ADAPTIVE MOTION ESTIMATION ARCHITECTURE FOR H.264/AVC VIDEO CODEC

by

Yang Song

A Dissertation Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements
For the Degree of

DOCTOR OF PHILOSOPHY

In the Graduate College

THE UNIVERSITY OF ARIZONA

2011

THE UNIVERSITY OF ARIZONA
GRADUATE COLLEGE

As members of the Dissertation Committee, we certify that we have read the dissertation prepared by Yang Song
entitled Adaptive Motion Estimation Architecture for H.264/AVC Video Codec
and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Ali Akoglu — Date: April 18, 2011

Salim Hariri — Date: April 18, 2011

Janet Wang — Date: April 18, 2011

Roman Lysecky — Date: April 18, 2011

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copies of the dissertation to the Graduate College.
I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director: Ali Akoglu — Date: April 18, 2011

STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

SIGNED: Yang Song

ACKNOWLEDGEMENTS

This dissertation holds far more than the culmination of years of doctoral studies. The following pages also reflect the lived experiences and the built relationships with many generous and inspiring people I have met since I started my research work. The list is long, but I appreciate each contribution to the completion of my dissertation:

To my advisor, Dr. Ali Akoglu. Dr. Akoglu has been a strong and supportive advisor to me throughout my doctoral program. I am honored to have had one of the leaders in my professional field as an advisor, a mentor and, a continuous voice of encouragement. In reviewing my writings, he offered thorough and critical comments, but always valuing my ideas. I read his comments on my work with gratitude and somewhat jealousy because he always found ways to significantly improve it. Dr. Akoglu also has helped me immeasurably in my professional development. This dissertation could not have been accomplished without the support of Dr. Akoglu.

To my dissertation committee, Dr. Salim Hariri, Dr. Janet Wang, and Dr. Roman Lysecky, who have generously given their time and expertise to better my work. I thank them for their contribution, encouragement and continuous support.

To my colleagues in the Reconfigurable Computing Laboratory (RCL), especially Ruchika Verma, Xuanxing Xiong, and Gregory Striemer. I will never forget our shared experiences, exchange of ideas and your support when I needed the most. I will keep each and every one of you in my thoughts.

To the ECE Department, especially our academic advisor Tami Whelan. I will always cherish each moment I spent at the department, which I felt was almost a second home.

To my invaluable network of supportive, generous and loving friends: Jifeng Chen, Deqiang Mao, Yi Lu, and Hanyu Liu.

Lastly and most importantly, I wish to thank my parents, Jili Song and Yonghua Jiao. They bore me, raised me, supported me, taught me, and loved me. It was under their watchful eyes that I gained so much drive and an ability to tackle challenges head on. Being the only child of the family, I made a tough decision on coming to the United States to pursue the degree. Now four years have passed, I am proud to pay them back with this dissertation and the Ph.D. degree. My wholehearted gratitude goes to my parents and their sacrifices for my better life. To them and my loving grandmother, who is 88 years old, I dedicate this dissertation.

DEDICATION

To my family.

TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This study contributes to the domain of application specific adaptive hardware architectures with a design approach on processing element array, interconnect structure and memory interface concurrently. As summarized below, our architectural design choices push the limits of on-chip data reuse and avoid redundant computations that are essential for the high throughput, small area, and low power demands of the consumer market.

Motion estimation (ME) is a key component in the H.264/AVC standard. Full Search (FS) based ME achieves optimal peak signal-to-noise-ratio (PSNR), and is the most adopted algorithm for developing hardware motion estimators. In this study, we first design a variable block size motion estimation (VBSME) engine based on hybrid grained processing elements (PEs) and a 2D programmable interconnect structure, which is adaptive to all block size configurations of H.264. PEs operate in bit-serial manner using MSB-first arithmetic for early termination to reduce the amount of computations, and the 2D architecture enables on-chip data reuse between neighboring PEs in a bit-by-bit pipelined fashion. Our design reduces the gate count by 7x compared to its ASIC counterpart, operates at a comparable frequency while sustaining 30 and 60 frames per second (fps); and outperforms bit parallel and bit

serial architectures in terms of throughput and performance per gate.

Numerous fast search algorithms (diamond, hexagon, three-step, etc.) have been developed to reduce the computation burden and the excessive amount of memory transactions required by FS, with a compromise in compression quality. We improve our VBSME engine and introduce the first adaptive ME architecture that provides the end user with the flexibility of choosing between the high quality video service during power-rich state (FS mode), and extended video service (fast search mode). We resolve the irregular indexing scheme challenge of three-step search (3SS) by introducing an on-chip buffer structure with a memory interface, which is adaptive to data access patterns of the FS and 3SS methods. The architecture sustains the real time CIF format (352×288) video encoding at 30fps with an operational frequency as low as 17.6MHz, and consumes 1.98mW based on the 45nm technology, outperforming all other FS and 3SS architectures.

CHAPTER 1

INTRODUCTION

1.1 Background

Block based motion estimation (ME) has been widely used in the video coding standards such as the H.26x and the MPEG-x families (Rao and Hwang, 1996). ME reduces the temporal redundancy that exists between the successive frames of a video sequence to achieve a high data compression ratio. As shown in Figure 1.1, in ME each frame is divided into a number of non-overlapping macroblocks (MBs), which are matched against the candidate blocks within a predetermined window (search range) in the previous frame. The sum of absolute difference (SAD) has been one of the most frequently used distortion criteria, since it provides a good tradeoff between the computational cost and the video coding quality. The SAD is obtained as depicted in Equation 1.1, where $C(m, n)$ denotes the pixel value of the current block ($N \times N$ pixels) at position (m, n); $R(m + v_x, n + v_y)$ represents the pixel value of the reference block with a displacement of v_x and v_y, which are within the search ranges $[-p, p - 1]$. After finding the minimum SAD among all the candidates (e.g., the best matching block), ME delivers the offset between the current MB and the reference block as the motion vector (MV).

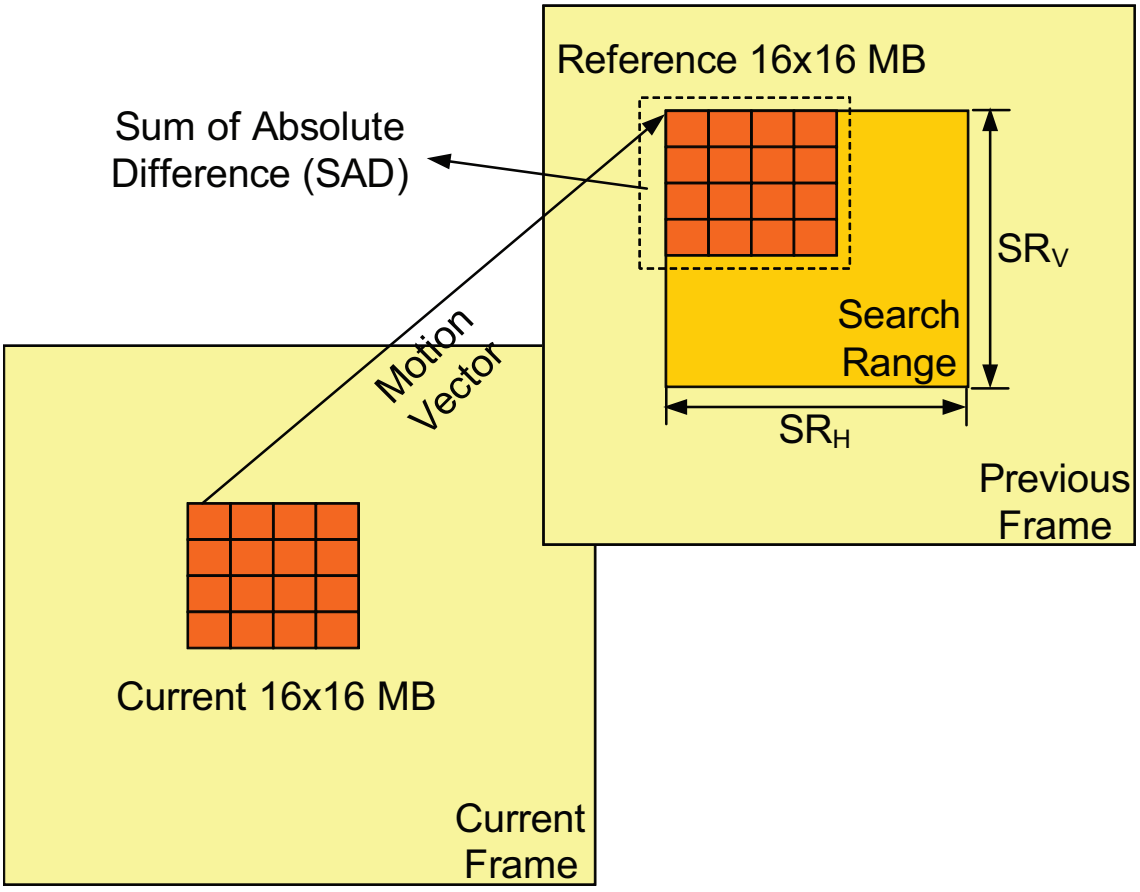


Figure 1.1: Working mechanism of motion estimation: 16×16 block in Current Frame searched within Previous Frame by overlapping the two blocks and computing the sum of absolute differences.

$$SAD(v_x, v_y) = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} |C(m,n) - R(m+v_x, n+v_y)| \tag{1.1}$$

$$-p \leq v_x, v_y \leq p-1 \tag{1.2}$$

$$MV = \{(v_x, v_y) : SAD(v_x, v_y) = \min_{(v_x, v_y)} SAD(v_x, v_y)\} \tag{1.3}$$

The previous video coding standards (H.261, H.263, etc) adopt the fixed block size motion estimation (FBSME), which uses the same block size for both static and moving objects. The latest H.264/AVC (Wiegand et al., July, 2003) provides better estimation of small and irregular motion fields in a video sequence by supporting variable block size ME (VBSME), which segments each MB into seven types of sub-blocks (4×4, 4×8, 8×4, 8×8, 8×16, 16×8, and 16×16), as shown in Figure 1.2.

Various motion estimation algorithms have been proposed for the H.264/AVC. The full search (FS) method has been the most adopted algorithm when developing hardware motion estimators, due to its regularity and data independence. FS (Figure 1.3a) exhaustively evaluates all the candidate blocks to generate the MV and achieves optimal performance in terms of the peak signal-to-noise ratio (PSNR). However, the PSNR performance is achieved at the expense of high computational burden and external memory accesses. According to the profiling studies (Chen et al., June, 2006), the motion estimation routine takes 74.29% (234 Giga Instructions Per Second) of the computations, and 77.49% (365 Giga Bytes Per Second) of the memory accesses for the whole encoder. Therefore, numerous fast search algorithms have been developed to reduce the over-

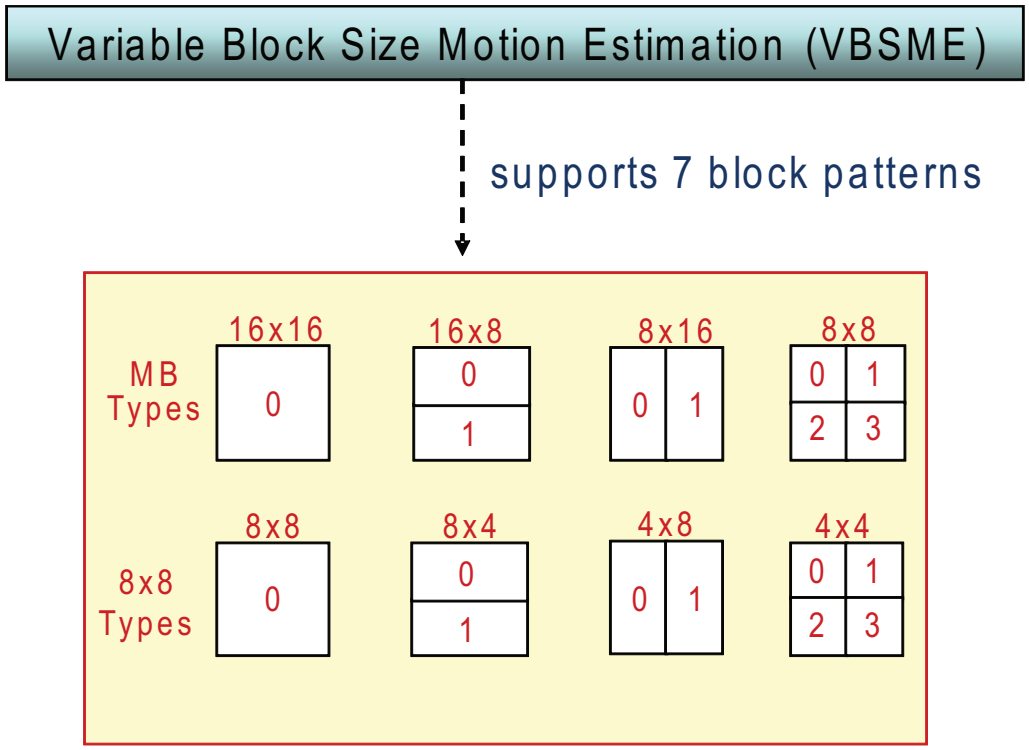
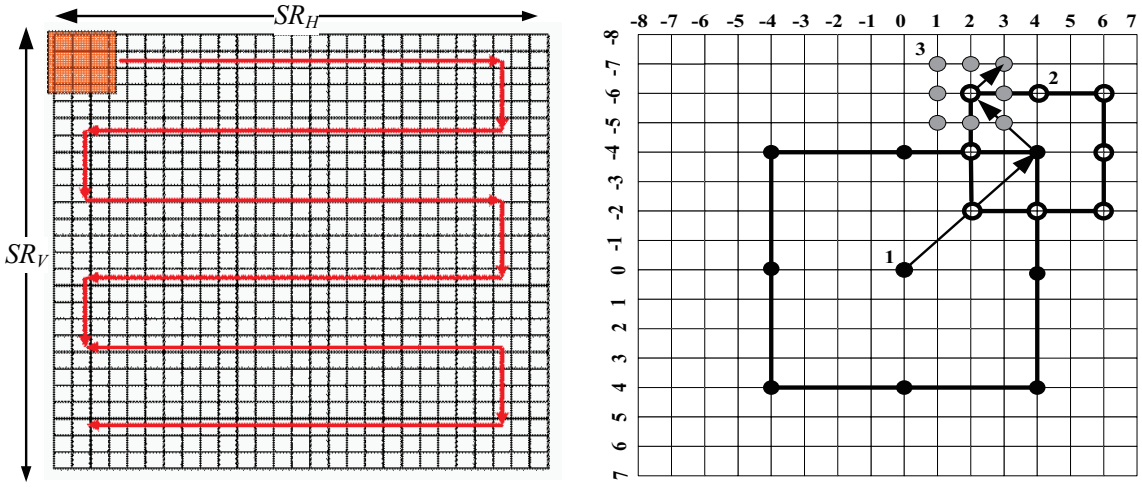


Figure 1.2: Partitioning of a macroblock for motion estimation in H.264.



(a) Full Search

(b) Three-Step Search

Figure 1.3: (a)Full search algorithm, SR_H and SR_V indicate the horizontal and vertical search range, respectively. (b)Three-step search, arrow indicates the direction of the flow in three steps.

head of FS with a compromise in the PSNR performance, including the three-step search (3SS) (Koga et al., 1981), four-step search (4SS) (Po and Ma, June, 1996), block-based gradient descent search (BBGDS) (Liu and Peig, April, 1996), diamond search (DS) (Zhu and Ma, February, 2000), hexagon-based search (HEXS) (Zhu et al., May, 2002), predictive MV field adaptive search technique (PMV-FAST) (Tourapis et al., January, 2001), unsymmetrical-cross multi-hexagongrid search (UMHexagonS) (Chen et al., April, 2006), and enhanced predictive zonal search (EPZS) (Tourapis, January, 2002), etc. For example, the 3SS evaluates a subset of the candidate blocks over a frame as illustrated in Figure 1.3(b). For a typical 16×16 search window size, the 3SS narrows the search space in stages and determines the motion vector after evaluating a total of 25 blocks. On the other hand, the FS method requires 256 such evaluations. While reducing the computation load and the amount of memory transactions, as for any fast search method, the 3SS may get trapped into the local minimum point in the search area, producing a sub-optimal result (i.e., worse PSNR).

1.2 Motivation

Designing an architecture for motion estimation of H.264 poses two challenges.

- Challenge 1–variable block size: Traditional ME hardware architectures are designed only for a fixed block size (either 8×8 or 16×16). H.264 enhances the coding efficiency by supporting variable block size ME (VBSME). Designing

hardware architectures for the VBSME is more complicated, since calculations over the seven block types (Figure 1.2) are needed to generate the motion vector. In other words, designing a processing element that operates at 4×4 granularity level is much simpler than designing a processing element that operates on seven different configurations of block sizes.

- Challenge 2–fast search algorithms: Advances in the mobile communication technologies have enabled portable devices to run complex multimedia applications such as video processing. A desirable solution for portable devices is an adaptive motion estimation engine that is capable of running the ME in full search and fast search modes. Full search mode is desirable for delivering high quality video service during power-rich state. Fast search mode is desirable for reducing the execution time of the ME and delivering an extended video service with a compromise in the compression quality.

  Designing an architecture that supports full search, fast search and the VBSME is a challenging task. Fast search algorithms require complex indexing schemes to access the reference block in the search range, and variations in the block size increases the complexity of the indexing scheme. These factors lead to poor inter-candidate data reuse when compared to the simple indexing scheme of the full search method. To the best of our knowledge, there is no hardware architecture that runs VBSME and is adaptable to full search and fast search modes.

1.3 Contributions

In order to resolve the two challenges identified in Section 1.2, we:

- design a bit serial hybrid grained (BSHG) VBSME engine (Song and Akoglu, 2010) based on hybrid grained processing elements (PEs) and a 2D programmable interconnect structure, which is adaptive to all block size configurations of H.264. PEs operate in bit-serial manner using most significant bit (MSB) first arithmetic for early termination to reduce the amount of computations, and the 2D architecture enables on-chip data reuse between neighboring PEs in a bit-by-bit pipelined fashion. Our design reduces the gate count by 7x compared to its ASIC counterpart, operates at a comparable frequency while sustaining 30 and 60 frames per second (fps); and outperforms the ASIC architectures in terms of throughput and performance per gate.

- improve our BSHG architecture and introduce the first adaptive ME architecture (Song and Akoglu, 2011; Xiong et al., 2011) that provides the end user with the flexibility of choosing between the high quality video service during power-rich state (full search mode), and extended video service (fast search mode). We resolve the irregular indexing scheme challenge of three-step search (3SS) by introducing an on-chip buffer structure with a memory interface, which is adaptive to data access patterns of the FS and 3SS methods. The architecture sustains the real time CIF format (352×288) video encoding at

30fps with an operational frequency as low as 17.6MHz, and consumes 1.98mW based on the 45nm technology, outperforming all other FS and 3SS architectures.

1.4 Organization

The rest of this dissertation is organized as follows. We study the evolution of the ME architectures with a broader perspective in Chapter 2, and then discuss the ASIC based ME architectures in detail in Chapter 3. We introduce the bit serial hybrid grained architecture (BSHG) for the FS, and evaluate its performance in Chapter 4. We then propose the adaptive motion estimation architecture (AMEA) for the FS and the 3SS in Chapter 5 followed by detailed performance analysis with respect to the state of the art. Finally, we present our conclusions and directions for future work in Chapter 6.

CHAPTER 2

RELATED WORK

Figure 2.1 classifies architectures for the SAD routine of ME based on the underlying technology, computation style, and topology. From a historical perspective, implementations of ME have evolved through general purpose processors (Lappalainen et al., 2001; Reader and Meng, 1999; Kuhn, 1999), ASIC (Shen et al., 2001; Yap and McCanny, 2004; Ou et al., 2005; Kim et al., 2005; Chen et al., 2006; Vanne et al., April, 2009), FPGA (Soohoo, 2005), and coarse grained architectures (Ebeling et al., 1999; Verma and Akoglu, 2007).

Several streaming approaches have been proposed for general purpose processors (Lappalainen et al., 2001; Reader and Meng, 1999; Kuhn, 1999). Alternatively, application-specific instruction set processors (ASIPs) allow designers to customize the microprocessor by adding custom instructions and execution units within the processor. However, such extensions are similar to VLIW or SIMD approaches, and are still limited to the data access through the processor's register file. Since our design is an application-specific architecture, we focus on ASIC based solutions in this study during our evaluations and performance comparisons.

Several ASIC based approaches have been proposed for full search based variable block size block matching algorithm to reduce computational complexities. How-

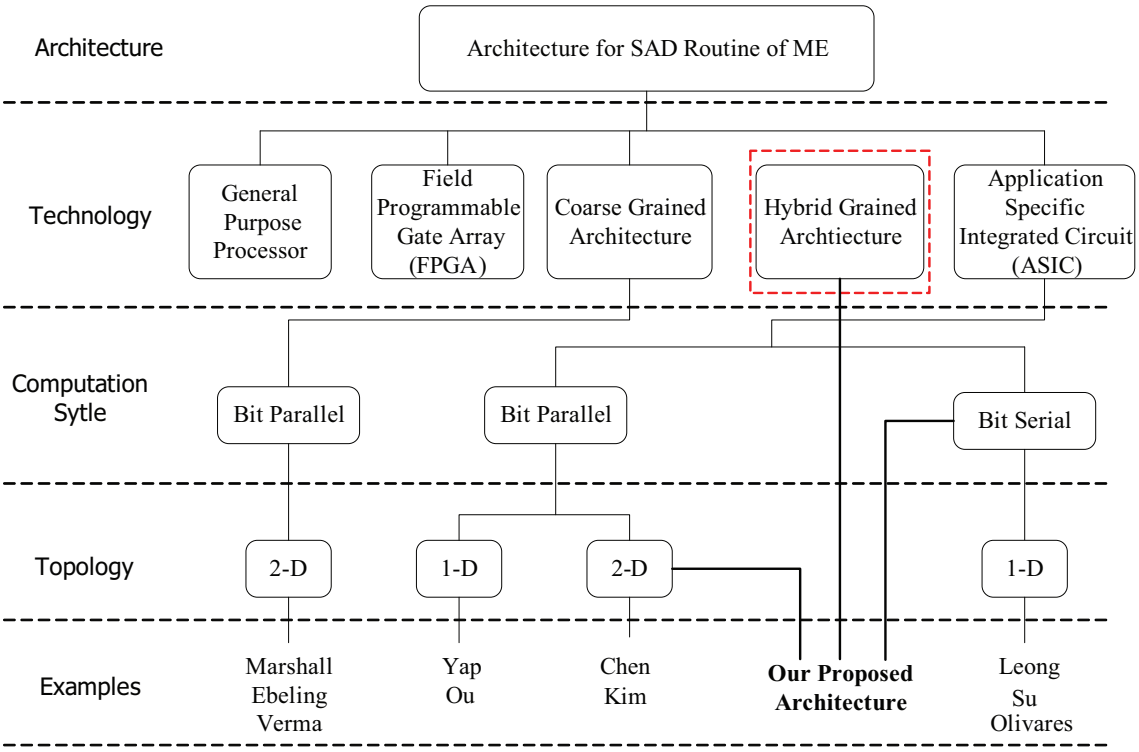


Figure 2.1: Hierarchy spectrum of architecture implementations of SAD routine for motion estimation.

ever, some of these architectures (Kuhn, 1999; Shen et al., 2001; Su and Jen, 2000; Olivares et al., 2006) are capable of processing some of the block sizes specified by the H.264. As an alternative, Yap (Yap and McCanny, 2004) promises to support all block sizes. Their architecture is formed of 16 processing elements (PEs) interconnected as a 1D systolic array where each PE computes a 4×4 SAD. As an improvement, Ou (Ou et al., 2005) introduces a hierarchical 1D systolic architecture that employs partial SAD computation technique. Additionally in (Ou et al., 2005), a *VBSME processor*, based on an adder tree structure, supports all block sizes. In overall, Ou (Ou et al., 2005) achieves lower latencies with higher throughput compared to existing VBSME architectures. However partial SAD computation requires delay registers and extra accumulators; and *VBSME processor* consists of a fixed set of redundant adders to accommodate the need for variable size adders based on the block sizes. Therefore this architecture has high area overhead which can be improved by using a flexible routing architecture.

Bitwise operation based applications (e.g. cryptography) have been successfully mapped onto fine grained reconfigurable architectures such as FPGAs with significant speedups. Granularity of a functional unit on a reconfigurable platform varies from single Configurable Logic Block (CLB), to a processing element (PE) either composed of multiple CLBs or to higher granularity level where each PE handles all types of operations required by the target application. However, there is a tradeoff between the complexity of logic blocks and area efficiency. For fine grained archi-

tecture, more logic blocks will be required to implement the circuit and routing area becomes excessive. Coarse grained architecture decreases total number of logic blocks hence interconnect complexity is reduced by localizing the connections. In general, coarse grained reconfigurable fabrics are composed of high level functional blocks with generic reconfigurable interconnect network.

Hence, coarse grained reconfigurable architectures RAW (Waingold et al., 1997), ChESS (Marshall et al., 1999), MATRIX (Mirsky and DeHon, 1996) and RaPiD (Ebeling et al., 1999) have been introduced to overcome the drawbacks of lookup table based fine grained reconfigurable architectures. While fewer configuration bits are needed for the PEs, fully utilizing the functionality of each PE is difficult, leading to significant underutilization of coarse grained fabrics. This can be avoided by tailoring the architecture to the computation characteristics of the algorithm with application specific hybrid grained processing elements interconnected. Motion Estimation is a highly critical algorithm for the video compression domain which involves recurring and parallel computation patterns demanding high flexibility. Therefore, we address the flexibility demand of VBSME by our application specific hybrid grained architecture (as shown in Figure 2.1). Since an application specific architecture is expected to be superior to coarse grain reconfigurable architectures in terms of performance, we analyze and compare the results of our architecture against only ASIC based approaches for fair comparison.

CHAPTER 3

ASIC APPROACHES

ASIC based architectures can be broadly classified into different categories depending upon various metrics such as style of the computation, topology of processing elements and methodology for accumulation of SADs etc, as shown in Figure 2.1. Based on the computation style, we categorize the architectures into bit parallel and bit serial operations. Most of the reported full search (FS) architectures were implemented using bit parallel approaches. The definition of bit parallel originates from the fact that in such architectures, all 8 bits of a pixel are transmitted and processed at the same time. The bit parallel architectures have the advantages of better data reuse, easier control and simpler design. On the other hand, bit serial implementations process the data loading, the absolute difference, and the summation from the most significant bit (MSB) to the least significant bit (LSB) and take advantage of the early termination scheme.

Based on methodology for accumulation of SADs, we classify the architectures as partial and parallel sum SADs. In partial sum SAD architectures, reference pixels are broadcasted and SAD computation for each 4×4 subblock is pipelined. Each processing element (PE) computes one pixel difference, accumulates it to the previous partial SAD and sends the computed partial SAD to the next processing

element. This kind of architecture uses large number of storage registers due to the accumulation of partial SADs in each PE. However, in parallel sum SAD architectures, all pixel differences for a 4×4 sub-block are computed concurrently and thus added in one clock cycle. In the parallel sum architecture, reference pixels are reused between different PEs which help decrease memory bandwidth requirements. The direction of data transfer among different PEs depends on the search pattern adopted.

Based on the topology, we classify the architectures as 1-D and 2-D. In 1-D structures, each PE calculates the absolute difference between reference and current block, adds the result to the already calculated partial sum for the same search position given by the neighboring PE, and passes the result to the next PE. At the end of the chain of PEs, the SAD calculation is finished. This architecture is scalable for search range and block size. The area used is small but the performance is slow compared to a 2-D array. In 2-D architectures, data reuse is possible by making use of delay registers and by moving data from one PE to the next. This design offers the advantage of further reducing memory bandwidth compared to the 1-D architecture. The current data is initially shifted to each PE and later stored and reused until the current block motion vector is found. 2-D architecture requires large area but the performance is high because of the number of parallel computations. We present an overview of the full search and fast search method based architectures in the following paragraphs.

3.1 Full Search Architecture

3.1.1 Bit Parallel, 1-D, and Partial Sum

Figure 3.1 depicts the bit parallel 1-D partial sum architecture (Ou et al., 2005), which promises to have lower latency and higher throughput over other existing VBSME architectures till date. This architecture consists of 16 separate SAD modules for processing sixteen 4×4 motion vectors. It also consists of a chain of adders and comparators, which the paper refers to as *VBSME processor*, used for the computation of larger block SADs. Each 1-D array consists of a 1-D systolic array of 4 PEs (Figure 3.1). The produced 4×4 SADs are then sent through a fixed series of adders and comparators to produce 4×4 motion vectors. The 4×4 SADs are also sent in parallel to four sets of adders, comparators to produce 4×8, 8×4 SADs. The two 8×4 SADs are then again sent to form 8×8 SAD. Similarly, 16×8, 8×16, and 16×16 SADs are computed through a set of adders and comparators. This architecture has separate SAD modules for 4×4 subblock computations with separate input ports for loading current block and search region data. Thus, it does not support reuse of search data between modules. This increases the amount of data required from the memory. Also, the VBSME is supported by a fixed set of adders based on a large adder tree (as shown in Figure 3.2). This leads to resource wastage which can be avoided by using intelligent routing scheme to use same set of adders to compute different motion vectors each time. Furthermore, this architecture does not support

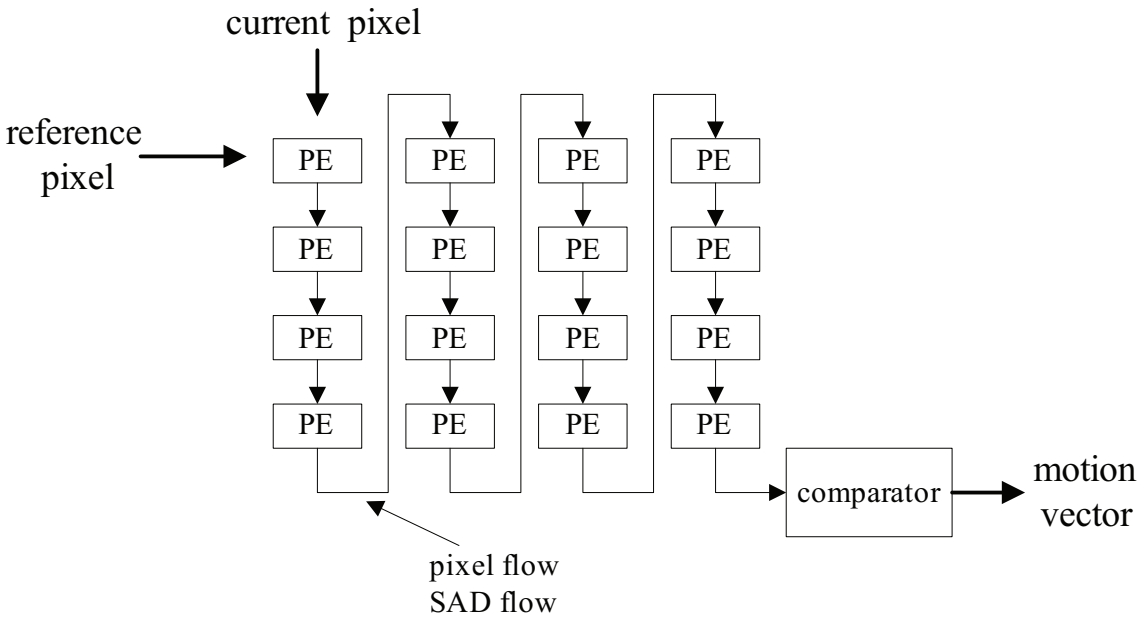


Figure 3.1: Bit parallel, 1-D, partial sum architecture, designed for 4×4 block size.

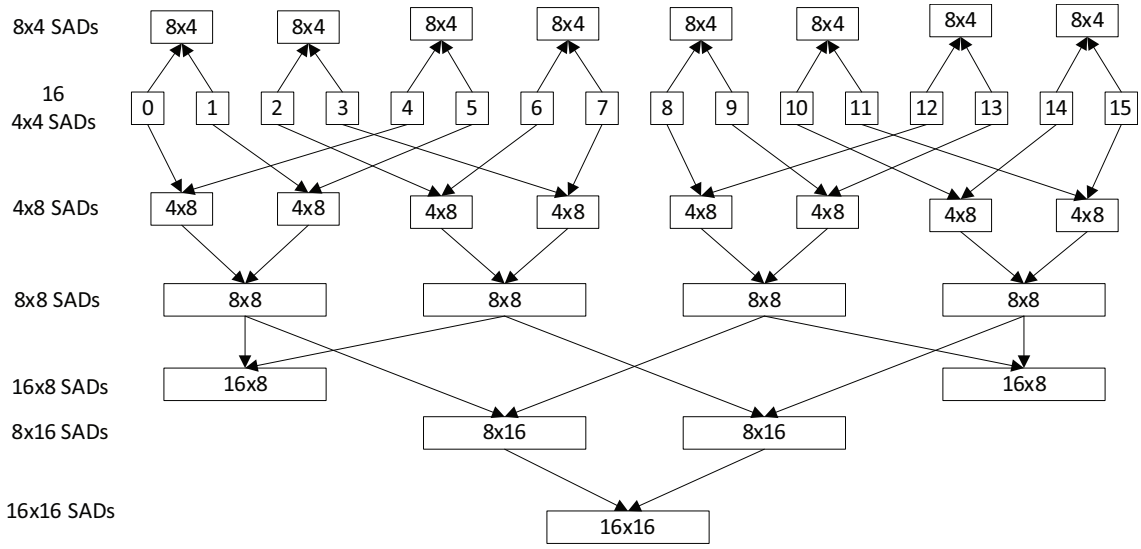


Figure 3.2: The adder tree dedicated for VBSME, used in either 1D or 2D architectures.

other fast search algorithms like diamond search, hexagonal search, etc.

3.1.2 Bit Parallel, 2-D, and Partial Sum

Figure 3.3 shows the datapath of bit parallel 2-D partial sum architecture (Shen et al., 2001). It consists of processing elements connected in a mesh-based architecture where SAD computations are direct-mapped. Each PE includes at least two registers to save the pixels from current block and reference block. At every cycle, each PE computes the absolute difference between two registers. The partial result is passed vertically to the end of the PE chain, where an adder is responsible to sum up the temporary SADs. After getting sixteen 4×4 SADs, the datapath directs them to the adder and the comparator to form SADs for 7 different block sizes. Thus it supports VBSME and finds the minimum SAD and corresponding motion vector. In this architecture, the reference data is passed horizontally from one processing element to the next. This architecture requires large area but the performance is high because it can be pipelined by using preloaded registers.

3.1.3 Bit Parallel, 1-D, and Parallel Sum

Figure 3.4 illustrates the 1-D parallel sum architecture (Jehng et al., 1993). Each PE computes the absolute difference between current and reference block. The differences are concurrently accumulated by the adders that comprise the binary tree architecture. The tree structure supports not only full search block matching

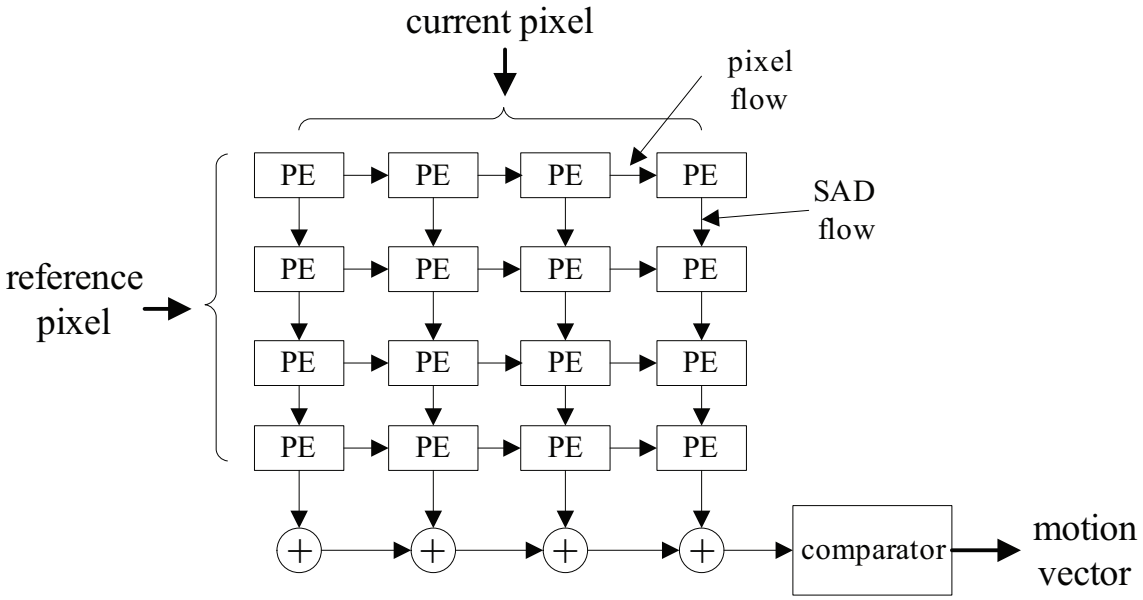


Figure 3.3: Bit parallel, 2-D, partial sum architecture, designed for 4×4 block size.

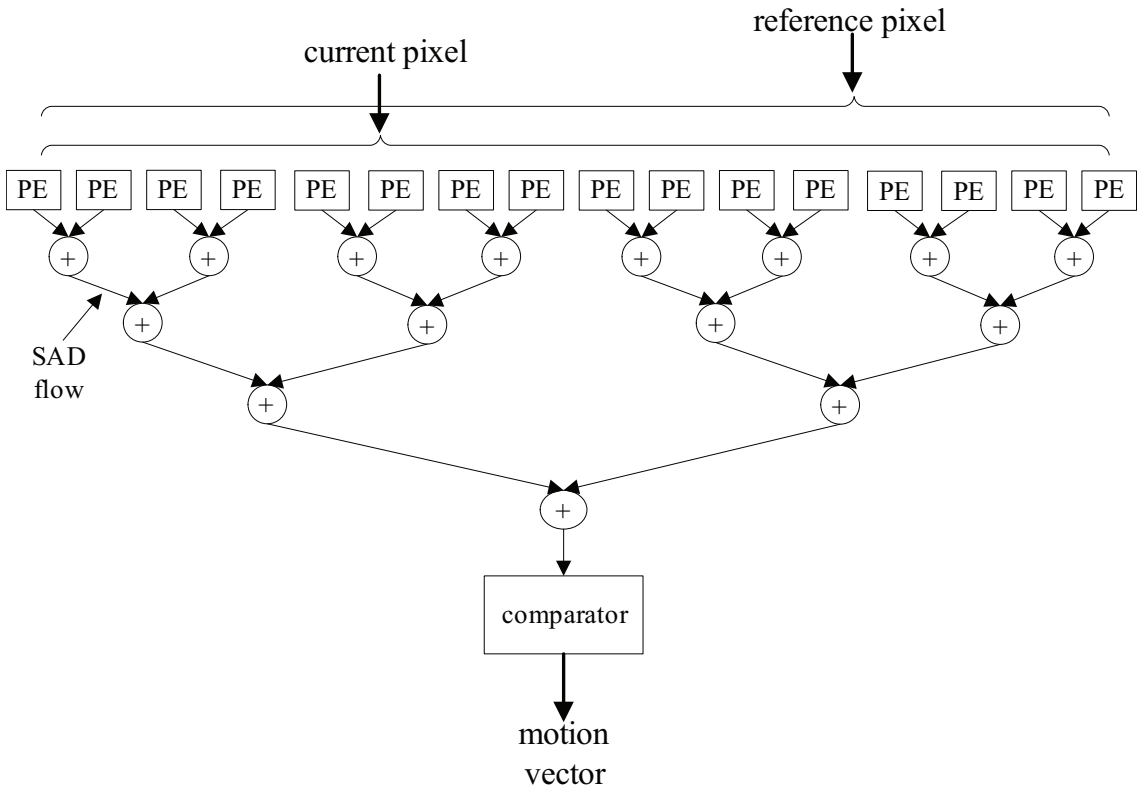


Figure 3.4: Bit parallel, 1-D, parallel sum architecture, designed for 4×4 block size.

algorithms (FSBMA) but also fast algorithms by decimating search positions, such as three step search. Concepts of memory interleaving and pipeline interleaving were proposed to enhance the supported memory bandwidth and to prevent the hardware from idling, respectively. However, when the block size increases, the required bitwidth of full tree becomes too large.

3.1.4 Bit Parallel, 2-D, and Parallel Sum

Figure 3.5 presents the datapath for bit parallel 2-D parallel sum architecture (Vos and Stegherr, 1989). The pixels of current block are kept in the corresponding PEs. All pixels of a reference block are properly move to the corresponding PEs. The last PE in each column outputs the SAD of a column, and an adder tree calculates the final SAD of a candidate block in parallel. Architecture supports full search by snake like scan when sweeping the search positions. The first row of search positions is scanned from left to right. Then the second row is scanned from right to left, followed by the third row scanned from left to right, and so on. In order to successively compute SAD values without extra cycles to load pixels, two sets of registers are required to prepare the proper search area pixels before hand and the data path should be periodically configured. The large register sets is a tradeoff for the bitwidth of memory access. The utilization of this architecture is higher (produces one SAD value of a search position per cycle) for larger search range application.

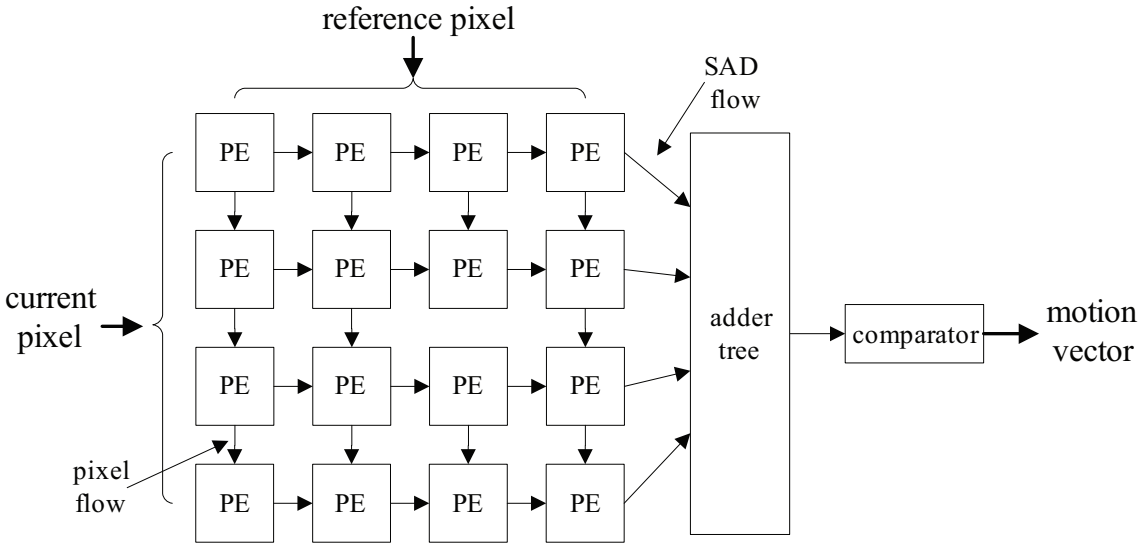


Figure 3.5: Bit parallel, 2-D, parallel sum architecture, designed for 4×4 block size.

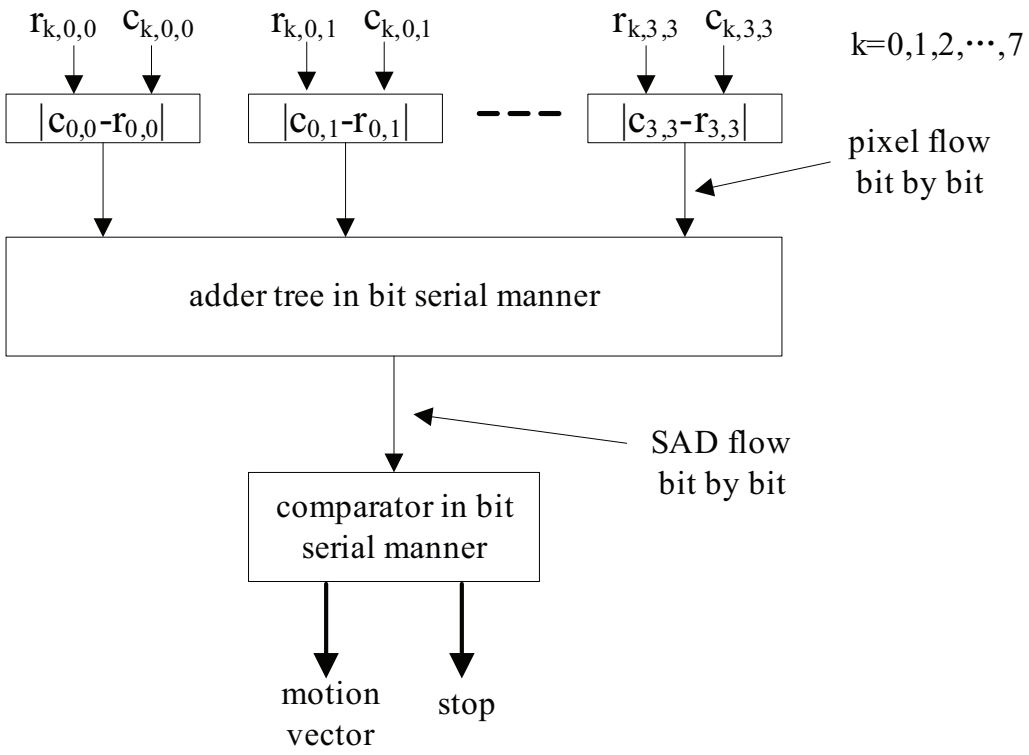


Figure 3.6: Bit serial, 1-D, MSB-first SAD processor architecture, designed for 4×4 block size.

In summary, the bit parallel architectures take advantage of the adder tree (Figure 3.2) to implement VBSME. High performance is achieved because the SAD computations can be pipelined by reusing the data in the preloaded registers. However, during the SAD calculations, comparison is carried out only after the summation of all pixel differences. In some cases, the current SAD reaches a value that is larger than the stored minimum SAD where the subsequent additions of the pixel differences to the current SAD becomes unnecessary, but the bit parallel architecture is not able to terminate the computation.

3.1.5 Bit Serial

Bit serial arithmetic performs operations on most significant bit (MSB) first (Li and Leong, 2008; Olivares et al., 2006; Su and Jen, 2000) providing the architecture with the early termination capability. Figure 3.6 illustrates a simplified bit serial MSB-first architecture. $c_{k,i,j}$ and $r_{k,i,j}$ denote the kth (k=0, 1, 2, ..., 7) bit of the current and reference pixels at location (i, j). The absolute value of the differences is computed for each pair of pixels ($|c_{i,j} - r_{i,j}|$), and their summation is calculated using the full adder tree in bit serial manner. The result is stored bit-by-bit in a SAD register and simultaneously compared with the corresponding bit of the SAD in the comparator. If at any point, the current SAD is larger than the stored minimum SAD, the comparator terminates the computation and chooses a new candidate block. Otherwise, the accumulation continues until finally the stored

minimum is replaced by the current SAD. Bit serial architecture reduces the number of data buses connecting blocks due to its serial nature. Early termination allows subsequent SAD calculations to start earlier, and eliminates the carry propagation chains. However, the bit serial architecture suffers from one drawback. It lacks the data reuse capability inside the PEs, thus results with a large amount of memory accesses.

3.2 Fast Search Architectures

Jung (Jung and Lee, May, 2004) introduces a 4-way pipelined processing architecture for the 3SS with a fixed block size, which consists of nine PEs to evaluate the nine candidate points in parallel. This approach is based on dividing the current block and search area into 4 subregions and processing them concurrently. Jung also develops a memory partitioning method to access pixel data from 4 subregions without memory conflict. Generating the motion vector for a 16×16 block requires 337 clock cycles. The architecture operates up to 100MHz and can process 311K blocks per second.

Chen proposes a content adaptive parallel variable block size four-step search (4SS) algorithm, and designs a low-power architecture in (Chen et al., April, 2007). This work modifies the original 4SS algorithm specifically for hardware implementation, and proposes a 2-D random access memory architecture for on-chip search range buffer. Low power dissipation is achieved by enabling intra-/inter-candidate

data reuse. Chen reduces the computational complexity and the memory bandwidth by 97% and 77.6% respectively over the full search method. Particularly, the power consumption is 2.13 mW for real-time encoding CIF (352×288, 30fps) videos with an operational frequency of 13.5 MHz.

Saponara (Saponara and Fanucci, January, 2004) presents a low-power VLSI architecture based on a data-adaptive ME algorithm with a fixed block size. The fast search algorithms break the regularity of the full search, which leads to poor data reuse. Saponara exploits the input data variations to dynamically reconfigure the search-window size to achieve the same performance as the FS, while reducing the power consumption. The architecture is compared with the conventional full search as well as other low-complexity ME techniques. Saponara reduces the power consumption to below 12 and 47 mW for the 30 fps QCIF and CIF cases, respectively.

3.3 Summary

In this chapter, we categorize the ASIC architectures based on computation style, accumulation type, and topology. We discuss their design methodologies with respect to processing speed and analyze their pros and cons. We observe that there is a trade off between 2D and bit serial architectures. An effective architecture should carry the data reuse capability of the 2D architectures and early termination of the bit serial architectures. The main challenge is to combine these features under one

architecture. In Chapter 4, we show how to design a full search ME architecture that takes advantage of both the data reuse capability from the bit parallel architectures and the early termination scheme from the bit serial architectures.

CHAPTER 4

BIT SERIAL HYBRID GRAINED ARCHITECTURE

In this chapter, we first give an overview of the MSB-first algorithm as it forms the baseline for our discussion on the performance and delay cycles of our architecture due to MSB-first arithmetic. We then describe our bit serial and hybrid grained (BSHG) 2D architecture. We explain how BSHG supports variable block size ME in detail and illustrate the functionality of BSHG for full search algorithm. Finally, we compare the performance of BSHG against the previously reported ASIC architectures.

4.1 MSB-first Arithmetic

4.1.1 Overview

We illustrate how minimum SAD is determined with a typical bit parallel architecture for a 4×4 block size in Figure 4.1(a). Each dot in the RB and CB indicates an 8-bit pixel. 16 pairwise absolute differences between the current and reference pixels (8-bit) are calculated and then accumulated into a 12-bit register ($SAD_{current}$). The result is then compared with the current minimum SAD (SAD_{min}) to decide whether to update the contents of the SAD_{min} or not. As illustrated in Table 4.2, let's assume that the SAD_{min} is $(011111111111)_2$ and the $SAD_{current}$ is $(100000000000)_2$.

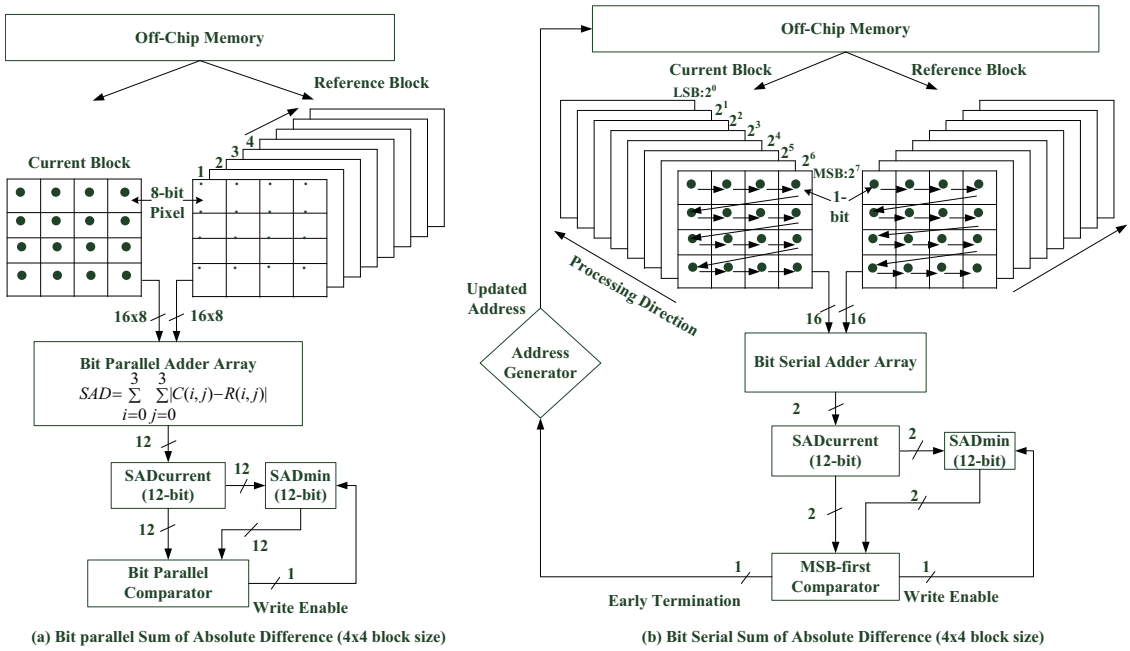
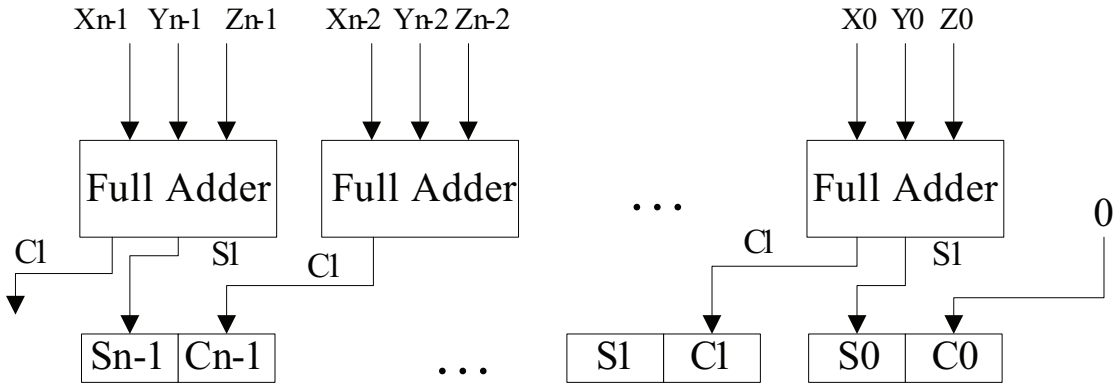


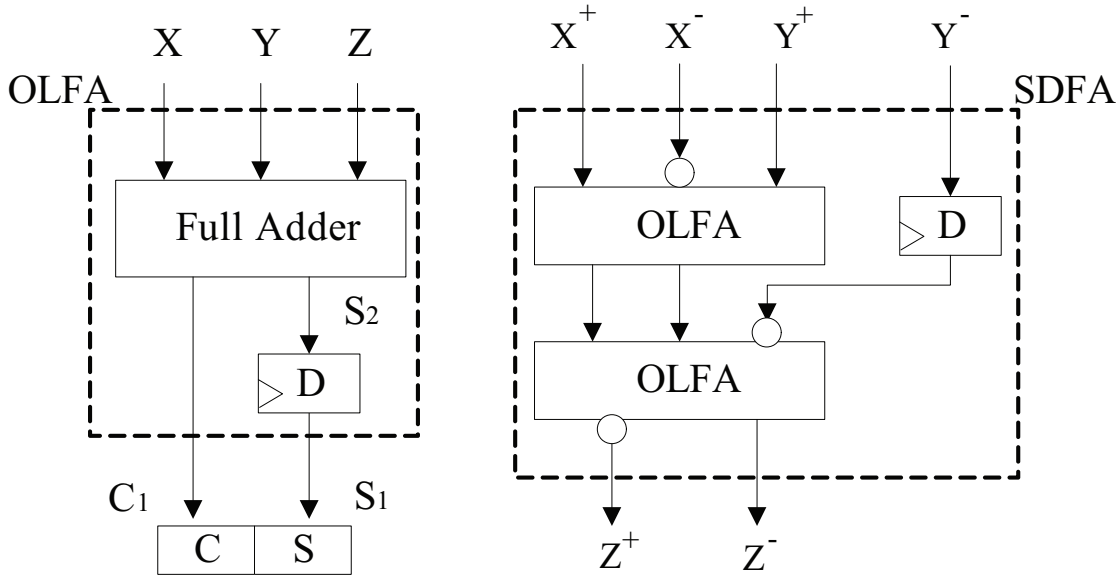
Figure 4.1: In (a), each dot indicates an eight-bit pixel. After matching the current block with the reference block 1, SAD computation is repeated on the reference block 2, 3, etc. In (b), each dot indicates one bit. After accumulating the MSBs of all the 16 pixel pairs, we move to the next bits until the LSB is reached. All the 16 pixel pairs are accumulated by the adder array and the $SAD_{current}$ is compared with the SAD_{min}. $WriteEnable$ is asserted if the $SAD_{current}$ is smaller than the SAD_{min}.

Since $SAD_{current}$ is larger than the SAD_{min}, contents of the SAD_{min} is not updated. However, if we had a way to calculate the most significant bit first for the $SAD_{current}$, we would eliminate the calculation of the remaining bits as the MSB is enough to determine that the $SAD_{current}$ will be larger than the SAD_{min}. MSB-first arithmetic allows this comparison to start with the most significant bit, which enables early termination without having to carry out the calculations for the remaining bits. However, if the $SAD_{current}$ is $(011111111110)_2$, the decision can be reached only with the least significant bit. In this worst case scenario, early termination is not possible. Therefore, we design our datapath completely to operate in bit-by-bit style to take advantage of the early termination when possible.

We illustrate the simplified MSB-first motion estimation system with a 4×4 block and 8-bit (2^7, 2^6,..., 2^0) pixels in Figure 4.1(b), where each dot indicates one bit of the RB and CB pixels. We start accumulating the difference of the 16 MSB pairs from the CB and the RB by using the bit serial adder array. The accumulated result is compared with the MSB of the stored SADmin. If the early termination decision cannot be reached, then we move to the next MSB. This process is iterated until the LSB is reached. The current SAD result is simultaneously compared with the minimum SAD bit by bit. If early termination occurs, the rest of the computations which contributes to the current SAD become unnecessary. The comparator notifies the address generator to update the reference block to start a new SAD computation.



(a) conventional carry save adder



(b) online full adder(OLFA) (c) signed-digit full adder(SDFA)

Figure 4.2: (a) carry save adder. (b) online full adder (OLFA): $(X+Y+Z)_{MSB-first} = (C+S)_{MSB-first}$ with 1 cycle online delay. (c) signed-digit full adder (SDFA): $X_{SD} + Y_{SD} = Z_{SD}$ with 2 cycles online delay.

Table 4.1: An example of the early termination. SAD_{min} represents the stored winner (the minimum in the register). $SAD_{current}$ represents for the SAD that is currently being computed. Assume the block size is 4×4. $SAD_{min} = (2047)_{10} = (011111111111)_2$. $SAD_{current} = (2048)_{10} = (100000000000)_2$ for the best case. $SAD_{current} = (2046)_{10} = (011111111110)_2$ for the worst case.

	MSB 11th	10th	9th	8th	7th	6th	5th	4th	3rd	2nd	1st	LSB 0th
Current SAD_{min}	**0**	1	1	1	1	1	1	1	1	1	1	**1**
Best Case $SAD_{current}$	**1**	0	0	0	0	0	0	0	0	0	0	0
Worst Case $SAD_{current}$	0	1	1	1	1	1	1	1	1	1	1	**0**

4.1.2 Bit Serial Algorithms

Conventional bit parallel addition architectures that operate based on least significant bit (LSB) first have carry propagation latency. Figure 4.2(a) shows the structure of a conventional carry save adder. We assume that X, Y and Z are three n-bit numbers in a two's complement system. A full adder (FA) performs the addition of three bits and generates the carry and sum bits. Figure 4.2(a) relies on parallel full adders, and shifts the carry bit to the left to complete the addition.

On the other hand, the MSB-first mode of computation requires a flexibility which is achieved by adopting redundant number systems (Ercegovac and Lang, 1989). Figure 4.2(b) shows an online full adder (OLFA) (Olivares et al., 2006) which performs addition with MSB-first. The result of OLFA is always given in carry-save (CS) representation. The OLFA has 1 full adder (FA) and 1-bit delay register D (s1 and s2 are two latches). The delay register synchronizes the generation of carry

Table 4.2: Online full adder example. X=$00001110_2(14_{10})$, Y=$01001001_2(73_{10})$, Z=$00000111_2(7_{10})$. Result [C+S] = $001011110_2(94_{10})$. Cycle 9, 0s padded to inputs X, Y and Z. Bold 0s highlight the requirement to compensate the online delay introduced by the MSB-first arithmetic.

cycle	X	Y	Z	C_1	S_2	S_1	C	S
1	0	0	0	0	0	0	-	-
2	0	1	0	0	1	0	0	0
3	0	0	0	0	0	1	0	0
4	0	0	0	0	0	0	0	1
5	1	1	0	1	0	0	0	0
6	1	0	1	1	0	0	1	0
7	1	0	1	1	0	0	1	0
8	0	1	1	1	0	0	1	0
9	**0**	**0**	**0**	0	0	0	1	0
10							0	0

and sum bits. Table 4.2 shows an example of the online addition of three 8-bit two's complement numbers on the architecture illustrated by Figure 4.2(b). This simple example is important for highlighting the amount of online delays and the 0 padding requirement which will be referred in Section 4.2.4. In this example, X, Y and Z are loaded to the OLFA from MSB to LSB. Since the output is latched into registers C1 and S2, it takes one cycle to generate the MSB of the result using OLFA, which is called online delay. One should also note the 0 padding during cycle 9. In this cycle, the final C and S values are being generated, therefore, in order to compensate for this stage, 0s are padded into X, Y and Z.

In our bit serial design we use a radix-2 Avizienis' signed-digit (SD) number

Table 4.3: Codification in radix-2 signed-digit system.

Signed-digit value x_i	Digit representation (x_i^+, x_i^-)
$+1$	$(1,0)$
-1	$(0,1)$
0	$(0,0)$
0	$(1,1)$

system (Avizienis, 1961) for the MSB-first arithmetic. Each digit in this system is coded using 2 unsigned binary bits, one positive and one negative, as $x_i{=}x_i^+{-}x_i^-$, where x_i^+,$x_i^-{\in}\{0,1\}$ and $x_i{\in}\{-1,0,1\}$. Table 4.3 shows the digit set and its representation in SD system. We observe a redundancy in the representation of 0, since it can be either (0,0) or (1,1) in the radix-2 SD system. Figure 4.2(c) shows an online signed-digit full adder (SDFA) which normally performs addition of the numbers in signed-digit representations ($X_{SD} + Y_{SD} = Z_{SD}$) where X^+ and Y^+ are the positive operands of two SD numbers X and Y, while X^- and Y^- are the negative operands. The SDFA requires a total of 2 online delays as there are 2 levels of OLFAs, each with 1 online delay. In the following paragraphs, we describe and illustrate the conversion of the standard SAD calculation shown in Equation 1.1 into the SD based calculation.

We rewrite the equation for absolute difference as shown in Equation 5.8, where $AD(i,j)$ represents the absolute difference between the current pixel and the refer-

ence pixel at location (i, j). k stands for the kth bit in the pixel.

$$\begin{aligned} AD(i,j) &= |CMB(i,j) - RB(i+m, j+n)| \\ &= \left| \sum_{k=0}^{k=7} 2^k \times (CMB(i,j,k) - RB(i+m, j+n, k)) \right| \end{aligned} \tag{4.1}$$

$$SAD = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} AD_{i,j} \tag{4.2}$$

The pair $CMB(i,j,k) - RB(i+m, j+n, k)$ in Equation 5.8 can be seen as a SD number, with $CMB(i,j,k)$ as the positive and $RB(i+m, j+n, k)$ negative operand. The sign of $CMB(i,j,k) - RB(i+m, j+n, k)$ should be changed if the end result is negative. In the SD system, the negation operation is performed by swapping the positive operand with the negative operand. The absolute difference (ABS) unit detects the sign by finding the first non-equal bit. If it is positive (10), which means the positive operand is larger than the negative operand, then the SD number is positive and no change is made. Nevertheless, if the first non-equal bit is negative (01), the ABS unit interchanges the two operands to make the SD number positive. For example, let $x_i^+ = 10011000_2(152_{10})$ and $x_i^- = 10111110_2(190_{10})$, because the third MSB of x_i^- is larger than that of x_i^+, operation $(x_i^+ - x_i^-)$ will result with a negative value. This requires swapping of the two operands eliminating the need for the absolute operation. The circuit that implements this functionality requires few hardware resources and introduces little computational delay. Figure 4.3 illustrates this sign detection.

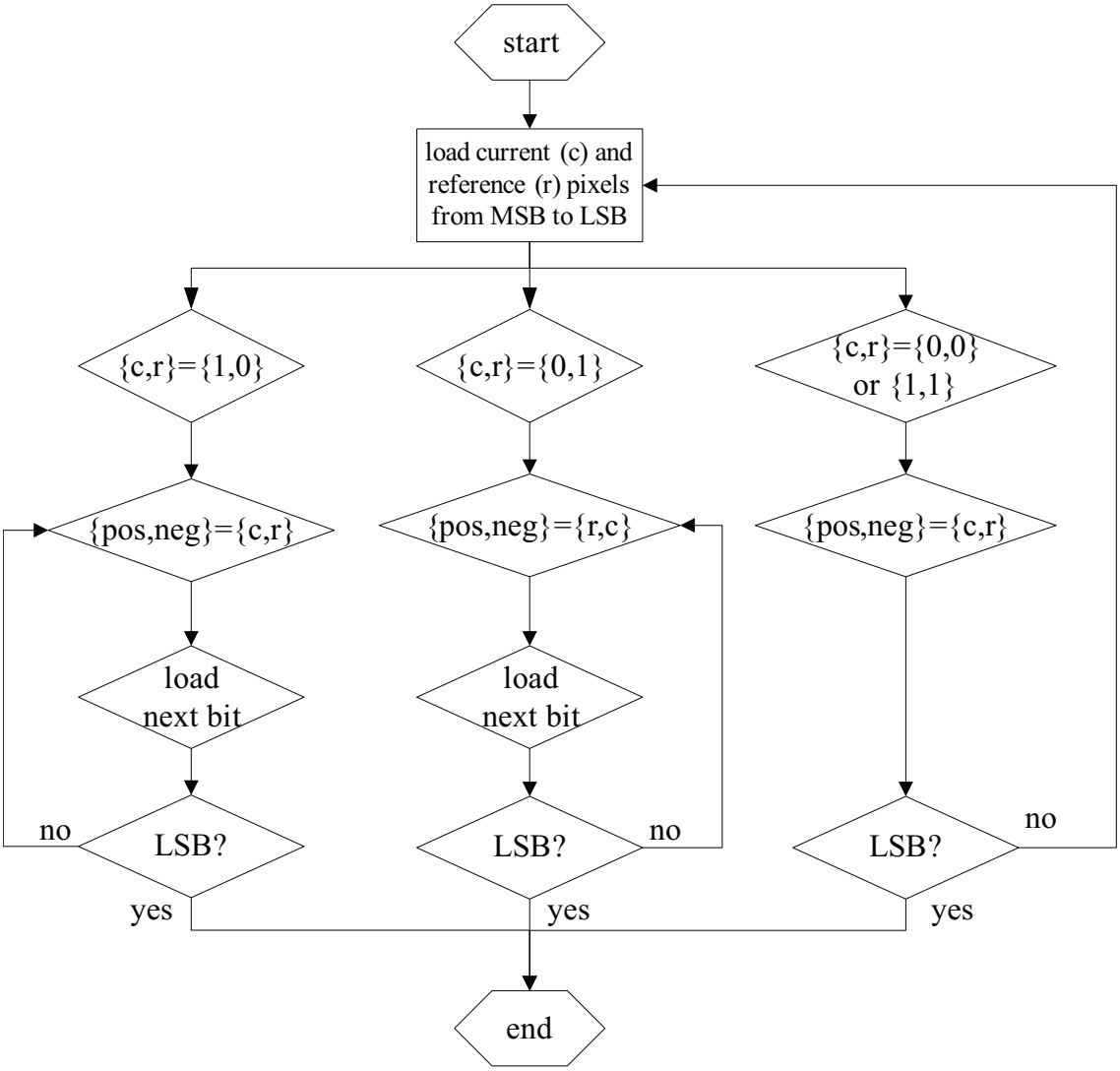


Figure 4.3: Flow chart of the algorithm for converting two positive integers to signed-digit numbers.

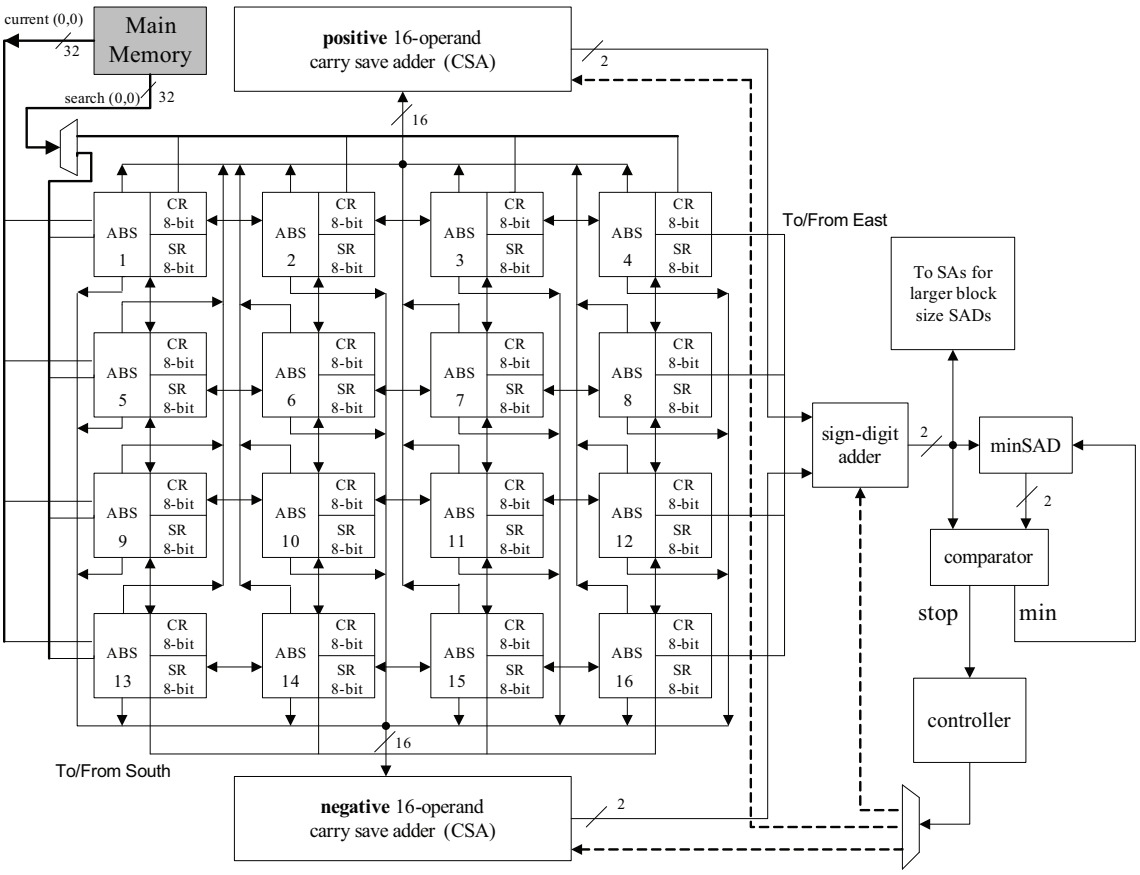


Figure 4.4: Basic Processing Element (BPE).

4.2 Processing Element based on the MSB-First Arithmetic

We propose a 2D architecture formed of two types of processing elements: Basic Processing Element (BPE) to carry out the SAD with MSB-first arithmetic using signed-digit number system for 4×4 block size, and the Select Adders (SA) to compute larger block size SADs. We first explain how BPE is structured to accommodate the MSB-first arithmetic.

4.2.1 Basic Processing Element (BPE)

Figure 4.4 illustrates the structure of the BPE. It is composed of sixteen 8-bit current pixel registers (CR), sixteen 8-bit search pixel registers (SR), sixteen absolute units (ABS), two 16-operand carry save adders (CSA), one signed-digit full adder (SDFA), one comparator, one minSAD unit, and one controller. Besides accessing data from the external memory, each BPE transfers data to/from its neighboring BPEs via the buses "To/From South" and "To/From East". At every cycle, the 16 SRs and 16 CRs receive pixels from the memory or the adjacent BPE, which forms 16 pairs of

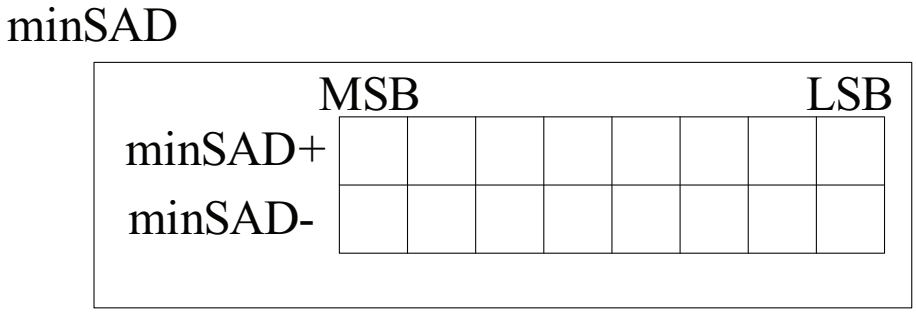


Figure 4.5: minSAD includes two sets of 8-bit registers to store the positive and negative operands of the minimum SAD value.

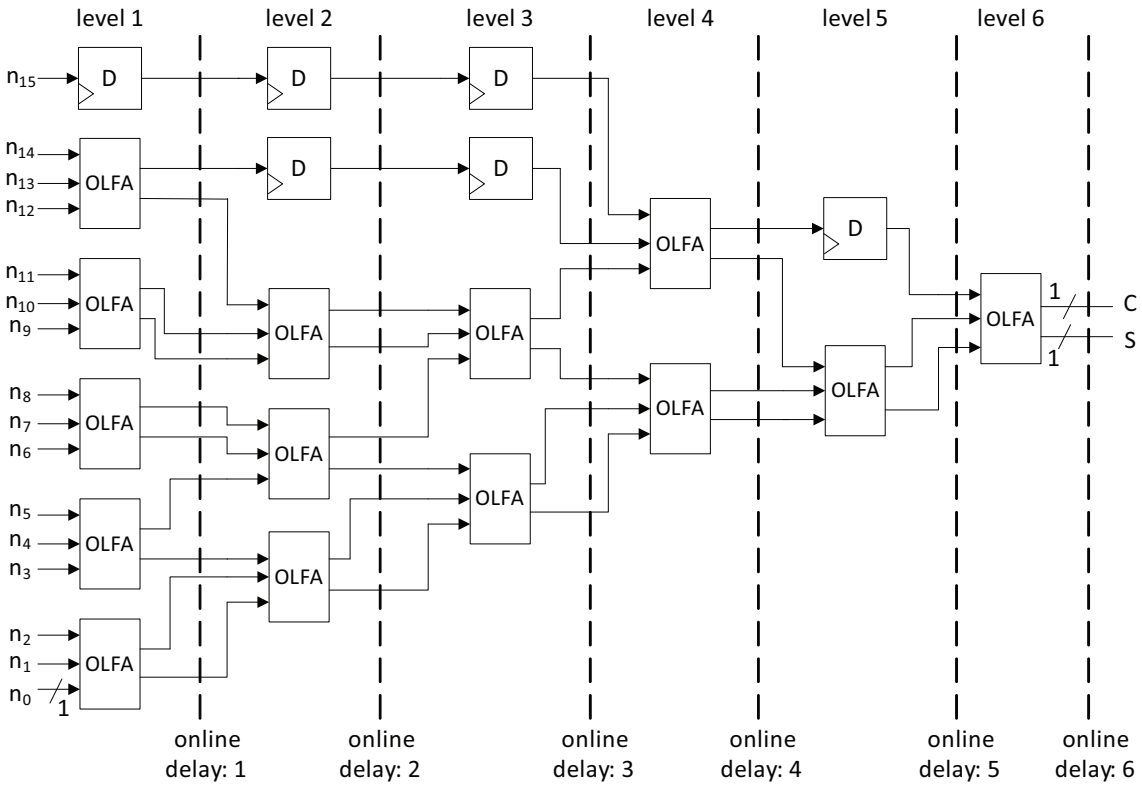


Figure 4.6: 16-operand carry save adder. n_0 to n_{15}, which represent 16 numbers in a two's complement system, are loaded into the carry save adder from MSB to LSB. The result is in C and S format. Since there are six levels of OLFAs concatenated, the total online delay is 6.

SD numbers. Each ABS unit checks the sign of the SD number and swapping is done when necessary to complete the absolute operation. Then all 16 positive components are loaded to the positive 16-operand CSA, while the 16 negative components to the negative 16-operand CSA. The 16-operand CSA can be implemented by 6 levels of OLFAs, which results in 6 online delay cycles as illustrated in Figure 4.6. After loading the last bit of the pixel to the OLFAs at level 1, 0s are padded to compensate the delay for generating the final C and S values as illustrated in Table 4.2. In other words, CSA unit takes a total of 7 cycles. The results of the two 16 operand CSAs (positive and negative) are accumulated with the SDFA. minSAD unit stores the minimum SAD value in a SD representation form as shown in Figure 4.5. In each iteration, from MSB to LSB, corresponding positive and negative bits of the minSAD are compared with the output of SDFA unit for potential early termination decision.

Leong (Li and Leong, 2008) in their bit serial architecture makes use of a signed-digit comparator (Su and Jen, 2000) that has 2 delay cycles in the best case scenario which is observed when the most significant bit is enough to determine the early termination. On the other hand, in our architecture we employ a signed-digit comparator with no online delay (Olivares et al., 2006). In the best case scenario, the MSB (7^{th} bit) of SDFA output is larger, which triggers the stop signal indicating an early termination since there is no need to continue with the computations. The stop signal refreshes the two 16-operand CSAs and the SDFA. Upon this stop signal, all searching pixel registers (SRs) load new candidate blocks and start new SAD com-

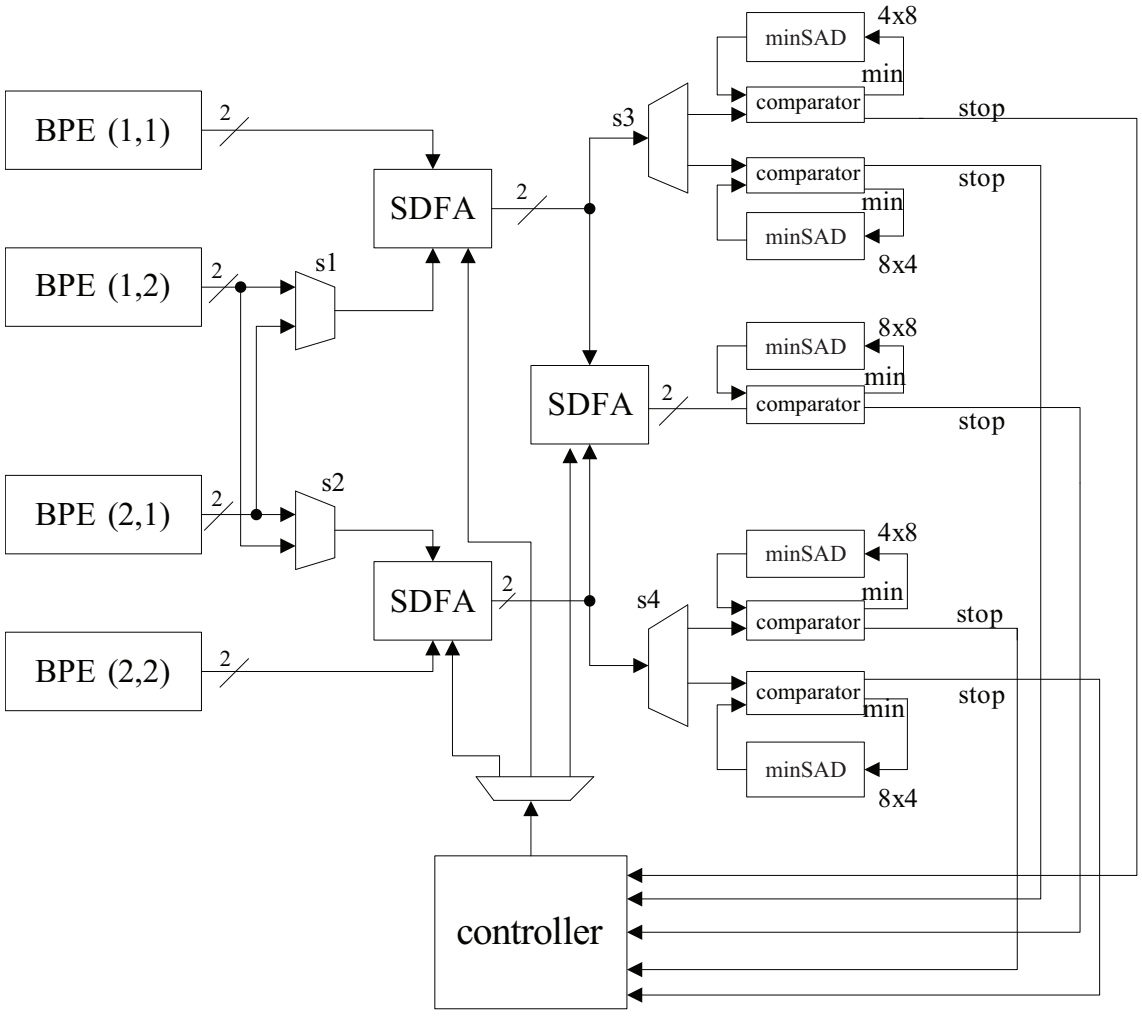


Figure 4.7: Select Adder 1 (SA1) to generate 4×8, 8×4, and 8×8 SADs.

putations. The worst case scenario occurs when the SDFA output is smaller than the minSAD and the decision is made based on the least significant bit (0^{th} bit) which takes 8 more cycles.

In the bit serial architectures, the width of the datapath is only 1-bit, whereas in BSHG, the granularity of a processing element is at a 4×4 level. At first, having MSB-first arithmetic in the BSHG architecture may seem to lead to low resource utilization. However, if the granularity of BPE is reduced to bit level (1×1), then it takes more cycles to load the pixels from the external memory into the registers in the BPE. Since 4×4 is the smallest block size in the ME, this level of granularity is suitable in terms of resource utilization for SAD computations. Our approach is to remain in the hybrid grained level, but transmit data bit-by-bit during the process of computation. We call this the bit serial manner execution for the hybrid grained architecture.

4.2.2 Select Adders (SA) for Variable Block Size

We introduce two types of Select Adders to accommodate variable block size ME. SA1 computes 4×8, 8×4 and 8×8 SADs. This unit (Figure 4.7) consists of one controller, two multiplexers (muxes *s1* and *s2*), two demultiplexers (demuxes *s3* and *s4*), three SDFAs, five comparators and five minSAD units. The SDFAs, comparators and minSADs are the same as in BPE. The controller detects all the stop signals from the comparators and terminates the computations if necessary.

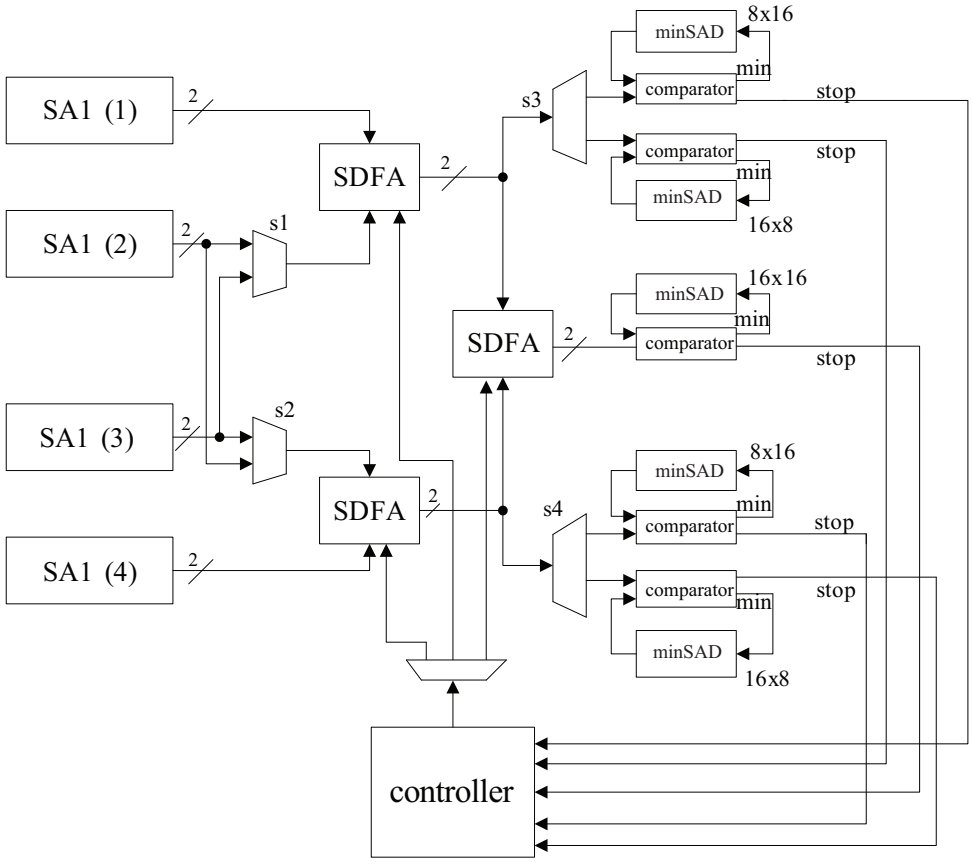


Figure 4.8: Select Adder 2 (SA2) to generate 8×16, 16×8, and 16×16 SADs.

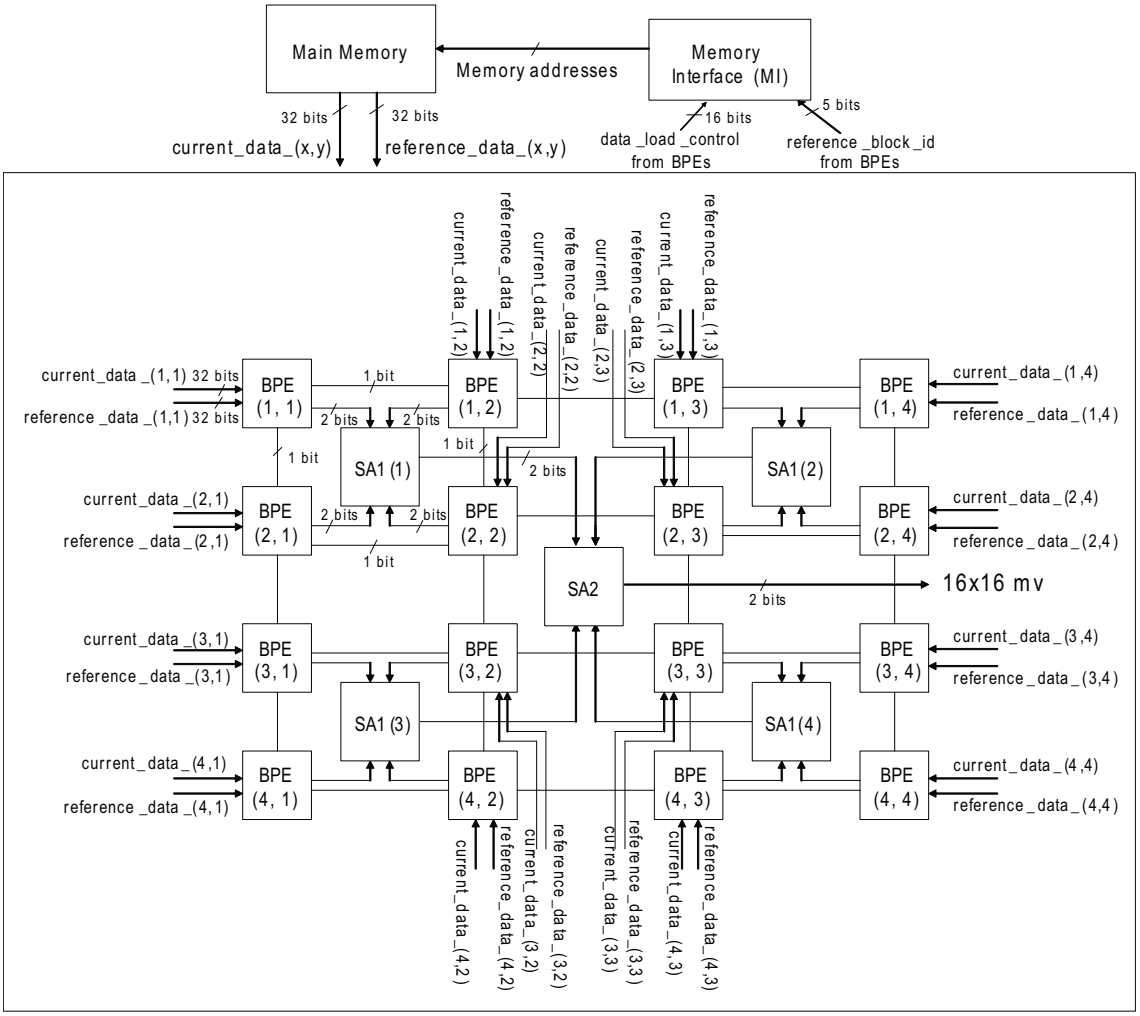


Figure 4.9: Proposed architecture: composed of 16 BPEs, 4 SA1s and 1 SA2. BPEs receive data from main memory through Memory Interface (MI) where MI computes memory address of the reference blocks.

In Figure 4.7, muxes *s1* and *s2* select the computation of either 4×8 or 8×4 SADs. Demuxes *s3* and *s4* direct 4×8 or 8×4 SAD to the corresponding comparators and registers. SA1 computes 8×8 SAD and sends it to SA2 to process larger size SADs (8×16, 16×8 and 16×16). SA2 works in a similar way as SA1 does, as shown in Figure 4.8.

4.2.3 Hybrid Grained Architecture

We form the 2D architecture based on BPEs and SAs with a tuned interconnect to accommodate variable block size ME for the full search algorithm. As shown in Figure 4.9, each node is labeled as "BPE(x, y)" where (x, y) represents grid coordinates from left to right. Each SA1 is represented as "SA1(z)" where "z" identifies the processing element. A "BPE" receives current and reference block from main memory through its dedicated 32-bit data input ports labeled as "*current_data_(x,y)*" and "*reference_data_(x,y)*" where x and y are 1, 2, 3 *or* 4.

"BPE" is responsible for block matching operations (SADs) of a 4×4 sub-block and its search region (Figure 1.1). "BPE" generates a 4×4 SAD which is then fed into "SA1" and "SA2" to generate larger block size SADs. The architecture by default supports full search (data transfer between adjacent BPEs), and accommodates data reuse with zigzag pattern (Figure 4.10).

Memory Interface (MI) BPEs request *reference blocks* from main memory through MI. MI computes memory addresses of those blocks based on the search

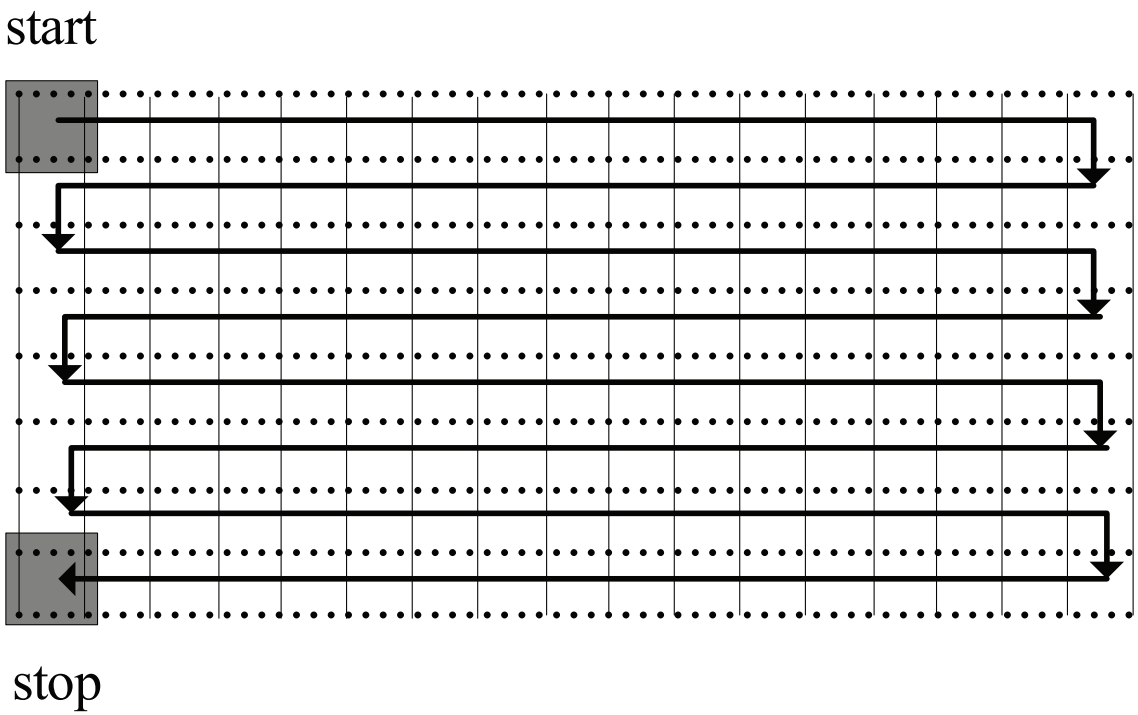


Figure 4.10: Zigzag search pattern.

pattern. It also holds memory address of sixteen current 4×4 blocks that are already in the BPEs. MI receives *data_load_control* (16 bits) and *reference_block_id* (5 bits) signals where *data_load_control* identifies the BPE and *reference_block_id* identifies the requested block. MI then feeds the *Main Memory* with the address of that requested block.

4.2.4 Full Search

In this subsection, we illustrate the functionality of BSHG for the full search algorithm with 16×16 block size. We present how to pipeline the data bit-by-bit on the architecture to support bit serial execution and examine its performance.

Table 4.4: Data scheduling in BPE (1,1). ABS units receive pixels from main memory at cycle 1 to 4. Shift pixels to CSAs from MSB to LSB at cycle 5 to 12. *padding 0s* at cycle 13 to compensate for the online delay. SAD0 and SAD1 are SADs on two neighboring reference blocks.

cycle	SAD0	SAD1
1	CR SR ABS(1 5 9 13)	
2	CR SR ABS(2 6 10 14)	
3	CR SR ABS(3 7 11 15)	
4	CR SR ABS(4 8 12 16)	
5	7th bit	
6	6th bit	7th bit
7	5th bit	6th bit
8	4th bit	5th bit
9	3rd bit	4th bit
10	2nd bit	3rd bit
11	1st bit	2nd bit
12	0th bit	1st bit
13	padding 0s	0th bit
14	best case 4×4	padding 0s
15		best case 4×4
16	best case 4×8 8×4	
17		best case 4×8 8×4
18	best case 8×8	
19		best case 8×8
20	best case 8×16 16×8	
21	worst case 4×4	best case 8×16 16×8
22	best case 16×16	worst case 4×4
23	worst case 4×8 8×4	best case 16×16
24		worst case 4×8 8×4
25	worst case 8×8	
26		worst case 8×8
27	worst case 8×16 16×8	
28		worst case 8×16 16×8
29	worst case 16×16	
30		worst case 16×16

Table 4.4 illustrates the data scheduling within the BPE(1,1) of Figure 4.9. At cycle 1, the main memory fills the current register (CR) and search register (SR) of the ABS units 1, 5, 9 and 13 (shown in Figure 4.4) with 8-bit pixels. These registers then shift the pixels to the registers of the neighbor ABS units (from left to right) 2, 6, 10 and 14 in the next cycle. While this shift is occurring, the memory loads another set of pixels into registers of ABS units 1, 5, 9 and 13. It takes four cycles until all the 16 CRs and 16 SRs of a BPE receive their pixels. At cycle 5, ABS(1) shifts the MSB of registers CR and SR to ABS(2), the MSB of the larger register to "positive CSA" and finally the MSB of the smaller register to "negative CSA". In fact, all ABS units carry out these shifting operations at the same time. For the ABS units that are on the rightmost column, MSB bits of CR and SR are shifted to the neighbor BPE (as in the case of from ABS(4) of BPE(1,1) to ABS(1) of BPE(1,2)). Since ABS unit decides which register is larger (CR or SR) without any delay, CSAs start operating on these MSBs in cycle 5 too. Also, in cycle 5 since MSBs are already loaded into the CSAs, the MSBs of CR and SR in ABS units 1, 5, 9 and 13 are overwritten with the new set of pixel values arriving from the memory. All these shifting operations are carried out from MSB to LSB in a pipelined way.

It takes 6 cycles for CSA (Figure 4.6) plus 2 cycles for SDFA (Figure 4.2c) plus the required padding cycle (Table 4.2) to generate the SD pair for the MSB of a 4×4 SAD (cycle 14 in Table 4.4). SA1 (Figure 4.7) and SA2 (Figure 4.8) each have 4 cycles of online delay (2 SDFAs). Therefore, the SD pair for the MSB passes

through SA1 at the end of cycle 18 and SA2 at the end of cycle 22. Since the design is pipelined, after cycle 22, each cycle generates the next set of SD bits of the 16×16 SAD. The best case scenario occurs when the MSB of the new SAD is larger than the MSB of the stored SAD, after which the operation is terminated (cycle 23 for 16×16). The worst case scenario occurs if the minimum SAD decision is made based on the least significant bit, which occurs in cycle 29. Best and worst case completion times for all block sizes are indicated in Table 4.4.

While loading bits to the adders, registers inside the BPE transmit the data to their neighboring registers, also bit-by-bit. Since the search window for the full search follows the zigzag pattern, it guarantees that the memory needs to load only one new row or new column into the system in each cycle. All the other data that is already stored locally in the registers gets shifted to the neighboring BPE following the zigzag pattern. For instance, at the beginning of cycle 5, after feeding the $7th$ bits of pixels to the adder trees, registers 1, 5, 9 and 13 shift their $7th$ bits to the registers 2, 6, 10 and 14, and the rest of the registers shift their bits in a similar manner. In the next cycle all registers get updated and start another iteration of SAD computations. This way we achieve data reuse among the processing elements, enable parallel execution and pipelining to speed up the SAD computations.

We make use of the pipelining scheme to feed pixel data bit-by-bit into each processing element. In other words, bit serial implementation is a way to utilize pipelining in the BSHG architecture. We call this bit-by-bit pipelining.

4.3 Performance analysis

In this section, we compare the performance of BSHG against the previously reported bit serial (Li and Leong, 2008) and bit parallel (Kim et al., 2005; Yap and McCanny, 2004; Ou et al., 2005; Yap and McCanny, 2003) VBSME designs on the full search algorithm. Similar to these studies, our main focus is the design of the processing elements and the interconnect. Therefore, our performance analysis assumes that the data is already on chip similar to the approach taken by these studies.

The proposed architecture was designed with Verilog-HDL description and synthesized by using a 90 nm CMOS standard cell library. Table 4.5 presents the performance of the BSHG with respect to its counterparts in terms of number of PEs, throughput (macroblocks per second), gate counts and gate performance (throughput per gate). We use CIF format as it is commonly used by other studies for performance comparison (Li and Leong, 2008; Yap and McCanny, 2004; Ou et al., 2005; Kim et al., 2005), and (Yap and McCanny, 2003). A decision for early termination during SAD calculations depends on the type of the data and the nature of the video input. The throughput of the proposed architecture will vary based on the actual input. Therefore, we present the best case and worst case results in Table 4.5 and Table 4.6 respectively.

As shown in Table 4.5, resource usage (technology independent gate count) of BSHG is about one-seventh of its counterpart (Ou et al., 2005). Given CIF format

Table 4.5: Comparisons of BSHG with other designs in number of processing elements (PEs), gate counts, throughput and performance per gate. Best performance in bold. Search pattern: full search. Block size: 16×16. Search range: 32×32. Video format: CIF (352×288).

Architectures		Number of PEs	Area (gates)	Throughput (macroblocks/sec)	Performance/gate (throughput/gate)
Others	BP1	256	597k	195k	0.327
	BP2	256	154k	97k	0.634
	BP3	**16**	108k	5k	0.05
	BP4	**16**	61k	17k	0.292
	BS only	N/A	**55k**	23k	0.417
BSHG	best	21	84k	**342k**	**4.0**
	worst	21	84k	337k	3.94
BSHG without early termination		21	82k	181k	2.207

* Note: For Tables Table 4.5 and Table 4.6, "BP1, BP2, BP3, BP4" represent bit parallel architectures in (Ou et al., 2005), (Kim et al., 2005), (Yap and McCanny, 2003), and (Yap and McCanny, 2004) respectively. "BS only" means bit serial architecture in (Li and Leong, 2008). "BSHG" represents our proposed architecture.

Table 4.6: The required clock rate of BSHG and its counterparts to sustain various video formats for different frame sizes and frame rates (full search, with 32×32 search range and 16×16 block size).

Frame size		QCIF (176×144)	CIF (352×288)	4CIF (704×576)	16CIF (1408×1152)	SDTV (1280×720)	HDTV (1280×720)	SHDTV (1920×1080)
Frame rate(fps)		30	30	30	30	30	60	60
Others (MHz)	BP1	3.04	12.1	48.65	194.6	110.6	221.2	**497**
	BP2	3.05	12.2	48.7	194.8	110.7	221.4	498
	BP3	53.5	214	854	3417	1942	3884	8739
	BP4	49	196	784	3136	1782	3563	8017
	BS only	54.8	219	876	3504	1991	3981	8957
BSHG (MHz)	best	3.61	14.4	57.8	231.1	131.8	263.5	**592.1**
	worst	3.66	14.6	58.7	234	133.7	267.3	**600.6**

video sequence, and search range of 32×32, a 16×16 macroblock is compared with 1024 reference blocks. (Ou et al., 2005) follows partial sum SAD approach and takes 4 cycles for producing the first 4×4 SAD and rest of the computations are pipelined. Thus, (Ou et al., 2005) requires 1024 cycles to produce a 16×16 motion vector for full search. For (Kim et al., 2005; Yap and McCanny, 2003, 2004), the number of reported required cycles are 1025, 16784 and 16384 respectively. By taking advantage of the early termination, the best case to get minimum SAD for 16×16 size occurs at cycle 22, while the worst case occurs at cycle 29. Rest of the computations are pipelined. Thus in summary, to produce a 16×16 motion vector, we need 1045 cycles for best case and 1052 cycles for worst.

For a bit serial architecture, early termination helps reduce the number of SAD computations and has a significant impact on the throughput. However, for a bit parallel architecture, performance improvement mainly relies on data reuse and pipelined datapath design. The proposed design takes advantage of the two approaches. In the following paragraphs, we analyze the performance benefit from early termination and pipelining schemes using Table 4.5.

Throughput performance of a bit parallel architecture relies on replicating processing elements in a 2D structure. As the amount of resources increase, the throughput of the system improves, however that does not necessarily mean that performance per gate will improve. From that perspective, we mainly use performance/gate metric as comparison criteria. For example in (Ou et al., 2005), the bit

parallel architecture heavily utilizes pipelining and achieves the best performance per gate among the bit parallel architectures. Compared to this architecture, as shown in Table 4.5, we achieve 6.3x improvement. In order to evaluate the factors contributing to this performance improvement, we disable the early termination feature in our design, and show the results in Table 4.5 labeled as "BSHG without early termination". Without early termination, the performance per gate is 2.207, which is a 3.46x improvement over (Ou et al., 2005). The following design choices contribute to this improvement.

- The bit serial hierarchical hybrid grained (BSHG) 2D architecture effectively utilizes the computational resources, and avoids redundancy resulting with a small footprint. Table 4.5 shows that BP1 (Ou et al., 2005) and BP2 (Kim et al., 2005) utilize 256 PEs (each PE computes the absolute difference of two pixels), BP3 (Yap and McCanny, 2003) and BP4 (Yap and McCanny, 2004) utilize 16 PEs of type 4×4 block size, and BSHG employs 21 hybrid grained processing elements out of which 16 of them are 4×4, 4 of them are 8×8 and 1 of them is at 16×16 granularity. Kim (Kim et al., 2005) employs an adder tree structure to compute larger SADs as shown in Figure 3.2. In our architecture, we introduce a hierarchy of processing elements and reduce the complexity of the adder tree structure. For example, in Kim's (Kim et al., 2005) adder tree, there are 8 adders for 4×8, and 8 adders for 8×4, while in our architecture we have 2 adders per SA1, and a total of 8 adders to derive each

4×8 and 8×4 block. Similarly, Kim (Kim et al., 2005) employs 2 adders for each 8×16 and 16×8 block, whereas in our architecture we have only 2 adders in SA2 to derive both 8×16 and 16×8 blocks. We use the same amount of adders as SA2 to derive 8×8 and 16×16 blocks.

- We employ muxes/demuxes to choose the appropriate 4×4 block for developing larger block size SADs. These muxes/demuxes do not exist in other bit parallel architectures, meaning that the architecture calculates all 4×4 blocks and then proceeds with the calculation of larger block sizes.

- We utilize shifting registers for data pipelining. In bit parallel architectures, such as BP3 (Yap and McCanny, 2003) and BP4 (Yap and McCanny, 2004), these registers do not exist.

As shown in Table 4.5, bit serial architecture BS (Li and Leong, 2008) is much more area efficient than bit parallel architectures. Compared to BS (Li and Leong, 2008), we achieve 9.6x improvement in performance per gate with early termination. Without early termination, even through the footprint of our architecture is larger due to 2D PE array architecture, and pixel registers for the shifting and pipelining, we observe 5.29x improvement.

In conclusion, the early termination capability further improves the performance per gate for the BSHG architecture by a factor of 1.82x. The overall performance improvement with respect to bit serial and bit parallel architectures is dependent

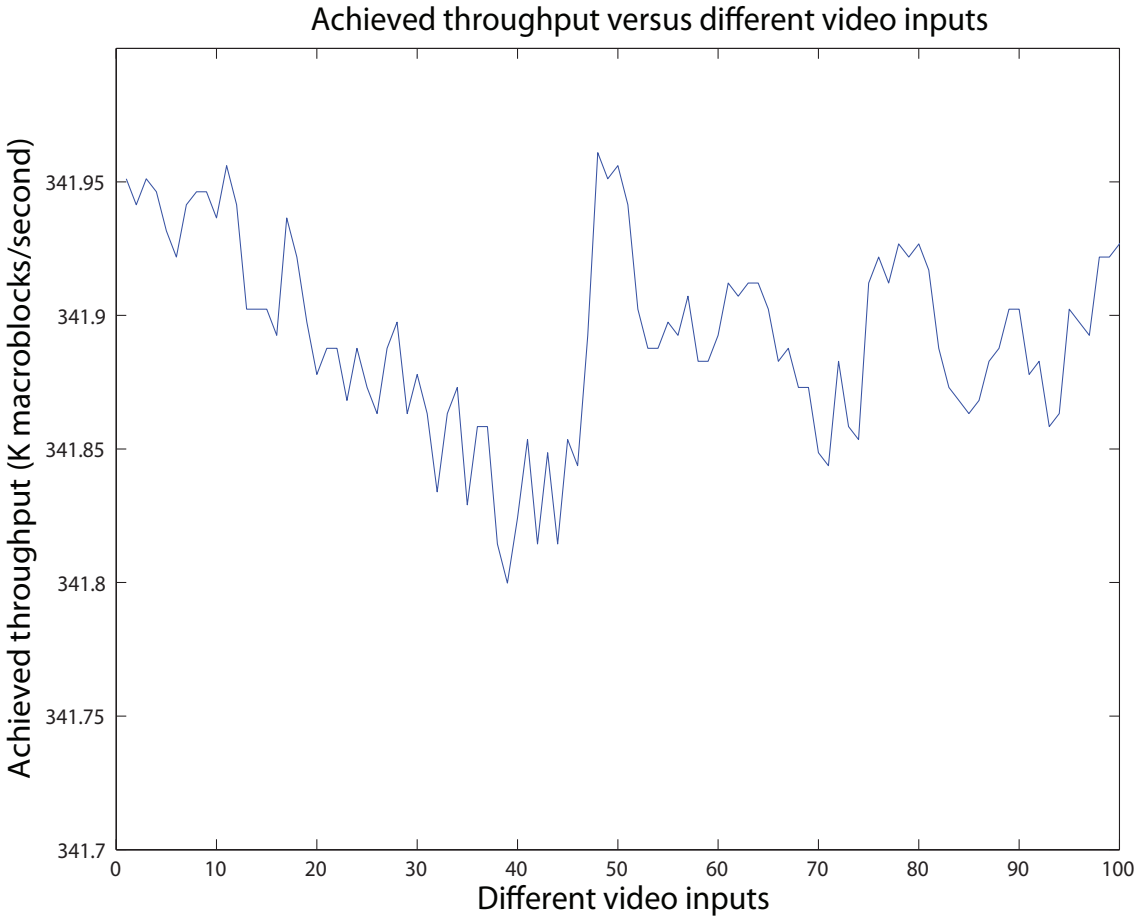


Figure 4.11: Achieved throughput (in terms of K macroblocks/second) versus different video inputs (100 frames in CIF format), full search, search range [-16, +15].

on the hierarchical PE array architecture along with the employed pipelining and data reuse mechanisms.

Table 4.6 shows the minimum clock rate that is required to sustain 30 fps and 60 fps frame rates by the proposed architecture and its counterparts. As shown in Table 4.6, for QCIF format, the ASIC approach (Ou et al., 2005) needs to operate at 3.04 MHz to sustain the 30 fps requirement, whereas BSHG needs a faster clock, because the number of cycles required to compute 16×16 motion vector is larger in BSHG (best case 1045 cycles) than in (Ou et al., 2005) (1024 cycles). For clock rates slower than 3.61 MHz, BSHG won't be able to sustain 30 fps. As shown in Table 4.6, for all frame sizes, our clock rate is comparable with (Ou et al., 2005). As expected, while the frame size gets larger, the clock rate needs to increase to respond to more computations with the 30 fps or 60 fps constraint.

In Table 4.6, we also observe that all architectures described in this section are capable of working at a practical frequency when encoding the QCIF and CIF video in real time. However, to sustain the requirements of a higher resolution video types, for example, SHDTV (1920×1080 at 60 fps), the bit serial only architecture (Li and Leong, 2008) needs to operate at 8.95 GHz to execute the motion estimation algorithm in real time, whereas ASIC designs (Yap and McCanny, 2004) and (Ou et al., 2005) require only about 500 MHz, which makes the bit parallel architecture a more practical solution. Furthermore, BSHG operates at a close frequency (600 MHz) to its ASIC-based counterparts.

As mentioned above, performance of the proposed architecture, due to its early termination capability, depends on the video input as well. For that purpose, we also present a histogram (Figure 4.11) to give a better indication about the achieved throughput versus different video inputs. We randomly choose 100 frames of different CIF video sequences, within the search range [-16, +15], to evaluate the throughput for each of them. Figure 4.11 verifies that all the throughputs fall in the range 337k∼342k, as shown in Table 4.5.

4.4 Summary

In this chapter, we introduce an application-specific hybrid grained processing element based architecture to support the variable block size motion estimation of H.264/AVC. The simplicity and highly parallel nature of the architecture with an excellent degree of data reuse makes our design suitable for run time adaptation for block size variations to respond to the video quality and/or timing constraints. Our 2D architecture with MSB-first arithmetic supports large reuse of search data between processing elements and thus reduces the memory transaction requirement, which is missing in existing state of the art approaches. Results show that for full search, BSHG outperforms other existing designs in terms of throughput and gate performance.

In Chapter 5, we design a variable block size motion estimation architecture that is adaptive to the full search (FS) and the three-step search (3SS) algorithms.

CHAPTER 5

ADAPTIVE MOTION ESTIMATION ARCHITECTURE

The BSHG architecture is limited with supporting only the full search method. Our objective in this chapter is to design an architecture that offers the end user with the flexibility of choosing between the high quality video service during power-rich state (FS mode), and extended video service (fast search mode). As opposed to the BSHG design methodology, in this chapter we start with the high level architecture diagram and zoom into individual components and discuss the decisions we made throughout the design process for the AMEA.

5.1 High Level Hardware Architecture

We illustrate the high level adaptive motion estimation architecture (AMEA) in Figure 5.1 with the on/off chip memories, the processing element (PE) array structure, and the address generator logic. We pay specific attention to reducing the amount of off-chip memory accesses through pixel-level data reuse, dictated by the search pattern (FS and 3SS) while computing the SADs for 4×4 block size, and SAD level data reuse among the PEs while computing the SADs for larger block sizes. We reduce the number of operations during SAD calculations by enforcing bit-serial execution throughout the pipelined datapath, which enables early termination while

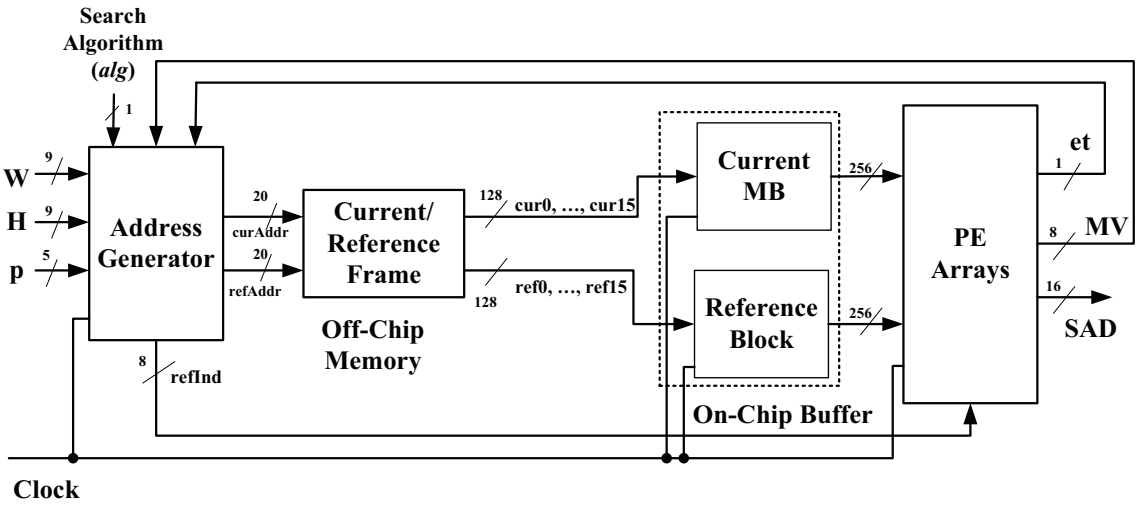


Figure 5.1: A high level block diagram of the ME architecture, where, W: width of the frame, H: height of the frame, p: search range (16), $refInd$: x-y coordinates of the reference block, alg: FS or 3SS, $curAddr$: address of current block, $refAddr$: address of reference block, $cur(i)$: 16 pixels of 8 bit current block, $ref(i)$: 16 pixels of 8 bit current block, et: early termination, MV: motion vector, SAD: sum of absolute difference.

determining the minimum SAD.

5.2 Pixel Level Data Reuse

The fast search algorithms primarily utilize the decimation of checking points to reduce the computation. In the first step of the 3SS, the block with the minimum SAD is identified after evaluating nine sparsely located candidates as labeled by "1" in Figure 1.3(b). In the second step, the search center is moved to the selected point of the previous step and the step size is reduced by half (refer to locations marked by "2" in Figure 1.3b). The search progresses around the block with the minimum SAD during the third step by evaluating another set of 8 blocks. We choose the three-

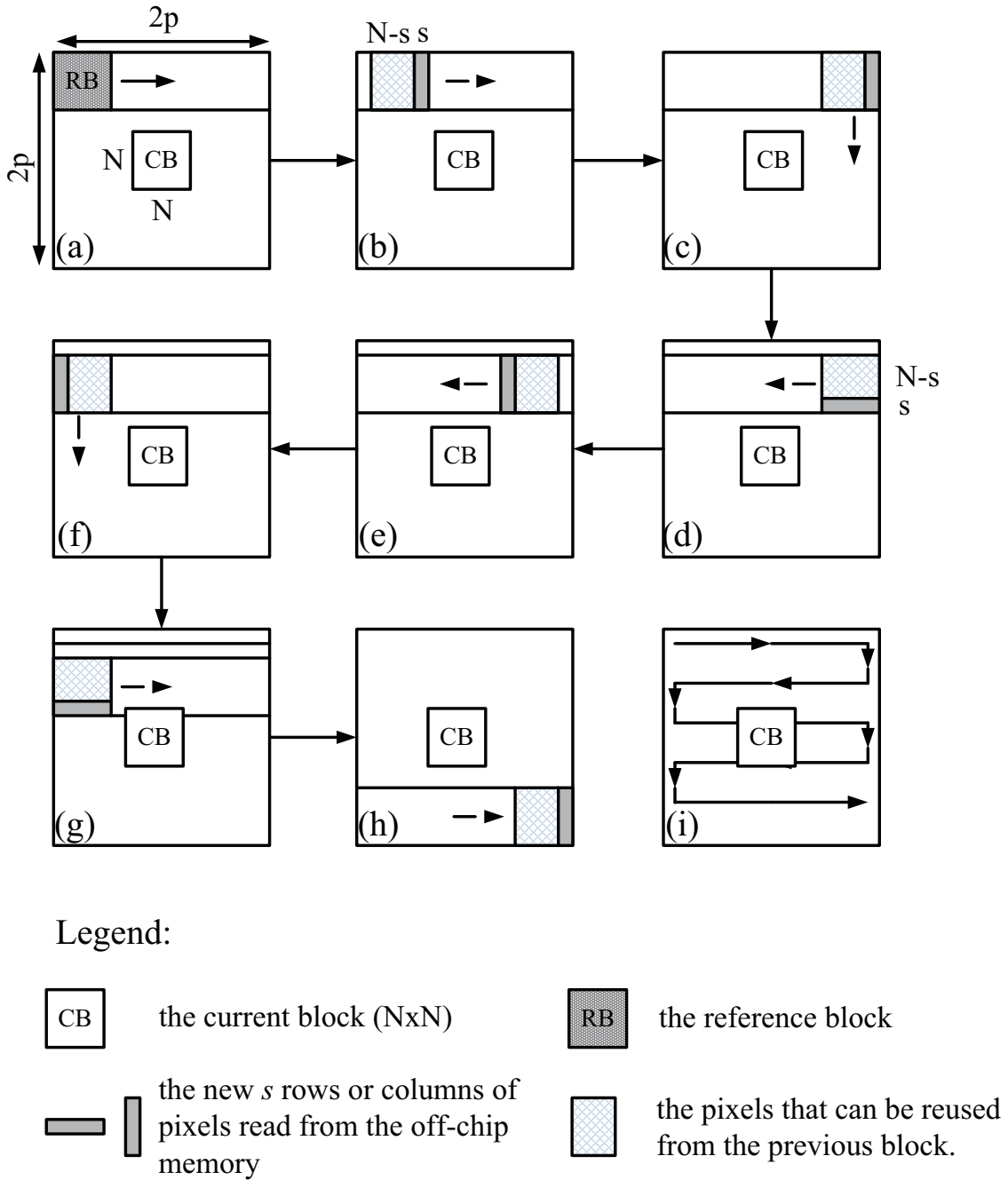


Figure 5.2: Zigzag scan pattern for the data reuse in both the full search and the three-step search algorithms. s indicates how many new rows or new columns of pixels are loaded from the off-chip memory. For the full search, $s = 1$. For the 3SS, $s = 4$ in step 1; $s = 2$ in step 2; and $s = 1$ in step 3.

step search for the AMEA based on the following reasons. The 3SS involves fixed-step operations, which makes it easier to synchronize the ME module with other components of the encoder. If the search step is unrestricted, such as the diamond search, the motion estimator may need a large number of iterations (consequently, a large number of clock cycles) to generate the motion vector. As the entire encoder must also be able to process the ME for the worst case scenario, all modules must be overestimated to achieve the desired goal, resulting in under-utilization of hardware resources. The 3SS has a "relatively" regular data access flow pattern. In 3SS, the memory accesses are carried out horizontally or vertically, whereas in the diamond or the hexagon search, the search points are arranged in a diagonal shape. This regularity feature presents better opportunity for exploiting the data-reuse. Finally, the 3SS has been tested as a robust method, and recommended by the CCITT RM8 (CCITT, June, 1989) and MPEG SM3 (MPEG-1, 1993) since the release of the H.261 standard.

We illustrate the pixel level data reuse while searching for a match for the current MB in the search region with Figure 5.2 for both the FS and the 3SS. First, the reference block (RB) is read into the on-chip buffer from the memory (Figure 5.2a). After the SAD calculation with the current macroblock (CB) and the RB, for the FS method, the RB is shifted by one pixel to right, as illustrated in Figure 5.2(b) where s is 1. Therefore, only the new one column of pixel data is read into the on-chip buffer. When the reference block reaches the end of the frame, it is shifted down

by one-pixel as shown in Figure 5.2(c) and (d). For the next SAD calculation, only the new one row of pixel data is read into the on-chip buffer. Same operations are then carried out by following the zigzag pattern till the end of the frame is reached (Figure 5.2h). The 3SS method skips "s" number of columns or rows when choosing the sparsely located reference blocks. In the first step, the row or column intervals between the successive candidate blocks are 4 ($s = 4$). Therefore, the $N - 4$ rows or columns of the previous reference block can be reused. In the second and the third steps, s becomes 2 and 1, indicating that $N - 2$ and $N - 1$ rows or columns are overlapped with the previous reference blocks, respectively.

5.3 Address Generator (AG) and On-Chip Buffer

The AMEA offers the flexibility to switch between the FS and the 3SS by changing the addresses for accessing the current and reference blocks based on the search pattern. The inputs of the AG are:

- width (W) and the height (H) of the frame;
- 1-bit search algorithm signal alg indicates FS or 3SS;
- 1-bit early termination signal et from the PE arrays. If et is asserted, the AG outputs the address of the next reference block;
- previous MV feedback from the PE arrays ($MV.x, MV.y$, 8-bit), which are used to determine the next reference block in the 3SS;

- search range p.

The outputs of the AG are:

- physical address of the current MB ($curAddr$, 20-bit);
- physical address of the reference block ($refAddr$, 20-bit);
- coordinates of the reference block ($refInd(x, y)$, 8-bit). The $refInd$ is populated to the PE arrays to determine the output MV which is associated with the minimum SAD.

Inside the AG, we also instantiate

- a register for storing the current block coordinates ($curInd$);
- a self-increment step counter (SC) recording the number of points that are checked. For the FS, $sc_{max} = 2p \times 2p$. For the 3SS, $sc_{max} = 25$.

For the FS ($alg = 0$), at the initial stage, the first current block is the top left MB ($curInd.x = 0, curInd.y = 0$)and Equation 5.1 is used to generate $curAddr$. We calculate the $refInd$ and $refAddr$ using Equation 5.2. The off-chip memory receives the $curAddr$ and $refAddr$ and loads the corresponding pixels to the on-chip buffer. Since we follow the zigzag scan pattern, we only need to read one new column or one new row into the on-chip buffer. Equation 5.3 is used to update the $refInd$. If at any cycle, the early termination (et) is asserted, then the $refInd$ is updated with the address of the next column or row. The step counter (sc) increments every

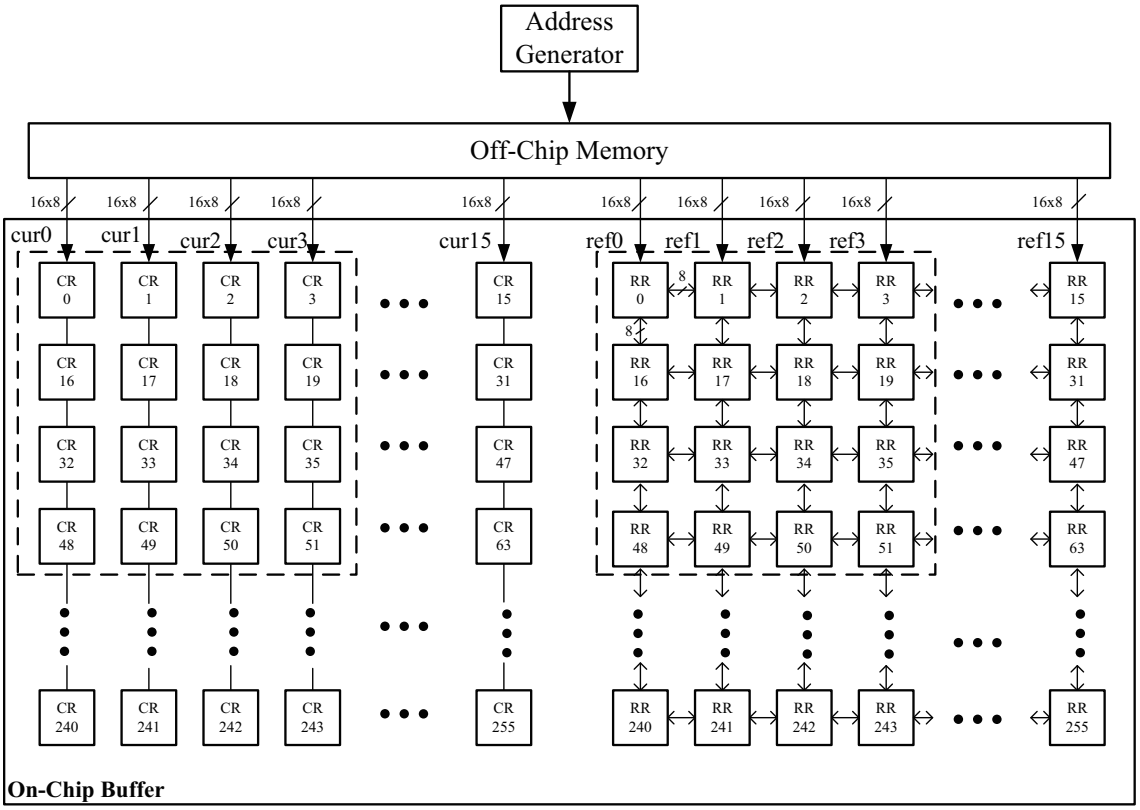
cycle until the $2p \times 2p$ is reached, which indicates the matching operations for the current MB are completed. The $curAddr$ is then updated using Equation 5.4 to move to the next current MB.

For the 3SS ($alg = 1$), we use Equations 5.1 and 5.5 to compute $curAddr$, $refInd$, and $refAddr$ at the initial stage. In step 1, after finding the best MV among the nine points, the AG receives the MV from the PE arrays and updates the $refInd$ using Equation 5.6. The AG then generates the new starting address of the reference block for step 2 with Equation 5.2. Step 3 updates the $refInd$ with Equation 5.7. In the 3SS mode, the sc increments every cycle until 25. Then the $curAddr$ is updated according to Equation 5.4.

$$curAddr = curInd.x * W + curInd.y \tag{5.1}$$

$$\begin{cases} (refInd.x, refInd.y) = (curInd.x - p, curInd.y - p); \\ refAddr = refInd.x * W + refInd.y \end{cases} \tag{5.2}$$

$$(refInd.x, refInd.y) = \begin{cases} (refInd.x + 1, refInd.y) & \text{if moving from left to right} \\ (refInd.x, refInd.y + W) & \text{if on the boundary} \\ (refInd.x - 1, refInd.y) & \text{if moving from right to left} \end{cases} \tag{5.3}$$



(a) Register array for the current pixels (256x8-bit) (b) Register array for the reference pixels (256x8-bit)

Figure 5.3: On-chip search range buffer. (a)Current pixel registers; (b)Reference pixel registers. In our design, we assume the pixels are loaded in a column wise manner. The 128-bit pixels are populated from top to bottom.

$$(curInd.x, curInd.y) = \begin{cases} (curInd.x + 16, curInd.y) & \text{if not on the boundary} \\ (0, curInd.y + 16) & \text{if on the boundary} \end{cases} \tag{5.4}$$

$$\begin{cases} (refInd.x, refInd.y) = (curInd.x - 4, curInd.y - 4); \\ refAddr = refInd.x * W + refInd.y \end{cases} \tag{5.5}$$

$$(refInd.x, refInd.y) = (MV.x - 2, MV.y - 2) \tag{5.6}$$

$$(refInd.x, refInd.y) = (MV.x - 1, MV.y - 1) \tag{5.7}$$

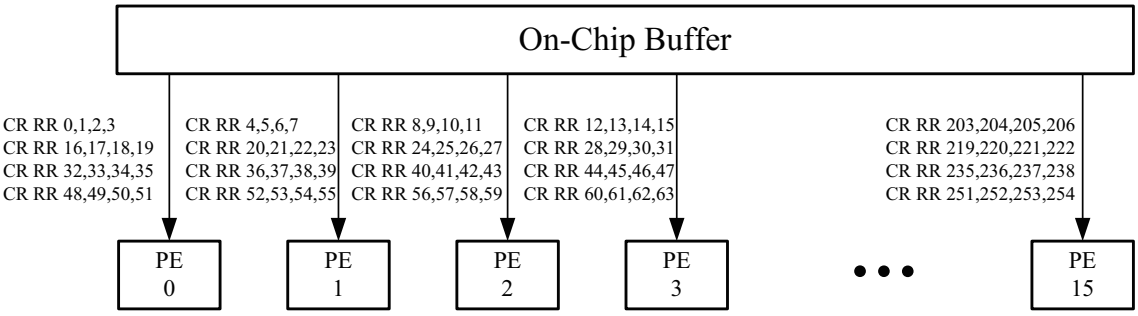
We illustrate the structure of the on-chip buffer for the current block and the reference block in Figure 5.3. The 256 8-bit registers (CR) store the current pixels and the 256 8-bit registers (RR) store the search pixels, where "i" corresponds to the position inside the macroblock. Besides accessing data from the off-chip memory, each RR also shifts the pixel to/from its neighboring register for data reuse between iterations. The off-chip memory receives the addresses of the current MB and the search block from the AG. The pixels are then read into the on-chip buffer at 128 bits (16×8) per cycle, shown as cur*i* and ref*i* in Figure 5.3, starting with the first row of CRs and RRs, and propagating towards the last row in each cycle.

5.4 4×4 PE

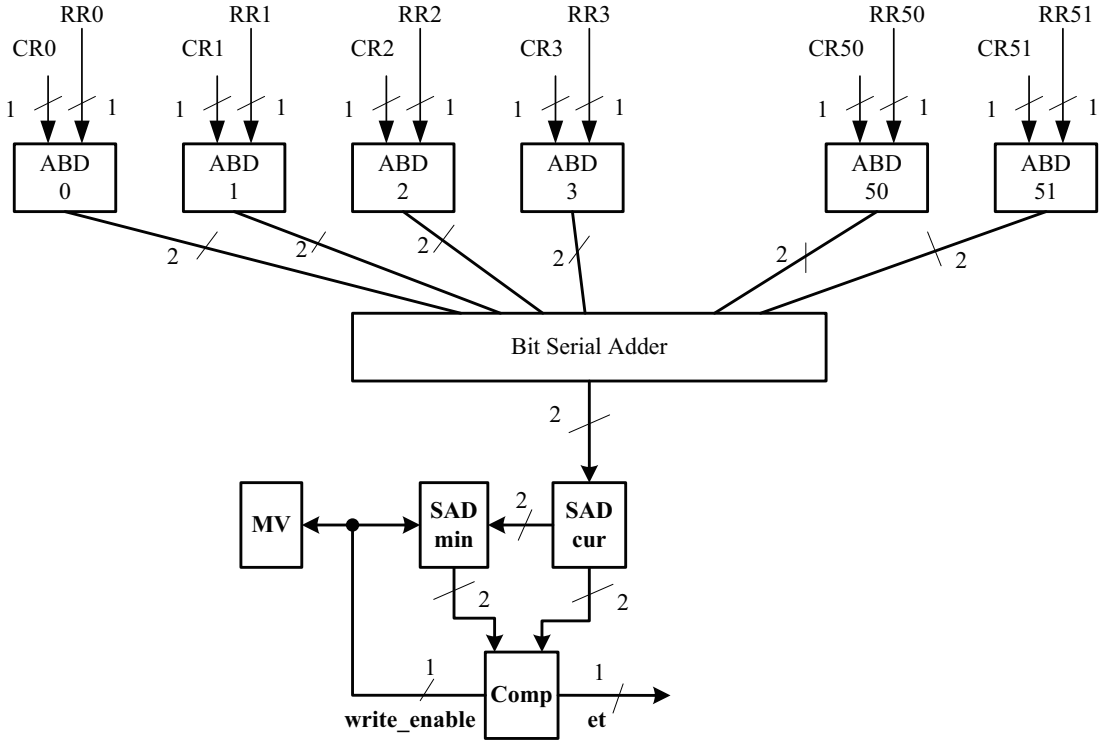
We rewrite the SAD equation to support bit-by-bit execution as shown in Equation 5.8, where k stands for the kth bit in the pixel, $pos(m,n,k)$ and $neg(m,n,k)$ represent the positive and negative operands after eliminating the absolute operation (Li and Leong, 2008).

$$\begin{aligned}
SAD(m,n) &= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}|C(m,n)-R(m+v_x,n+v_y)| \\
&= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|\sum_{k=0}^{k=7}2^k\times C(m,n,k)-\sum_{k=0}^{k=7}2^k\times R(m+v_x,n+v_y,k)\right| \\
&= \sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|\sum_{k=0}^{k=7}2^k\times (C(m,n,k)-R(m+v_x,n+v_y,k))\right| \\
&= \sum_{k=7}^{k=0}2^k\times(\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}(pos(m,n,k)-neg(m+v_x,n+v_y,k))) \\
&= \sum_{k=7}^{k=0}2^k\times(\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}pos(m,n,k)-\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}neg(m+v_x,n+v_y,k))
\end{aligned} \tag{5.8}$$

The AMEA employs 16 PEs to operate on 4×4 blocks. We illustrate the interaction between the PEs and the on-chip buffers in Figure 5.4(a). Each PE is composed of sixteen absolute difference units (ABD), one bit serial adder, one comparator, one register for storing the minimum SAD (SAD_{min}), and one register for storing the motion vector (MV). At the beginning of each clock cycle, ABDs receive 16 pairs of bits from the CR and RR starting with the MSB. Each ABD checks the sign of the bit pair and swaps them when necessary to complete the absolute operation, which forms 2 groups of positive and negative operands (as shown in Equation 5.8).



(a) On-chip buffer and the 16 PEs



(b) Structure of the PE 0

Figure 5.4: (a) 16 PEs receive the current and the reference pixels from the on-chip buffer. (b) Structure of the Processing Element for the 4×4 block size.

Then all 16 positive and 16 negative operands are loaded to the bit serial adder to generate the corresponding bit of the $SAD_{current}$. This value is then compared with the corresponding bit of the SAD_{min} for a potential early termination decision. Each PE receives its share of bits from the RR and CR buffers as illustrated in Figure 5.4(b). We note that, the bit serial adder is implemented by 6 levels of OLFAs (Figure 4.2b), which results with 6 delay cycles (Olivares et al., 2006).

We also use a radix-2 signed-digit (SD) system with digit set {-1,0,1} in our design (Avizienis, 1961). Therefore, 2-bits are being read from the SAD_{min}, ABDs and the adder. We employ a bit serial comparator with no delay as suggested in (Olivares et al., 2006). In the best case scenario, the MSB (11^{th} bit) of the bit serial adder output is larger, which asserts the early termination (et) to stop rest of the computations. The AG receives the et signal and starts loading the next reference block for new SAD computations. The worst case scenario occurs when the bit serial adder output is smaller than the SAD_{min} and the decision is reached based on the least significant bit (0^{th} bit).

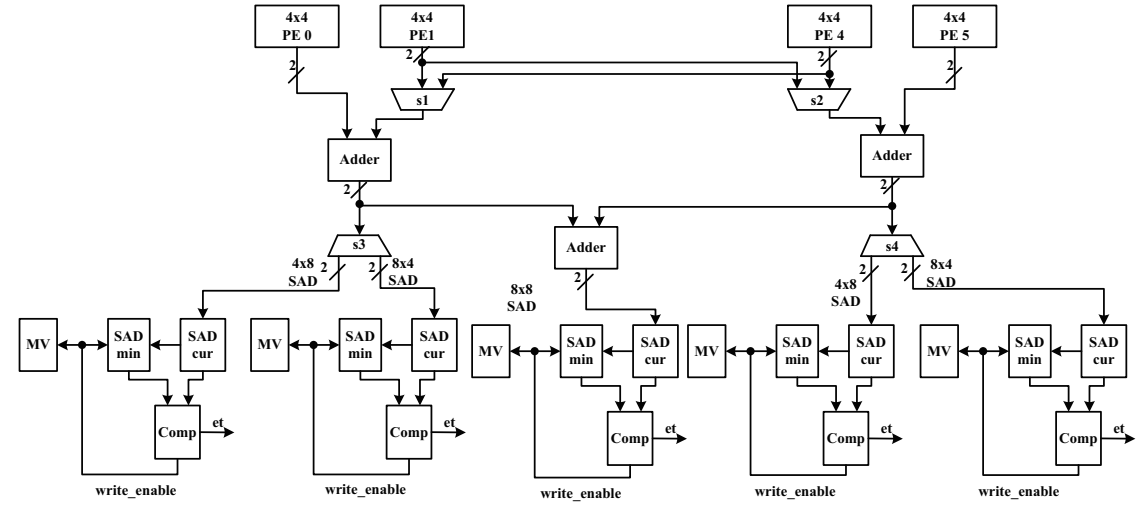
Each PE also includes a register to store the MV ($refInd$) associated with the minimum SAD. The $refInd$ is updated simultaneously with the SAD_{min} and feeds back to the AG for generating the new reference block in the 3SS mode.

5.5 Adder Tree

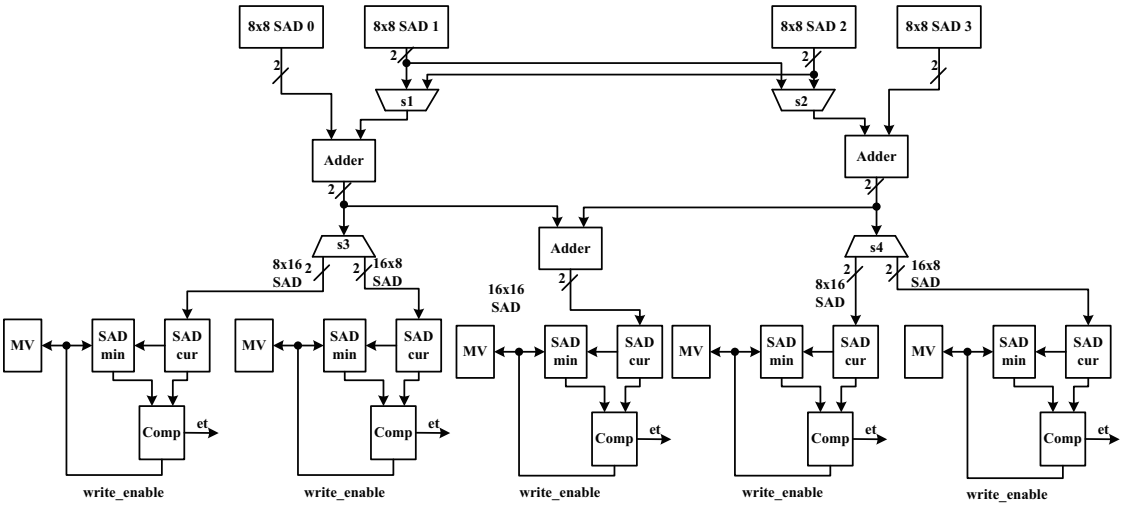
We introduce two types of adders to accommodate variable block size ME. Type 1 adder (Figure 5.5a) computes 4×8, 8×4 and 8×8 SADs. Type 1 adder consists of two multiplexers (muxes *s1* and *s2*), two demultiplexers (demuxes *s3* and *s4*), three bit serial adders, five comparators, five SAD_{min} registers, and five MV registers. The bit serial adders, comparators, SAD_{min}, and MV are the same as in the 4×4 PE.

In Figure 5.5(a), muxes *s1* and *s2* select the computation of either 4×8 or 8×4 SADs. Demuxes *s3* and *s4* direct 4×8 or 8×4 SAD to the corresponding comparators and registers. Type 1 adder computes 8×8 SAD and sends it to Type 2 adder to derive larger size SADs (8×16, 16×8 and 16×16). Type 2 adder works in a similar way as Type 1 adder does, as shown in Figure 5.5(b).

Type 1 and Type 2 adders effectively utilize the computational resources, and avoid redundancy resulting with a small footprint. For example, Kim (Kim et al., 2005) employs 8 adders for 4×8 block, and 8 adders for 8×4 block, while in our architecture we have 2 adders per Type 1 adder, and a total of 8 adders to derive each 4×8 and 8×4 block. Similarly, Kim (Kim et al., 2005) employs 2 adders for each 8×16 and 16×8 block, whereas in our architecture we have only 2 adders in Type 2 adder to derive both 8×16 and 16×8 blocks. We use the same amount of adders as Type 2 adder to derive 8×8 and 16×16 blocks. We also employ muxes/demuxes to choose the appropriate 4×4 block for developing larger block size SADs. These

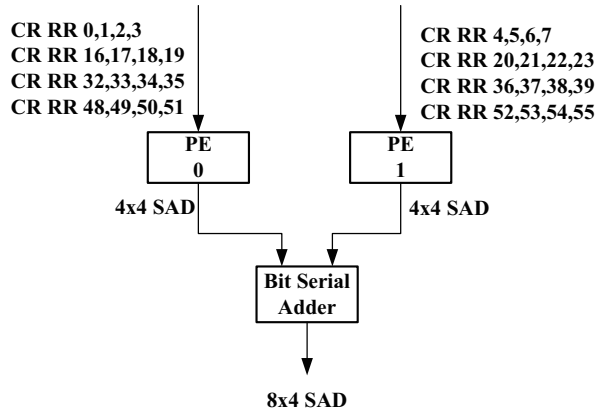


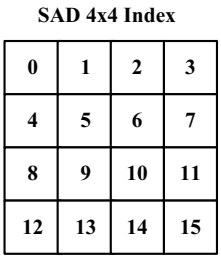
(a) Type 1 adder.



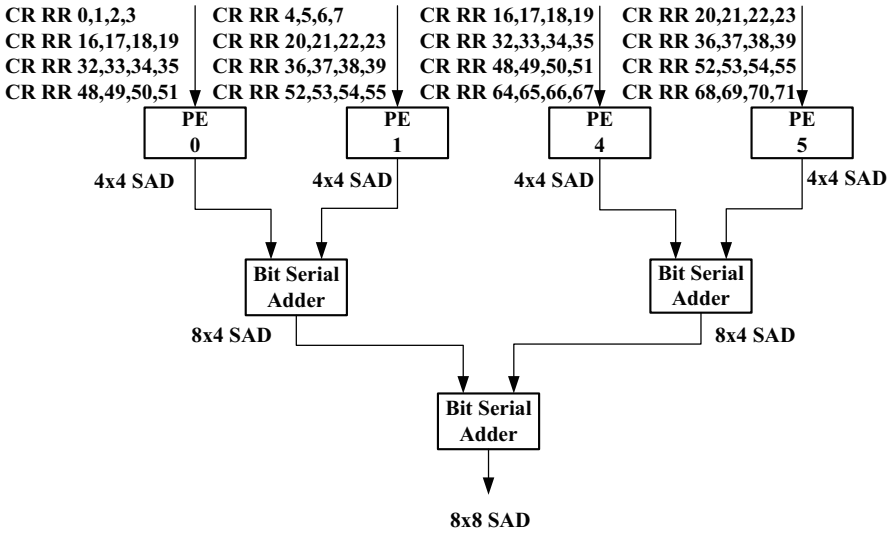
(b) Type 2 adder.

Figure 5.5: (a)Type 1 adder to generate 4×8, 8×4, and 8×8 SADs. (b)Type 2 adder for 8×16, 16×8, and 16×16 SADs. All the SAD outputs are compared with the minimum SADs for early termination.





(a) 8x4 SAD is accumulated by 4x4 SADs of PE 0 and PE 1



(b) 8x8 SAD is accumulated by 4x4 SADs of PE 0, PE 1, PE 4, and PE 5.

Figure 5.6: Example of SAD level data reuse.

muxes/demuxes do not exist in other bit parallel architectures, meaning that the architecture calculates all 4×4 blocks and then proceeds with the calculation of larger block sizes.

5.6 SAD Level Data Reuse

In a full search based architecture, all the SADs of the smallest 4×4 blocks are computed first, and the larger SADs can be accumulated by summing up the corresponding 4×4 SADs. As shown in Figure 5.6(a), we accumulate the 8×4 SAD by summing up the 4×4 SADs of PE 0 and PE 1. When computing the 8×8 SAD for the block (0, 1, 4, and 5, Figure 5.6b), the 4×4 SAD from PE 0 and PE 1 can be reused.

5.7 Functionality Analysis

In this section, we illustrate the functionality of the proposed architecture for the FS and the 3SS algorithms.

5.7.1 Full Search

After AG generates the addresses for the current and reference blocks ($curAddr$, $refAddr$) , 16 cycles are needed to read a 16×16 block into the on-chip RR and CR buffers (Figure 5.3) since the bandwidth is 128 bits/cycle as defined in Section 5.3. At cycle 17, PEs will start receiving the MSB bits from their corresponding buffers

as illustrated in Figure 5.4(a). At the same time, since the search pattern is zigzag, RRs shift their MSB bits to the neighboring column of RRs (RR0 to RR1, RR1 to RR2, RR16 to RR17, etc). Since ABD unit decides which register is larger (CR or RR) without any delay, the bit serial adders start operating on these MSBs in cycle 17 too. Also, in cycle 17, since MSBs are already in use by the PEs, the MSBs of first RR column (RR0, 16, 32, ..., 240 in Figure 5.3b) are overwritten with the new set of pixels arriving from the off-chip memory. It takes 8 cycles (Figure 4.6) for the bit serial adder to generate the MSB of a 4×4 SAD. Type 1 adder (Figure 5.5a) and Type 2 adder (Figure 5.5b) each introduces 4 cycles of delay. Therefore, the MSB passes through the Type 1 adder at the end of cycle 29 and the Type 2 adder at the end of cycle 33. Since the design is pipelined, after cycle 33, each cycle generates the next set of bits of the 16×16 SAD. The best case scenario occurs when the MSB of the new SAD is larger than the MSB of the stored SAD, after which the operation is terminated (cycle 33 for 16×16). The worst case scenario occurs if the minimum SAD decision is reached based on the least significant bit, which occurs in cycle 49. The best and worst case completion times for all block sizes are indicated in Table 5.1.

5.7.2 Three-Step Search

The AG generates the $curAddr$ and the $refAddr$ for the 3SS mode at the initial stage. The off-chip memory requires 16 cycles to load the pixels to the on-chip

Table 5.1: Number of clock cycles required to calculate the SADs for various block sizes for Full Search (FS) and Three-step search(3SS) with best case and worst case scenarios.

SAD Types	4×4	4×8	8×4	8×8	8×16	16×8	16×16
FS Cycles(Best)	25	27	27	29	31	31	33
FS Cycles(Worst)	37	40	40	43	46	46	49
3SS Cycles(Best)	32	34	34	36	38	38	40
3SS Cycles(Worst)	43	47	47	50	53	53	56

buffer. The step 1 skips 4 rows or columns between the successive candidate blocks. Therefore, the RRs shift 4 cycles to their neighboring columns/rows to compensate the gap. At cycle 17, the CRs and RRs fill the PE arrays with their MSBs and in the meantime, the first column of RRs (RR0, 16, 32, ..., 240 in Figure 5.3b) shift MSBs to their neighboring RRs (RR1, 17, 33, ..., 241). The AG updates the $refAddr$ to the next column and replaces contents of the first column of RRs (RR0, 16, 32, ..., 240) with the MSBs of the new pixels at cycle 17. This procedure (i.e., reading new bits and shifting the old ones) continues until 4 columns of pixels are skipped and all the RRs receive the corresponding bits to compute the SAD for the next point (cycle 20). In this scenario, the PE array stalls for three cycles. Therefore, the best and the worst case cycles are delayed by 4 cycles compared to the FS (Table 5.1).

After finding the minimum SAD among the nine points in step 1, the PE array returns the MV as the feedback to the address generator. The AG then generates the new address based on Equation 9. New reference block is read from the off-chip memory into the on-chip buffer. In step 2, the registers shift the bits for 2 cycles to meet the "s = 2" requirement, except its center point since it is already computed in

the step 1. Therefore, the RRs shift the pixel bits for 4 cycles to bypass the central point. Therefore, in step 2, the PE array has a stall of 2 cycles.

The step 3 follows the same flow of step 2. The MV of the minimum SAD is sent back to the AG to generate the new starting address for the off-chip memory access. The RRs shift the pixel bits 1 cycle for all the 8 blocks, except for the center which requires 2 cycles. Step 3, therefore has a stall of 1 cycle. After completion of all the steps on the current block, the AG updates the $curInd$ and starts a new round of SAD computations.

5.8 Performance Analysis

In this section, we compare the performance of our architecture with the previously reported designs.

5.8.1 Verification Approach

AMEA was designed with Verilog-HDL description and synthesized by using Synopsys Design Compiler with a 45 nm standard cell library technology under the nominal operating conditions (1.1 V, 25℃). Similar to the following published studies (Celebi and Erturk, 2010; Zhang and Wen, 2007; Lee et al., 2006; Cheng and Dung, 2005; Tuan et al., 2002; Vanne et al., April, 2009), we simulate the architecture to evaluate its performance, and use ASIC tools to synthesize the design for resource comparison without a system prototype.

5.8.2 Required frequencies to sustain different video formats

Table 5.2 shows the minimum clock rate that is required to sustain 30 fps and 60 fps frame rates by the proposed AMEA and its counterparts. As shown in Table 5.2, for QCIF format, the ASIC approach (Ou et al., 2005) needs to operate at 3.04 MHz to sustain the 30 fps requirement, whereas AMEA needs a faster clock, because the number of cycles required to compute 16×16 motion vector is larger in AMEA (best case 1056 cycles) than in (Ou et al., 2005) (1024 cycles). For clock rates slower than 4.36 MHz, AMEA won't be able to sustain 30 fps. As the frame size gets larger, the clock rate needs to increase to respond to more computations with the 30 fps or 60 fps constraint.

In Table 5.2, we also observe that all architectures described in this section (Ou et al., 2005; Yap and McCanny, 2004; Kim et al., 2005; Li and Leong, 2008) are capable of working at a practical frequency when encoding the QCIF and CIF video in real time. However, to sustain the requirements of a higher resolution video types, for example, SHDTV (1920×1080 at 60 fps), the bit serial architecture (Li and Leong, 2008) needs to operate at 8.95 GHz to execute the motion estimation algorithm in real time, whereas ASIC designs (Ou et al., 2005) and (Yap and McCanny, 2004) require only about 500 MHz, which makes the bit parallel architecture a more practical solution. Furthermore, AMEA operates at a close frequency (724 MHz) to its ASIC-based counterparts. The following two architectural features contribute to the fact that our AMEA requires 1.5x faster clock frequency

Table 5.2: The required clock rate of AMEA and its counterparts to sustain various video formats for different frame sizes and frame rates (full search, with [-16,+15] search range and 16×16 block size).

Frame size		QCIF (176×144)	CIF (352×288)	4CIF (704×576)	16CIF (1408×1152)	SDTV (1280×720)	HDTV (1280×720)	SHDTV (1920×1080)
Frame rate(fps)		30	30	30	30	30	60	60
Others (MHz)	Ou	3.04	12.1	48.65	194.6	110.6	221.2	**497**
	Yap	3.05	12.2	48.7	194.8	110.7	221.4	498
	Kim	53.5	214	854	3417	1942	3884	8739
	Leong	54.8	219	876	3504	1991	3981	8957
AMEA (MHz)	best	4.36	17.3	69.7	278.6	158.9	317.6	**713.7**
	worst	4.41	17.6	70.8	282.1	161.2	322.2	**724.1**

than (Ou et al., 2005) for the SHDTV format.

- Ou's architecture does not include the memory interface between the off chip memory and the PE arrays. Ou assumes that the pixel data is ready on chip for the computations. However in our AMEA, we design a specific memory architecture to address this realistic data transfer challenge. The number of cycles per MB reported in Table 5.3 includes the delay cycles from the off chip memory to the on chip buffer and from the on chip buffer to the PE arrays.

- Ou's architecture supports only the FS algorithm. Ou configures the data-path to pipeline the SAD computations without any stalls or data hazards, which results in a less number of cycles per MB. In our AMEA, we make the architecture adaptive to both the FS and the 3SS. This requires additional components such as the address generator and registers to adapt to two algo-

rithms, which is discussed in Section 5.3. These components lead to increase in the number of cycles required to process one MB.

5.8.3 Synthesis Results and Comparison

We report the clock frequency and gate equivalent area results along with other features such as power consumption, process technology, and search range in Table 5.3. The area values are based on equivalent 2-input NAND gates, whereas the clock frequency is based on the critical path delay. We also present the performance of the other 3SS and FS architectures using their published values for these performance metrics. The reported results are based on the single reference frame from a CIF format video sequence, and search range of $[-16, +15]$.

In addition, area metrics for all the architectures are tabulated without their memory modules. Since different processes and supply voltages are used by the other architectures, we also derive a normalized power consumption value for each architecture as in (Mudge, April, 2001) for a fair power performance comparison. We extend our evaluations to architectures without the VBSME support as most of the 3SS implementations are based on the FBSME. We report the best case and worst case cycles per macroblock for the AMEA. Best case is observed when early termination occurs at the MSB and worst case is observed when there is no early termination.

For the search range of $[-16, +15]$, a 16×16 macroblock is compared with 1024

Table 5.3: Performance comparison of the proposed AMEA with other designs. Block size: 16×16. Video format: CIF (352×288, 30 fps). Since early termination is adopted, we provide the results in both the best and the worst cases for our architecture.

Architecture	Algorithmm	VBS ME	PEs	Cycles per MB	freq.[1] (MHz)	Area (kgates)	Power (mW)	Process (nm)	Voltage (V)	Search Range(p)	norm. pow. (mW)[2]
Ours (AMEA)	FS 3SS	Yes	21	1056/1072 880/1280	17.6	34.3	1.98	45	1.1	16	1.98
Yap	FS	Yes	16	261	294[3]	61	95.04	130	1.2	8	27.64
Ou	FS	Yes	256	256	200[3]	597	20.48	180	1.8	8	1.91
Kim	FS	Yes	256	1025	100[3]	154	N.A.[4]	180	N.A.	16	N.A.
Leong	FS	Yes	N.A.	18432	420[3]	55	13919	180	N.A.	16	N.A.
Vanne	HEXS 3SS	No	16	390 680	200	14.2	59	130	1.2	16	17.16
Jung	3SS	No	9	337	100	N.A.	N.A.	N.A.	N.A.	8	N.A.
Jong	3SS	No	9	794	40	13.6	N.A.	N.A.	N.A.	8	N.A.
Chen	3SS	No	9	851	50	11.6	350	800	5.0	8	0.95
Zhang	FS 3SS	No	48	1591 204	N.A.	24	N.A.	N.A.	N.A.	8	N.A.
Chen	4SS	Yes	256	N.A.	13.5	131.2	2.13	180	1.3	16	0.38
Saponara	FFS[5]	No	64	6084	72.28	35	47	250	2.5	16	1.64
Chao	DS FFS	No	8	437 2879	50	9.0	224	350	3.3	16	3.21

[1]freq. = Frequency.

[2]norm. pow.=Normalized Power=$Power \times \frac{1.1^2}{Voltage^2} \times \frac{45}{Process}$.

[3]Only max frequency is reported.

[4]N.A. = Not Available.

[5]FFS = Fast Full Search.

reference blocks in the full search algorithm. With the early termination, minimum SAD for a 16×16 block size is ready at the end of cycles 33 and 49 in best and worst case scenarios respectively (Section 5.7). Rest of the computations are pipelined. Thus, to produce a 16×16 motion vector, 1056 cycles are needed for the best case and 1072 cycles are needed for the worst case.

As discussed in Section 5.7.2, the data stall introduced by the irregular memory access prevents the AMEA pipeline the SAD computation for the 3SS. During step 1, determining the minimum SAD for 16×16 block size occurs at the end of cycles 37 and 53 in best and worst case scenarios respectively. We observe the 4-cycle delay compared to the FS. It takes 333 cycles in best case and 477 cycles in worst case to complete 9 points. For the step 2, it takes 35 and 51 cycles to process one 16×16

block in the best and the worst case scenarios, respectively. Therefore, 8 checking points lead to 282 ($= 35 \times 8 + 2$) and 410 ($= 51 \times 8 + 2$) cycles for the best and worst cases. Similarly in step 3, the cycles for the single point are 33 and 49 in the best and the worst cases, which contribute to a total of 265 ($= 33 \times 8 + 1$) and 393 ($= 49 \times 8 + 1$) cycles. Thus in summary, in order to produce a 16×16 motion vector, the AMEA requires 880 cycles for the best case and 1280 cycles for the worst case scenarios.

The total logic gate count is 34.3k for the AMEA. Our design employs 21 hybrid grained processing elements out of which 16 of them are 4×4, 4 of them are 8×8 and 1 of them is at 16×16 granularity. This design, when clocked at 17.6 MHz, supports real time CIF encoding (352×288,30 fps) with the search range $[-16, +15]$. At 17.6 MHz, the power consumption of the implementation is 1.98 mW. The maximum operational clock frequency is 500 MHz. For higher resolution sequences, the architecture with the same search range can be accelerated under more strict delay constraints to 500 MHz to meet the real time encoding requirement. The power consumption in this case is 74.6 mW.

The AMEA, while supporting both the FS and the 3SS, is area efficient compared to all FS architectures. Among the FS architectures, Ou's design (Ou et al., 2005) performs the best in terms of its cycles per MB. This is due to the hierarchical 1D systolic array architecture, which employs the partial SAD computation technique. However, Ou's architecture has a large area overhead as it employs re-

dundant adders forming a larger adder tree structure to derive SADs for larger block sizes. This architecture has separate SAD modules for 4×4 sub-block computations with separate input ports for loading current block and search region data. Thus, reuse of search data between modules is not possible. This increases the amount of data required from the memory. Delay registers are used to schedule the datapath while accumulating the partial sum. The adders and registers contribute to the large area overhead. Yap's (Yap and McCanny, 2004) architecture implements the FS with less number of PEs, but at a cost of considerably increased clock cycles per MB (4 times more than the AMEA). This is because Yap's PE array requires 256 cycles for matching one single block, and it repeats 16 times to complete a full search for a 16×16 block. In terms of normalized power performance, the AMEA is comparable with the Ou's design. Redundant resources of Ou's architecture enables better performance in terms of cycles per MB with an 17x area overhead compared to the AMEA. Even though Leong's bit-serial design (Li and Leong, 2008) is close to the performance of the AMEA in terms of maximum clock rate and area metrics, this design uses 17 times more clock cycles per MB and suffers from large power consumption as it does not allow data reuse. After processing one reference block, Leong's architecture needs to load the next reference block completely from the off-chip memory, which leads to large memory transactions and consequently, a large number of cycles overhead. Leong also uses a redundant adder tree structure similar to Ou's (Ou et al., 2005), which results in larger area than our AMEA implementa-

tion. With a 500MHz peak clock rate, the AMEA design is more suitable compared to the FS architectures for sustaining the frame rate of other formats such as SDTV, HDTV, and beyond.

Since the 3SS operates on 9 points in the first steep and 8 points in the following two steps, the 3SS architectures (Jung (Jung and Lee, May, 2004), Jong (Jong et al., August 1994), and Chen (Chen, August 1998)) employ 9 PEs at 16×16 granularity level to exploit the parallelism. In other words, these architectures operate on 9MBs concurrently. In AMEA, we employ 16 PEs at 4×4 granularity level. Therefore, the 3SS architectures achieve better performance in terms of cycles per MB. However, the AMEA sustains the CIF format at a lower operational frequency compared to the 3SS based architectures, due to its intensive data reuse capability. The 3SS architectures require loading the new reference block completely from the off-chip memory for the next SAD computation, since data reuse is not available. The maximum operational frequency of the AMEA is higher than the 3SS architectures, due to our pipelined datapath design approach. Since our objective is to be able to run both the 3SS and the FS on the same hardware, we chose to design our architecture at a granularity of 1MB. We calculate one 16×16 SAD with the entire PE array and adder tree structures. This approach enables us to utilize the PEs effectively for both the 3SS and the FS.

The AMEA with fine grained PEs (16 PEs at 4×4 level) occupies larger area, even though the 3SS based architectures employ 9 PEs at 16×16 level. In the 3SS

based implementations, architecture is formed by simply replicating the 16×16 PEs nine times. The 3SS architectures don't employ registers since there is no data reuse mechanism. Since these architectures don't support VBSME, they also don't employ adder structures to derive SADs for different block size types. Although the AMEA is at 1MB granularity, the datapath is designed to compute a total of 41 SADs for all the seven different block sizes. Additionally, the delay registers within the OLFAs are needed by PEs to operate in bit serial manner. These factors contribute to the area overhead compared to the FBSME based 3SS implementations.

Zhang's architecture supports both the FS and the 3SS for a fixed block size (Zhang and Tsui, April, 1997). Zhang utilizes 48 PEs in the design, which are divided into 3 rows and 16 columns. Each row is responsible for one 16×16 SAD computation. In order to implement the FS and 3SS, Zhang distributes different blocks among the 3 rows of PEs. This method introduces idle cycles and requires a complicated switching network for the simultaneous data access and workload distribution. Our AMEA takes advantage of the data reuse and a MB-level granularity to avoid these issues.

According to (Etoh and Yoshimura, January 2005), if media coding power dissipation increases beyond a modest 100 mW, it will be hard to implement the media application in portable devices. The power consumption of our AMEA is only 1.98 mW, which is the lowest among all the references listed in the Table 5.3. Chen's architecture (Chen et al., April, 2007) consumes 2.13 mW for a single 4SS algorithm.

Our AMEA supports both the FS and the 3SS algorithms. We believe that we can reduce the power dissipation further if we remove the support for the FS. However, we maintain the support of two algorithms to provide the end user with the flexibility of choosing between the quality of the video service and the energy consumption. On one hand, the AMEA with its on-chip buffers introduces additional hardware resources compared to the bit serial and bit parallel architectures. On the other hand, this overhead is compensated by eliminating the redundant hardware overhead observed in the adder tree structures of the bit parallel architectures. From energy consumption point of view, the on-chip buffers in the AMEA increase the power consumption as the pixels are shifted among the neighboring registers. However, several architectural choices also compensate for this energy consumption overhead as summarized below:

- We avoid the redundancy of the adder tree (Figure 3.2) commonly employed by the ASIC designs. For example, in Figure 3.2, there are 8 adders for 4×8, and 8 adders for 8×4 in the bit parallel architectures, while in AMEA there are 2 adders per Type 1 adder, and a total of 8 adders to derive each 4×8 and 8×4 block. Similarly, Figure 3.2 shows 2 adders for each 8×16 and 16×8 block, whereas in AMEA there are only 2 adders in Type 2 adder to derive both 8×16 and 16×8 blocks. We employ muxes/demuxes to choose the appropriate 4×4 block for developing larger block size SADs. These muxes/demuxes do not exist in other ASIC architectures, meaning that the architecture calculates all

4×4 blocks and then proceeds with the calculation of larger block sizes.

- We utilize early termination to avoid redundant SAD calculations which also contributes to reducing the power consumption. In AMEA, the early termination scheme makes it possible to compare the SADs starting from the MSB. In the best case scenario, only 1 bit (MSB) is sufficient to make the decision and the subsequent computations to the current SAD become unnecessary.

We report other implementations such as the 4SS of Chen (Chen et al., April, 2007), Fast FS of Saponara (Saponara and Fanucci, January, 2004) and Diamond Search (DS) with Fast FS of Chao (Chao et al., May, 2002) to set the stage for the AMEA with a comprehensive overview of the state of the art. Early termination, intensive data reuse, pipelined datapath with bit serial execution, and memory access management tailored to the search pattern form the key features of the AMEA. In overall, with its normalized power performance and ability to operate with the FS and 3SS, and 500MHz peak operational frequency–the AMEA offers the end user with the flexibility of choosing between the compression quality and energy consumption as a priority.

5.8.4 Industrial H.264 IP Cores

In this subsection, we present a summary of the industrial IP cores for a thorough comparison in Table 5.4. Even though it is hard to reach to a conclusion on the relative performance of the AMEA compared to the industrial IP cores due to limited

Table 5.4: A summary of the industrial H.264 IP cores.

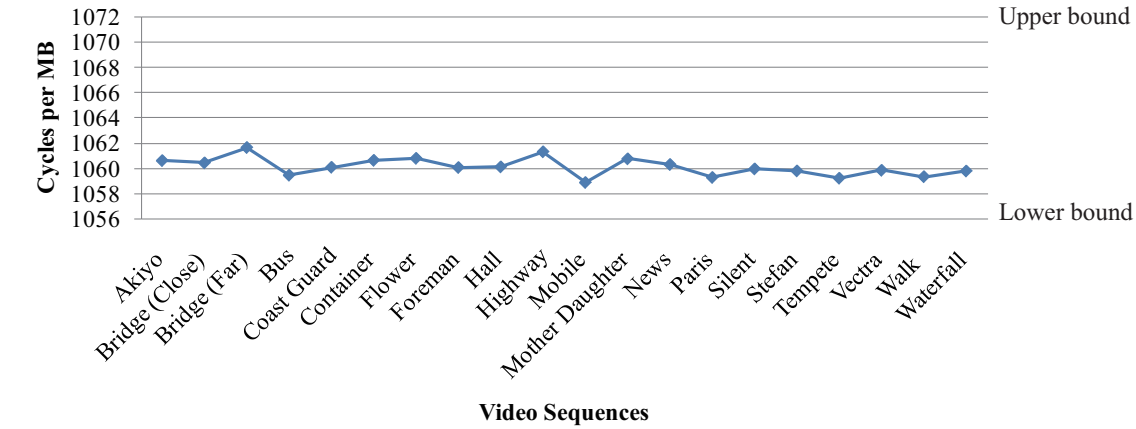
Vendor	IP Core Name	Video Format	Required Frequency	Search Algorithm	Process (nm)	Power Consumption(mW)
Chips&Media[1]	CODA960	1080p[6]	185MHz	N.A.[8]	N.A.	N.A.
Videantis[2]	v-MP4180HDX i	1080p	250MHz	N.A.	65nm	102mW
Silicon Image[3]	CineramIC 4K-3D	720p[7]	33MHz	N.A.	N.A.	N.A.
EyeLytics[4]	EyeLytics	1080p	300MHz	N.A.	130nm	N.A.
Jointwave[5]	E740	1080p	152MHz	Full Search	90nm	98mW

[1]Source: `http://www.chipsnmedia.com`
[2]Source: `http://www.videantis.com`
[3]Source: `http://www.siliconimage.com`
[4]Source: `http://www.eyelytics.com`
[5]Source: `http://www.jointwave.com`
[6]1920×1080, 30fps
[7]1280×720, 30fps
[8]N.A.=Not Available

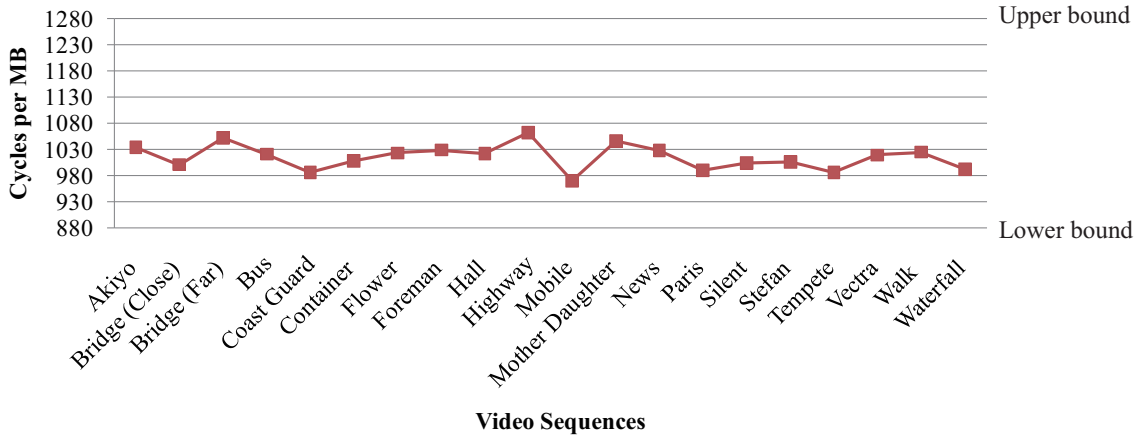
amount of information release for each product, we believe that the adaptive nature of the proposed architecture makes it a desirable solution for embedding in future encoder systems.

5.8.5 Functionality Verification

Performance of the proposed architecture, due to its early termination capability, depends on the video input as well. Essentially, our architecture is designed for general videos, which can be dramatically dynamic and diversified. As it is intractable for the end user to experiment with all types of videos, researchers always employ representative video sequences to evaluate the effectiveness and efficiency of their algorithm or architecture. For example, Chen (Chen et al., April, 2007) employs "stefan", "foreman", "silent" and "coastguard"; Olivares (Olivares et al., 2006) adopts "flower", "hall monitor", "tennis", and "coast guard". To verify the effectiveness of our architecture, we have experimented using a total of 20 standard



(a) Full Search



(b) Three-Step Search

Figure 5.7: Cycles per MB versus different video sequences, CIF format, search range [-16, +15]. (a) Full search. (b) Three-step search.

video sequences, and presented the results (cycles per MB) for both the FS and the 3SS in Figure 5.7 accordingly. These video benchmarks have been accepted by the video processing community as they carry the variations observed in video sequences. Therefore, we believe that the experimental results are sufficient to show the efficiency of our architecture for supporting real video sequences. Figure 5.7 verifies that the cycles per MB of all the video sequences fall in the range 1056∼1072 for the FS and 880∼1280 for the 3SS, as shown in Table 5.3.

5.9 Summary

In this chapter, we introduced an adaptive hardware architecture for the variable block size motion estimation routine of the H.264/AVC. Architecture utilizes several features such as bit serial execution on a pipelined datapath, early termination, intensive data reuse, and hybrid grained processing elements and memory access management tailored to the full search and three-step search methods. These strategies contributed towards designing a low-power architecture. The granularity of the whole architecture (computing 16×16 SAD for 1 macroblock with the whole PEs), the hybrid grained PEs (the bit serial execution and transmission), the on-chip buffer and the address generator for pixel level data reuse were the design choices we made when implementing the motion estimation architecture.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Summary of the Research

Variable block size motion estimation (VBSME) algorithm of H.264 supports seven block sizes. In addition to full search method, H.264 employs fast search algorithms for ME. Support for these features poses as a major barrier for the hardware implementations.

In this study, we first introduce a bit serial hybrid grained processing element based 2D architecture that supports the full search and the VBSME of H.264. PEs operate on most significant bit-first arithmetic for early termination and the 2D architecture enables on-chip data reuse between neighboring PEs in a bit-by-bit pipelined fashion. Hybrid grained PEs reduce the hardware overhead of conventional adder tree structures used for implementing the VBSME. Our design reduces the gate count by 7x compared to its ASIC counterpart, operates at a comparable frequency while sustaining 30fps and 60fps; and outperforms bit parallel and bit serial architectures in terms of throughput and performance per gate for various video formats.

We then improve our BSHG and introduce the first adaptive ME architecture

that provides the end user with the flexibility of choosing between the high quality video service during power-rich state (FS mode), and extended video service (fast search mode). We resolve the irregular indexing scheme challenge of three-step search (3SS) by introducing an on-chip buffer structure with a memory interface, which is adaptive to data access patterns of the FS and 3SS methods. The architecture sustains the real time CIF format (352×288) video encoding at 30fps with an operational frequency as low as 17.6MHz, and consumes 1.98mW based on the 45nm technology, outperforming all other FS and 3SS architectures.

6.2 Future Research Directions

To make the current research more complete, we plan to extend the current work along several directions as detailed below.

- Initial motion vector prediction in BSHG: We plan to further enhance the performance of BSHG by employing motion vector prediction mechanism borrowed from ASIC domain. An initial motion vector prediction (Wiegand et al., July, 2003) technique helps to choose a strategic initial motion vector and reduces the number of SAD computations. We believe there is significant potential in motion vector prediction combined with bit serial execution. Rate distortion optimization (RDO) technique considers the real cost of coding the macroblock before making a decision on which motion vector is the best. We will also consider how this technique could be used with the bit serial execu-

tion.

- Adaptive to other fast search algorithms in AMEA: The current shifting scheme inside the on-chip buffer is only applicable to the horizontal or vertical memory access patterns. This limits the application of this architecture to algorithms such as FS, 3SS, and 4SS. If a fast search method requires diagonal access pattern such as in the diagonal search and the hexagon search, our architecture cannot offer data reuse. Improving the architecture to function for a large set of fast search algorithms requires introducing diagonal interconnections between the reference and current block registers. Another potential improvement is the inclusion of dynamic control logics inside the PEs to terminate the searching at different cycles and avoid resource under-utilization potentiality for the diamond and hexagon searches.

- Further power reduction techniques in AMEA: Our focus on intensive data reuse and early termination capabilities have been beneficial in terms of power performance as well. However, we believe that there is room for further reduction in the power consumption. For example, if we treat hybrid grained processing elements as individual processing cores, we can turn on/off these components on a need basis. Such a mechanism also provides the end user with the choice of trading off between the video quality and the computation complexity. For video sequences with regular and smooth motion fields,

we can turn off the 4×4 SAD computations since 4×4 blocks are more efficient for the fierce and fine granularity movement (Wiegand et al., 2003). We are also interested in evaluating the feasibility of other power reduction techniques for our AMEA, including clock gating, frequency scaling, voltage islands (Chin and Yu, October, 2005), etc.

- Fabrication: We believe that our design and optimization approaches for the ME have reached to a situation point. Therefore, besides the other future work listed above, our main priority is to have this design fabricated.

REFERENCES

Avizienis, A. (1961). Signed-Digit Number Representations for Fast Parallel Arithmetic. *IRE Trans. Electornic Computers*, **EC-10**(9), pp. 389–400.

CCITT (June, 1989). Description of Reference Model 8 (RM8). Technical Report Document 525, Study Group XV, Working Party XV/4, Specialists Group on Coding for Visual Telephony.

Celebi, A. and S. H. J. L. Erturk (2010). Bit Plane Matching Based Variable Block Size Motion Estimation Method and Its Hardware Architecture. *IEEE Transactions on Consumer Electronics*, **56**(3), pp. 1625–1633.

Chao, W. M., C. W. Hsu, Y. C. Chang, and L. G. Chen (May, 2002). A Novel Hybrid Motion Estimator Supporting Diamond Search and Fast Full Search. In *Proceedings IEEE International Symposium on Circuits and Systems*, pp. 492–495. IEEE, Phoenix, AZ.

Chen, C. Y., S. Y. Chien, Y. W. Huang, T. C. Chen, T. C. Wang, and L. G. Chen (2006). Analysis and architecture design of variable block-size motion estimation for H.264/AVC. *IEEE Transactions on Circuits Syst. Video Techonology*, **53**(2), pp. 578–593.

Chen, T. C., Y. H. Chen, S. F. Tsai, S. Y. Chien, and L. G. Chen (April, 2007). Fast Algorithm and Architecture Design of Low-Power Integer Motion Estimation for H.264/AVC. *IEEE Transactions On Circuits and Systems for Video Technology*, **17**(5), pp. 568–577.

Chen, T. C., S. Y. Chien, Y. W. Huang, C. H. Tsai, C. Y. Chen, T. W. Chen, and L. G. Chen (June, 2006). Analysis and Architecture Design of an HDTV720p 30 frames/s H.264/AVC Encoder. *IEEE Transactions On Circuits and Systems for Video Technology*, **16**(6), pp. 673–688.

Chen, T. H. (August 1998). A Cost-Effective Three-Step Hierarchical Search Block-Matching Chip for Motion Estimation. *IEEE Journal of Solid-State Circuits*, **33**(8), pp. 1253–1258.

Chen, Z., J. Xu, Y. He, and J. Zheng (April, 2006). Fast Integer-pel and Fractional-pel Motion Estimation for H.264/AVC. *Journal of Visual Communication and Image Representation*, **17**(2), pp. 264–290.

Cheng, H. W. and L. R. Dung (2005). A Content-Based Methodology for Power-Aware Motion Estimation Architecture. *IEEE Transactions on Circuits and Systems II: Express Briefs*, **52**(10), pp. 631–635.

Chin, P. Y. and C. C. Yu (October, 2005). A voltage level converter circuit design with low power consumption. In *6th International Conference On ASICON 2005*, pp. 309–310. ASIC, Shanghai.

Ebeling, C., D. C. Cronquist, P. Franklin, and C. Fisher (1999). RaPiD - A Configurable Computing Architecture for Compute-Intensive Applications. Technical Report TR-96-11-03, University of Washington, Department of Computer Science & Engineering.

Ercegovac, M. D. and T. Lang (1989). On-line Arithmetic for DSP Applications. In *32nd Midwest Symposium on Circuits and Systems*. Urbana.

Etoh, M. and T. Yoshimura (January 2005). Advances in wireless video delivery. In *Proceedings of the IEEE*, volume 93, pp. 111–122. DoCoMo Communications Laboratories, San Jose, CA.

Jehng, Y. S., L. G. Chen, and T. D. Chiueh (1993). An Efficient and Simple VLSI Tree Architecture for Motion Estimation Algorithms. *IEEE Transactions on Circuits Syst. Video Techonology*, **41**(2), pp. 889–900.

Jong, H. M., L. G. Chen, and T. D. Chiueh (August 1994). Parallel Architectures for 3-Step Hierarchical Search Block-matching Algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, **4**(4), pp. 407–416.

Jung, S. T. and S. S. Lee (May, 2004). A 4-way Pipelined Processing Architecture for Three-Step Search Block-matching Motion Estimation. *IEEE Transactions on Consumer Electronics*, **50**(2), pp. 674–681.

Kim, M., I. Hwang, and S. I. Chae (2005). A fast VLSI Architecture for Full-search Variable Block Size Motion Estimation in MPEG-4 AVC/H.264. In *Proc. ASP-DAC*, pp. 631–634.

Koga, T., K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro (1981). Motion Compensated Interframe Coding for Video Conferencing. In *Proceedings National Telecommunications Conference*, pp. G5.3.1–G5.3.5. IEEE, New Orleans, LA.

Kuhn, P. M. (1999). Fast MPEG-4 Motion Estimation: Processor Based and Flexible VLSI Implementations. *Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, **23**(1), pp. 67–92.

Lappalainen, V., A. Hailapuro, A. Hamalainen, and T. D. Hamalainen (2001). Performance of H.26L video encoder on general-purpose processor. *The Journal of VLSI Signal Processing*, pp. 266–267.

Lee, J., N. Vijaykrishnan, M. J. Irwin, and W. Wolf (2006). An Efficient Architecture for Motion Estimation and Compensation in the Transform Domain. *IEEE Transactions on Circuits and Systems for Video Technology*, **16**(2), pp. 191–201.

Li, B. M. H. and P. H. W. Leong (2008). Serial and Parallel FPGA-based Variable Block Size Motion Estimation Processors. *Journal of VLSI Signal Processing*, **51**(1), pp. 77–98.

Liu, L. K. and E. Peig (April, 1996). A Block-based Gradient Descent Search Algorithm for Block Motion Estimation in Video Coding. *IEEE Transactions on Circuits and Systems for Video Technology*, **6**(8), pp. 419–423.

Marshall, A., T. Stansfield, I. Kostarnov, J. Vuillemin, and B.Hutchings (1999). A Reconfigurable Arithmetic Array for Multimedia Applications. In *Proc. ACM/SIGDA FPGA'99*, pp. 21–23. Monterey.

Mirsky, E. and A. DeHon (1996). MATRIX: A Reconfigurable Computing Architecture with Configurable Instruction Distribution and Deployable Resources. In *Proc. IEEE FCCM'96*, pp. 17–19. Napa, CA, USA.

MPEG-1 (1993). Information Technology-Coding of Moving Pictures and Associated Audio for Digital Storage Media at up to about 1.5 Mbits/s: Video. Technical report, ISO/IEC CD 11172-2 (MPEG-1 Video).

Mudge, T. (April, 2001). Power: A First-Class Architectural Design Constraint. *Computer*, **34**(4), pp. 52–58.

Olivares, J., J. Hormigo, J. Villalba, I. Benavides, and E. L. Zapata (2006). SAD Computation based on Online Arithmetic for Motion Estimation. *Microprocessors and Microsystems*, **30**(5), pp. 250–258.

Ou, C.-M., C.-F. Le, and W.-J. Hwang (2005). An Efficient VLSI Architecture for H.264 Variable Block Size Motion Estimation. *IEEE Transactions on Consumer Electronics*, **51**(4), pp. 1291–1299.

Po, L. M. and W. C. Ma (June, 1996). A Novel Four-Step Search Algorithm for Fast Block Motion Estimation. *IEEE Transactions On Circuits and Systems for Video Technology*, **6**(6), pp. 313–317.

Rao, K. R. and J. J. Hwang (1996). *Techniques and Standards for Image, Video, and Audio Coding*. Prentice Hall, Upper Saddle River, N.J.

Reader, S. and T. Meng (1999). Performance Evaluation of Motion Estimation Algorithms for Digital Signal Processors. Technical report, Stanford University.

Saponara, S. and L. Fanucci (January, 2004). Data-Adaptive Motion Estimation Algorithm and VLSI Architecture Design for Low-Power Video Systems. *IEE Proceedings Computers and Digital Techniques*, **151**(1), pp. 51–59.

Shen, J.-F., T.-C. Wang, and L.-G. Chen (2001). A Novel Low-Power Full-Search Block-Matching Motion-Estimation Design for H.263+. *IEEE Transactions on Circuits and Systems for Video Technology*, **11**(7), pp. 890–897.

Song, Y. and A. Akoglu (2010). Bit-by-Bit Pipelined and Hybrid-Grained 2D Architecture for Motion Estimation of H.264/AVC. *Journal of Signal Processing Systems, Accepted*.

Song, Y. and A. Akoglu (2011). An Adaptive Motion Estimation Architecture for H.264/AVC. *ACM Transactions on Embedded Computing Systems (in review)*.

Soohoo, A. (2005). FPGA Co-Processing Architectures for Video Compression. Technical report, Altera Corporation.

Su, C.-L. and C.-W. Jen (2000). Motion Estimation Using On-Line Arithmetic. In *IEEE International Symposium on Circuits and Systems*. Switzerland.

Tourapis, A. M. (January, 2002). Enhanced Predictive Zonal Search for Single and Multiple Frame Motion Estimation. In *Proceedings of Visual Communications and Image Processing (VCIP '02)*, pp. 1069–1079. SPIE, San Jose, CA.

Tourapis, A. M., O. C. Au, and M. L. Liou (January, 2001). Predictive Motion Vector Field Adaptive Search Technique (PMVFAST): Enhancing Block-based Motion Estimation. In *Proceedings of Visual Communications and Image Processing (VCIP '01)*, pp. 883–892. SPIE, San Jose, CA.

Tuan, J. C., T. S. Chang, and C. W. Jen (2002). On the Data Reuse and Memory Bandwidth Analysis for Full-Search Block-Matching VLSI Architecture. *IEEE Transactions on Circuits and Systems for Video Technology*, **12**(1), pp. 61–72.

Vanne, J., E. Aho, K. Kuusilinna, and T. D. Hamalainen (April, 2009). A Configurable Motion Estimation Architecture for Block-Matching Algorithms. *IEEE Transactions on Circuits and Systems for Video Technology*, **19**(4), pp. 466–476.

Verma, R. and A. Akoglu (2007). A coarse grained reconfigurable architecture for variable size block motion estimation. In *IEEE International Conference on Field-Programmable Technology 2007 (ICFPT'07)*, pp. 81–88. Kitakyushu, Japan.

Vos, L. D. and M. Stegherr (1989). Parameterizable VLSI Architectures for the Full-search Block-matching Algorithm. *IEEE Transactions on Circuits Syst.*, **36**(2), pp. 1309–1316.

Waingold, E., M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasingbe, and A. Agarwal (1997). Baring it all to Software: RAW Machines. *IEEE Computer*, pp. 86–93.

Wiegand, T., G. J. Sullivan, G. Bjontegaard, and A. Luthra (2003). Overview of the H.264/AVC video coding standard. *IEEE Transactions On Circuits and Systems for Video Technology*, **13**(7), pp. 560–576.

Wiegand, T., G. J. Sullivan, G. Bjontegaard, and A. Luthra (July, 2003). Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions On Circuits and Systems for Video Technology*, **13**(7), pp. 560–576.

Xiong, X. X., Y. Song, and A. Akoglu (2011). Architecture Design of Variable Block Size Motion Estimation for Full and Fast Search Algorithms in H.264/AVC. *Computers Electrical Engineering, Accepted*.

Yap, S. Y. and J. V. McCanny (2003). A VLSI Architecture for Advanced Video Coding Motion Estimation. In *Proc. IEEE Intl. Conf. applications-specific systems, arch., processors*, pp. 293–301.

Yap, S. Y. and J. V. McCanny (2004). A VLSI architecture for variable block size video motion estimation. *IEEE Transactions on CAS II*, **51**(7).

Zhang, L. and G. Wen (2007). Reusable Architecture and Complexity-Controllable Algorithm for the Integer/Fractional Motion Estimation of H.264. *IEEE Transactions on Consumer Electronics*, **53**(2), pp. 749–756.

Zhang, X. D. and C. Y. Tsui (April, 1997). An Efficient and Reconfigurable VLSI Architecture for Different Block Matching Motion Estimation Algorithms. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, pp. 603–606. IEEE, Munich, Germany.

Zhu, C., X. Lin, and L. P. Chau (May, 2002). Hexagon-based Search Pattern for Fast Block Motion Estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, **12**(5), pp. 349–355.

Zhu, S. and K. K. Ma (February, 2000). A New Diamond Search Algorithm for Fast Block Matching Motion Estimation. *IEEE Transactions Image Processing*, **9**(2), pp. 287–290.