

HIGH THROUGHPUT ARCHITECTURES FOR VARIABLE BLOCK SIZE MOTION ESTIMATION

by

STEPHEN WARRINGTON

A thesis submitted to the
Department of Electrical and Computer Engineering
in conformity with the requirements for
the degree of Master of Science (Engineering)

Queen's University
Kingston, Ontario, Canada
September 2006

Copyright © Stephen Warrington, 2006



Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 978-0-494-18841-5

Our file Notre référence

ISBN: 978-0-494-18841-5

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.


Canada

Abstract

This thesis addresses the most computationally complex portion of a video encoder: motion estimation (ME). The high complexity of the ME process requires dedicated hardware for real-time encoding in embedded systems. In addition, new standards have added new features. These new features require more complex forms of ME to take advantage of the possible rate-distortion (R-D) gains. First described is a scalable high-throughput VLSI design for full-search (FS) H.264/AVC variable block size motion estimation (VBSME). The proposed ME architecture offers a high level of throughput, superior to other full search architectures in the literature. This architecture is suitable for standard definition resolution.

Next proposed is an extremely high throughput VBS ME architecture suitable for HD video with multiple reference frames (MRF). This architecture uses a search algorithm that offers a tradeoff between high spatial resolution in a single reference frame or multiple reference frames at a lower spatial resolution. This tradeoff allows for the efficient use of MRF coding. The architecture is shown to be suitable for HD resolutions and frame rates. The architecture is novel in that it is the first in the literature to offer a tradeoff between spatial resolution and multiple reference frames.

Acknowledgments

I would like to thank my supervisors, Dr. Wai-Yip Geoffrey Chan and Dr. Subramania Sudharsanan. Over the past two years, they have given me a tremendous amount of valuable help. In addition to their vast technical expertise, their guidance has helped me develop into being a more rigorous and proficient researcher and engineer.

I would like to thank the Natural Sciences and Engineering Research Council for providing the funding for my graduate studies. In addition, I would like to thank CMC Microsystems for their valuable support for our research.

Lastly, I would like to thank family and friends for their moral support during my graduate studies. It's been much appreciated, thank you!

Contents

Abstract	i
Acknowledgements	ii
Contents	iii
List of Tables	vi
List of Figures	vii
List of Abbreviations	ix
Chapter 1 Introduction	1
1.1 Digital Video	1
1.2 Hybrid Video Coding	3
1.3 Motion Estimation	5
1.3.1 Overview	5
1.3.2 ME Computational Requirements	7
1.3.3 Full Search ME	9
1.3.4 Fast Motion Estimation (FME)	10

1.3.5	Advanced ME Features	11
1.3.6	Memory Bandwidth	14
1.4	Optimizations and Parallelism in ME	16
1.4.1	Overview	16
1.4.2	SAD Parallelism	16
1.4.3	Candidate MV Parallelism	17
1.4.4	VBS Partial SAD Sums	17
1.5	Motivation of Thesis Work	17
1.6	Thesis Contributions	18
Chapter 2	Literature Review	20
2.1	VLSI Architectures for Full Search	20
2.2	Fast Motion Estimation (FME) Algorithms	21
2.2.1	FME for VBSME	25
2.3	VLSI Architectures for Fast Motion Estimation	27
2.4	VLSI Architectures for Variable Block Size ME	27
Chapter 3	VBSME Architecture	32
3.1	Introduction	32
3.2	Architecture	32
3.2.1	Architecture Fundamentals	32
3.2.2	Processing Element (PE)	34
3.2.3	Inter-PE Interaction	45
3.3	Architecture Comparison	51
3.4	Local Memory Structure	54

Chapter 4	HD Resolution VBSME	57
4.1	VBSME for HD	57
4.2	Sub2-1 and Sub4-5	58
4.2.1	Algorithm Description	58
4.2.2	Computational Complexity	61
4.3	Algorithm Performance	63
4.3.1	Sub4-5 and Sub2-1 versus Full Search	63
4.3.2	Sub4-5 versus Sub2-1	64
4.4	Implementation	67
4.4.1	Architecture Overview	67
4.4.2	Synthesis Results	70
4.4.3	Comparison	71
4.5	Future Improvements	73
4.5.1	Overview	73
4.5.2	R-D Operation	74
4.5.3	Fractional Pel ME	74
Chapter 5	Conclusions	76
	Bibliography	78

List of Tables

3.1	SAD Output Schedule, 1x16 implementation	39
3.2	SAD Output Schedule, 1x4 implementation	40
3.3	Sub-PE Bus Timing	43
3.4	Required Frequency of Operation (1x16 architecture)	44
3.5	Pixel Data Schedule, 1x16 Architecture, 16 PEs	48
3.6	Performance comparison to other VBS designs (Results normalized to 16x16 search range)	52
4.1	Search Schemes with and Without Frame Level Switching Foreman 352x288 at 30 FPS, QP 30	67
4.2	Architecture Comparison	73

List of Figures

1.1	Hybrid video encoder.	4
1.2	ME Block Matching.	5
1.3	Block patterns found in H.264/AVC standard.	13
2.1	Diagram of the three stages of the TSS search.	23
3.1	Pixel numbering for architecture in [24].	36
3.2	4x4 block SAD numbering.	38
3.3	1x4 VBS PE structure.	41
3.4	1x16 VBS PE structure.	43
3.5	High level view of motion estimation architecture.	45
3.6	Processing element (PE) array.	46
3.7	Pixel transmission order for 1x4 and 1x16 implementations.	47
3.8	Reference frame bus activity for 32 1x16 PEs, 32x32 search area.	49
3.9	Local memory tiles, read versus write mode for three macroblocks.	56
4.1	Mobile 720x576 at 30 FPS encoded using different search schemes.	64
4.2	Football 720x480 at 30 FPS encoded using different search schemes.	65
4.3	Foreman 352x288 at 30 FPS encoded using different search schemes.	65

4.4	High Level view of proposed architecture.	68
4.5	Structure of Stage 1 PE.	69

List of Abbreviations

Abbreviation	Description
AD	Absolute Difference
FME	Fast Motion Estimation
FPS	Frames per Second
FS	Full Search
HD	High Definition
MCP	Motion Compensated Prediction
ME	Motion Estimation
MRF	Multiple Reference Frame
MSE	Mean-Squared Error
MV	Motion Vector
NTSS	New Three Step Search
PE	Processing Element
PSNR	Peak Signal-to-Noise Ratio
R-D	Rate-Distortion
SD	Standard Definition
SAD	Sum of Absolute Differences
TSS	Three Step Search
VBS	Variable Block Size
VBSME	Variable Block Size Motion Estimation
VLSI	Very Large-Scale Integration

Chapter 1

Introduction

1.1 Digital Video

There has been a growing demand for a large variety of digital video products. Leading the rapid expansion has been entertainment applications, such as DVD video, high definition (HD) television, Internet video streaming, digital cameras and recorders, HD video camcorders, personal video recorders for television, and many other applications. In addition to entertainment applications, commercial and industrial applications such as video teleconferencing and surveillance have also resulted in a growing demand for video.

A variety of new codecs have been developed to meet the growing consumer expectations for video quality. These new codecs are able to provide superior video quality and less distortion at the same bitrate, or provide equivalent video quality at a lower bitrate. To meet these demands, the new video codecs employ more advanced features than previous codecs, many of which have a high computational complexity.

The H.264/AVC video codec [3] offers a high level of compression efficiency over

past standards, but with much higher computational requirements. For storage applications, H.264/AVC has taken a dominant role for upcoming digital media. It is a mandatory codec for the upcoming HD-DVD and Blu-ray high-definition standards. Therefore, any decoder for these two new formats will need to be able to decode H.264/AVC. One common point of comparison is that H.264/AVC offers the same video quality at half the bitrate of the popular MPEG-2 [1] video standard. Due to its compression efficiency and its wide adoption, H.264/AVC has become the most important new standard for digital video. The standard has been offered for use in many types of applications, with different features supported at different application profile levels. A baseline profile, intended for low cost applications is offered. An extended profile, intended for streaming applications is also offered. Two profiles, the Main Profile and the High Profile are intended for broadcast and storage applications.

This thesis addresses the most computationally demanding aspect of the H.264/AVC encoder: variable block size motion estimation (VBSME). The architecture in this thesis is suitable for the fine grained level of VBSME required in H.264/AVC. However, it can also be used in other video standards which employ a similar form of VBS encoding. One notable codec that uses VBS coding is the Microsoft-led VC-1 (SMPTE 421M) standard [6], which uses a subset of the block sizes found in H.264/AVC.

Two high throughput VLSI architectures for VBSME are proposed to address motion estimation, one suitable for single reference frame and another suitable to multiple reference frames (MRF).

1.2 Hybrid Video Coding

Most current video standards, such as MPEG-2 [1], VC-1 [6] and H.264/AVC [3], use a block-based hybrid motion compensated prediction (hybrid MCP) structure. MPEG-4 [4] also includes a block-based motion compensation mode.

The codecs are block based in the sense that a video frame is decomposed into a set of rectangular blocks called *macroblocks*, each of which is coded separately by the encoder. A combination of approaches to exploit spatial and temporal redundancy are employed in order to code each of the blocks, hence the term *hybrid*. Spatial redundancy is exploited by using a transform coder, such as the discrete cosine transform (DCT), and using adjacent blocks for predicting coefficients. Temporal redundancy is exploited using *motion compensated prediction*. A block diagram of a typical hybrid video encoder is given in Figure 1.1.

Note that Figure 1.1 is a simplified version of a common video coder. Certain advanced features, are omitted from the diagram. For example H.264/AVC contains an in-loop deblocking filter. Also, it is possible for an encoder to have a rate control system to modify quantization levels and other characteristics to maintain a specific bitrate, which is omitted from the diagram.

Of the blocks found in the video encoder, the motion estimation (ME) block is by far the most computationally demanding. As an example, consider the standard definition 720x480, 30 frame per second “Mobile and Calendar” sequence with full search (to be defined later), using the JM standard reference code [5]. On a Pentium 4 2.8 GHz machine with 2.0 GB of RAM, motion estimation comprised 88% of the overall computation time. Note that the JM encoder is created to generate a bitstream using all the options available in the H.264/AVC standard, and is not

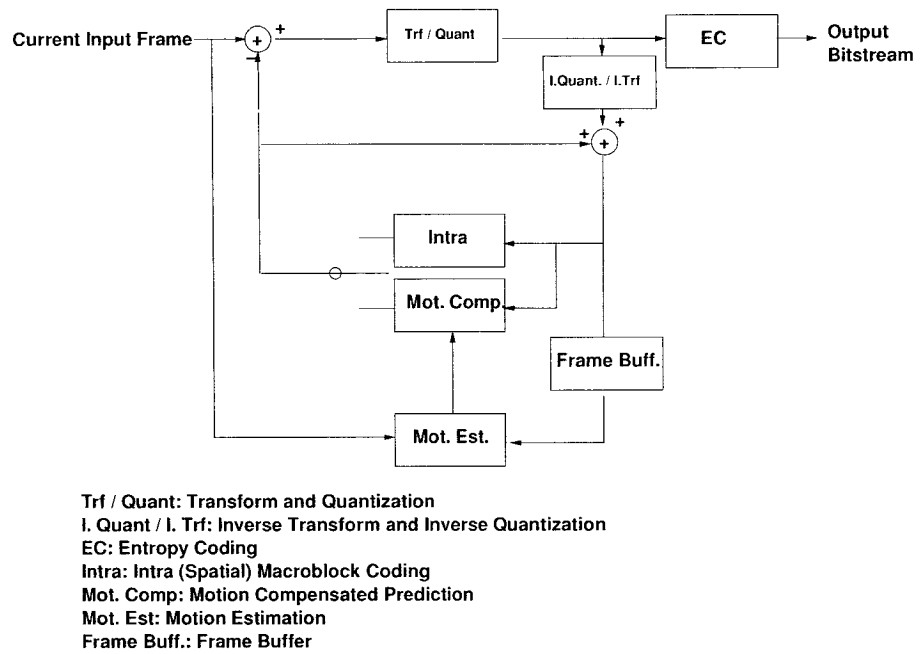


Figure 1.1: Hybrid video encoder.

heavily optimized for speed. On the test machine, the operation of the encoder is much less than real-time. This figure is sequence and search algorithm dependant, but ME can almost always be expected to be the most computationally dominant part of the video encoder. However, ME algorithms also contain a large amount of inherent parallelism, as will be described in the next section. Therefore, motion estimation is a prime target for optimization when attempting to create a real-time encoder.

1.3 Motion Estimation

1.3.1 Overview

In order to reduce the bitrate and achieve compression, video codecs exploit both temporal and spatial correlation to achieve compression. Current video coders use motion-compensated prediction (MCP) to exploit temporal redundancy in a video sequence to achieve compression. The underlying assumption for MCP is that two temporally adjacent (or temporally close) frames have similar content. This means that prior frame data can be used to form a good prediction for a block in the current frame. Due to camera movement and moving objects in the scene, a block may be located at a different point in the current encoded “target” frame than in the past frame. To accommodate this, the encoder will typically segment the current frame into regions referred to as “macroblocks”, and then use a block matching process to find the best match in a prior decoded frame, referred to as the “reference frame”. This block matching process is motion estimation (ME). A motion vector (MV) indicates the relative position of the current frame macroblock to the equivalent block in the reference frame. This is shown graphically in Figure 1.2. The motion vector in Figure 1.2 is shown to extend from one frame to another for visual purposes, the coded motion vector is a 2-D vector on the (x,y) plane.

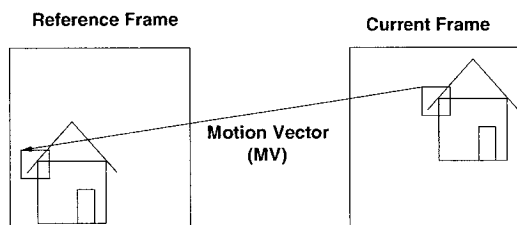


Figure 1.2: ME Block Matching.

For each macroblock, one or more reference frames are searched to find the “best” matching block. “Best” in this case means minimizing some form of distortion measure between the current macroblock and the matching block. The displacement of the best matching block relative to the macroblock provides a motion vector estimate. Most video coders operate in the YUV colour space, where there is one channel for luminance information (Y) and two channels for chrominance information (U,V). The luminance channel contains more information regarding object edges, which is necessary for proper motion estimation. The search is typically calculated with only the Y channel, to reduce the overall computation requirement of the ME process.

One common distortion measure is the sum of absolute differences (SAD). A basic ME search using the SAD is shown in Equations (1.1) to (1.3) where C represents the current macroblock of $N \times N$ pixels ($N=16$), and R represents a macroblock in the reference frame, displaced by the vector $\mathbf{v}$ from the position of the current macroblock, with $\mathbf{v}$ selected from the search space S. The result of the search is a motion vector (MV).

$$\mathbf{v} = (v_x, v_y) \quad (1.1)$$

$$SAD(\mathbf{v}) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} |C(i, j) - R(i + v_x, j + v_y)| \quad (1.2)$$

$$MV = \underset{\mathbf{v} \in S}{\operatorname{argmin}}(SAD(\mathbf{v})) \quad (1.3)$$

The choice of the search space and distortion measure can affect the quality of the selected prediction block. The “quality” of the prediction here refers to the rate-distortion (R-D) performance of the coder based on the predictions. The rate (R) is affected by the size of the residual coefficients that need to be coded, and the side information that needs to be transmitted for the MVs. Distortion (D) can be affected

directly from quantization effects, and indirectly from higher levels of quantization being applied due to poor predictions and large residuals in a rate-controlled encoder.

Motion estimation can be separated into integer pel ME and fractional pel ME. The H.264/AVC standard includes fractional pel ME down to a resolution of $1/4$ pel. Since the reference frame does not contain pixel values at fractional positions, it is necessary to perform some form of interpolation between integer positions to generate the fractional pel information. Because of the high computational requirement for generating fractional pel pixels, it is typically performed as a refinement step following the integer pel search. The computation and storage requirement for performing a fully fractional pel search would also be prohibitively high, since 16 pixel values are required every integer pel position, at $1/4$ resolution. This thesis focuses on the first part of the search, the integer pel position estimation, as it forms the majority of the complexity of the motion estimation operation.

1.3.2 ME Computational Requirements

In this thesis, systems for real-time ME are considered. Using this requirement, all macroblocks in a frame must be processed at the desired frame rate, at a minimum. To compute the computational requirements for a real-time search, we define the following terms:

- W : Width of the current frame
- H : Height of the current frame
- N_h : Horizontal width of an individual macroblock
- N_v : Vertical height of an individual macroblock

- *FPS*: Frame rate (Frames per second)
- *C*: Cycles per macroblock
- *f*: Clock frequency

Using these terms, the maximum time period to perform block matching on one macroblock is:

$$max_MB_period = \frac{1}{(\frac{W}{N_h} \times \frac{H}{N_v} \times fps)} \quad (1.4)$$

The *max_MB_period* given in Equation 1.4 represents the total amount of time that can be spent per macroblock while still being able to support the desired FPS. Using this equation, we can determine the below relation for real-time operation.

$$MB_period < \frac{C}{f} \quad (1.5)$$

In the above equation, the macroblock cycle time *C* refers to the number of cycles per macroblock. *f* refers to the maximum achievable clock frequency of the system. Control of these two variables allows a designer to optimize an architecture for high performance. In a software system, *f* is fixed. In a hardware solution, *f* can be controlled by changing the design complexity of the internal logic, and through attention to VLSI design principles such as internal pipelining. For a hardware solution, some tradeoff can be made to increase *f*, typically at the cost of a higher system area. The macroblock cycle time, *C*, offers a high degree of control, both in terms of software and hardware solutions. A high-level control is in the selection of the ME algorithm. At a lower level, within an individual search algorithm, it is possible to optimize *C*

through the overall implementation that is used. These details will be explored in later sections.

Although this covers the computational requirements of the base integer pel ME algorithm, certain advanced features such as variable block size ME can result in additional computational complexity.

1.3.3 Full Search ME

A common ME search method is to find the reference block which minimizes the SAD in a rectangular search area centered at the location of the current macroblock. This is referred as the full search (FS) algorithm. In the rest of this document, an $M \times N$ search rectangle refers to a rectangular search region, of size M horizontally and N vertically, centered on the current macroblock's position. The search range can affect the overall R-D performance and computational requirement of the full search. The search range that is necessary is data-dependant, both in terms of the resolution of the sequence and the amount of motion found within the scene. Search ranges can vary from approximately 16 for Quarter Common Intermediate Format (QCIF) (176x120) resolutions, to over 128 for high definition (HD) (eg. 1920x1088) resolution. Since full search performs an exhaustive search within the specified search region, it provides the optimal R-D performance for the given search region. However, it is a computationally expensive form of search. For example, with 16x16 macroblocks and a 64x64 search region, this requires over one million pixel absolute difference (AD) accumulations per macroblock. For a 480p30 input (720x480 at 30 frames per second), this involves over 40 billion absolute difference calculations per second. Due to the

high computational demands, some modern multimedia single-instruction multiple-data (SIMD) instructions for ME. One example is the SIMD instructions for ME that are implemented Intel's SSE3 instruction set [2]. Therefore, full search is not suitable for real time software implementations, and can be challenging for dedicated hardware implementations as well.

1.3.4 Fast Motion Estimation (FME)

Equations 1.1 through 1.3 describe the fundamental SAD-minimization ME process. FME algorithms reduce the computations required through a variety of methods. The fundamental idea of FME algorithms is to reduce the search space S , to reduce the number of computations required for the distortion measure, or some combination of the above methods.

Algorithms to reduce the size of the search space include such methods as 2D Logarithmic Search [12], Three Step Search (TSS) [15], New Three Step Search (NTSS) [18], among others. These algorithms work on the assumption of monotonically reduction of the SAD value as the candidate MV approaches the MV with the global minimum SAD. In these algorithms, a set of search points distributed across the overall search space is checked in the first step. The lowest SAD in this step is used as an indicator of the globally minimum SAD (and the best MV), and is used to update the search center for the second step, and so on. Specific examples of these algorithms are described in Chapter 2. This class of algorithm has the problem of missing the global minimum SAD due to following local minima. Also, these searches perform all operations on a full resolution version of the current and reference frames, which can result in high bandwidth for high resolution video.

Reduced SAD complexity algorithms include methods such as pixel subsampling [16]. However, although these schemes provide reduced computational complexity, they can have a negative impact on the overall R-D performance of the coder.

FME algorithms can be seen as a necessary compromise of R-D performance to fit within computational constraints. While full search can provide the optimal R-D performance for a given search range, the ME implementation must be matched to some maximum complexity constraint while still supporting a certain desired resolution and frame rate.

1.3.5 Advanced ME Features

1.3.5.1 Variable Block Size (VBS)

A variable block size (VBS) motion compensation scheme allows for different block sizes in the MCP process. In portions of the frame where there is a finer granularity of motion, this can provide better R-D performance. With one motion vector per smaller block, there can be multiple motion vectors per macroblock.

The estimation algorithm attempts to find the best motion vectors by performing the ME search of different block types. To provide the lowest energy in the residual, the set of block types resulting in the minimum overall SAD for a full 16x16 macroblock will be the one selected. However, VBS coding introduces an additional concern in terms of the rate for coding motion vectors. For example, a 16x16 macroblock coded into 16 4x4 blocks requires 16 times as many motion vectors as a block coded as a single 16x16 macroblock. Therefore, it is common to weigh the search using R-D constraints which favour large blocks, as coding additional vectors adds to the bitrate.

The H.264/AVC standard includes several different block partitions. A 16x16 macroblock can be partitioned into one 16x16 block, two 16x8 blocks, two 8x16 blocks, four 8x8 blocks, eight 8x4 blocks, eight 4x8 blocks, or 16 4x4 blocks. Altogether, this gives 41 different block types for one macroblock. These are shown in Figure 1.3. For each candidate MV position, the pixel SAD calculations that need to be done correspond to the multiplication of the width and the height. For example, for an 8x4 block, 32 pixel SAD calculations must be done. Accounting for all the different block types, we obtain 10496 pixel SAD calculations, a seven-fold increase over the 256 pixel SAD calculations required for single block size ME. However, data reuse in the VBS algorithm can be exploited to reduce the calculations. For example, any 16x16 macroblock requires 256 pixel SAD calculations. Subdividing the 16x16 macroblock gives four 8x8 regions, amounting to four 64 pixel SAD calculations. Each of these SADs corresponds to the 8x8 block SADs for the corresponding candidate motion vector (i.e. the 16x16 SAD for candidate MV (k,j) is comprised of the sum of the four nested 8x8 SADs for candidate MV (k,j)). This makes it natural to calculate VBS SADs through a bottom-up approach: large block SADs are obtained by accumulating the SADs of the smaller blocks nested within. Therefore, though VBS adds an additional computational burden, it can be minimized through appropriate architectural or software design.

VBS coding has also been adopted for the VC-1 video coding standard. Any video encoding system intended to take maximum advantage of the compression capabilities of the H.264/AVC and VC-1 standards will require a ME unit capable of supporting VBSME. Note that VC-1 offers up to four vectors per macroblock (either one vector for the 16x16 or four for each 8x8 sub-block). This is a subset of the block partitions

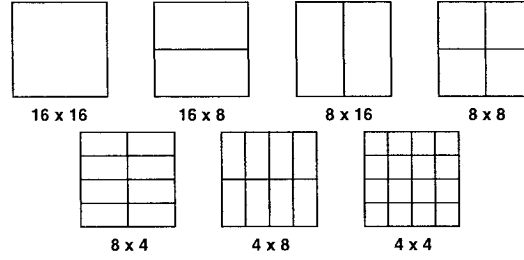


Figure 1.3: Block patterns found in H.264/AVC standard.

offered in H.264/AVC. A VBSME implementation that is designed for H.264/AVC can be equally applied to VC-1, provided that only the common block partitions are used. The MPEG-4 codec also employs VBS coding with block sizes ranging from 16x16 to 4x4, which would be fully supported by the proposed VBSME architectures.

1.3.5.2 Multiple Reference Frames (MRF)

An additional feature in the H.264/AVC standard is the concept of multiple reference frames (MRF). In single reference frame coders, the immediately preceding reference frame is used for predictive coding of P pictures. With multiple reference frames, any of a number of frames selected from a list can be chosen as a reference. This can improve R-D performance in a number of scenarios. First, there may be occlusion of part of the background by an object in the foreground. This occlusion may not be present in a different reference frame. For a noisy video source, there may be less distortion in a previous frame. Also, some objects may be rotating or going through other transformations, such that a better reference can be found beyond the first reference frame. In the absence of fractional-pel ME, it is also possible that occurrence of fractional-pel motions will result in better predictions in reference frames other than the immediately previous one.

For a set of N reference frames, the full search algorithm can be extended to MRF by performing N independent full searches across each of the reference frames. This provides an optimal R-D performance for MRF, similar to how single reference frame full search provides optimal R-D performance for one reference frame. Supporting this requires a linear increase in the computational requirements of a motion estimation architecture. Therefore, supporting MRF requires a much higher throughput ME architecture than single reference frame ME.

1.3.6 Memory Bandwidth

The block matching process used in ME can potentially have a very high memory requirement. A key point however, is that ME is highly regular, and offers many opportunities for data reuse between search points. As an example, the memory requirement for 720x480x30 SD resolution ME is described, with a 32x32 search range. This is equivalent to 40500 macroblocks undergoing the ME process per second, which involves matching against 1024 search points. With no data reuse at all, then at each of these search points, a 16x16 current macroblock and 16x16 reference frame region must be loaded. This “load-always” memory arrangement would necessitate a memory bandwidth of approximately 23 GB/s, which is very high, and is much more than could be supported by a realistic embedded system.

The topic of memory arrangements for ME is well explored in [20]. As described in [20], the regularity of the full search ME process offers a large potential for memory bandwidth reduction through data reuse. Data reuse is achieved by using local storage, rather than fetching all data from off-chip memory. This can be seen as similar to caching as used in general purpose processing, but used for a completely

deterministic algorithm. Due to the determinism of the algorithm, it is possible to achieve large reductions in the overall memory bandwidth of the ME system. The “redundant access” factor Ra is defined in [20], through Equation 1.6, where MA refers to the total number of memory accesses for a pixel, and PC refers to the total number of pixels required for the ME process.

$$Ra = \frac{MA}{PC} \quad (1.6)$$

Ra can be further broken down to make the distinction between the current frame and the previous reference frame, giving $Ra_{current}$ and $Ra_{previous}$, respectively. Using this distinction, we get the Equation 1.7, also from [20]. In the equation, f refers to the frame rate, while W and H give the width and height of the frame, respectively.

$$BW = f * W * H * Ra_{current} + f * W * H * Ra_{previous} \quad (1.7)$$

For ideal data reuse, both $Ra_{current}$ and $Ra_{previous}$ should be 1. In that case, all pixel data for the current and reference frames are only read once from the system bus. Returning to the 720x480x30 case, this would achieve a memory bandwidth of approximately 21 MB/s. This bandwidth requirement is well within the performance of even slow SDRAM systems.

Achieving an $Ra_{current}$ of 1 is relatively straightforward. Assuming that an architecture performs the ME process on one current block at a time, $Ra_{current}$ of 1 is achieved by using a local store of the 16x16 current macroblock until ME on that block is complete. Obtaining $Ra_{previous}$ of 1 is significantly more challenging. In general, a tradeoff must be made between achieving a low $Ra_{previous}$ to achieve a low bandwidth, and controlling the local cache size to reduce total area.

1.4 Optimizations and Parallelism in ME

1.4.1 Overview

VLSI architectures to implement block-based motion estimation have been examined by several sources in literature. In [16], a variety of different architectures for ME are examined. As described in the paper, a hardware ME system can make use of many different forms of parallelism within the block matching algorithm. This section describes the different levels of parallelism, and how they can be implemented. Although this section focuses on SAD-based ME on the luma component only, the concepts can be generalized to other distortion measures, such as mean-square error (MSE), or distortion measures including chroma.

1.4.2 SAD Parallelism

The absolute difference calculation, shown as the inner portion of Equation 1.2 is an easy candidate for parallelism. The absolute difference between pixels can be computed independently. Also, a 16x16 SAD calculation can be segmented into several sums, with each individual sub-sum combined in a tree-like manner. This method is used in many architectures, such as described in [16] among many others. With 16x16 sized macroblocks, this type of parallelism can potentially give a speedup of up to 256 times to the SAD portion of ME. Exploiting this parallelism to a high degree is a key part of many recent high-throughput architectures, such as [14].

1.4.3 Candidate MV Parallelism

As shown in Equation 1.3, the final selection of motion vector is based on a comparison between different candidate MVs. Calculation of candidate MV SADs can be done in parallel. In theory, with an $M \times N$ search area, exploiting this parallelism could give an approximate speedup of MN . Due to the large search area required for most video resolutions, obtaining MN level parallelism at the candidate MV level is not typically feasible while observing power and area constraints. However, this level of parallelism can be exploited to improve the throughput of a system.

1.4.4 VBS Partial SAD Sums

As standards such as H.264/AVC utilize VBS, efficient VBS ME is another source of optimization for an implementation. Efficient re-use of block SADs can greatly reduce the computational requirements of an implementation. As described earlier, architectures can be developed to efficiently reuse the block SADs of small blocks for calculation of the SAD of blocks comprised of those smaller blocks.

1.5 Motivation of Thesis Work

Of the functional blocks contained in a video coder, motion estimation forms the bulk of the overall computations. With the additional burden of VBSME (found in H.264/AVC and VC-1) along with multiple reference frame coding (found in H.264/AVC), the design of a hardware unit for real-time ME can be extremely challenging. The motivation behind this thesis research was to address the issue of VBSME for SD and HD video from both an algorithmic and an architectural viewpoint.

The specific algorithm that is applied for ME is not normative to most video coding standards, which focus on the implementation of the decoder. Since ME is the most computationally demanding portion of the encoder, this offers a great deal of flexibility in creating a tradeoff between computational performance, hardware cost, power consumption, and overall R-D performance of the encoder. The quality of the ME used can have a significant effect on the final R-D performance of the encoder. One of the primary goals for this research was to find or generate an algorithm which provides good R-D performance for VBSME encoding. As will be described in Chapter 2, VBSME presents a unique challenge for fast motion estimation architectures. Due to the large number of block types, it is difficult to effectively support VBSME while still providing the computational savings from FME. Addressing this issue was also a goal when creating an architecture suitable for HD.

Along with providing good R-D performance, part of the thesis research was to create an architecture that is synthesizable for ASIC. The algorithm must be designed with hardware consideration in mind, as the final architecture must be implementable at a low area and power cost, and be able to meet real-time constraints. The work in this thesis targets an architecture that is superior to current VBSME architectures in terms of overall macroblock throughput, as well as being competitive in terms of area and power requirements.

1.6 Thesis Contributions

This thesis examines two different VBSME systems. One is intended for SD, or resolutions at less than SD for multiple reference frames. The second is intended for multiple reference frame HD. The architectures have been designed for the purpose

of creating a high macroblock throughput. The architectures have been designed with support for all the block partitions required in the H.264/AVC standard. By supporting all of the block partitions for H.264/AVC, it is also suitable for standards such as VC-1, which use a subset of the partitions employed in H.264/AVC. Both architectures have been synthesized for ASIC using the CMC Synopsys 0.18 μm process.

The first architecture is designed for full search at SD resolutions. By using full search, this architecture provides a high level of R-D performance. Due to the high macroblock throughput of the design, it is able to meet the real-time constraints of SD video. Additional goals of the architecture were to be competitive or superior to existing integer pel ME architectures in terms of macroblock throughput and hardware area, while still being able to support VBSME. Additional requirements were to have a scalable architecture, such that it could be modified at the implementation level to be usable in a variety of intended applications.

In addition to the SD architecture above, a second goal for the thesis was in the design of a fast search algorithm suitable for multiple reference SD as well as HD. The algorithm that is proposed in this thesis allows for the tradeoff between the number of reference frames and the spatial subsampling that is applied to a reference frame. This form of tradeoff is novel among fast search algorithms found in the literature. The architecture employs a two stage search, where the first stage is on a fixed block size and the second uses VBSME. The final architecture achieves a significantly higher maximum throughput than other HD and MRF architectures proposed in the literature.

Chapter 2

Literature Review

2.1 VLSI Architectures for Full Search

The area of full search architectures for motion estimation has been well studied in literature. [19] is a classic paper in full-search motion estimation. In [19], the problem is explored by a dependency graph analysis. Based on the analysis, the author advocates various different methods of exploiting parallelism in the ME block-matching algorithm, using several different systolic array architectures. Systolic array architectures are attractive due to the architectural simplicity, and relatively straightforward mapping for the full search ME algorithm. [19] also makes an initial analysis of the high memory bandwidth issue inherent to ME algorithms, in particular the very high potential memory demands for full search. The paper explores different mappings of PE definitions, data flow, and local memories for full-search ME. In the PE definitions used in [19], each PE performs an absolute difference (AD) calculation. Different data flow schedules are employed in a variety of 1-D and 2-D architectures in order to calculate minimum SAD values across the entire search area. A 1-D architecture

employs a one dimensional array of PEs to perform calculations. A 2-D architecture employs a 2-D array.

Assignment of different AD calculations to different PEs is very common, and is used in many architectures [21], [14], [23], among many others. However, [16] takes a more general view for the role of the PE. In [16], the authors examine not only the PE in terms of an absolute difference generating unit, but consider applying it for other forms of parallelism, as well. In [16], two major categories of distinction are made between architectures: the type of processing array, and the function of the processing element. For processing arrays, both 1-D and 2-D architectures are considered. [16] categorizes the function of PEs as either AD or SAD units. For AD PEs, the function of an individual PE is to perform the pixel-wise AD calculation, with accumulation into a SAD done elsewhere. For SAD PEs, each PE is given a candidate MV position, and is responsible for the calculation of the SAD for that position. [16] also explores the memory advantages of using systolic 1-D and 2-D architectures with inter-PE communication to avoid high system bus bandwidth.

2.2 Fast Motion Estimation (FME) Algorithms

There are a wealth of papers which describe various means to reduce the complexity of integer pel ME. One of the most common forms of fast motion estimation algorithm is to reduce the number of checking points. Classic versions of this are the 2-D logarithmic search[12] and Three Step Search (TSS) [15]. The three step search is shown graphically in Figure 2.1. In the first step, nine candidate MVs (L0 through L8) are checked. Their vectors are given by the grid $x = [-4, 0, 4]$ and $y = [-4, 0, 4]$. A first stage vector, P1, is selected from this grid based on minimum SAD, at position

$(a1, b1)$. The grid width is halved for the second stage, giving a second stage grid of $x = [a1-2, a1, a1+2]$, $y = [b1-2, b1, b1+2]$. A second stage vector, $P2 = (a2, b2)$, is selected based on the new minimum SAD location in this grid. The final MV selection is made by a one-pel search around the second stage vector. Overall, this search requires nine checking points in the first stage, and eight checking points each in Stage 2 and Stage 3 (assuming the common checking point between stages is not recalculated). This results in a 25 checking point search, in comparison to the 225 checking point search that a full 15×15 search would require. A larger search could be accommodated by using more points in the first stage, adding more stages, or some combination of the two. One flaw with these algorithms is the assumption that the SAD will descend monotonically as the candidate MV approaches the true “best” MV. Since later stages are constrained by the minimum SADs found in earlier stages, it is possible to get trapped in a local minima, while missing the global minimum. An additional issue that these algorithms do not typically address is the high memory bandwidth requirement for HD resolution ME, especially with multiple reference frames. In these algorithms, the entire reference frame (or all reference frames) must be loaded at full resolution. Although only a portion of the reference frame will be required due to the multi-stage algorithm, the selection cannot be known beforehand. Therefore, although these algorithms reduce the computational requirements in terms of the number of checking points, they do not assist in reducing the memory requirement.

An additional type of search which can be seen as a sub-category of the reduced checking point search, is exemplified by the four-step search (4SS). Rather than using a deterministic number of steps, this search relies on a “shifting” window in the direction of the SAD reduction gradient. The four step in this search refers to four

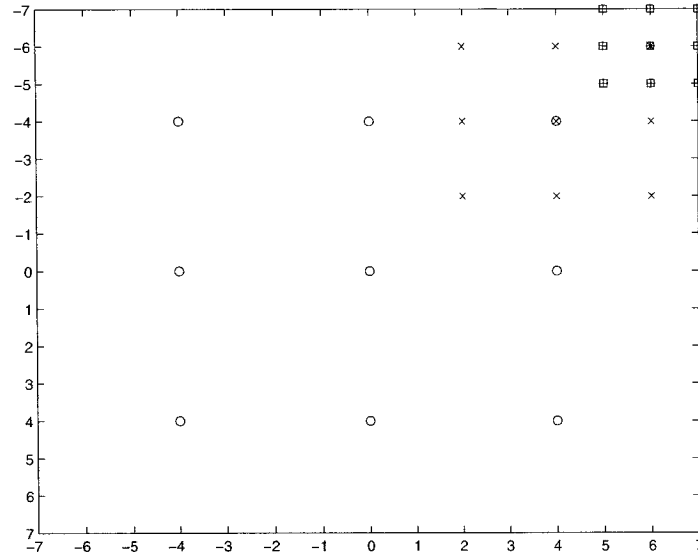


Figure 2.1: Diagram of the three stages of the TSS search.

stages that the search goes through. The first stage is an initial 9-point grid. The second stage is a set of patterns that are used to follow the minimum SAD if it is found at one of the eight edge points of the grid. If the minimum SAD is found at the center point, then the search goes to a final search pattern, with the minimum SAD at that stage selected as the best motion vector. The final MV is found by the vector selected from the part D search. The advantage of this search is that it exploits the fact that there are many stationary regions in most video sequences. In this search, the search will terminate early for smaller motion vectors. For example, if the (0,0) MV has the minimum SAD, then this search will terminate after two search stages. This can result in a great reduction in the overall computational complexity if there is a large search window size. However, this non-deterministic execution can have a negative impact for a hardware realization. The algorithm has an indeterminate number of steps, which can complicate a hardware implementation. In comparison,

searches with a determinate number of hierarchy stages (ie. one stage for full search, three stages for TSS) can be more efficiently implemented and pipelined in hardware. Also, this algorithm can have a poor worst-case performance. As an example, we assume that the true MV for a macroblock should be (7,0). We also assume there is a monotonically decreasing SAD towards (7,0), so the algorithm will not be trapped in a local minima. TSS will uncover the MV (7,0) in three steps ((4,0), (6,0), (7,0)), while 4SS will uncover it in four ((2,0), (4,0), (6,0), (7,0)). This poor worst-case performance can hinder an implementation where we wish to guarantee real-time in all sequences, since it would have to be designed for the worst case.

Another general class of search algorithm are multi-stage algorithms which employ pixel subsampling. In this search, a first stage search is performed on a reduced resolution version of the current macroblock and the reference frame. Based on this search, refinements are made on successively higher resolutions of the frame, typically ending in a final refinement step at the full resolution. This type of search is normally referred to in the literature as a “hierarchical search”. It is referred to in [16]. Pixel subsampling offers to types of computational savings:

- Reduction in the number of pixels per macroblock.
- Reduction in the overall search area.

To demonstrate these computational savings, we take the example of subsampling by two in each dimension. If all the pixels in the current and reference frame are subsampled by two, then “macroblock” becomes represented by an 8x8 block rather than a full 16x16. This reduces the amount of computation required for finding the SAD. In addition, subsampling by two also means that the search area can be

reduced by two in each dimension and maintain the same effective search area. The same concepts can be applied to different subsampling ratios.

Another form of fast search exploits pixel resolution. In conventional video encoders, the encoder operates at 8 bits per pixel (considering the luma portion only). Instead of operating at the full 8 bits of resolution, it is possible to use only some upper k bits of the full 8-bit resolution. Although this does not reduce the number of checking points, nor typically the number of cycles per macroblock, it can have an effect on the overall performance of the encoder. By using smaller-width registers, adders, comparators, and multiplexers, these algorithms reduce the hardware area and the amount of combinatorial logic between pipeline stages. They can also increase the macroblock throughput of the design. This approach is one of the methods used to obtain speedup in [8]. However, employing a reduced pixel resolution has a negative impact on the overall R-D performance of an encoder.

2.2.1 FME for VBSME

A special consideration needs to be made when designing a fast search architecture intended for VBSME. In a typical FME algorithm, a multi-stage search is applied. The results of the first stage are used to constrain the second stage, such that a smaller area than the full search window is used. This results in computational savings over the full search. For the VBSME case, preserving the computational savings over full search requires further investigation.

In the H.264/AVC standard, there are seven block partitions resulting in a total of 41 block types. One potential method to support VBSME is to perform an independent search for each block type, or to perform 41 different searches. For a full search

architecture, the effect of performing 41 independent searches can be duplicated by performing a SAD-reusing search at the 16x16 macroblock size, and preserving the results for the sub-blocks. In this method, there is only an incremental increase in the overall SAD calculation overhead.

The situation is different in a multi-stage search. Consider the TSS algorithm, where nine points are checked in each of three stages. Referring to Figure 2.1, there are nine candidate MVs checked in the first stage. With independent searches, it is possible that each of these points is selected as the candidate MV for at least one of the block types. This results in a grid of checking points formed by $x = -6, -4, -2, 0, 2, 4, 6$ and $y = -6, -4, -2, 0, 2, 4, 6$. This gives a total of 49 different checking points in Stage 2. By comparison, the single block size TSS algorithm had only 9 checking points in Stage 2. In the third stage, up to 41 of these checking points can be followed, which in the worst case can result in a larger total checking points than if full search were applied. To maintain the computational savings of a VBSME algorithm, it is desirable to constrain this tree-like expansion of checking points. This can be done by disallowing certain block types at early stages, to restrict the number of candidate MVs that must be followed in later stages. Different versions of this approach are used in [7] and [17], which are described in Section 2.4.

2.3 VLSI Architectures for Fast Motion Estimation

There exist many architectures to support several of the fast search algorithms found in the literature. [11] describes an architecture for the classic three-step search algorithm. In [11], each PE is assigned to one of nine checking points. The three-step search algorithm requires a sparse checking-point grid in the first step, followed by a denser one in the second step, and an even denser grid in the last step. Since the same PEs are used in each of these steps, [11] addresses the issue of a memory structure to accommodate the different checking point resolution grids.

An architecture employing hierarchical search motion estimation is found in [7]. It is described in more detail in the next section.

2.4 VLSI Architectures for Variable Block Size ME

With the inclusion of VBS ME in standards such as H.264/AVC, there has been a number of papers published which address architectures to support variable block size ME. One early architecture to support searches on variable block sizes is presented in [21]. The architecture in [21] employs a 2-D systolic array architecture with global accumulation. However, it is not optimized to be used for VBSME. The architecture in [21] is primarily designed to support 16x16 macroblocks. The variable block size support that is provided in the architecture is primarily designed to support fast searches where there is a single, smaller block size used at different stages of the hierarchy. Variable block sizes are made possible through the selective masking of some PEs in the 2-D array. For example, to produce results for 8x8 blocks, 3/4 of the

2-D array is deactivated by selective masking. It is not well suited to VBS ME in that it does not support data reuse for multiple block sizes. For example, the 41 different block types for H.264/AVC would require repeated computation of block SADs for each of the different block types. The architecture also has throughput inefficiencies when block sizes smaller than 16x16 are employed. When portions of the PE array are masked, the array utilization and the effectiveness of the architecture is lowered. Due to the selective masking approach to supporting variable block sizes, this architecture is ill-suited to VBSME with a large number of different block types.

The architecture in [9] is more suitable for VBSME. The architecture there is intended for the advanced coding mode of the H.263 standard, which allows for a single macroblock to be coded with four vectors corresponding to four contained sub-blocks. At the high level, this architecture employs a 1-D array of PEs. Each of the PEs is a basic AD computation unit, with one addition for the purposes of VBS ME. Two parallel accumulation channels are found at the output of the PE array: one leading to four different sub-block accumulators selected by multiplexer, and a second going to an accumulator for the overall macroblock. A comparator is used to select either the four sub-blocks or the full macroblock, based on selecting which produces the minimum SAD. If the full block produces the minimum SAD, one MV is produced as the result. If the sub-blocks provide the minimum SAD, four vectors are output. This architecture is closer to supporting VBSME as required by H.264/AVC than [21], but also has limitations. First, there are only two types of block supported, either 16x16 macroblocks or four 8x8 sub-blocks. Therefore, it still lacks many of the block types required for H.264/AVC. It also is lacking in the throughput required for SD resolutions, as it is limited to one pixel pair per cycle, or 256 cycles to produce the

SAD for one candidate MV, per PE. This architecture does however present several useful concepts. The concept of pipelining operation between PEs in the 1-D array, in order to reduce the data transmission requirements is useful in the design of 1-D PE architectures. It is also the first architecture that can truly support VBSME through data-reuse, rather than sequentially repeating the search as in [21].

In [24], an architecture suitable for all the block types found in H.264/AVC is presented. This architecture makes use of a 1-D array of PEs operating in a pipelined manner, similar to [9]. Using accumulation and multiplexing of partial SADs within each PE, the architecture of [24] is able to produce all the necessary block SADs for H.264/AVC. Note that this is done differently than the method used in [9]. The accumulation of sub-block sums in [9] is done using a global accumulation circuit outside of the PE array. In [24], the accumulation of sub-blocks is done internally within a PE. This is in comparison with [9], where accumulation of 8 pixels is made before an output is produced from the PE. With [24], the 41 block types are accumulated internally, and are output onto SAD output buses serially. Minimization circuits on each SAD bus are used to find the minimum SAD and the corresponding vector. The architecture has some limitations, primarily in its overall block throughput. The architecture is designed for scalability, with a flexible 1-D processor element array. However, its design is only suited for CIF resolutions with a relatively small, 16x16 search area at the standard PE array length. To support SD resolutions with a larger search area would require scaling the architecture to have a high area and a large power consumption. Therefore it is not well suited to SD resolutions.

An alternate architecture for VBSME is proposed in [14]. This architecture is a

high-throughput architecture that fully supports VBSME. In the architecture proposed in [14], each PE is a AD computation unit. The results of the AD calculations are passed to an adder tree, which produces outputs for the various block types required of VBSME. This architecture offers a high throughput by a 16x16 2-D systolic array design. By using a 16x16 PE grid, the architecture is able to calculate all the ADs required for a macroblock in a single cycle. The individual ADs that are produced by the PE grid are passed to an SAD accumulation tree, which generates all the relevant sub-block SADs. The architecture is limited in scalability in the sense that the 2-D grid of PEs must be 16x16. It is difficult to improve the throughput, as it is only possible through replicating the entire 16x16 array. It is not possible to achieve a low-area implementation, as any implementation must contain the full 16x16 grid.

In addition to full-search VBS architectures, there have also been some fast search architectures proposed in the literature. One architecture is proposed in [7]. This architecture performs a hierarchical search, while supporting VBSME. In [7], the authors address the difficulty of supporting VBS searches in a multi-stage search. A key issue is in determining which block types will be allowed at which stages. If all block types are searched at all stages, then the size of the total search area can become excessively large. In [7], this is constrained by using a three stage hierarchical search. In the second stage, four vectors corresponding to the four 8x8 sub-blocks of a macroblock are searched. Each of these 8x8 vectors are divided into 4x4s and are searched in the final stage, resulting in a total of 16 possible vectors. This is in comparison to the 41 possible vectors that could be searched in an unconstrained search.

Another fast search VBS architecture is found in [17]. This architecture employs a hierarchical search, using different pixel subsampling levels in each hierarchical stage. At each stage, a full search is employed to find the best match at the current resolution level. The vector at that stage is refined on the next, higher-resolution, hierarchy stage. This architecture that is generated has a very high throughput, and a more detailed comparison is made to it in Chapter 4.

Chapter 3

VBSME Architecture

3.1 Introduction

The goal of the thesis research was to generate an ME architecture suitable for H.264/AVC at SD resolutions and below. To fully support H.264/AVC video, this required support for VBS with a high enough throughput to potentially use multiple reference frames. This chapter describes the proposed architecture, and makes comparisons of the architecture to prior works.

3.2 Architecture

3.2.1 Architecture Fundamentals

Our architecture achieves high throughput through a combination of parallel candidate MV calculation in addition to parallel absolute difference calculation. Candidate MV parallelism is achieved by assigning a separate candidate MV to each processing

element (PE) of an array of N PEs, achieving N -way parallelism at the candidate MV level. Each of these PEs processes M pixel pairs in parallel, where one of each pixel pair is from the current macroblock, and the other pixel is from the reference frame. We have examined designs for both $M=4$ and $M=16$. Variable block size computations make use of partial SAD accumulation, minimizing the overhead required in computing SADs for different block types.

Several design challenges are addressed to achieve a high-throughput VBS architecture that makes use of a high degree of mixed-type parallelism. Efficient transmission of data between PEs is addressed, to avoid creating an architecture with wide internal bus widths for inter-PE communication. The final design employs a combination of a latched delay line and broadcast buses, which are described in detail later in Section 3.3. In conjunction with pipelined PE operation, this allows high throughput (measured in macroblocks per second) while minimizing communication requirements between PEs and the system bus. Pipelining within PEs is also to improve the throughput. In addition, the architecture was designed with a high degree of scalability: the architecture can function with a many different PE array lengths, allowing for a tradeoff between performance and area requirements at the implementation level. To our knowledge, this scalability is a first in the literature for high-throughput architectures.

In the next two sections, the function of a single PE and its underlying hardware structure is described. The section following those gives the interaction between PEs.

3.2.2 Processing Element (PE)

3.2.2.1 Operation

The fundamental PE design that is used is very similar to that used in [24]. In [24], each PE is assigned a different candidate MV. This has some advantages over the typical PE design, where each PE is essentially a SAD computation unit. The main advantage is found in the greater degree of scalability. While a SAD computing array can provide a high throughput, it is difficult to increase the performance through the addition of PEs, or to achieve a low-area design by reducing the number of PEs. By following the candidate MV PE design philosophy of [24], while working to greatly increase the overall throughput, we achieve a scalable architecture that is suitable for SD.

A key limitation to the architecture found in [24] was the lack of pixel-pair parallelism in the architecture. At any one cycle, each PE only receives one current macroblock pixel and one reference frame pixel. Therefore, each PE can perform all calculations for a candidate MV position at a maximum throughput of 256 cycles (for a 16x16 macroblock). Assuming a 720x480x30 video, with a search range of 32x32 and 32 PEs, this architecture would require an operating frequency of over 300 MHz, which could not be achieved by their synthesis results. A longer PE array could be used, but the area costs of such a design could be prohibitive. Therefore, a design using [24] as a basis, while exploiting AD-level parallelism was considered.

The pixels in the architecture in [24] are processed in a raster-scan order. This is depicted in Figure 3.1, where each number on the figure represents the cycle in which is processed by the PE. Both a pixel from the current macroblock and a pixel from the reference block enter on the input pixel bus. 4x4 block SADs are produced on

the cycle immediately after the 4x4 block has fully entered the PE. For example, the first 4x4 block is fully processed after cycle 51, the second 4x4 block is fully processed after cycle 55, and so on. Outputs for combined blocks are made between outputs for the 4x4 blocks. For example, the first 8x4 block SAD is produced at cycle 56. By interleaving the SAD values of combined sub-blocks between those of the 4x4 blocks it is possible to produce variable block size SADs with little extra cycle overhead compared to a single block size architecture. In order to minimize the number of extra cycles required to output these combined blocks, the output SAD values for combined blocks are transmitted “in-between” results for 4x4 blocks. A full schedule of output cycles can be found in [24], but analysis shows that there are many “slack” cycles where no outputs are being produced. For example, in the first 50 cycles of [24], no outputs are being produced from the PE. With a wider input, and more use of pixel AD parallelism, this architecture could provide additional throughput. Therefore, the first step to improve the architecture of [24] was to introduce some pixel level parallelism.

Using [24] as a basis, the decision must be made on how to increase the number of pixel input pairs to improve the throughput. This is essentially a scheduling decision, where it is desired to select at which cycle each pixel pair should arrive at the PE. Possible candidates would be to use a horizontal vector of M pixel pairs or a vertical vector of M pixel pairs. There are a wide variety of other options, for example a set of 4 pixels spaced by 4 every cycle could be used ($\{1,4,8,12\}$ in the first cycle, $\{2,5,9,13\}$ in the second, and so on). The ultimate choice of design must take into account two main factors:

- Adaptability to the PE array structure.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111
112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175
176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207
208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239
240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Figure 3.1: Pixel numbering for architecture in [24].

- Adaptability to the data broadcast and transmission structure.

The first item, adaptability to the PE array structure, refers to how well the pixel schedule could be adapted to internal multiplexing and accumulation within the PEs. A complex pixel schedule could result in a high overall internal multiplexing complexity within the PE in comparison to a more regular schedule. For example, in the original [24] architecture, a switch between 4x4 blocks will occur every four cycles. For example, in cycles 1 to 4, the pixels being accumulated are for the first 4x4 block. In 5 to 8 it is for the second, and so on. Because of the modulo-4 nature of these switches, it is sufficient to make a comparison against the lower two bits of the state counter, so a switch between sub-block accumulation registers is made every time the lower two bits are “00”. To constrain the area and logic complexity of the architecture, it is desirable to have a relatively simple pixel schedule.

The second item, the adaptability to the pixel transmission structure, is also a

major concern. As will be described in detail further in the thesis, the architecture makes use of the adjacency of candidate MV positions to lower the pixel data bandwidth that is required. Each PE operates on a separate candidate MV position, with a PE operating on a candidate MV position horizontally adjacent to the PE before it in the array. To lower pixel bandwidth requirements, it is advantageous for the pixel parallelism to be along an orthogonal dimension to the candidate MV parallelism, for reasons to be described more in detail later in the thesis. The final architecture uses horizontal candidate MV parallelism, while it uses vertical pixel-pair parallelism.

The first architecture that was designed processes four pixel pairs per cycle, in a vertical 1x4 vector, and is referred to as the “1x4” design through the rest of the thesis. As in [24] the architecture proposed here takes advantage of SAD reuse. The SAD value of a block is obtained by summing the SAD values of the smaller blocks that tile the block. All output SADs are formed by the addition of 4x4 sub-block SADs, which are indexed as from left to right, then top to bottom, as shown in Figure 3.2. As in [24], the architecture proposed here takes advantage of SAD reuse. A schedule for the output of the 1x4 design is shown in Table 3.2. Using this design, 4x4 sub-block SADs can be completed every four cycles. This avoids the lengthy idle output period that was found in [24], where no output is produced until the 51st cycle. With this design, the number of cycles to compute and output all 41 SADs is shown to be 70 cycles for the 1x4 design. This allows for a much higher motion vector throughput than attained in [24], which requires 256 cycles per motion vector. Therefore, the 1x4 design offers an overall speedup of 3.65 over [24]. This gain comes with only a moderate increase in complexity for the PE. The specific architectural design is described later in the thesis.

Although the change from a single pixel to a 1x4 design achieved a large speedup, even greater speedup is necessary to support SD resolutions at a reasonable area and operating frequency. Therefore, a 1x16 architecture was designed. The 1x16 architecture is more complex than the 1x4 architecture, in that there are multiple sub-blocks being accumulated in any one cycle. For example, by the fourth cycle, a 4x16 area of the macroblock has been processed, which corresponds to four distinct 4x4 sub-blocks. Therefore, there must be a mechanism to process multiple candidate sub-blocks in parallel within the PE. This is described in the architectural description of the 1x16 PE.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Figure 3.2: 4x4 block SAD numbering.

3.2.2.2 1x4 PE Structure

The structure of the processing elements in our 1x4 and 1x16 implementations are based around the shuffling and accumulation of partial SAD values. The fundamental structure of our 1x4 PE is shown in Figure 3.3, and is similar to the PE structure used in [24]. A fundamental difference is that our architecture makes use of parallel absolute difference computations among multiple pixels. In the 1x4 PE, a set of four

Table 3.1: SAD Output Schedule, 1x16 implementation

Cycle	Sub-PE A	Sub-PE B	Sub-PE C	Sub-PE D
4	0	4	8	12
5	0,4	none	8,12	none
8	1	5	9	13
9	1,5	none	9,13	none
10	0,1	4,5	8,9	12,13
11	0,1,4,5	none	8,9,12,13	none
12	2	6	10	14
13	2,6	none	10,14	none
16	3	7	11	15
17	2,3	6,7	10,11	14,15
18	3,7	2,3,6,7	11,15	10,11,14,15
19	0,1,4,5,8,9,12,13	0,1,2,3,4,5,6,7	8,9,10,11,12,13,14,15	2,3,6,7,10,11,14,15
20	Full MB	none	none	none

Table 3.2: SAD Output Schedule, 1x4 implementation

Cycle	Output	Cycle	Output
4	0	40	9
8	1	41	8,9
9	0,1	44	10
12	2	48	11
16	3	49	10,11
17	2,3	52	12
20	4	53	8,12
21	0,4	56	13
24	5	57	9,13
25	1,5	58	12,13
26	4,5	59	8,9,12,13
27	0,1,4,5	60	14
28	6	61	10,14
29	2,6	62	0,1,4,5,8,9,12,13
32	7	64	15
33	3,7	65	11,15
34	6,7	66	14,15
35	2,3,6,7	67	10,11,14,15
36	8	68	8,9,10,11,12,13,14,15
37	0,1,2,3,4,5,6,7	69	2,3,6,7,10,11,14,15
		70	Full Macroblock

absolute difference computations are done in parallel. To accommodate this, both the current macroblock and reference frame inputs to the PEs must be widened to 32 bits (four pixels) accordingly. In Stage 0, the absolute difference between current macroblock and the corresponding search area pixels are found. Since there are four pairs of pixels per cycle in the 1x4 architecture, there are also two stages of addition to form a 1x4 SAD. As these two stages can be pipelined, they do not impact the overall throughput of the PE.

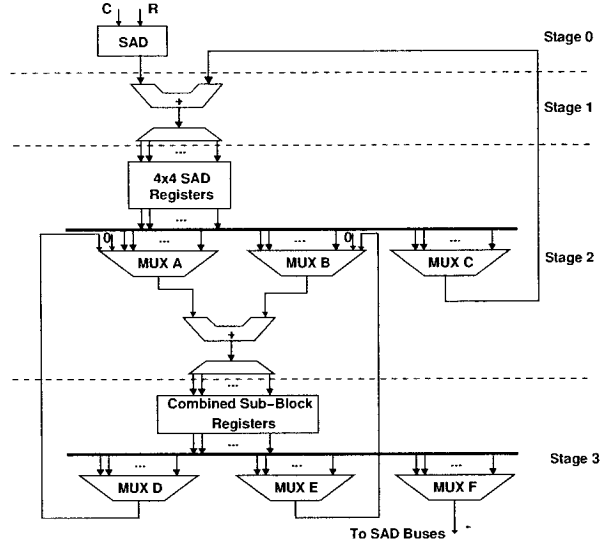


Figure 3.3: 1x4 VBS PE structure.

Using scheduling logic and feedback from Mux C, 1x4 SAD values are combined to form 4x4 block SADs in Stage 1. For the 1x4 implementation, there are eight 12-bit registers. These registers are 12-bit to accommodate the sum of 16 absolute-difference values. These eight registers (shown as 4x4 SAD registers in Figure 3.3) hold the SADs for blocks 0 through 7 for the first half of the processing time. They are reset after the first half of the macroblock has been processed, and hold block

SADs for blocks 8 through 15 in the second half of the macroblock processing cycle.

In Stage 2, block SADs are combined to form the SAD of larger blocks. This is performed by an adder whose inputs are selected based on scheduling logic. The inputs to the adder come from either Stage 2 4x4 block SAD registers, the Stage 3 combined SAD block registers, or are a zero SAD input. The Stage 3 block SAD registers hold SADs for all of the possible block sizes, and therefore must be 16 bits wide. To handle all of the shuffling required, the proposed implementation requires six 16-bit registers.

In Stage 3, values are fed back from the Stage 3 registers to the Stage 2 MUXes, and are fed forward to the SAD bus. MUX control logic used to determine where to forward SAD values is based on a state counter.

3.2.2.3 1x16 PE Structure

Our 1x16 processing element is formed by a set of four sub-PE elements. Individually, these sub-PEs are similar in structure to the 1x4 PE, but with different scheduling logic. Each sub-PE is responsible for handling one quarter of the 4x4 block SADs, plus a subset of the larger blocks. Specific outputs for each of the sub-PE units are shown in Table 3.1. The major difference between these sub-PEs and a full 1x4 PE is the addition of intra-PE buses to communicate between the sub-PE units in Stage 2. This communication is required to distribute SAD values to form results for blocks larger than 4x4. Using the schedule shown in Table 3.1, each sub-PE only needs to transmit at most one of its results per cycle, meaning that each sub-PE only requires one intra-PE bus. The complexity of each of the sub-PE units is less than that of a 1x4 PE; since each sub-PE works on a smaller subset of blocks than a full PE, the

internal MUX logic for stages 2 and 3 is simplified.

A high level view of the intra-PE communication conducted by the sub-PEs is shown in Figure 3.4. For example, as shown in Table 3.1, in cycle 5 sub-PE A is responsible for sending the 8x4 sub-block formed by blocks 0 and 4. Block 0 is stored in sub-PE A, but block 4 is stored in sub-PE B. Therefore, sub-PE B must place the 4x4 SAD value of block 4 on its intra-PE bus on the prior cycle. The placement schedule for the intra-PE buses are shown in Table 3.3.

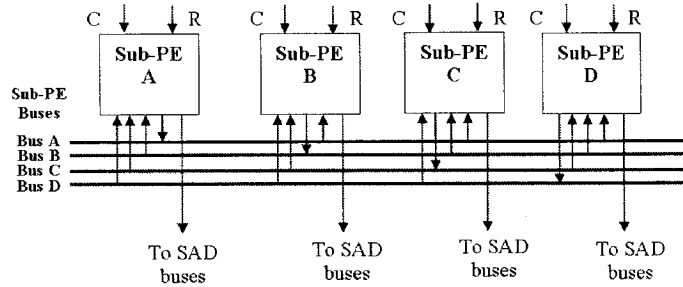


Figure 3.4: 1x16 VBS PE structure.

Table 3.3: Sub-PE Bus Timing

Clock	Bus A	Bus B	Bus C	Bus D
4	-	4	-	12
8	-	5	-	13
10	-	4,5	-	12,13
12	-	6	-	14
17	2,3	7	15	10,11
18	0,1,4,5	2,3,6,7	8,9,12,13	10,11,14,15
19	-	-	-	2,3,6,7,10,11,14,15

As shown in Table 3.1, the 1x16 PE requires 20 cycles to calculate all the necessary

SADs for one macroblock. Therefore, it provides 3.5 times speedup over the 1x4 PE. It may be expected that the area of the 1x16 PE would be four times that of a 1x4 PE. However, the ratio in area between the 1x16 and 1x4 PE is in fact 2.25, as shown in Table 3.6. Given the performance figures shown in Table 3.6, this means that the 1x16 PE is more area-efficient when normalized to the same throughput. There are two main reasons that the 1x16 PE is more area-efficient. Although the sub-PEs in the 1x16 PE are similar in structure to 1x4 PEs, the MUX and control logic are less complex in the sub-PE case. This is due to the reduced number of blocks SADs a sub-PE must calculate. Further savings come from the less than linear increase in the number of registers required between the 1x4 and 1x16 case. Both implementations need to store the same sub-block SADs internally in registers. In the 1x16 case, these sub-block SADs are distributed among all the sub-PEs. Therefore, the storage requirement of one 1x16 PE is less than four times that of four 1x4 PEs.

Table 3.4: Required Frequency of Operation (1x16 architecture)

Total PEs	Res.	Frame Rate	Search Range	Ref. Frames	Freq. (MHz)
16	CIF	30	16x16	1	3.8
16	CIF	30	16x16	5	19.0
16	720x480	30	32x32	1	51.8
16	720x480	30	32x32	2	103.7
32	720x480	30	64x64	1	103.7

3.2.3 Inter-PE Interaction

A high level view of the architecture described in this thesis is shown in Figure 3.5. At the input of the motion estimation unit are a set of current macroblock and reference frame buses, for conveying the current macroblock and reference frame pixel data to a one-dimensional array of PEs (Fig. 3.6). The output from the PE array is a set of sum-of-absolute differences (SAD) values. These output SAD buses are connected to a minimization circuit which determines the minimum SAD through a comparator. The mode selection uses the resultant minimum SAD values to determine the proper coding mode for the macroblock, and forwards the motion vectors to the rest of the coder. For the rest of this thesis, we focus on the design of the PE array.

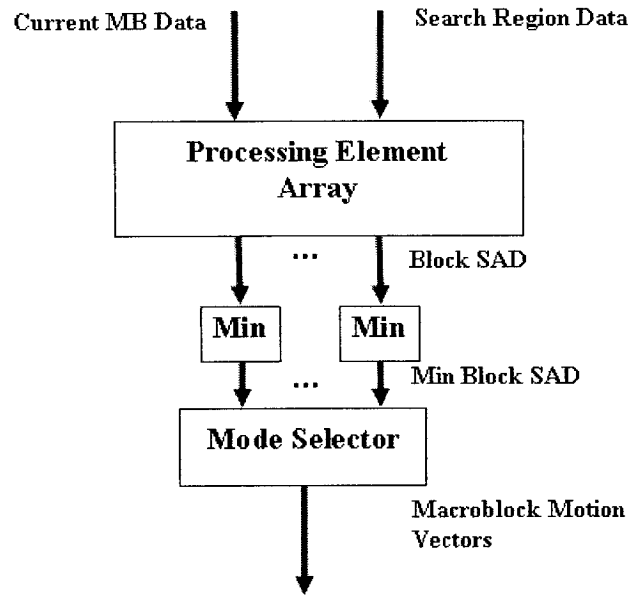


Figure 3.5: High level view of motion estimation architecture.

Figure 3.6 shows the basic design of the 1-D PE array. Each PE is responsible for

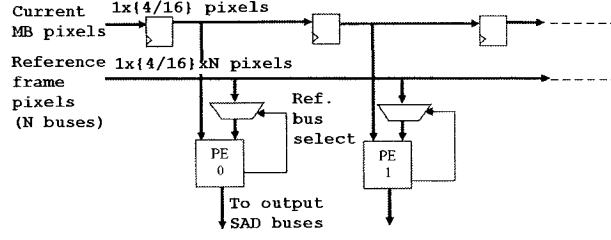


Figure 3.6: Processing element (PE) array.

the calculation of all the SADs of all nested blocks within one macroblock, for one candidate motion vector. The array of PEs allows for parallel computation of the SADs for a row of candidate motion vectors. The 41 SAD values, one for each of the 41 blocks, are computed by one PE for a candidate motion vector value. The number of PEs in the row is also the number of candidate motion vectors for which SADs are computed in parallel. Each PE works on a set of pixels at a time. We have examined both a 4 pixel-parallel implementation and a 16-pixel parallel implementation. From the high level view shown in Figure 3.6, the two designs are almost identical.

Both designs make use of two sets of input pixel data: a set of current macroblock pixel data, and a set of search area pixel data. This data is sent in a left-to-right, top-to-bottom order to the PEs. For our 1x4 architecture, 64 cycles are required to send a 256-pixel macroblock, with 4 vertically aligned pixels per cycle. For our 1x16 architecture, 16 cycles are needed, with 16 vertically aligned pixels per cycle. The process is shown in Figure 3.7. Processing of multiple pixels per cycle is one of the ways in which we obtain the high level of parallelism in our architecture. By processing up to 16 pixel pairs per cycle, we obtain a much higher degree of throughput than in a single pixel pair per cycle architecture such as [24].

The current macroblock pixel data is distributed using a delay line. By contrast,

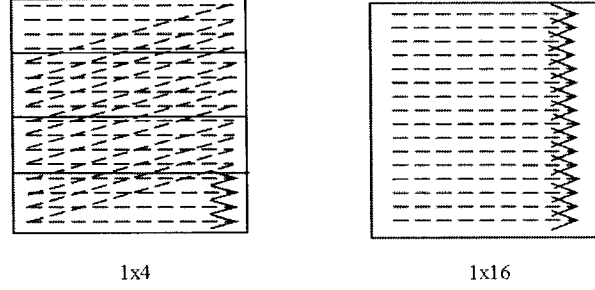


Figure 3.7: Pixel transmission order for 1x4 and 1x16 implementations.

the search area pixels are distributed on a set of shared buses. This design allows for parallel computation amongst the PEs while helping to minimize the amount of pixel I/O to the PE array. For the sake of clarity, the rest of this section will describe the 16 pixel-parallel design.

The schedule for sub-block calculations is shown in Table 3.5. The table shows which sets of current macroblock pixels (C) and reference search area pixels (R) are being used to calculate partial SADs by each PE at any given cycle. As Table 3.5 is for the 1x16 architecture, 16 pixels from each of C and R are compared in every cycle. $C(a, 0:15)$ and $R(b, 0:15)$ refer to vertical columns of pixels located at horizontal position a in the current macroblock and horizontal position b in the reference frame search region. For example, in Cycle 0, $C(0, 0:15)$ and $R(0, 0:15)$ are processed by PE_0 . These represent a column of pixels formed by $x = 0$ and $y = [0:15]$. In cycle 1, $C(1, 0:15)$ and $R(1, 0:15)$ are processed. These represent $x = 1$ and $y = [0:15]$. Note that each PE processes the same current macroblock pixels ($C(0:15, 0:15)$), but each is assigned a different part of the reference frame search area ($R(0:15, 0:15)$, $R(1:16, 0:15)$, $\dots$, $R(15:31, 0:15)$ for 16 PEs).

Each PE in a row requires 16 consecutive input sets ($C(0, 0:15)$ to $C(15, 0:15)$)

Table 3.5: Pixel Data Schedule, 1x16 Architecture, 16 PEs

Cycle	PE ₀	PE ₁	...	PE ₁₅
0	C(0, 0:15), R(0, 0:15)	Previous MB	...	Previous MB
1	C(1, 0:15), R(1, 0:15)	C(0, 0:15), R(1, 0:15)	...	Previous MB
2	C(2, 0:15), R(2, 0:15)	C(1, 0:15), R(2, 0:15)	...	Previous MB
...	...	...	...	...
15	C(15, 0:15), R(15, 0:15)	C(14, 0:15), R(15, 0:15)	...	C(0, 0:15), R(15, 0:15)
16	no input	C(15, 0:15), R(16, 0:15)	...	C(1, 0:15), R(16, 0:15)

from the current pixel delay line. Each of these input sets is a 16-pixel vector, or 128 bits. For PE₀ these input sets are available at clock cycles 0 through 15, for PE₁ these are available at cycles 1 through 16, and so on. Table 3.5 also demonstrates why different means are required to distribute the current macroblock and search area pixel data. The same current macroblock pixel data, offset by one cycle, must be fed to each consecutive PE. This is accomplished by the delay line. Each PE requires a different set of search area pixels, depending on which candidate motion vector the PE is responsible for checking. Due to the cycle offset operation of the PEs, several PEs are operating on the same set of search area pixels during a cycle. This allows for the use of a broadcast bus for the search area pixels, which reduces the I/O and buffering requirement. Note that in cycle 16, PE₀ does not operate on any input pixel data. This is due to the four cycles that are required to combine sub-blocks after receiving all the pixel pairs for a macroblock, which will be described in detail further in the thesis.

Multiple search area buses (SR buses) are required to avoid delays as a PE switches

from one macroblock to the next. The number of buses required is a function of the search range and the number of PEs. As an example, for a search range of 16×16 (corner-to-corner of the search area) with 16 1×16 PEs, two buses will be necessary. For a search range of 32×32 with 32 1×16 PEs, a set of three buses will be necessary. An example bus interleaving schedule for 32 1×16 PEs operating on a 32×32 search area is shown in Figure 3.8.

Note that this scheduling is the reason why the pixel-pair AD parallelism is made along an orthogonal dimension to the candidate MV parallelism. If AD parallelism was along the horizontal dimension, the amount of search area data that needs to be broadcast to each PE would be significantly increased. For example, using horizontal parallelism, PE0 would need a vector formed by $x = [0:15]$, $y = 0$ in Cycle 0, then $x = [0:15]$, $y = 1$ in Cycle 1. PE1, operating at an offset of 1 cycle from PE0, would require the vector formed by $x = [1:15]$, $y = 0$ in Cycle 1. The two PEs no longer require the same data in Cycle 1, increasing the amount of data that must be broadcast.

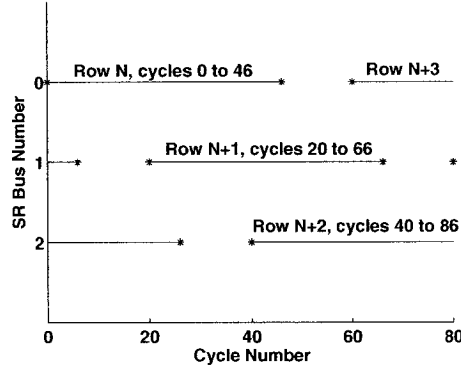


Figure 3.8: Reference frame bus activity for 32 1×16 PEs, 32×32 search area.

Each 1×16 PE requires 20 cycles to generate all block SADs for a candidate MV. Therefore, after every 20 cycles each PE must switch to the next reference frame bus,

cycling back to bus 0 after the last reference frame bus. Using the SR broadcast bus scheme shown in Table 3.5, a total of 47 cycles are required per search region row, to broadcast the 32 different positions of the 16 pixel width macroblock. To accommodate this, we require a minimum of three buses, using the schedule shown in Figure 3.8 for multiplexing between buses. As shown in the figure, the multiplexing logic is relatively simple. Every twenty cycles, each PE switches to the next reference frame bus. In our example 32x32 search region with 32 PEs, the reference frame buses achieve 76.7% utilization. In general, if we assume that an array of N PEs is used for an $N \times N$ search area, the number of reference frame buses is $\lceil (N+15)/20 \rceil$. It is straightforward to scale the length of the PE array and the search range to achieve 100% input bus utilization if desired. For example, if the search range was expanded to 45x45, with a 45 PE array, then all three input buses would be fully utilized. Alternatively, if the search area is scaled down to 25x25 with 25 PEs, then two fully utilized buses would be required.

The output from each PE is a set of 41 SAD values, one for each block. For each block, it is necessary to minimize over the SAD values associated with different motion vector candidate values. These values appear across the PEs at different points in the processing schedule, which is shown in Tables 3.1 and 3.2. Each consecutive PE has a schedule delayed by one cycle from the immediately preceding PE in the array. There are 41 different blocks per PE, so in a simple approach, there can be 41 different SAD buses. A minimization circuit on each of the buses finds the minimum SAD, through use of a comparator.

This high-level design is similar to the one proposed in [24]. However, as attested by performance results in the next section, our architecture allows for a much higher

throughput than in [24]. A major source of the increase in throughput is by making use of parallel absolute difference computations in each of our PEs to obtain a higher degree of parallelism than obtained in [24]. We do this while still being able to perform parallel computations for multiple candidate MV positions. We also obtain a higher scalability and a moderately higher throughput than the architecture shown in [14]. This is obtained by using a flexible 1-D PE array design, which can be scaled to meet the application requirements.

3.3 Architecture Comparison

In this section, the performance of the 1x4 and 1x16 designs are compared to the performance of other VBSME architectures. Our designs were implemented using VHDL, and can therefore be easily synthesized in a variety of fabrication environments and technologies. To characterize our design, synthesis was done for ASIC on TSMC 0.18 μm technology. Details on the synthesis environment are provided in Appendix A. Performance results are summarized in Table 3.6. Results have been normalized to a 16x16 (corner-to-corner) search area to provide a fair comparison between architectures. Gate count results for our 1x4 and 1x16 implementations are formed based on the PE array, and the minimization elements needed for the output bus. The power results are given for a macroblock throughput of 70 thousand macroblocks per second, with a search range of 16x16. This throughput is used because it is the approximate maximum achievable in [24], even though our architecture can achieve a much higher throughput.

As shown in the table, the macroblock MV throughput of our 1x16 implementation is greater than prior works, in particular those proposed in [24] and [23]. The high

Table 3.6: Performance comparison to other VBS designs (Results normalized to 16x16 search range)

	Yap 2004 [24]	Wei 2003 [23]	Kim 2005 [14]	1x4 VBS	1x16 VBS
PE count	16	16	16x16	16	16
Process (μm)	0.13	-	0.18	0.18	0.18
Max freq. (MHz)	294	120	100	155	155
Max MB / sec	71k	29k	390k	138k	484k
Gate count	61k	n/a (FPGA)	132k	69k	155k
Power (mW) (for 70k MB/s)	560	-	-	73	68

macroblock throughput of the 1x16 implementation makes it suitable for standard definition video; 720x480x30 video with a search range of 32x32 and two reference frames can be processed in real-time.

Our implementations are synthesized with a 0.18 μm process, in comparison to the 0.13 μm process employed in [24]. Our architecture was designed without using explicit low-power techniques and was not optimized for power. Despite this, we achieve a much lower power dissipation than that stated for [24] when normalized for the same block throughput. A large contributor to this reduced power consumption is the lower frequency required to achieve a given throughput in our 1x4 and 1x16 VBS architectures. A set of required frequencies of operation for the 1x16 architecture are shown in Table 3.4. As shown in the table, either single or two reference frame SD or multiple reference frame CIF coding is possible with our current architecture.

Moreover, multiple reference frame CIF coding is achievable at a low frequency of operation. This makes the proposed architecture attractive for low power applications.

There is a higher gate count associated with our architecture in comparison to that in [24]. However, the 1x16 implementation provides 6.82 times the maximum throughput of [24], while requiring only 2.54 times the effective gate count. This shows the 1x16 implementation as being much more area efficient than the architecture in [24]. Relative to [14], we see that the 1x16 implementation allows for 1.24 times the maximum throughput, while requiring an increase in area of 1.17. Therefore, we obtain higher throughput with slightly better area efficiency relative to [14].

The proposed architecture is highly adaptable to the processing requirements of targeted H.264/AVC applications. The 1x4 and 1x16 architectures shown in this paper can be scaled by modifying the number of PEs contained in the processing array. This level of fine grain scalability is not found in the architecture of [14], wherein the hardware can be scaled only by integer multiples of the entire 16x16 PE array. The architecture presented here can be scaled by incrementally adding more PEs to accommodate higher search ranges, or by reducing the number of PEs for designs which require a small search range. As described earlier in the thesis, scaling the PE array also entails scaling the number of reference frame input buses.

The architectures proposed here are scalable to higher resolution than those proposed in [24] and [23]. For example, we examine the motion estimation requirements for standard definition NTSC video (720x480 at 30 FPS) with a search range of 32x32. For the architecture in [24], this would require a frequency of operation of 664 MHz when 16 PEs are used, which far exceeds the maximum frequency of 294 MHz that was reported. In our 1x16 architecture, the required frequency is 51.8 MHz for

720x480x30 with a search range of 32x32 and 16 PEs. The 1x16 architecture was synthesized with a maximum possible frequency of 155 MHz, so this is easily achieved. A larger search range is achievable in our proposed architecture by increasing the size of the PE array.

3.4 Local Memory Structure

Although the research for this thesis has focused on the computational side of the problem via the PE array, there are standard local memory structures found in the literature which can be applied to this architecture. Two that can be applied for this architecture is described in principle in [20]. A set of local memory structures for use with full search ME are examined in [20], all of which make use of overlapping pixel regions in the current and reference frames during full search. These architectures are equally applicable to variable block size ME.

Two forms of the locality described in [20] could potentially be used for this architecture, what are classed as “Class C: Global locality within search area strip” or “Class D: Global locality Among Search Area Strips” .. The Class C locality refers to loading a local copy of a horizontal search region strip into a local memory, which is used for processing. This search region strip is made up of one macroblock’s height vertically (16 pixels, or 16 bytes), and covers the whole horizontal search region. As an example, we use a 32-pixel width search region, which would require $32+15 = 47$ pixels in width, or 47 bytes. This means that there is an active buffer of $16 \times 47 = 752$ bytes. Two sets of buffers can be used, one which is being read for the current search area strip being processed, while a second is being filled from the system bus. With the two search region buffers, this would require an overall local buffer of 1502

bytes. With a 32 PE array implementation, each buffer needs to be filled from the system bus at a rate of 752 bytes every 20 cycles, for the 1x16 architecture. With an operation frequency of 150 MHz, this would result in a required memory bandwidth of 5.64 GB/s, which would place a significant burden for standard DRAM architectures.

Since Class C locality does not provide sufficiently low bandwidth, the next option is to use Class D. With Class D, the data common to adjacent search strips is reused. One means to achieve this is to cache the entire search area locally. After each current macroblock in the raster-scan processing order, a new block of width 16 pixels and height equal to the vertical search range must be read. Therefore, the local memory used for buffering should contain the whole search area (47x47 pixels for the specifications above, but can be expanded to 48x48 for simplicity by to make it a multiple of 16) which is used for processing. While the 48x48 memory is being used for processing, another tile of 16x48 pixels is being filled from the system bus. After the current macroblock switches, the active region shifts horizontally by 16 pixels. This leads to an architecture with several 16x48 tiles in this implementation. The structure is shown in Figure 3.9. The figure shows the progress of tiles being in "read" mode, where data is being read by the processing array, and "write" mode, where data is being written to them from the system bus. This form of tiled local memory is similar to the "half search area" structure that is employed in [11]. With a circular buffer type approach, we can achieve a high level of data reuse. Using this approach, each pixel of the search area only needs to be read once per macroblock. The memory bandwidth requirement, using the same assumptions as Class C locality, this results in a system bus bandwidth requirement of 180 MB/s when operating at maximum throughput. Supporting 720x480x30 SD resolution requires a bandwidth

of 31 MB/s which is sufficiently low to be supported in conventional DRAM.

16 x 48 Read x = 0:15	16 x 48 Read x = 16:31	16 x 48 Read x = 32:47	16 x 48 Write x = 48:63
16 x 48 Write x = 64:79	16 x 48 Read x = 16:31	16 x 48 Read x = 32:47	16 x 48 Read x = 48:63
16 x 48 Read x = 64:79	16 x 48 Write x = 80:95	16 x 48 Read x = 32:47	16 x 48 Read x = 48:63

Figure 3.9: Local memory tiles, read versus write mode for three macroblocks.

Chapter 4

HD Resolution VBSME

4.1 VBSME for HD

In the previous chapter, a hardware architecture for supporting VBSME with full search was proposed. The architecture of the previous chapter is suitable for SD resolutions with one reference frame, or sub-SD resolutions with multiple reference frames. However, H.264/AVC is also intended for use at high definition (HD) resolutions. Also, an SD system could benefit from an architecture which is able to support SD resolutions with multiple reference frames.

The architecture of the previous chapter is scalable, and could potentially be used for HD. However, the area and power requirements for HD with multiple reference frames could be prohibitive. Therefore, a fast search implementation is explored, which is able to support VBSME with several reference frames.

4.2 Sub2-1 and Sub4-5

4.2.1 Algorithm Description

To reduce the computational requirements of single reference frame full search, we have examined several fast search algorithms. Search area subsampling schemes [13] function by reducing the number of candidate motion vectors (MVs) that are checked, and include TSS, NTSS, among others. However, these methods do not address the memory bandwidth increase in changing from single frame search to MRF. Although search area subsampling schemes reduce the number of candidate motion vectors that are verified, all decisions are made on full resolution versions of the current and reference frames. For HD resolutions, this can require a high amount of memory bandwidth. An alternate scheme, pixel subsampling (pel subsampling) [16], can reduce the overall memory bandwidth requirement. In these schemes, a search is made on low resolution versions of the current macroblock and the reference frames. The early vectors are refined using higher resolution frames until the final refinement is done at full resolution. These schemes can offer two types of gain: both the search range and block size used for comparison can be reduced in the subsampled space. Another advantage of pel subsampling comes in the hardware implementation. The subsampling can be performed as a preprocessing step. This makes the search in the subsampled space effectively the same as a full search. Full search architectures can exploit a high degree of parallelism, and architectures for full search have been widely explored in literature.

The level of subsampling used impacts the overall R-D performance of the coder, with higher subsampling leading to inferior R-D performance [16]. As shown in [10],

there are video sequences that gain little from MRF coding. Therefore, in these sequences it is not worthwhile to sacrifice the R-D performance of a single frame search in order to perform a multiple frame search. Given a fixed computational budget, the problem can be viewed as a tradeoff between increasing the number of reference frames and decreasing the spatial resolution (increasing the subsampling rate). To accommodate this tradeoff, we introduce a two-stage search algorithm. The tradeoff between subsampling and the number of reference frames is made in the first stage. We have selected two subsampling rates for the algorithm: subsampling by two in each dimension, and subsampling by four in each dimension. It is possible that another implementation could use different subsampling rates, but subsampling by four and by two have been used with good R-D performance in past literature [7], [16], [17]. When the implementation is subsampling by four, we implement a search across multiple reference frames. This makes the choice at the first stage be between the following:

- Search subsampled-by-four, over $k \times k$ range, across N reference frames
- Search subsampled-by-two, over $2k \times 2k$ range, across one reference frame

From this generic description of the algorithm, we create an algorithm with search ranges suitable for HD video. A full search range of $[-64, 63]$ (or 128×128 corner-to-corner) is reasonable for HD video, which results in the search ranges in the subsampled frame given below. The effective first stage search range is $[-64, 63]$ horizontally and vertically at full resolution. This wide search range is applied since the architecture makes the search around the $(0,0)$ vector rather than the predicted motion vector. Searching around the $(0,0)$ vector eliminates the search center dependency on

past macroblocks for the loading of the reference frame data. This allows for a more regular, deterministic memory access pattern which facilitates hardware design. It also allows for superior potential R-D performance than a small search centered at a predicted MV.

We select five reference frames for the subsample-by-four case. This gives the following specific stage 1 definition:

- Search subsampled-by-four, over $[-16, 15]$ range, across five reference frames
- Search subsampled-by-two, over $[-32, 31]$ range, across one reference frame

Note that the first stage search is performed on a fixed block size. As described in Chapter 2, this is desirable to constrain the computational requirements of the second stage. By allowing VBSME in the first stage, the second stage search would be much less regular, leading to a much more complicated hardware design. This would have a negative impact both in terms of macroblock throughput and in area and power of the final implementation. Constraining the first stage to a single block size will have an impact on the R-D performance of the encoder, but it is a necessary step to be able to ensure that the VBSME unit can support HD resolutions with multiple reference frames.

In the second stage, the MV from the first stage is used as the search center to perform a $[-8, 7]$ refinement search at full resolution, using variable block sizes. We call the two search schemes Sub4-5 and Sub2-1. Sub4-5 is the subsampled-by-four five reference frame search, while Sub2-1 is the subsample-by-two one reference frame search.

To mitigate aliasing effects, a 2D Gaussian filter is applied identically to the current and reference frame data prior to decimation. The cutoffs of these filters are

at $1/2$ and $1/4$ the Nyquist rate for Sub2-1 and Sub4-5, respectively. As this can be done in a preprocessing step, we do not consider the complexity of this operation for the evaluation of our VBS ME system. It is possible to obtain a complexity-performance tradeoff for the overall encoding system by changing the filter used for subsampling.

In the above search schemes, the Stage 1 search is performed over a single block size: either 4×4 for Sub4-5 or 8×8 for Sub2-1. In the second stage, VBS SADs are calculated for all the sub blocks of the full resolution 16×16 macroblock. By limiting VBS to the second stage, we obtain a deterministic search time. If VBS were included in the first stage, the search time would be non-deterministic, since disparity between sub block MVs could result in a widening of the Stage 2 search area. In a worst case scenario, the search region required for Stage 2 could potentially approach that of a full search. In addition, implementing VBS search in the first stage of the search would introduce a significant additional burden to hardware implementation, due to the non-deterministic operation. Therefore, using a single block size for Stage 1 ensures a fixed overall throughput for the architecture, and simplifies the overall hardware complexity.

4.2.2 Computational Complexity

To compare the implementation complexity, we compare our algorithms to FS. To estimate the overall computational complexity, we compare the number of pixel absolute difference (AD) operations that need to be performed. For FS across $[-64, 63]$ with 16×16 macroblocks, we require $16 \times 16 \times 128 \times 128 = 4.2$ million AD operations per macroblock, for one reference frame. 26.2 million are required for five reference

frames. In comparison, with Sub2-1, the first stage $[-32, 31]$ with 8×8 sized blocks requires $8 \times 8 \times 64 \times 64 = 0.26$ million AD for the first stage. In the second stage, the $[-8, 7]$ search with 16×16 sized blocks requires $16 \times 16 \times 16 \times 16 = 0.065$ million AD operations. Combined, these two stages make 0.33 million AD operations, which is 12.8 times less overall AD comparisons than the full search with one reference frame, and 64 times less AD comparisons than FS with five reference frames. Using similar calculations, we can show that the Sub4-5 scheme requires 0.15 million AD operations, making it 28.4 time less computationally demanding than FS with 1 reference frame, and 142 times less demanding than FS with 5 reference frames.

Note that these comparisons are only in terms of AD totals. There are also additional overheads incurred from performing VBS search, in terms of combining small sub-block SADs to form the SADs of larger blocks. These overheads favour Sub2-1 and Sub4-5, since they perform VBS only in the smaller second stage search. However, additional overheads are incurred by Sub2-1 and Sub4-5 in terms of decision logic between the first and second stages of the search. However, the overall complexity of the decision logic is far less than required by the SAD calculations. Also, in the comparison above we have considered the computational complexity of the two stages of Sub2-1 and Sub4-5 by summing the complexity of the two stages. However, these two stages can be executed in a pipelined manner, which allows for greater throughput by implementing the two search stages as two separate circuits.

4.3 Algorithm Performance

4.3.1 Sub4-5 and Sub2-1 versus Full Search

In this section, the performance of Sub2-1 and Sub4-5 are compared to the performance of the full search algorithm. Comparisons are made using the Mobile and Football SD sequences, as well as the Foreman CIF sequence. The first 90 frames are used for Mobile, 90 frames of a high-motion high-occlusion portion of the Football sequence are used, and the first 300 frames are used for Foreman. Coding is done using one I frame followed by all P frames. Fixed QPs of 25, 30 and 35 are used to find average PSNR and P-frame bitrates when the different search schemes are applied. In the figure “Full-1” and “Full-5” refer to full search with one and five reference frames, respectively, both using a $[-64, 63]$ search range. “Sub2-1” and “Sub4-5” refer to the proposed search schemes described earlier.

As shown in Figure 4.1, the Mobile sequence gains significantly from MRF coding. There is a dramatic reduction in bitrate between the full search with one reference frame and that with five reference frames. The subsampling performed in Sub2-1 does not seem to significantly impact the R-D performance with respect to Full-1. The subsampling by four does cause a noticeable R-D impact for Sub4-5 with respect to Full-5. However, the R-D performance of Sub4-5 is still significantly superior to either Sub2-1 or Full-1 for this sequence. Therefore using Sub4-5 we are able to achieve superior R-D performance to Full-1 while still requiring significantly less computational complexity. The performance of Sub4-5 in this sequence is approaching that of Full-5, while requiring drastically less computational complexity.

The coding results for the 90 frames of the Football sequence are shown in Figure

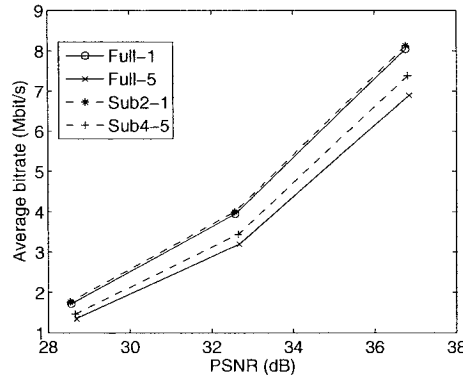


Figure 4.1: Mobile 720x576 at 30 FPS encoded using different search schemes.

4.2. In this sequence, there is a less significant gain from MRF than Mobile. In this sequence, Sub2-1 provides better performance than Sub4-5. This is due to the limited gains from MRF coding being less than the R-D penalty associated with Sub4-5's high subsampling ratio. For this sequence, Sub2-1 is the superior fast search method.

Figure 4.3 shows a significant R-D gain from MRF coding for full search. However, we see that both Sub2-1 and Sub4-5 have performance slightly inferior to that of the single reference frame full search, Full-1. However, as will be shown in the next section, allowing for frame-level switching between Sub2-1 and Sub4-5 can result in significant R-D performance gains for this sequence.

4.3.2 Sub4-5 versus Sub2-1

In the previous section, encoding an entire sequence using either Sub2-1 or Sub4-5 was considered. However, the best coding method may not be fixed for a video stream due to “non-stationary” characteristics of the source data. Therefore, the best performance may be achieved by switching between Sub2-1 and Sub4-5 based

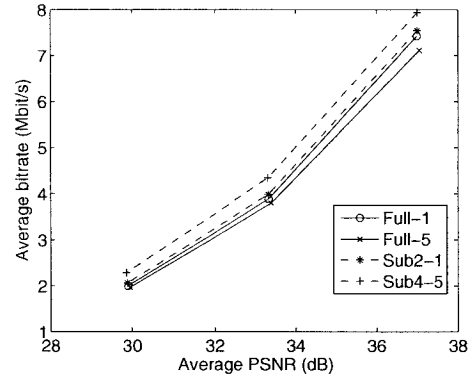


Figure 4.2: Football 720x480 at 30 FPS encoded using different search schemes.

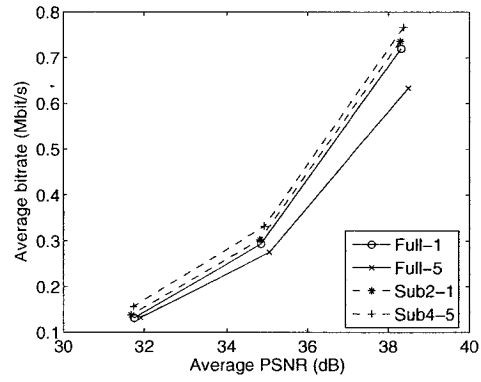


Figure 4.3: Foreman 352x288 at 30 FPS encoded using different search schemes.

on the source video stream being encoded.

The Foreman sequence is an example of a “non-stationary” video sequence. The video can be decomposed into a talking head sequence, followed by a rapid pan, then a shaky camera viewing a still background. We use the Foreman sequence encoded at a fixed QP of 30 for evaluating frame-level switching between Sub4-5 and Sub2-1. In Figure 4.3, it was shown that Sub2-1 provided better performance than Sub4-5 when applied to the whole sequence. However, there are some gains possible from MRF coding in this sequence, as can be seen by the R-D performance difference between Full-1 and Full-5. Viewing coding statistics, we find 89 frames where Sub2-1 provides lower bitrate at higher PSNR, and 65 frames for Sub4-5. Therefore, there are 154 frames where either Sub2-1 or Sub4-5 are clearly superior. The remaining frames offer some R-D tradeoff between the two search methods.

As a view of possible gains from frame-level switching, Table 4.1 shows statistics for encoding the sequence with Sub2-1, Sub4-5, and two theoretical switching algorithms: `low_bitrate` and `high_PSNR`. In these two theoretical algorithms, we make the assumption that we know which method will provide the lowest bitrate or highest PSNR prior to encoding. Also, these statistics are taken from the Sub2-1-only and Sub4-5-only encoding statistics, rather than an encoded stream using either `low_bitrate` and `high_PSNR`, so the figures provided are only an estimate. However, these statistics can give a guideline on some best case possible gains achievable by frame-level switching between search algorithms. As shown in the table, the `low_bitrate` selection scheme gives the lowest estimated bitrate while having an estimated PSNR between Sub2-1 and Sub4-5. The overall estimated R-D performance is slightly better than the single reference frame full search. Similarly, the `high_PSNR` scheme gives the

Table 4.1: Search Schemes with and Without Frame Level Switching Foreman 352x288 at 30 FPS, QP 30

Search Scheme	Bitrate (kbit/s)	PSNR (dB)
Full-1	293.4	34.85
Full-5	275.4	35.06
Sub2-1	302.8	34.82
Sub4-5	332.1	34.93
low_bitrate	290.2	34.89
high_PSNR	305.7	34.97

highest PSNR, while having a bitrate almost as low as purely using Sub2-1.

As will be shown later, the Sub2-1 search scheme incurs a higher computational time in its first stage than Sub4-5. Therefore, the a frame-level switching system can also be used to balance between the overall ME macroblock throughput and the R-D quality of the output video.

4.4 Implementation

4.4.1 Architecture Overview

An implementation of the Sub2-1/Sub4-5 architecture was synthesized using Synopsys tools with a TSMC 0.18 micron process. A block diagram of the implementation is shown in Figure 4.4. The Stage 1 and Stage 2 processing element (PE) arrays act in parallel in a pipelined manner. While the Stage 2 PE array is calculating the necessary SADs for macroblock n , the Stage 1 PE array is calculating the SADs for macroblock $n+1$. The architecture was synthesized for 32 PEs in Stage 1 and 32 PEs in Stage 2. An alternate implementation could use 16 PEs in both stages to reduce the overall area of the design at the cost of a reduced overall throughput.

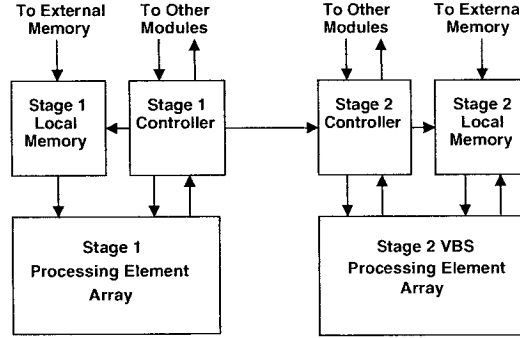


Figure 4.4: High Level view of proposed architecture.

The Stage 1 1-D PE array functions in two modes of operation: Sub2-1 and Sub4-5. In both of these modes of operation, each PE calculates a set of single block size SADs after a fixed number of cycles, and returns the lowest SAD of the set. In Sub2-1 mode, each PE calculates and compares two 8x8 block SADs every 8 cycles. The lowest SAD among these two candidates is output, along with a vector identifying the output. In Sub4-5 mode, four 4x4 block SADs are calculated and compared every 4 cycles. Again, the lowest SAD and its corresponding vector are output. A minimization circuit external to the PE array finds the lowest SAD in the search region, and selects the Stage 2 search center corresponding to the vector. A diagram of the Stage 1 PE is shown in Figure 4.5.

These two modes of operation yield an effective PE throughput of one candidate MV checked every 4 cycles for Sub2-1 and every cycle for Sub4-5. In Sub2-1 operation, there are 64x64x1 candidate MVs to be considered per macroblock. In Sub4-5 operation, there are 32x32x5 candidate MVs to be considered. Therefore, using a 32 PE array, Sub2-1 operation requires 512 cycles per macroblock, while Sub4-5 operation requires 160 cycles.

The Stage-2 1-D PE array is based on the architecture presented in [22]. This array

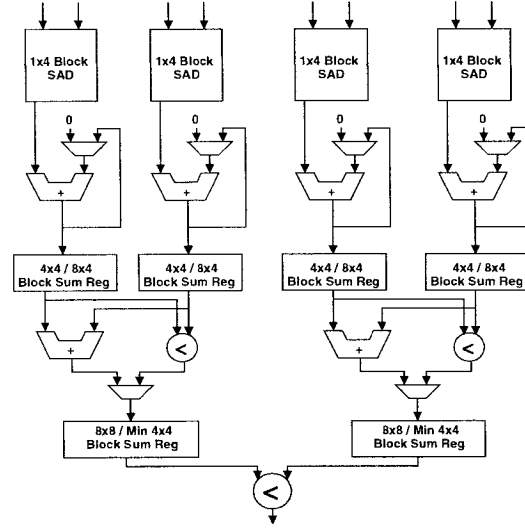


Figure 4.5: Structure of Stage 1 PE.

performs a $[-8, 7]$ search around the center selected in Stage 1. Each PE requires 20 cycles to produce the VBS SADs for a candidate MV. Therefore, with 32 PEs and a 16×16 search range, the second stage requires 160 cycles per macroblock. With both the Stage 1 and Stage 2 PEs operating in a pipelined fashion, the architecture has an overall throughput of 160 cycles per macroblock while operating in Sub4-5 mode, and 512 cycles while operating in Sub2-1 mode.

In addition to the PE arrays, there are controllers for Stage 1 and Stage 2. These controllers select the proper vectors of pixels from memory to be distributed to the different stages, send control signals to the PE arrays as necessary, and perform the minimization of the SAD values returned by the PEs. The Stage 1 controller is also responsible for transmitting the search location to the Stage 2 controller based on the minimum SAD found in the Stage 1 search. The Stage 2 controller is responsible for outputting the minimum SAD values for all the VBS sub-blocks after the search is

complete.

4.4.2 Synthesis Results

The PE arrays and controllers were synthesized using Synopsys Design Compiler. PE arrays and controllers were synthesized, giving a total gate count of 556k gates. The gate count does not include Sub2-1/Sub4-5 mode decision logic or the local memory. The maximum operating frequency achieved was 155 MHz, giving an overall macroblock throughput of 969k macroblocks/s when operating in Sub4-5 mode, and 303k macroblocks/s when operating in Sub2-1 mode. In the worst case for throughput, where Sub2-1 operation is always used, the architecture can support 720p60 (1280x720x60) which contains 216k macroblocks/s. Worst-case all Sub2-1 operation also will support the popular 1080p30 format (1920x1080x30). H.264/AVC profile level 5.0 [3], which requires coding of up to 589824 macroblocks/s can be supported with some restrictions on the number of frames Sub2-1 is applied to encode, as 589k macroblocks/s falls between the minimum and maximum throughput that the architecture can achieve at its top operating frequency.

A reduced version of this proposed architecture, using 16 PE arrays for Stage 1 and Stage 2, could support up a maximum of 242k macroblocks per second in best case operation. This is suitable for 720p60 video when exclusively Sub4-5 operation is employed. This reduced architecture can support 720p30 in worst case Sub2-1 operation. The estimated gate count for this reduced architecture would be 271k gates.

Employing search area buffers in local memory [20], the regularity of the full search algorithm is exploited so only a portion of the search area pixels need to be read from

memory for each new macroblock. This makes use of the overlapping of adjacent search regions to lower the overall external memory bandwidth.

4.4.3 Comparison

Other fast search architectures have been proposed to support HD resolution VBSME. Results are summarized in Table 4.2. Architectures are listed based on fast search method, effective search range, gate count, operating frequency, and overall macroblock throughput. Note that worst case and best cases are included for macroblock throughput. In our proposed architecture, best case throughput is achieved when Sub4-5 mode is used exclusively, while worst case throughput occurs when Sub2-1 mode is used exclusively. For the best case one reference frame throughput, the result represents Sub4-5 being used with only one reference frame in the first stage. In this case, the second stage of the pipeline is the critical stage, so the overall throughput is the same as Sub4-5 in five reference frame operation. The power figure stated is an approximation of the power required for 1080p30 operation using Sub4-5 mode only. The power takes uses the power dissipation for the two processing arrays, along with an approximation of the power dissipation for stage controllers and the minimization circuit. It does not account for the power dissipation of the local memory.

A 720p architecture is presented in [8]. The architecture provides sufficiently high throughput for HD, but does not provide a high enough macroblock throughput for MRF HD. The architecture uses a combination of methods to reduce computational demands, including searching around a predicted MV center, using only the 5 most significant bits of each 8-bit pixel for computation, and a subsampling factor of two

is applied to the reference frame data. Using their fast search, their architecture is able to support 720p30 with one reference frame with a 330k gate design operating at 108 MHz. In the worst case operation of our architecture, 720p30 with one reference frame is supported in a 556k gate design operating at 55 MHz, approximately 35% of the maximum frequency the architecture has achieved in synthesis. Exact R-D performance comparisons of the fast search algorithms are not possible due to differences in test sequences. Their architecture offers a lower degree of subsampling than ours ($1/2$ rather than $1/4$ or $1/16$). However a second-stage full-resolution correction search is not included in [8]. The calculations in [8] are made using only 5 bits of the full 8-bit pixel bit width. In addition, a predicted MV is used to search only a portion of the overall search range in [8]. In sequences where there is a significant R-D gain achievable by MRF coding, it is expected that the architecture proposed here will outperform [8]. The superior search algorithm is not clear for sequences where MRF does not provide a benefit, and Sub2-1 coding is primarily used.

Another fast search architecture is proposed in [17]. In this architecture, a three stage hierarchical pel subsampling search algorithm is employed, Subsampling of $1/4$ and $1/2$ along both axes are used for the first and second hierarchy stages, and full resolution used in the third stage. In Table 4.2, the subsampled hierarchy stages are referred to as $1/16$ and $1/4$ 2-D subsampling, which represents the effective number of pixels used. A multiple survivor search is used for the second stage. The two lowest SAD candidates from the first stage in addition to a predicted center based on a median of surrounding macroblock MVs are used as search centers. The architecture in [17] was synthesized to operate at a frequency of up to 54 MHz, where it can support 480p30 with up to 7 reference frames. This is equivalent to 720p30 at 2.6

Table 4.2: Architecture Comparison

Architecture	Chen '06 [8]	Lee '04 [17]	Proposed
Subsampling (2-D)	1/2	1/16 and 1/4	1/16 or 1/4
Hierarchy Levels	1	3	2
Pixel Resolution	5-bit	8-bit	8-bit
Search Range (h x v)	128x64	128x64	128x128
Gate Count	330k	-	556k
Operating Frequency	108 MHz	54 MHz	155 MHz
Macroblock throughput/s (worst case, 1 ref. frame)	108k	284k	303k
Macroblock throughput/s (best case, 1 ref. frame)	108k	284k	969k
Macroblock throughput/s (worst case, 5 ref. frames)	21.6k	56.7k	303k
Macroblock throughput/s (best case, 5 ref. frames)	21.6k	56.7k	969k
Power Dissipation (mW) (1080p30, 5 ref. frames)			320

reference frames. This block throughput is slightly less than the worst case of the architecture proposed here, and is 3.4 times lower than the best case throughput of our proposed architecture.

4.5 Future Improvements

4.5.1 Overview

This thesis presents a high performance integer-pel VBSME architecture. It is well suited to being the core MV search in an encoding architecture. However, there are some features which were not considered, which could make for a more powerful ME

system. These are described in the sections to follow.

4.5.2 R-D Operation

The distortion measure that is being used in the current architecture is the SAD. This is standard in the literature, but it does not take into account the bitrate cost of the MV side information. In the H.264/AVC standard, these MVs are coded predictively using either variable length codewords or arithmetic coding. MVs which are farther away from the prediction receive longer codewords than those that match the prediction. Because of this, not all MVs are “equal” in terms of bitrate, even if they produce the same SAD. In the H.264 reference code [5], this is accounted for using an R-D offset on the SAD which favours vectors closer to the prediction. Unfortunately, the predicted motion vector depends on knowing the motion vectors that were selected for all previously coded neighbouring macroblocks. This would introduce an additional dependency to the search, where data from previous MV search decisions would have to be stored.

A simplified, though less R-D optimal approach would be to weight MV SADs based on their distance from the (0,0) vector. This could be accomplished by some logic and communication between PEs, such that they can add an additional weight to the SAD based on the current absolute MV position.

4.5.3 Fractional Pel ME

The architecture described is for integer pel ME. Fractional pel ME is typically done as an interpolation step following integer pel ME. This could be potentially implemented as another pipeline stage in the PE. However, this would necessarily increase the

amount of input pixels required per PE, as they would require adjacent pixel values for calculating interpolation.

Chapter 5

Conclusions

This thesis has examined the design of an architecture for high-throughput full-search variable block size motion estimation. Several issues are addressed in the research for this thesis. The first issue addressed in this architecture is to create a design to achieve a high overall throughput without degrading rate-distortion performance. To achieve optimal R-D performance, the architecture employs the full search. Despite the high computational requirements of full search, the architecture proposed can offer a throughput which is sufficient for SD, or multiple reference frame CIF.

The architecture described in this thesis was designed to support all of the block sizes required by the H.264/AVC standard. The architecture described in this thesis supports all of the 41 block types required for the H.264/AVC implementation of variable block size motion compensation.

The architecture described in this thesis achieves a very high throughput, making it suitable for SD video. The 1-D processing element array design of the architecture gives it a high degree of scalability, making it adaptable to a range of application requirements.

Also proposed this thesis is a fast search VBSME architecture which is suitable for MRF and HD video. The search algorithm that is employed is a two stage hierarchical search, where the first stage involves a search over either $[-16, 15]$ in five subsampled-by-four reference frames (Sub4-5), or a $[-32, 31]$ search in one subsampled-by-two frame (Sub2-1). The second stage of the search is $[-8, 7]$ in a full resolution frame. By allowing for a tradeoff between Sub2-1 and Sub4-5, the architecture proposed here supports both sequences where there are significant R-D gains from MRF, and those where there are not.

An architecture to implement the proposed algorithm was synthesized using Synopsys with a 0.18 micron process. The synthesized architecture requires 556k gates and allows for a macroblock throughput between 303k-969k macroblocks per second.

Bibliography

- [1] ISO/IEC 13818:2000 Information technology – generic coding of moving pictures and associated audio information, 2000.
- [2] Block-matching in motion estimation using streaming SIMD extensions 3, 2003.
- [3] ITU-T Rec. H.264 (ISO/IEC 14496-10 AVC), 2003.
- [4] ISO/IEC 14496:2004 Information technology – coding of audio-visual objects, 2004.
- [5] H.264/AVC software coordination (JM reference software), 2006.
- [6] SMPTE 421M-2006 television VC-1 compressed video bitstream format and decoding process, 2006.
- [7] H.F. Ates and Y. Altunbasak. SAD reuse in hierarchical motion estimation for the H.264 encoder. In *Proceedings, IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2005)*.

- [8] Ching-Yeh Chen, Shao-Yi Chien, Yu-Wen Huang, Tung-Chien Chen, Tu-Chih Wang, and Liang-Gee Chen. Analysis and architecture design of variable block-size motion estimation for H.264/AVC. *IEEE Trans. Circuits Syst. I*, 53:578–593, February 2006.
- [9] G. Fujita, T. Onoye, and I. Shirakawa. A new motion estimation core dedicated to H.263 video coding. In *Proceedings of 1997 IEEE International Symposium on Circuits and Systems (ISCAS '97)*.
- [10] Y.-W. Huang, B.-Y. Hsieh, S.-Y. Chien, S.-Y. Ma, and L.-G. Chen. Analysis and reduction of multiple reference frames motion estimation in H.264/AVC. *IEEE Trans. Circuits Syst. Video Technol.*, 16:507–522, April 2006.
- [11] Her-Ming Jong, Liang-Gee, and Tzi-Dar Chiueh. Parallel architectures for 3-step hierarchical search block-matching algorithm. *IEEE Trans. Circuits Syst. Video Technol.*, 4:407–416, August 1994.
- [12] Jain J.R. and Jain A.K. Displacement measurement and its application in inter-frame image coding. *IEEE Trans. Commun.*, 29:1799–1808, December 1981.
- [13] Jong-Nam Kim and Tae-Sun Choi. A fast three-step search algorithm with minimum checking points unimodal error surface assumption. *IEEE Trans. Consumer Electron.*, 44:638–648, August 1998.
- [14] Minh Kim, Ingu Hwang, and Soo-Ik Chae. A fast VLSI architecture for full-search variable block size motion estimation in MPEG-4 AVC/H.264. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC 2005)*.

- [15] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro. Motion-compensated interframe coding for video conferencing. In *Proceedings, NTC '81*.
- [16] P.M. Kuhn. Fast MPEG-4 motion estimation: Processor based and flexible VLSI implementations. *J. VLSI Signal Processing Syst. Signal, Image, Video Technology*, 23:67–92, October 1999.
- [17] Jae Hun Lee and Nam Suk Lee. Variable block size motion estimation algorithm and its hardware architecture for H.264/AVC. In *Proceedings of 2004 IEEE International Symposium on Circuits and Systems*, pages 741–744, Vancouver, Canada, May 2004.
- [18] Renxiang Li, Bing Zeng, and Ming L. Liou. A new three-step search algorithm for block motion estimation. *IEEE Trans. Circuits Syst. Video Technol.*, 4:438–442, August 1994.
- [19] P. Pirsch, N. Demassieux, and W. Gehrke. VLSI architectures for video compression-a survey. *Proc. IEEE*, 83:220–246, February 1995.
- [20] Jen-Chieh Tuan, Tian-Sheuan Chang, and Chein-Wei Jen. On the data reuse and memory bandwidth analysis for full-search block-matching VLSI architecture. *IEEE Trans. Circuits Syst. Video Technol.*, 12:61–72, January 2002.
- [21] L. De Vos and M. Schobinger. VLSI architecture for a flexible block matching processor. *IEEE Trans. Circuits Syst. Video Technol.*, 5:417–428, October 1995.

- [22] Stephen Warrington, Wai-Yip Chan, and Subramania Sudharsanan. Scalable high-throughput architecture for H.264/AVC variable block size motion estimation. In *Proceedings of 2006 IEEE International Symposium on Circuits and Systems*, pages 3830–3833, Kos, Greece, May 2006.
- [23] Cao Wei and Mao Zhi Gan. A novel SAD computing hardware architecture for variable-size block motion estimation and its implementation with FPGA. In *Proc. 5th International Conference on ASIC, 2003*.
- [24] Swee Yeow Yap and John V. McCanny. A VLSI architecture for variable block size video motion estimation. *IEEE Trans. Circuits Syst. II*, 51:384–389, July 2004.