AN EFFICIENT AND RECONFIGURABLE VLSI ARCHITECTURE FOR DIFFERENT BLOCK MATCHING MOTION ESTIMATION ALGORITHMS

Xiao-Dong Zhang
Dept. of EE
University of Sci. and Tech.
AnHui, HeFei. P.R. China

Chi-Ying Tsui
Dept. of Electrical and Electronic Engineering
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong

Abstract— This paper describes a VLSI architecture which can be reconfigured to support both Full Search Block–Matching algorithm and 3–step Hierarchical Search Block–Matching algorithm. By using a reconfigurable register–mux array and a parameterizable adder tree, the 2–D array architecture provides efficient real time motion estimation for many video applications. We also propose a memory architecture and an associated switching network to solve the simultaneous data access problem.

1. INTRODUCTION

Motion estimation algorithms play an important role in video compression. Block matching algorithms are the most commonly used algorithm and several VLSI architectures have been proposed for different algorithms. These include architectures for full search block-matching algorithms (FBMA) [1],[2],[3] and architectures for 3–step hierarchical search block matching algorithms (HSA) [4],[5],[6].

Different video applications may need to adopt different motion estimation algorithms. Therefore, it will be better if we can have an architecture that can support different algorithms and operation conditions, and also can be reconfigured in real time. In this paper, a reconfigurable, low–latency and high throughput 2–D array architecture is presented. Using the notion of Register–Multiplexer Array (RMA) and Reconfigurable Adder Tree (RAT), the architecture can cater for irregular block matching and is able to cover both HSA and FBMA efficiently. Comparing with other 2–D systolic array architectures, the data skew in the data flow is minimized. This can significantly reduce the latency and the number of idle cycle. At the same time the power consumption is also reduced. On-chip memory and switching network are used to solve the simultaneous data access problem. By using this switching network, the computation of Mean Absolute Differences (MADs) can be begun at any row. Also the architecture has the flexibility to handle various block sizes.

2. BLOCK MATCHING ALGORITHMS

The purpose of the block matching algorithm is to find a best matched displaced block within a search range from the previous frame F_{t-1} for each $N \times N$ block in the current frame F_t. FBMA exhaustively matches all possible candidates in the search range to find the displacement (motion vector) with a minimal distortion. As a criterion of measuring distortion, the mean absolute difference (MAD) is calculated for each candidate (u, v) as follows:

$$MAD(u,v) = \sum_{l=0}^{N-1} \sum_{m=0}^{N-1} |a_{x+l,y+m} - b_{x+l+u,y+m+v}| \quad (1)$$

where (x, y) is the coordinate of the left-top pixel of the current block in F_t, $a_{x,y}$ and $b_{x,y}$ are the values of pixels in F_t and F_{t-1}, respectively, and the values of u and v are limited to between $-p$ to $p-1$ where p is the search range.

To reduce the heavy computational cost resulting from the massive number of candidate locations, HSA searches for the near best motion vector in a coarse-to-fine manner. In the first step, nine sparsely located candidates are evaluated and the one with the minimum MAD is picked out. In the second step, the search is focused on the area centered at the winner of the previous step, but distances between candidate locations are reduced by half. In a similar manner, step three compares the MADs of the nine locations around the winner of the second step and then gives the final motion vector.

3. THE PROPOSED ARCHITECTURE

Figure 1 is the overall block diagram of the proposed architecture. It consists of a Process Element Array (PEA), on–chip memories for current and previous block data, a Register–Multiplexer Array(RMA) for

previous frame pixel data flow, a Reconfigurable Adder Tree (RAT) for parallel computation of the MADs, an address generator and a controller.

The PEA is composed of 3 × 16 PEs (Figure 2). These PEs are divided into three rows and 16 columns. Each row is responsible for calculating the MAD for a candidate block during the block matching. So it takes 16 cycles for each row to complete all the absolute difference computations for one MAD calculation and 3 MADs are calculated at the same time. The previous data are input to the PEs through the RMA. The RMA is used to minimize the memory bandwidth or the number of banks used in the on-chip memory. The connection between PEA and RMA is shown in Figure 3. Search area data is input to the top register and is shifted into the register array. When the lower three registers are full, the computation starts. At each computation cycle, the RMA provides three different search area data for the three PEs of each PE column. At the same time, the Current Block Area's on–chip memories provide a current block data for each PE column. Each PE cell computes the absolute difference of its two inputs and outputs the result to its corresponding adder in the RAT. The RAT is a pipelined adder tree which computes the sum of the 16 absolute differences (Figure 4).

The proposed architecture uses the Register–Memory Array (RMA) to buffer the data from the search area memory to the PEA. It consists of 16 columns, each column has 6 registers and 3 2–to–1 multiplexers. These registers are divided into two groups, high and low groups. The data from the search area's on–chip memory enters at the top of these register groups and shifts through them cycle by cycle. The multiplexers select the data from the right register group and feed them into the corresponding PEs in the PE column. The operation can be configured for both 3–step HSA and FBMA (Figure 5).

For 3–step HSA, the multiplexers select data from the low register group. In each step, the 9 candidate block matching positions are divided into 3 row. The MADs of the candidate blocks in the same row are computed simultaneously by the three PE rows. The on–chip memory outputs data into the register array in an interleaving order to make sure that the data will be shared in the computation. In step one the distance of the search data input to the PEs in the same column is 4. For example, at the first PE column the computations at the first cycle for the 3 PEs are $|a_{1,1} - b_{5,5}|$, $|a_{1,1} - b_{9,5}|$ and $|a_{1,1} - b_{13,5}|$ where $a_{i,j}$ and $b_{m,n}$ are the current block data at location (i, j) and search area data at location (m, n), respectively. So the interleaving data order is: 5,9,13,17,21,25,6,10,14,18,22,26,7,11,15,19,23,27,8,12, 16,20,24,28. For step two the interleaving data order is: 3,5,7,9,11,13,15,17,19,21,4,6,8,10,12,14,16,18,20,22.

For the step 3 of HSA, it is similar to the configuration for FBMA. Each row of the PE array is responsible for calculating the MADs of three adjacent candidate blocks. The high and low register groups will be used alternatively to minimize the idle cycles required to fill up the register array. When one group is outputting data to the PE arrays, the other group is storing the search data for the next three MADs computations. For example, when $b_{1,1}, b_{2,1}, b_{3,1}$ are shifted in the low register group, the computation of the first three MADs begins. The search data will be continuously shifted into the register array in a sequential manner. When $b_{4,1}, b_{5,1}, b_{6,1}$ are shifted in, they are also shifted in and stored in the high register group. After 16 cycles, the first three MADs' computations are finished and the next three computations begin with the search area data $b_{4,1}, b_{5,1}, b_{6,1}$ which are now available from the high group. Therefore the computations can start immediately without waiting for the data to shift in from the memory. The idle cycle between each MAD computation is minimized and the performance of the architecture is improved.

Comparing with the other array architectures [2],[3],[4],[6], the PE elements used here only calculate the absolute difference of two pixels and are much simpler. The area cost of the PE and the adder used in the RAT can be reduced by having a simpler and slower design. On the other hand, we can also reduce the power consumption. The power consumption of a CMOS circuit is mainly due to the dynamic power consumption which is used to charge and discharge the capacitance. For a circuit element n, the dynamic power P_n is equal to

$$P_n = C_{nL} V_{dd}^2 f_d \tag{2}$$

where C_{nL} is the effective switching load capacitance of n, f_d is the operating frequency, and V_{dd} is the supply voltage. For the proposed architecture, the critical path of a pipeline stage is either that of the adder in the RAT or that of the PE. For the other array architectures, the critical path is the sum of the delay of the adder and absolute difference. For a fixed throughput requirement, the proposed architecture can afford to have slower adders and PEs. It means that lower supply voltage can be used and significant power reduction can be achieved since power is proportional to the square of V_{dd}. Moreover dynamic power consumption

also depends on the switching activity of the circuit. It is shown that an adder tree structure has much smaller switching activity and the power consumption is less than 50% of that of the traditional array structures[7].

4. THE MEMORY ARCHITECTURE

During the block–matching, most pixels are used several times to evaluate different candidate locations. To lower the I/O bandwidth requirement, on–chip memories are used in the proposed architecture. For block size N and maximum displacement p, $\frac{3}{2}(N+2p-1)^2$ bytes and $2N^2$ bytes on–chip memories are used for search area and current block data, respectively, to realize zero waiting between adjacent block matchings. The search area memory are divided into 3 modules and each module has 16 banks. The organization of the search area memory is shown in Figure 6-8. During the computation, the search area memory provides 16 pixels in a column to each PE row for the MAD calculation. Each pixel in the column comes from consecutive memory rows and hence different memory banks. The use of memory bank provides simultaneous access of data, however it stipulates that every PE column be able to access each of the 16 memory banks. This is solved by using a switching network which can rotate the connections between the memory banks and PE columns in a required order. The network consists of $N \times \log N$ (16 × 4) 2–to–1 multiplexers as shown in Figure 9.

5. FLEXIBILITY FOR DIFFERENT BLOCK SIZE

Using the RAT, the proposed architecture has the flexibility to cater for different block sizes for FBMA (Figure 4). When the block size is 16 × 16 or 8 × 16, the full adder tree is used to calculate the MAD and the partial result are accumulated in the Accumulator $Accu_{1,j}$. After 16 or 8 cycles, a MAD is computed and output to the Comparator $Comp_{1,j}$ for comparison with the previous minimum. For block size of 8×8 and 8×16, the adder tree is split into two subtrees and the partial results from the two sub-trees are accumulated in the Accumulators $Accu_{1,j}$ and $Accu_{2,j}$, respectively. After 8 or 16 cycles, two MADs belonging to two neighboring current blocks are computed and output to the Comparators $Comp_{1,j}$ and $Comp_{2,j}$, respectively.

6. PERFORMANCE ANALYSIS AND COMPARISON WITH EXISTING ARCHITECTURES

The performance of the proposed architecture for a 16 × 16 block size, maximum displacement 8, 3–step HSA is as follows. For step 1, we need to shift in 24 search area data into each PE column for the MAD calculation. After the computation of all 3 rows of candidate blocks is finished, we need another extra 7 cycles for the RATs and the comparators to produce the address for the next step search. The total cycle required is $24 \times 3 + 7 = 79$. For step 2 and step 3, the procedure is similar except that the numbers of search data needed to be shifted in are 20 and 18, respectively. So the cycles required for step 2 and 3 are 67 and 61, respectively, and the total number of cycles required is 207.

For FBMA, due to the use of the high and low register group, it only takes 16 cycles to compute the MADs for 3 adjacent candidate blocks except for the first three blocks which need extra 3 cycles to shift the initial data into the register array. So the cycle required to compute MADs for a row in the search area is $T_0 = 3+16\times\lceil\frac{2p}{3}\rceil = 99$ for $p = 8$. The total clock cycles required for the full block matching is $T_0\times2p+7 = 1591$.

Comparing with existing architectures, the proposed architecture has higher performance and flexibility for various block sizes, lower cost, lower power consumption and can efficiently cover both FBMA and 3–step HSA. The comparisons with existing architectures are summarized in Table I.

REFERENCES

[1] T. Komarek and P. Pirsch. Array architectures for block matching algorithms. In *IEEE Trans. Circuits Syst.*, volume 36, pages 1317–1325, Oct 1989.

[2] L. De Vos and M. Schőbinger. Parameterizable VLSI architectures for the full–search block–matching algorithm. In *IEEE Trans. Circuits Syst.*, volume 36, pages 1309–1316, Oct 1989.

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

[4] H. M. Jong, L. G. Chen, and T. D. Chiueh. Parallel architecture for 3-step hierarchical search block–matching algorithm. In *IEEE Trans. Circuits Syst. For Video Technology*, volume 4, pages 407–416, Aug 1995.

[5] Y. S. Jehng, L. G. Chen, and T. D. Chiueh. A motion estimator for low bit–rate video codec. In *IEEE Trans. Consumer Elelctronics*, volume 38, May 1992.

[6] Z. L. He, M. L. Liu, P. C. H. Chan, and R. Li. An efficient VLSI architecture for new three-step search algorithm. In *Proc. of Midwest Symp. on Circuit and Systems, Rio de Janeiro*, Aug 1995.

[7] H. Chan and C.Y. Tsui. Exploring the power consumption of different motion estimation architectures for video compression. In *HKUST Dept. of EEE Technical Report.*

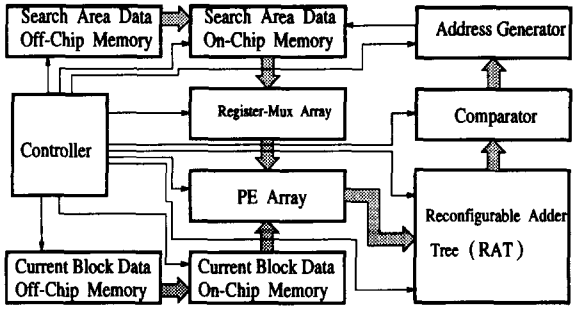

Fig. 1 The block diagram of the proposed architecture

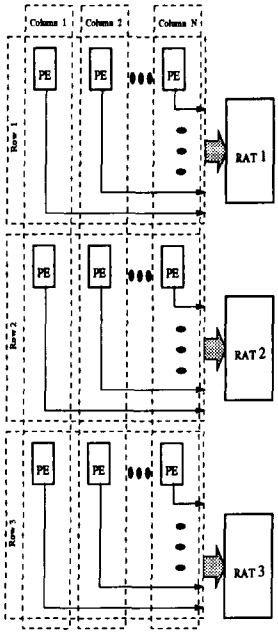

Fig 2 The PEA structure

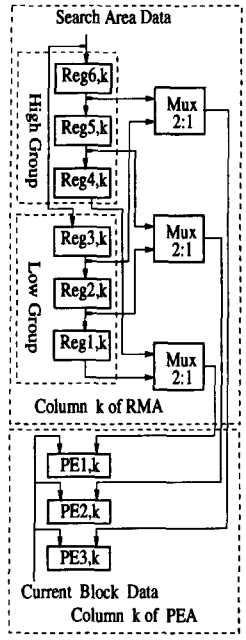

Fig 3 Connection between the PEA and the RMA

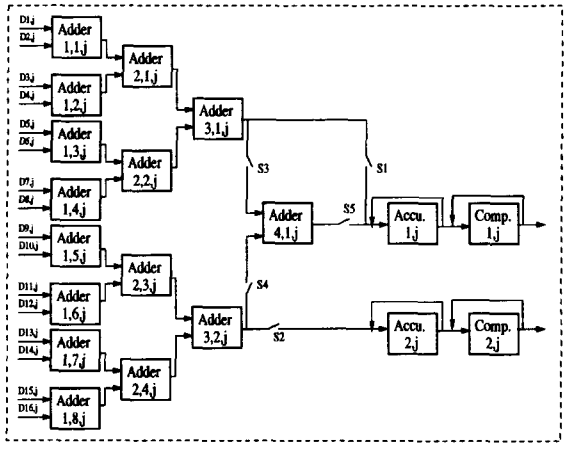

Fig 4 RAT for the PEA row j

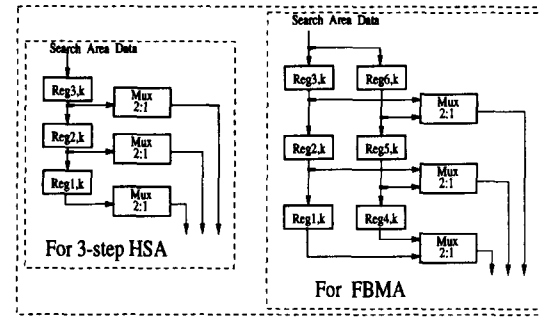

Fig 5 Configuration of the RMA for 3-step HSA and FBMA

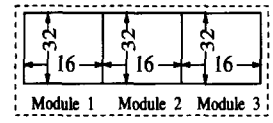

Fig 6 Memory structure of search area

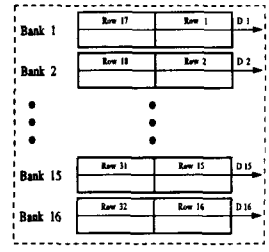

Fig 7 Structure of search area module

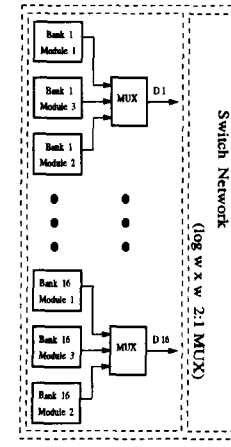

Fig 8 Memory organization of search area data

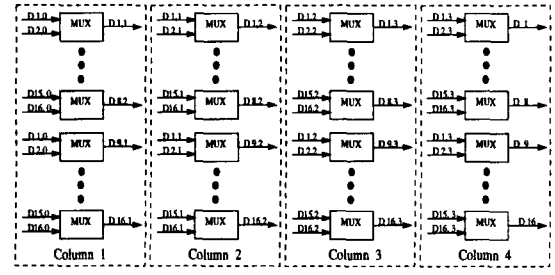

Fig 9 Switching network for the search area data

	Proposed Archi.	Archi. in[4]	Archi. in[5]	Archi. in[6]	Archi. in[3]	Archi. in [2]		
PE Number	48	9	16	48	16	256	128	64
Gate Number	24k*	13.6k	30k	32.9k	13k	192k	155k	136k
Timing (FS)	1591	/	/	1870	4096	256	512	1024
Timing (TSS)	204	794	3114	230	/	/	/	/
External data access times / Block Matching	768	1280	/	1280	11520	768	768	768

Table I Comparison between the proposed architecture and existing architectures
(* Not include on-chip Memory)