International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:12, 2007

New VLSI Architecture for Motion Estimation Algorithm

V. S. K. Reddy, S. Sengupta, and Y. M. Latha

***Abstract*—**This paper presents an efficient *VLSI* architecture design to achieve real time video processing using Full-Search Block Matching *(FSBM)* algorithm. The design employs parallel bank architecture with minimum latency, maximum throughput, and full hardware utilization. We use nine parallel processors in our architecture and each controlled by a state machine. State machine control implementation makes the design very simple and cost effective. The design is implemented using *VHDL* and the programming techniques we incorporated makes the design completely programmable in the sense that the search ranges and the block sizes can be varied to suit any given requirements. The design can operate at frequencies up to 36 MHz and it can function in *QCIF* and *CIF* video resolution at 1.46 MHz and 5.86 MHz, respectively.

***Keywords*—**Video Coding, Motion Estimation, Full-Search, Block-Matching, *VLSI* Architecture.

I. Introduction

WITH the growing need of real-time video applications, video compression plays a vital role in achieving bandwidth efficiency for both transmission and storage and efficient motion estimation is a key factor for achieving enhanced compression ratio. However, motion estimation involves high computational complexity, causing bottleneck in the real-time applications. To meet real-time processing needs, several motion vector search strategies and hardware designs have been proposed. These primarily focus on reducing the number of Sum-of-Absolute-Difference *(SAD)* operations at the cost of controller complexity. One of the main design goals is to reduce the computational complexity and power consumptions, without sacrificing image quality. Some algorithms and architectures succeeded in reducing power consumption and satisfied the required performance.

The simplest and most effective method of motion estimation is to exhaustively compare each *N*x*N* macro block of the current frame with all the candidate blocks in the search window defined with in the previous processed frame and find the best matching position with the lowest distortion. This is called Full Search Block Matching algorithm *(FSBM)* algorithm. A full search block matching process with a search range *p* has a search window of size *(2p+N)* x *(2p+N)* pixels and a total of $(2p+1)^2$ candidate blocks in the reference frame for each block of the current frame. The distortion values are computed for each of the candidate blocks and its minimum value is found from the set of $(2p+1)^2$ candidate blocks. The distortion measure is Sum of Absolute Difference for its simplicity, in which the candidate block with minimum amount of distortion is considered as the best-match.

To achieve a best trade-off between the computational complexity of *FSBM* and degraded *PSNR* of motion compensated frame using faster algorithms, recently some researchers have investigated reduction of computational complexities of *FSBM* [1]-[5]. All these algorithms are not optimal in the sense that instead of exhaustive search, only some fixed positions are searched, based on the predictions of motion. Any error in motion prediction may lead to wrong motion vectors, resulting in poor peak signal-to-noise ratio *(PSNR)* of the motion-compensated frame.

The computationally intensive nature of *FSBM* and the demand of real-time processing render the *VLSI* implementation of *FSBM* is a necessity. Due to the repetitive nature of the algorithm and demand for high throughput, several *VLSI* architectures of *FSBM* were implemented [6]-[10], but most of them are based on dataflow management and not suitable for good throughput/area efficiency and low power *VLSI* implementations. The main draw back of these architectures is that considerable extra hardware is required estimating the motion vector. Luc De Vos&M.Schobinger [11] and others [12]-[13], suggested processor based architectures, offering high flexibility and programmability, but compared to *ASICs* [14]; it offers higher power consumption and lower throughput. In this paper we propose efficient and low power *VLSI* architecture, which has been developed to meet the speed requirement of the current video coding system.

In this paper we propose a programmable, low power, high throughput and efficient architecture that outperform the state of the art, making practical implementation of *FSBM* in multimedia terminals feasible. The rest of the paper is organized as follows. Section II Presents the proposed parallel bank architecture and blocks description. Computational Complexity and Clock Rate Requirements is presented in Section III. Section IV concludes the paper with directions for

Manuscript received January 8, 2007. This work was supported in part by the World Bank Project under Grant for TEQIP.

Vustikayala Sivakumar Reddy is with the Sreenidhi Institute of Science and Technology, JNT University, Hyderabad, 501301, India (phone: 91-8415-223001; e-mail: vskreddy2003@yahoo.com).

Somnath Sengupta is with the Department of Electronics and Electrical Communication Engineering, Indian Institute of Technology, Kharagpur, 721302, India (phone: 91-3222-255321; e-mail: ssg@ece.iitkgp.ernet.in).

Y. Madhavee Latha is with the G. Narayanamma Institute of Technology and Science, JNT University, Hyderabad, 500008, India (phone: 91-40-23565648; e-mail: madhuvsk2003@yahoo.co.in).

future work.

II. Proposed Parallel Bank Architecture

We propose our general motion estimation *VLSI* architecture. It is divided into four parts, i.e., computation core, address generator, on-chip buffer, and off-chip memory, as shown in Fig. 1. The computation core consists of many processing elements *(PE),* which are connected into *1D* array. The address generator and data mapper unit is a bridge between data path and off-chip memories. It generates the address for the data which will be needed in the next search step, then accesses the memories and fetches them into the on-chip buffers or directly loads them into the processing elements.

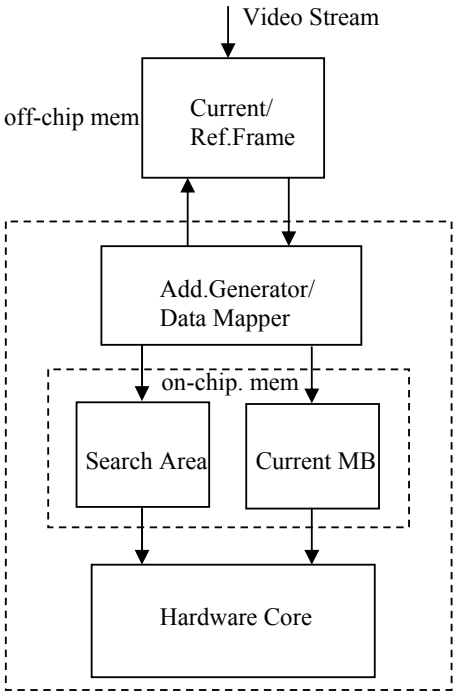

Fig. 1 Block diagram of the motion estimation architecture

We present a flexible, simple and yet a very efficient architecture for Motion Estimation based on partial search algorithm. We use nine parallel processors is employed each controlled by a state machine. The state machine employed is a Moore model Finite State Machine (*FSM)* and it makes the design very simple and cost effective. The execution of all the 9 processors in parallel makes it a faster approach. The design is implemented in *VHDL* and the programming techniques we incorporated makes the design completely programmable in the sense that the search ranges and the block sizes can be varied to suit any given requirements. The blocks occupying *9* parallel processors as shown in Fig.2 and each processor perform computations on different set of data in parallel. The basic approach is as follows:

The *176*x*144* frame is divided into *99* blocks of each *16*x*16* size. These *99* blocks can be arranged in *11*x*9* block matrix. Each row of *11* blocks is given to each processor for MV computation as shown in Fig. 2. Thus, there are *9* processors working in parallel with each processing *11* blocks individually. A unique number called the Index identifies each block, thus *block index number* that we use frequently in our project description refers to the starting number of the block.

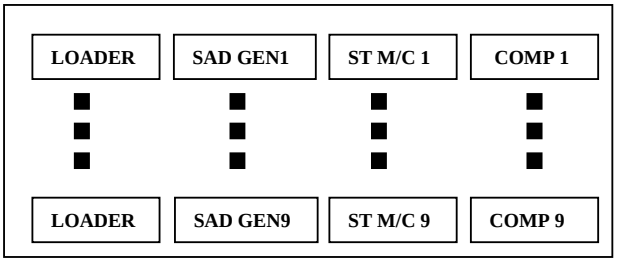

Fig. 2 Parallel Processor Architecture

Each processor internally contains four modules: Search and Block loader, *SAD* Generator, Comparator and a State Machine controlling all the three modules. The interconnection between the modules is shown in Fig. 3.

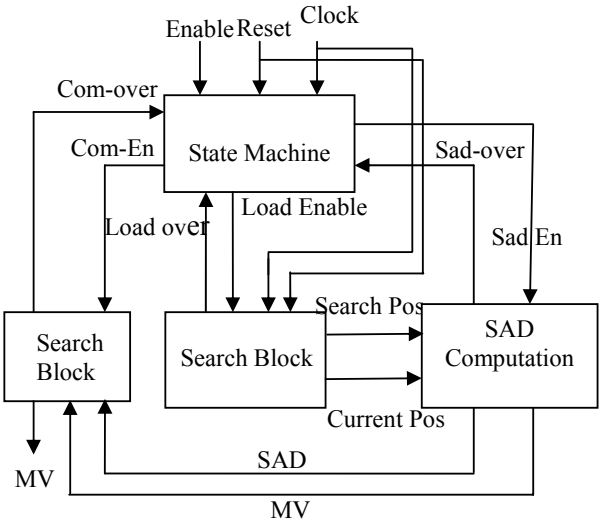

Fig. 3 Proposed Architecture

A. Search and Load Module

This module outputs the index numbers of the blocks of the previous frame that have to be compared with each of the current frame blocks with candidate blocks based within the search range. The state machine module enables this module immediately after the idle state.

For each current frame block there are $(2p+1)^2$ candidate blocks within its search area range, all these block indexes are generated. The outputs indicate the current block index number and previous frame block index.

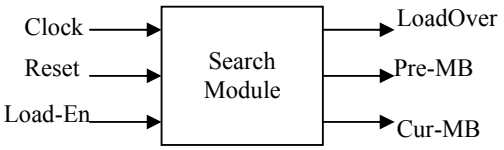

Fig. 4 Block Diagram of Search and Load Module

- Clock : Clock Signal
- Reset : Reset signal for the module
- Load-En : Enable signal for this module
- Pre-MB : Signal Indicates the block index number of previous frame
- Cur-MB : This signal indicates the block index number of current frame

B. Search and Load Module

This module gets its inputs from Search and Loader module and gives outputs to Comparator module. It is activated or enabled by the State Machine module. The inputs indicate the block indices and this module calculates the sum of absolute difference of these two blocks by obtaining the pixel values from the memory. This module finally outputs the *SAD* of the two blocks along with the block indices.

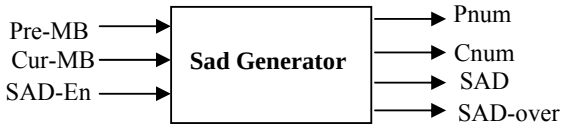

Fig. 5 Block Diagram of SAD Generator Module

- SADEn : Enable signal for this module.
- Pre-MB : This signal indicates the block index number of previous frame.
- Cur-MB : Signal indicates the block index number of current frame.
- Pnum : Index number of the block of previous frame generated the SAD.
- Cnum : Index number of the block of current frame.
- SAD : SAD value that is generated by two blocks with index numbers Pnum and Cnum.
- SAD-over : Signal that indicates the control logic that operation of the block is completed.

C. Comparator Module

This module gets its inputs from *SAD* Generator module and outputs are the final outputs. It is activated or enabled by the State Machine module. It collects all the *SADs* corresponding to one current block and finds minimum value and outputs the block indexes that generated this *SAD.*

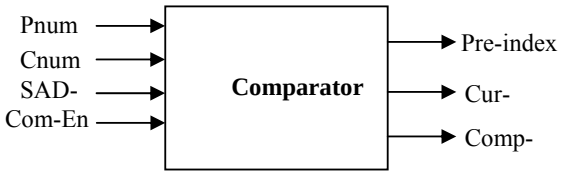

Fig. 6 Block Diagram of Comparator Module

- Com-En : Enable signal for this module
- Pnum : Index number of the block of the previous frame
- Cnum : This is the index number of the block of current frame which comes from the *SAD* Generator.
- SAD-in : This is the *SAD* value that is generated by two blocks with index numbers Pnum and Cnum coming from the *SAD* generator block.
- Pre-index : This signal indicates the block index number of previous frame that has given the least *SAD* on comparison with all the *SAD* inputs.
- Cur-index : This signal indicates the block index number of current frame that has given the least *SAD* on comparison with all the *SAD* inputs.
- Com-over : This signal indicates the control logic that operation of the block is completed.

D. State Machine

State machine module is the main module that controls all the modules. There are four states: Idle state, Block loader state, SAD Generator state and Comparator states.

Idle-state: This is the initial state on reset.

Block loader state: This state is the immediate state after the idle state. The enable signal for the Block loader state is asserted.

SAD Generator state: This state is after the Block loader state and the enable signal for the *SAD* Generator state is asserted.

Comparator state: This state is entered after the *SAD* Generator state. Here the enable signal for the Comparator state signal is asserted.

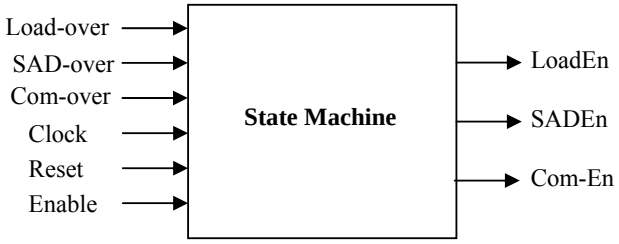

Fig. 7 Block Diagram of Search and Load Module

- Clock : Input is the clock signal for this module
- Reset : Reset signal for the module
- Enable : Enable signal for this module
- Load-over : Handshaking signal from Search and Loader module.
- SAD-over : Handshaking signal from SAD-Generator module.
- Com-over : Handshaking signal from Comparator module.
- LoadEn : Handshaking signal from Search and Loader module that enables the block.
- SADEn : Handshaking signal from SAD Generator module that enables the block.
- Com-En : Handshaking signal from Comparator module that enables the block.

III. Computational Complexity and Clock- Rate Requirements

The *QCIF* frame of 176x144 pixels accessed from memory and processed by the individual modules. The search and loader module extracts the blocks that are within the search range of *(2P+1)* x *(2P+1),* where search range *P* is taken 7. Then the *SAD* Generator module computes the Sum of Absolute Difference between the blocks of one frame with the other. The Comparator gets the *SAD* value from the *SAD* Generator module and compares the *SAD*s and selects the least *SAD* and the blocks that generated the minimum *SAD* are the blocks that are given as output so that the prediction error can be computed from the blocks.

To analyze the clock frequency of the architecture, we define the following quantities:

Size of the frame: $N_h \times N_v$

No. of frames per second: N_{fs}

Size of the block: $N \times N$

Full search range: $\pm p$ pixels

Number of blocks per frame: $B = \frac{N_h \times N_v}{N^2}$

No. of candidate blocks to be matched for one current block: $N_c = (2p)^2$

No. of blocks to be matched per frame: $B \times (2p)^2$

No. of pairs of blocks required to be matched per second: $B \times (2p)^2 \times N_{fs}$

No. of computations required to be performed per second: $B \times (2p)^2 \times N_{fs} \times N_B^2$

Total number of clocks required per second: $B \times (2p)^2 \times N_{fs} \times N_B^2$

No. of process elements used in the architecture: N_{PE}

Therefore, the minimum clock frequency f_c is expressed as

$$f_c = \frac{B \times (2p)^2 \times N_B^2 \times N_{fs}}{N_{PE}}$$

IV. Conclusion

A low cost *VLSI* architecture to compute the motion vectors required by the *H.264* video coding standard is presented in this paper. The current implementation uses fixed size windows for the small and large motion regions. In the future decreasing the search range can further reduce the computations. The design can operate at frequencies up to 36 MHz and it can function in *QCIF* and *CIF* video resolution at 1.46 MHz and 5.86 MHz respectively. The proposed architecture is easily scalable and parallel implementations can be efficiently realized to obtain higher speed with a reasonable increase in hardware resources requirement.

References

[1] Kuo-Liang Chung and Lung-Chun Chang, "A New Predictive Search Area Approach for Fast Block Motion Estimation", IEEE Transactions on Image Processing, vol. 12, no. 6, June 2003.
[2] Michael Brunig and Wolfgang Niehsen, "Fast Full-Search Block Matching", IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, No. 2, Feb. 2001.
[3] J. R. Jain and A. K. Jain, "Displacement measurement and its application in inter-frame image coding," IEEE Transactions on Communications, vol. 29, pp. 799-808, Dec. 1981.
[4] Vasily G. Moshnyaga "A New Computationally Adaptive Formulation of Block-Matching Motion Estimation" IEEE Transactions on Circuits and Systems for Video Technology, vol. 11, no. 1, Jan. 2001.
[5] L.M. Po and W.C. Ma, "A novel four-step search algorithm for fast motion estimation," IEEE Transactions on Circuits and Systems for Video Tech., pp.313-317, June 1996.
[6] C. H. Hsieh and T. Lin, "VLSI architecture for block-matching motion estimation algorithm", IEEE Transactions on Circuits and Systems for Video Technology, vol. 2, pp. 169-175, June 1992
[7] T. Komarek, and P. Pirsch, "Array architectures for Block-Matching Algorithms," IEEE Transactions on Circuits and Systems, vol. 36, pp. 1301-1308, Oct. 1989
[8] L. Do. K. Yun, "A Low-Power VLSI Architecture for Full-Search Block-Matching Motion Estimation," IEEE Transactions on Circuits and Systems for Video Technology, vol. 8, pp. 393-398, Aug 1998
[9] K.M. Yang and L. Wu, "A Family of VLSI Design for the motion compensation Block-Matching Algorithm", IEEE Transactions on Circuits and Systems, vol. 36, no. 10, pp. 1317-1325, Oct. 1989
[10] Seung Hyun Nam and Moon Key Lee, "Flexible VLSI architecture of motion estimator for video image application," IEEE Transactions on Circuits and Systems-II, vol. 43, no. 6, pp.467-470, June 1996
[11] Luc de Vos & M. Schobinger, " VLSI Architecture for a Flexible Block Mathing Processor", IEEE transactions on Circuits and Systems for Video Technology, vol. 5 no. 5 pp. 417-428. Oct. 1995.
[12] Rizzo D and Colavin O, **"**A video compression case study on a reconfigurable VLIW architecture, Europe Conference and Exhibition in Design, Automation and Test, P. 540 -546, Mar. 2002
[13] Gao R, Xu D and Bentley J.P, **"**Reconfigurable hardware implementation of an improved parallel architecture for MPEG-4 motion estimation in mobile applications", IEEE Transactions on Consumer Electronics, Vol. 49, P.1383-1390, Nov. 2003
[14] H. Fujiwara et al. "An All-ASIC Implementation of Low Bit-Rate Video Decoder," IEEE Transactions on Circuits and Systems, June 1992