An Architecture of Full-Search Block Matching for Minimum Memory Bandwidth Requirement

Jen-Chien Tuan[*] *Chein-Wei Jen*[+]

Department of Electronics Engineering
National Chiao Tung University
Email: {*tuan, +cwjen}@ee.nctu.edu.tw

ABSTRACT

In this paper an architecture of full-search block matching motion estimation suitable for high quality video is proposed. Minimum memory bandwidth is an important requirement in motion estimation architecture especially when dealing with high quality video such as large frame size video. Memory bandwidth will increase to an unrealistically high value without careful consideration, which no cost efficient solution can afford it. This architecture is designed for overcoming the frame memory bandwidth bottleneck by exploiting the maximum data reuse property. This is done by setting up local memory for storing frame data. The size of local memory is also optimized to near minimum value, only little overhead is introduced. Due to the reduction of memory bandwidth, the costs of frame memory modules, I/O pin count and the power consumption can be reduced but 100% hardware efficiency is still achieved. Simple and regular interconnections is featured to ensure high speed operation by an efficient and distributed local memory organization.

1. INTRODUCTION

Motion estimation is widely adopted to be the basis of video compression. Motion estimation essentially requires high computation complexity and huge memory bandwidth. Computation complexity requirement can be fulfilled by multiple PEs implementation. Memory bandwidth problem can be solved by careful scheduling the data sequence and setting up on chip memories. In fact, motion estimation is an I/O bound problem rather than a computation bound one in straight forward implementation. A well designed motion estimation architecture can reduce the memory bandwidth, the required I/O pin count, and interconnection complexity but still maintain high hardware efficiency.

Full-search block matching algorithm (FS-BMA) is one of the algorithms used in motion estimation process. FS-BMA searches through every candidate locations to find the best match one. Various fast algorithms are proposed [10][11] to reduce the computation complexity of FS-BMA but the costs of irregular control and lower prediction video quality must be paid. So FS-BMA is still widely adopted for its simplicity and regularity and best prediction video quality. Many architectures are proposed to implement the FS-BMA [1]-[9][12]. These architectures use systolic array [1]-[8] or tree structure [9] even 1D structure [12] to provide enough computation power for solving the computation problem. But all of them have only limited solution for overcoming the frame memory bandwidth bottleneck,. That is, these architectures can not satisfy the memory bandwidth requirement of high quality video such as HDTV video format. High quality video means large frame size, high frame rate, and large search range. The required memory bandwidth increases with the increasing of these parameters. The total bandwidth required is the sum of bandwidth of current frame and bandwidth of previous frame, which is expressed as:

$$Bandwidth = \underbrace{f \cdot W \cdot H \cdot RAC_{curr}}_{current\ frame} + \underbrace{f \cdot W \cdot H \cdot RAC_{prev}}_{previous\ frmae} \quad (1)$$

where f is the frame rate, W and H are the width and height of frame, RAC_{curr} and RAC_{prev} are the redundancy access count (RAC) for both current frame and previous frame. The RAC is defined to be the ratio of how many times the same pixel is accessed from frame memory. For those systolic array architectures [1]-[5], RAC_{curr} equals 1 but RAC_{prev} depends on the search range, i.e., each pixel in previous frame is accessed more than once. This really causes a memory bandwidth problem when the search range becomes larger and larger. In the paper we propose an architecture which features both the RAC_{curr} and RAC_{prev} equal 1. This is the minimum requirement of frame memory bandwidth.

2. THE ARCHITECTURE

In our architecture, the mean absolute difference (MAD) is used for the matching criteria of FS-BMA because of its simplicity and suitable for hardware

This work was supported by National Science Council, Taiwan , R.O.C. under contract NSC86-2221-E009-014.

0-8186-8409-7/98 $10.00 © 1998 IEEE

implementation.

2.1. The PE module architecture

Fig 1 shows the architecture of a single PE module. There are one ALU performing the absolute difference (la-bl) operation, two registers (AR and PR registers) propagating current block data, one three inputs multiplexer, and one search area memory module (SAM) within each PE module. There are totally seven I/O ports for each PE module, each of them is one pixel wide. Three kind of data run through each PE module, the current block data, the search area data, and the absolute difference data. In the following we describe this architecture by considering the data path and interconnections according to these three kind of data.

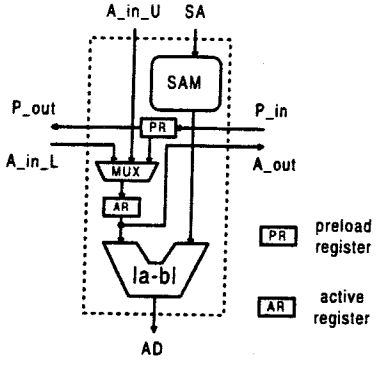

Fig 1: Architecture of PE module.

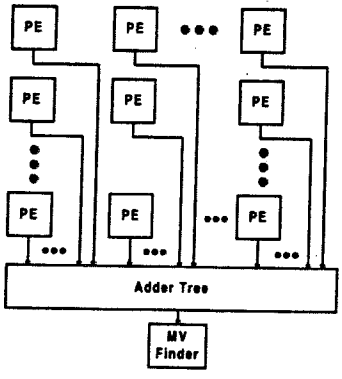

Fig 2: Data path of AD port.

2.2. Data path of AD port

The absolute difference (AD) port outputs the absolute difference of two pixels. These data are then summed to form the MAD of certain candidate location. The summation operation in this architecture is implemented as an adder tree. This is shown in Fig 2. In Fig 2 only the interconnections of AD ports of PE modules are shown. After the adder tree a motion vector (MV) finder follows for sequentially comparing two MADs obtained from adder tree and finds the smaller one and its corresponding motion vector.

2.3. Data path of A_out, A_in_L & A_in_U ports

There are two types of current block data in this architecture, active data and preload data. Active data are those current block data currently being used for calculating MADs. A_out, A_in_L, and A_in_U ports are for this active data type. These active data run through PE modules following a ring like path in both horizontal and vertical directions. The data path of active data is shown in Fig 3. In Fig 3 only the interconnections of three active data ports are shown. Almost all of these interconnections are connections between neighboring PE modules, except for the boundary ones. Each PE module receives active data propagated from neighboring PE modules in two directions. Multiplexer is used to select the correct one.

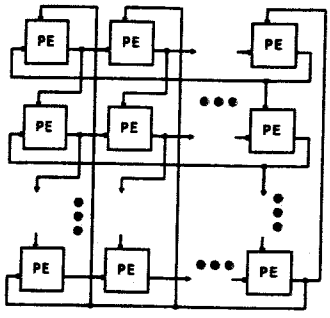

Fig 3: Data path of A_out, A_in_L, and A_in_U ports.

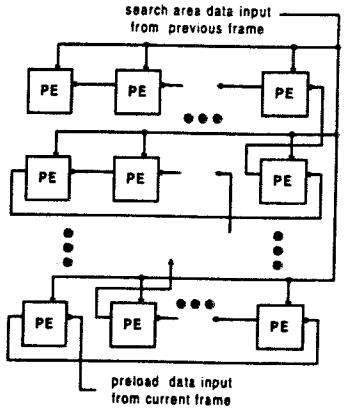

Fig 4: Data path of P_in, P_out and SA ports.

2.4. Data path of P_in, P_out & SA ports

Besides active data, preload data are also current block data. But preload data are not currently used, they are prefetched from frame memory for the preparation of next macroblock task in order to achieve a fluent execution during the transition between current task and next task. P_in and P_out ports are for this preload data type. These preload

data are propagated in a single non-intersect path among PE modules. This is shown in Fig 4. Preload data are normally propagated through preload registers (PR) and pumped into active registers (AR) in parallel at the transition of two tasks.

The search area data are inputted from previous frame through SA ports and stored in SAMs within PE modules. The data path of SA ports are shown in Fig 4. There are only two input ports from external frame memory, one for current block data and the other for search area data. Both of them are shown in Fig 4.

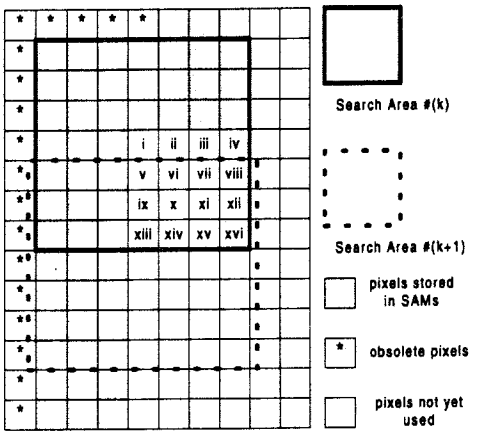

Fig 5: Execution operation example.

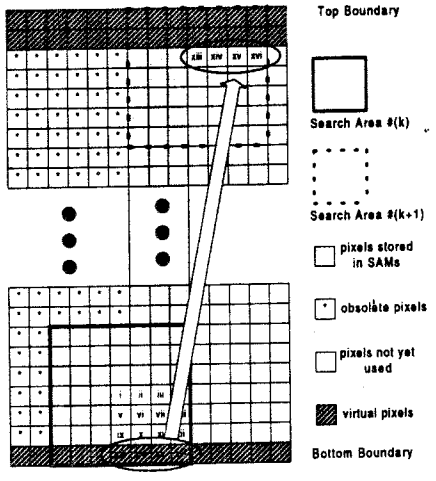

Fig 6: Bottom boundary task.

2.5. Local memory organization

Search area data loaded from frame memory are averagely distributed in SAMs without redundancy placement. If each pixel is numbered with indexes (i,j), where $0 \leq i < W$, $0 \leq j < H$, and there are $N \times N$ PE modules, each of them is indexed by (p, q), where $0 \leq p, q < N$, both of the indexes are ordered from left to right, top to bottom, then pixel (i, j) is fixed to be stored in SAM within PE module $(Res(\frac{i}{N}), Res(\frac{j}{N}))$, where $Res(\cdot)$ returns the residual of operand. Under the scan-line block wise execution order assumption the theoretical local memory size = $H \times (SR-1)+(SR-1) \times N$. But N overhead are introduced in our architecture. So the total size of local memory is $H \times (SR-1)+SR \times N$, where SR is the search range.

3. THE EXECUTION OPERATION

The execution operation is illustrated using a simple example of N=SR=4 in Fig 5. Each little square represents a pixel in previous frame. In Fig 5 we can see that in the beginning there are 16 unloaded pixels marked from i to xvi. While processing the upper left candidate block in the beginning of task #(k), pixel i is required but not loaded. So this pixel must be loaded on-the-fly serving as active data for calculating MAD and stored into corresponding SAM at the same time. While processing the second block (to the right of last candidate block), pixel ii is again required but not loaded. The same operations described above repeat for not only pixel ii but all other 14 pixels. One pixel of previous frame is loaded at each clock cycle. After 16 clock cycles all candidate location have been evaluated and those 16 pixels, i, ii, ... xvi, have been loaded and stored. The task #(k) is completed without idle cycles. The same situation applies to all other tasks.

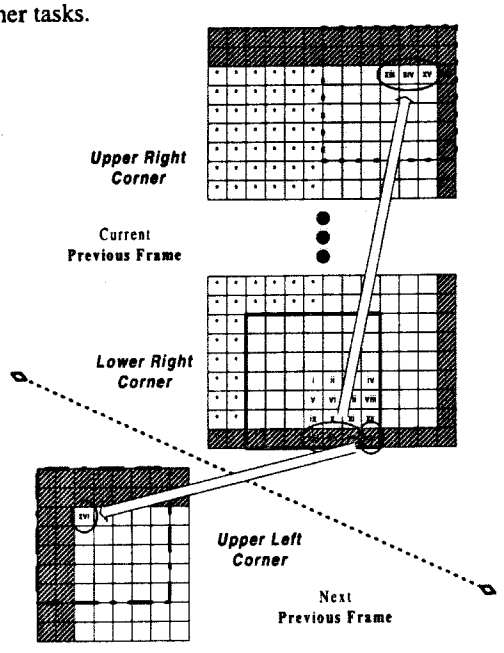

Fig 7: Right boundary task.

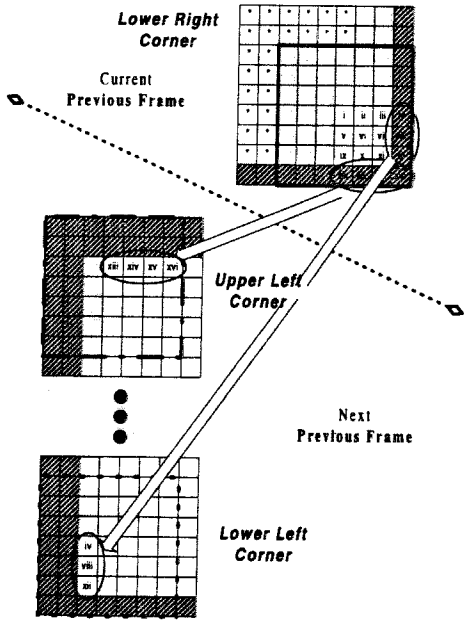

Fig 8: Lower right boundary task.

Some tricks must be considered in order to make sure 100% hardware efficiency while boundary tasks encountered. These tricks are based on the concepts of loading real pixels instead of virtual pixels. Real pixel pixels means those pixels within W × H region in a frame. Virtual pixels means those pixels outside frame, which are virtually existed due to the extension of search range. When processing boundary tasks, virtual pixels become part of the search area. The clock cycles which are originally used to load these virtual pixels can now be used for load real pixels of next tasks. There are three kind of boundary tasks, bottom boundary task shown in Fig 6, right boundary task shown in Fig 7, and lower right corner boundary task shown in Fig 8. With the consideration of boundary tasks the hardware efficiency achieves 100%.

4. COMPARISON

Table 1 and Table 2 are the comparisons between proposed architecture and other presented architectures. In the comparisons the HDTV video format is used to show the impact of large frame size. In Table 1 N = 16 and SR = 16, in Table 2 N = 16 and SR = 32 are used to show the affection of large search range. The comparisons are focused on frame memory bandwidth, input pin count and hardware utilization. Each pixel is assumed to be 8 bits wide. The Bandwidth is measured in megabyte per second.

From Table 1 and Table 2 we can see that our architecture maintains the same low memory bandwidth and input pin count when the search range changes but still achieve 100% hardware utilization. This property is quite suitable for high quality video format.

Table 1. Comparison with N=16, SR=16.

Arch.	# of PE	BW	Input Pin Count	Utilization
[1]	256	188	24	100%
[2]	256	2,005	136	52%
[3]	256	298	16	28%
[4]	256	188	24	100%
[8]	256	16,105	2056	100%
Our	256	125	16	100%

Table 2. Comparison with N=16, SR=32

Arch.	# of PE	BW	Input Pin Count	Utilization
[1]	1024	250	32	100%
[2]	256	5,954	136	68%
[3]	256	603	16	47%
[4]	1024	564	72	100%
[8]	256	64,235	2056	100%
Our	256	125	16	100%

5. CONCLUSION

In this paper we describe a new architecture of FS-BMA. In summary, this architecture has the following features: 1). Minimum memory bandwidth requirement. 2). Minimum I/O pin count. 3). 100% hardware efficiency. 4). Power consumption reduction due to bus transition reduced. 5). Simple and regular interconnections. These features enable this architecture to be suitable for high quality video motion estimation. This is because high quality video motion estimation demands unreasonable high memory bandwidth without special design. The size of local memory is also reduced to near minimum value. Besides, the costs of external frame memory modules can also be reduced. In a complete video compression system, other components can obtain more bandwidth owing to the memory bandwidth requirement of motion estimation is reduced.

REFERENCE

[1] Luc De Vos, Michael Stegherr, "Parameterizable VLSI Architectures for the Full-Search Block-Matching Algorithm", IEEE Transactions on Circuits and Systems, vol.36, no.10, pp. 1309-1316, Oct 1989.

[2] Thomas Komarek, Peter Pirsch, "Array Architectures for Block Matching Algorithms", IEEE Transactions on Circuits and Systems, vol.36, no.10, pp. 1302-1308, Oct 1989.

[3] Chaur-Heh Hsieh, Ting-Pang Lin, "VLSI Architecture for Block-Matching Motion Estimation Algorithm", IEEE Transactions on Circuits and Systems for Video Technology, vol.2, no.2 pp. 169-175, Jun 1992.

[4] Hangu Yeo, Yu Hen Hu, "A Novel Modular Systolic Array Architecture for Full-Search Block Matching Motion Estimation", IEEE Transactions on Circuits and

Systems for Video Technology, vol.5, no.5, pp. 407-416, Oct 1995.

[5] Sung Bum Pan, et al., "VLSI Architectures for Block Matching Algorithms Using Systolic Arrays", IEEE Transactions on Circuits and Systems for Video Technology, vol.6, no.1, pp. 67-73, Feb 1996.

[6] Jianhua Lu, Ming L. Liou, "A Simple and Efficient Search Algorithm for Block-Matching Motion Estimation", IEEE Transactions on Circuits and Systems for Video Technology, vol.7, no.2, pp. 429-433, Apr 1997.

[7] Shifan Chang, et al., "Scalable Array Architecture Design for Full Search Block Matching", IEEE Transactions on Circuits and Systems for Video Technology, vol.5, no.4, pp. 332-343, Aug 1995.

[8] Yeong-Kang Lai, et al., "A Novel Scalable Architecture with Memory Interleaving Organization for Full Search Block-Matching Algorithm", 1997 IEEE International Symposium on Circuits and Systems, pp. 1229-1232, Jun 1997.

[9] Yeu-Shen Jehng, et al., "An Efficient and Simple VLSI Tree Architecture for Motion Estimation Algorithms", IEEE Transactions on Signal Processing, vol.41, no.2, pp. 889-900, Feb 1993.

[10] Jaswant R. Jain, Anil K. Jain, "Displacement Measurement and Its Application in Interframe Image Coding", IEEE Transactions on Communications, vol.COM-29, no.12, pp. 1799-1808, Dec 1981.

[11] T. Koga, et al., "Motion compensated interframe coding for video conferencing", Proc. Nat. Telecommun. Conf., New Orleans, LA, pp. G5.3.1-5.3.5, Nov. 29-Dec. 3, 1981.

[12] Kun-Min Yang, et al., "A Family of VLSI Designs for the Motion Compensaton Block-Matching Algorithm", IEEE Transactions on Circuits and Systems, vol.36, no.10, pp. 1317-1325, Oct 1989.