A Low-Cost Gray Prediction Search Chip for Motion Estimation

Jer Min Jou, Yeu-Horng Shiau, Pei-Yin Chen, and Shiann-Rong Kuang

***Abstract*—Taking advantage of the prediction ability provided by gray system theory, the gray prediction search (GrPS) algorithm can determine the motion vectors of image blocks correctly and quickly [6]. A dedicated GrPS chip, which is with low cost and has the properties of regular-data-flow computations, is proposed in this paper to support the MPEG video resolution in real time. With 0.6-μm CMOS technology, the proposed chip needs a die size of 2.8×2.9 mm^2 with about 54-K transistors, and can work with a clock rate of 66 MHz. Since GrPS performs better than other fast search algorithms, such as TSS, CS, PHODS, FSS, and SES, this low-cost GrPS chip is a good candidate for real-time motion estimation.**

***Index Terms*—Block motion estimation, gray prediction search, motion vector.**

I. Introduction

In RECENT years, several video compression standards, such as H.263 and MPEGs, had been proposed for different applications. One common feature of these standards is that they use DCT transform coding to reduce spatial redundancy in an image frame and block motion estimation/compensation to reduce the temporal redundancy among image frames. Generally, the encoding complexity of these video standards is dominated by the motion estimation, if full search (FS) is used as the block-matching algorithm (BMA). FS exhaustively searches all possible displaced candidate blocks within the search area in the previous frame, in order to find the block with the minimum distortion and then the best motion vector. Massive computation is, therefore, required in the implementation of FS. As a result, many fast and efficient BMAs such as the three-step search algorithm (TSS) [1], the cross-search algorithm (CS) [2], the parallel hierarchical one-dimensional search algorithm (PHODS) [3], the four-step search algorithm (FSS) [4], and the simple and efficient search algorithm (SES) [5] have been developed to reduce the computational complexity. Even when fast BMAs are applied, the computations for video systems with higher data rates are still massive. Consequently, the development of application-specific VLSI architectures is necessary for real-time motion estimation.

In general, two main considerations: coding performance and VLSI implementation are used to evaluate a BMA for time-critical applications. The coding performance of a BMA depends heavily on the accuracy, speed and effectiveness of motion estimation. However, the characteristics of various image sequences bear a lot of uncertainty and are hard to be extracted, so it is not easy to devise a good estimation method that always provides accurate and fast motion estimation for different types of image sequences. In [6], we have proposed an efficient gray prediction search (GrPS) algorithm which performs better than other search algorithms, such as TSS [1], CS [2], PHODS [3], FSS [4], and SES [5].

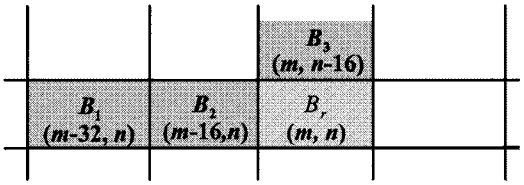

Fig. 1. Position–relation of the reference block and three adjacent blocks.

In addition, the primary criterion to choose a proper BMA suitable for hardware realizations is that it must comprise regular data flow and lower hardware cost. Some previous BMA implementations [7]–[9] concern FS due to its regular data flow and low control overhead. However, because of the inherent high complexity of FS, high-speed motion estimators can only be provided by large arrays of processing elements (PEs). To achieve high performance and real-time necessity, the dedicated designs for realizing the TSS are attractive for the purpose of low-cost implementation to various video applications. Some parallel architectures for TSS block-matching algorithm were developed in [10] for low bit-rate video and HDTV systems. For supporting real-time videoconferencing standards, Costa *et al*. proposed a TSS-based motion estimation unit of 3-PE realization [11]. A real-time TSS-based motion estimation LSI for MPEG2 was proposed in [12]. In [13], a joint algorithm-architecture design of a programmable motion estimator chip was developed. To support NTSC resolution video, Chen proposed a cost-effective TSS block-matching chip [14]. However, the costs of respective hardware described above are still higher, and their coding performance is also an issue.

In this paper, we present a dedicated low-cost GrPS chip with 3-PE architecture to provide suitable solutions for MPEG2 ITU-R601 (720 pixels $\times$ 480 lines, 30 frames/s) in real time. The chip consists of four memory banks, four address generators, and the 3-PE motion vector generator (MVG). With 0.6-μm CMOS technology, the proposed chip needs a die size of 2.8×2.9 mm^2 with 54K transistors. Since GrPS performs better than other fast search algorithms [6], this low-cost GrPS chip is a good candidate for real-time motion estimation.

Manuscript received September 6, 1999; revised December 20, 2001. This work was supported in part by the National Science Council, R.O.C, under Grant NSC-90–2218-E-006–047. This paper was recommended by Associate Editor R. Sridhar.

The authors are with the Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan, R.O.C.

Publisher Item Identifier 10.1109/TCSI.2002.800468.

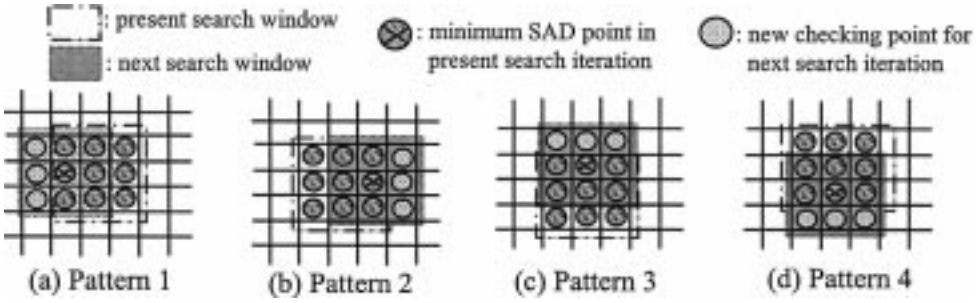

Fig. 2. Search patterns for the next search iteration when the minimum SAD point is located at the middle of the boundary column (row) of the present search window. (a) Pattern 1. (b) Pattern 2. (c) Pattern 3. (d) Pattern 4.

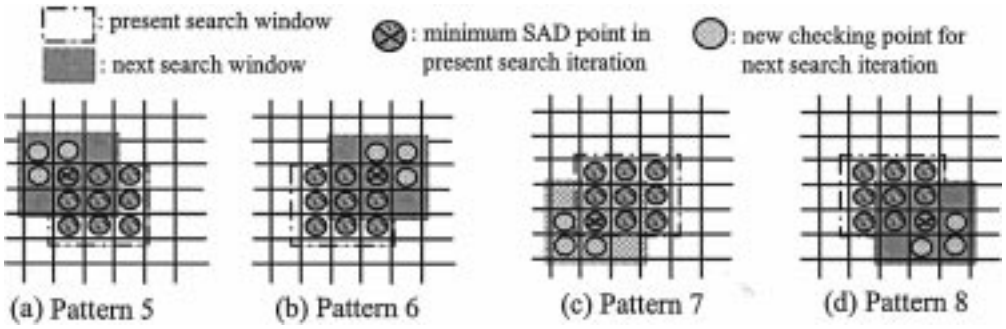

Fig. 3. Search patterns for the next search iteration when the minimum SAD point is located at the corner of the present search window. (a) Pattern 5. (b) Pattern 6. (c) Pattern 7. (d) Pattern 8.

The rest of this paper is organized as follows. In Section II, some background information on the block-matching algorithm and gray prediction is summarized first. Then, the GrPS algorithm and implementation issues for GrPS chip are described. In Section III, we describe the system architecture for GrPS. The VLSI implementation is presented in Section IV. Finally, Section V gives some conclusions.

II. Block-Matching Algorithm

In this section, the concepts of BMA are introduced first. Then, the GrPS algorithm is briefly reviewed.

A. An Overview of Basic Principles of Block-Matching Algorithm

In the typical BMA, an image frame is divided into nonoverlapped blocks of $N \times N$ pixels. Then for a maximum motion displacement of w pixels per frame, the block of pixels in the current frame (called a reference block B_r) is compared with the corresponding blocks (called the candidate blocks) within a search area of size $(N+2w) \times (N+2w)$ pixels in the previous frame. When the best-matched (or lowest-distortion) candidate block is found, the motion vector, representing the coordinate difference between B_r and the best-matched candidate block, is recorded. Consequently, the motion vector and the prediction error between B_r and the best-matched candidate block can be coded and transmitted instead of the B_r; therefore, temporal redundancy among image frames is removed. The better the prediction method, the smaller the prediction error, and hence the transmission bit rate.

The block matching process is performed on the basis of minimum distortion. We adopt SAD, sum of absolute difference between the pixel values of any two blocks, as the block distortion measure. Assume $B_r(m,n)$ is the reference block of size $N \times N$ pixels whose the upper most left pixel is at the location (m,n) of the current frame, and $C(m+u, n+v)$ is a candidate block within the search area of the previous frame with (u,v) displacement from B_r. Let w be the maximum motion displacement and $P_t(m,n)$ be pixel value at location (m,n) in frame t, SAD between B_r (in frame t) and C (in frame $t-1$) is defined as shown in (1) at the bottom of the page. The SAD is computed for each candidate block within the search area. A block with the minimum SAD is considered the best-matched block, and the value (u,v) for the best-matched block is called motion vector. That is, motion vector MV is given by

$$\mathrm{MV} = (\mathrm{u}, \mathrm{v})|_{\min \mathrm{SAD(u,v)}}.$$

$$\mathrm{SAD}(u,v) = \sum_{j=0}^{N-1} \sum_{i=0}^{N-1} |p_t(m+i, n+j) - p_{t-1}(m+i+u, n+j+v)| \quad \text{where } -w \le u, v \le w \tag{1}$$

```
/* Input image frame size: 720 (width), 480 (height); Block size: 16×16;
   Each frame is divided into (720/16)×(480/16) = 45×30 =1350 blocks;          */

INPUT:  block index (bx, by) for Br   0 ≤ bx ≤ 44, 0 ≤ by ≤ 29.
OUTPUT: motion vector (mxr, myr)      −7 ≤ mxr, myr ≤ 7.

GrPS(bx, by)
{  Get the three adjacent blocks' motion vectors by using (bx, by);
L1:  Use the three motion vectors to determine mxr and myr according to gray prediction [6];
       i=0;          /* initial values */

L2:    While (i<=7)                          /* start of the search iteration */
       { dx=0; dy=0;                         /* (dx,dy): displacement of current search iteration */
         if (i=1) find the minimum SAD point in the nine points of the present 3×3 search window;
         else{ find the minimum SAD point in the present search window by adding three
               checking points according to its search pattern listed in Fig. 2(a) to Fig.3(d);   }
         Use the location of the minimum SAD point to determine dx and dy.
         if (dx=0 AND dy=0) break;           /* terminating the entirely search process */
         mxr = mxr + dx; myr = myr + dy;     /* accumulating the motion vector */
         Move to the next search window according to dx and dy.
         i++;          }                     /* end of the search iteration */
     return (mxr, myr);
}
```

Fig. 4. Algorithm of GrPS.

B. Review of the Gray Prediction Search Algorithm

The gray theory, applicable to the prediction problem of a time-varying nonlinear system, was first proposed by Deng in 1982 [16], and has been successfully and widely used in many fields [16], [17]. In GrPS [6], a kind of single variable and first-order linear dynamic gray model, named as GM(1,1) [15], is adopted to perform the prediction job.

GrPS contains two main phases: determining the initial search center and finding the MV within the search area. In the first phase, GrPS finds each block's predicted motion vector based on gray model and then uses the predicted vector to determine the initial search center. With the help of gray model prediction, the predicted motion vector is determined by using three adjacent motion vectors. Let the size of each block is 16×16, Fig. 1 shows the position-relation of the reference block and three adjacent blocks used for prediction. The current reference block B_r is located at (m, n), where $m(n)$ is the column (row) number. Its three neighboring blocks, at location $(m-32, n)$, $(m-16, n)$, and $(m, n-16)$, are denoted as B_i for $i \in \{1, 2, 3\}$. Let the motion vector of B_i be represented by $V_i = [mx_i, my_i]$, and $\hat{P}_r = [mx_r, my_r]$ mean the predicted motion vector of B_r. To determine $\hat{P}_r$, we use V_1, V_2, and V_3 to construct the input data sequence for gray prediction. Based on gray prediction procedure mentioned [6], the $\hat{P}_r$ is predicted efficiently. After $\hat{P}_r$ is determined, the initial search center (c_x, c_y) for B_r will be set at $(m + mx_r, n + my_r)$.

Using the search center (c_x, c_y) determined in the first phase, GrPS, in the second phase, starts searching the whole search area with a 3×3 movable search window until the local minimum SAD point (or block) lies in the center of the present search window, or the number of iteration of the search loop reaches the given maximum, which is set to 7 in this design. At each search iteration, B_r is compared with the points of the present 3×3 search window using the SAD operation. If the minimum SAD point is located at the center of the present search window, the search loop ends. Otherwise, some search patterns are used to perform the next search iteration. If the minimum SAD point is located at the middle of the boundary column (row) of the present search window, the center of the next search window is shifted to the minimum SAD point, and three additional vertical (horizontal) checking points as shown in Fig. 2(a)–(d) are added for next search. If the minimum SAD point is located at the corner of the present search window, the center of the next search window is shifted to the minimum SAD point, and 3 additional checking points as shown in Fig. 3(a)–(d) are added for the next search. The search process will repeat until the iteration times larger than 7, or will end as soon as the minimum SAD point is located at the center of present search window. Fig. 4 shows the GrPS algorithm in the C language

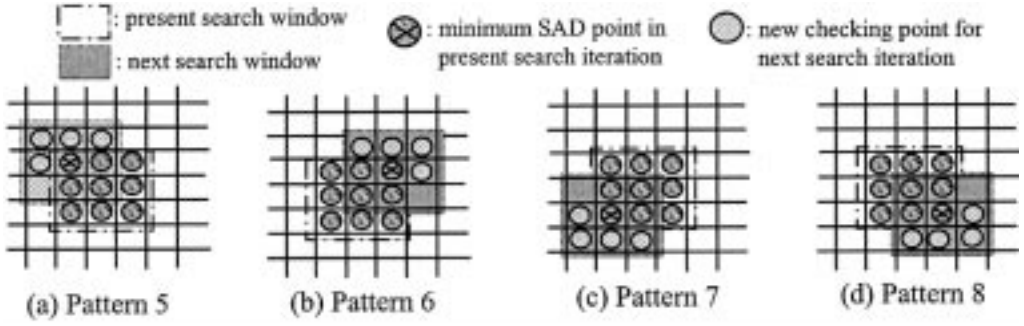

Fig. 5. New patterns for the next search iteration when the minimum SAD point is located at the corner of the present search window. (a) Pattern 5. (b) Pattern 6. (c) Pattern 7. (d) Pattern 8.

TABLE I
VALUES OF R_N AND R_P FOR DIFFERENT SEARCH PATTERNS

Value / Pattern	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	Pattern 8
R_n	3	3	1	1	2	2	2	2
R_p	-1	-1	-1	1	-1	-1	0	0

style. GrPS performs better than other algorithms, such as TSS, CS, PHODS, FSS, and SES, in terms of six measures. For more details, please refer to [6].

C. Hardware Implementation Issues for GrPS

The first difficulty to implement GrPS with hardware is that gray prediction in label L1 of Fig. 4 requires extensive and complex computations. According to our simulation, gray prediction in GrPS needs about 19 addition operations, 30 multiplication operations, and two exponential operations to perform its function. To solve the problem, the table-look-up technique is adopted to reduce implementation complexity and to improve predicting speed. The inputs for gray prediction are the vertical or horizontal component of three adjacent blocks' motion vectors, and the output is the vertical or horizontal component of $\hat{P}_r$. Since four bits are used to represent the vertical (horizontal) component of a motion vector, between -7 and 7, there are up to $2^{3\times4}$ distinct combinations of 0s and 1s. For each possible input combination, we precalculate the corresponding vertical (horizontal) component values of each $\hat{P}_r$ using GM(1, 1) gray prediction, and then store them in a table, called the Gray table. That is, the Gray table is constructed with $2^{3\times4}$ entries, and each of the entries stores one corresponding vertical (horizontal) component value, represented with 4 b. It is to note that the vertical and horizontal components of a vector use the same look-up-table in this design.

The second difficulty is that the required number of checking points and those points' locations are different every while-iteration (see label L2 of Fig. 4). In the first while-iteration, we check nine points of a 3×3 search window to find the minimum SAD point. But, only three additional points, shown in Figs. 2 and 3, are necessary in the following while-iterations. To solve the problem and to consider the regularity for hardware realization, we use a 3-PE architecture to implement GrPS. The three processing elements perform SAD function on three checking points of a row of a 3×3 search window nearly in parallel. In other words, three PEs can accomplish a row-SAD operation simultaneously, and thus the architecture may perform one, two or three row-SAD operations, based on the current search pattern number, every while-iteration. To regulate the data flow, we modify the search patterns shown in Fig. 3 by adding one additional checking point, and the new patterns shown in Fig. 5 are adopted for hardware implementation. Let R_n represent the number of row-SAD operations required for the next search iteration and R_p represent the corresponding vertical displacement between the minimum SAD point in the present search window and the start position of the first row-SAD operation for the next search iteration, Table I shows the values of R_n and R_p for the eight search patterns (Figs. 2 and 5) used in the hardware design. Therefore, the hardware behavioral description for GrPS can be written as Fig. 6. In the Figure, we number the main operations as o_i. The hardware components used to implement those operations are described in the next section in detail.

III. CHIP ARCHITECTURE FOR GRPS

Fig. 7 shows the block diagram of VLSI architecture for GrPS. The architecture consists of four memory banks, four address generators, the 3-PE MVG, and the controller. Each of them is introduced in the following subsections.

A. Memory Banks

Four memory banks are used to store motion vectors, the Gray table, pixels of B_r, and pixels of all possible candidate blocks, respectively. The memory for motion vector, denoted as *MM*, stores all blocks' motion vectors in the current frame. A frame is of size 720×480, the required size is $(720/16)\times(480/16) = 45 \times 30 = 1350$ bytes for *MM*. The memory for the Gray table, denoted as *MG*, stores the vertical (or horizontal) component of the predicted motion vectors. Three adjacent motion vectors are used for gray prediction, so a table with $2^{4\times3}$ entries must be constructed, in which each entry needs 4 b to store a predicted vertical (or horizontal) component value between -7 and 7. Thus, the size of *MG* is 2 K bytes.

```
/* MM: memory for motion vector;   AG_MM: address generator for MM;   AD_MM: address line for MM;
   MG: memory for Gray table;      AG_MG: address generator for MG;   AD_MG: address line for MG;
   T[1]~T[3]: 8-bit registers for three adjacent motion vectors;
   T[i].x: 4-bit horizontal vector of T[i];   T[i].y: 4-bit vertical vector of T[i];   */

   GrPS(b_x, b_y)
   {
o1     AD_MM = AG_MM(b_x-2, b_y);          /* determine address for V1 */
o2     T[1]=MM[AD_MM];                     /* read V1 from MM */
o3     AD_MM = AG_MM(b_x-1, b_y);          /* determine address for V2 */
o4     T[2]=MM[AD_MM];                     /* read V2 from MM */
o5     AD_MM = AG_MM(b_x, b_y-1);          /* determine address for V */
o6     T[3]=MM[AD_MM];                     /* read V3 from MM */
o7     AD_MG = AG_MG(T[1].x,T[2].x,T[3].x);  /* determine address for horizontal predicted vector */
o8     mx_r = MG[AD_MG];                     /* read the horizontal predicted vector from MG */
o9     AD_MG = AD_MG(T[1].y,T[2].y,T[3].y);  /* determine address for vertical predicted vector */
o10    my_r = MG[AD_MG];                     /* read the vertical predicted vector from MG */
o11    min=0xFFF;                          /* min: minimum SAD found in the previous iterations */
o12    R_p=-1; R_n=3; i=1; Exit =0;   /* initial values */

o13    While ( i<=7 && Exit=0 )
o14    {  dx=0; dy=0; j=R_p;
o15           While ( j<(R_p+R_n) )
o16           { sad1=SAD(-1, j);          /* see Equation (1) */
o17             sad2=SAD(0, j);
o18             sad3=SAD(1, j);
o19             if (sad1<min) { min= sad1;  dx=-1;  dy=j;}
o20             if (sad2<min) { min= sad2;  dx=0;   dy=j;}
o21             if (sad3<min) { min= sad3;  dx=1;   dy=j;}
o22             j++;    }
o23       (R_n, R_p, Exit)=Update(dx, dy); /* Update will be described in Section III.C */
o24       mx_r = mx_r + dx; my_r = my_r + dy;  /* accumulate the motion vector */
o25       i++;
       }

o26    MM[AG_MM(b_x, b_y)]=(mx_r, my_r);        /* write motion vector */
   }
```

Fig. 6. Hardware behavioral description for GrPS.

The memory for B_r, denoted as *MR*, stores the pixel values of B_r in the current frame. Thus, the required size is $16 \times 16 = 256$ B. The memory for candidate blocks, denoted as *MC*, stores the pixel values of all possible candidate blocks in the whole search area of the previous frame. Since the maximum of search iteration is 7, the required size for *MC* is $30 \times 30 = 900$ B. To reduce the loading time for *MC*, the 16-b data line is adopted to load two adjacent pixel values simultaneously. In the design, *MM*, *MR*, and *MC* are all stored in the row-major format.

B. Address Generators

Four address generators, AG_{MM}, AG_{MG}, AG_{MR}, and AG_{MC} are used to generate the addresses for memory banks, *MM*, *MG*, *MR*, and *MC*, respectively. AG_{MR} is used to generate the addresses of the pixels of B_r step by step, and AG_{MC} is used to generate the addresses of the pixels of all possible candidate blocks in sequence.

AG_{MM} uses the block index (b_x, b_y) to generate the address AD_{MM} for accessing the block's motion vector. Fig. 8 shows

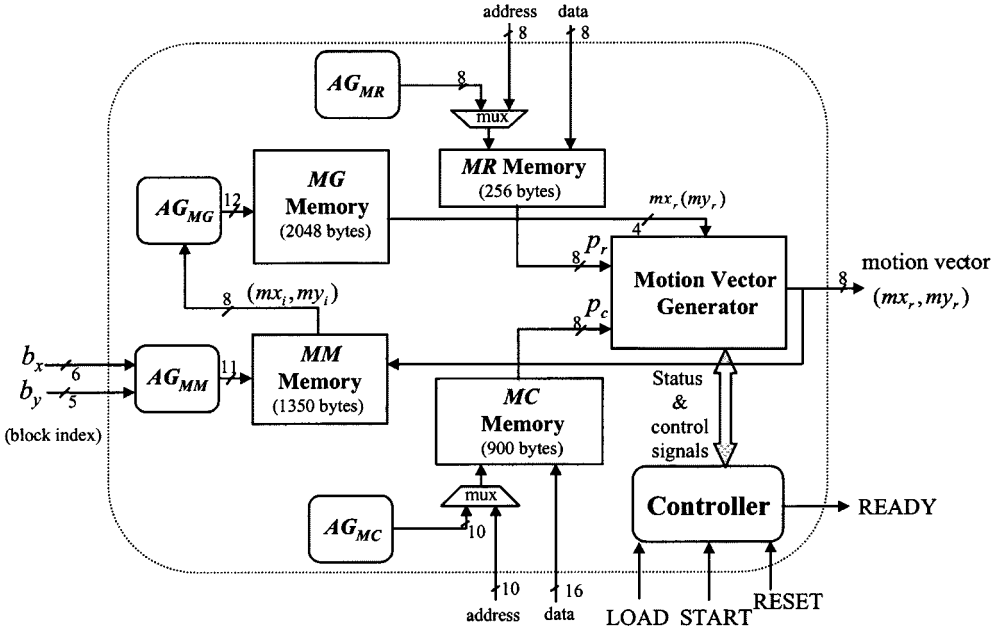

Fig. 7. Block diagram of the VLSI architecture for GrPS.

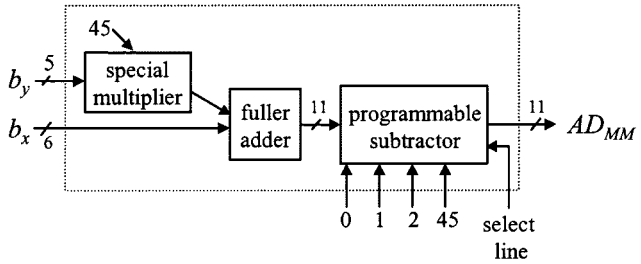

Fig. 8. AG_{MM} architecture.

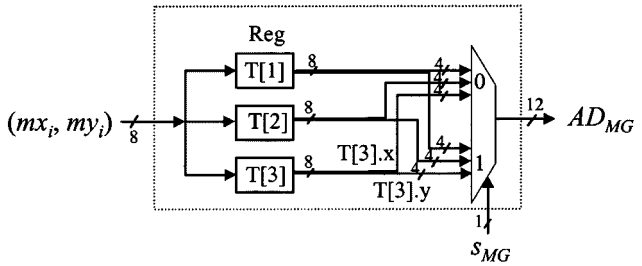

Fig. 9. AG_{MG} architecture.

the architecture of AG_{MM}. Since *MM* memory is stored in the row-major format and a row consists of 45 blocks, the register records the corresponding address to store V_r. The multiplier is implemented with shift, subtraction, and addition operations due to the constant 45. There are four constants: 0, 2, 1, and 45 as one of the subtractors' operand. The input 0 is used to generate the address for writing the motion vector V_r (o_{26} of Fig. 6). The inputs 2, 1, and 45 are used to generate the addresses for reading $V_1(o_1)$, $V_2(o_3)$, and $V_3(o_5)$, respectively. Note that the programmable subtractor is optimally designed due to that one of its operand is constant (0, 2, 1, or 45).

After motion vectors, V_1, V_2, and V_3 are read out from *MM* in sequence, AG_{MG} keeps them in T[1], T[2] and T[3], respectively. Then, the horizontal 4-bit components of T[1], T[2] and T[3] are used to generate the address AD_{MG} for reading the horizontal component of $\hat{P}_r$. After that, the vertical 4-b components of T[1] to T[3] are used for reading the vertical component of $\hat{P}_r$. Fig. 9 shows the architecture of AG_{MG}. Using the signal s_{MG}, generated by the controller, AG_{MG} can generate the addresses for reading mx_r and my_r in sequence.

C. MVG

The MVG is used to execute the SAD computations ($o_{16} \sim o_{18}$ of Fig. 6), to find the best-matched candidate block ($o_{19} \sim o_{21}$), and then to accumulate the displacement (dx and dy) of each iteration in order to produce the final motion vector ($o_{23} \sim o_{24}$). Fig. 10 shows the block diagram of the MVG. It consists of five components: the processor array (PA), the displacement finding unit (DFU), the vector accumulation unit (VAU), the update unit, and loop decision unit. The signals, denoted as s_k, are control signals generated by the controller, and the signals, sent to the controller for determining status, are denoted as r_k.

The PA which is composed of three processing elements (PEs) performs the calculation of a row-SAD operation between B_r and three candidate blocks nearly in parallel. Each PE performs SAD operation, and its internal logic diagram is shown in Fig. 11. In this structure, the inputs p_r^i and p_c^i are applied to the PE simultaneously to compute the value of $|p_r^i - p_c^i|$, where p_r^i and p_c^i are the pixel values of B_r in current frame and the pixel values of a candidate block in the previous frame, respectively. Then, the value of $|p_r^i - p_c^i|$ is stored in the register for accumulating the SAD. Since each pixel need one clock cycle for computing the SAD value, a PE spends $16 \times 16 = 256$ clock cycles for one candidate block.

Considering the reuse of input pixel values, the PA is designed with the architecture shown in Fig. 12. Using two delay registers, the pixel values of B_r can enter each PE1, PE2 and PE3 in correct sequence. With the selection signals, s_1, s_2, and

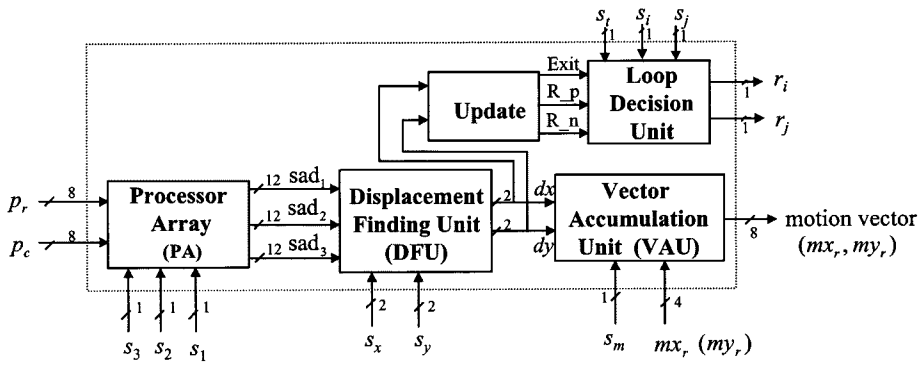

Fig. 10. Block diagram of the MVG.

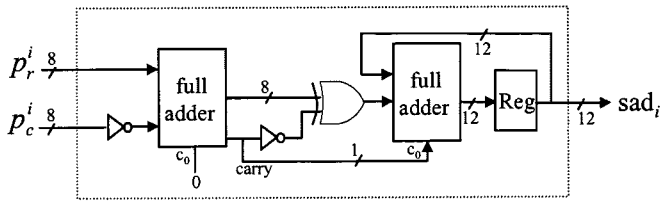

Fig. 11. Internal logic diagram of each PE i.

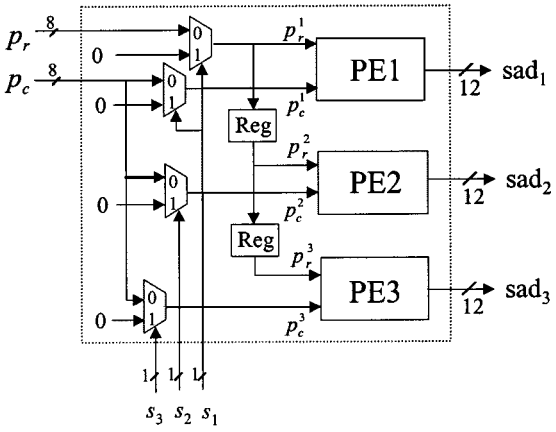

Fig. 12. The PA.

CYCLE TIME	Data Sequence		PE1	PE2	PE3
T	p_r	p_c			
1	$p_r(0,0)$	$p_c(0,0)$	$p_r(0,0)$- $p_c(0,0)$		
2	$p_r(0,1)$	$p_c(0,1)$	$p_r(0,1)$- $p_c(0,1)$	$p_r(0,0)$- $p_c(0,1)$	
3	$p_r(0,2)$	$p_c(0,2)$	$p_r(0,2)$- $p_c(0,2)$	$p_r(0,1)$- $p_c(0,2)$	$p_r(0,0)$- $p_c(0,2)$
:	:	:	:	:	:
16	$p_r(0,15)$	$p_c(0,15)$	$p_r(0,15)$- $p_c(0,15)$	$p_r(0,14)$- $p_c(0,15)$	$p_r(0,13)$- $p_c(0,15)$
17		$p_c(0,16)$		$p_r(0,15)$- $p_c(0,16)$	$p_r(0,14)$- $p_c(0,16)$
18		$p_c(0,17)$			$p_r(0,15)$- $p_c(0,17)$
18+1	$p_r(1,0)$	$p_c(1,0)$	$p_r(1,0)$- $p_c(1,0)$		
18+2	$p_r(1,1)$	$p_c(1,1)$	$p_r(1,1)$- $p_c(1,1)$	$p_r(1,0)$- $p_c(1,1)$	
18+3	$p_r(1,2)$	$p_c(1,2)$	$p_r(1,2)$- $p_c(1,2)$	$p_r(1,1)$- $p_c(1,2)$	$p_r(1,0)$- $p_c(1,2)$
:	:	:	:	:	:
18+16	$p_r(1,15)$	$p_c(1,15)$	$p_r(1,15)$-$p_c(1,15)$	$p_r(1,14)$- $p_c(1,15)$	$p_r(1,13)$- $p_c(1,15)$
18+17		$p_c(1,16)$		$p_r(1,15)$- $p_c(1,16)$	$p_r(1,14)$- $p_c(1,16)$
18+18		$p_c(1,17)$			$p_r(1,15)$- $p_c(1,17)$
.	.	.	.	.	.
.	.	.	.	.	.
15x18+1	$p_r(15,0)$	$p_c(15,0)$	$p_r(15,0)$-$p_c(15,0)$		
15x18+2	$p_r(15,1)$	$p_c(15,1)$	$p_r(15,1)$- $p_c(15,1)$	$p_r(15,0)$- $p_c(15,1)$	
15x18+3	$p_r(15,2)$	$p_c(15,2)$	$p_r(15,2)$- $p_c(15,2)$	$p_r(15,1)$- $p_c(15,2)$	$p_r(15,0)$- $p_c(15,2)$
:	:	:	:	:	:
15x18+16	$p_r(15,15)$	$p_c(15,15)$	$p_r(15,15)$- $p_c(15,15)$	$p_r(15,14)$- $p_c(15,15)$	$p_r(15,13)$- $p_c(15,15)$
15x18+17		$p_c(15,16)$		$p_r(15,15)$- $p_c(15,16)$	$p_r(15,14)$- $p_c(15,16)$
15x18+18		$p_c(15,17)$			$p_r(15,15)$- $p_c(15,17)$

Fig. 13. Data flow in the PA.

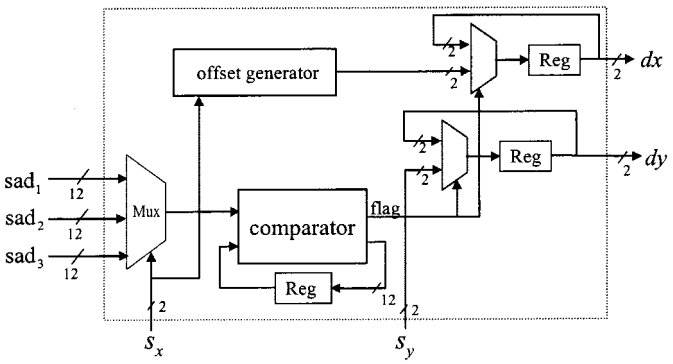

Fig. 14. Displacement finding unit.

s_3, generated by the controller, each PE can obtain the correct input data, p_r^i and p_c^i, to perform the SAD computations. Fig. 13 shows the data flow in the PA for calculating three candidate blocks in parallel. From Fig. 13, we find that it needs $(16 \times 18) = 288$ clock cycles for completing the SAD computations for the three candidate blocks because of the input data dependency, and sad_1, sad_2, and sad_3 will come out at the 286th, 287th, 288th cycle, respectively. In other words, the PA performs the three operations, o_{16}, o_{17} and o_{18} shown in Fig. 6, nearly in parallel. In the design, the hardware utilization of each PE is about 89% (16/18).

The function of the displacement finding unit (DFU) is to find the minimum SAD among three SADs generated by PA, and to determine the corresponding displacement (dx and dy) for the current search iteration. As mentioned, the three SADs, sad_1, sad_2, and sad_3, are generated one by one, so one comparator is enough to perform the comparisons. Its architecture is shown in Fig. 14. The selection signals s_x and s_y, generated by the controller, are used to indicate the displacement of x-ordinate and y-ordinate of the SAD block under comparing. When the sad_1, sad_2, or sad_3 is selected, the offset generator will generate -1, 0, or 1, respectively. With one comparator, DFU performs the three operations, o_{19}, o_{20} and o_{21}, one by one. After one, two, or three row-SAD operations, the DFU transmits the final dx and dy for this search iteration to vector accumulation unit (VAU). The VAU, shown in Fig. 15, performs the operation o_{24}

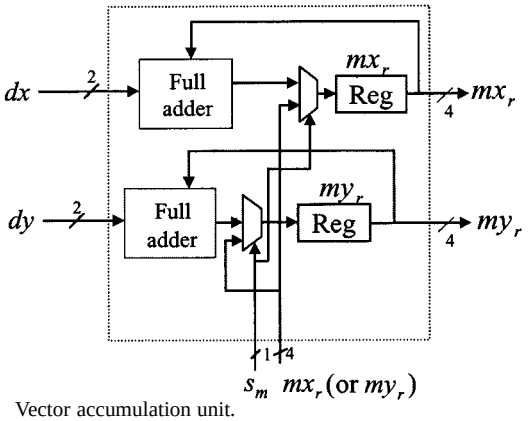

Fig. 15. Vector accumulation unit.

```
Update (Input: dx, dy; Output:R_n, R_p, Exit)
{   if (dx= 0 AND dy= 0) Exit= 1;
    if (|dx|= |dy|)
       { R_n= 2;
         if (dy= -1) R_p= -1 else R_p= 0;}
    if(|dx|= 1 AND dy= 0) { R_n= 3 ; R_p= -1; }
    if(dx= 0 AND |dy|= 1) { R_n= 1; R_p= dy; }
}
```

(a)

dx	dy	R_p	R_n	Exit
0 0	0 0	x x	x x	1
0 0	0 1	0 1	0 1	0
0 0	1 0	1 0	0 1	0
0 0	1 1	x x	x x	x
0 1	0 0	1 0	1 1	0
0 1	0 1	0 0	1 0	0
0 1	1 0	1 0	1 0	0
0 1	1 1	x x	x x	x
1 0	0 0	1 0	1 1	0
1 0	0 1	0 0	1 0	0
1 0	1 0	1 0	1 0	0
1 0	1 1	x x	x x	x
1 1	0 0	x x	x x	x
1 1	0 1	x x	x x	x
1 1	1 0	x x	x x	x
1 1	1 1	x x	x x	x

(b)

	Decimal	Binary
dx and *dy*	-1	10
	0	00
	1	01
R_p	-1	10
	0	00
	1	01
R_n	1	01
	2	10
	3	11

(c)

Fig. 16. (a) Procedure for update unit. (b) Truth table for the update unit. (c) Binary representations for dx, dy, R_p, and R_n.

to accumulate the dx and dy. When the whole search process ends, the contents of the two registers, denoted as mx_r and my_r, in VAU are the final MV.

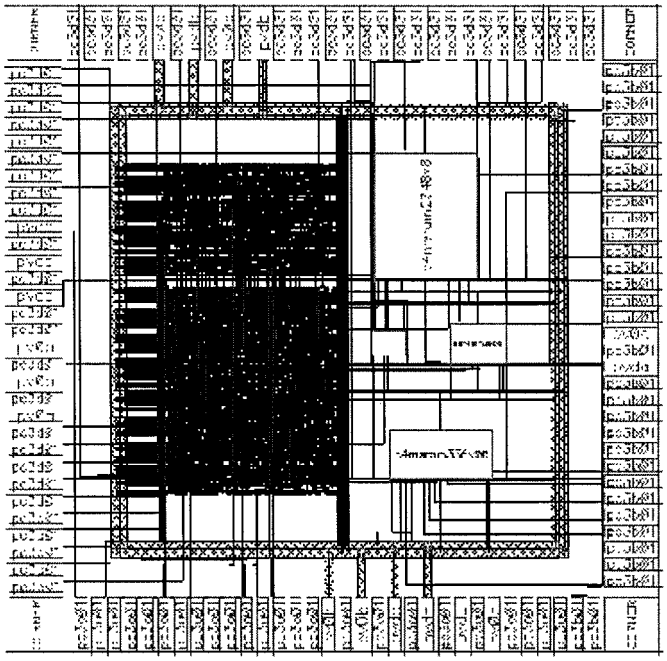

Fig. 17. Layout of the GrPS chip.

TABLE II
FEATURES OF FOUR MOTION ESTIMATION CHIPS

	[12]	[13]	[14]	This chip
Function	Motion estimation and compensation	Motion estimation	Motion estimation	Motion estimation
Algorithm	TSS-Based	Programmable	TSS	GrPS
Memory	20Mb	6Kb	12Kb	5Kb
PEs	49	72	9	3
Technology	0.5 μm 3-level metal CMOS	0.5 μm 3-level metal CMOS	0.8 μm 2-level metal CMOS	0.6 μm 3-level metal CMOS
Supply voltage	3.3 V	3.3 V	5V	5V
Die size	16.5×16.5 mm^2	4.2×6.6 mm^2	6.9×5.9 mm^2	2.8×2.9 mm^2
Transistors	2.0 M	162K	120K	54K
Clock rate	81MHz	66MHz	50MHz	66MHz
Pins	340	96	48	128
Coding standard	MPEG2	H.26x, CIF 30f/s	MPEG2	MPEG2
Resolution	720x480 pels, 30f/s	Programmable	720x480 pels, 30f/s	720x480 pels, 30f/s
Control design	RISC processor	Host processor	Random logic	Random logic

The DFU also feeds the final dx and dy of the current search iteration to the update unit. The update unit uses dx and dy to determine whether the whole search process should be terminated. If yes, set Exit signal to 1. If not, update unit will generate the new R_n and R_p used for the next search iteration. According to data shown in Table I, Fig. 16(a) shows the procedure to generate the new R_n and R_p (operation o_{23}). The update unit is implemented with an optimized combinational circuit. Fig. 16(b) shows the truth table used for the update unit where the binary representations for dx, dy, R_p, and R_n are shown in Fig. 16(c). The loop decision unit, shown in Fig. 10, uses the Exit, R_n and R_p, generated by the update unit, to determine the conditions for i-loop (o_{13}) and j-loop (o_{15}), respectively. This unit will generate two signals, r_i and r_j, to inform the controller the status of the two loops.

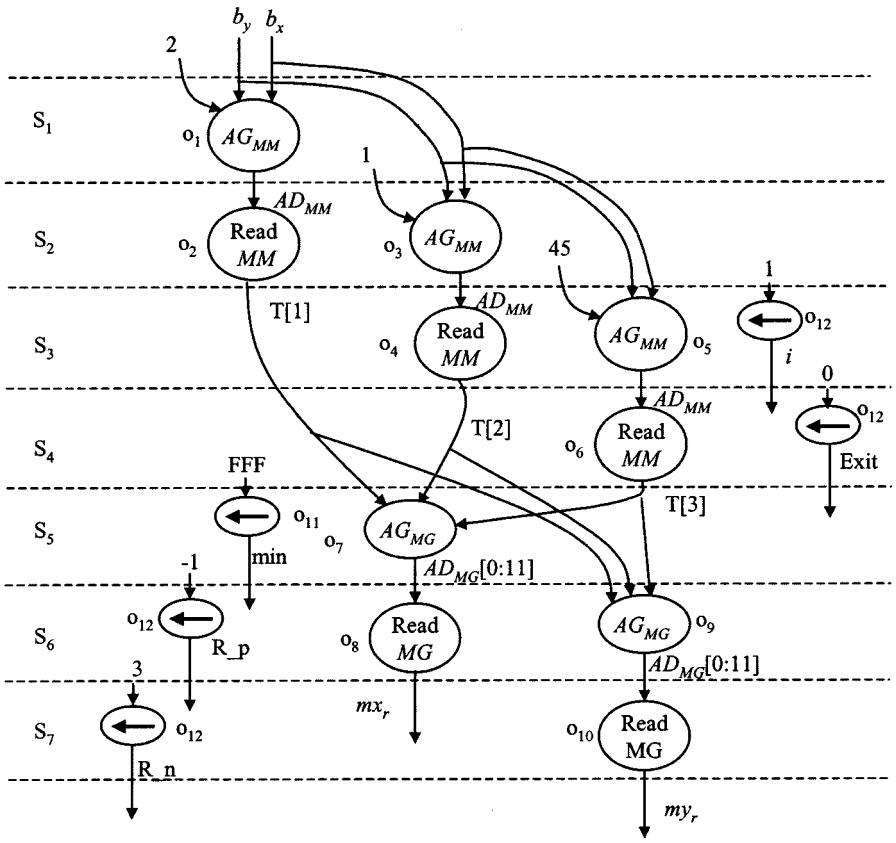

Fig. 18. Scheduling of the operations, o_1 to o_{12}.

D. Control Unit

The control unit monitors the data flow and sends proper control signals to the MVG, to the four memory blocks, and to the four address generators. In addition, two input signals, LOAD and START, and one output signal READY, shown in Fig. 7, are used to set up the working mode of our design. When the LOAD signal is high, the design loads data (pixel values) from the external environment into the two memory banks, *MR* and *MC*. When the LOAD signal is set to low, bringing the START high will activate the design to search the MV for the current block with the index (b_x, b_y). When the design terminates the search loop and finds the final motion vector, it will set the Ready signal to high to notify that it is ready for the next block's motion vector search.

IV. VLSI Implementation

A prototype of the chip has been implemented by using the ASIC standard-cell from COMPASS' 0.6-μm SPTM cell library [18]. The function of the proposed GrPS chip is first verified by using Verilog HDL. Then, we employ Synopsys tools to synthesis the design. Finally, the layout for the design is generated with the CADENCE tools, Preview (for floorplan), Silicon Ensemble (for placement and routing), and Dracula (for DRC, ERC and LVS checks). The core size of the design is about 2.8×2.9 mm^2 with about 54–K transistors. The layout of the design is shown in Fig. 17. For easy comparison, Table II lists features of the GrPS chip and other recent motion estimation chips [12]–[14]. Obviously, the GrPS chip has the minimum numbers of PEs and transistors among these chips, and other chips have at least three more times of the PE number than it. Note that all PEs in these chips have the similar functions and then may be though with the same cost. As to the clock frequency, the minimum one [14] is 50 MHz but with 9 PEs and 120-K transistors, our chip is 66 MHz with only 3 PEs and 54-K transistors. In addition, the GrPS obtains excellent video quality (little worse than FSS but better than TSS) as shown in [6]. Therefore, our design is an efficient and low-cost motion estimator for real-time video coding.

In the design, one clock cycle is used to generate the corresponding addresses for *MM* and three cycles are used to read three motion vectors from *MM*. The address for *MG* is generated in the following cycle, and another two cycles are used to obtain mx_r and my_r from *MG*. Then, the initial values for the search process are set in the following clock cycle. Totally, eight clock cycles are required to perform the operations, o_1 to o_{12}, before the search process can start. Fig. 18 shows the scheduling of those operations.

The number of cycles needed to complete the entire search process, Z, is

$$Z = N \times \mathrm{num}_i$$

where num_i, between 1 and 7, is the required number of search iteration (the while i_loop in Fig. 6), and N is the number of cycles used to complete one search iteration. N is given as

$$N = 1 + (S \times \mathrm{num}_j) + 2$$

TABLE III
VALUES OF num_i AND num_j FOR DIFFERENT VIDEO SEQUENCES

	Football	Mobile	Windmill	Flower	Tennis	Salesman	Miss-A	Average
num_i	1.79	1.36	1.31	1.01	1.32	1.39	1.05	1.32
num_j	1.93	1.84	1.95	1.92	2.02	2.08	1.93	1.95

where the one clock cycle is used to perform o_{13} and o_{14} of Fig. 6 simultaneously, S represents the cycles (288) used to perform a row-SAD operation, num_j represents the required number of row-SAD operations (the while j_loop in Fig. 6) in the current search iteration, and the last two cycles are used for DFU and update unit, respectively. Using the seven test image sequences (Football, Mobile, Windmill, Flower, Tennis, Salesman, and Miss America [6]) for experiment, we found num_j is equal to 1.95 and num_i is about 1.32 in average for GrPS, as shown in Table III. Thus, in average $N = 1 + 288 \times 1.95 + 2 \cong 565$ clock cycles. Hence, $Z = 565 \times 1.32 \cong 746$ cycles. For the worst case in the seven sequences, *i.e.*, Football sequence, $N = 1 + 288 \times 1.93 + 2 \cong 559$ cycles, then $Z = 559 \times 1.79 \cong 1001$ cycles.

After the search process ends, one cycle is used to write MV into the *MM*. Thus, totally we need $7 + 746 + 1 = 754$ clock cycles to find a MV for a block. The loading time for *MR* and *MC* need 256 cycles and $900/2 = 450$ cycles, respectively. Because *MR* and *MC* are loaded simultaneously, 453 (three additional cycles for R/W handshaking) cycles are used for initial loading. Thus, an image frame requires $(453 + 754) \times (720/16 \times 480/16) = 1\,629\,450$ cycles to process. For the frame rate 30/s, the running speed is at least $1\,629\,450 \times 30 \cong 48.88$ MHz in average. In the worst case of the Football sequence, the running speed is at least $1\,973\,700 \times 30 \cong 59.21$ MHz. The design works at 66-MHz clock frequency, so it can support the MPEG video resolution (720 pixels $\times$ 480 lines, 30 frames/s) in real time. Although, the frequency margin is not high.

V. CONCLUSIONS

An efficient GrPS chip is proposed in this paper. Using the gray theory, GrPS can determine the motion vectors of image blocks quickly and correctly. To reduce the implementation complexity and to prompt the prediction speed, the gray prediction is implemented with the table-look-up approach. With a 3-PE structure, the GrPS chip works fast enough to provide suitable solutions for MPEG (720 pixels $\times$ 480 lines, 30 frames/s) in real time.

ACKNOWLEDGMENT

The authors wish to thank the anonymous reviewers for their valuable suggestions and careful review that helped to enhance the quality of the manuscript.

REFERENCES

[1] T. Koga, K. Iinuma, A. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. NTC81*, New Orleans, LA., 1981, pp. C9.6.1–9.6.5.

[2] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, vol. 38, pp. 950–953, July 1990.

[3] L.-G. Chen, W.-T. Chen, Y.-S. Jehng, and T.-D. Chiueh, "An efficient parallel motion estimation algorithm for digital image processing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 1, pp. 378–385, Apr. 1991.

[4] 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, Mar. 1996.

[5] J. Lu and M. L. Liou, "A simple and efficient search algorithm for block-matching motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 429–433, Feb. 1997.

[6] J. M. Jou, P.-Y. Chen, and J.-M. Sun, "The grey prediction search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 9, pp. 843–848, June 1999.

[7] 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.

[8] E. Chan and S. Panchanathan, "Motion estimation architecture for video compression," *IEEE Trans. Consumer Electron.*, vol. 39, pp. 292–297, Mar. 1993.

[9] A. S. H. Nam and M. K. Lee, "Flexible VLSI architecture of motion estimator for video image compression," *IEEE Trans. Circuits Syst. II*, vol. 43, pp. 467–470, June 1996.

[10] H.-M. Jong, L.-G. Chen, and T.-D. Chiueh, "Parallel architectures for 3-step Hierarchical search block-matching algorithm," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, pp. 407–416, Apr. 1994.

[11] A. Costa, A. D. Gloria, P. Faraboschi, and F. Passaggio, "A VLSI architecture for hierarchical motion estimation," *IEEE Trans. Consumer Electron.*, vol. 41, pp. 248–257, Feb. 1995.

[12] K. S. Suguri *et al.*, "A real-time motion estimation and compensation LSI with wide search range for MPEG2 video encoding," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1733–1741, Nov. 1996.

[13] H. D. Lin, A. Anesko, and B. Petryna, "A 14-GOP's programmable motion estimator for H.26x video coding," *IEEE J. Solid-State Circuits*, vol. 31, pp. 1742–1750, Nov. 1996.

[14] T.-H. Chen, "A cost-effective three-step hierarchical search block-matching chip for motion estimation," *IEEE J. Solid-State Circuits*, vol. 33, pp. 1253–1258, Aug. 1998.

[15] J. Deng, "Control problems of grey system," *Syst. Control Lett.*, vol. 5, pp. 288–294, 1982.

[16] Y.-P. Huang and C.-C. Huang, "The integration and application of fuzzy and grey modeling methods," *Fuzzy Set Syst.*, vol. 78, no. 1, pp. 107–119, 1996.

[17] Y.-P. Huang and T.-M. Yu, "The hybrid grey-based models for temperature prediction," *IEEE Trans. Syst., Man, Cybern. B*, vol. 27, pp. 284–292, 1997.

[18] *TSMC 0.6 μm SPTM Technology Manual*, Chip Implementation Center, R.O.C, 1996.

Jer Min Jou received the Ph.D. degree in electrical engineering and computer science from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 1987.

In 1989, he was an Associate Professor in the Department of Electrical Engineering and Computer Science, National Cheng Kung University, Tainan, Taiwan, R.O.C. Currently, he is a Professor at the same university. His research interests include ASIC design/synthesis, SoC hardware-software codesign, System design, VLSI CAD, and Asynchronous circuit design.

Dr. Jou was the recipient of a Distinguished Paper Citation at the 1987 IEEE ICCAD Conference in Santa Clara, CA. He received the Longterm Best paper award from Acer Foundation in 1998 and 1999. He also is the supervisor of the first rank research paper of the Chinese Institute of Electrical Engineering 2001, and is a reviewer of many journals including IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS I.

Yeu-Horng Shiau received the B.S. degree in electrical engineering from National Cheng Kuang University, Taiwan, R.O.C., in 1997, and is currently working toward the Ph.D. degree in electrical engineering. His research interests include video coding system and VLSI chip design.

Pei-Yin Chen received the B.S. degree from National Cheng Kung University, Taiwan, R.O.C., in 1986, the M.S. degree from Penn Sate University, Pennsylvania, in 1990, and the Ph.D. degree from National Cheng Kung University, in 1999, all in in electrical engineering.

He is currently an Associate Professor in the Electronic Engineering at Southern Taiwan University of Technology, Tainan, Taiwan, R.O.C. His research interests include VLSI chip design, fuzzy logic control, gray prediction, and video compression.

Shiann-Rong Kuang received the B.S. degree from National Center University, Taiwan, R.O.C., in 1990, and the M.S. and Ph.D. degrees from National Cheng Kung University, Taiwan, R.O.C., in 1992 and 1998, respectively, all in electrical engineering.

He is currently an Assistant Professor in the Department of Electronic Engineering, Southern Taiwan University of Technology, Taiwan, R.O.C. His research interests include VLSI chip design, high-speed digital circuit design, and data compression.