2010

Reconfigurable Architecture For H.264/avc Variable Block Size Motion Estimation Based On Motion Activity And Adaptive Search Range

Sumedha Kodipyaka
University of Central Florida

RECONFIGURABLE ARCHITECTURE FOR H.264/AVC VARIABLE BLOCK SIZE MOTION ESTIMATION BASED ON MOTION ACTIVITY AND ADAPTIVE SEARCH RANGE

by

SUMEDHA GUPTA KODIPYAKA
B.E. Osmania University, 2007

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2010

Major Professor

Jooheung Lee

ABSTRACT

Motion Estimation (ME) technique plays a key role in the video coding systems to achieve high compression ratios by removing temporal redundancies among video frames. Especially in the newest H.264/AVC video coding standard, ME engine demands large amount of computational capabilities due to its support for wide range of different block sizes for a given macroblock in order to increase accuracy in finding best matching block in the previous frames.

We propose scalable architecture for H.264/AVC Variable Block Size (VBS) Motion Estimation with adaptive computing capability to support various search ranges, input video resolutions, and frame rates. Hardware architecture of the proposed ME consists of scalable Sum of Absolute Difference (SAD) arrays which can perform Full Search Block Matching Algorithm (FSBMA) for smaller 4x4 blocks. It is also shown that by predicting motion activity and adaptively adjusting the Search Range (SR) on the reconfigurable hardware platform, the computational cost of ME required for inter-frame encoding in H.264/AVC video coding standard can be reduced significantly.

Dynamic Partial Reconfiguration is a unique feature of Field Programmable Gate Arrays (FPGAs) that makes best use of hardware resources and power by allowing adaptive algorithm to be implemented during run-time. We exploit this feature of FPGA to implement the proposed reconfigurable architecture of ME and maximize the architectural benefits through prediction of motion activities in the video sequences ,adaptation of SR during run-time, and fractional ME refinement. The implemented ME architecture can support real time applications at a maximum frequency of 90MHz with multiple reconfigurable regions.

When compared to reconfiguration of complete design, partial reconfiguration process results in smaller bitstream size which allows FPGA to implement different configurations at higher speed. The proposed architecture has modular structure, regular data flow, and efficient memory organization with lower memory accesses. By increasing the number of active partial reconfigurable modules from one to four, there is a 4 fold increase in data re-use. Also, by introducing adaptive SR reduction algorithm at frame level, the computational load of ME is reduced significantly with only small degradation in PSNR ($\leq$0.1dB).

To my wonderful parents, Chandra Sekhar and Vishalakshmi.

ACKNOWLEDGMENTS

I would like to first thank my parents, Chandra Sekhar and Vishalakshmi, and my brother, Siddharth, who have been with me in every step of my life, helping me to succeed and instilling in me the confidence that I am capable of doing anything I put my mind to.

Special thanks to you my friend, Hari, for your practical and emotional support, helping me to consistently keep up to the competing demands of work, study and personal development.

I would also like to thank Dr. DeMara and Dr. Wang for their review and encouragement as Committee Members.

Most of all, I would like to thank Dr. Jooheung Lee for his excellent technical inspiration and editorial suggestions that helped me shape up this dissertation. Thank you for being a constant support as a mentor, right from the day one of my MS studies.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

1.1 Video Coding Standard

Digital video compression is widely used and plays an integral part in providing 'network friendly' video representation. The technology progress in the fields of digital multimedia and wireless communication systems has demanded the development of several video compression standards over past two decades. The main goal of these standards is to develop efficient video coding design with improved picture quality and high rate-distortion efficiency. The H.264/AVC is the latest video coding standard developed by the ITU-T Video Coding Experts Group and the ISO/IEC Moving Picture Experts Group in 2003 [1], [2].

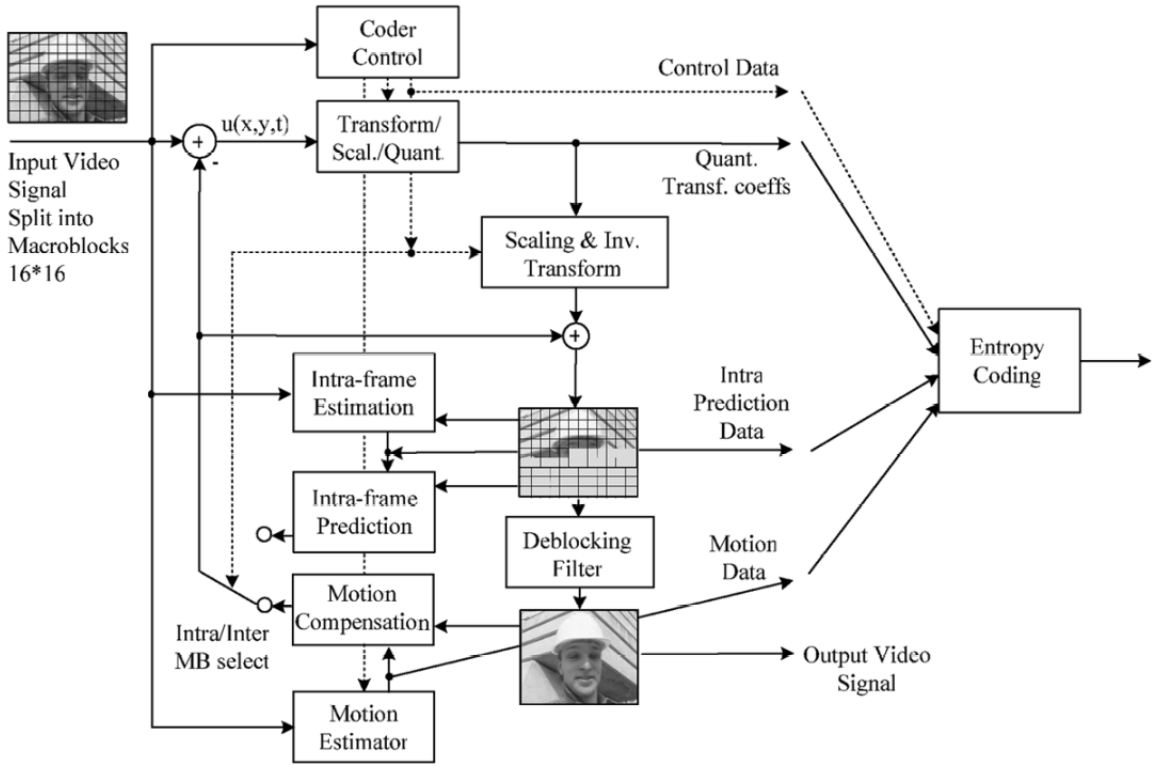

Figure 1 Block diagram of hybrid encoding process for the H.264/AVC video coding standard [1].

Compared to previous video coding standards, such as H.261/3 and MPEG-1/2/4, H.264/AVC provides many advanced coding techniques, such as integer DCT transform, intra prediction in the spatial domain, multiple reference pictures, variable block size motion estimation and compensation, context adaptive variable length coding, and context adaptive binary arithmetic coding to achieve higher coding efficiency. The standard related documents and reference software can be found in [3].

The H.264/AVC technique is based on hybrid video coding process. The general block diagram for hybrid H.264/AVC encoding process is shown in Figure 1. A given video is encoded frame by frame and each input frame is divided into several macroblocks. Each macroblock consists of 16x16 pixel data which is coded in Intra or Inter mode. In inter mode, the macroblock is predicted using Motion Vectors (MVs) which correspond to the displacement of the current block from its corresponding position in already coded previous frame(s). In Intra mode, the frame is coded without reference frame. The prediction scheme for Intra mode uses information of previous blocks of the same frame. In inter mode, the predictions error is transformed using integer transform, quantized and then the calculated coefficients are encoded. For frame reconstruction, the quantized coefficients of each block are inverse transformed at the decoder side. Since the encoding is done block by block, the reconstructed frame will have visible block structures. In order to reduce this blockiness, H.264/AVC introduces an in-loop deblocking filter. After the filter, each macroblock is completely decoded and stored for further processes.

1.2 Motion Estimation Technique

Motion Estimation (ME) is a part of Motion Compensation prediction which provides the best MV and distortion data of all possible modes of a given macroblock. The motion prediction of macroblock is performed using the information of already transmitted previous image as a reference.

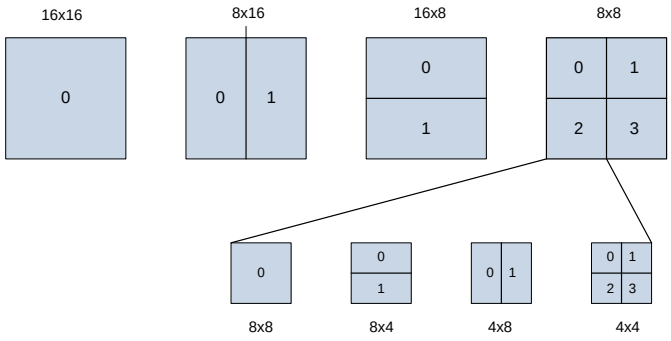

Figure 2 Partitioning of macroblock in seven different block sizes

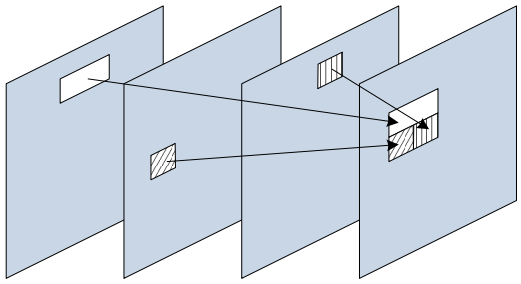

Figure 3 Motion Estimation with multiple reference frames.

ME techniques play a key role in the video coding systems to achieve high compression ratios by removing temporal redundancies among video frames. Especially in the H.264/AVC standard, ME engine demands large amount of complex computational capabilities due to its support for wide range of different block sizes for given macroblock in order to increase accuracy in finding best matching block. The partitions of a macroblock and sub-macroblock are shown in Figure 2. Also, in H.264/AVC, it is possible to refer to multiple reference frames. Figure 3. shows the concept of motion compensated prediction with multiple reference frames. It is shown that more than 60% of the video encoders computational time is consumed by the ME module [4].

The macroblock is partitioned as 16x16, 16x8, 8x16, and 8x8 block sizes. The 8x8 sub-block, can be further divided into partitions with block sizes 8x4, 4x8, and 4x4. Introduction of such smaller

blocks can more accurately find the best matching block match in the previous frame and also reduce the resultant residual errors to be encoded [5]. Hence, Variable Block Size Motion Estimation (VBSME) can achieve higher coding performance at the cost of increase in computational complexity. Therefore, efficient and hardware-friendly VBSME architectures are critical for high-performance video encoder.

In Full Search Block Matching Algorithm (FSBMA), for each macro block (MB) in the current frame, the most similar MB within all possible locations in the Search Range (SR) in the reference frame is chosen. Sum of Absolute Differences (SAD) is a commonly used matching criterion for ME algorithm [6]. For a block-based ME, its basic functions are to calculate and compare the cost function used as a matching criterion between the current image block and all candidate blocks in the search range of reference frame. Let the block size be N×N and location of each block in the current frame (C) is represented by (i, j). This block must be matched with a block within the search window (h, v) in the reference frame (R). SAD of such searching candidate block is given by

$$SAD_{(i,j)}(x,y) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\left|C_{(i,j)}(x,y) - s\_w_{(i,j)}(x1+x+h, y1+y+v)\right| \qquad (1)$$

Where, c(x,y) and s_w(x,y) represent pixel value in current block and search candidate block respectively. (x1, y1) represents center of the search range for the given macroblock in the reference frame. While computationally expensive, FSBMA offers high encoding efficiency, very regular computational algorithm, and hence good visual quality. In order to speed-up this process, data-adaptive ME algorithms have been introduced to modify the search center location by adaptively forming correlations among neighboring MVs.

1.3 Partial Reconfiguration and its Benefits

Field Programmable Gate Arrays (FPGAs) are digital Integrated Circuits (ICs) that contain configurable logic blocks and interconnects. FPGAs are being targeted in many applications, such as high performance signal processing applications, to provide real-time computing capabilities. This is due to the fact that they incur very low-to-none Non-Recurring Engineering (NRE) costs and faster development time. Each generation of FPGAs are made significantly useful by introducing additional benefits and utilities with larger size and faster speed. Among such significant advancements is Dynamic Partial Reconfiguration [7]. Their main feature of partial reconfiguration process is to reconfigure a part of the device while the other parts of the FPGA are still active.

In SRAM-based FPGAs, initialization of device involves loading the device with configuration data or configuration bitstream in order to make FPGA perform some task. This configures all the logic blocks, interconnects and Input/Output (I/O) interfaces. FPGAs can be partially reconfigured by loading it with partial configuration bitstream file. The partial bitstreams can be generated by two methods: difference-based and modular-based processes [8]. In difference-based method, partial bitstreams contain information of the difference between original and new bitstream file so that only those logic blocks are modified. This method may not apply to real-time adaptive applications as both the original and the new configuration bitstream files may not be always available.

In modular-based partial reconfiguration, specific areas are assigned as Partial Reconfigurable Regions (PRRs) which can have several Partial Reconfigurable Modules (PRMs). Figure 4 shows the design layout of modular-based process. The static and reconfigurable regions communicate through I/O ports called Bus Macros (BMs). When FPGA is loaded with partial bitstreams, PRR region can be modified.

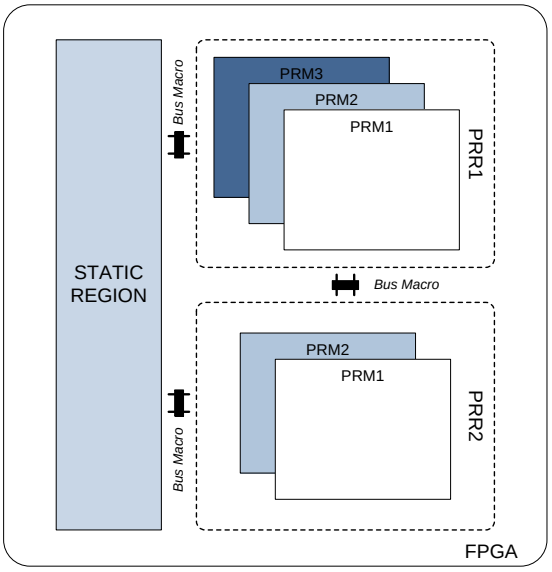

Figure 4 Design layout of static and configurable regions in FPGA. This example consists of two PRRs with three and two PRMS respectively**.**

The main benefits of partial reconfiguration on FPGA are a) the unchanged part is not affected and, in some cases, may continue execution, and b) a partial bitstream size is smaller than a full bitstream and hence lower reconfiguration time. By time-to-time analysis, the modules which are not necessary can be detected and loaded with 'blank' bitstreams to reduce dynamic power consumption. Reconfigurable hardware of FPGA is one solution which can provide benefits of the performance of Application Specific Integrated Circuits (ASICs) and the flexibility of General Purpose Processors (GPPs). They now accommodate digital systems with more than 10 million equivalent gates in its reconfigurable fabric. Its capability to support various design configurations during run-time and dedicated hardware components, such as microprocessors, Digital Signal Processing (DSP) logics, memory blocks, and other specific modules, make FPGAs one of the ideal platforms to implement and test those computationally demanding real-time applications.

1.4 Motivation

Firstly, ME is the most computationally intensive part in the entire video encoding system. In Full Search (FS) method, best MV with minimum SAD is calculated among HxV search positions as discussed in Section 1.2. For example, assuming each pixel undergoes 3 operations for given SR, i.e., absolute difference between current and reference pixel, adding the residues of all the pixels in given macroblock, and accumulate, ME for a QCIF (176x144), 30 frames per second (fps) video sequence with [-8, +7] SR takes 584 Mega operations per second (MOPS) and ME for SD (720x480), 30fps video sequence with [-16, +15], SR takes 32 Giga operations per second (GOPS). So, there is need to speed up ME process which can be done by methods like simplification of search criterion, data-adaptive algorithms, predictive search, etc, at algorithm level and enhanced data re-use, parallelizing SAD computations, etc, at architecture level. Keeping this thought, we develop an approach to gain algorithmic and architectural benefits through a) an adaptive search range reduction algorithm to reduce number of search candidates during run-time and b) a pipelined systolic SAD array architecture to improve performance and data reuse at intra 4x4 block level as well as inter block level.

Secondly, due to the ever increasing complexity of today's chip design, it becomes common to use pre-defined Intellectual Property (IP) cores to simplify the system design and meet the requirements such as manufacturing yield and time-to-market schedule. However, commercial IP cores available in the market are pre-constructed circuits with details of pre-determined area, power, and performance provided by the vendors, and typically target for both Application Specific Integrated Circuits (ASICs) and FPGAs markets. As a consequence, there is a need to develop IP cores which can be easily customizable to avoid mismatches of computing capability between users' applications and the IP cores, and run-time adaptable for time varying loads of computing while reducing design efforts from the users. These features are particularly desirable for real time video

processing systems involving dynamic adaptation to the characteristics of the target applications or other communication systems. Our design approach is towards developing a reconfigurable IP core to perform the computationally complex ME module, so that it can be used in diversified applications with user parameters passed to the control system in the IP core for adjustment of its architecture. We present in details hardware - reconfigurability strategy for development of IP core that can support block matching ME. It is also shown how our ME architecture can adapt with various search ranges (SR), video formats, and frame rates by designing efficient data sharing platform, memory organization, modular approach in hardware design, and different configurations of processing element (PE) arrays during run-time.

1.5 Thesis Organization

Chapter 2 investigates various ME algorithms and architectures. The algorithms presented in this Chapter are mainly categorized as lossy and lossless methods. In lossy algorithms, the video quality is degraded when compared to FSBMA where in lossless methods the results are same as FSBMA. The various architectures discussed mainly consider memory organization, data flow, and latency issues. Chapter 3 presents a modular design strategy for ME hardware architecture to support different video formats and frame rates. This Chapter also discusses the implementation flow using partial reconfiguration on FPGA. As an extension, Chapter 4 presents search reduction algorithm to improve performance of FSBMA and hardware architecture with adaptive SR support. Also, we discuss an approach on time multiplexing reconfigurable area using dynamic partial reconfiguration by integrating the integer and fractional ME architectures. Finally, concluding remarks are given in Chapter 6.

BACKGROUND AND RELATED WORK

2.1 Motion Estimation Algorithms

As discussed in Chapter 1, although its computations are costly, FSBMA is most accurate algorithm. Hence, it is taken as reference for best video quality. In this Chapter, various strategies to reduce the ME computational load and related work are discussed. The methods in Sections 2.1.1, 2.1.2, 2.1.4 are lossy and Section 2.2.3 is lossless [54].

2.1.1 Adaptive Search Locations

In this class of lossy algorithm, ME is performed only on certain search points in order to reduce number of SAD computations. This approach can be classified into three types: a) Careful prediction of motion direction and selecting certain points for which ME is performed in given SR. b) Reducing the search window itself and perform ME in all locations of new SR. c) Hierarchical search with pyramid structure. All these algorithms are based on the assumptions that the motion in sequence of frames is regular and the distortion increases when the search locations move away from the minimum distortion position. Hence, by skipping such search points, computation load on ME is reduced significantly but with degradation of video quality. Some of the examples of type one are, three-step search developed in [9], [10], logarithmic search [11], [12], four step search [13], diamond search [14]. However, the search point selection is not regular and so is the data flow. Therefore, parallelizing the process of SAD computations and efficiency of memory access may not be feasible. The work of [15], considers these issues and shows an approach towards system optimization. For video sequences with large motion, these algorithms perform very poor and are sensitive to local minimum which further degraded ME accuracy.

The second type also called as Predictive Search, can be one of the solutions for the issues in type one algorithms. In this method, first the motion vector is predicted using the motion information of spatial or temporal neighboring blocks. This information along with the distortion (SAD) threshold, decides the final Motion Vector Prediction (MVP). In [16], MV of previous MBs on the top left, top, and top right are taken and their median is calculated. This result, zero MV block, and MV of corresponding MB in previous frames are considered to find MVP. In [17], the motion of MB is categorized as slow, medium, and fast mode to decide the search range.

In hierarchical ME algorithms, a pyramid structure of coarse to fine level search is adopted. First, ME is performed at coarse level and then refined by estimating around the initial MV. Two to three levels of hierarchy are used for finding final MV. The 3SS in [9], can be considered as hierarchical search. In the work of [18], a threshold for SAD value is introduced. The MV refinement is continued for several steps until an MV with SAD value less than the threshold is obtained. ME hardware architecture using hierarchical search method are given in [19], [20].

2.1.2 Cost Reduction of Matching Criterion

In general, SAD block matching scheme involves matching of all pixel in a given block. One way to reduce this is sub-sampling scheme. In the work of [21], only every alternate pixel is considered for matching in both horizontal and vertical direction. This way, the computation load is reduced by four times than typical matching method. In the work of [22], only the edge pixels which are supposed to have important information are considered.

Pixel truncation is another lossy method to reduce hardware cost with small degradation in video quality. For example, 8-bit pixel can be reduce to 4-bits to reduce hardware resources to be used and can also reduce computations. The work of [23] presents an approach where pixel width can be changed adaptively.

2.1.3 Lossless Fast Full Search Algorithm

One direct method of lossless algorithm is Partial Distortion Elimination (PDE) algorithm presented in [24]. If the accumulated partial distortion of candidate block is already larger than existing minimum SAD, the block can be skipped without any loss in image quality. Many other algorithms were developed based on partial distortion elimination method such as normalized PDE [25], probabilistic PDE [26], spiral PDE [27]. These algorithms find an approach to increase the early rejection process.

Successive Elimination Algorithm (SEA) is proposed in [28]. It is based on the mathematical fact that $|A + B| \leq |A| + |B|$ for all real values of A and B. If the absolute difference between sum of current block pixel sum and sum of reference (or candidate) block pixel is greater than existing minimum SAD value, then the corresponding block is eliminated. This is illustrated as

$$SAD(x,y) = \sum_{i=0}^{N} \sum_{j=0}^{N} |C(i,j) - R(i+x, j+y)| \;\geq\; |C_s - R_s| (SEA\ value) \qquad (2)$$

Where C_s sum of is current block pixel and R_s is reference block pixel sum. Therefore, if SEA value is greater than minimum SAD, then it implies that its SAD value is greater than SADmin. SEA combined with good MVP method can increase the rejection ratio of SAD computations.

Winner Update algorithm presented in [29], [30] is based on the idea that we need not require to calculate the total sum in order to eliminate that candidate block. This can be explained by simple strategy employed in the game of poker. The winning player will have minimum sum of his cards. Each player shows one card initially and the player with lowest value card is allowed to show his second card and so on until the player with no cards left is the winner. The basic idea is if the intermediate sum of cards of a particular player already exceeds the total sum of the winning player then that particular player has no chance of winning. The same strategy can be applied to find minimum SAD of candidate blocks. However, implementation of sorting the SAD values is very

expensive in terms of hardware realization. All the algorithms explained in this Section have very minor difference in SAD calculation when compared to FSBMA. However, these minor differences do not cause noticeable effect on video quality.

2.1.4 Mode Decision and Other Encoding Parameters

Mode decision is another important aspect which contributes towards increased computations of H.264/AVC ME. In the JM reference software, all the 16 sub-macroblocks starting with smallest 4x4 blocks are used and exhaustively searches for best mode. With this process, the encoding time for ME is increased significantly. The work in [31], presents a simplified version of mode decision strategy to use reduced number of sub-macroblock modes and effectively saves about 27% of bit-rate and significant computational time for ME with very little degradation in Peak Signal-to-Noise Ratio (PSNR) of the video sequence. In [32], Quantization Parameter (QP) is used for early detection of macroblocks with SKIP mode or with all-zero residue values and eliminates ME computations for that macroblock. Works in [33], [34] discuss about reduction in number of reference frames. All these algorithms are 'software-oriented' and become very complex to realize efficient hardware architectures.

2.1.5 Fractional Motion Estimation Algorithm

The H.264/AVC VBSME consists of seven different block sizes ranging from 16x16 to 4x4. These sub-partitions lead to a large number of possible combinations within each macroblock. In general, large blocks are appropriate for homogeneous areas and small partitions are appropriate for textured and variant motion area. Especially in complex textured pictures, the accuracy of motion compensation is in quarter and half pixel resolution for H.264/AVC, which can provide better compression performance [3], [48]. The half-pixel MV refinements are performed around the best integer search positions, I, from IME results. The search range of half-pixel MV refinements is ±1/2

pixel along both horizontal and vertical directions. This refinement has nine candidates, including the refinement center and its eight neighborhoods, for the best match. Then, quarter pixel refinement is performed in same fashion as half-pixel. The inter mode decision is done after all costs are computed in half-pixel and (or) quarter-pixel precision in all reference frames.

The reference pixels are interpolated to produce fractional pixels for each search candidate. Afterward, residues are generated by subtracting the corresponding fractional pixels from current pixels. Then, the absolute values of the 4×4-based residues are accumulated as distortion cost called SAD. The final matching cost is calculated by adding the SAD with the MV cost. The cost can be correctly derived only after prediction modes of the neighboring blocks are determined.

2.2 Motion Estimation Hardware Architectures

2.2.1 FSBMA Architectures

Integer pixel Full Search ME algorithm has an inherent property of common data dependency, which makes pipelining of such data possible in order to speed up the ME computations. Based on this algorithm, some efficient full search algorithms are proposed having 1-D and 2-D PE structures. In general, ME hardware architectures can be classified into two types: inter-level SAD architecture where each processing element (PE) is responsible for complete SAD for specific search location and intra-level SAD architecture where each PE is responsible for the partial absolute difference of a fixed current pixel location in the current MB, but for all search locations. In intra level architecture, the PE matrix size is confined to the MB size, hence occupies less area and further can be pipelined to design a faster ME engine.

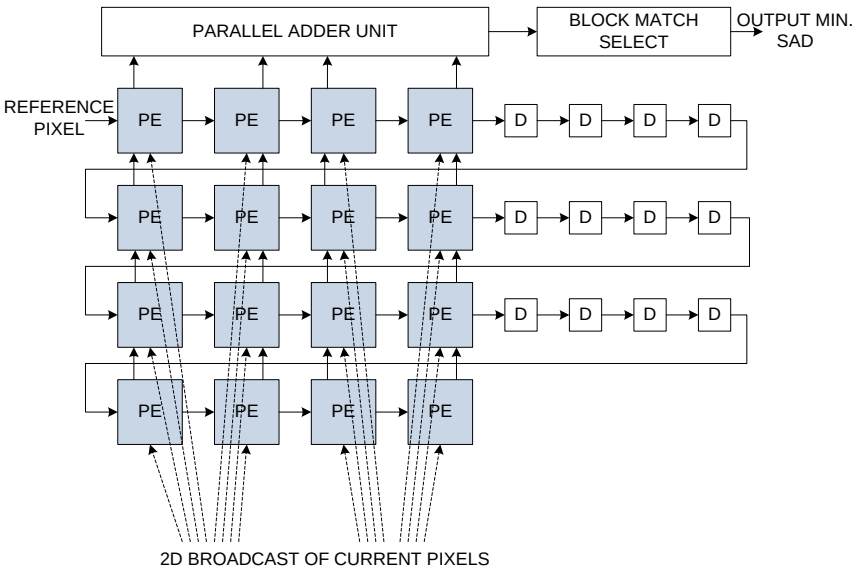

Figure 5 Generalized 2D intra-level SAD hardware architecture supporting block size of 4x4.

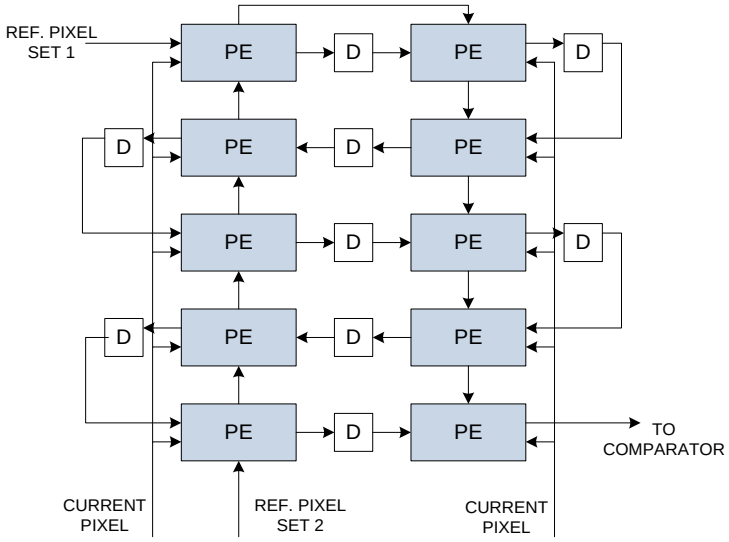

Figure 6 Generalized 1D inter-level SAD hardware architecture supporting SR of 8 horizontal direction.

Figure 5. shows generalized intra-level SAD architecture. In [35], [36], 2-D systolic PE array architecture was presented, where the number of PEs is equal to the MB size. The architecture has excess registers for the reference data to propagate and the search sequence is meander-like scanning.

Due to number of additional registers, this architecture has comparably low performance and latency is degraded. In [37], [38], intra level 2-D array architecture was proposed which satisfies H.264 requirements with high performance. However, their architecture is supported by a data flow where actually 16 different PEs are accessing 16 different reference pixel data simultaneously, which increases the memory access per clock cycle.

In inter-level SAD architecture, the PE matrix size is dependent on the search window (SW) size and with a regular data flow design, the architecture can be designed for a 1-D to a 2-D systolic structure. Figure 6. illustrates inter-level SAD architecture. In such architectures, reference frame data are broadcasted into corresponding PEs to compute SADs. Although it requires more hardware to have high parallelism, this type of approach effectively reduces the memory bit width. Hence, there is always a trade-off in ME engine performance, hardware area utilization, and most importantly memory bandwidth.

The work in [39] implements 1-D PE architecture which uses pipelining to improve the data reuse. However, it may not support the present ME which deals with various sub-blocks within each macroblock. The proposed works in [38], [40] have such 2-D array structures and can achieve high throughput. These architectures are scalable and have the scope for extending the level of parallelism. However, both of the works use ASIC design based on fixed hardware architecture.

2.2.2 Fast ME Architectures

Architectures for fast ME algorithms can reduce the hardware cost and decrease the computation time with acceptable video quality. The main challenges include design of unpredictable data flow, memory access and sorting out systolic mapping. For example, a simplified architecture of three step search is presented in [41]. A real chip for 3SS is realized in [42] with 0.8um technology. Several architectures for other fast ME algorithms are also discussed. But in general, only the

simplified fast ME algorithms are chosen to facilitate the hardware realization such as the designs for Hierarchical Search [19], [20], Three Step Search [13], and Diamond Search [43]. In the work of [44], a hybrid model to support various algorithms is presented. The average computational cost reduction in this work is about 24%.

In this thesis, first reconfigurable PE arrays were designed to improve performance and data reuse at 4x4 block level, but for fixed SR. This can be used to extend the architecture from 1-D to a faster 2-D SAD architecture. Further, we show how this SAD array architecture can be modified and reconfigured to support various SRs. If there is few motion activities in the sequence (for example, video captured by the surveillance camera in the hall way during night time), SR can be reduced further, resulting in reduction of power and hardware resources used.

2.3 Summary

Motion estimation engine is usually the most important module in the video encoding process. Significant amount of time is spent in software based module and demands good amount of hardware and memory bandwidth. In this Chapter, we made a detailed study of block matching algorithms and architectures during the past two decades. A lot of work has been done to reduce the computational load of ME through variety of software algorithms. However, hardware realization may not be feasible for each and every algorithm developed on software. The main challenge lies in developing hardware-oriented computational reduction algorithms, which can adapt different parameters and their corresponding adaptable hardware resources during run-time without turning-off the ME process for reconfiguration. The works discussed in this Chapter can be an inspiration for solving this challenge.

MODULAR H.264/AVC VBSME APPROACH

3.1 Top Level Architecture

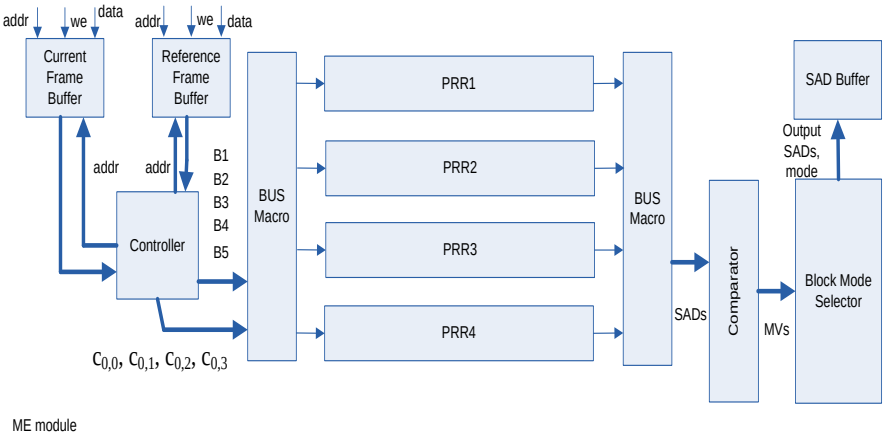

Figure 7 Top level architecture of proposed VBSME.

The top level architecture of proposed scalable ME design is shown in Figure 7. The ME module is divided into static region and reconfigurable region. The static region of ME module consists of current frame buffer, reference frame buffer, controller, comparator, block mode selector, and SAD buffer. These modules remain unchanged after initial configuration. Controller generates the address to fetch the data from current frame buffer and reference frame buffer. The results of 12 bit SAD value of 4x4 blocks from the reconfigurable region and 8 bit motion vector information from the comparator unit are propagated into the block mode selector unit where the best block size is determined. These final block modes, SADs, and corresponding MVs are stored in SAD buffer. In our implementation, 16 minimum SADs for each MB are computed and the best block mode and corresponding MV are stored into SAD buffer. The tracking range used for our implementation purpose is [-8, +7]. In this work, four modules, i.e., from PRR1 to PRR4, are included and tested to implement scalable ME computation. Each module is defined as a partial reconfigurable region

(PRR). Bus Macros (BM) are used to connect signals between static region and PRR. Each PRR can compute ME computation for 4×4 blocks within the search window.

3.2 Memory Management

The current frame buffer stores all the 256 pixels of current MB in internal BRAM memory. In order to increase throughput of memory access, it is divided into 4 dual port BRAMs with data width of 8 bits and depth of 64 pixels as shown in Figure 8(a). Each BRAM stores pixel information of four 4×4 blocks in a row at a given current MB. Depending on the number of active PRRs, the data flows into each PRR through the controller. For example, if all four PRRs are active, then PRR1-PRR4 receives current pixel information from C#1 - C#4 respectively.

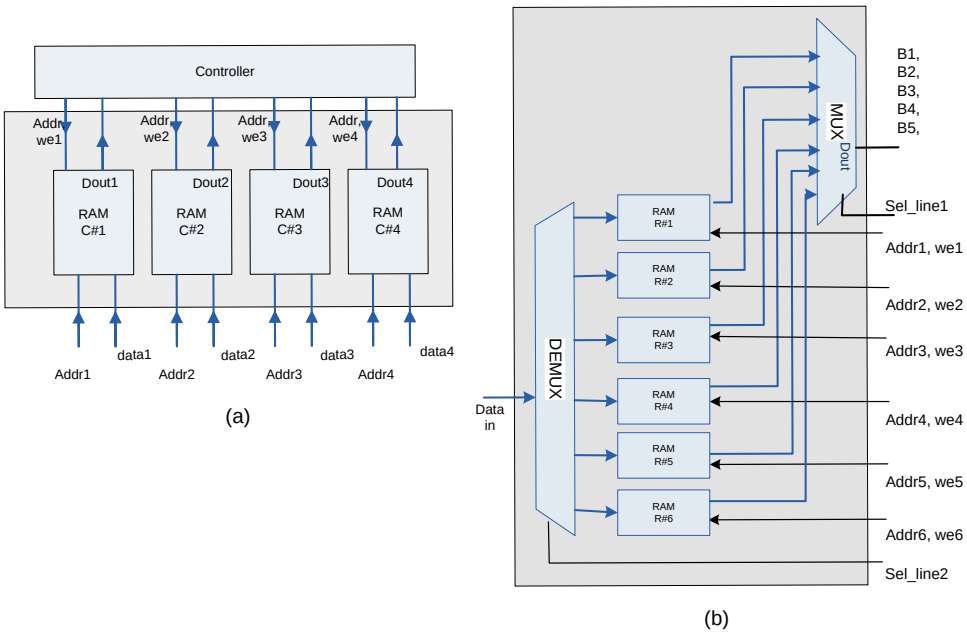

Figure 8 (a) Current frame buffer structure (b) Reference frame buffer

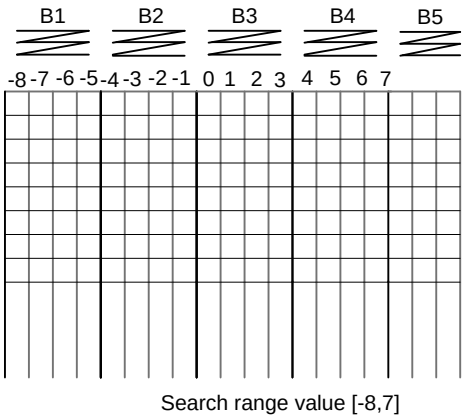

Figure 9 Data scanning in the search window

The design also includes six memories for reference frame buffer as shown in Figure 8(b). The search window is divided into five columns, having row size of 4 pixels, as shown in Figure 9. The column height is equal to the vertical size of search window. In our case, it is 31 pixel high. This information is stored in the BRAM with data width of 8 bits and depth of 31x4 pixels. The sixth BRAM is used as a temporary buffer for loading next search pixel values in parallel.

At the beginning of the motion estimation process, the first search window is stored in the reference buffer R#1 – R#5 as shown in Figure 8(b). While first round of SAD computations is taking place, the memory R#6 is loaded with next 31×4 pixels required for SAD computation of four 4×4 blocks of 2nd column of corresponding MB. Now, for this 2nd round of SADs, memories R#2 – R#6 are used, and simultaneously R#1 is updated with next pixel information. Therefore, bottle neck problem of memory accesses can be avoided in consecutive SAD iterations.

3.3 Partial Reconfigurable Module

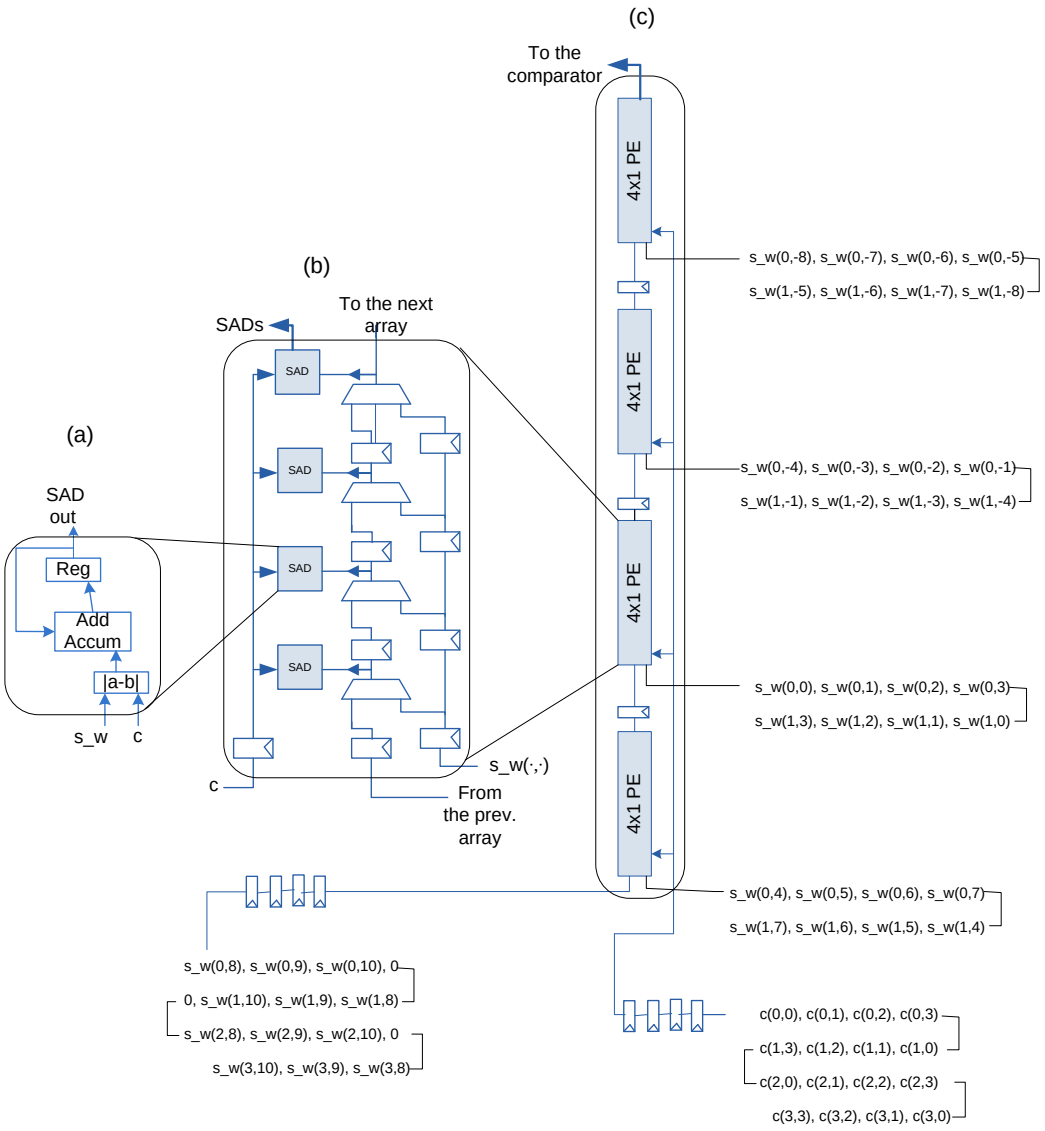

Figure 10 (a) Structure of SAD block (b) Building block of one 4×1 PE

(c) Architecture of one PRR with 16×1 PEs

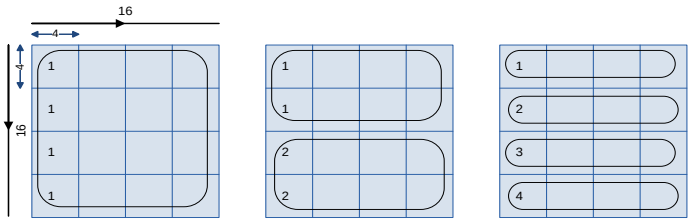

Figure 11 Number of blocks covered given the number of PRRs in active mode.

More the number of divisions imply more the degree of parallelism.

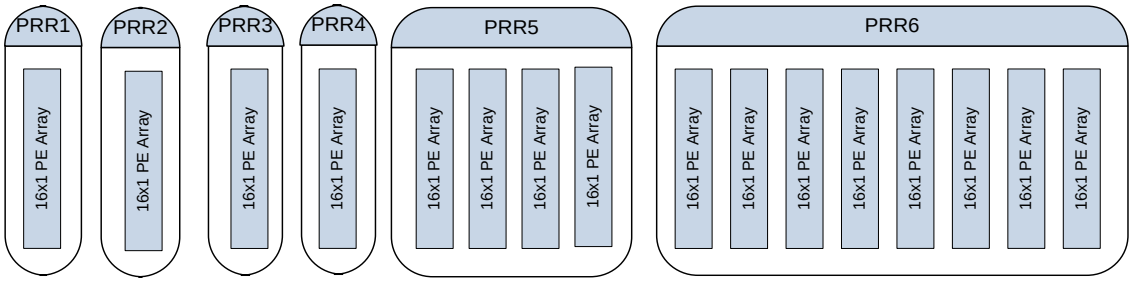

Figure 12 Different configurations of 16x1 PE Arrays

Figure 10. shows the internal structure of a 16×1 PE array. This unit consists of 16 SAD Units, responsible for simultaneously calculating SADs of all the search locations of one row [-8, +7] (i.e., 16 pixels) in the search window. The four PRRs are responsible for calculating SADs of four ‘different 4x4’ blocks simultaneously. For this, 1, 2 or 4 PRRs can be configured for computation.

Now, by increasing number of 16×1 PE Arrays, for calculating SAD for the ‘same 4x4’ current block, data reuse and performance can be improved. This can be done by introducing PRR5, which contains additional 4 16x1 PE arrays, the data reuse for each block is improved and so is the SAD computation speed. Each 16x1 PE array here will compute SADs for the first column of four 4x4 blocks in Figure 11. While the PRR1-4 compute SADs of search candidates of a particular row, PRR5 computes SADs for search candidates of next row simultaneously but with initial latency of 4

clock cycles. The various types of modules that can be configured are shown in Figure 12. For implementation purpose, we use reconfigurable PRR1 – PRR4.

3.4 Data Flow

The absolute difference (AD) of specific pixel in current block and reference block can be computed as shown below

$$AD_{(i,j)}^{(x,y)}(h,v) = |C_{(i,j)}(x,y) - s\_w_{(i,j)}(x+h, y+v)| \quad (3)$$

Here, (x, y) is pixel location within block, i.e., 0≤(x, y)≤n-1. Part of search window (s_w) is observed to be common with neighboring blocks. So, this search window data can be exploited by using 4 PRRs where each PRR computes SAD of neighboring block. Hence, data reuse is improved by increasing number of PRRs from 1 to 2 and 2 to 4. From equation (2), it is clear that current pixel information c(x, y) is common for evaluating every SAD (h, v) where -8≤(h,v)≤+7. So, this data can be shared with all search candidates row-wise (h) as well as column-wise (v). If we implement one 16×1 PE Array, all search locations with common 'h' are computed in parallel. This parallelism can be extended to 2-D, i.e., for all 'h' as well as 'v' by implementing two 16×1 (partial data reuse improvement) or four 16×1 (full data reuse improvement) PE Arrays. These 16x1 PE arrays share the current block data and search window data. Now each of PRR1 – PRR4 uses one PE array. If we introduce four more arrays (PRR5), two rows of pixel values in the search window can be processed simultaneously. Similarly, if we introduce PRR6, then four rows of pixel values in the search window can be processed simultaneously.

Table 1 Dataflow for Proposed Reconfigurable Motion Estimation Algorithm, for Search Window [-8, +7]

B5, B4, B3, B1, B2	c	SAD locations (h, v)	SADs of corresponding 4x4 Blocks, i.e., $SAD_{(i, j)}$			Showing SAD flow of first current block	Showing SAD flow of first current block
			(1PRR)	(2PRRs)	(4PRRs)	PRR(1-5)	PRR(1-6)
s_w(-8,-8)-(-8,10)							
s_w(-7,-7)-(-7,10)	c(0,0)-c(0,3)						
s_w (-6,-6)-(-6,10)	c(1,0)-c(3,3)						
s_w (-5,-5)-(-5,10)	c(2,0)-c(3,3)						
s_w (-4,-4)-(-4,10)	c(3,0)-c(3,3)						
s_w (-3,-3)-(-3,10)	c(0,0)-c(3,3)	(-8,-8) - (-8,7)	$SAD_{(0,0)}$	$SAD_{(0,0)}$	$SAD_{(0,0)}$	SAD(-8,-8) - (-8,7)	SAD(-8,-8) - (-8,7)
s_w (-2,-2)-(-2,10)	c(1,0)-c(3,3)					SAD(-7,-8) - (-7,7)	SAD(-7,-8) - (-7,7)
s_w (-1,-1)-(-1,10)	c(2,0)-c(3,3)						SAD(-6,-8) - (-6,7)
s_w (0,0)-(0,10)	c(3,0)-c(3,3)						SAD(-5,-8) - (-5,7)
s_w (1,1)-(1,10)		(-4,-8) - (-4,7)	$SAD_{(0,0)}$	$SAD_{(0,0)}$	$SAD_{(0,0)}$	SAD(-4,-8) - (-4,7)	SAD(-4,-8) - (-4,7)
s_w (2,2)-(2,10)	…	(-8,-8) - (-8,7)		$SAD_{(1,0)}$	$SAD_{(1,0)}$	SAD(-3,-8) - (-3,7)	SAD(-3,-8) - (-3,7)
s_w (3,-8)-(3,10)	…						SAD(-2,-8) - (-2,7)
..	…						SAD(-1,-8) - (-1,7)
..	…	(0,-8) - (0,7)	$SAD_{(0,0)}$	$SAD_{(0,0)}$	$SAD_{(0,0)}$	SAD(0,-8) - (0,7)	SAD(0,-8) - (0,7)
..		(-4,-8) - (-4,7)		$SAD_{(1,0)}$	$SAD_{(1,0)}$	SAD(1,-8) - (1,7)	SAD(1,-8) - (1,7)
s_w (6,-8)(6,10)		(-8,-8) - (-8,7)			$SAD_{(2,0)}$		SAD(2,-8) - (2,7)
s_w (7,-8)-(7,10)	c(3,0)-c(3,3)						SAD(3,-8) - (3,7)
	c(0,0)-c(0,3)	(4,-8) - (4,7)	$SAD_{(0,0)}$	$SAD_{(0,0)}$	$SAD_{(0,0)}$	SAD(4,-8) - (4,7)	SAD(4,-8) - (4,7)
		(0,-8) - (0,7)		$SAD_{(1,0)}$	$SAD_{(1,0)}$	SAD(5,-8) - (5,7)	SAD(5,-8) - (5,7)
		(-4,-8) - (-4,7)			$SAD_{(2,0)}$		SAD(6,-8) - (6,7)
		(-8,-8) - (-8,7)			$SAD_{(3,0)}$		SAD(7,-8) - (7,7)
		(-7,-8) - (-7,7)	$SAD_{(0,0)}$	$SAD_{(0,0)}$	$SAD_{(0,0)}$	SAD(-6,-8) - (-6,7)	
		(4,-8) - (4,7)		$SAD_{(1,0)}$	$SAD_{(1,0)}$	SAD(-5,-8) - (-5,7)	
		…			…	…	…

In the proposed architecture, the search window is divided into 5 columns and the pixel data are scanned simultaneously from these columns as shown in Figure 9. During initial 4 clock cycles, the pixel information of first row in the search window (s_w(0,0)(-8,v) - s_w(0,0)(7,v)) is stored in the latches. In the next clock cycle, the current pixel information (c(0,0)) becomes available to each 16x1 PE array when it starts SAD computations. During this cycle, the next row pixel information in the search window is continuously read and pipelined into the latches. More precise details of data

flow into the latches are shown in Figure 10(c). These partial SADs are stored in each PE array After 16 clock cycles SADs are propagated into the comparator unit for MV generation. The detailed data flow for different PRRs architecture is shown in Table 1. According to the number of PRRs operating, the controller unit can efficiently have the data flow pipelined among the PRRs to increase data reuse.

3.6 Block Mode Selector

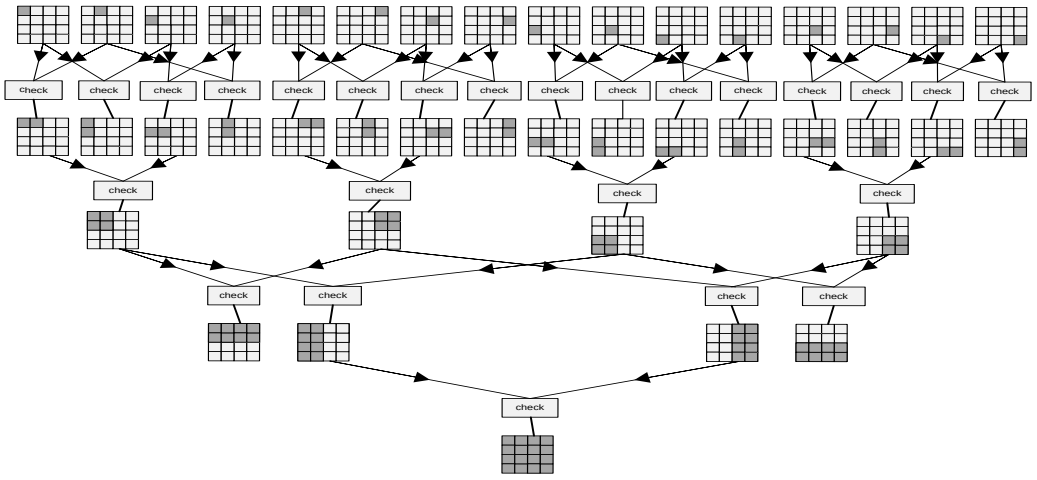

Figure 13 Variable Block Mode Selector for H.264/AVC Standard

In the H.264/AVC standard, mode decision algorithm is left open to the developer. Given the quantization parameter (QP) and the Lagrange parameter, λ_{mode}, the mode decision can be performed by minimizing the following equation,

$$J_{mode}(MB_k, I_k|QP, \lambda_{mode}) = Distortion(MB_k, I_k|QP, \lambda_{mode}) + \lambda_{mode} \cdot Rate(MB_k, I_k|QP, \lambda_{mode}) \quad (4)$$

Where I_k denotes all possible coding modes and their corresponding MVs. Due to the huge computation complexity and sequential issues in the high complexity mode of H.264/AVC, it is less suitable for real-time applications. In this Chapter, we discuss a low complexity mode decision [31].

The function of this unit is to merge SADs of 4x4 blocks that are exact matches. The resulting merge follows the rules of allowable block patterns for the H.264/AVC ME. For our implementation, merging decision is based on motion vectors of neighboring 4×4 blocks. If the MV displacements are exact match, then we merge the neighboring blocks [31]. The Variable Block Size Decision (VBSD) approach reduces the number of optimal sub-macroblocks for motion compensation operations and uses less hardware resources when compared to the technique used in JM software. Table 2. gives the comparison results of PSNR and bitrate with JM software. On an average, it reduces 5% of bitrate with only 0.04dB of PSNR degradation. In other words, data computational load is reduced and so is the ME computation time.

Figure 13. shows the block mode selector tree. The values of SADs and MVs are read from output of comparator unit. The comparator unit finds the best SAD for given block, by reading the output SADs from the PRR(s). The final resultant block mode and corresponding SADs are stored in the output SAD buffer unit.

Table 2 Bitrate and PSNR comparison of VBSD scheme

		Container	Carphone	Foreman	Football	Hall	Suzie
JM	PSNR(dB)	36.348	37.398	36.711	36.560	37.711	37.568
	Bitrate(Kpbs)	45.382	119.000	139.300	745.820	60.890	91.820
VBSD Scheme [31]	PSNR(dB)	36.319	37.345	36.655	36.472	37.676	37.555
	Bitrate(Kpbs)	44.007	117.020	124.350	696.330	58.059	90.440
	ΔPSNR	-0.029	-0.053	-0.056	-0.088	-0.035	-0.013
	Δbitrate	-1.375	-1.980	-14.950	-49.490	-2.831	-1.380

3.6 Experimental Results

Table 5 gives the performance characteristics of our design when various reconfigurable structures are used. From this Table, it is clear that as the number of PRRs increases, the number of SADs that are computed in parallel also increases. This actually depicts the degree of parallelism and data reuse.

Table 6 shows the time required for SAD computations for each frame at a maximum clock frequency of 90MHz. Suppose the frame rate of a video sequence be 30fps, then the maximum allowable time that ME engine can take is 33ms for each frame. If the image resolution is either QCIF or CIF, we can reduce the hardware by using only 1 or 2 PRRs and still maintaining required ME time. However, if the image is of higher resolution, then a faster ME engine is needed. In that case, number of PRRs configured can be increased. Partial reconfiguration of these modules can allow best utilization of FPGA hardware resources.

At architecture level, the performance of our proposed architecture for different operating PRRs is compared with previous works of [45], [37], [46], [47]. Table 3 shows general characteristics of number of PEs, latency, operating cycles, bit width information summarizing the level of parallelism and data reuse. It also includes the scalability factor which plays important role in reconfigurable design methodology. The macro block size is taken as N×N and the smaller block as n×n. For block in the current frame, one reference block that is best match to it, in the search range of H×V, is selected. ‘P’ represents the number of 16x1 PE arrays present, i.e., P=1,2,4,8,16 for combination of PRRs(1), (1&2), (1-4), (1-5) and (1-6) respectively. ‘q’ represents number of current blocks accessed in parallel, i.e., q=1,2 or 4 for combination of PRRs(1), (1&2), [(1-4) or (1-5) or (1-6)] respectively. From the Table 5, it is clear that the ME engine speeds up with increase in PRRs and

at the same time only 1 PRR can be configured in applications where hardware resource and power consumption are critical criteria.

The memory bit width is defined as the number of bits the algorithm accesses from the memory in each cycle. It is one of the factors to determine efficient memory management and degree of parallelism. The works of [45], [37], and [47] compute 16 SADs in parallel hence have large bit width, but their data flow is not regular. Memory bandwidth is defined as the number of memory access to complete ME for one MB. It is the one to determine the efficient data reuse capability. From Table 4, it is shown that by increasing the number of PRRs, data reuse is significantly increased when compared to previous works.

Table 7 gives the synthesis result and Table 8 shows the partial bit stream size of each PRR module. The design is implemented on Xilinx Virtex2Pro (XC2VP30) FPGA development board. Synthesis is done using ISE Foundation Tool. The full and partial bit streams are generated using PlanAhead 10.1 Tool. Bitstreams obtained from the synthesized design are used to reconfigure the FPGA through JTAG.

Table 3 Various Criteria of Different Architectures

Architecture	Latency	Bit Width for ref. frame	Operating Cycles	Scalability/ Modularity
[45]	N	$2 \cdot N \cdot 8$	$[H \cdot V] + N - 1$	Memory bitwidth constraints
[37]	N	$(N + 1) \cdot 8$	$[H \cdot V] + N - 1$	Memory bitwidth constraints
[46]	$N \cdot (V+1)$	$2 \cdot 8$	$[H \cdot V] + N \cdot (V+1)$	-
[47]	$H + N$	$N \cdot 8$	$H \cdot (V + N)$	-
Proposed work	$n \cdot (n + 1)$	$[(H/n) + 1] \cdot 8$	$[H \cdot V] \cdot (n^3/P \cdot R)$	Yes

Table 4 Comparison of Different Hardware Architectures for VBSME Algorithm for Search Range [-8, 7], N=16, N=4

Architecture (with r = 16x1)	Bit width for ref. frame (bits/cycle)	Bit width for current frame (bits/cycle)	Bandwidth for ref. Frame (Kbits/MB)
1 PRR	40	8	184
2PRRs	40	16	92
4PRRs	40	32	46
8PRRs	40	64	23
16PRRs	40	128	11.5
[45]	256	-	76
[37]	136	-	48
[46]	16	-	10.5
[47]	128	-	77.6

Table 5 Various Criteria of Different Architectures

No. of PRRs	No. of SADs	Buffer cycles for each MB
1	16/16 cycles	16
2	32/16 cycles	32
4	64/16 cycles	64
5	128/16 cycles	72
6	256/16 cycles	88

Table 6 Various Reconfigurable Architectures to Support Different Image Resolutions

Display Type	Type of Reconfigurable architecture	Estimated time for SAD computation for each frame @ 90MHz
QCIF (174×144)	1PRR	5.5ms
CIF (352×288)	1PRR	17.7ms
	2PRR	9ms
720×480	1PRR	62ms
	2PRR	31.7ms
	4PRR	15.9ms
1920×1080	1PRR	366ms
	2PRR	183ms
	4PRR	92ms
	5PRR	47.8ms
	6PRR	24.6ms

Table 7 Hardware Resources

Module	LUTs	Slice Flip Flops	MUX 18x18	BRAMs
16×1 PE Array	1109	871	-	-
Static & Memory Buffers	874	327	7	15

Table 8 Bitstream Information

Bitstreams	Size (Bytes)
PRR1	90640
PRR2	105472
PRR3	90640
PRR4	105472

3.7 Summary

In this Chapter, we presented an approach to design integer pixel H.264/AVC variable block size scalable ME engine. Its regular structure, regular data flow, and memory management make it possible to implement IME-FSBMA. The Chapter includes all the modules like PE Array, mode selector, and memory unit. The proposed architecture has lower memory bandwidth and latency. Simulation results show that our design can increase data reuse significantly and thereby reduce the memory bandwidth overhead. Further through partial reconfiguration approach, depending on input image resolution, it can adopt suitable number of PRRs during runtime. In summary, our architecture presents an inherent capacity to adjust the hardware resources, which forms basis for a novel self adaptive full-search algorithm.

ADAPTIVE SEARCH RANGE ALGORITHM AND IMPLEMENTATION

4.1 Video Coding Efficiency with Variable Search Range

In video coding applications, the search range for motion estimation should be large enough to cover the motion displacements, but at the same time, increase in the SR also causes a problem of increasing computational overhead. Figure 14. shows the Peak Signal to Noise Ratio (PSNR) performance of H.264/AVC using motion estimation with different Search Ranges (SR), and 40 frames from different video sequences with CIF (352×288 pixels) resolution are encoded at 30fps.

The results show that after SR (± 32), the PSNR is saturated indicating significant increase in computational overhead. This is illustrated in Figure 15, where computational cost is calculated as the number of SAD evaluation for full search ME algorithm.

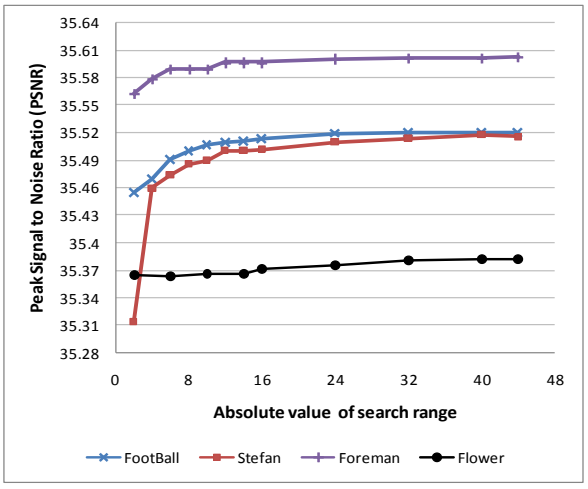

Figure 14 PSNR performance of H.264/AVC motion estimation for various video sequences

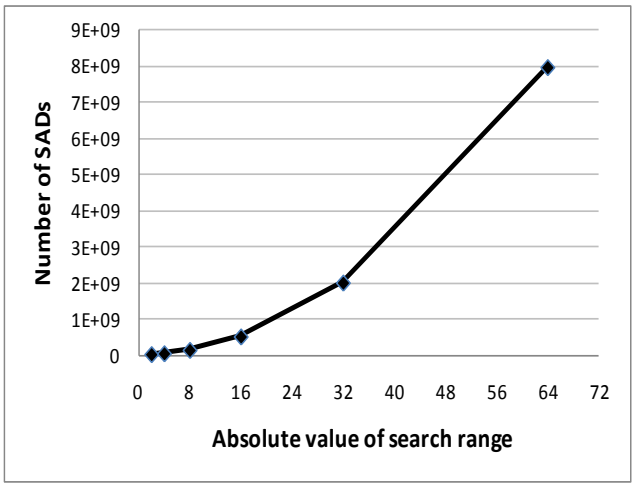

Figure 15 SAD computations of H.264/AVC ME for various search ranges

H.264/AVC standard introduces other new features, such as multiple reference frames and up to quarter pixel resolution ME. Applying these two methods for all the search points, further increases the computational load, making hardware implementation essential. In order to accelerate these computations, quarter pixel ME can be performed by using certain integer pixel points which have lowest coding cost. Therefore, the performance of integer pixel ME at the first round needs to be more accurate so that the following sub-pixel ME can produce the best results. Therefore, after an efficient motion vector prediction, the search range center can be set depending on surrounding motion vectors. This can be useful to reduce the SR size [JVT-P026]. There is a good chance that the search range can be reduced for the same video sequence depending on the behavior of motion vectors of neighboring blocks as well as previous frames without much degradation of PSNR and compression ratio.

Table 9 Percentage of the best matching MVs covered with different search ranges for football video sequence (QCIF @30fps)

SR	PSNR	Bitrate	% of vectors covered
±8	36.545	738.75	87.61
±16	36.553	713.42	91.66
±24	36.596	705.67	98.49
±32	36.617	692.93	100.00

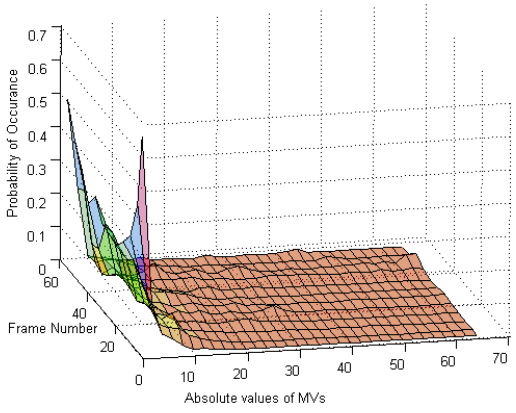

Figure 16 Distribution of MVs for football video sequence (QCIF @30fps)

Figure 16. shows the example of absolute motion vector displacements after applying prediction-based MV and re-evaluating center of search range. The x-axis gives the absolute value of motion displacement vector and the y-axis gives the probability of occurrence for different values of MVs. Table 9. gives the performance variation for different values of SRs. Analysis has been performed for various sequences and is derived that most of the best motion vectors are very near the predicted center of SR and hence by adaptively reducing the SR, we can achieve great reduction in computational complexity as well as hardware resources used.

From careful analysis of these plots for various video sequences, we conclude that more than 90% of the MVs fall under the search range of ±6 to ±12 (for QCIF resolution @30fps) with very small degradation in video quality. From the above analysis, it is derived that by reducing the SR further, we can reduce the computational complexity of ME.

4.2 Search Range Reduction Algorithm

Performing exhaustive search is computationally very expensive and adds great complexity to ME. A full search algorithm for each block candidate requires [(2R+1)·2] SAD computations where [+R,-R] is the search window size. We present an approach to reduce this search range (R) adaptively based on the correlation of the motion vectors as described in Section 4.1. This approach is based on the assumptions that motion field is smooth and changes slowly frame by frame. The correlation among MVs for the neighboring blocks is very high and hence current MV is likely to be near the search center which is predicted by using the previously encoded MVs of the neighboring blocks.

The adaptive search range algorithm can be described in the following steps:

a) Initially set the SR for all the reference frames (for the case of multiple or single referenced frame algorithm) to maximum SR (i.e., [-32, +32]).

b) Perform prediction algorithm as used in the JM software and obtain Motion Vector Prediction (MVP). Then, set the SR center in the reference frame accordingly. For simplicity in hardware implementation, all the block types for current macroblock share the same MVP (Fast Full Search in JM).

c) Perform Fast Full Search ME and simultaneously store the number of MVs falling into different category of absolute displacement. (i.e., number of MVs with displacement 0,1,2, and so on). The averaged MV, MV_{avg}, for a given macroblock in previous R reference frames is estimated by the equation below:

$$MV_{avg} = \frac{1}{R} \sum_{r=1}^{R} \frac{1}{M} \sum_{i=1}^{M} mv_{ri} \quad (5)$$

Where M is number of sub-blocks in each macroblock. The optimal SR can be estimated by using this MV_{avg}.

d) Set the search range for the next frame such that the SR covers at least Th (Threshold) % of total MVs. 'Th' can vary from 90-100% depending on the user requirement: higher performance/PSNR, less computational complexity or hardware resources utilized, and so on. There are some algorithms which update SR at macroblock level for software oriented implementations. In our approach, SR is updated at frame level. Hence, it is assumed that SR is fixed for all macroblocks in particular frame.

e) Given minimum SAD threshold increase or maintain the new SR. The SAD threshold for given average MV of a frame is listed in [51].

f) Continue from step (b) onwards.

4.3 Proposed ME Architecture

As discussed in Chapter 1, the basic functions of a block-based ME are to calculate and compare the matching criterion between the current image block and all candidate blocks in the search range of reference frame. In H.264/AVC, seven different block sizes are specified for ME process as shown in Figure 2. Therefore, an ideal encoder has to examine all possible combinations of MB division modes to select the best mode among them. Hence, VBSME improves motion tracking over fixed block size algorithm especially by giving attention to highly active sub-block. A 4x4 block can be considered as a preliminary block. Using SAD of this sub-block, SADs of larger blocks can be computed by simply adding the corresponding sub-block SADs. A total of 41 sub-blocks including macroblock have to be evaluated for the motion vector selection. In SR reduction approach discussed earlier (Sections 4.1, 4.2), we consider MVs corresponding to all the 41 sub-blocks for setting suitable SR for the following frame.

The ME architecture is separated into different Partial Reconfigurable Regions (PRRs). Each PRR consists of PE arrays, where each PE is responsible for computing SAD of a specific searching candidate. In the later Sections, we show how each PRR can be configured to support increasing or decreasing SR in steps of 2, 4 and 8 (in both X and Y directions). Increasing the number of PRRs: 1,2,3,4 can support SRs with ±2, ±4, ±6, ±8 respectively if the PRR is configured for 2-step SR. Similarly, to support set of SRs [±4, ±8, ±12, and ±16], the PRRs can be configured for 4-step SR.

4.3.1 Top Level Architecture

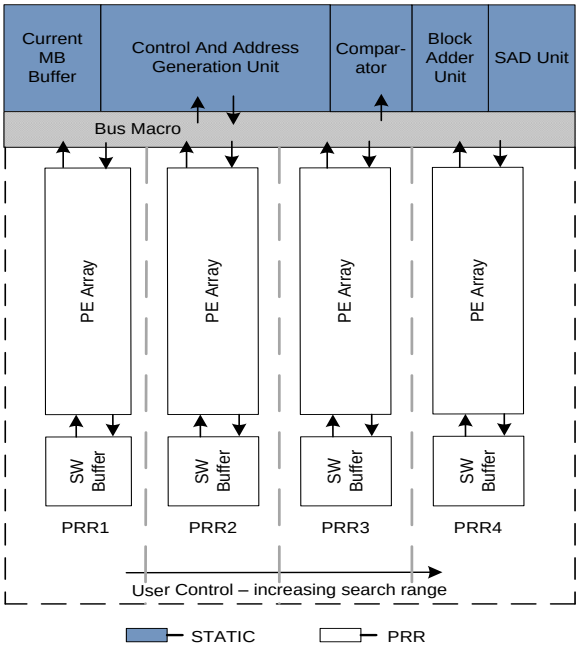

Figure 17 Top level architecture for partially reconfigurable ME supporting different search ranges

The modified top level architecture to support different SRs is shown in Figure 17. The architecture discussed in Chapter 3, forms the basis of this new architecture. In Section 3.2, it is discussed that the reference frame buffer size depends on the number of horizontal search range pixels. For example, if SR is ±8, we need five buffers; each one storing four column lengths of reference pixels. But now as SR is not fixed, the reference frame buffer also changes. Hence, it is desirable to move this buffer into reconfigurable area along with PE array. In our implementation, SADs for each 4x4 block are computed, sent to block mode selector unit, and the resultant MVs and corresponding SADs are stored in SAD buffer. For implementation purpose, we configured the PRRs for 2-step SR. In this work, four modules, i.e., from PRR1 to PRR4, are included and tested to implement scalable ME computation.

4.3.2 Memory Management

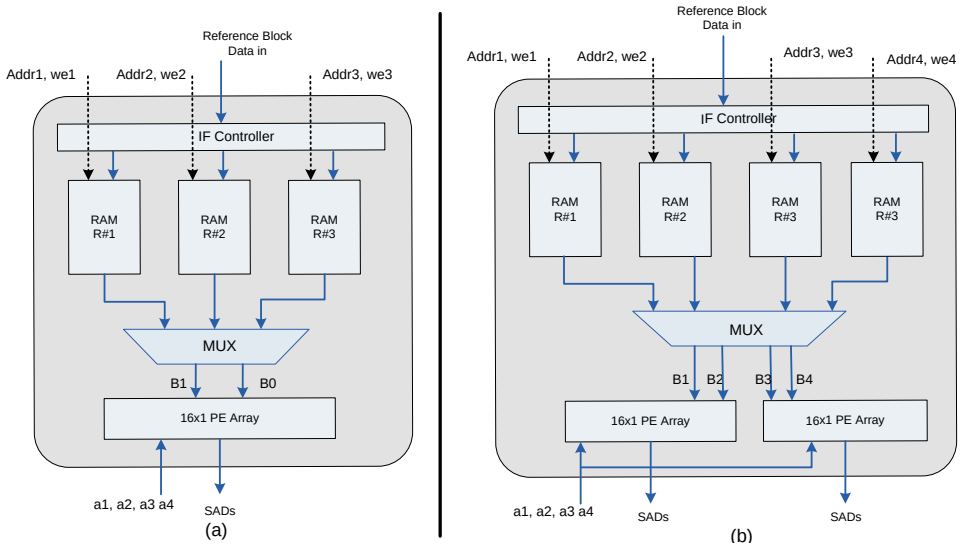

Figure 18 (a) General architecture of partial reconfiguration module for 2-step SR **(b)** General architecture of partial reconfiguration module for 4-step SR

Memory management becomes an essential aspect for modular designs as the data flow into each module should be made regular and at the same time memory bandwidth should be at the minimum level. The current frame buffer architecture is discussed in Section 3.2. Its size is equal to MB size, i.e. 16x16 pixels. As all the sub-blocks in a macroblock are assumed to share same MVP, SR is overlapped between different 4×4 sub-blocks. In our PE design, we pipeline four 4×4 blocks to effectively re-use the SR data.

To support 2-step SR, the reference frame buffer includes 3 BRAMs as shown in Figure 18(a). Each BRAM stores 4×V pixels from the reference frame, where 'V' is the vertical height of SR covering one macroblock. R#1 stores these pixel data and R#2 stores the additional 3 pixel data needed for border 4×4 block positions in SR. Third BRAM is used as a temporary buffer for loading next search pixel values simultaneously. This can be one approach where the data flow is not interrupted and SAD computations can be performed continuously for different blocks.

4.3.3 Partial Reconfigurable Module

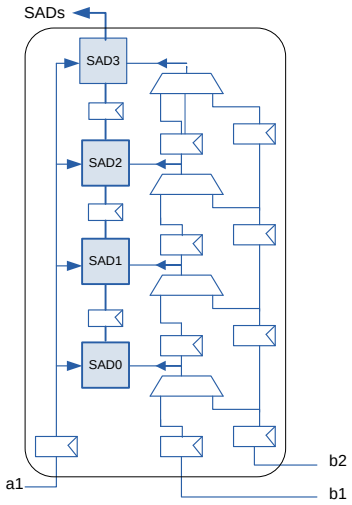

Figure 19 Internal structure of 4x1 PE array

Figure 19. shows the internal structure of 4×1 PE array, having 4 SAD units. Each PRR consists of four such PE arrays including total of 16 SAD units (SAD0-SAD15). The four sets of SAD array (SAD0-SAD3), (SAD4-SAD7), (SAD8-SAD11), and (SAD12-SAD15) are responsible for simultaneously calculating SADs of four 4×4 blocks corresponding to a column of particular macroblock.

Each PRR is responsible for SADs of ±2 SR for four blocks and by increasing the PRRs from 1, 2, 3, and 4, the ME design can support ±2, ±4, ±6, and ±8 SR respectively. Similarly, the PRR structure can be designed to support increasing/decreasing SR in steps of 4 (Figure 18(b)) and 8. This can be useful for video with very high motion activities and high resolution which require optimum SR to be greater than ±8. For implementation purpose, we use 4 PRRs which can support increasing/decreasing SR in steps of ±2.

Here from this work and also from work from Chapter 3, it can be shown that by rearranging the SAD units and with proper memory management unit, we can design a modular ME engine to support various requirements from the user, such as frame rate, video resolution, and search range,

and to maximize the benefits by fully utilizing the trade-offs among video signal characteristics, computing power, and reconfigurable hardware resources.

4.3.4 Data Flow

Modified data flow is shown in Table 10. We have discussed some of the previous ME architectures in Chapter 2 which perform FS with highly parallel computations. All PEs are connected through fixed wires and the level of parallelism is fixed. In the work of [39], computations of horizontal pixels in parallel can be performed, but if the SR is changed, it should be fully reconfigured to support variations in SR. The architecture proposed in [37], can perform parallel computation of fixed 16x16 pixels. However, such kinds of fixed architectures may not be suitable if SR is varying during run time.

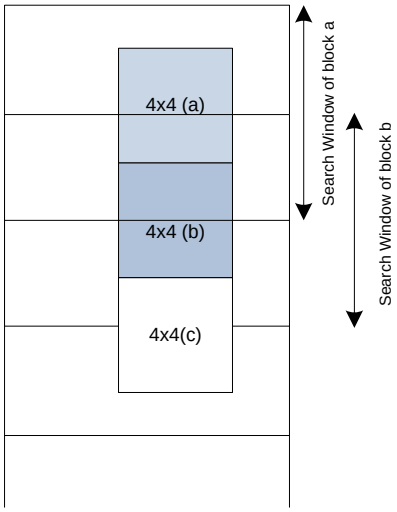

Figure 20 Data sharing among neighboring 4x4 blocks

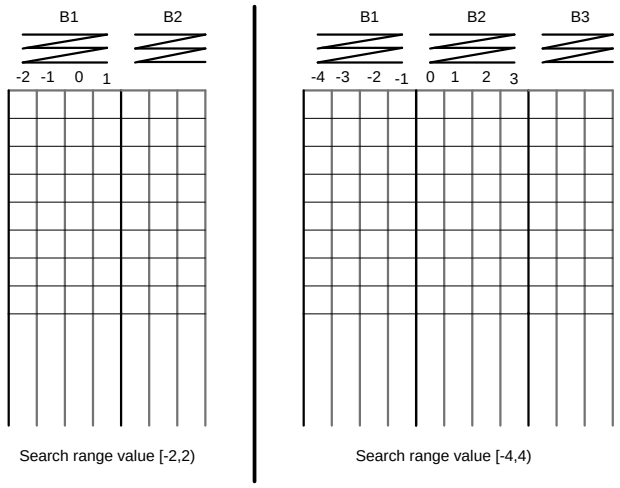

Figure 21 Partitioning of SW into different 4 pixel width columns

The search window data can be shared among different blocks (Figure 20) as well as among SADs of different locations of the same block. In this work, we divide the SW into different columns in order to pipeline the data to increase the performance of ME. However, the number of parts divided depends on the SR value. For example, the SW is divided into 2, 3, 4, 8 parts for SR values of ±4, ±6, ±8, ±16 respectively. Figure 21 shows the division of SW for different configurations of SR including extra 3-pixel width column for border search locations.

Table 10 Data Flow for different PRRs

B5, B4, B3, B2, B1	c	SADs of corresponding 4x4 Blocks, i.e., $SAD_{(i, j)}$	SAD locations (h, v)			
			(1PRR)	(2PRRs)	(3PRRs)	(4PRRs)
s_w(-8,-8)-(-8,10)			Initial Latency: 4 clock cycles	Initial Latency: 8 clock cycles	Initial Latency: 12 clock cycles	Initial Latency: 16 clock cycles
s_w(-7,-7)-(-7,10)	c(0,0)-c(0,3)					
s_w (-6,-6)-(-6,10)	c(1,0)-c(3,3)					
s_w (-5,-5)-(-5,10)	c(2,0)-c(3,3)					
s_w (-4,-4)-(-4,10)	c(3,0)-c(3,3)					
s_w (-3,-3)-(-3,10)	c(0,0)-c(3,3)	$SAD_{(0,0)}$	(-2,-2)-(-2,1)	(-4,-4)-(-4,3)	(-6,-6)-(-6,5)	(-8,-8) - (-8,7)
s_w (-2,-2)-(-2,10)	c(1,0)-c(3,3)					
s_w (-1,-1)-(-1,10)	c(2,0)-c(3,3)					
s_w (0,0)-(0,10)	c(3,0)-c(3,3)					
s_w (1,1)-(1,10)		$SAD_{(0,0)}$	(-2,-2)-(-2,1)	(0,-4)-(0,3)	(-2,-6)-(-2,5)	(-4,-8) - (-4,7)
s_w (2,2)-(2,10)	…	$SAD_{(1,0)}$		(-4,-4)-(-4,3)	(-6,-6)-(-6,5)	(-8,-8) - (-8,7)
s_w (3,-8)-(3,10)	…					
..	…					
..	…	$SAD_{(0,0)}$	(-2,-2)-(-2,1)		(2,-6)-(2,5)	(0,-8) - (0,7)
..		$SAD_{(1,0)}$		(0,-4)-(0,3)	(-6,-6)-(-6,5)	(-4,-8) - (-4,7)
s_w (6,-8)(6,10)		$SAD_{(2,0)}$		(-4,-4)-(-4,3)	(-6,-6)-(-6,5)	(-8,-8) - (-8,7)
s_w (7,-8)-(7,10)	c(3,0)-c(3,3)					
	c(0,0)-c(0,3)	$SAD_{(0,0)}$	(-2,-2)-(-2,1)			(4,-8) - (4,7)
		$SAD_{(1,0)}$			(2,-6)-(2,5)	(0,-8) - (0,7)
		$SAD_{(2,0)}$		(0,-4)-(0,3)	(-6,-6)-(-6,5)	(-4,-8) - (-4,7)
		$SAD_{(3,0)}$		(-4,-4)-(-4,3)	(-6,-6)-(-6,5)	(-8,-8) - (-8,7)
		…	…	…	…	…

Table 12 shows various levels of parallelism adapted by our proposed ME architecture, when SR is varied dynamically. The number of SADs computed in parallel for each block is equal to the horizontal width of the SW. This 1-D parallel architecture can be extended to 2-D by simply adding additional PE arrays to perform SADs of different rows for the same 4x4 block. The detailed description of this is given in Chapter 3, where the PRR number 5 and 6 account for pipelining the data in vertical direction and PRRs 1,2 and 4 account for pipelining the data in horizontal direction of SW.

The architecture implemented supports parallelism in horizontal direction. For example, if the SR value is ±8, the set of row pixels; [-8,-5], [-4,-1], [0, 3], [4, 7] flow into the PRRs 1, 2, 3, and 4 respectively, and these PRRs perform corresponding SADs in parallel. In addition to this, each PRR is further divided into four 4x1 PE Arrays to compute SADs for four different 4x4 blocks. These four blocks correspond to a particular column in the given macroblock.

4.4 Experimental Results

4.4.1 Simulation Results for Search Range Reduction Algorithm

The performance comparison of Search Range Reduction Algorithm with the JM 12.4 Reference Software is given in Table 11. Simulations are done for different threshold values. The algorithm is compared in terms of performance; PSNR, bitrate, computational complexity, and hardware utilization.

All the simulations are performed on various video sequences (in QCIF Format, 30 fps) to verify the proposed algorithm. These are compared with the Fast Full Search algorithm in JM software with parameters set to baseline profile (no B Slices), one reference frame, and fixed SR for

60 encoded frames. Fast Full Search is chosen because it uses same SR for all blocks of particular macroblock.

The computational complexity is determined by the following parameter:

$$\text{Reduction in Computational Cost}=\left(1-\frac{P\times P}{Original\_SR}\right)\times 100\% \quad (6)$$

where 'P×P' corresponds to the average SR calculated for particular video sequence corresponding to SR Reduction Algorithm, and 'Original_SR' is the reference SR (i.e., ±32). From Table 11, it is clear that the computation cost is greatly reduced by the adaptive SR reduction algorithm at the cost of little degradation in video quality (small decrease in PSNR and increase in bitrate). Thus, such algorithms will be very useful in reducing hardware costs significantly.

The hardware utilization is determined by the following:

$$\text{Hardware Utilization }=\left(\frac{Used\_PE\_Aray}{Total\_PEs\_Static}\right)\times 100 \quad (7)$$

where 'Used_PE_Array' is given by average number of PE Arrays which are configured to process the video sequence, and 'Total_PE_Static' is total number of PE Arrays that support reference SR (i.e., ±32). For videos with very little or still motion, such as container sequence, very less hardware is required and for videos with irregular and high motion like football sequence, more hardware resources are required.

4.4.2 Proposed Partially-Reconfigurable ME Architecture Evaluation and Comparisons

Table 12 gives the performance characteristics of our design when various reconfigurable structures are used. As the SR increases, the ME Engine should adaptively set the hardware architecture so that it can cope with the timing requirements. From the table, it is clear that as SR increases, the number of SADs computed also increases accordingly, depending on type of SR

configuration; 2,4, or 8-step SR, so that the total SAD computation time for each MB remains nearly the same for a given SR. For example, to configure for SR value ±4, it takes 2PRRs when 2-step SR approach is used, and 1PRR when 4-step SR approach is used. However, time estimated in either of approaches remains almost the same (i.e., 5.55ms in 2-step approach and 5.33ms in 4-step approach). This depicts the degree of parallelism.

Table 12 also shows the time required for SAD computations for each frame (QCIF resolution) at a maximum clock frequency of 90MHz. Suppose the frame rate of a video sequence be 30fps, then the maximum allowable time that ME engine can take is 33ms for each frame. The architecture described in this paper is a 1-D SAD parallel architecture and this can be extended toward 2-D parallel architecture so as to support higher resolutions.

Table 11 Simulation results of SR Reduction Algorithm

		Threshold	Container	Carphone	Foreman	Football	Hall	Suzie
JM	PSNR(dB)	100.00%	36.348	37.398	36.711	36.617	37.717	37.562
	Bitrate(kbps)	100.00%	45.380	119.000	139.660	692.930	60.630	91.920
SR Reduction Algorithm	PSNR(dB)	99.50%	36.344	37.370	36.693	36.578	37.720	37.567
		99.00%	36.345	37.370	36.692	36.570	37.721	37.560
		97.00%	36.343	37.369	36.680	36.581	37.703	37.554
		95.00%	36.334	37.350	36.642	36.542	37.696	37.545
	Bitrate(kbps)	99.50%	45.270	119.180	139.310	710.950	60.890	92.320
		99.00%	45.270	118.180	140.310	712.930	60.740	92.350
		97.00%	46.880	118.000	141.730	745.820	60.920	92.370
		95.00%	47.250	118.110	150.570	750.800	60.920	94.120
	Δ^{PSNR}	99.50%	-0.004	-0.028	-0.018	-0.039	0.003	0.005
		99.00%	-0.003	-0.028	-0.019	-0.047	0.004	-0.002
		97.00%	-0.005	-0.029	-0.031	-0.036	-0.014	-0.008
		95.00%	-0.014	-0.048	-0.069	-0.075	-0.021	-0.017
	$\Delta^{Bitrate}$	99.50%	-0.110	0.180	-0.350	18.020	0.260	0.400
		99.00%	-0.110	-0.820	0.650	20.000	0.110	0.430
		97.00%	1.500	-1.000	2.070	52.890	0.290	0.450
		95.00%	1.870	-0.890	10.910	57.870	0.290	2.200
	Computational Cost Reduction %	99.50%	97.890	85.940	72.960	42.350	77.440	92.250
		99.00%	98.120	90.890	85.300	62.870	84.500	97.260
		97.00%	99.230	89.720	88.400	66.580	90.310	97.260
		95.00%	99.230	91.120	92.610	71.780	92.550	98.030
	Hardware Utilization	99.50%	14.500	37.500	52.300	75.937	47.500	27.500
		99.00%	13.750	30.200	38.300	60.937	39.375	17.600
		97.00%	8.790	28.200	34.062	56.875	31.125	17.600
		95.00%	8.790	27.630	27.187	53.125	27.347	14.060

4.5 Hardware Implementation Results

Table 13 gives the synthesis result and shows the partial bit stream size of each PRR module. The design is implemented on Xilinx Virtex2Pro (XC2VP30) FPGA development board. Synthesis is done using ISE Foundation Tool. The full and partial bit streams are generated using PlanAhead 10.1 Tool. Bitstreams obtained from the synthesized design are used to reconfigure the FPGA through JTAG.

Table 12 Performance results for different PRR structures

PRR	Steps ±2SR			Steps ±4SR			Steps ±8SR		
	No. of SADs/ 16cycles	Buffer cycles in MB	Time (ms)	No. of SADs/ 16cycles	Buffer cycles in MB	Time (ms)	No. of SADs/ 16cycles	Buffer cycles in MB	Time (ms)
1	4	768	2.96	8	1024	5.33	16	1024	8.56
2	8	1216	5.55	16	1216	8.78	32	1216	14.24
3	12	1408	7.87	24	1152	10.94	-	-	-
4	16	1600	9.20	32	1600	14.66	-	-	-

Table 13 Hardware resources and bitstream information

Module	LUTs	SliceFFs	MUX	BRAMs		PRR1	PRR2	PRR3	PRR4
16x1 PE Array	1036	679	2	4	Bitstreams	105472	105472	90640	90640
Static	996	435	6	6					

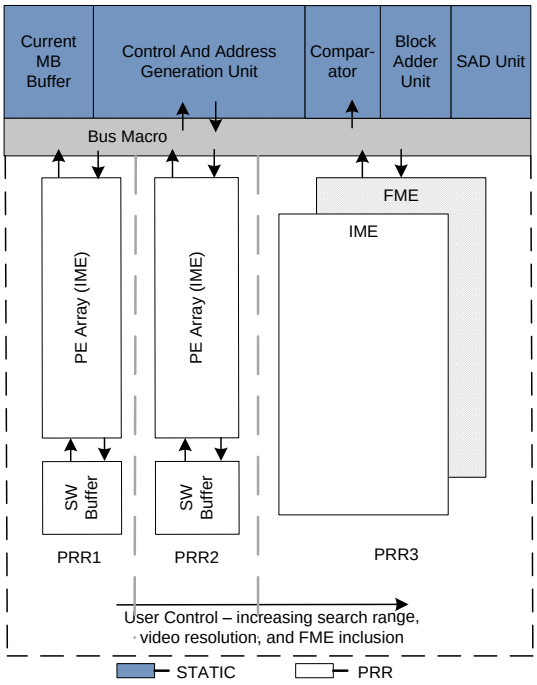

Figure 22 Top view of an integrated Integer-Fractional ME architecture on FPGA

4.6 Extended Approach for time multiplexing PRRs

As discussed in this Chapter, the approach of dynamic Search Range adaptability for Integer Motion Estimation (IME) hardware implementation exploits the very advantage of space-multiplexing the reconfigurable hardware resources. However, in a video coding standard, integer and fractional ME can work together and hence can share hardware resources. Now this Section further extends an idea on benefits of time multiplexing a reconfigurable area between two functionalities, integer and fractional ME. We see from Table 11 & 12 that in particular video scenarios, only a few PRRs are active. So, the unused PRRs can form basis of fractional ME and hence utilizing PR benefits to its best.

Figure 22 depicts an approach on the integrated architecture of Integer-Fractional ME. We now know that depending on the Search Range, video resolution, and frame rate, the PR Region can be active or blank. There are two ways of reconfiguring this PR Region to achieve good scalability of IME computations. Firstly, PE arrays for IME computation can be added using dynamic partial reconfiguration to improve timing constraints for high video formats. Secondly, if it is unused by IME, the PRR can be utilized for Fractional Motion Estimation (FME) functionality using dynamic partial reconfiguration to refine the results. This FME will be especially useful in video sequences involving complex textured pictures, in order to provide better compression performance. In this way ME computations are not interrupted when switching between different numbers of PE arrays. This scenario is particularly useful in applications where hardware resources are critical and demands effective usage.

4.7 Summary

In this Chapter, we presented an improved design of Motion Estimation processing engine which is able to execute for different search range values. The architecture is based on scalable design proposed in Chapter 3.The proposed ME design first introduces a search range reduction technique at frame level, in order to smartly adapt towards various video sequences and at the same time reduce the computational complexity wherever possible. Simulation results show that by reducing the SR, the computational complexity of ME can be reduced significantly with only small degradation in PSNR ($\leq$0.1dB). Results show that our design increases the performance of ME engine for different PRR structures and the architecture can support various resolutions of video sequences with a maximum frequency of 90MHz. Further through partial reconfiguration approach, depending on input search range, it can adopt suitable number of PRRs during runtime without turning off the FPGA. Moreover, it will be also beneficial since small number of PRRs can be selected and used for the FPGA device with less hardware resource available.

CONCLUSIONS

Because of ever growing technology of multimedia, the communication of the image and video data is an essential part. In order to employ effect in a limited transmission bandwidth, in conveying the most high quality user information, it is necessary to have efficient compression method in image and data. Motion Estimation (ME) and Compensation techniques, which can eliminate temporal redundancy between adjacent frames effectively, have been widely applied to the video compression coding standards. A systolic VBSME architecture with advanced search window memory organization is proposed, which is suitable for low-end applications with limited bandwidth. Then, scalable parallel-tree VBSME architecture is discussed for high-end products. The key feature of this VBSME architecture is that the number of PE groups is configurable. For a search range of H×V pixels, where H is width and V is height, up to H SAD groups can be configured to work in parallel with a processing speed of 16 clock cycles to fulfill a VBSME. Search Range reduction based Fast ME algorithm proposed in Chapter 4 further adds to speed-up the process without much degradation in PSNR. By using this algorithm, about 70% of computational savings and about 42% hardware resource savings are achieved.

Traditional IPs implemented on FPGA device provide a simplified solution for a given complex system, but, it is not feasible to achieve adaptive computation capabilities with hardware efficiency. Reconfigurable hardware shows a possibility to adaptively share hardware resources through spatial and time-multiplexing procedures. In this thesis, we explore proposed architectural and algorithmic design approaches for Motion Estimation computations in order to adaptively share hardware resources. The two algorithms: integer and fractional pixel ME are used as an example to analyze the trade-offs between video quality and hardware resources, and through these parameters, the benefits of dynamic partial reconfiguration are explored.

REFERENCES

1. RICHARDSON, I. E. G. 2003. *H.264 and MPEG-4 Video Compression*. Wiley, NJ.
2. WIEGAND, T., AND SULLIVAN, G. J. 2007, The H.264/AVC Video Coding Standard. *IEEE Signal* Processing Magazine 24, Mar., 148-153.
3. H.264/AVC Reference Software Version JM12.4. Available from http://iphome.hhi.de/suehring/tml/.
4. CHUNG, W. C. 2004. Implementing the H.264/AVC Video Coding Standard on FPGAs. Xcell Journal i51.
5. WEDI, T., AND MUSMANN, H. G. 2003, Motion- and aliasing-compensated prediction for hybrid video coding, *IEEE Transactions on Circuits and Systems for Video Technology 13,* 577–587, July 2003.
6. SUBRAMANYA, S. R., PATEL, H., AND ERSOY, I., 2004. Performance evaluation of block-based motion estimation algorithms and distortion measures, *Proc. International Conference on Information Technology: Coding and Computing 2*, 2-7, Apr. 2004.
7. LYSAGHT, P., BLODGET, B., MASON, J., YOUNG, J. 2006, Invited paper: Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs. In Proceedings of International Conference on Field Programmable Logic, 1-6.
8. XILINX 2006. Early Access Partial Reconfiguration. User Guide 208.
9. KOGA, T., LINUMA, K., HIRANO, A., IIJIMA, Y., AND ISHIGURO, T. 1981, Motion compensated interframe coding for video conferencing. *Proc. National Telecomm. Conf. 9*, no. 6, 1–5, 1981.
10. LI, R., ZENG, B., AND LIOU, M. L. 1994, A New Three-step Search Algorithm for BlockMotion Estimation. *IEEE Trans. Circuits Syst. Video Technol. 4*, no. 4, 438-442, Aug. 1994.

11. JAIN, J. R., AND JAIN, A. K. 1981, Displacement measurement and its application in interframe image coding, *IEEE Trans. Comm. 29*, no. 12, 1799-1808, Dec. 1981.
12. KAPPAGANTULA, S., AND RAO, K. R. 1985, Motion Compensated Interframe Image Prediction, *IEEE Trans. Comm. 33*, no. 9, 1011–1015, 1985.
13. RICHMOND II, R. S., AND HA, D. S. 2001, A low-power motion estimation block for low bit-rate wireless video, *ACM International Symposium on Low-Power Electronics Design 2001*, 60-63, Aug. 2001.
14. THAM, J.Y., RANGANATH, S., RANGANATH, M., AND KASSIM, A. A. 1998, A Novel Unrestricted Center-biased Diamond Search Algorithm for Block Motion Estimation, *IEEE Trans. Circuits Syst. Video Technol. 8*, no. 4, 369–377, 1998.
15. HUANG, Y. W., MA, S. Y., SHEN, C. F., AND CHEN, L. G. 2003, Predictive Line Search: An Efficient Motion Estimation Algorithm for MPEG-4 Encoding Systems on Multimedia Processors, *IEEE Trans. Circuits and Syst. Video Technol. 13*, no. 1, 111–117, 2003.
16. TOURAPIS, A. M., AU, O. C., AND LIU, M. L. 2002, Highly Efficient Predictive Zonal Algorithms for Fast Block-matching Motion Estimation, *IEEE Trans. Circuits Syst. Video Technol. 12*, no. 10, 934–947, 2002.
17. LEE, L. W., WANG, J. F., AND LEE, J. Y. 1993, Dynamic search-window adjustment and interlaced search for Block-Matching Algorithm, *IEEE Trans. Circuits Syst. Video Technol. 3*, no. 1, 85-87, Feb. 1993.
18. SHI, Y. Q., AND XIA, X. 1997, A Thresholding Multiresolution Block Matching Algorithm, *IEEE Trans. Circuits Syst. Video Technol. 7*, no. 2, 437-440, Apr. 1997.
19. WANG, B. M., YEN, J. C., AND CHANG, S. 1994, Zero waiting-cycle hierarchical block matching algorithm and its array architectures, *IEEE Trans. Circuits Syst. Video Technol. 4*, no. 1, 18-28, Feb. 1994.

20. LEE, J. H., LIM, K. W., SONG, B. C., AND RA, J. B. 2001, A Fast Multiresolution Block Matching Algorithm and its VLSI Architecture for Low Bit-rate Video Coding, *IEEE Trans. Circuits Syst. Video Technol. 11*, no. 12, 1289– 1301, 2001.
21. BIERLING, M. 1988, Displacement Estimation by Hierarchical Block Matching, *Proc. of SPIE Visual Comm. Image Processing 1988*, 942–951, 1988.
22. WANG, Y., AND KURODA, H. 2000, A Globally Adaptive Pixel decimation Algorithm for Block-motion Estimation, *IEEE Trans. Circuits Syst. Video Technol. 10*, no. 6, 1006–1011, 2000.
23. HE, Z. L., TSUI, C. Y., AND CHAN, K. K. 2000, Low-power VLSI design for motion estimation using adaptive pixel truncation, *IEEE Trans. Circuits Syst. Video Technol. 10*, no. 5, 669-678, Aug. 2000.
24. BEI, C. D., AND GRAY, R. M. 1985, An improvement of minimum distortion encoding algorithm for vector quantization, *IEEE Trans Comm. 33*, no. 10, 1132-1133, Oct. 1985.
25. CHEUNG, C. K., AND PO, L. M. 2000, Normalized Partial Distortion Search Algorithm for Block Motion Estimation, *IEEE Trans Circuits Syst. Video Technol. 10*, no. 3, 417– 422, 2000.
26. LENGWEHASATIT, K., AND ORTEGA, A. 2001, Probabilistic Partialdistance Fast Matching Algorithms for Motion Estimation, IEEE Trans. Circuits Syst. Video Technol. 11, no. 2, 139–152, 2001.
27. KIM , J. N., RYU, T. K., AND JEONG, Y. J. 2006, A Fast Partial Distortion Elimination Algorithm Using Selective Matching Scan, *International Symposium on Computer and Information Sciences (ISCIS 2006)*, 125-133, Nov. 2006.
28. LI, W., AND SALARI, E. 1995, Successive elimination algorithm for motion estimation, *IEEE Trans. Image Processing 4*, no. 1, 105-107, Jan. 1995.
29. CHEN, Y. S., HUANG, Y. P., AND FUH, C. S. 2001, Fast block matching algorithm based on the winner-update strategy, IEEE Trans. Circuits Syst. Video Technol. 10, no. 8, 1212-1222, Aug. 2001.

30. ZHOU, J. L., LI, J., AND YU, S. S. 2004, Modified winner-update search algorithm for fast block matching, *Pattern. Recognition Letters 25*, no. 7, 807-846, May 2004.
31. HSU, C. L., AND HO, M. H. 2007, High-Efficiency VLSI Architecture Design for Motion-Estimation in H.264/AVC, *IEICE Trans. Fundam. Electron. Commun. Comput. Sci. E90-A*, no. 12 2818-2825, Dec. 2007.
32. PAO, I. M., AND SUN, M. T. 1999, Modeling Dot Coefficients for Fast Video Encoding, *IEEE Trans. Circuits Syst. Video Technol. 9*, no. 4, 608–616, 1999.
33. CHEN, M. J., CHIANG, Y. Y., LI, H. J., AND CHI, M. C. 2004, EfficientMulti-frame Motion Estimation Algorithms forMPEG-4 AVC/JVT/H.264, *Proc. of IEEE Int. Symp. Circuits Syst. (ISCAS 2004)*, 737–740, 2004.
34. SU, Y., AND SUN, M. T. 2004, Fast Multiple Reference Frame Motion Estimation for H.264, *Proc. of IEEE International Conference on Multimedia and Expo (ICME 2004)*, no. 1, 695-698, 2004.
35. VOS, L. D., AND STEGHERR, M., 1989, Parameterizable VLSI architectures for the full-search blockmatching algorithm. *IEEE Trans. Circuits Syst. 36*, 1309–1316, Oct. 1989.
36. KOMAREK, T., AND PIRSCH, P. 1989, Array Architectures for Block Matching Algorithms, *IEEE Trans. Circuits Syst. 36*, no. 2,1301–1308, 1989.
37. CHEN, C.Y., CHIEN, S.Y., HUANG, Y.W., CHEN, T.C., WANG, T.C. AND CHEN, L.G. 2006. Analysis and architecture design of variable block-size motion estimation for H.264/AVC. *IEEE Trans. Circuits Syst. 2*, 53, 578-593, Feb. 2006.
38. KIM, M., HWANG, I., AND CHAE, S. 2005. A fast VLSI architecture for full-search variable block size motion estimation in MPEG-4 AVC/H.264. *Proceedings of the 2005 Asia and South Pacific Design Automation Conference (ASP-DAC '05)*, 631-634, Jan. 2005.
39. YANG, K. M., SUN, M. T., AND WU, L., 1989. A family of VLSI designs for motion compensation Block Matching Algorithm. *IEEE Trans. Circuits and Systems 36*, 1317-1325 Oct. 1989.

40. LIU, Z., SONG, Y., IKENAGA, T., AND GOTO, S., 2006, A fine-grain scalable and low memory cost variable block size motion estimation architecture for H.264/AVC. *IEICE Transactions on Electronics 89*, 1928-1936, Dec. 2006.
41. JONG, H. M., CHEN, L. G., AND CHIUEH, T. D. 1994, Parallel Architectures for 3-step Hierarchical Search Block-matching Algorithm, *IEEE Trans. Circuits Syst. Video Technol. 4*, no. 4, 407–416, 1994.
42. CHEN, T. H. 1998, A cost-effective three-step hierarchical search block-matching chip for motion estimation, *IEEE J. Solid State Circuits 33*, no. 8, 1253-1258, Aug. 1998.
43. CHAO, W. M., HSU, C. W., CHANG, Y. C., AND CHEN, L. G. 2002, A Novel Motion Estimator Supporting Diamond Search and Fast full Search, *Proc. of IEEE Int. Symp. Circuits Syst. (ISCAS 2002)*, 492–495, 2002.
44. CHAO, W. M., CHEN, T. C., HSU, C. W., CHANG, Y. C., AND CHEN, L. G. 2003, Computationally Controllable Integer, Half, and Quarter-pel Motion Estimator for MPEG-4 Advanced Simple Profile, *Proc. of IEEE Int. Symp. Circuits Syst. (ISCAS'03)*, 788–791, 2003.
45. HUANG, Y. W., WANG, T. C., HSIEH, B. Y., AND CHEN, L. G. (2003) Hardware architecture design for variable block size motion estimation in MPEG-4 AVC/JVT/ITU-T H.264. In Proceedings of IEEE International Symposium on Circuits and Systems, Bangkok, Thailand. 2, 796-799.
46. ROMA, N., AND SOUSA, L., (2002) Efficient and Configurable Full-Search Blockmatching Processors. IEEE Trans. Circuits Systems Video Technol. 12, 1160–1167.
47. ZHANG, L., AND GAO, W., (2005) Improved FFSBM Algorithm and its VLSI Architecture for Variable Block Size Motion Estimation of H.264, IEEE Int. Symp. Intell. Signal Process. Comm. Syst. 445–448.
48. CHEN, T.-C., CHEN, Y.-H., TSAI, C.-Y., AND CHEN, L.-G. 2006. Low power and power aware fractional motion estimation of H.264/AVC for mobile applications. *In Proceedings of IEEE international symposium on circuits and systems (ISCAS'06)*.

49. JVT-P026, 2005, Fast ME in the JM reference software, Available from wftp3.itu.int/av-arch/jvt-site/2005_07_Poznan/JVT-P026r1.doc.

50. CHEN, Y., CHEN, T., CHIEN, S., HUANG, Y., AND CHEN, L. 2008. VLSI Architecture Design of Fractional Motion Estimation for H.264/AVC. *J. Signal Process. Syst. 53*, 3 (Dec. 2008), 335-347.

51. SONG; T., OGATA, K.; SAITO, K.; SHIMAMOTO, T., 2007. Adaptive Search Range Motion Estimation Algorithm for H.264/AVC.*IEEE International Symposium on Circuits and Systems, 2007*. (May 2007) 3956-3959.

52. CHEN, Z., SONG, Y., IKENAGA, T., AND GOTO, S. 2008. Adaptive Search Range Algorithms for Variable Block Size Motion Estimation in H.264/AVC. *IEICE Trans. Fund. Electron. Comm. Comp. Sci. E91-A*, 4 (Apr. 2008), 1015-1022.

53. HUANG, J., PARRIS, M., LEE, J., AND DEMARA, R. F. 2009. Scalable FPGA-based architecture for DCT computation using dynamic partial reconfiguration. *ACM Trans. Embed. Comput. Syst. 9*, 1 (Oct. 2009), 1-18.

54. HUANG, Y. W., CHEN, C. Y., TSAI, C., SHEN, C. F., AND CHEN, L. G. 2006. Survey on Block Matching Motion Estimation Algorithms and Architectures with New Results. *Journal of VLSI Signal Processing* 42. (Feb. 2006), 297-320.

55. XILINX 2007b. Virtex-4 Family Overview. Xilinx Data Sheet 112.

56. XILINX 2008. Virtex-4 FPGA Configuration. Xilinx User Guide 071.

PUBLICATION

1. GUPTA, S. K., AND LEE, J., 2009, A Scalable H.264/AVC Variable Block Size Motion Estimation Engine Using Partial Reconfiguration. *In Proceedings of International Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, U.S.A, 219-225. July 2009.