, and I. Ahmad, "Optimizing the MPEG-4 encoder—Advanced diamond zonal search," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS'00)*, 2000, pp. 674–677.

[14] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, pp. 148–157, Apr. 1993.

[15] M. J. Chen, L. G. Chen, T. D. Chiueh, and Y. P. Lee, "A new block matching criterion for motion estimation and its implementation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 5, pp. 231–236, June 1995.

[16] M. J. Chen, "Predictive motion estimation algorithms for video compression," *J. St. John's & St. Mary Inst. Technol.*, vol. 15, pp. 197–214, June 1997.

[17] Z. L. He, C. Y. Tsui, K. K. Chan, and M. L. Liou, "Low-power VLSI design for motion estimation using adaptive pixel truncation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 10, pp. 669–678, Aug. 2000.

[18] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. Image Processing*, vol. 4, pp. 105–107, Jan. 1995.

[19] X. Q. Gao, C. J. Duanmu, and C. R. Zou, "A multilevel successive elimination algorithm for block matching motion estimation," *IEEE Trans. Image Processing*, vol. 9, pp. 501–504, Mar. 2000.

[20] M. Brünig and W. Niehsen, "Fast full-search block matching," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 11, pp. 241–247, Feb. 2001.

[21] Digital Video Coding Group, *ITU-T Recommendation H.263 Software Implementation*, Telenor R&D, 1995.

[22] Y. S. Chen, Y. P. Huang, and C. S. Fuh, "Fast block matching algorithm based on the winner-update strategy," *IEEE Trans. Image Processing*, vol. 10, pp. 1212–1222, Aug. 2001.

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

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

[25] 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.

[26] Y. S. Jehng, L. G. Chen, and T. D. Chiueh, "An efficient and simple VLSI tree architecture for motion estimation algorithms," *IEEE Trans. Signal Processing*, vol. 41, pp. 889–900, Feb. 1993.

[27] 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.

[28] Y. K. Lai and L. G. Chen, "A data-interlacing architecture with two-dimensional data-reuse for full-search block matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, pp. 124–127, Apr. 1998.

[29] Y. H. Yeh and C. Y. Lee, "Cost-effective VLSI architectures and buffer size optimization for full-search block matching algorithms," *IEEE Trans. VLSI Syst.*, vol. 7, pp. 345–358, Sept. 1999.