MOTION ESTIMATION BASED
FRAME RATE CONVERSION HARDWARE DESIGNS

by
ÖZGÜR TAŞDİZEN

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Sabancı University
June 2010

MOTION ESTIMATION BASED
FRAME RATE CONVERSION HARDWARE DESIGNS

APPROVED BY:

Assist. Prof. Dr. İlker Hamzaoğlu …….

(Dissertation Supervisor)

Prof. Dr. Aytül Erçil …….

Assoc. Prof. Dr. Erkay Savaş …….

Assist. Prof. Dr. H. Fatih Uğurdağ …….

Assist. Prof. Dr. Hakan Erdoğan …….

DATE OF APPROVAL: …….

To my Family

MOTION ESTIMATION BASED
FRAME RATE CONVERSION HARDWARE DESIGNS

Özgür Taşdizen

EECS, PhD Thesis, 2010

Thesis Supervisor: Assist. Prof. Dr. İlker Hamzaoğlu

Keywords: Frame Rate Up-Conversion, Hardware Implementation,
Motion Estimation, Video Enhancement

ABSTRACT

Frame Rate Up-Conversion (FRC) is the conversion of a lower frame rate video signal to a higher frame rate video signal. FRC algorithms using Motion Estimation (ME) obtain better quality results. Among the block matching ME algorithms, Full Search (FS) achieves the best performance since it searches all search locations in a given search range. However, its computational complexity, especially for the recently available High Definition (HD) video formats, is very high. Therefore, in this thesis, we proposed new ME algorithms for real-time processing of HD video and designed efficient hardware architectures for implementing these ME algorithms. These algorithms perform very close to FS by searching much fewer search locations than FS algorithm. We implemented the proposed hardware architectures in VHDL and mapped them to a Xilinx FPGA.

ME for FRC requires finding the true motion among consecutive frames. In order to find the true motion, Vector Median Filter (VMF) is used to smooth the motion vector field obtained by block matching ME. However, VMFs are difficult to implement in real-time due to their high computational complexity. Therefore, in this thesis, we proposed several techniques to reduce the computational complexity of VMFs by using data reuse methodology and by exploiting the spatial correlations in the vector field. In addition, we designed an efficient VMF hardware including the computation reduction techniques exploiting the spatial correlations in the motion vector field. We implemented the proposed hardware architecture in Verilog and mapped it to a Xilinx FPGA.

ME based FRC requires interpolation of frames using the motion vectors found by ME. Frame interpolation algorithms also have high computational complexity. Therefore, in this thesis, we proposed a low cost hardware architecture for real-time implementation of frame interpolation algorithms. The proposed hardware architecture is reconfigurable and it allows adaptive selection of frame interpolation algorithms for each Macroblock. We implemented the proposed hardware architecture in VHDL and mapped it to a low cost Xilinx FPGA.

HAREKET TAHMİNİ TABANLI ÇERÇEVE HIZI YÜKSELTME DONANIMLARI TASARIMI

Özgür Taşdizen

MDBF, Doktora Tezi, 2010

Tez Danışmanı: Yrd. Doç. Dr. İlker Hamzaoğlu

Anahtar Kelimeler: Çerçeve Hızı Yükseltme, Hareket Tahmini, Donanım Gerçekleme, Video İyileştirme

ÖZET

Çerçeve hızı yükseltme, düşük çerçeve hızına sahip bir videonun daha yüksek çerçeve hızına sahip bir videoya dönüştürülmesidir. Hareket tahmini tabanlı çerçeve hızı yükseltme algoritmaları yüksek kaliteli sonuçlar elde etmektedirler. Arama alanındaki bütün arama noktalarını aradığı icin blok eşleştirmeli hareket tahmini algoritmaları arasında en iyi başarımı gösteren tam arama algoritmasıdır. Ancak, tam arama algoritmasının gerektirdiği işlem miktarı özellikle günümüzde yaygınlaşan yüksek tanımlı video çerçeveleri için çok yüksektir. Bu nedenle, bu tezde yüksek tanımlı video çerçevelerinin gerçek zamanlı işlenebilmesi için hareket tahmini algoritmaları ve bu hareket tahmini algoritmalarını etkin bir şekilde gerçekleştirebilecek donanım mimarileri önerdik. Bu algoritmalar tam arama algoritmasından çok daha az arama noktasını arayarak tam arama algoritmasına çok yakın sonuç elde etmektedirler. Önerilen donanım mimarilerini VHDL ile sahada programlanabilen kapı dizilerinde gerçekledik.

Çerçeve hızı yükseltme için yapılan hareket tahmininin ardışık çerçeveler arasındaki gerçek hareketi bulması gereklidir. Ardışık çerçeveler arasındaki gerçek hareketi bulabilmek için blok eşleştirmeli hareket tahmininin elde ettiği hareket vektörü alanı vektör ortanca süzgeci kullanılarak düzeltilir. Ancak, vektör ortanca süzgeçlerinin gerçek zamanda gerçeklenmeleri gerektirdikleri yüksek işlem miktarı nedeniyle zordur. Bu yüzden, bu tezde veri tekrar kullanımı yöntemiyle ve vektör alanındaki benzerliklerin incelenmesiyle vektör ortanca süzgeçlerinin gerektirdikleri işlem miktarını azaltan teknikler önerdik. Ayrıca, vektör alanındaki benzerliklerin incelenmesiyle işlem miktarını azaltan tekniği de gerçekleyen etkin bir vektör ortanca süzgeci donanımı tasarlayıp sahada programlanabilen kapı dizilerinde gerçekledik.

Hareket tahmini tabanlı çerçeve hızı yükseltme hareket vektörlerini kullanarak yeni çerçevelerin sentezlenmesini gerektirmektedir. Çerçeve sentezleme algoritmaları da yüksek miktarda işlem gerektirmektedirler. Bu yüzden, bu tezde çerçeve sentezleme algoritmalarının gerçek zamanda gerçeklenmelerini sağlayacak düşük maliyetli uyarlanır bir donanım mimarisi önerdik. Önerilen donanım mimarisi her blok için farklı bir çerçeve sentezleme algoritması kullanabilmektedir. Önerilen donanım mimarisini VDHL ile düşük maliyetli sahada programlanabilen kapı dizilerinde gerçekledik.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Dr. İlker Hamzaoğlu. I appreciate his guidance during this thesis.

I want to thank Dr. H. Fatih Ugurdağ as well for his help and valuable feedback throughout the thesis.

I am grateful to Halil Kükner and Abdülkadir Akın for their significant contributions. We worked together for almost one year and for the first time I had the chance to lead a team.

I want to thank TÜBİTAK for their support to my thesis.

My special thanks go to Gülin Alkan.

Finally, I would like to thank to my family for their unlimited support and trust in me, which made everything possible for me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

2D-LOGS	Two Dimensional Logarithmic Search
3D-RS	Three Dimensional Recursive Search
ADCS	Adaptive Dual Cross Search
AMCI	Adaptive Motion Compensated Interpolation
APDS	Adaptive Predicted Direction Search
ARPS	Adaptive Rood Pattern Search
ASIC	Application Specific Integrated Circuit
ASIP	Application Specific Instruction Set Processor
ASNMV	Average Spatially Neighboring Motion Vector
BBGDS	Block Based Gradient Descent Search
BDM	Block Distortion Measure
BM	Block Matching
BRAM	Block Random Access Memory
CIF	Common Intermediate Format
CLB	Configurable Logic Block
CMF	Cascaded Median Filtering
DCS	Dual Cross Search
DMF	Dynamic Median Filtering
DS	Diamond Search
DVD	Digital Versatile Disc
DVSS	Dynamically Variable Step Search
FD	Frame Distance
FPGA	Field Programmable Gate Array
FPS	Frames per Second
FRC	Frame Rate Up-Conversion
FS	Full Search
FSS	Four Step Search
FTS	Flexible Triangle Search
GOPS	Giga Operations per Second
HD	High Definition
HDTV	High Definition Television
HEXBS	Hexagon Based Search

LCD	Liquid Crystal Display
LDSP	Large Diamond Search Pattern
LI	Linear Interpolation
LNMV	Left Neighboring Motion Vector
LUT	Look-Up Table
MAD	Mean Absolute Difference
MB	Macroblock
MC	Motion Compensation
MCA	Motion Compensated Averaging
ME	Motion Estimation
MPEG	Motion Picture Experts Group
MSE	Mean Square Error
MV	Motion Vector
MVF	Motion Vector Field
NTSS	New Three Step Search
PE	Processing Element
PSNR	Peak Signal-to-Noise Ratio
RAM	Random Access Memory
RDVSS	Recursive Dynamically Variable Step Search
RGB	Red Green Blue color space
RTL	Register Transfer Level
TSS	Three Step Search
SAD	Sum of Absolute Differences
SAMND	Sum of Minimum Neighboring Absolute Differences
SD	Spatial Distance
SDSP	Small Diamond Search Pattern
SMF	Static Median Filtering
SND	Sum of Neighboring Differences
SS	Soft Switching
ST	Sum of Absolute Differences Threshold
TD	Spatial Distance
VHDL	Very High Speed Integrated Circuit Hardware Description Language
VMF	Vector Median Filter
WAMCI	Weighted Adaptive Motion Compensated Interpolation

CHAPTER 1

INTRODUCTION

Frame Rate Up-Conversion (FRC) is the conversion of a lower frame rate video signal to a higher frame rate video signal. FRC is used in many devices like televisions, Digital Versatile Disc (DVD) players, portable DVD players, and mobile phones [1]. Recent Liquid Crystal Display (LCD) panels have a frame rate up to 240 Hz, whereas movies are usually recorded at 24 Hz, 25 Hz or 30 Hz and the broadcasted video material is either 50 Hz or 60 Hz. Since the input source and the display have different frame rates, conversion between the received input signal and the output signal sent to the display is necessary. FRC can be done by interpolating a new frame between every two consecutive original frames like in 25 Hz to 50 Hz conversion, and it can be done by interpolating three new frames between every two consecutive original frames like in 25 Hz to 100 Hz conversion. FRC for 1:4 conversion ratio is illustrated in Figure 1.1. In this figure, F(t-1), F(t), F(t+1) are the original frames and the dashed frames are the interpolated frames.

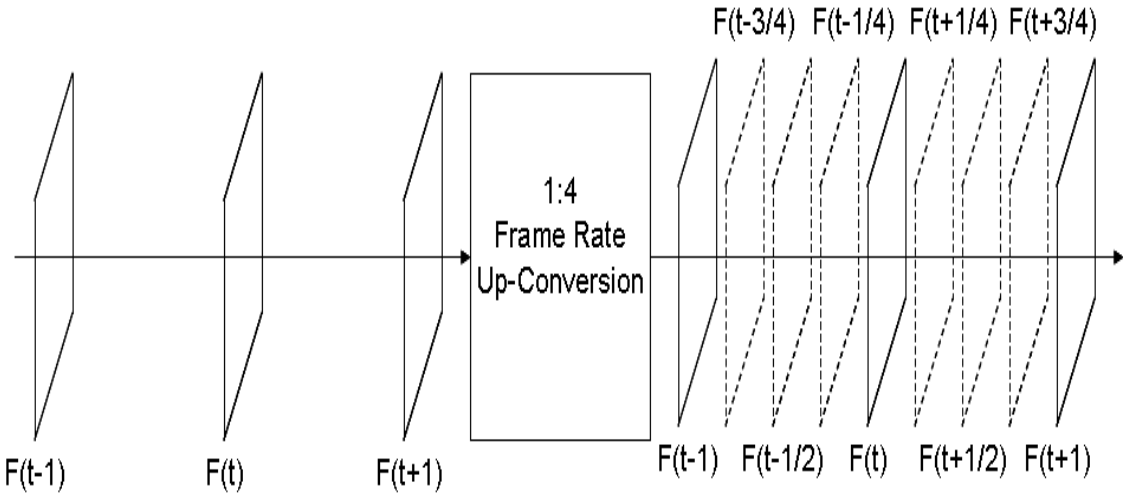


Figure 1.1 The FRC process

FRC can be implemented with simple interpolation techniques or it can be implemented with Motion Estimation (ME) based techniques which require more hardware resources [1]. The quality of the displayed video depends on the performance of the FRC. FRC implemented by simple techniques degrades the quality by creating motion judder and motion blur effects which are the results of the sample and hold nature of the displays [2]. ME based FRC is necessary in order to overcome these artifacts. ME is computationally the most intensive part of video compression and video enhancement systems [3, 4]. Among the Block Matching (BM) ME algorithms, Full Search (FS) achieves the best performance since it searches all search locations in a given search range. However, its computational complexity, especially for the recently available High Definition Television (HDTV) video formats (1920x1080 pixels), is very high, while the Peak Signal-to-Noise-Ratio (PSNR) obtained by fast search algorithms is low.

ME for FRC requires finding the true motion among consecutive frames. In order to find the true motion, Vector Median Filter (VMF) is used to smooth the Motion Vector Field (MVF) obtained by BM ME. The output of the VMF is chosen as the vector that minimizes the sum of distances to all the other vectors [5]. If the current MV, which is in the middle of the VMF window, is not correlated with its neighboring MVs, then the current MV will be replaced with the output of the VMF. However, VMFs are difficult to implement in real-time due to their high computational complexity [6]. ME based FRC requires interpolation of frames using the motion vectors found by ME. Frame interpolation algorithms also have high computational complexity.

Therefore, in this thesis, we proposed new ME algorithms for real-time processing of HD video and designed efficient hardware architectures for implementing these ME algorithms. These algorithms perform very close to FS by searching much fewer search locations than the FS algorithm. In addition, we proposed several techniques to reduce the computational complexity of VMFs by using data reuse methodology and by exploiting the spatial correlations in the vector field. In addition, we designed an efficient VMF hardware including the computation reduction techniques exploiting the spatial correlations in the motion vector field. Finally, we proposed a low cost hardware

architecture for real-time implementation of frame interpolation algorithms. The proposed hardware architecture is reconfigurable and it allows adaptive selection of frame interpolation algorithms for each Macroblock (MB).

We first proposed a ME algorithm, which is a generalization of the Hexagon-Based Search (HEXBS) algorithm, and two hardware architectures for its implementation [7]. These architectures are named as the generic architecture and the systolic architecture. The simulation results showed that the Mean Absolute Difference (MAD) performances obtained by the proposed HEXBS algorithm are better than the MAD performances obtained by other fast search algorithms. Both hardware architectures are implemented in Very High Speed Integrated Circuit Hardware Description Language (VHDL). They can run at 144 MHz on a Xilinx XC3S1200E-5 FPGA and process 25 1920x1080 frames per second (fps) for a (±32,±16) pixel search range. Various fast search algorithms can be implemented using the generic hardware architecture. The main disadvantage of the generic architecture is that it uses 80 Block Random Access Memories (BRAMs). The systolic architecture is designed to efficiently implement proposed HEXBS algorithm. The systolic architecture uses only 16 Block RAMs. A novel data-reuse method is used in this architecture to reduce the number of internal memory accesses, and it has a low control overhead because of its regular data flow.

We proposed Dynamically Variable Step Search (DVSS) ME algorithm and a reconfigurable systolic ME hardware architecture for its implementation [8, 9]. This architecture is implemented in VHDL and mapped to a Xilinx XC3S1200E-5 FPGA. We then proposed Recursive Dynamically Variable Step Search (RDVSS) ME algorithm [10]. The proposed DVSS and RDVSS algorithms work on a search range of (±48, ±24) and (±64, ±64) pixels, respectively. An early search termination mechanism based on a Sum of Absolute Differences (SAD) threshold is implemented in these algorithms in order to trade off speed and quality. DVSS algorithm implemented by the proposed reconfigurable systolic ME hardware architecture requires 467 clock cycles to find the Motion Vector (MV) of a 16x16 MB on the average when the early search termination threshold is set to 256. For this threshold value, the proposed hardware on the average can process 34.3 HD fps. The FS algorithm checks 16641 search locations in a search range of (±64, ±64) pixels, whereas the RDVSS algorithm on the average

checks only 418 search locations when the early search termination threshold is set to 1024. On the other hand, MAD performance of the RDVSS algorithm on the average is only 14.7% lower than MAD performance of the FS algorithm when the early search termination threshold is set to 256. Performing that close to the FS algorithm for such a large search range is very important.

We proposed two techniques to reduce the computational complexity of 1-norm VMF for FRC by using data reuse methodology and by exploiting spatial correlations in the MVF [11]. Since 3x3 window size is used in FRC papers in the literature, we also used this window size. However, the proposed techniques are scalable to any window size. Data reuse technique stores the sum of 1-norm distances between the vectors in a filtering window and uses them for the next filtering window instead of computing them again. The spatial correlations based techniques check the spatial correlations between neighboring MVs and avoid calculating the previously calculated values again. In addition, we proposed an efficient VMF hardware architecture implementing the proposed computation reduction techniques exploiting the spatial correlations in the MVF. To the best of our knowledge, a VMF hardware implementing these techniques is not presented in the literature. The proposed hardware is implemented for a 3x3 window size, but it is scalable to any window size. The proposed hardware is implemented in Verilog HDL, and mapped to a low cost Xilinx XC3S400A-5 FPGA. It consumes 1426 slices and works at 145 MHz. It can process more than 94 HD fps.

We finally proposed a low cost reconfigurable hardware architecture for the interpolation of HD frames [12]. The proposed hardware architecture implements Linear Interpolation (LI), Static Median Filtering (SMF), Dynamic Median Filtering (DMF), Soft Switching (SS) and Cascaded Median Filtering (CMF) frame interpolation algorithms and it allows adaptive selection of these algorithms for each MB. This hardware architecture is implemented in VHDL and mapped to a low cost Xilinx XC3SD3400A-4 FPGA. The implementation results show that the proposed hardware can run at 101 MHz on this FPGA and it consumes 32 BRAMs and 15592 slices.

The rest of this thesis is organized as follows. Chapter 2 explains FS ME algorithm and various fast search ME algorithms. Chapter 3 explains proposed HEXBS ME algorithm, and the generic and systolic hardware architectures proposed for its

implementation. Chapter 4 explains proposed DVSS and RDVSS ME algorithms, and the proposed reconfigurable systolic hardware architecture. Chapter 5 explains VMFs, the proposed techniques to reduce their computational complexity, and the proposed VMF hardware architecture. Chapter 6 presents the proposed hardware architecture for frame interpolation. Chapter 7 concludes the thesis.

CHAPTER 2

MOTION ESTIMATION ALGORITHMS

ME is the part that has the highest computational complexity in video compression and video enhancement systems. ME is used to reduce the bit-rate in video compression systems by exploiting the temporal redundancy between successive frames, and it is used to enhance the quality of displayed images in video enhancement systems by extracting the true motion information. ME is used in video compression standards such as ITU-T H.261/263/264 and ISO MPEG-1/2/4 [3,4], and in video enhancement algorithms such as FRC, de-interlacing, de-noising and super resolution.

ME examines the movement of objects in an image sequence to obtain MVs representing the estimated motion [3,4]. Many different ME techniques are proposed in the literature. These techniques can be categorized as pixel based ME, object based ME, and block based ME. Pixel based techniques require very high computational complexity and they are not suitable for real-time applications. Object based techniques reduce the computational complexity significantly but they cannot obtain high quality results. Block based ME uses BM which is suitable for hardware implementation and can obtain high quality results. Therefore, BM is the most preferred technique.

BM partitions current frame into non-overlapping NxN rectangular blocks and tries to find a block from a reference frame in a given search range that best matches the current block with respect to a Block Distortion Measure (BDM) [3,4]. SAD is the most preferred BDM because of its suitability for hardware implementation. An SAD value is computed with three operations; difference, absolute value, and addition. For NxN block size, the SAD value of a search location defined by the MV $d(d_x,d_y)$ is calculated as in (2.1), where $c(x,y)$ and $r(x,y)$ represent current and reference frames, respectively.

The coordinates *(i,j)* denote the offset locations of current and reference blocks. Since a MV shows the relative motion of the current block in the reference frame, MVs are specified in relative coordinates. If the location of the best matching block in the reference frame is *(x+u, y+v)*, then the corresponding MV is *(u,v)*. Figure 2.1 shows the BM ME process and Figure 2.2 shows a BM ME example and the resulting MVF.

$$SAD(\vec{d}) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\left|c(x+i, y+j) - r(x+i+d_x, y+j+d_y)\right| \tag{2.1}$$

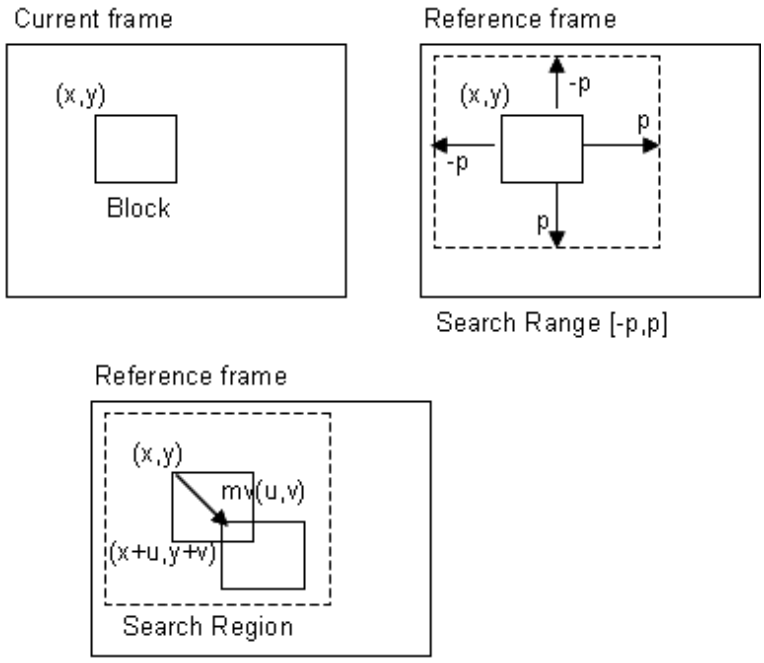


Figure 2.1 BM ME

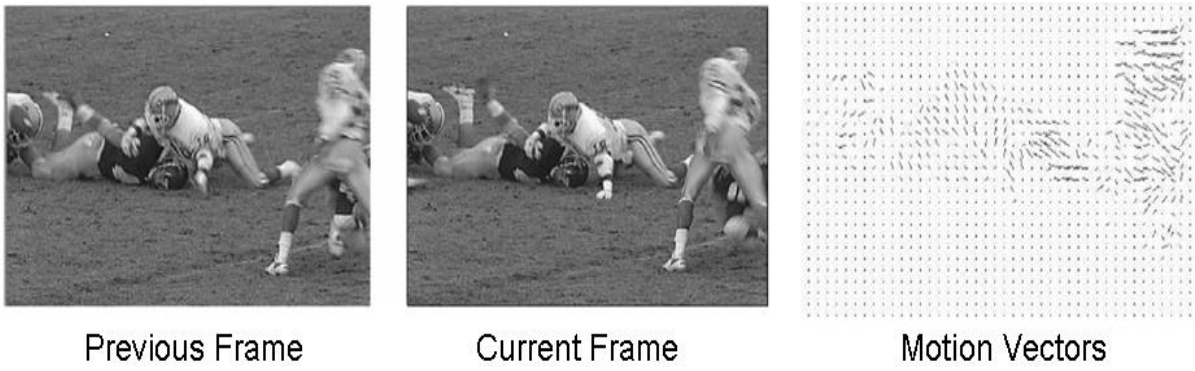


Figure 2.2 A BM ME example and the resulting MVF

In ME, there is a tradeoff between the number of search locations searched and the resulting PSNR. The other two commonly used quality metrics are MAD and Mean

Square Error (MSE). The formulas used to calculate the MAD, MSE, and PSNR are given in (2.2), (2.3), and (2.4), respectively. In these equations, the coordinates *(u,v)* denote the x and y components of the MV.

$$\text{MAD (u,v)} = \frac{1}{N^2}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1}\left|C(x,y) - R(x+u, y+v)\right| \tag{2.2}$$

$$\text{MSE (u,v)} = \frac{1}{N^2}\sum_{y=0}^{N-1}\sum_{x=0}^{N-1}\left[C(x,y) - R(x+u, y+v)\right]^2 \tag{2.3}$$

$$\text{PSNR (u,v)} = 10\log_{10}\left[\frac{255^2}{MSE}\right] \tag{2.4}$$

The FS algorithm gives the best PSNR results, because it finds the reference block that best matches the current block by computing the SAD values for all search locations in a given search range. The computational complexity of the FS algorithm is very high, especially for the recently available consumer electronic devices such as HD digital video broadcasting and high resolution & high frame rate flat panel displays. Because of the large frame sizes in these applications, there are large motions between successive frames and this requires a larger search range to find the best MV.

Several fast search ME algorithms are developed for low bit-rate applications like video conferencing and video phone, which use small frame sizes and require small search ranges. These algorithms try to approach the PSNR of the FS algorithm by computing the SAD values for fewer search locations in a given search range. The most successful fast search ME algorithms are Three Step Search (TSS) [13], Two Dimensional Logarithmic Search (2D-LOGS) [14], New Three Step Search (NTSS) [15], Four Step Search (FSS) [16], Block-Based Gradient Descent Search (BBGDS) [17], Diamond Search (DS) [18], HEXBS [19], Adaptive Rood Pattern Search (ARPS) [20], Adaptive Dual Cross Search (ADCS) [21] and Flexible Triangle Search (FTS) [22].

Fast search ME algorithms perform very well for low bit-rate applications such as video phone and video conferencing [23]. In most of the low bit-rate videos, fast and complex movements are seldom, and nearly 80% of the blocks can be regarded as stationary or quasi-stationary, therefore most of the MVs can be found in a search range of (±5,±5) pixels. However, fast search ME algorithms do not produce satisfactory results for the recently available consumer electronic devices such as HD digital video broadcasting and high resolution & high frame rate flat panel displays, because of the larger movements between successive frames in these videos.

ME for FRC requires finding the true motion among consecutive frames. The true motion is the projection of the physical three dimensional motion on to the two dimensional image space. In order to minimize the amount of information to be transmitted, block based video coding standards encode the displaced difference block instead of the original block. Although BM ME algorithms finding the minimal residue are good at removing temporal redundancies, they are not sufficient alone for finding the true motion.

2.1 The Full Search Algorithm

Since the FS algorithm computes the SAD value for each search location in the search range, it is computationally the most expensive BM ME algorithm. There are $(2p + 1)^2$ search locations in a (±p, ±p) search window. Figure 2.3 shows the search locations of the FS algorithm for (±4, ±4) search range. For this search range, there are $(2x4 + 1)^2 = 81$ search locations. Calculating the SAD value for a search location for an MxN MB requires $(2p+1)^2$ x MN x 3 operations. The operations per second required for calculating the SAD values for an IxJ frame size and an F fps frame rate is given in (2.5). For a 16x16 MB size, 1920x1080 pixels frame size, and 25 fps frame rate, the FS algorithm requires 34.99 GOPS (Giga Operations Per Second) and 149.45 GOPS when p is equal to 7 and 15, respectively.

$$\frac{I\,J\,F}{M\,N}(2p+1)^2 x\,M\,N\,x\,3 \qquad (2.5)$$

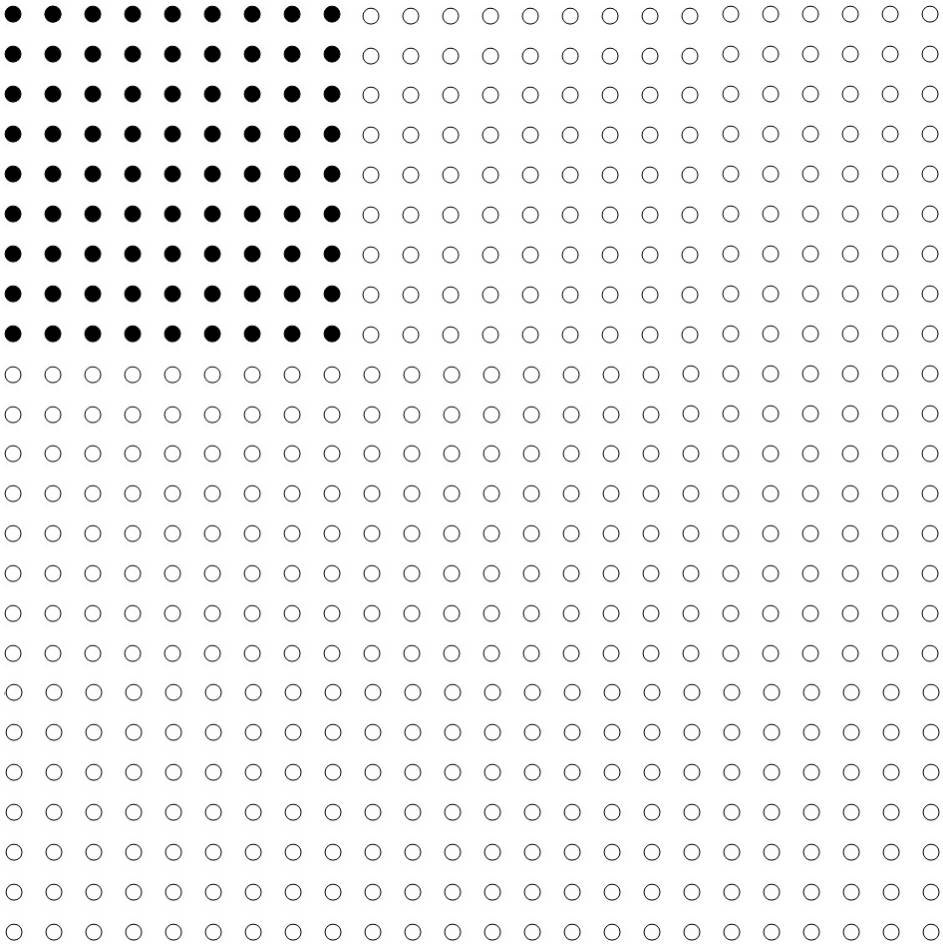


Figure 2.3 Search locations of the FS algorithm for (±4, ±4) search range

2.2 Fast Search Motion Estimation Algorithms

TSS is one of the oldest fast search ME algorithms [13]. As shown in Figure 2.4, TSS searches the best MV in a coarse to fine search pattern. In the first step, nine search locations including the origin are evaluated and the search location giving the minimum SAD is selected as the center of the next search step. In the second step, the distance between search locations is reduced by half. The third step searches the area centered at the location giving the minimum SAD in the second step and the distance between search locations is shortened by half again.

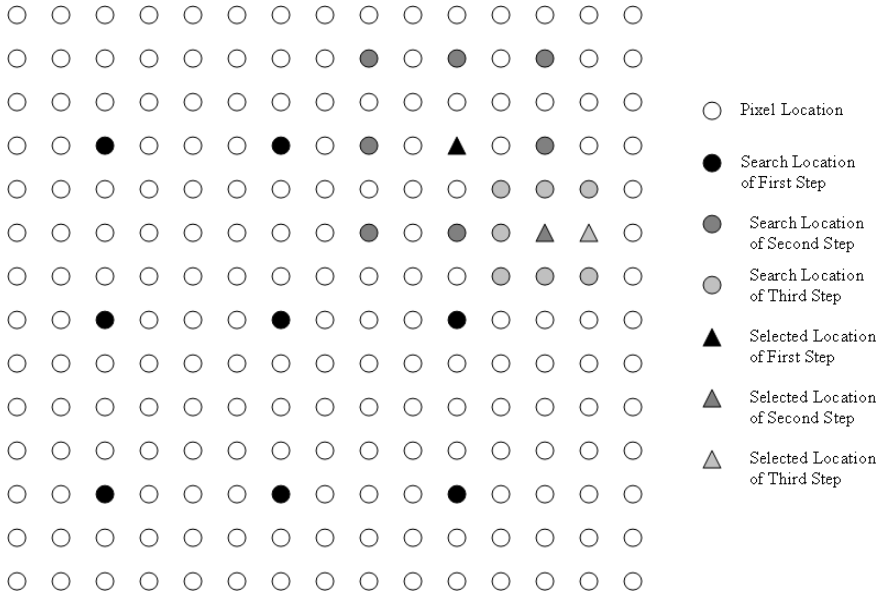


Figure 2.4 The TSS algorithm

The 2D-LOGS [14] algorithm is similar to the TSS algorithm. As shown in Figure 2.5, the 2D-LOGS algorithm searches the MV by successively moving towards the location giving the minimum SAD using a shrinking step size. This algorithm starts with a pre-determined step size “s” and checks five search locations in the first step. If the minimum SAD is found at the center search location, the step size is reduced to “s/2”. Otherwise, the search center is set to the search location giving the minimum SAD and the search continues with step size “s”. Whenever the step size becomes equal to one, as the final search step, the 2D-LOGS algorithm checks the neighboring search locations of the search location giving the minimum SAD in the previous step.

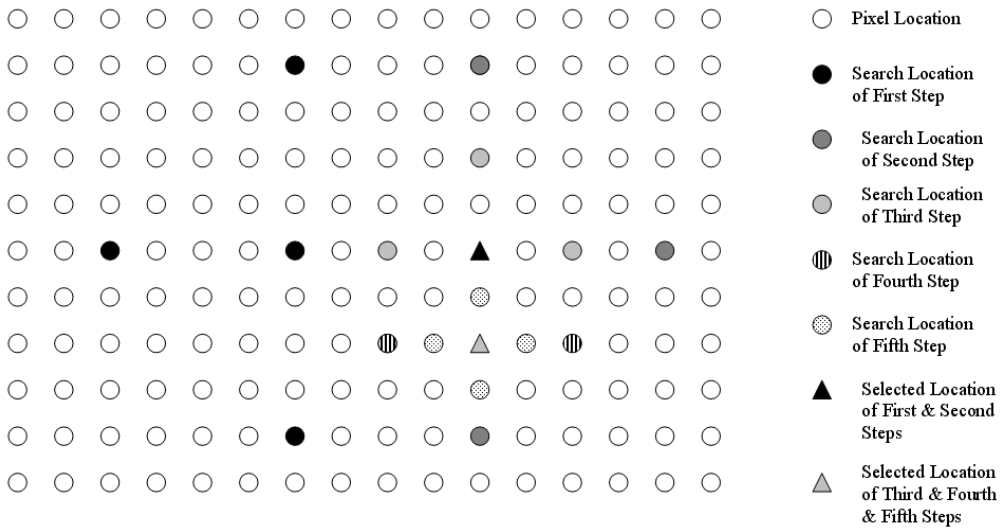


Figure 2.5 The 2D-LOGS algorithm

The NTSS algorithm improves TSS by using a center biased search scheme and reduces the computational complexity by using an early termination technique [15]. As shown in Figure 2.6, NTSS uses eight additional search locations around the center search location in the first step. Therefore, better results are obtained for quasi-stationary blocks. In addition, an early termination technique is used for stationary and quasi-stationary blocks. If the minimum SAD in the first step is found at the center search location, the search is finished. This is called as the first step stop. If the minimum SAD in the first step is found at one of the first tier neighbors of the search center, then the second step is performed for the first tier neighbors of this search location and the search is finished. This is called as the second step stop. The second step stop technique uses three or five new search locations in the second step. Figure 2.7 (a) and (b) show example cases where three and five additional search locations are used. If the minimum SAD after the first step is found at one of the original eight search locations of the TSS algorithm, the search continues as the TSS algorithm.

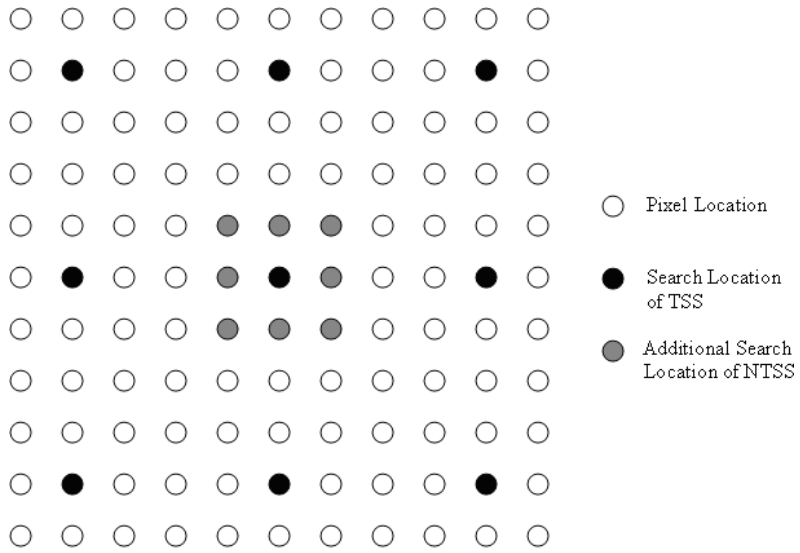


Figure 2.6 Search locations of the first step of the NTSS algorithm

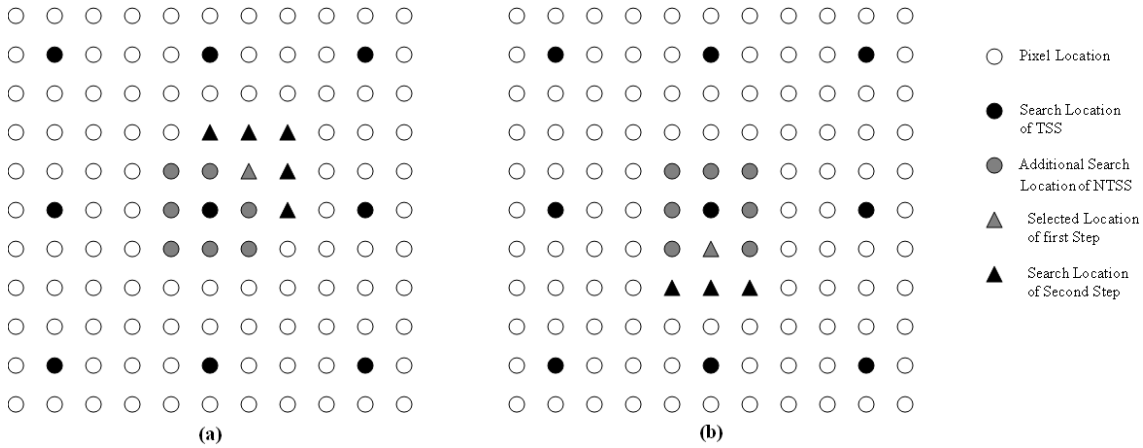


Figure 2.7 Search locations of the first and second steps of the NTSS algorithm

The FSS algorithm also uses a center biased search scheme and an early termination technique [16]. The FSS algorithm performs better than the TSS algorithm and obtains similar results with the NTSS algorithm. When compared with the NTSS algorithm, the FSS algorithm reduces the worst case computational complexity from 33 to 27 search locations. As shown in Figure 2.8, step sizes for the first, second, and third steps of the FSS algorithm are two pixels and step size for the last step is one pixel. In the first step, nine search locations are checked. If the minimum SAD is found at the center search location, the FSS algorithm continues with the fourth step. If the minimum SAD is found at one of the eight neighboring search locations of the center search location, the FSS moves the search center to this location and continues with the second step. If the minimum SAD in the second step is found at the center search location, the FSS algorithm continues with the fourth step. Otherwise, it continues with third step. After the third step, the FSS algorithm continues with the fourth step. In the second and third steps, three or five new search locations are checked based on the search location giving the minimum SAD in the previous step.

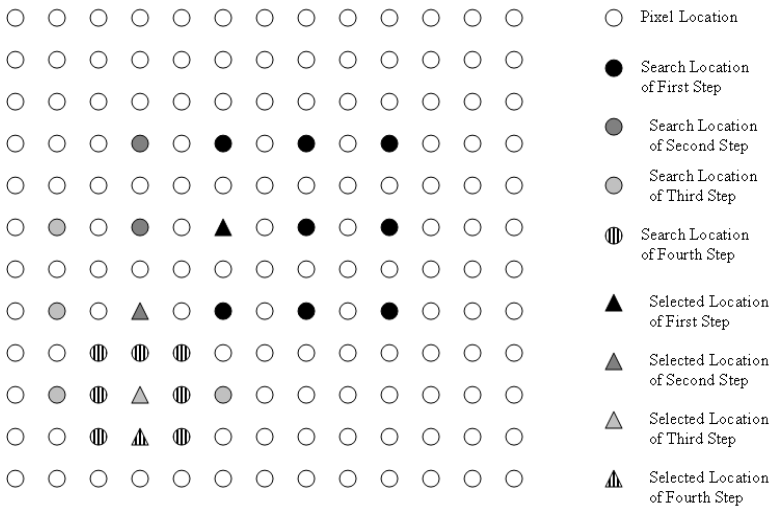


Figure 2.8 The FSS algorithm

As shown in Figure 2.9, the BBGDS algorithm starts by performing FS in a search range of (±1, ±1) pixels around the center search location [17]. If the minimum SAD is found at the center search location, the search finishes. If the minimum SAD is found at one of the other search locations, it moves the center search location to this location and performs FS. Therefore, in each step, three or five new search locations are checked depending on the search location giving the minimum SAD in the previous step.

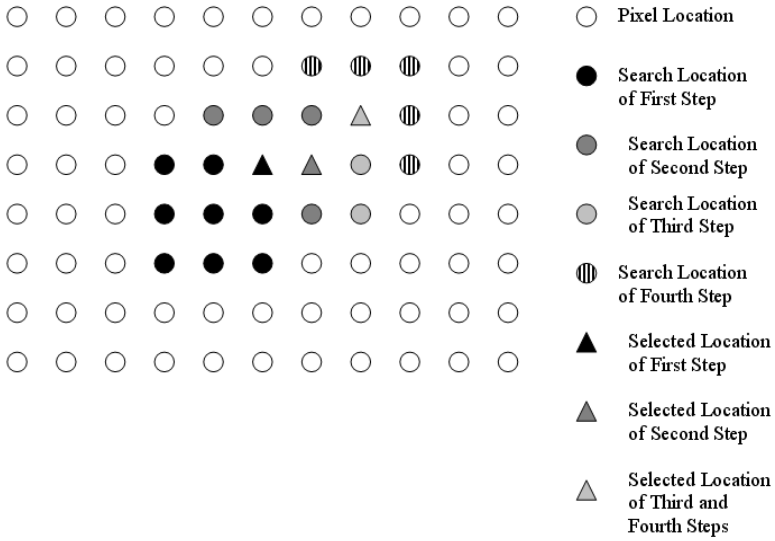


Figure 2.9 The BBGDS algorithm

The DS algorithm is similar to the FSS algorithm. In the DS algorithm, the search pattern is changed from a square to a diamond, and there is no limit on the number of steps performed [18]. The DS algorithm obtains better PSNR results than TSS, 2D-LOGS, NTSS and FSS algorithms. Figure 2.10 shows the two different search patterns, the Large Diamond Search Pattern (LDSP) and the Small Diamond Search Pattern (SDSP), used by the DS algorithm. LDSP is used in all the steps except the last step, SDSP is used in the last step. As shown in Figure 2.11, the number of search locations checked in the next step, which is either three or five, depends on the position of the search location giving the minimum SAD in the current step. If in the current step the minimum SAD is found at the center search location, then the DS algorithm performs the last step.

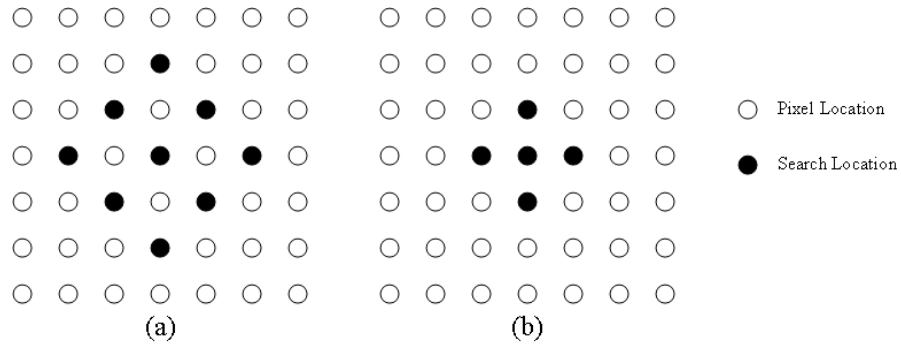


Figure 2.10 The DS algorithm (a) LDSP, (b) SDSP

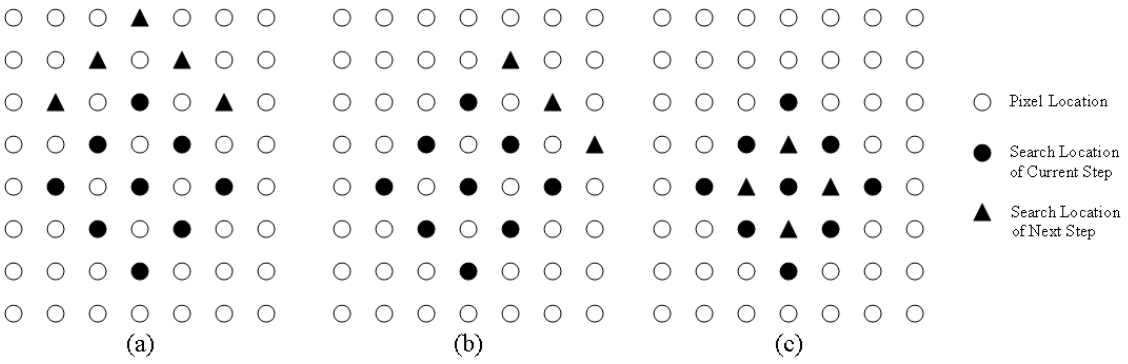


Figure 2.11 Search locations of the DS algorithm for the next step

The HEXBS algorithm uses two search patterns, coarse pattern and fine pattern [19]. Figure 2.12 (a) and (b) show these coarse and fine search patterns. Coarse search pattern is used in all the steps except the last step, fine search pattern is used in the last step. If the search location giving the minimum SAD is found at the center of the hexagon, the algorithm performs the fine search pattern. As shown in Figure 2.13, when

the coarse search pattern is used in the next step, only three new search locations are checked. When the fine search pattern is used in the next step, four neighboring search locations of the center search location are checked.

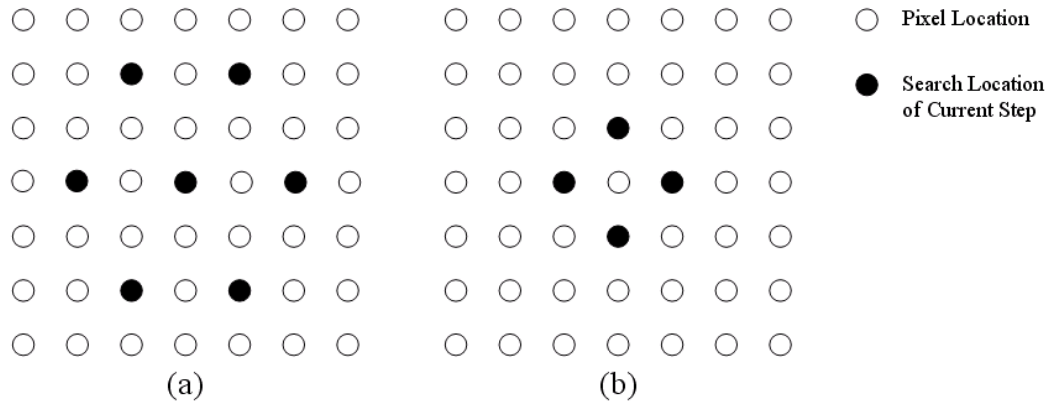


Figure 2.12 The HEXBS algorithm (a) coarse pattern, (b) fine pattern

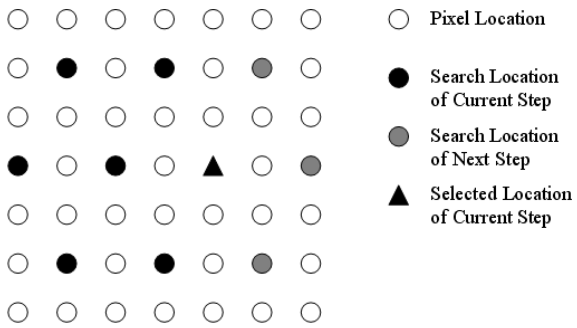


Figure 2.13 Search locations of the HEXBS algorithm

The ARPS algorithm uses a rood shaped search pattern and the MV of the left neighboring MB which is called as predicted MV [20]. The predicted MV and the search pattern of the ARPS algorithm are shown in Figure 2.14. The initial length of the rood is determined as the maximum of the absolute values of x and y coordinates of the predicted MV. The four arms of the rood have equal length. In the first step, the ARPS algorithm checks the search location pointed by the predicted MV, search locations on the rood pattern, and the center search location. The search continues by forming a new rood pattern around the search location giving the minimum SAD in the current step, and the length of the rood is reduced by half in each step. The ARPS algorithm finishes if the minimum SAD obtained in a step is less than a pre-determined threshold or after the step with rood length one.

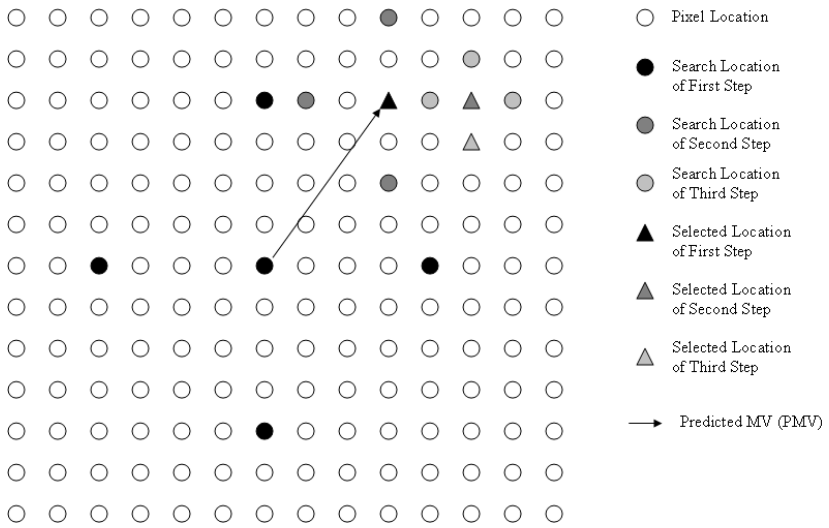


Figure 2.14 The ARPS algorithm

As shown in Figure 2.15, in the first search step, the ADCS algorithm checks the search locations pointed by the MVs of the left neighboring MB and the upper neighboring MB, and the center search location [21]. The search location giving the minimum SAD is selected as the starting location for the dual cross search. If the minimum SAD is below a threshold value, the search finishes. Otherwise, a 2x2 cross pattern around the starting location is searched. If the minimum SAD is found at the cross center, the search finishes and the cross center is selected as the MV. Otherwise, a 4x4 cross pattern around the search location giving the minimum SAD is searched. This 4x4 cross search pattern is repeated until the minimum SAD is found at the cross center. In the last search step, the ADCS algorithm checks three intermediate search locations between the search location on the current 4x4 cross pattern giving the minimum SAD and the current 4x4 cross center.

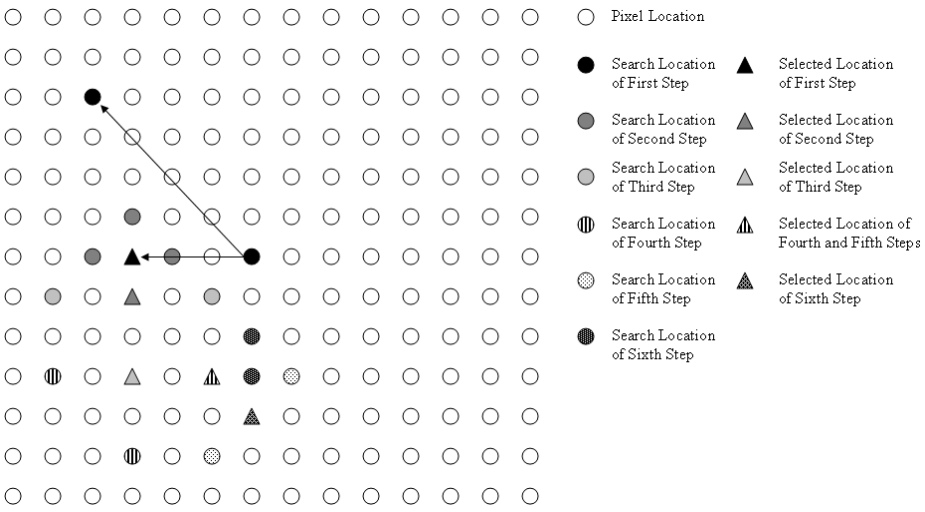


Figure 2.15 The ADCS algorithm

The FTS algorithm searches the search locations on different size triangles [22]. The triangles with larger sizes are used to perform coarse search and the ones with smaller sizes are used to perform fine search. The level of a triangle shows its size, and the FTS algorithm switches between triangles with different levels. Figure 2.16 shows the search locations forming level 0, 1, and 2 triangles.

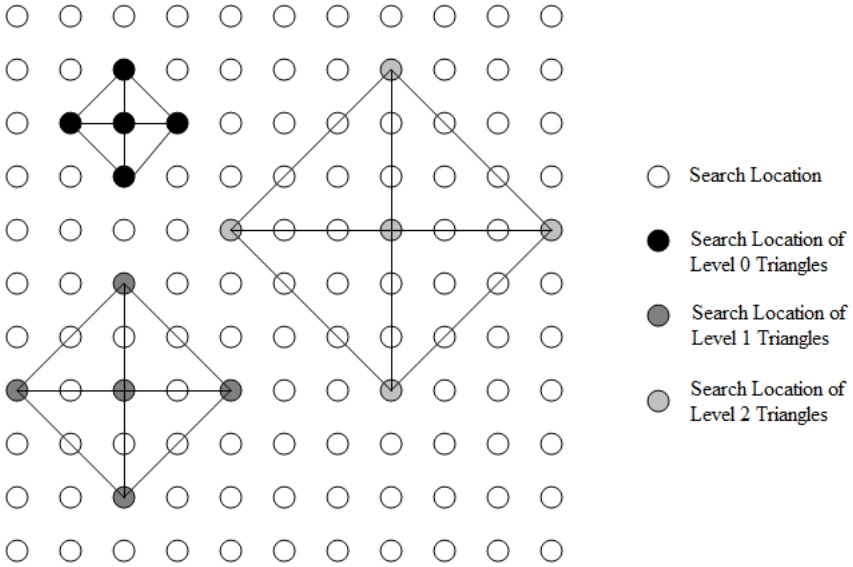


Figure 2.16 The FTS algorithm

2.3 The Three Dimensional Recursive Search Algorithm

The 3D-RS algorithm is one of the most popular true ME algorithms in the literature [24]. The 3D-RS algorithm exploits the correlation of the MVs of neighboring MBs to find the true motion of the current MB. Figure 2.17 shows the neighboring MBs used by the 3D-RS algorithm.

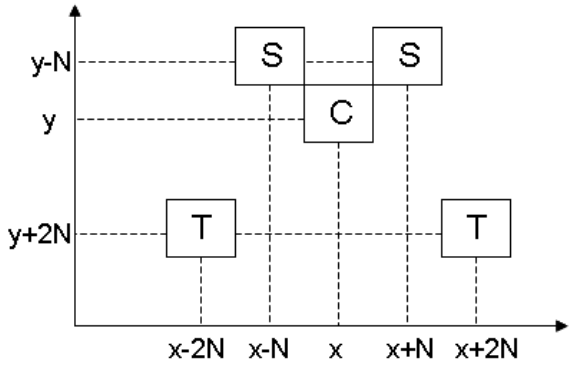


Figure 2.17 Spatial and temporal neighbors for the 3D-RS algorithm

The 3D-RS algorithm is based on two assumptions. The first assumption is that objects are larger than MBs, and the second assumption is that objects have inertia. Therefore, it uses a candidate set that contains the MVs of the spatial and temporal neighboring MBs shown as "S" and "T" in Figure 2.17 [24]. When the spatial neighboring MB is not available, temporal neighboring MB is used. At initialization, all the MVs are set to zero. In addition to the MVs of the spatial and temporal neighboring MBs, an additional update set is used for permitting small deviations from the original candidate set [24]. A pseudo random update vector is added to the MV of one of the spatial neighboring MBs, and this is used as an additional candidate [25]. The candidate MV set of the 3D-RS algorithm is shown in Figure 2.18. The random update vector, shown as $\vec{U}$ (r,t), is used for obtaining the candidate MV C_3, and it is selected from the Update Set ($\vec{US}$). The computational complexity of the 3D-RS algorithm is low, because it checks a few search locations for each MB. The main drawback of the 3D-RS algorithm is its recursive nature. It converges to the true motion a few frames after the initialization.

$$\vec{0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\vec{C}_1 = \vec{d}\left(\vec{r} - \begin{pmatrix} N \\ N \end{pmatrix}, t \right)$$

$$\vec{C}_2 = \vec{d}\left(\vec{r} - \begin{pmatrix} -N \\ N \end{pmatrix}, t \right)$$

$$\vec{C}_3 = \left[\vec{d}\left(\vec{r} - \begin{pmatrix} N \\ N \end{pmatrix}, t \right) + \vec{U}(\vec{r}, t) \right] v \left[\vec{d}\left(\vec{r} - \begin{pmatrix} -N \\ N \end{pmatrix}, t \right) + \vec{U}(\vec{r}, t) \right]$$

$$\vec{C}_4 = \vec{d}\left(\vec{r} + \begin{pmatrix} 2N \\ 2N \end{pmatrix}, t - 1 \right)$$

$$\vec{C}_5 = \vec{d}\left(\vec{r} + \begin{pmatrix} -2N \\ 2N \end{pmatrix}, t - 1 \right)$$

$$\vec{US} = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ -2 \end{bmatrix}, \begin{bmatrix} 3 \\ 0 \end{bmatrix}, \begin{bmatrix} -3 \\ 0 \end{bmatrix} \right\}$$

Figure 2.18 Candidate MV set

CHAPTER 3

HEXAGON BASED MOTION ESTIMATON ALGORITHM AND HARDWARE ARHITECTURES FOR ITS IMPLEMENTATION

Since the computational complexity of the FS algorithm is too high and the performances of fast search algorithms are not enough for the recently available HD video formats, we proposed an ME algorithm [7], which is a generalization of the HEXBS algorithm [19], and two hardware architectures for its implementation [7]. These architectures are named as the generic architecture and the systolic architecture. Many hardware architectures for the FS algorithm are proposed in the literature. However, only a small number of hardware architectures for fast search ME algorithms are proposed. To the best of our knowledge, no hardware architecture is presented for the HEXBS ME algorithm in the literature.

3.1. Hexagon Based Motion Estimation Algorithm

The proposed HEXBS ME algorithm consists of main and fine search patterns [7]. The search location of the main search pattern giving the minimum SAD is selected as the center for the fine search pattern. Main search patterns consist of all the search locations that can be checked by the HEXBS algorithm during several iterations in horizontal and vertical directions. For example, 32x16 main search pattern consists of all the search locations that can be checked by the HEXBS algorithm during 16 iterations in the horizontal direction and 8 iterations in the vertical direction. Figure 3.1 shows some of the search locations of 32x16 pattern. The numbers in Figure 3.1 represent iterations in which these search locations would be checked by the HEXBS algorithm.

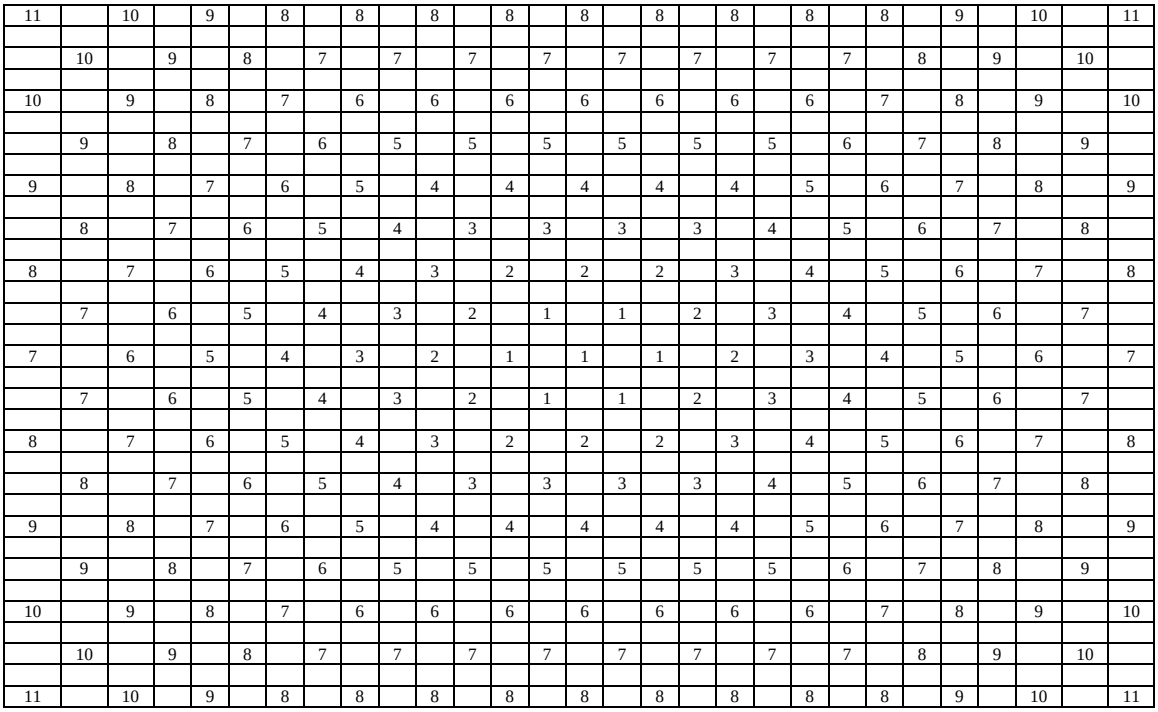


Figure 3.1 Some of the search locations of 32x16 pattern

We also proposed 10x9, 12x12 and 14x15 main search patterns. The difference between these patterns and 32x16 pattern is that these patterns have a gap of two pixels in the vertical direction compared to the one pixel gap of 32x16 pattern, and these patterns have less computational complexity than 32x16 pattern. Figure 3.2 shows the search locations of 10x9 pattern. 12x12 pattern adds one more line in the upper and lower boundaries of the search range and two more pixels in the horizontal direction. 14x15 pattern enhances the search range to ±14 pixels in the horizontal direction and to ±15 pixels in the vertical direction. 12x12 and 14x15 search patterns are shown in Figures 3.3 and 3.4, respectively. In these figures, "o" represents the center search location, and "x" represents the other search locations. Search patterns 10x9, 12x12, 14x15, and 32x16 have 73, 113, 159, and 553 search locations, respectively. In order to determine the trade-off between having one pixel gap and two pixels gap between search locations in the vertical direction, we also implemented 32x16(Y) pattern which has two pixels gap in the vertical direction.

We used the three fine search patterns shown in Figure 3.5. Tables 3.1, 3.2, 3.3, 3.4, and 3.5 show the performances of different combinations of fine search patterns and main search patterns for various Frame Distances (FD). FD is the gap between the

frames for which the ME is done. Since increasing FD is identical to lowering the frame rate of the video, large movements between successive frames are introduced by increasing FD. The results show that "Double Cross" fine search pattern improves the performance up to 1% over other fine search patterns. Therefore, we used this fine search pattern with our main search patterns in the rest of the thesis.

	x		x		x		x		x		x		x		x		x		x	
x		x		x		x		x		x		x		x		x		x		x
	x		x		x		x		x		x		x		x		x		x	
x		x		x		x		x		O		x		x		x		x		x
	x		x		x		x		x		x		x		x		x		x	
x		x		x		x		x		x		x		x		x		x		x
	x		x		x		x		x		x		x		x		x		x	

Figure 3.2 Search locations of 10x9 pattern

x		x		x		x		x		x		x		x		x		x		x		x		x
	x		x		x		x		x		x		x		x		x		x		x		x	
x		x		x		x		x		x		x		x		x		x		x		x		x
	x		x		x		x		x		x		x		x		x		x		x		x	
x		x		x		x		x		x		O		x		x		x		x		x		x
	x		x		x		x		x		x		x		x		x		x		x		x	
x		x		x		x		x		x		x		x		x		x		x		x		x
	x		x		x		x		x		x		x		x		x		x		x		x	
x		x		x		x		x		x		x		x		x		x		x		x		x

Figure 3.3 Search locations of 12x12 pattern

We compared the performance of our algorithm with the performances of FS, DS [18], and HEXBS [19] algorithms based on the MAD metric. We used “Flowers”, “Mobile Calendar”, “Table Tennis”, “Susie”, “Spider”, and “Irobot” videos for the simulations. Each video has 100 frames. “Spider” and “Irobot” videos, which contain large motion between frames, are taken from “Spiderman 2” and “Irobot” movies, respectively. The resolution of these two videos is 720x576 pixels and their frame rate is 25 fps. The other videos are the up-scaled versions of the widely used Common Intermediate Format (CIF) resolution benchmark videos, and they have a resolution of 704x480 pixels and a frame rate of 29 fps. The simulations are done using 8 bit luminance data for 16x16 MB size with Matlab.

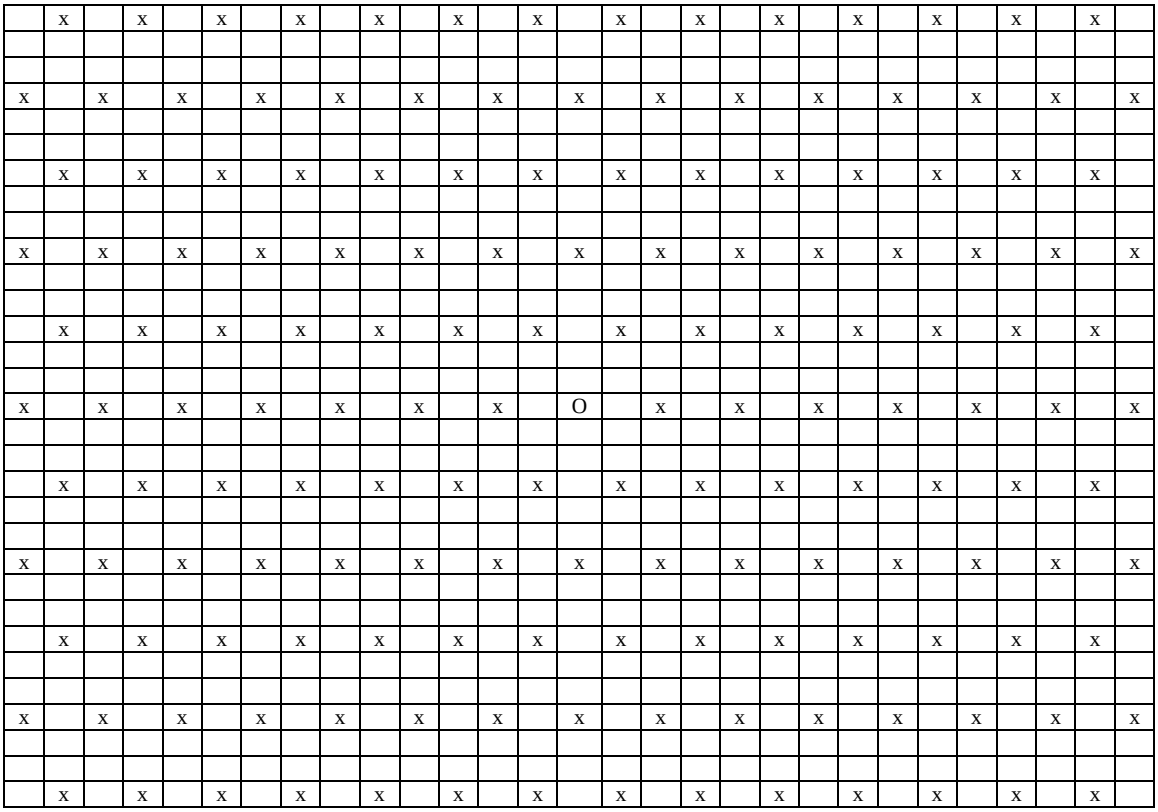


Figure 3.4 Search locations of 14x15 pattern

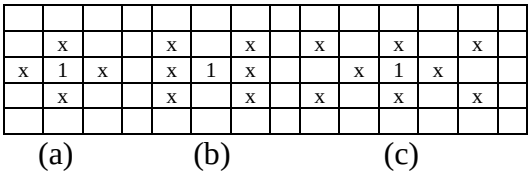


Figure 3.5. Fine search patterns: (a) plus, (b) side, (c) double cross

Table 3.1 MAD results for 32x16 pattern (FD = 1)

Algorithm	Flowers	Mobile Calendar	Table Tennis	Susie
Plus	6.8377	11.3642	4.0670	3.0709
Side	6.8780	11.5426	4.0690	3.0823
Double Cross	6.8295	11.3217	4.0650	3.0612
Imp. of Double Cross over Plus	0.12%	0.37%	0.05%	0.32%
Imp. of Double Cross over Side	0.71%	1.91%	0.10%	0.68%

Table 3.2 MAD results for 32x16 pattern (FD = 2)

Algorithm	Flowers	Mobile Calendar	Table Tennis	Susie
Plus	8.5085	12.0274	4.4561	3.6617
Side	8.5108	12.1349	4.4513	3.6742
Double Cross	8.4789	11.9386	4.4449	3.6496
Imp. of Double Cross over Plus	0.38%	0.74%	0.25%	0.33%
Imp. of Double Cross over Side	0.37%	1.62%	0.14%	0.67%

Table 3.3 MAD results for 32x16 pattern (FD = 3)

Algorithm	Flowers	Mobile Calendar	Table Tennis	Susie
Plus	9.6198	12.7159	4.8755	4.3242
Side	9.6147	12.8476	4.8701	4.3348
Double Cross	9.5820	12.6338	4.8633	4.3112
Imp. of Double Cross over Plus	0.39%	0.65%	0.25%	0.30%
Imp. of Double Cross over Side	0.34%	1.66%	0.14%	0.54%

Table 3.4 MAD results for 10x9 pattern (FD = 1)

Algorithm	Flowers	Mobile Calendar	Table Tennis	Susie
Plus	6.7892	11.5170	4.2255	3.5101
Side	6.8510	11.6879	4.2188	3.5070
Double Cross	6.7747	11.4531	4.2101	3.4742
Imp. of Double Cross over Plus	0.21%	0.55%	0.36%	1.02%
Imp. of Double Cross over Side	1.11%	2.00%	0.21%	0.94%

Table 3.5 MAD results for 10x9 pattern (FD = 2)

Algorithm	Flowers	Mobile Calendar	Table Tennis	Susie
Plus	8.8149	13.1091	4.7517	4.6164
Side	8.8111	13.2902	4.7455	4.5996
Double Cross	8.7374	12.8067	4.7380	4.5742
Imp. of Double Cross over Plus	0.88%	2.31%	0.29%	0.91%
Imp. of Double Cross over Side	0.84%	3.64%	0.16%	0.55%

The simulation results show that proposed search patterns outperform the DS and the HEXBS algorithms. The reason for this is that our patterns are able to find the search location giving the globally minimum SAD by checking more search locations in the search range than the DS and the HEXBS algorithms. The performance difference between proposed patterns and fast search algorithms increases with increased amplitude of motion in the benchmark videos. In order to show this, the performances of the proposed patterns are analyzed for different FDs. The simulation results of 10x9, 12x12, 14x15, 32x16 and 32x16(Y) patterns for different FDs are shown in Tables 3.6, 3.7, and 3.8. As shown in Table 3.6, when the FD is one, 10x9, 12x12, 14x15, 32x16(Y), and 32x16 patterns improve the performance of the HEXBS algorithm on the average by 2.76%, 3.35%, 4.21%, 8.27%, and 10.11%, respectively. For videos having almost no motion in the vertical direction, DS and HEXBS algorithms obtain 1% better results, because DS and HEXBS algorithms have only one pixel gap between search locations in the vertical direction, whereas proposed patterns, except 32x16 pattern, have two pixels gap between search locations in the vertical direction. As shown in Table 3.7, when the FD is two, 10x9, 12x12, 14x15, 32x16(Y), and 32x16 patterns improve the performance of the HEXBS algorithm on the average by 7.46%, 8.12%, 9.19%, 8.20%, and 9.89%, respectively. When the FD is three, 12x12, 14x15, 32x16(Y), and 32x16 patterns improve the performance of the HEXBS algorithm by 11.61%, 12.94%, 14.44%, 19.72%, and 22.43%, respectively. The performance improvements for different FDs are also shown in Figures 3.6, 3.7, and 3.8. Figure 3.6 and Figure 3.7 show the improvements of 10x9 pattern over the HEXBS algorithm frame by frame for “Flowers” video sequence when the FD is one and two, respectively. Figure 3.8 shows the improvement of 12x12 pattern over the HEXBS algorithm for the “Flowers” video sequence when the FD is three.

Table 3.6 MAD results comparison (FD = 1)

Algorithm	Search Range	Flowers	Mobile Calendar	Table Tennis	Susie	Spiderman	Irobot
FS	±10,±9	6.59	10.95	4.07	3.17	9.29	7.58
DS	±10,±9	6.68	11.05	4.16	3.33	9.72	8.12
HEXBS	±10,±9	6.87	11.40	4.17	3.43	10.24	8.45
10x9	±10,±9	6.77	11.45	4.21	3.47	9.34	7.69
Improvement over HEXBS		1.39%	-0.45%	-0.78%	-1.26%	8.73%	8.90%
FS	±12,±12	6.59	10.94	4.05	3.07	8.27	7.14
DS	±12,±12	6.68	11.05	4.15	3.26	8.98	7.79
HEXBS	±12,±12	6.86	11.40	4.16	3.32	9.33	8.04
12x12	±12,±12	6.77	11.46	4.19	3.35	8.30	7.24
Improvement over HEXBS		1.33%	-0.54%	-0.66%	-0.95%	11.04%	9.90%
FS	±14,±15	6.58	10.94	4.04	3.02	7.43	6.82
DS	±14,±15	6.68	11.05	4.15	3.23	8.46	7.57
HEXBS	±14,±15	6.86	11.40	4.15	3.28	8.80	7.82
14x15	±14,±15	6.77	11.46	4.18	3.32	7.48	6.93
Improvement over HEXBS		1.32%	-0.59%	-0.61%	-1.25%	14.99%	11.42%
FS	±32,±16	6.58	10.86	4.03	2.96	5.43	5.66
DS	±32,±16	6.68	11.05	4.14	3.20	7.65	6.97
HEXBS	±32,±16	6.86	11.40	4.15	3.23	7.95	7.21
32x16	±32,±16	6.82	11.32	4.06	3.06	5.47	5.72
32x16(Y)	±32,±16	6.78	11.44	4.17	3.26	5.53	5.79
32x16's Improvement over HEXBS		0.58%	0.7%	2.09%	5.43%	31.23%	20.62%
32x16(Y)'s Improvement over HEXBS		1.27%	-0.42%	-0.54%	-0.82%	30.46%	19.67%

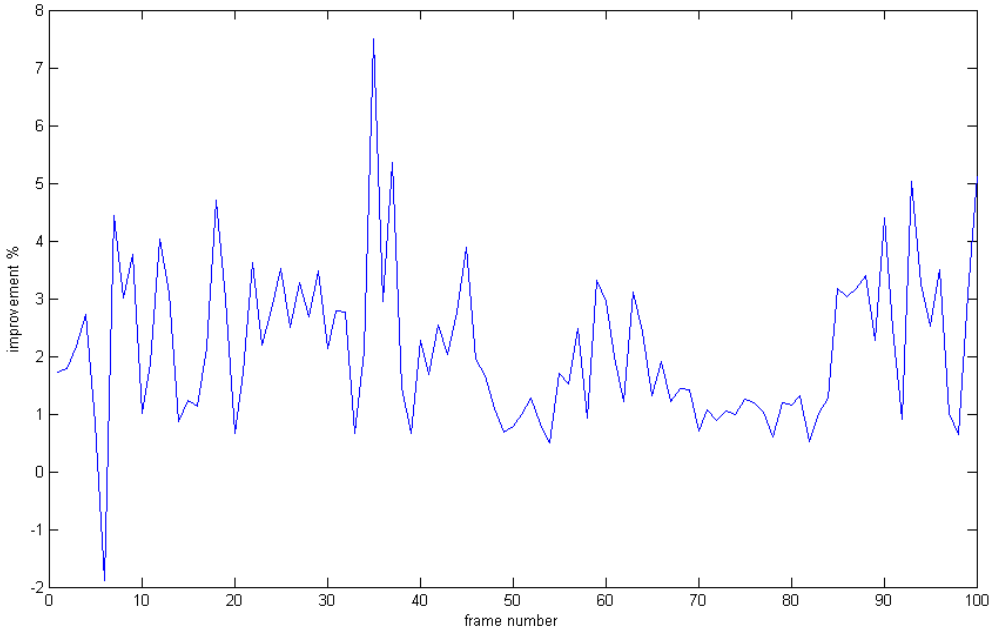


Figure 3.6 Improvement of the 10x9 pattern over HEXBS (FD = 1)

Table 3.7 MAD results comparison (FD = 2)

Algorithm	Search Range	Flowers	Mobile Calendar	Table Tennis	Susie	Spiderman	Irobot
FS	±10,±9	8.41	11.29	4.64	4.33	13.22	12.05
DS	±10,±9	9.82	12.64	4.82	4.62	13.62	12.80
HEXBS	±10,±9	10.36	13.45	4.89	4.84	14.26	13.30
10x9	±10,±9	8.73	12.80	4.73	4.57	13.27	12.18
Improvement over HEXBS		15.72%	4.82%	3.22%	5.60%	6.94%	8.43%
FS	±12,±12	8.33	11.26	4.54	4.08	12.07	11.14
DS	±12,±12	9.79	12.64	4.77	4.43	12.74	12.20
HEXBS	±12,±12	10.33	13.44	4.81	4.59	13.16	12.55
12x12	±12,±12	8.67	12.86	4.63	4.31	12.10	11.25
Improvement over HEXBS		16.10%	4.31%	3.78%	6.12%	8.09%	10.34%
FS	±14,±15	8.32	11.24	4.49	3.91	11.12	10.41
DS	±14,±15	9.79	12.63	4.75	4.31	12.10	11.80
HEXBS	±14,±15	10.33	13.44	4.78	4.46	12.56	12.14
14x15	±14,±15	8.66	12.90	4.59	4.17	11.16	10.55
Improvement over HEXBS		16.11%	4.04%	4.06%	6.62%	11.16%	13.12%
FS	±32,±16	8.31	11.12	4.41	3.55	8.71	8.41
DS	±32,±16	9.79	12.62	4.73	4.14	11.07	10.97
HEXBS	±32,±16	10.33	13.43	4.76	4.27	11.47	11.26
32x16	±32,±16	8.47	11.93	4.44	3.64	8.72	8.49
32x16(Y)	±32,±16	8.67	12.94	4.53	3.83	8.79	8.57
32x16's Improvement over HEXBS		17.94%	11.17%	6.66%	14.64%	23.97%	24.56%
32x16(Y)'s Improvement over HEXBS		16.06%	3.71%	4.76%	10.22%	23.38%	23.82%

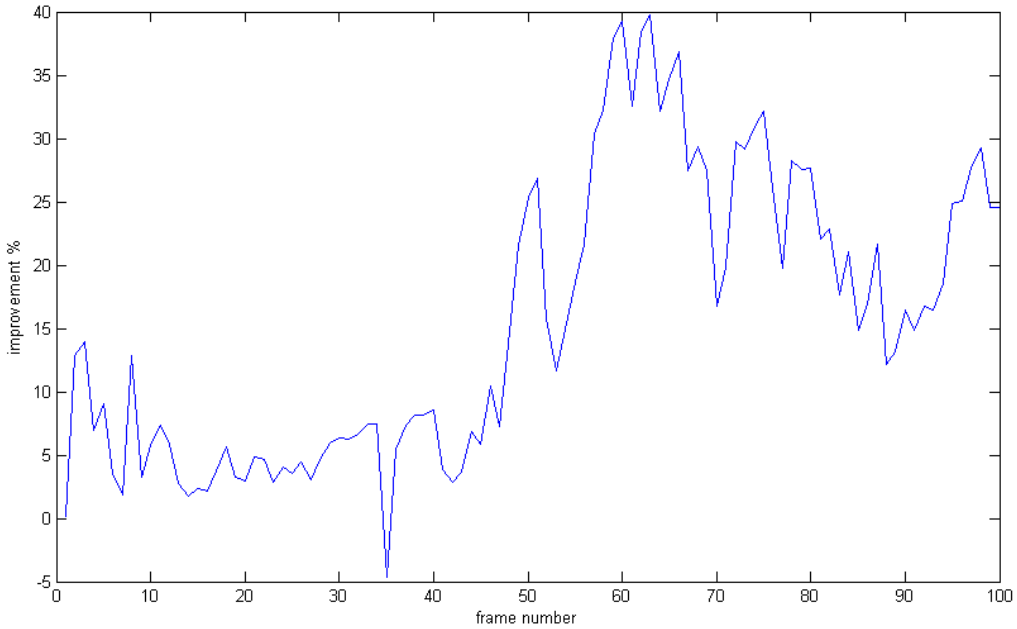


Figure 3.7 Improvement of the 10x9 pattern over HEXBS (FD = 2)

Table 3.8. MAD results comparison (FD = 3)

Algorithm	Search Range	Flowers	Mobile Calendar	Table Tennis	Susie	Spiderman	Irobot
FS	±10,±9	10.62	12.23	5.25	5.44	16.45	14.98
DS	±10,±9	14.49	15.69	5.48	5.86	16.84	15.80
HEXBS	±10,±9	15.16	16.57	5.57	6.15	17.54	16.36
10x9	±10,±9	11.03	13.92	5.32	5.63	16.49	15.10
Improvement over HEXBS		27.23%	15.96%	4.51%	8.30%	5.94%	7.71%
FS	±12,±12	9.88	12.18	5.10	5.07	15.20	13.93
DS	±12,±12	14.32	15.68	5.40	5.59	15.88	15.09
HEXBS	±12,±12	15.00	16.54	5.46	5.79	16.33	15.46
12x12	±12,±12	10.30	13.94	5.17	5.25	15.22	14.03
Improvement over HEXBS		31.30%	15.76%	5.30%	9.21%	6.79%	9.27%
FS	±14,±15	9.55	12.14	5.01	4.81	14.15	13.08
DS	±14,±15	14.27	15.67	5.37	5.41	15.16	14.61
HEXBS	±14,±15	14.96	16.54	5.41	5.60	15.67	14.98
14x15	±14,±15	9.99	13.95	5.09	5.01	14.18	13.20
Improvement over HEXBS		33.21%	15.67%	5.93%	10.51%	9.46%	11.84%
FS	±32,±16	9.36	12.01	4.83	4.23	11.40	10.22
DS	±32,±16	14.26	15.67	5.33	5.11	13.94	13.58
HEXBS	±32,±16	14.94	16.53	5.36	5.31	14.43	13.88
32x16	±32,±16	9.58	12.63	4.86	4.31	11.40	10.29
32x16(Y)	±32,±16	9.85	13.97	4.93	4.45	11.48	10.55
32x16's Improvement over HEXBS		35.90%	23.59%	9.41%	18.81%	20.98%	25.87%
32x16(Y)'s Improvement over HEXBS		34.05%	15.51%	8.16%	16.14%	20.43%	24.03%

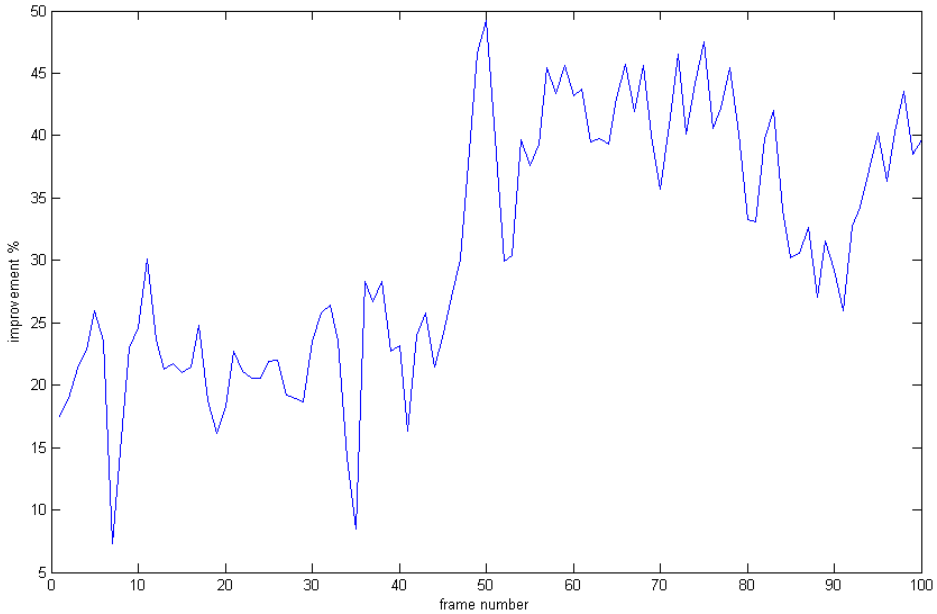


Figure 3.8 Improvement of the 12x12 pattern over HEXBS (FD = 3)

Table 3.9 Total number of search locations for hundred frames (FD = 1)

Algorithm	Search Range	Flowers	Mobile Calendar	Table Tennis	Susie	Spider	Irobot
DS	±10,±9	2368828	2210984	1794202	2551537	5474507	4954167
HEXBS	±10,±9	1819199	1723104	1480132	1890779	3228507	2687300
DS	±12,±12	2376538	2212459	1797994	2587820	6127956	5521728
HEXBS	±12,±12	1822081	1725384	1481011	1932625	3705149	2868489
DS	±14,±15	2382128	2213137	1799742	2612773	6640820	6004468
HEXBS	±14,±15	1823014	1725592	1482769	1953086	4014340	2975468
DS	±32,±16	2389644	2213295	1801483	2639287	7443832	7494935
HEXBS	±32,±16	1823591	1725714	1484750	1979635	4556908	3253332

Table 3.10 Total number of search locations for hundred frames (FD = 2)

Algorithm	Search Range	Flowers	Mobile Calendar	Table Tennis	Susie	Spiderman	Irobot
DS	±10,±9	2839416	2707125	1870785	2972983	5674083	5379456
HEXBS	±10,±9	2081194	2025332	1507523	2073247	3270665	2847186
DS	±12,±12	2857585	2713436	1886509	3071880	6431147	6074068
HEXBS	±12,±12	2094245	2031730	1522287	2159607	3800457	3120219
DS	±14,±15	2866216	2716244	1896276	3145417	7049596	6652328
HEXBS	±14,±15	2097620	2034001	1529376	2206610	4160162	3288727
DS	±32,±16	2875455	2718188	1905789	3256697	8061850	8289404
HEXBS	±32,±16	2099428	2036426	1538257	2287782	4813062	3686533

Table 3.9 and Table 3.10 show the total number of search locations checked by DS and HEXBS algorithms for various benchmark videos for different FDs. For example, the HEXBS algorithm checks 4556908 search locations for 100 frames of the "Spider" video, when the search range is (±32,±16) pixels and FD is one. On the average, 28.1 search calculations are checked to find a MV.

3.2. Generic Motion Estimation Hardware Architectures

We proposed the generic hardware architecture for implementing various fast search algorithms. We proposed two different implementations of the generic hardware architecture, named as the implementation Type I and the implementation Type II, for calculating an SAD value, and we designed two different PE architectures for these implementations. Figure 3.9 shows the block diagrams of PE_I and PE_{II}. In both PEs, the

absolute difference between the current pixel and the reference pixel is calculated and stored in the SAD register. The difference between PE_I and PE_{II} is the multiplexer in the PE_{II}. This multiplexer allows zeros to be feed into the adder tree, which is needed for the implementation type II.

The block diagrams of the implementation type I and type II for a MB size of 16x16 pixels are shown in Figure 3.10 and Figure 3.11, respectively. In both implementations, the outputs of PEs are added with an adder tree. Implementation type I has a 16x16 PE_I array, and horizontal shifters are used to align the reference MB read from BRAMs with the current MB in the PE_I array. In this implementation, the current MB is loaded into the current registers of the PE_I array only once. In implementation type II, smaller horizontal shifters are used to align the current MB, but a 20x16 PE_{II} array is used. The advantage of using a larger PE array, which is capable of feeding zeros into the adder tree, is that there is no need for shifting the reference data read from BRAMs. On the other hand, the current MB has to be aligned and loaded into the current registers of the PE_{II} array as many times as the number of search locations. The trade-off between these implementation types is shown in Table 3.11. Based on this analysis, implementation type I is determined to be better than implementation type II. Therefore, it is called as the “16x16 Generic Architecture” and used in the rest of this thesis.

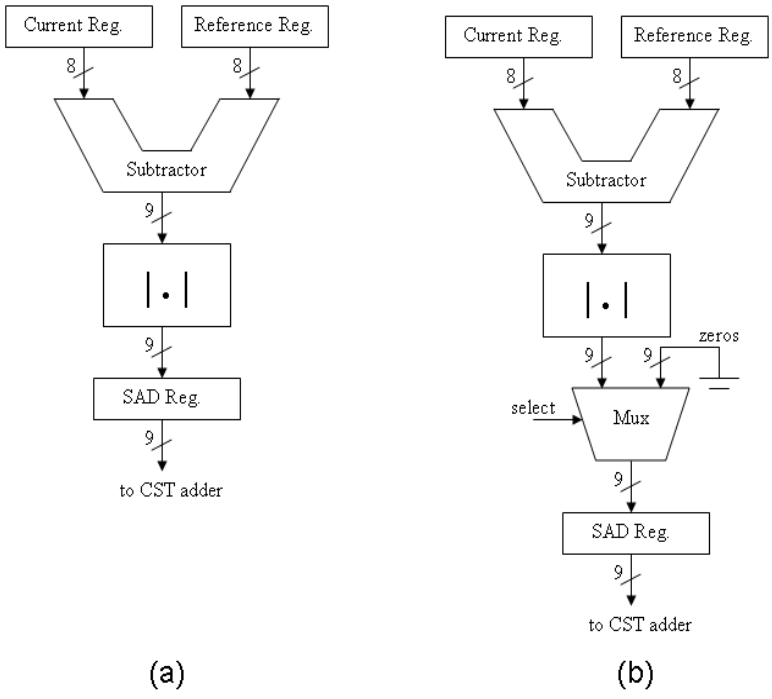


Figure 3.9 Block diagram of processing elements: (a) PE_I , (b) PE_{II}

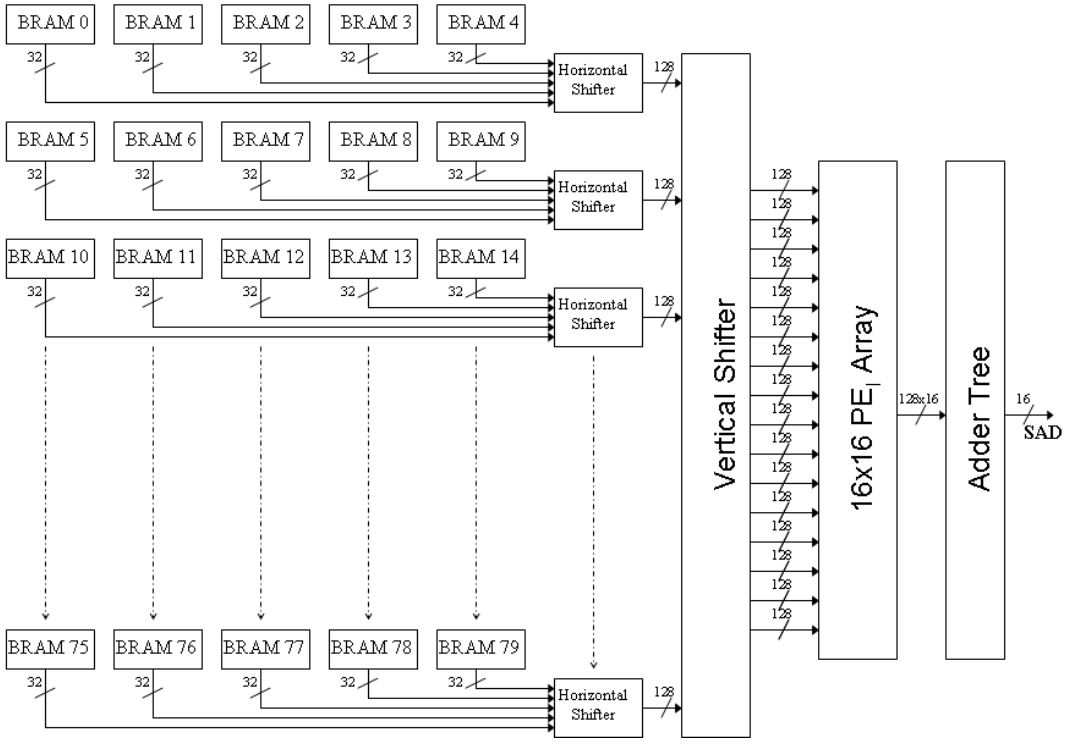


Figure 3.10 Block diagram of the implementation type I

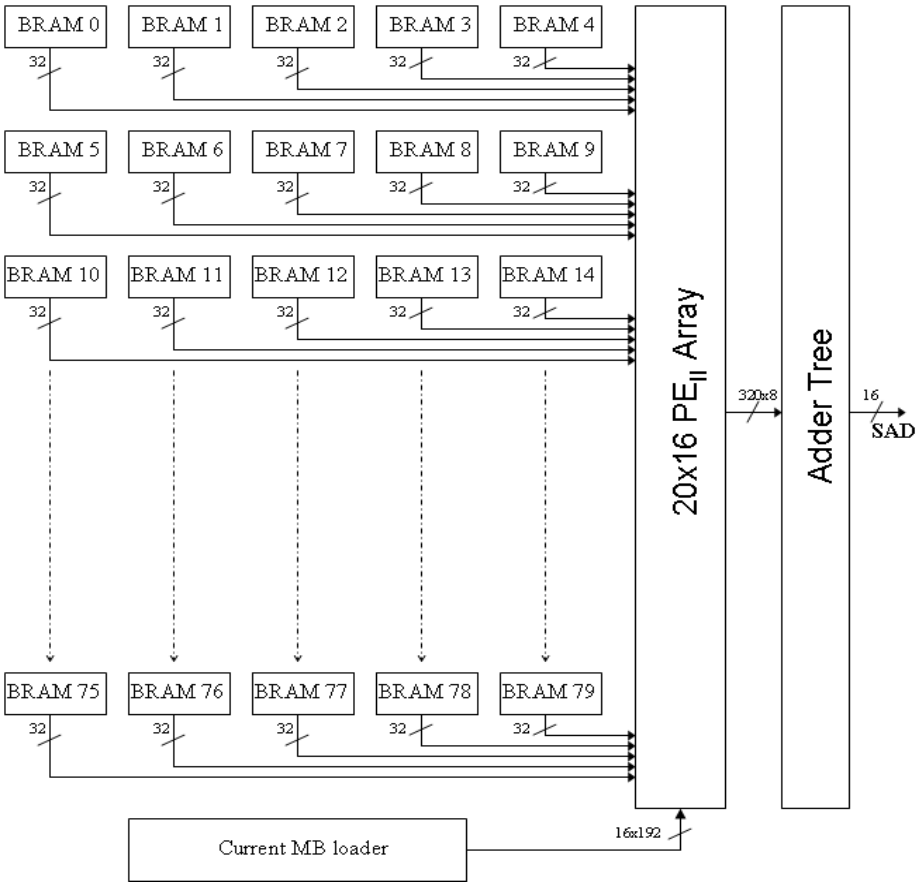


Figure 3.11 Block diagram of the implementation type II

Table 3.11 Trade-off between implementation types I and II

Modules	**Implementation Type I**	**Implementation Type II**
PE	256 PEI	320 PEII
Horizontal Shifter	128 20:16	128 16:16
Vertical Shifter	128 16:16	128 16:16
Adder Tree	N	1.25 N

The generic architecture has seven pipeline stages. In order to calculate the SAD of a search location for a 16x16 MB in one clock cycle, 256 PEs are used and their outputs are added with an adder tree. If MBs are divided into blocks, and a block is processed in one clock cycle, smaller number of PEs, adders and shifters can be used. The generic architectures for the block sizes of 16x8, 16x6, 16x4, and 16x2, are shown in Figures 3.12, 3.13, 3.14, and 3.15 respectively. Area and performance comparison of these generic architectures on a Xilinx Spartan 3E FPGA is given in Table 3.12. Area comparisons of horizontal and vertical shifters for these generic architectures are given in Tables 3.13 and 3.14, respectively.

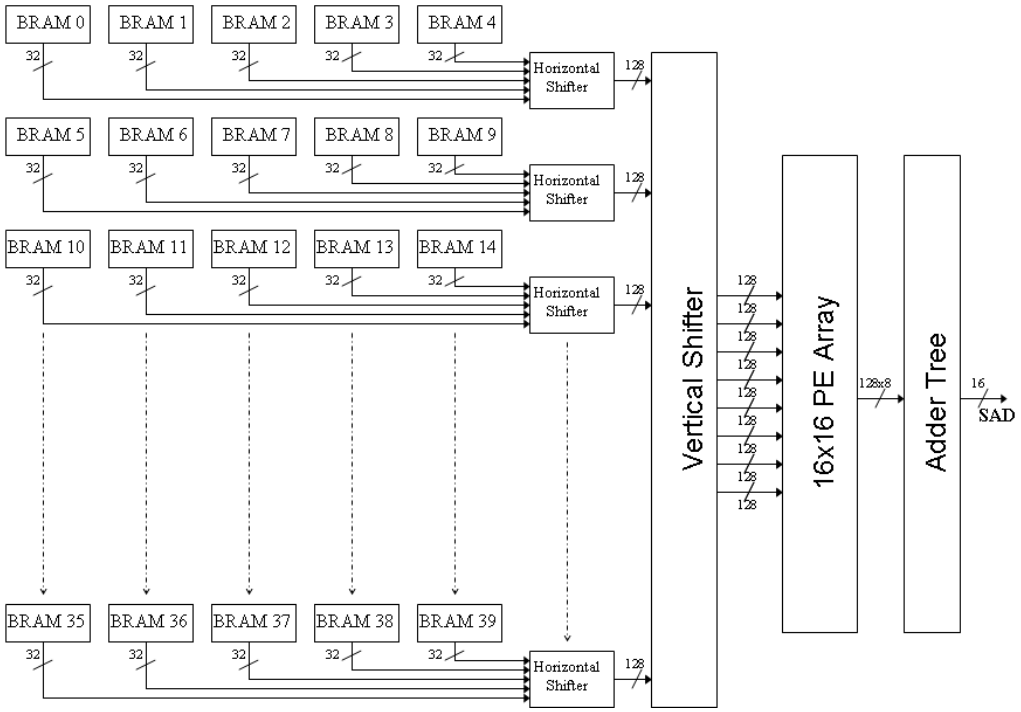


Figure 3.12 16x8 generic architecture

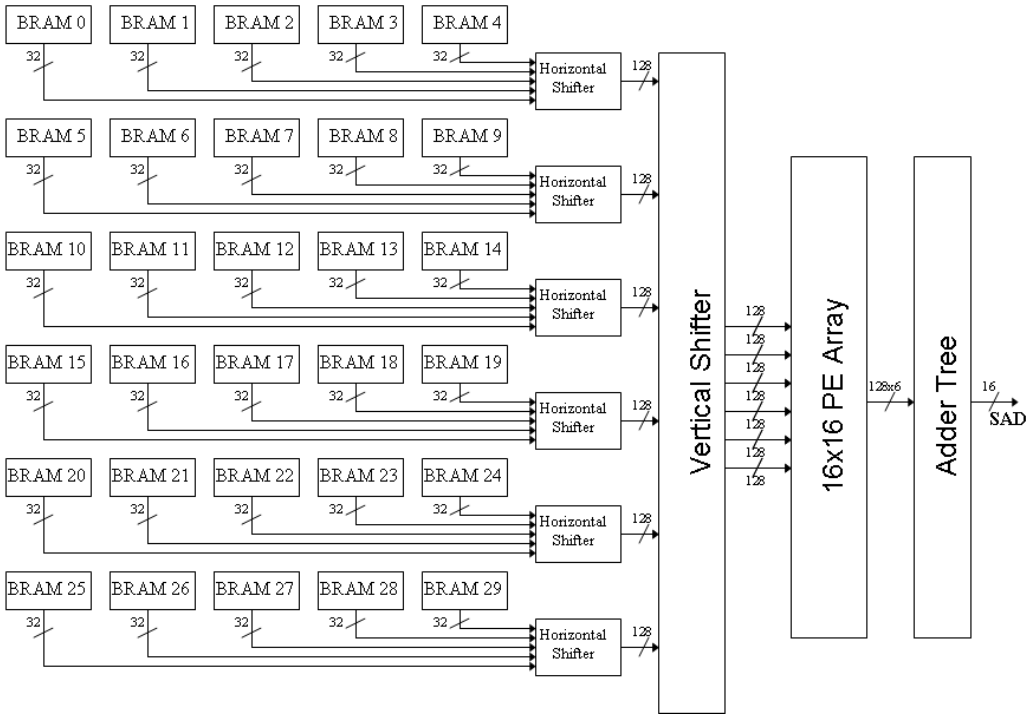


Figure 3.13 16x6 generic architecture

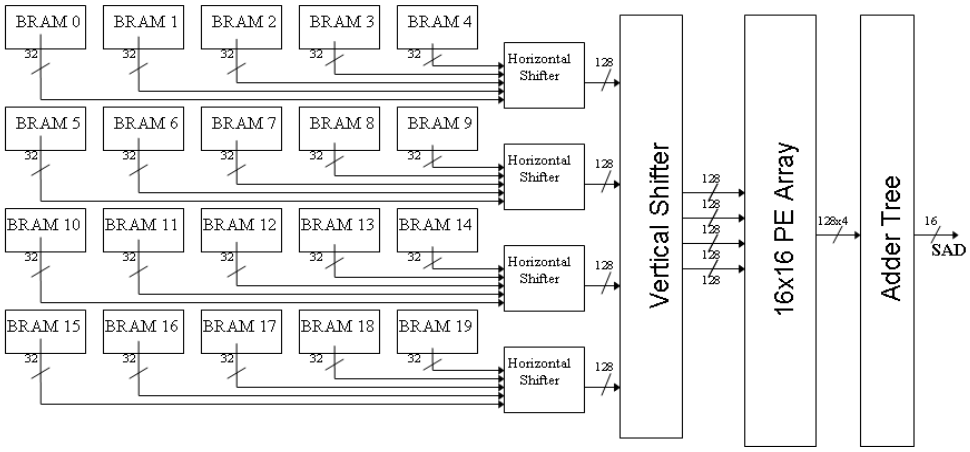


Figure 3.14 16x4 generic architecture

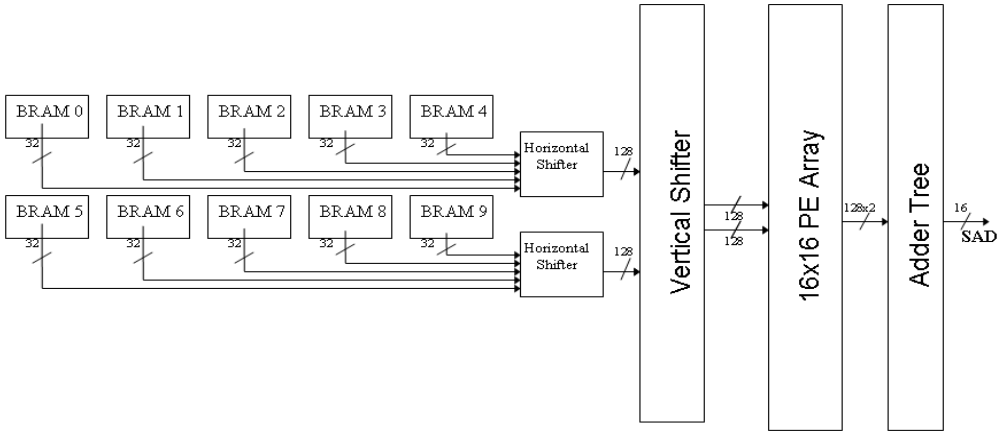


Figure 3.15 16x2 generic architecture

Table 3.12 Comparison of generic architectures for various block sizes

Block Size	Number of BRAMs	Number of PEs	SAD of a 16x16 MB (Cycles)	Total PE Array Area with Adder Tree (LUTs)	Total Area (LUTs)
16x16	80	256	1	6940	31416
16x8	40	128	2	3463	14675
16x6	30	96	3	2580	9447
16x4	20	64	4	1726	6304
16x2	10	32	8	857	2889

Table 3.13 Comparison of horizontal shifters for various generic architectures

Block Size	Number of Horizontal Shifters	Number of 20 to 16 Shifters in a Horizontal Shifter	Total Number of 20 to 16 Shifters	Total Area (LUTs)
16x16	16	8	128	14208
16x8	8	8	64	7104
16x6	6	8	48	5328
16x4	4	8	32	3552
16x2	2	8	16	1776

Table 3.14 Comparison of vertical shifters for various generic architectures

Block Size	Number of 128bit lines in a Vertical Shifter	Type of Shifters	Number of Shifters	Total Area (LUTs)
16x16	16	16 to16	128	10268
16x8	8	8 to 8	128	4108
16x6	6	6 to 6	128	1539
16x4	4	4 to 4	128	1026
16x2	2	2 to 2	128	256

The data layout in BRAMs is shown in Figure 3.16. Five BRAMs are used to store one line of the search window. This is done to avoid data collisions that can occur while accessing the reference MB for a search location. Since the maximum word length of BRAMs in the state of the art FPGAs is 32 bits, each memory location stores four pixels. In Figure 3.16, each box represents a pixel and the number in the box indicates the BRAM storing that pixel. Dark shaded area shows the reference MB for an example search location for 16x16 MB size. In order to access the reference MB for an arbitrary search location, outputs of the BRAMs should be aligned. This is done by horizontal and vertical shifters. For the example shown in Figure 3.16, in order to align the reference MB with the current MB, horizontal shifters should rotate their 160 bit input ten bytes to left and clip the least significant four bytes, and the vertical shifter should rotate its inputs to left by six lines. Figure 3.17 and Figure 3.18 show these horizontal and vertical rotate operations.

0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	0	0	0	0	1	1	1	1	2	2
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	9	9	9	9	5	5	5	5	6	6	6	6	7	7
10	10	10	10	11	11	11	11	12	12	12	12	13	13	13	13	14	14	14	14	10	10	10	10	11	11	11	11	12	12
15	15	15	15	16	16	16	16	17	17	17	17	18	18	18	18	19	19	19	19	15	15	15	15	16	16	16	16	17	17
20	20	20	20	21	21	21	21	22	22	22	22	23	23	23	23	24	24	24	24	20	20	20	20	21	21	21	21	22	22
25	25	25	25	26	26	26	26	27	27	27	27	28	28	28	28	29	29	29	29	25	25	25	25	26	26	26	26	27	27
30	30	30	30	31	31	31	31	32	32	32	32	33	33	33	33	34	34	34	34	30	30	30	30	31	31	31	31	32	32
35	35	35	35	36	36	36	36	37	37	37	37	38	38	38	38	39	39	39	39	35	35	35	35	36	36	36	36	37	37
40	40	40	40	41	41	41	41	42	42	42	42	43	43	43	43	44	44	44	44	40	40	40	40	41	41	41	41	42	42
45	45	45	45	46	46	46	46	47	47	47	47	48	48	48	48	49	49	49	49	45	45	45	45	46	46	46	46	47	47
50	50	50	50	51	51	51	51	52	52	52	52	53	53	53	53	54	54	54	54	50	50	50	50	51	51	51	51	52	52
55	55	55	55	56	56	56	56	57	57	57	57	58	58	58	58	59	59	59	59	55	55	55	55	56	56	56	56	57	57
60	60	60	60	61	61	61	61	62	62	62	62	63	63	63	63	64	64	64	64	60	60	60	60	61	61	61	61	62	62
65	65	65	65	66	66	66	66	67	67	67	67	68	68	68	68	69	69	69	69	65	65	65	65	66	66	66	66	67	67
70	70	70	70	71	71	71	71	72	72	72	72	73	73	73	73	74	74	74	74	70	70	70	70	71	71	71	71	72	72
75	75	75	75	76	76	76	76	77	77	77	77	78	78	78	78	79	79	79	79	75	75	75	75	76	76	76	76	77	77
0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	4	4	0	0	0	0	1	1	1	1	2	2
5	5	5	5	6	6	6	6	7	7	7	7	8	8	8	8	9	9	9	9	5	5	5	5	6	6	6	6	7	7
10	10	10	10	11	11	11	11	12	12	12	12	13	13	13	13	14	14	14	14	10	10	10	10	11	11	11	11	12	12
15	15	15	15	16	16	16	16	17	17	17	17	18	18	18	18	19	19	19	19	15	15	15	15	16	16	16	16	17	17
20	20	20	20	21	21	21	21	22	22	22	22	23	23	23	23	24	24	24	24	20	20	20	20	21	21	21	21	22	22
25	25	25	25	26	26	26	26	27	27	27	27	28	28	28	28	29	29	29	29	25	25	25	25	26	26	26	26	27	27
30	30	30	30	31	31	31	31	32	32	32	32	33	33	33	33	34	34	34	34	30	30	30	30	31	31	31	31	32	32
35	35	35	35	36	36	36	36	37	37	37	37	38	38	38	38	39	39	39	39	35	35	35	35	36	36	36	36	37	37
40	40	40	40	41	41	41	41	42	42	42	42	43	43	43	43	44	44	44	44	40	40	40	40	41	41	41	41	42	42
45	45	45	45	46	46	46	46	47	47	47	47	48	48	48	48	49	49	49	49	45	45	45	45	46	46	46	46	47	47
50	50	50	50	51	51	51	51	52	52	52	52	53	53	53	53	54	54	54	54	50	50	50	50	51	51	51	51	52	52
55	55	55	55	56	56	56	56	57	57	57	57	58	58	58	58	59	59	59	59	55	55	55	55	56	56	56	56	57	57
60	60	60	60	61	61	61	61	62	62	62	62	63	63	63	63	64	64	64	64	60	60	60	60	61	61	61	61	62	62
65	65	65	65	66	66	66	66	67	67	67	67	68	68	68	68	69	69	69	69	65	65	65	65	66	66	66	66	67	67
70	70	70	70	71	71	71	71	72	72	72	72	73	73	73	73	74	74	74	74	70	70	70	70	71	71	71	71	72	72
75	75	75	75	76	76	76	76	77	77	77	77	78	78	78	78	79	79	79	79	75	75	75	75	76	76	76	76	77	77

Figure 3.16 Data layout in BRAMs

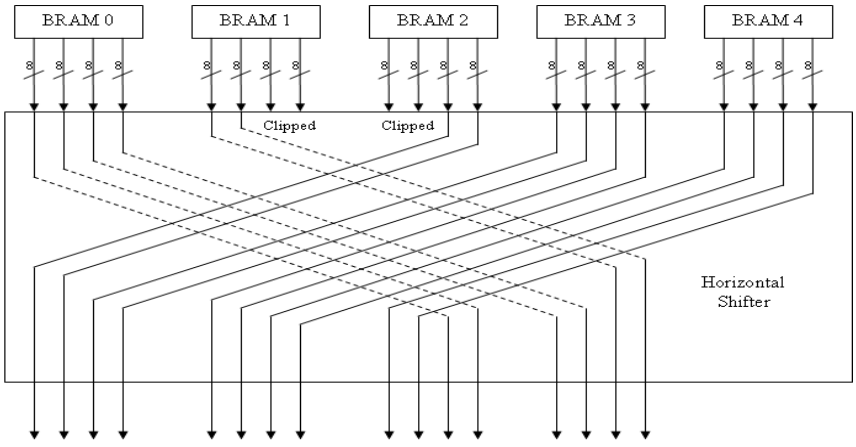


Figure 3.17 Ten byte rotate left operation done by the horizontal shifter

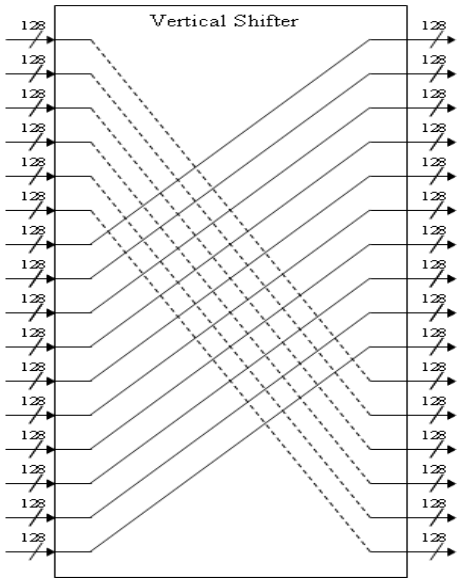


Figure 3.18 Six line rotate left operation done by the vertical shifter

In the proposed generic hardware architecture, there are three pipeline stages named as SHFT, SAD, ADD. Reference MB is read from the BRAMs and aligned by shifters in the SHFT stage. The absolute differences between corresponding current and reference pixels are calculated in the SAD stage. The SAD for a 16x16 MB is calculated by adding these absolute differences in the ADD stage. The pipelining in the proposed generic hardware architecture is shown in Table 3.15. “a1” to “a7” represent the seven search locations in the first iteration of the HEXBS algorithm. Similarly, “b1”, “b2”, and “b3” represent the three search locations in the next iteration. The pipeline has to stall between iterations, because the next iteration is dependent on the data obtained from the previous iteration. The number of stall cycles is equal to the number of pipeline stages minus one. Therefore, the three stage pipelined datapath must be stalled for two cycles between iterations. In the HEXBS algorithm, the number of search iterations is limited by the search window size. For a search window of (±32,±16) pixels, if the search continues horizontally, the datapath will be stalled 16 times, i.e. 32 cycles.

Table 3.15 Pipelining in the generic hardware architecture

Clock cycles	SAD a1	SAD a2	SAD a3	SAD a4	SAD a5	SAD a6	SAD a7	SAD b1	SAD b2	SAD b3	SAD c1
1	SHFT										
2	SAD	SHFT									
3	ADD	SAD	SHFT								
4		ADD	SAD	SHFT							
5			ADD	SAD	SHFT						
6				ADD	SAD	SHFT					
7					ADD	SAD	SHFT				
8						ADD	SAD	stall			
9							ADD	stall			
10								SHFT			
11								SAD	SHFT		
12								ADD	SAD	SHFT	
13									ADD	SAD	stall
14										ADD	stall
15											SHFT
16											SAD
17											ADD

The proposed generic hardware architecture is implemented in VHDL, verified with Register Transfer Level (RTL) simulations using Mentor Graphics Modelsim 6.3c and mapped to Xilinx XC3S1200-5 FPGA using Xilinx ISE 9.2.04. The proposed hardware can work at 144 MHz on this FPGA. Therefore, for the largest search window size of (±32,±16) pixels, it can process 206743 MBs per second. Therefore, it is capable of processing 127 fps, 57 fps, and 25 fps for 720x576, 1280x720 and 1920x1080 resolutions, respectively. The disadvantage of the generic architecture is that it uses 80 BRAMs.

Since 16x16 and 16x8 generic hardware architectures use large number of BRAMs, it is not possible to implement them on current low cost FPGAs. Although 16x4 and 16x2 generic hardware architectures can be implemented on a low cost FPGA, they are not suitable for real-time implementation of high frame size and high frame rate applications, because they require large number of clock cycles to calculate an SAD value. Therefore, in the next section, we propose a systolic ME hardware architecture for real-time implementation of high frame size and high frame rate applications on a low cost FPGA.

3.3. Systolic Motion Estimation Hardware Architecture

The systolic ME hardware architecture proposed to efficiently implement the proposed HEXBS ME algorithm and its datapath are shown in Figures 3.19 and 3.20. This systolic architecture is designed to reduce the internal memory bandwidth by applying data-reuse [7]. It has six pipeline stages. It has 256 PEs and accumulates their results with an adder tree. The main difference between the systolic architecture and the generic architecture is it that not all of the PEs receive their reference data directly from BRAMs. 16 BRAMs, configured for 16 bit port width, are connected to 32 PEs. The remaining 224 PEs receive their reference data from their neighboring PEs. Reference data is shifted to right in the PE array. Loading the reference data of a search location has a start-up cost of 8 cycles. After the PE array is loaded, SAD values of the search locations in the same line is obtained in each clock cycle.

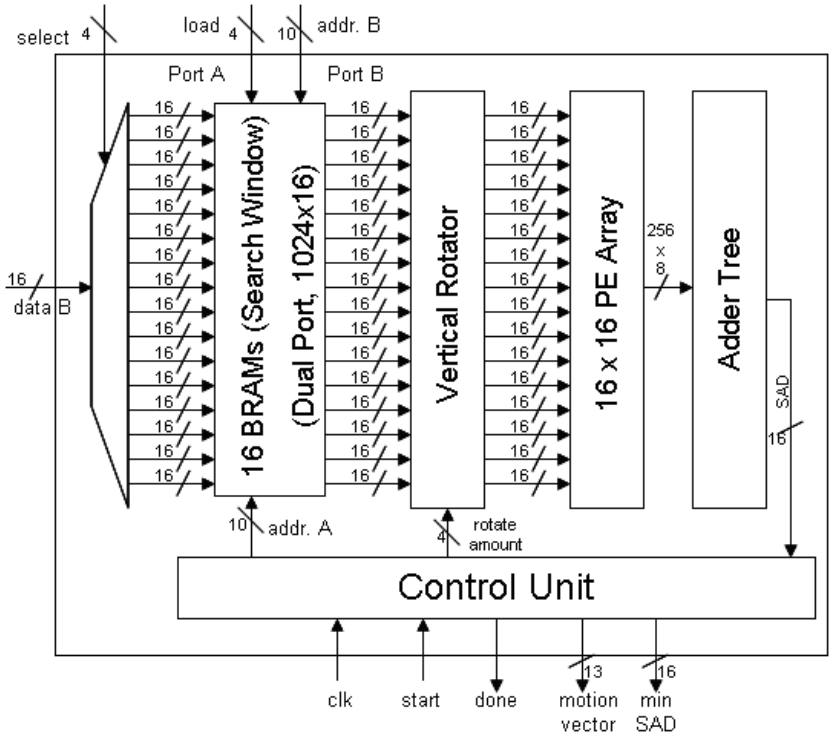


Figure 3.19 Top-level block diagram of the systolic architecture

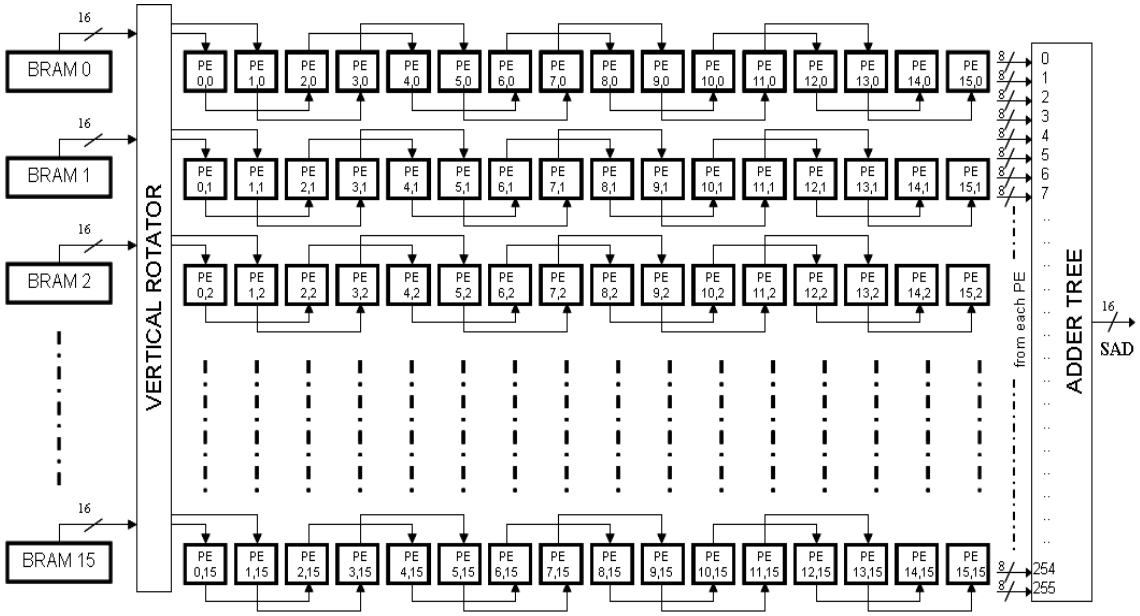


Figure 3.20 Datapath of the systolic architecture

Table 3.16 shows the total number of search locations in different search patterns and the number of clock cycles required to check these search locations on the systolic architecture. “Double Cross” fine search pattern has an overhead of four clock cycles compared to “Plus” fine search pattern.

Table 3.16 Search patterns

Search Range	**Number of Search Locations**	**Required Clock Cycles**
±10, ±9	73	122
±12, ±12	113	176
±14, ±15	159	236
±32, ±16	553	672
Fine Search Pattern	**Number of Search Locations**	**Required Clock Cycles**
Plus	4	25
Side	6	27
Double Cross	8	29

Table 3.17 shows the data flow through the proposed systolic architecture. Let A1 – L2 shown in Figure 3.21 denote the pixels in these columns. In this figure, search locations of the proposed HEXBS patterns are shown as bold. A1 denotes the pixels in the column A1 and A2 denotes the pixels in the right neighboring column. Assuming that D1 is the first search location in the line, in the first clock cycle, the PE array is

filled with the pixels in columns L1 and L2. In the second clock cycle, these pixels are shifted to the right in the PE array by two pixels and the pixels in columns K1 and K2 are loaded into two left end columns of the PE array. Therefore, in the 8th clock cycle, the SAD value of search location D1 is obtained. In the 9th, 10th and 11th clock cycles, SAD values of search locations C1, B1 and A1 are obtained.

	X		**x**		**x**		**x**		**x**		**x**		**x**		**x**		**x**		**x**		**x**
A1	A2	**B1**	B2	**C1**	C2	**D1**	D2	**E1**	E2	**F1**	F2	**G1**	G2	**H1**	H2	**J1**	J2	**K1**	K2	**L1**	L2
	X		**x**		**x**		**x**		**x**		**x**		**x**		**x**		**x**		**x**		**x**
x		**x**		**x**		**x**		**x**		**X**		**x**		**x**		**x**		**x**		**x**	

Figure 3.21 Search locations of the proposed HEXBS patterns

Table 3.17 Data flow through the systolic PE array

Clock Cycles	Processing Elements															
	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12	Col 13	Col 14	Col 15
1	L1	L2														
2	K1	K2	L1	L2												
3	J1	J2	K1	K2	L1	L2										
4	H1	H2	J1	J2	K1	K2	L1	L2								
5	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2						
6	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2				
7	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2		
8	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2	L1	L2
9	C1	C2	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2	K1	K2
10	B1	B2	C1	C2	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2	J1	J2
11	A1	A2	B1	B2	C1	C2	D1	D2	E1	E2	F1	F2	G1	G2	H1	H2

Pixel organization in BRAMs is shown in Figure 3.22. Each BRAM has three regions (0, 1, 2) for storing three different lines of the search window. For example, BRAM 0 stores 0th, 16th, and 32th lines of the search window. The outputs of BRAMs are aligned with vertical rotator. The vertical rotator consists of 16 16-bit rotators. Rotate amount signal generated by the control unit determines how many lines the outputs of the BRAMs will be rotated by the vertical rotator. The rotate amounts for different search locations are shown in Figure 3.23. For the search locations in the first line of the search window, the rotate amount is zero and it increases by two for the search locations in the following lines of the search window. After 16, the rotate amount repeats itself. For the search location shown as "X0" in Figure 3.23, the rotate amount is zero and the required reference data is in the first region (region 0) of all the BRAMs. For rotate amounts other than 0, 16, and 32, two different address values are sent to BRAMs. For the search location shown as "X6" in Figure 3.23, the rotate amount is six and the required reference data is in the first region (region 0) of BRAMs 6-15 and in the second region (region 1) of BRAMs 0-5.

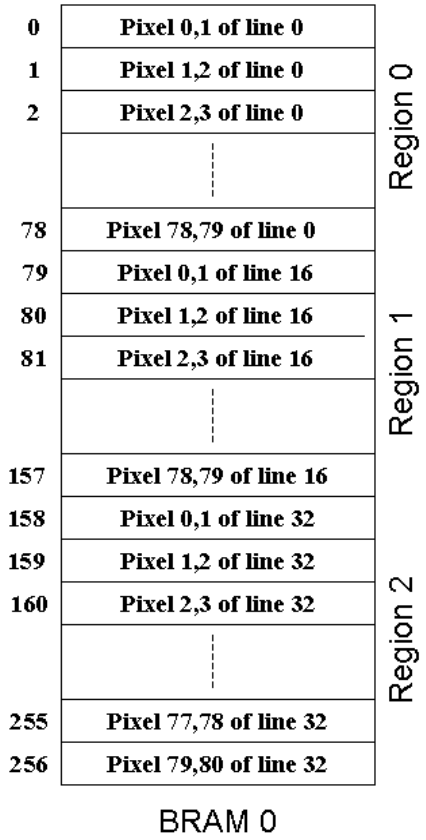


Figure 3.22 Pixel organization in BRAMs of the systolic architecture

	0		**X0**		0		0		0		0		0		0		0		0		0	
2		2		2		2		2		2		2		2		2		2		2		2
	4		4		4		4		4		4		4		4		4		4		4	
6		6		**X6**		6		6		6		6		6		6		6		6		6
	8		8		8		8		8		8		8		8		8		8		8	
10		10		10		10		O		10		10		10		10		10		10		10
	12		12		12		12		12		12		12		12		12		12		12	
14		14		14		14		14		14		14		14		14		14		14		14
	0		0		0		0		0		0		0		0		0		0		0	
2		2		2		2		2		2		2		2		2		2		2		2
	4		4		4		4		4		4		4		4		4		4		4	
6		6		6		6		6		6		6		6		6		6		6		6

Figure 3.23 Rotate amounts

The systolic hardware architecture is implemented in VHDL, verified with RTL simulations using Mentor Graphics Modelsim 6.3c and mapped to Xilinx XC3S1200-5 FPGA using Xilinx ISE 9.2.04. It can work at 144 MHz on this FPGA. Same as the generic architecture, for the largest search window size of (±32, ±16) pixels, it can process 206743 MBs per second. Therefore, it is capable of processing 127 fps, 57 fps, and 25 fps for 720x576, 1280x720, and 1920x1080 resolutions, respectively. The proposed systolic architecture consumes 6648 LUTs and 16 BRAMs. Because of the regular data flow, control unit consumes only 265 LUTs. Therefore, the systolic hardware fits into a state of the art low cost Xilinx Spartan-3E FPGA. Compared to the generic architecture, the systolic architecture uses smaller number of BRAMs and no horizontal rotators, and the input data width of the vertical rotator is reduced to 16 bits.

CHAPTER 4

DYNAMICALLY VARIABLE STEP SEARCH MOTION ESTIMATION ALGORITHMS AND A HARDWARE ARCHITECTURE FOR THEIR IMPLEMENTATION

We propose the DVSS and RDVSS ME algorithms for processing HD video formats [9, 10]. The proposed ME algorithms exploit MV correlations between neighboring MBs. We also propose a dynamically reconfigurable systolic ME hardware architecture for efficiently implementing these algorithms [9]. The proposed ME hardware is compared with several ME hardware implementations presented in the literature [26-31].

Several ME algorithms exploiting MV correlations between spatial and temporal neighboring MBs are proposed in the literature [32-38]. However, to the best of our knowledge, no ME algorithm utilizing the difference of the MVs of the temporal neighboring MBs as proposed in the RDVSS algorithm is presented in the literature. ARPS [20] and ADCS [21] algorithms adapt their initial search locations based on the MV of the previous MB. Adaptive Predicted Direction Search (APDS) [32] algorithm finds the initial search location by calculating the angles of the MVs of spatial and temporal neighboring MBs.

In [33], some of the candidate search locations are eliminated adaptively if their partial SAD value exceeds a dynamically determined threshold. In [34], the size and SAD values of the MVs of the previous blocks are used to adaptively change the search window size of the FS algorithm for the current block. The techniques proposed in [35, 36] are developed for fast ME algorithms which are not suitable for processing HD video. The dynamic adjustment of search window is a modification to the TSS algorithm and it adapts the search window size of the next step based on the result of

the previous step [35]. The dynamic adjustment of search window with variable size of block technique adaptively adjusts the search window and can be used with fast ME algorithms like NTSS and FSS [36]. In [36,37], MVs of upper, left, upper-left, and upper-right spatial neighboring MBs are used to determine the initial search location. In [37], in addition to MVs of these spatial neighboring MBs, MV of the temporal neighboring MB is also used for determining the initial search location. The algorithm proposed in [37] performs 7% close to the FS algorithm for low resolution videos where the search range is (±15, ±15) pixels. Since this ME algorithm performs hierarchical four levels of multi-resolution search with variable block size for each level and implements the FS algorithm for MBs where neighboring correlations are not available, its hardware implementation will be quite complex and it will perform significant number of memory accesses. In [38], if the spatial neighboring MBs of the current MB have identical MVs, this MV is used for the current MB as well without any search. This technique achieves good results only for low bit-rate video where search is performed in a very small search range, e.g. (±7, ±7) pixels, and therefore the MVs are similar.

4.1 Dynamically Variable Step Search Motion Estimation Algorithm

We propose the DVSS algorithm [9] in order to obtain a performance very close to the FS algorithm by searching even fewer search locations than the ME algorithms proposed in [7, 8]. The DVSS algorithm has a maximum of three different granularity search steps. First, the entire search window is searched with a coarse granularity search step. Then, two finer granularity search steps are performed around the search locations from previous steps with minimum SAD. The number of steps and the search range of each step are determined for the current block based on the size and the SAD value of the previously found MV for the left neighboring block. It is possible to use one of many different search patterns for a given block. Some of these search patterns, named as A1 [8], A2, A3, B and C, and the search patterns used in [7] are shown in Table 4.1. As shown in this table, skipping the coarse and medium steps and doing the fine step on the entire search range is identical to the FS algorithm. The search pattern A1, as shown in Figure 4.1, has 3 steps and the search ranges of coarse, medium, and fine steps are

(±48,±24), (±6,±6), (±3,±3) pixels, respectively. The search pattern A2 is the same as A1 except that the search range of its first step is (±24, ±12) pixels. The search pattern A3, as shown in Figure 4.2, has only medium and fine steps. In Figures 4.1 and 4.2, numbers represent the steps and shaded numbers show the search locations with minimum SAD for these steps.

Table 4.1 Several search patterns

Search Pattern	Search Range of First Step	Search Range of Second Step	Search Range of Third Step	Number of Search Locations
10x9 [7]	-	±10, ±9	±3, ±3	73
14x15 [7]	-	±14, ±15	±3, ±3	159
A1 [8]	±48, ±24	±6, ±6	±3, ±3	405
A2	±24, ±12	±6, ±6	±3, ±3	161
A3	-	±18, ±10	±3, ±3	249
32x16 [7]	-	±32, ±16	±3, ±3	553
B	±48, ±24	±12, ±12	±6, ±6	565
C	±48, ±24	±24, ±12	±12, ±6	793
48x24 [7]	-	±48, ±24	±3, ±3	1221
FS	-	-	±48, ±24	4753

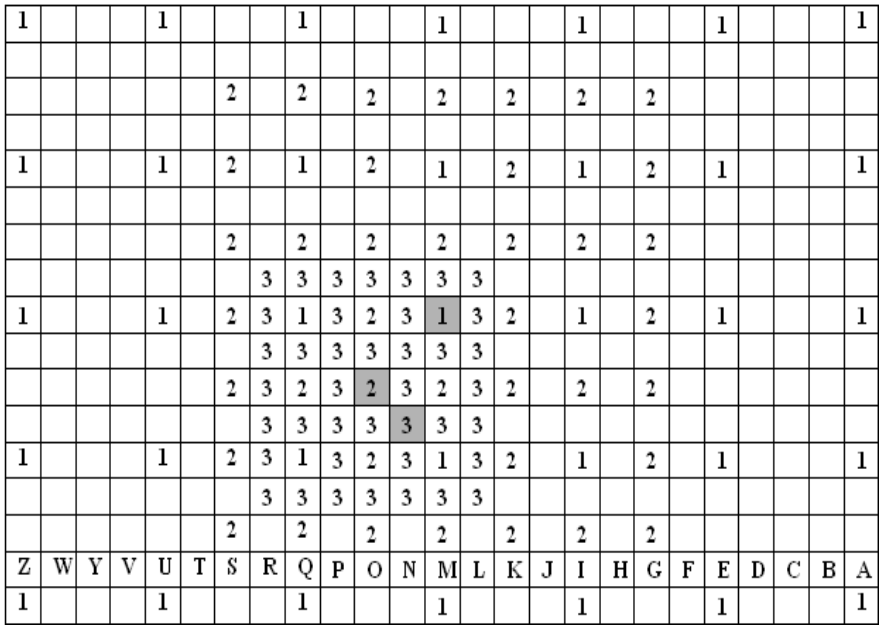


Figure 4.1 Search pattern A1

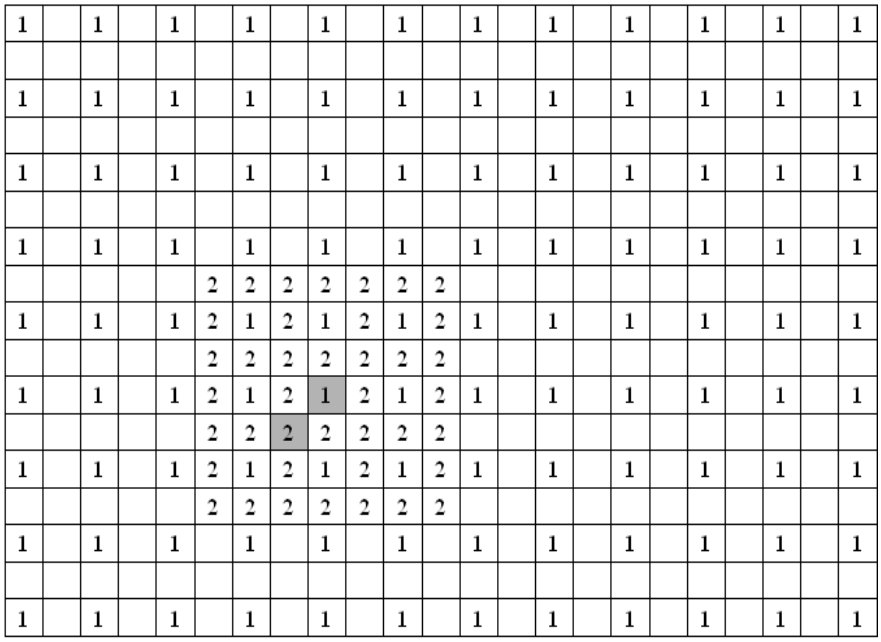


Figure 4.2 Search pattern A3

The number of steps and sizes of search ranges for each step determine the computational complexity of a search pattern and the MAD performance obtained by it. The DVSS algorithm decreases the computational complexity by adaptively changing between search patterns A1, A2, A3 for each block based on the size and SAD value of the previously found MV for the left neighboring block, which is called as Left Neighboring Motion Vector (LNMV). It uses FS, A3, A2, and A1 search patterns for small, medium, medium-to-large and large motions, respectively.

The pseudo code of the DVSS algorithm is shown in Figure 4.3. If LNMV falls within a smaller search range, it decreases the search granularity and the search range size, because for small motions doing the search in a smaller search range is sufficient and doing a finer granularity search in a smaller search range can give better MAD results. If the SAD value for LNMV is higher than a pre-determined threshold level ($\boldsymbol{\tau}$), it increases the search granularity and the search range size. The threshold level $\boldsymbol{\tau}$ is set to 256 and 1024 in our simulations. By setting $\boldsymbol{\tau}$ to a higher value, many search locations can be skipped and higher processing speeds can be achieved with a slight decrease in the MAD performance.

If there is no left neighboring block

Do pattern A1

Else if SAD value of LNMV exceeds the threshold (τ)

Switch to next coarser pattern

Else

If LNMV is within (±8, ±4) pixels

Do FS in a search range of (±10, ±5) pixels

Else if LNMV is within (±16, ±8) pixels

Do pattern A3

Else if LNMV is within (±24, ±12) pixels

Do pattern A2

Else

Do pattern A1

Figure 4.3 The pseudo code of the DVSS algorithm

The performances of the DVSS algorithm and its search patterns are compared with the performances of successful fast ME algorithms with respect to the MAD criterion and the results are shown in Table 4.2 and Table 4.3. Seven 100 frame video sequences are used for comparison, which are also used in Chapter 3.1 except the "Gladiator" video sequence. The "Gladiator" video is taken from the movie with the same name and it contains large motions. The frame size and rate of these benchmark videos are given in Tables 4.2 and 4.3. In the simulations, among the previously proposed fast search algorithms only the NTSS and the FSS algorithms have a search range of (±16, ±16) pixels. The other fast search algorithms have a search range of (±48, ±24) pixels. The FS is performed for both search ranges.

The simulation results showed that DVSS algorithm performs very close to the FS algorithm by searching much fewer search locations than the FS algorithm and it outperforms successful fast search ME algorithms by searching more search locations than these algorithms. The DVSS algorithm obtains similar performance results by searching fewer search locations than the search patterns proposed in Chapter 3.1. Even though, the FS algorithm with (±48, ±24) search range checks 4753 search locations in comparison to 405 search locations checked by the search pattern A1, its MAD performance is on the average only 7.5% better than the performance of the search

pattern A1. The performance of the FS algorithm with (±16, ±16) search range is very low for videos with large motion content.

Table 4.2 MAD results for fast search algorithms

Video Sequence (Frame Size & Rate)	FS ±48,±24	FS ±16,±16	NTSS [15]	FSS [16]	BBGDS [17]	DS [18]	HEXBS [19]	ARPS [20]	ADCS [21]	FTS [22]
Spiderman (720x576, 25fps)	4.20	6.96	10.71	10.81	7.47	7.20	7.37	6.07	6.24	6.87
Gladiator (720x576, 25fps)	2.83	5.38	8.68	8.79	5.68	5.43	5.61	3.93	3.73	6.00
IRobot (720x576, 25fps)	2.92	3.71	5.48	5.55	4.53	4.39	4.51	3.88	4.03	4.87
Susie (704x480, 15fps)	3.22	3.42	4.05	4.08	3.81	3.6	3.71	3.62	3.62	3.92
Flowers (704x480, 15fps)	8.39	8.41	10.47	11.12	10.6	10.31	10.9	8.70	8.95	13.11
Table Tennis (704x480, 15fps)	3.48	3.58	3.97	4.01	3.86	3.80	3.83	3.73	3.74	3.88
Foreman (352x288, 15fps)	4.17	4.23	4.81	4.86	4.51	4.56	5.08	4.54	4.69	5.69

Table 4.3 MAD results for proposed search algorithms

Video Sequence (Frame Size & Rate)	10x9 [7]	14x15 [7]	32x16 [7]	48x24 [7]	A1 [8]	B	C	DVSS τ = 256	DVSS τ = 1024
Spiderman (720x576, 25fps)	9.34	7.48	5.53	4.22	4.27	4.26	4.25	4.39	4.54
Gladiator (720x576, 25fps)	7,29	5,84	3,32	2.88	2.97	2.93	2.92	3.14	3.26
IRobot (720x576, 25fps)	7.69	6.93	5.72	3.08	3.23	3.15	3.10	3.29	3.33
Susie (704x480, 15fps)	3.92	3.72	3.40	3.33	3.41	3.34	3.32	3.29	3.29
Flowers (704x480, 15fps)	8.89	8.79	8.62	8.61	9.26	9.06	8.95	8.51	8.48
Table Tennis (704x480, 15fps)	3.79	3.66	3.56	3.51	3.57	3.55	3.54	3.55	3.57
Foreman (352x288, 15fps)	5.02	4.95	4.67	4.66	4.87	4.70	4.60	4.51	4.39

The performance gap between fast search algorithms and the proposed search patterns increase with increased video resolution and motion between consecutive frames. On the other hand, as it can be seen from “Foreman” benchmark video, when the resolution is very low and the motion can be detected in a search range of (±16, ±16) pixels, the performance gap decreases. The DVSS algorithm decreases the computational complexity significantly with a small decrease in the MAD performance. It even sometimes gives better MAD results than the pattern A1. The reason for this improvement is that search patterns with finer granularities perform better for small motions and the DVSS algorithm dynamically decreases its granularity when small MVs are found for the previous blocks.

4.2 Reconfigurable Motion Estimation Hardware Architecture

The reconfigurable systolic ME hardware architecture is based on the ME hardware presented in Chapter 3.3. The major differences between them are the proposed hardware is dynamically reconfigurable and it implements the DVSS algorithm. For each MB, the proposed ME hardware can be dynamically reconfigured to execute different number of steps and different search ranges for each step. Top-level block diagram of the proposed ME hardware architecture is shown in Figure 4.4. The hardware is highly pipelined and its latency is eight clock cycles; one cycle for synchronous read from memory, one cycle for shift registers, two cycles for the reconfigurable systolic PE array and four cycles for the adder tree.

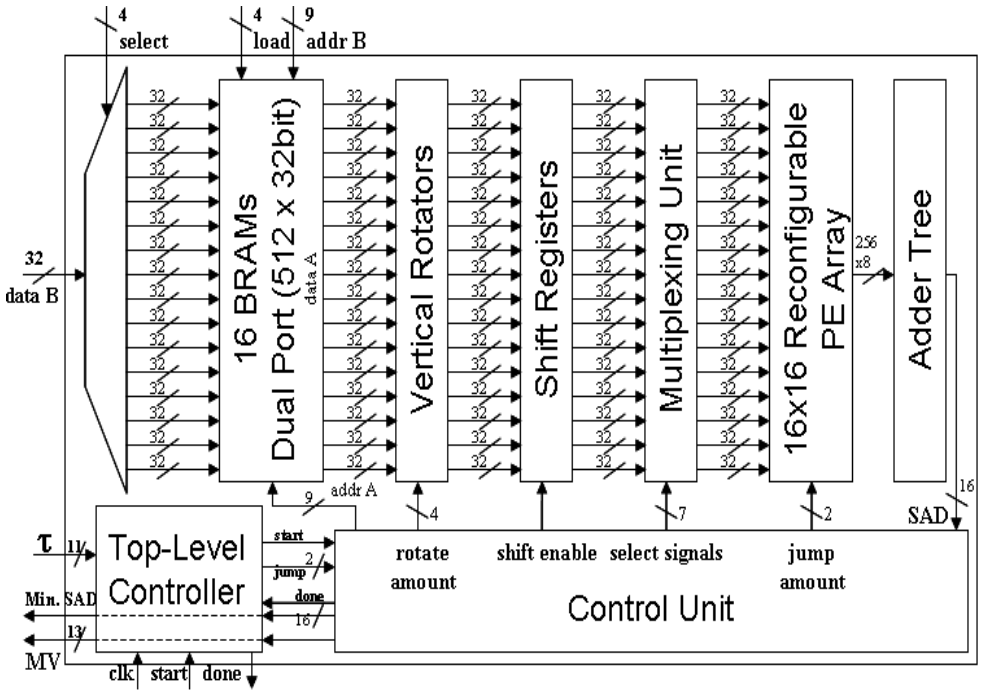


Figure 4.4 Top-level block diagram

The proposed ME hardware finds an MV for a 16x16 MB based on the minimum SAD criterion in a maximum search range of (±48, ±24) pixels using the luminance data. The "top-level controller" takes the threshold level (τ) as an input and determines the number of search steps and their search ranges for each block adaptively. The "control unit" finds the MV for each block by generating required address and control

signals to compute the SAD values of the search locations in the search window determined by the top-level controller for each step.

The search locations in a search window are searched line by line. First, SAD values of the search locations in the top line of the search window are calculated starting from the right most search location in the top line. Then, SAD values of the search locations in the next line of the search window are calculated starting from the right most search location in the next line. The first step ends after SAD values of the search locations in the bottom line of the search window are calculated. The next step around the search location with the minimum SAD is done in the same way.

16 BRAMs in the FPGA are used to store the search window. BRAMs are configured as dual port memories for overlapping the ME of the current MB with the loading of the search window of the next MB. The vertical rotator is used to align the outputs of the BRAMs and it has 32 identical rotators each 16 bits long. The reference MB data read from BRAMs must be matched with the current MB data, which is loaded into the PE array previously, by rotating the data lines. For example, for the search locations in the fourth line of the search window, the rotate amount will be equal to four so that first line of the reference data will be read from the fourth BRAM.

The SAD value for a search location is calculated by summing the outputs of all 256 PEs in the reconfigurable PE array by an adder tree. The adder tree has four pipeline stages; SAD values of 4x4 blocks are calculated in the first two clock cycles, in the third clock cycle SAD values of 8x8 blocks are calculated and in the fourth clock cycle SAD value of 16x16 MB is calculated.

The reconfigurable systolic PE array is shown in Figure 4.5. 256 PEs are used to calculate the SAD of a 16x16 MB. A PE is used to calculate the absolute difference between a current pixel and the corresponding reference pixel. The latency of the PE array is two clock cycles, because reference and current pixel inputs and the absolute difference output are registered. The reconfiguration of the PE array is achieved with the multiplexers placed between the PEs that process the same line in a MB. Since the PE array explained in Chapter 3.3 is not reconfigurable, these multiplexers bring a slight area overhead in comparison to the PE array proposed in Chapter 3.3. But, they

do not affect the clock frequency since they are not placed on the critical path. In Figure 4.5, interconnects used for implementing 4, 2 and 1 shift amounts are illustrated with dashed, thin and bold lines respectively. Interconnects marked with “m” are connected to BRAMs.

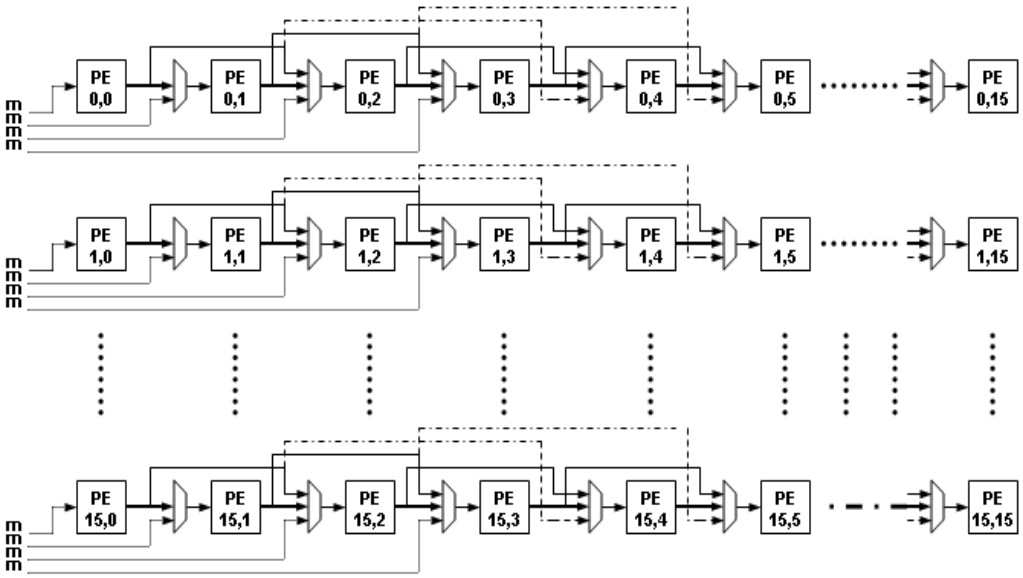
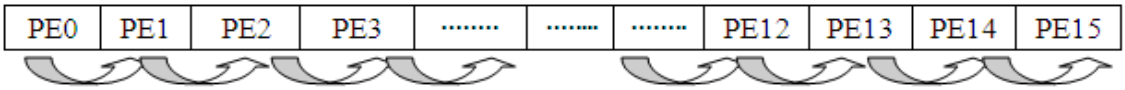
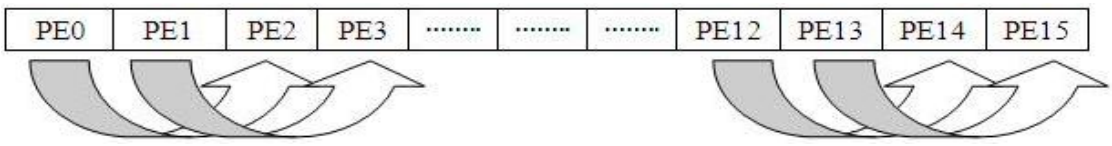


Figure 4.5 Reconfigurable systolic PE array

The reference pixels for the first search location in a line of the search window are loaded in four clock cycles. After the SAD value of the first search location is calculated, the SAD value of the next search location is calculated in one cycle. After the SAD value of the first search location is calculated, reference data is shifted to the right in the PE array in each consecutive clock cycle and shift amount can be 4, 2 or 1 pixels depending on the type of the step; coarse, medium or fine, respectively. Figure 4.6 demonstrates the shifting in the PE array when the shift amount is equal to 1 and 2 pixels. For example, when the shift amount is equal to 2 pixels, PE0 shifts its content to PE2, PE1 shifts its content to PE3, and PE2 shifts its content to PE4.



(a)



(b)

Figure 4.6 Shifting in PE array (a) 1 pixel, (b) 2 pixels

The data flow through the reconfigurable systolic PE array is shown in Table 4.4. Let capital letters “A” to “Z” shown in Figure 4.1 denote all pixels in these columns respectively. Assuming that “P” is the first search location, four clock cycles will be required to feed the reference data for this search location to the PE array, because regardless of the search pattern during the loading of reference pixels for the first search location the multiplexing unit feeds first four columns of the PE array. Assuming that after “P”, the search pattern continues with search locations “R, T and V” (two pixel gap between consecutive search locations), multiplexing unit will feed only first two columns of the PE array. Therefore, reference pixels for these search locations will be in the PE array in 5th, 6th and 7th clock cycles, respectively.

Table 4.4 Dataflow through the reconfigurable systolic PE array

Clock Cycle	Processing Elements															
	Col 0	Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8	Col 9	Col 10	Col 11	Col 12	Col 13	Col 14	Col 15
1	D	C	B	A												
2	H	G	F	E	D	C	B	A								
3	L	K	J	I	H	G	F	E	D	C	B	A				
4	P	O	N	M	L	K	J	I	H	G	F	E	D	C	B	A
5	R	Q	P	O	N	M	L	K	J	I	H	G	F	E	D	C
6	T	S	R	Q	P	O	N	M	L	K	J	I	H	G	F	E
7	V	U	T	S	R	Q	P	O	N	M	L	K	J	I	H	G

In order to calculate the SAD values of search locations at the rate of one SAD value per clock cycle, pixels for a particular search location must be brought to the PE array in one clock cycle, and this requires many accesses to the memory in the same clock cycle. This memory requirement cannot be satisfied by an FPGA without data-reuse. The systolic hardware architecture proposed in Chapter 3.3 reduces the internal memory bandwidth by applying data-reuse and it uses only 16 BRAMs for storing the reference pixels of a search window for a search range of (±32, ±16) pixels. BRAMs are configured as 16 bits wide because of the two pixel distance between consecutive search locations.

The ME hardware proposed in this chapter also applies data-reuse. However, it uses only 16 BRAMs for storing the reference pixels of a search window for a search range of (±48, ±24) pixels. The proposed ME hardware further reduces the internal memory bandwidth by feeding only 64 PEs from BRAMs, the remaining PEs receive

reference pixels from neighboring PEs. BRAMs are configured as 32 bits wide and they are connected to the four left end columns of the PE array. Therefore, loading the reference pixels for the first search location into the PE array takes four clock cycles.

Each BRAM stores four lines of reference pixels. Storing a line of reference pixels uses 28 address locations. Therefore, addresses 0-111 are occupied to store four lines of reference pixels. Figure 4.7 shows the layout of the reference pixels in the first BRAM, which stores 0th, 16th, 32th and 48th lines of the reference pixels in four distinct regions. The remaining BRAMs have the same organization.

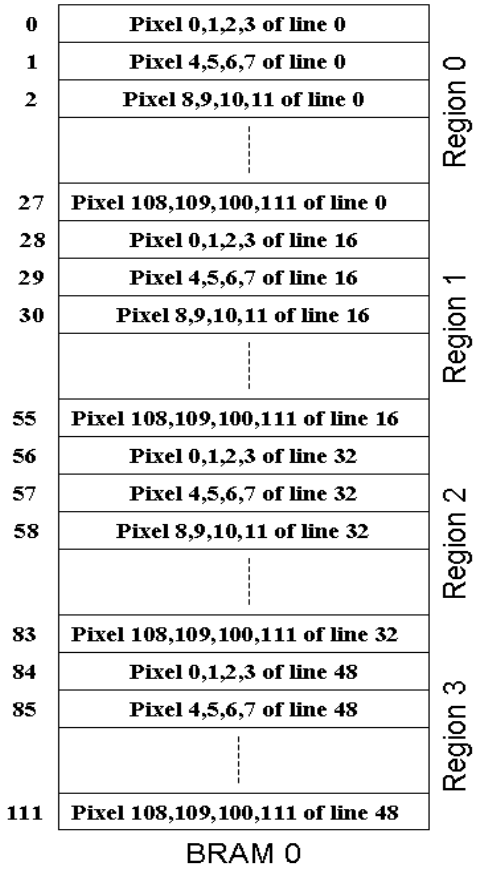


Figure 4.7 Memory organization

The “multiplexing unit”, shown in Figure 4.8, is used to feed the correct data to the PE array. The data received from the vertical rotator is captured in a 56 bit long shift register, which stores 7 pixels. If the enable signal of the shift register is high, it shifts its content 32 bits to right. In order to support horizontal distances of one, two, and four between consecutive search locations, multiplexing unit is designed to feed first one, two, or four left end columns of the PE array. Independent from the search pattern,

reference pixels for the first search location are loaded by feeding the four columns. Therefore, four clock cycles are required to fill the PE array with the reference pixels for the first search location. The reference pixels for the next search location will be available in the next clock cycle. If the distance between two search locations is four pixels, “4 select” multiplexers otherwise “2 select” multiplexers are used to select the corresponding reference pixels from the shift register. Table 4.5 shows the output of the multiplexing unit for different pixel locations. The content of the shift register, which is shown with capital letters in Figure 4.8, is also given in Table 4.5. If the search location is aligned with the memory content, the most significant four bytes (G, F, E, D in Figure 4.8) will be selected as the output. Otherwise 1, 2, or 3 pixel shift will be performed.

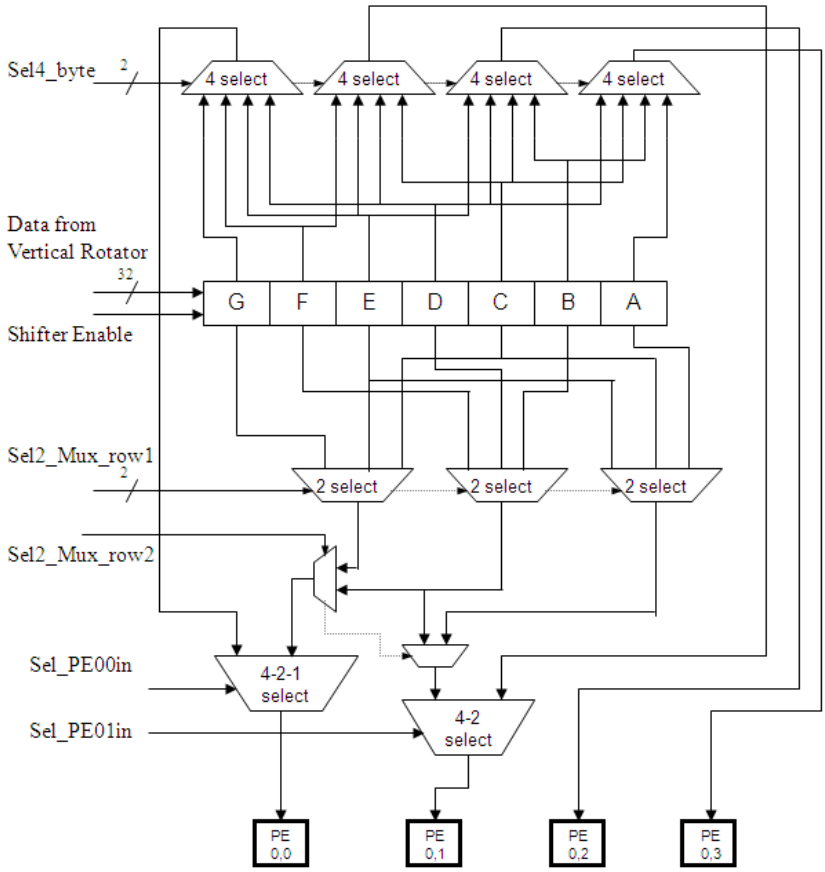


Figure 4.8 Multiplexing unit

Table 4.5 Output of the multiplexing unit for different pixel locations

Clock Cycle	Shift Register Content	Aligned Out	1 Pixel Shifted Out	2 Pixel Shifted Out	3 Pixel Shifted Out
1	D C B A - - -	D C B A			
2	H G F E D C B	H G F E	E D C B	F E D C	G F E D
3	L K J I H G F	L K J I	I H G F	J I H G	K J I H
4	P O N M L K J	P O N M	M L K J	N M L K	O N M L

The proposed hardware architecture is implemented in VHDL, verified by RTL simulation using Modelsim 6.3c, and mapped to an XC3S1500-5 FPGA using Synplify Pro 8.9 and ISE 10.1. The proposed hardware works at 130MHz and consumes 9128 slices (2282 CLBs) and 16 BRAMs. The reconfigurable systolic PE array with the adder tree consumes 7510 slices.

The number of clock cycles per MB required by the proposed hardware depends on the search pattern. Starting a step has a start-up cost of 15 clock cycles, which is called as the step latency, and starting the search on a line has a start-up cost of 8 clock cycles, which is called as the line latency. The total number of clock cycles per MB required to complete a search pattern is given by (4.1). The performance of proposed ME hardware for several search patterns are calculated based on (4.1) and given in Table 4.6.

$$\sum_{1}^{n_s}\left(\tau_s + \left(n_{sad} - 1\right)\times n_{line} + \left(n_{line} - 1\right)\times \tau_{line}\right) \qquad (4.1)$$

In (4.1), "n_s, n_{sad}, n_{line}" are the number of steps, search locations per line, and lines per step, respectively. "τ_s" and "τ_{line}" are step and line latencies, respectively. Based on this equation, for the coarse, medium and fine steps the start-up latency is 45 clock cycles. For these three steps, there is 192 clock cycles of line latency and 396 clock cycles are required for remaining search locations. Therefore, pattern A1 requires 633 clock cycles to find the MV of a MB. Patterns A2 and A3 requires 357 and 380 clock cycles, respectively. FS with a search range of (±10, ±5) pixels requires 304 clock cycles.

The performance of the DVSS algorithm on the proposed ME hardware for different threshold values is shown in Table 4.7. The DVSS algorithm achieves much better real-time performance, with a small decrease in the MAD performance, since it adaptively changes the search patterns and uses the pattern A1 only for large motions, patterns A2 and A3 for medium motions and FS only for small motions. As it can be seen in Table 4.7, increasing the threshold value increases the supported frame rate.

Table 4.6 Performance of the proposed hardware for several search patterns

Search Pattern	Required Clock Cycles per MB	Processed MBs per Second	Supported Frame Size & Rate
A1 [8]	633	205371	1920x1080, 25.3 fps
B	957	135841	1366x768, 33.1 fps
C	1221	106470	1366x768, 25.9 fps
10x9 [7]	122	1180327	1920x1080, 145.7 fps
14x15 [7]	236	610169	1920x1080, 75.3 fps
32x16 [7]	672	214285	1920x1080, 26.4 fps
48x24 [7]	1425	101052	1366x768, 24.6 fps
FS	5103	25475	720x576, 15.7 fps

Table 4.7 Performance of the proposed hardware for the DVSS algorithm

Video Sequence	Threshold (τ)	Required Cycles for 100 Frames	MBs per Frame	Average Cycles per MB	Supported 1920x1080 fps
Spider	256	96094246	1620	594	27.0
Spider	1024	90284377	1620	558	28.7
Gladiator	256	87299334	1620	539	29.7
Gladiator	1024	80952068	1620	500	32.1
Irobot	256	77966499	1620	482	33.3
Irobot	1024	74177157	1620	458	35.0
Susie	256	59212520	1320	449	35.7
Susie	1024	51666864	1320	392	41.0
Flowers	256	52181938	1320	396	40.5
Flowers	1024	49586582	1320	376	42.7
TableTennis	256	53382291	1320	405	39.6
TableTennis	1024	47136775	1320	358	44.9
Foreman	256	15926153	396	403	39.9
Foreman	1024	14250681	396	360	44.5

The proposed ME hardware is compared with several ME hardware implementations presented in the literature in Table 4.8. The proposed ME hardware consumes less area than the implementation of one of the best performing fast search ME algorithms in the same FPGA [22]. The MAD performance of this hardware is lower than the MAD performance of the proposed ME hardware, since it implements the FTS algorithm. In [26], a hybrid architecture supporting both FS and DS is presented. This architecture speeds up FS by successively eliminating some of the search locations. In addition, it is suitable for the irregular data flow of fast search algorithms and it consumes less area than the dedicated FS systolic array

implementations. However, it has lower throughput than the proposed ME hardware.

Because of the overhead of the reconfigurability and additional complexity of the control unit, the proposed ME hardware consumes 2363 slices more than the ME hardware proposed in [7] in the same FPGA. 1136 slices are used by the multiplexing unit, 836 additional slices are used by the multiplexers in the PE array and the remaining additional slices are used by the additional complexity of the control unit. Because of the overhead of the dynamic reconfigurability, which is implemented in the top-level controller, the proposed ME hardware consumes slightly more area than the ME hardware proposed in [8] in the same FPGA.

The throughput of the proposed ME hardware is much higher than the FS hardware implementations in [27,28]. An Application Specific Integrated Circuit (ASIC) implementation of the FS algorithm utilizing 256 PEs in 0.25μm CMOS technology is given in [27]. This architecture is a modified version of the AB2 type systolic array [29]. Another ASIC implementation of the FS algorithm is given in [28]. The throughput of this architecture is low, because it has only 64 PEs, it is optimized for low power consumption and it is implemented in an older technology. A real-time ME hardware implementing the FS algorithm for HD video is given in [30]. However, since this hardware is implemented on a high-end FPGA, it is not suitable for consumer electronics products. The FPGA implementations of the systolic architectures AS1, AB2, AS2 are presented in [31]. Despite using large number of PEs, the throughputs of these ME hardware are much lower than the throughput of the proposed ME hardware, because they are implementing the FS algorithm. The area results presented in [31] include only the datapath and do not include the control unit and the memory.

Table 4.8 Comparison of ME hardware architectures

HW	Algorithm	Technology	MB size	# of PEs	Search Range	Area	Speed [MHz]	Cycles per 16x16 MB	Supported 1920x1080 [fps]
[9]	DVSS	XC3S1500-5 FPGA	16x16	256	(±48, ±24)	2282 CLBs	130	467 (τ = 256)	34.3 (τ = 256)
[22]	FTS	XC3S5000 FPGA	16x16	16	(±16, ±16)	6142 CLBs	74	202	45.2
[26]	FS & DS	Unknown	8x8, 16x16	Dedicated HW	(-16, +15) in both axis	9K gates	50	2879 (average)	2.1
[7]	32x16 [7]	XC3E1200E-5 FPGA	16x16	256	(±32, ±16)	1692 CLBs	144	672	26.4
[8]	A1 [8]	XC3S1500-5 FPGA	16x16	256	(±48, ±24)	2271 CLBs	130	633	25.3
[27]	FS	0.25µm CMOS 1P5M	16x16	256	(-16, +15) in both axis	16.07 mm2	36	1421	3.1
[28]	FS	0.6µm SPTM CMOS	8x8, 16x16, 32x32	64	(±32, ±32)	267K gates	60	4209	1.7
[30]	FS	XC4VLX100 FPGA	16x16	Dedicated HW	(±16, ±16)	380 LUTs	221	1111	24.5
AS1 [31]	FS	XC40250 FPGA	16x16	33	(±16, ±16)	1214 CLBs	24	25344	0.1
AB2 [31]	FS	XC40250 FPGA	16x16	256	(±16, ±16)	948 CLBs	30	1584	2.3
AS2 [31]	FS	XC40250 FPGA	16x16	528	(±16, ±16)	3732 CLBs	22	768	3.5

4.3 Recursive Dynamically Variable Step Search Motion Estimation Algorithm

The proposed RDVSS [10] algorithm searches fewer search locations than the DVSS algorithm for the same size search window. RDVSS dynamically determines the search patterns that will be used for each MB based on the MVs of its spatial and temporal neighboring MBs assuming that objects are bigger than a MB and motion between consecutive frames is continuous. By using a larger search range, the RDVSS algorithm gives better PSNR results than the DVSS algorithm for the benchmark videos with large motions. For the benchmark videos with smaller motions, the DVSS algorithm gives slightly better PSNR results by checking more search locations in the same search range. The RDVSS algorithm gives much better PSNR results than fast search ME algorithms. In addition, the RDVSS algorithm has a regular data flow and it can be efficiently implemented using the reconfigurable ME hardware architecture proposed in this chapter.

The search patterns used in the RDVSS algorithm are listed with their search ranges and total number of search locations in Table 4.9. Similar to the DVSS algorithm, each search pattern has a maximum of three different granularity search steps with different size search ranges. In the first, second, and third steps, horizontal and

vertical distances between search locations are 4, 2, and 1 pixels, respectively. Figure 4.9 shows a portion of main large search pattern. In this figure, numbers represent the steps and dark shaded numbers show the search locations with minimum SAD for these steps. Main large search pattern, first, searches the given (±64, ±64) search window with a coarse granularity search step. It, then, performs a finer granularity search step in the (±4, ±4) search window around the search location that has the minimum SAD in the previous step. It, finally, performs an even finer granularity search step in the (±1, ±1) search window around the search location that has the minimum SAD in the previous step.

Table 4.9 Search patterns used in the RDVSS algorithm

Search Pattern	**Search Range of First Step**	**Search Range of Second Step**	**Search Range of Third Step**	**Number of Search Locations**
Main Large	±64, ±64	±4, ±4	±1, ±1	1113
A1 [8]	±48, ±24	±6, ±6	±3, ±3	405
Main Medium	±32 ±32	±4, ±4	±1, ±1	313
Main Small	±16, ±16	±4, ±4	±1, ±1	161
Recursive Large	±16, ±16	±2, ±2	±1, ±1	97
Recursive Medium	±8, ±8	±2, ±2	±1, ±1	41
Recursive Small	-	±4, ±4	±1, ±1	33
3x3 Full Search	-	-	±3, ±3	49
1x1 Full Search	-	-	±1, ±1	9

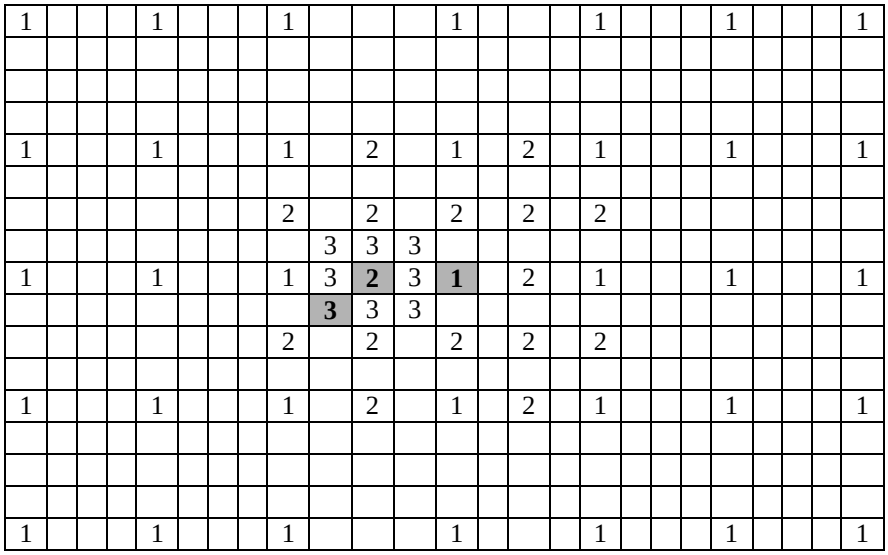


Figure 4.9 Main large pattern

The pseudo code of the RDVSS algorithm is given in Figure 4.10. The RDVSS algorithm performs three search iterations for each MB. The first iteration is used for tracking global motions like camera movement assuming that motion between consecutive frames is continuous. The second iteration is used for tracking complex motions of objects assuming that objects are larger than a MB. If the first and second iterations do not find a satisfactory MV, main search patterns with large search ranges are used around (0,0) location for finding a better MV.

The RDVSS algorithm determines the search patterns that will be used in each iteration for the current MB dynamically based on the MVs of its spatial and temporal neighboring MBs. After performing each search pattern for the current MB, the RDVSS algorithm compares the minimum SAD obtained so far with the SAD threshold determined for this MB and it terminates the ME for this MB if the SAD is less than the SAD threshold. Therefore, for each MB, the RDVSS algorithm calculates Spatial Difference (SD), Average Spatial Neighboring MV (ASNMV), Temporal Distance (TD) and SAD Threshold (ST) by using the MVs of its available spatial neighboring MBs. Figure 4.11 shows the spatial neighboring MBs of MB(i,j,t), where "i" and "j" denote the x and y coordinates of the MB in a frame and "t" denotes the frame containing this MB. Therefore, for example, only the left spatial neighboring MB is available for the MBs in the first row of a frame.

SD is the maximum absolute difference in the x and y coordinates of MVs of four spatial neighboring MBs; MB(i-1,j-1,t), MB(i,j-1,t), MB(i+1,j-1,t), and MB(i-1,j,t). As shown in (4.2), ASNMV is the average of the MVs of these four spatial neighboring MBs. As shown in (4.3), ST is determined by comparing the minimum SAD value of these four spatial neighboring MBs with the pre-determined SAD threshold for the video frame ($\boldsymbol{\tau}$) and selecting the larger one.

$$ASNMV = \frac{1}{4}\left[MV(i-1,\, j-1,t) + MV(i,\, j-1,t) + MV(i+1,\, j-1,t) + MV(i-1,\, j,t)\right] \quad (4.2)$$

$$ST = MAX\left[\tau, MIN\left[SAD(i-1,\, j-1,t), SAD(i,\, j-1,t), SAD(i+1,\, j-1,t), SAD(i-1,\, j,t)\right]\right] \quad (4.3)$$

Iteration 1:

If (TD is equal or less than (±4,±4) pixels)

Do *Recursive Small Pattern* around MV(i,j,t-1)

Else if (TD is equal or less than (±8,±8) pixels)

Do *Recursive Medium Pattern* around MV(i,j,t-1)

Else if (TD is equal or less than (±16,±16) pixels)

Do *Recursive Large Pattern* around MV(i,j,t-1)

Else

Do *1x1 Full Search Pattern* around MV(i,j,t-1)

Iteration 2:

If (SD is equal or less than (±3,±3) pixels)

Do *3x3 Full Search Pattern* around ASNMV

Else

Do *1x1 Full Search Pattern* around MV(i-1,j-1,t), MV(i,j-1,t), MV(i+1,j-1,t), and MV(i-1,j,t)

Iteration 3:

If (SD is equal or less than (±16,±16) pixels)

Do *Main Small Pattern* around (0,0)

Else if (SD is equal or less than (±32,±32) pixels)

Do *Main Medium Pattern* around (0,0)

Else

Do *Main Large Pattern* around (0,0)

Until (*Main Large Pattern* is used)

Do next larger Main Pattern around (0,0)

Figure 4.10 Pseudo code of the RDVSS algorithm

Figure 4.11 Spatial neighboring MBs of MB(i,j,t)

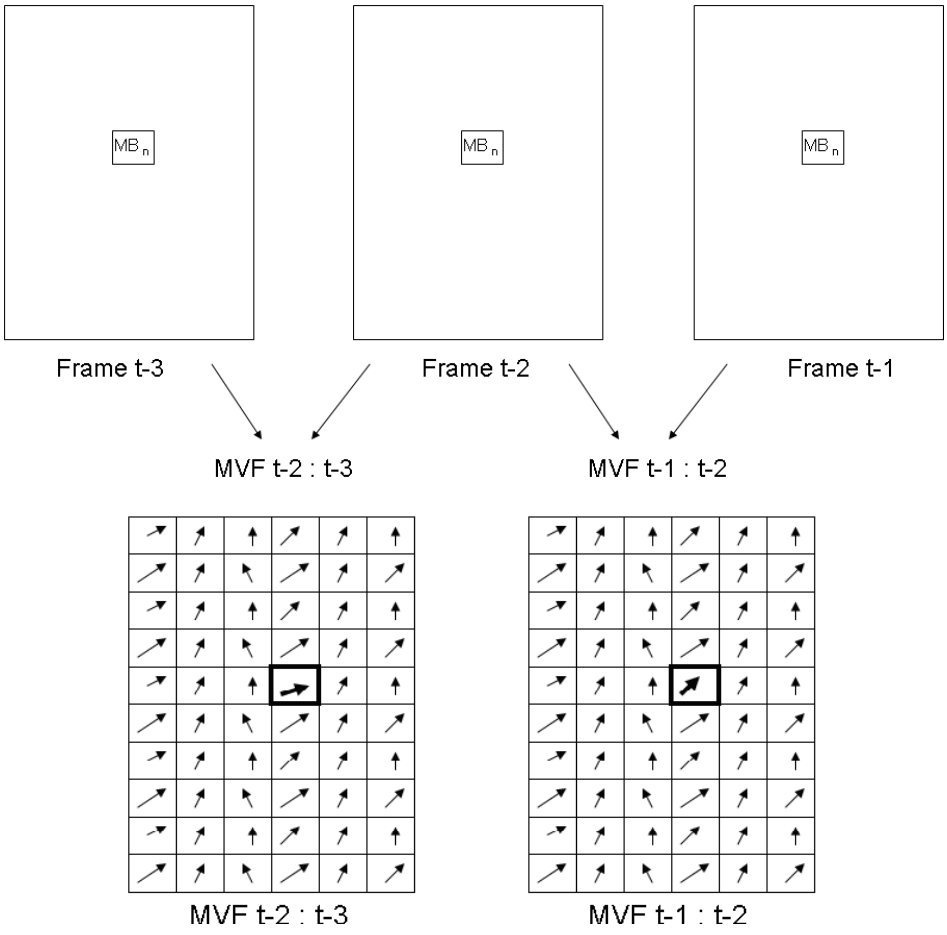


Figure 4.12 Temporal correlation

Figure 4.12 shows three consecutive frames and their MVFs. “MVF t-2 : t-3” is obtained by performing ME between the frames at the time instances “t-2” and “t-3”, and “MVF t-1 : t-2” is obtained by performing ME between the frames at the time instances “t-1” and “t-2”. TD is the difference between MV(i,j,t-2) and MV(i,j,t-1). Therefore, while processing previous frame “t-1”, for each MB, its MV in the “MVF t-1 : t-2” and a two bit value indicating whether its TD is equal or less than (±4,±4), (±8,±8), (±16,±16) pixels or not should be stored in a memory. In Figure 4.12, TD value

for MB_n will be calculated by finding the difference between the two MVs shown with bold lines in the consecutive MVFs.

The RDVSS algorithm is compared with the successful fast ME algorithms for several video sequences with respect to the MAD criterion and the comparison results are shown in Table 4.10. RDVSS is simulated for various SAD thresholds ($\boldsymbol{\tau}$) to show the trade-off between the obtained image quality and the number of search locations. The number of search locations checked by the RDVSS algorithm for these video sequences are shown in Table 4.11. The luminance components of eight video sequences with various resolution and frame rates are used for the comparison. The resolution and frame rates of these video sequences are given in Table 4.11. Among these videos "IceAge2", "ParkJoy1080p", "Ducks", and "ParkJoy720p" are 50 frames long and the other videos are 100 frames long. "ParkJoy1080p", "Ducks", and "ParkJoy720p" HD video sequences are available from Video Quality Experts Group [39]. These videos contain complex but slow motion. "IceAge2", "Spider3", and "Spider2" video sequences are taken from "Ice Age 2", "Spiderman 3", and "Spiderman 2" movies where there are fast and complex movements. "Susie" and "Table Tennis" video sequences are the up-scaled versions of the widely used CIF resolution benchmark videos.

In our simulations, only the NTSS and the FSS algorithms have a search range of (±16, ±16) pixels because their initial step size is equal to 8. The other ME algorithms have a search range of (±64, ±64) pixels. The threshold value required for the ADCS algorithm is set to 1024. Since the weights used in the APDS algorithm are not specified in [32], we set them to 1. As shown in Table 4.10, the RDVSS algorithm obtains better results than the well known fast ME algorithms. The performance gap between the RDVSS and other ME algorithms increase with increased motion between consecutive frames. Although the RDVSS algorithm on the average searches 34.1% to 62.4% less search locations than "Main Large" search pattern, it obtains similar MAD results with the "Main Large" search pattern. If only the early search termination is used for the "Main Large" search pattern without using the spatial and temporal correlations, MAD results decrease significantly, especially for videos containing fast motion. When compared with the DVSS algorithm for a maximum search range of (±48, ±24) pixels and for the same threshold level ($\boldsymbol{\tau}$=256), RDVSS searches 34% less search locations on

the average while giving better PSNR results for videos containing large motions. For videos containing very small motions, the DVSS algorithm gives slightly better results by checking more search locations.

Table 4.10 MAD results

Video Sequence	**FS**	**NTSS [15]**	**FSS [16]**	**BBGDS [17]**	**DS [18]**	**HEXBS [19]**	**A1 [8]**	**Main Large**
ParkJoy1080p	8.91	12.77	13.51	13.57	12.85	12.99	9.86	9.24
IceAge2	2.52	8.16	8.22	5.21	5.08	5.35	4.20	2.95
Ducks	3.81	5.26	5.44	5.29	5.27	5.40	5.07	4.93
ParkJoy720p	8.43	12.45	12.58	12.97	12.05	12.36	10.18	9.64
Spider3	2.44	8.21	8.30	5.30	5.21	5.39	3.30	2.81
Spider2	2.96	10.72	10.82	7.09	6.94	7.08	4.28	3.07
Susie	3.17	4.05	4.09	3.81	3.62	3.69	3.51	3.51
Table Tennis	3.42	3.97	4.01	3.86	3.80	3.83	3.57	3.55

Video Sequence	**APDS [32]**	**ARPS [20]**	**ADCS [21]**	**DVSS τ = 256**	**DVSS τ = 1024**	**RDVSS τ = 256**	**RDVSS τ = 512**	**RDVSS τ = 1024**
ParkJoy1080p	13.70	10.82	10.37	9.02	9.07	9.43	9.54	9.64
IceAge2	5.21	3.97	4.97	4.29	4.79	3.15	3.39	3.92
Ducks	5.24	5.27	5.39	5.00	5.02	5.07	5.07	5.08
ParkJoy720p	12.99	10.70	10.22	9.01	9.17	9.92	10.04	10.16
Spider3	5.34	3.65	3.68	3.36	4.39	2.88	3.04	3.54
Spider2	7.10	5.39	5.02	4,39	4,53	3.11	3.21	3.82
Susie	3.96	3.58	3.58	2,99	2,99	3.47	3.51	3.85
Table Tennis	3.89	3.71	3.72	2,76	2,77	3.55	3.71	3.72

Table 4.11 Average number of search locations per MB

Video Sequence	RDVSS τ = 256	RDVSS τ = 512	RDVSS τ = 1024
ParkJoy1080p (1920x1080, 25fps)	959	933	738
IceAge2 (1920x1080, 25fps)	601	448	301
Ducks(1280,760, 25fps)	380	372	366
ParkJoy720p (1280x720, 25fps)	921	805	723
Spider3 (1280x576, 25fps)	529	429	322
Spider2 (720x576, 25fps)	843	660	327
Susie (704x480, 15fps)	850	729	365
Table Tennis (704x480, 15fps)	782	716	204

The RDVSS algorithm searches much less search locations than the FS algorithm. The FS algorithm checks 16641 search locations in a search range of (±64, ±64) pixels, whereas the RDVSS on the average checks only 418 search locations, when the SAD threshold ($\boldsymbol{\tau}$) is set to 1024. On the other hand, MAD performance of the RDVSS algorithm on the average is only 14.7% lower than MAD performance of the FS algorithm, when the SAD threshold ($\boldsymbol{\tau}$) is set to 256. Performing that close to the FS algorithm for such a large search window is very important.

CHAPTER 5

COMPUTATION REDUCTIONS FOR VECTOR MEDIAN FILTERING

VMFs are widely used in image and video processing applications [5]. VMFs are non-linear filters and they require dealing with multi dimensional data. Because of their edge-preserving characteristics, they are mainly used for removing the noise from a signal by smoothing out the signal. Because of their smoothing capability, they are also used in video compression [40-43]. In [40], vector median filtering is applied adaptively on the obtained MVF in order to improve the visual quality and in [41] a VMF is used to estimate the MVs based on previously found MVs. In [42], by using adaptively weighted VMF in the encoder, a smoother MVF is obtained. In [43], VMF is applied at the decoder to smooth out irregular MVs. Recently, VMFs are used for FRC [44-52].

In order to achieve high quality results for FRC, the true motion between consecutive frames should be found [44-52]. While ME for video compression needs to find the MVs giving the minimum SAD, ME for FRC should find the MVs corresponding to the physical motion of the objects. In order to find the true motion between consecutive frames, VMFs are used to smooth the MVF obtained by the ME. An example of smoothing an MVF is shown in Figure 5.1. In this example, the MV in the middle of the 3x3 filtering window is replaced by the output of the VMF applied to 9 MVs in this 3x3 filtering window.

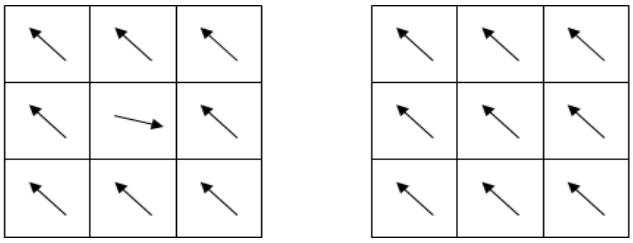


Figure 5.1 Smoothing MVF

A frame from the Foreman video sequence and its MVF found by the FS algorithm are shown in Figure 5.2. FS is implemented in a search range of (±8, ±8) pixels for 16x16 MB size. The original MVF and the smoothed MVF by 3x3 VMF are shown in Figure 5.3. The man in the video sequence shakes his head and most of the corresponding MVs in the MVF point to vertical direction. However, some of these MVs point to horizontal direction. Smoothing the MVF by applying the VMF corrects some of the outlier MVs. For example, after the VMF operation, the MVs of the MBs containing the face of the man become more accurate. MVs on the boundaries of the frame are not filtered.

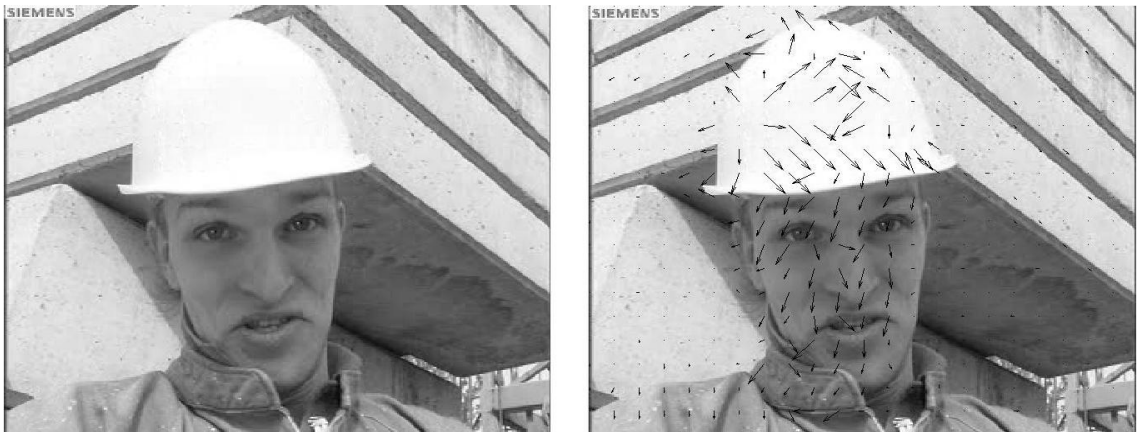


Figure 5.2 Current frame and its MVF

(a) (b)

Figure 5.3 MVF (a) and smoothed MVF (b)

The median of a given set of scalar values is found by numerically sorting these scalar values and selecting the one in the middle. VMFs require ordering multi dimensional data. Several ordering methods such as Aggregate Ordering (A-Ordering), Reduced Ordering (R-Ordering), and Marginal Ordering (M-Ordering) are used for VMF [53]. For a given set of input vectors, A-Ordering based VMFs calculate the sum of distances of each vector to the other input vectors and select the vector with the minimum distance as the output. R-Ordering based VMFs calculate the distance of each input vector to a predefined reference, which may be the origin or the arithmetic mean. In R-Ordering based VMFs, selection of the reference point significantly affects the performance.

M-Ordering based VMFs use scalar median operation for finding the medians of each vector dimension separately. They order the input vectors along each dimension, find the medians of each dimension separately and generate the output vector using these medians. M-Ordering based VMFs are not suitable for FRC, because they usually output a new vector that does not exist in the input vector set. An example showing the disadvantage of M-Ordering based VMFs is shown in Figure 5.4. In this figure, a transition from black to white in RGB color domain at the time instance "t_n" is shown. At the time instance "t_{n-2}" an impulsive noise occurs in the red dimension, which is suppressed by the median filter. However, the median filter also changes this signal at "t_{n-1}" from low to high. This means during the transition from black to white a red output appears incorrectly.

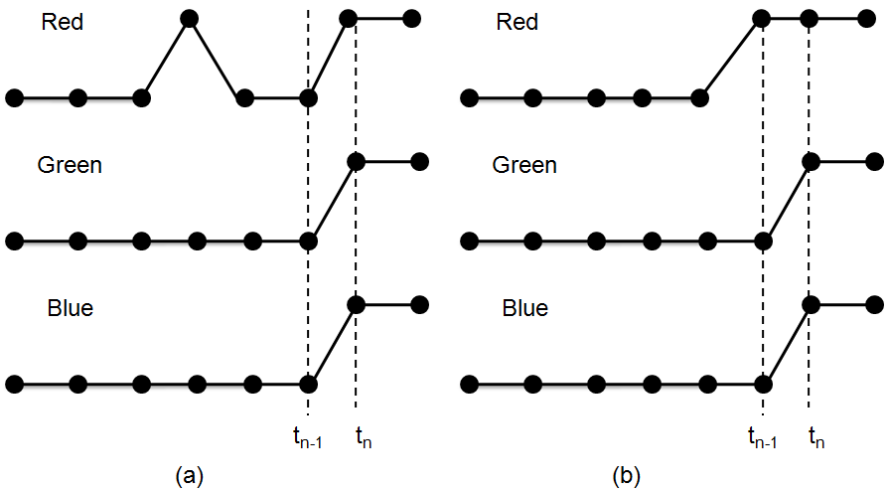


Figure 5.4 M-Ordering based VMF (a) input, (b) output

In this thesis, we used A-ordering method, since it is more suitable for FRC. The computational complexity of A-Ordering based VMF depends on the metric (norm) used to calculate the distance between two vectors [54]. The absolute norm (1-norm), the Euclidean norm (2-norm), or the squared Euclidean norm (squared 2-norm) can be used for A-Ordering. The computational complexities of distance metrics for calculating the sum of distances of a vector to the other vectors in an NxN filtering window are shown in Table 5.1.

Table 5.1 Comparison of distance metrics

Arithmetic Operation	1-norm	2-norm	Squared 2-norm
Add / Sub	$4N^2$-5	$4N^2$-5	$4N^2$-5
Absolute	$2N^2$-2	-	-
Multiplication	-	$2N^2$-2	$2N^2$-2
Square Root	-	N^2-1	-

2-norm has the highest computational complexity since it uses a square root operation for calculating a distance. Squared 2-norm has lower computational complexity since it does not use square root operations. 1-norm has the lowest computational complexity since it does not use square and square root operations. The output of 1-norm VMF for N^2 input vectors is given in (5.1), where N^2 is the number of vectors in the filtering window, j denotes a vector in the window, and i denotes the other vectors in the window. 1-norm distance between two vectors is calculated as shown in (5.2) [49, 50].

$$\vec{v}_m = \arg\min_j \sum_{i=1, i\neq j}^{N^2} \left\| \vec{v}_j - \vec{v}_i \right\|_1 \tag{5.1}$$

$$\left\| \vec{v}_j - \vec{v}_i \right\|_1 = \left| v_{jx} - v_{ix} \right| + \left| v_{jy} - v_{iy} \right|, \text{ where } \vec{v}_j = \left(v_{jx}, v_{jy} \right) \text{ and } \vec{v}_i = \left(v_{ix}, v_{iy} \right) \tag{5.2}$$

A disadvantage of VMFs is the lack of control on the operations of filter. Weighted median filters are proposed in order to overcome this drawback [40, 42, 48, 55, 56]. Weighted VMF operation is shown in equation (5.3), where weights are shown with w_i [55]. In [56], algorithms for fast optimization of weights for the weighted VMF are given.

$$\vec{v}_m = \arg\min_j \sum_{i=1, i\neq j}^{N^2} w_i \left\| \vec{v}_j - \vec{v}_i \right\|_1 \quad (5.3)$$

VMFs are difficult to implement in real-time because of their high computational complexity [6, 54]. Several techniques to reduce the computational complexity of VMFs are developed. In [57], an approximation to the Euclidean norm for VMF is proposed. The square root operation of the Euclidean norm is avoided by a linear approximation. However, this technique requires sorting the vector dimensions according to their absolute values and then weighting the greater dimensions more heavily. In [58], an iterative technique for VMF is proposed. This technique requires less than five iterations for a window size of 3x3, on the average. The authors indicate this as an advantage over the existing techniques, which require nine passes in order to calculate the distance of each vector to the remaining vectors. Because of the sequential nature of this technique, it is not very suitable for hardware implementation.

In [54], an algorithm to reduce the computational complexity of squared 2-norm VMF is presented. The input that minimizes the sum of the squared Euclidean distances to other inputs will be the mean vector of the input set. Therefore, rather than computing the difference of each vector to the remaining vectors, it will be enough to compute the difference of each vector to the mean vector of the input set. This technique reduces the order of computation from N^4 to N^2. However, a mean operation is required by this technique and the mean operation requires a division.

In [54], a technique to reduce the computational complexity of 1-norm VMF is presented as well. To compute the 1-norm median value, the proposed fast technique first applies the scalar median for each dimension. This technique reduces the computational complexity to N^2, but applying the scalar median for each dimension is identical to marginal ordering and this initial step of the proposed technique has a high computational complexity. In addition, the complexity reduction proposed in this paper depends on the variance of the input set and the size of the window. The proposed technique is more effective for an input set having a lower variance and for windows larger than 5x5.

In [54], a pre-computation technique, which we used in this thesis, is mentioned as well. It is indicated that, without using the pre-computed values, for an NxN window "$N^2(N^2-1)$" distances must be calculated. By storing the distances that have already been calculated, only "$N(N^2-1) + N(N-1) / 2$" distances must be calculated. In this way, the computational complexity of the 1-norm VMF is reduced to N^3.

In [59], a performance improvement technique utilizing the redundancy in images is presented. This technique is based on window memoization. In order to reduce the amount of memory, only the two most significant bits of pixels are used for memoization. Since MVs have two dimensions, this technique requires a large area for FRC applications. In addition, using only the two most significant bits of vectors will decrease the visual performance for FRC applications.

There are several papers in the literature presenting hardware implementations of scalar median filters. In [60], 1D median filtering is implemented using a cumulative histogram. The design is scalable for any window length. For 8 bit input samples a histogram with 256 bins is used to find the median value. Proposed architecture is synthesized to Xilinx XC2V6000 FPGA. In [61], median filtering is implemented with a ranking method. Proposed architecture is implemented on a Xilinx XC4013XL-1 FPGA. This architecture consumes large area, because of the large number of required comparators. In [62], an area efficient median based genetic algorithm is developed. Rather than using larger window size, the authors developed a filter bank consisting of 3x3 filters. After training the algorithm on a test image, the resulting filter bank is implemented on a Xilinx Virtex II Pro XC2VP50-7 FPGA. The authors claim that the filter bank technique requires less hardware resources.

There are few papers presenting hardware implementations of VMFs [63]. In [63], VMF is adaptively applied on the MVF. First, a mean vector is calculated for each window position. Then, the mean of the distances between all the vectors in the window to the mean vector is calculated. The proposed hardware implementation consumes 927 slices and works at 117.63 MHz on a Xilinx XC4VLX60 FPGA.

5.1 Computation Reductions for Vector Median Filtering

We propose several techniques to reduce the computational complexity of 1-norm VMF for FRC by using data reuse methodology and by exploiting spatial correlations in the MVF [11]. To the best of our knowledge, there is no paper in the literature which reduces the amount of computations performed by VMFs by analyzing the spatial correlations between neighboring MVs. Since 3x3 window size is used in FRC papers in the literature, we also used this window size. However, the proposed techniques are scalable to any window size.

5.1.1 Data-Reuse Technique

Three consecutive 3x3 filtering windows are shown in Figure 5.5. The numbers in this figure show the vectors in the filtering windows. Since the filtering window slides from left to right over the MVF, vectors 1, 4, and 7 that are in the first filtering window are not in the next filtering window. Therefore, data reuse technique is applicable to 6 out of 9 vectors in the current filtering window, and 5 out of 8 distances for each vector can be stored and reused for the next filtering window.

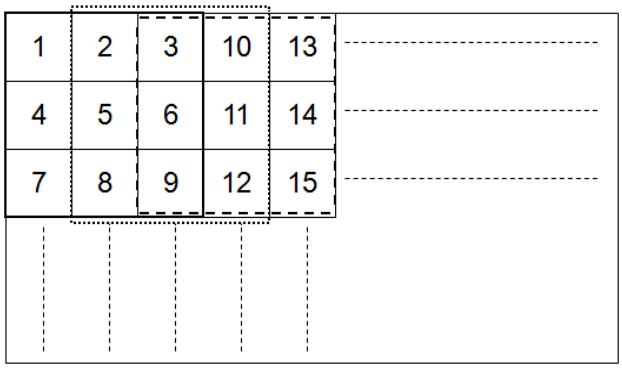


Figure 5.5 3x3 Filtering windows

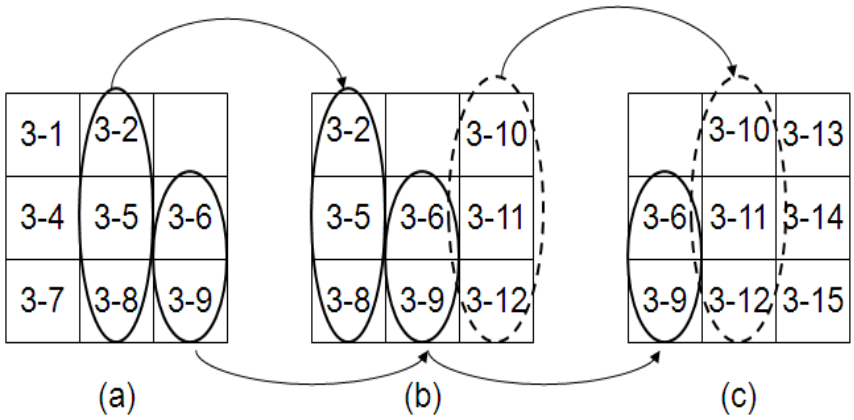


Figure 5.6 The distances between vector 3 and other vectors in three consecutive filtering windows

The 1-norm distances between vector 3 and other vectors in three consecutive filtering windows are shown in Figure 5.6. The 1-norm distances of vector 3 to the other vectors in the current filtering window are shown in Figure 5.6(a). For example, 3-5 denotes the 1-norm distance between vectors 3 and 5 in the current filtering window. As shown in Figures 5.6(b) and 5.6(c), some of these 1-norm distances are also used to compute the VMF for the next filtering windows.

Calculating the sum of 1-norm distances of a vector to the remaining vectors in a 3x3 filtering window requires 16 subtraction, 16 absolute value and 15 addition operations. Therefore, calculating the sum of 1-norm distances of each vector to the remaining vectors in a 3x3 filtering window without data reuse technique requires 16*9=144 subtraction, 16*9=144 absolute value and 15*9=135 addition operations. The number of arithmetic operations required for any filtering window size can be calculated as follows. In an NxN filtering window, there are N^2 vectors. Calculating the sum of 1-norm distances of a vector to the remaining vectors in an NxN filtering window requires $2(N^2-1)$ subtraction, $2(N^2-1)$ absolute value and $2(N^2-2)+1$ addition operations. Therefore, for N^2 vectors, $2N^2(N^2-1)$ subtraction, $2N^2(N^2-1)$ absolute value and $2N^2(N^2-2)+N^2$ addition operations are required. The numbers of arithmetic operations required for various filtering window sizes without proposed data reuse technique are shown in Table 5.2. In this table, required arithmetic operations are given per filtering operation and per HD frame. A 1920x1080 HD frame consists of 8100 16x16 MBs. Therefore, there are 7730 filtering windows for 3x3 VMF. For 5x5 VMF and 7x7 VMF, there are 7368 and 7014 filtering windows, respectively.

Table 5.2 Required arithmetic operations without proposed technique

Arithmetic Operation	Per Filtering Operation			Per HD Frame		
	3x3 VMF	5x5 VMF	7x7 VMF	3x3 VMF	5x5 VMF	7x7 VMF
Absolute value	144	1200	4704	$1.113x10^6$	$8.841x10^6$	$32.993x10^6$
Subtraction	144	1200	4704	$1.113x10^6$	$8.841x10^6$	$32.993x10^6$
Addition	135	1175	4655	$1.043x10^6$	$8.657x10^6$	$32.650x10^6$

The number of these arithmetic operations can be significantly reduced by data reuse technique. Data reuse technique is applicable to 6 vectors out of 9 vectors in a 3x3 filtering window. When the filtering window slides to right over the MVF, the current filtering window has 3 new vectors that are not in the previous filtering window. Data reuse technique stores the sum of 1-norm distances between the other 6 vectors that are in the previous filtering window in 6*2=12 registers. For example, as shown in Figure 5.6, for vector 3, the sum of distances 3-2, 3-5, 3-8 are stored in a register, and the sum of distances 3-6, 3-9 are stored in a register.

The sum of 1-norm distances of these 3 new vectors to the remaining vectors in the filtering window should be calculated, and this requires 16*3=48 subtraction, 16*3=48 absolute value and 15*3=45 addition operations. In an NxN filtering window, $2N(N^2-1)$ subtraction, $2N(N^2-1)$ absolute value, and $2N(N^2-2)+N$ addition operations are required.

The 1-norm distances of the remaining 6 vectors in the filtering window to these new 3 vectors should be calculated, and this requires 6*6=36 subtraction, 6*6=36 absolute value and 6*5=30 addition operations. In order to find the sum of 1-norm distances of the remaining 6 vectors to all the other vectors in the filtering window, these 1-norm distances should be added to the previously calculated and stored sum of 1-norm distances between these 6 vectors, and this requires 6*2=12 addition operations. In an NxN filtering window, $2N(N^2-N)$ subtraction, $2N(N^2-N)$ absolute value, and $(3N-2)(N^2-N)$ addition operations are required.

Therefore, calculating the sum of 1-norm distances of each vector to the remaining vectors in a 3x3 filtering window with data reuse technique requires 48+36=84 subtraction, 48+36=84 absolute value and 45+30+12=87 addition operations. Calculating the sum of 1-norm distances of each vector to the remaining vectors in an NxN filtering window with data reuse technique requires $2N(2N^2-N-1)$ subtraction, $2N(2N^2-N-1)$

absolute value and $5N^3$-$5N^2$-N addition operations. The numbers of arithmetic operations required for various filtering window sizes with proposed data reuse technique are shown in Table 5.3. For 3x3 filtering window size, the proposed data reuse technique avoids 60 subtraction, 60 absolute value and 48 addition operations, and it only requires 12 store operations. The number of store operations required for NxN filtering window size is $2(N^2-N)$.

Table 5.3 Required arithmetic operations with proposed technique

Arithmetic Operation	Per Filtering Operation			Per HD Frame		
	3x3 VMF	5x5 VMF	7x7 VMF	3x3 VMF	5x5 VMF	7x7 VMF
Absolute value	84	440	1260	$0.649x10^6$	$3.241x10^6$	$8.837x10^6$
Subtraction	84	440	1260	$0.649x10^6$	$3.241x10^6$	$8.837x10^6$
Addition	87	495	1463	$0.672x10^6$	$3.647x10^6$	$10.261x10^6$

5.1.2 Spatial Correlations Technique

The proposed spatial correlations based techniques try to avoid redundant computations for calculating sum of 1-norm distances between the vectors in the current filtering window based on the spatial correlations between the neighboring MVs [11].

1-norm VMF calculates the sum of 1-norm distances of each vector to the other vectors in the current filtering window and selects the vector with the minimum distance as the output. The sum of 1-norm distances of a vector to the other vectors can be calculated by finding the sum of absolute differences between the x dimension of this vector and the x dimensions of the other vectors, and the sum of absolute differences between the y dimension of this vector and the y dimensions of the other vectors, and adding them.

When the filtering window slides to right in the MVF, Correlation 1 technique compares the x dimensions and y dimensions of 3 new vectors in the current filtering window. If the x dimensions of these 3 vectors are equal, it calculates the sum of absolute differences between this x dimension and the x dimensions of the other vectors

in the filtering window, and uses the same result for all 3 vectors. The same is done for the y dimension. For example, since the y dimensions of vectors (4,9), (6,9), and (7,9) are equal, the sum of absolute differences between 9 and the y dimensions of the other vectors in the filtering window is calculated once and the same result is used for all 3 vectors.

When the filtering window slides to right in the MVF, Correlation 2 technique compares the x dimension and y dimension of each new vector with the x dimension and y dimension of the vector in the middle of the current filtering window. For example, it compares the new vectors 10, 11, 12 with vector 6 in the second 3x3 filtering window in Figure 5.5. If the x dimension of a new vector is equal to the x dimension of the vector in the middle, it uses the previously calculated and stored sum of absolute differences between x dimension of the vector in the middle (vector 6 in the second 3x3 filtering window in Figure 5.5) and the x dimensions of the 5 old vectors in the filtering window (vectors 2, 3, 5, 8, 9 in the second 3x3 filtering window in Figure 5.5) for this new vector. The same is done for the y dimension.

When the filtering window slides to right in the MVF, Correlation 3 technique compares the x dimension and y dimension of each new vector with the x dimension and y dimension of the old vectors in the current filtering window. For example, it compares the new vectors 10, 11, 12 with vectors 2, 3, 5, 6, 8, 9 in the second 3x3 filtering window in Figure 5.5. If the x dimension of a new vector is equal to the x dimension of any compared vector, it uses the previously calculated and stored sum of absolute differences between x dimension of this old vector and the x dimensions of the remaining 5 old vectors in the filtering window for this new vector. The same is done for the y dimension.

The overhead of proposed techniques for various filtering window sizes are given in Table 5.4 and Table 5.5. For an NxN filtering window, Correlation 1 requires (N^2-N) comparison operations, whereas Correlation 2 requires 2N comparison and 2 store operations and Correlation 3 requires $2(N^3- N^2)$ comparison and $2(N^2-N)$ store operations.

Table 5.4 Comparison overhead of spatial correlation techniques

Proposed Technique	Per Filtering Operation			Per HD Frame		
	3x3 VMF	5x5 VMF	7x7 VMF	3x3 VMF	5x5 VMF	7x7 VMF
Correlation 1	6	20	42	4.638×10^4	1.4736×10^5	2.94588×10^5
Correlation 2	6	10	14	4.638×10^4	7.368×10^4	9.8196×10^4
Correlation 3	36	200	588	2.7828×10^5	1.4736×10^6	3.913812×10^6

Table 5.5 Store overhead of spatial correlation techniques

Proposed Technique	Per Filtering Operation			Per HD Frame		
	3x3 VMF	5x5 VMF	7x7 VMF	3x3 VMF	5x5 VMF	7x7 VMF
Correlation 1	0	0	0	0	0	0
Correlation 2	2	2	2	1.546×10^4	1.4736×10^4	1.4028×10^4
Correlation 3	12	40	84	9.276×10^4	2.9472×10^5	5.89176×10^5

The computation reductions achieved by spatial correlation techniques for a 3x3 filtering window are shown in Table 5.6 and Table 5.7. The simulations are done for the first 50 frames of the "Ducks" and "SthlmPan" video sequences and for the first 100 frames of the other video sequences. The resolutions and frame rates of these video sequences are given in Table 5.6. The MVFs are obtained by FS algorithm with 16x16 MB size on a search range of (±8,±8) pixels for CIF sized videos and on a search range of (±16,±16) pixels for remaining videos. The simulation results in Table 5.6 show the percentages of x dimensions and y dimensions of the 3 new vectors for all 3x3 filtering windows in these video frames for which the sum of absolute differences computations are avoided.

The proposed spatial correlation techniques do not require the x dimension and y dimension of a new vector to be equal. They can avoid the sum of absolute differences computations for only x dimension or y dimension of a new vector. In order to quantify the impact of this, we modified the Correlation 1 and Correlation 2 techniques so that they require the equality of x dimension and y dimension of a new vector in order to avoid the sum of absolute differences computations for this vector. The simulation results in Table 5.7 show the percentages of x dimensions and y dimensions of the 3 new vectors for all 3x3 filtering windows in these video frames for which the sum of absolute differences computations are avoided by these modified correlation techniques.

Table 5.6 Computation reductions for 3x3 VMF

Video Sequence	Resolution	fps	Correlation 1	Correlation 2	Correlation 3
CoastGuard	**(352x240)**	**30**	44.980 %	64.366 %	78.028 %
Flowers H	**(352x240)**	**29**	41.027 %	55.163 %	74.050 %
Foreman	**(352x288)**	**30**	38.072 %	48.282 %	73.524 %
M. Calendar L	**(352x240)**	**29**	49.088 %	71.061 %	79.829 %
Susie L	**(352x240)**	**29**	37.524 %	48.398 %	72.016 %
Table Tennis L	**(352x240)**	**29**	50.912 %	73.085 %	79.082 %
M. Calendar H	**(704x480)**	**29**	48.249 %	67.501 %	84.591 %
Susie H	**(704x480)**	**29**	30.476 %	38.539 %	67.298 %
Table Tennis H	**(704x480)**	**29**	54.212 %	78.535 %	86.289 %
Flowers H	**(704x480)**	**29**	40.527 %	56.890 %	75.077 %
Gladiator	**(720x576)**	**25**	22.267 %	26.405 %	54.535 %
Spiderman	**(720x576)**	**25**	15.148 %	15.858 %	43.050 %
Irobot	**(720x576)**	**25**	25.357 %	33.081 %	61.834 %
Spider3	**(1280x528)**	**23**	37.845 %	52.946 %	70.189 %
Ducks	**(1280x720)**	**50**	44.611 %	62.986 %	85.788 %
SthlmPan	**(1280x720)**	**50**	47.674 %	68.687 %	81.234 %

Table 5.7 Computation reductions by modified correlation techniques for 3x3 VMF

Video Sequence	Correlation 1	Correlation 2
CoastGuard	34.745 %	46.695 %
Flowers L	31.204 %	39.329 %
Foreman	29.134 %	35.188 %
M. Calendar L	42.924 %	60.590 %
Susie L	27.552 %	33.921 %
Table Tennis L	48.136 %	68.252 %
M. Calendar H	39.610 %	53.291 %
Susie H	20.139 %	24.874 %
Table Tennis H	50.599 %	72.935 %
Flowers H	32.132 %	45.279 %
Gladiator	9.212 %	11.945 %
Spiderman	4.376 %	4.131 %
Irobot	15.073 %	20.026 %
Spider3	30.279 %	43.129 %
Ducks	32.050 %	42.146 %
SthlmPan	41.989 %	59.919 %

We propose using a threshold, called “dif”, for increasing the computation reductions achieved by the proposed spatial correlation techniques. The proposed techniques require the dimensions of the compared vectors to be equal in order to avoid computations. The proposed techniques using “dif” avoid the computations for similar vectors as well by allowing a maximum difference of “dif” between the dimensions of the compared vectors. For example, when “dif” is set to 2, a reduction in computations will be achieved when the absolute value of the difference in any dimensions of the compared vectors is less than or equal to 2 pixels. The computation reductions achieved by the proposed spatial correlation techniques using “dif” for a 3x3 filtering window are

shown in Table 5.8. The computation reductions achieved by the proposed modified spatial correlations techniques using "dif" for a 3x3 filtering window are shown in Table 5.9. When "dif" is set to 2 for modified spatial correlations techniques, a reduction in computations will be achieved when the absolute value of the difference in both dimensions of the compared vectors is less than or equal to 2 pixels. The modified spatial correlations techniques achieve less computation reduction than the original spatial correlations techniques. The difference between the computation reductions achieved by the modified and the original spatial correlations techniques for various "dif" values are shown in Table 5.10.

Table 5.8 Computation reductions for 3x3 VMF using "dif"

Video Sequence	dif = 2		dif = 4	
	Correlation 1	Correlation 2	Correlation 1	Correlation 2
CoastGuard	54.790 %	82.129 %	55.817 %	83.639 %
Flowers L	53.465 %	79.293 %	54.604 %	81.254 %
Foreman	54.115 %	79.396 %	56.117 %	83.326 %
M. Calendar L	55.704 %	83.379 %	56.157 %	84.160 %
Susie L	53.982 %	79.630 %	55.301 %	82.237 %
Table Tennis L	55.096 %	81.731 %	55.855 %	83.448 %
M. Calendar H	58.895 %	87.492 %	59.902 %	89.141 %
Susie H	49.734 %	71.195 %	54.156 %	78.508 %
Table Tennis H	59.530 %	88.712 %	60.415 %	89.936 %
Flowers H	52.989 %	77.214 %	55.727 %	81.881 %
Gladiator	35.067 %	45.234 %	40.075 %	52.704 %
Spiderman	26.456 %	30.027 %	32.910 %	38.770 %
Irobot	39.693 %	55.391 %	44.493 %	63.011 %
Spider3	46.487 %	66.401 %	49.734 %	71.198 %
Ducks	61.554 %	92.060 %	63.088 %	94.534 %
SthlmPan	56.742 %	83.411 %	58.796 %	86.812 %

Table 5.9 Computation reductions by modified correlation techniques for 3x3 VMF using "dif"

Video Sequence	dif = 2		dif = 4	
	Correlation 1	Correlation 2	Correlation 1	Correlation 2
CoastGuard	53.324 %	79.913 %	55.328 %	82.833 %
Flowers L	51.377 %	75.644 %	53.306 %	78.936 %
Foreman	52.131 %	76.031 %	54.861 %	81.255 %
M. Calendar L	55.046 %	82.254 %	55.929 %	83.753 %
Susie L	52.471 %	76.969 %	54.513 %	80.725 %
Table Tennis L	54.224 %	79.808 %	55.489 %	82.664 %
M. Calendar H	56.664 %	83.727 %	58.327 %	86.413 %
Susie H	44.456 %	63.608 %	50.353 %	72.688 %
Table Tennis H	58.392 %	86.057 %	59.596 %	88.410 %
Flowers H	48.564 %	70.437 %	51.851 %	75.692 %
Gladiator	22.171 %	30.312 %	27.285 %	36.664 %
Spiderman	12.280 %	13.563 %	17.749 %	19.703 %
Irobot	29.919 %	42.004 %	35.101 %	49.797 %
Spider3	39.252 %	56.976 %	42.689 %	61.785 %
Ducks	59.560 %	88.808 %	62.622 %	93.747 %
SthlmPan	51.879 %	75.446 %	54.459 %	79.192 %

Table 5.10 Difference between the computation reductions achieved by the modified and the original spatial correlations techniques

Video Sequence	dif = 0		dif = 2		dif = 4	
	Corr. 1	Corr. 2	Corr. 1	Corr. 2	Corr. 1	Corr. 2
CoastGuard	-22.754%	-27.453%	-2.675%	-2.698%	-0.876%	-0.963%
Flowers L	-23.942%	-28.704%	-3.905%	-4.601%	-2.377%	-2.852%
Foreman	-23.476%	-27.119%	-3.666%	-4.238%	-2.238%	-2.485%
M. Calendar L	-12.557%	-14.735%	-1.181%	-1.349%	-0.406%	-0.483%
Susie L	-26.575%	-29.912%	-2.799%	-3.341%	-1.424%	-1.838%
Table Tennis L	-5.452%	-6.612%	-1.582%	-2.352%	-0.655%	-0.939%
M. Calendar H	-17.905%	-21.051%	-3.788%	-4.303%	-2.629%	-3.060%
Susie H	-33.918%	-35.457%	-10.612%	-10.656%	-7.022%	-7.413%
Table Tennis H	-6.664%	-7.130%	-1.911%	-2.992%	-1.355%	-1.696%
Flowers H	-20.714%	-20.409%	-8.350%	-8.776%	-6.955%	-7.558%
Gladiator	-58.629%	-54.762%	-36.775%	-32.988%	-31.915%	-30.434%
Spiderman	-71.111%	-73.950%	-53.583%	-54.830%	-46.068%	-49.179%
Irobot	-40.556%	-39.463%	-24.624%	-24.168%	-21.108%	-20.970%
Spider3	-19.992%	-18.541%	-15.563%	-14.194%	-14.165%	-13.220%
Ducks	-28.156%	-33.086%	-3.239%	-3.532%	-0.738%	-0.832%
SthlmPan	-11.924%	-12.765%	-8.570%	-9.549%	-7.376%	-8.777%

Since the proposed spatial correlations techniques are scalable to larger window sizes, we obtained the performance results for larger filtering window sizes. The simulation results for 5x5 VMF and for 7x7 VMF are given in Table 5.11 and Table 5.12, respectively. The simulation results for various "dif" values and filtering window sizes are given in Table 5.13. As "dif" value increases, computation reductions increase, especially for videos having large motions.

Based on these results, Correlation 2 technique performs slightly better than Correlation 1 technique, especially for 3x3 filtering window. This is an expected result, because for stationary frames and for frames having a global motion Correlation 2 should perform better. For these types of frames, MVs entering the filtering window will be equal, and therefore Correlation 2 will avoid the computations for all these MVs, whereas Correlation 1 will perform one computation. Because, Correlation 1 technique performs at least one computation independent of the new vectors entering the filtering window. Correlation 1 can avoid at most 2/3, 4/5, and 6/7 of the computations for 3x3, 5x5, and 7x7 filtering windows, respectively. The performance difference between Correlation 1 and Correlation 2 techniques decreases for larger filtering windows. One reason for this is that in Correlation 2 for larger filtering windows, the difference between the physical locations of new vectors entering the filtering window and the old

vector compared with them, which is the vector in the middle of the filtering window, is large. Therefore, the compared vectors are less correlated.

As the filtering window size gets larger, the performances of the proposed techniques decrease. Because for larger filtering windows, objects become smaller than the filtering window, and the correlation between compared vectors decrease. Because of this reason, FRC algorithms reported in the literature use a 3x3 filtering window.

Table 5.11 Computation reductions for 5x5 VMF

Video Sequence	dif = 0		dif = 2		dif = 4	
	Corr. 1	Corr. 2	Corr. 1	Corr. 2	Corr. 1	Corr. 2
CoastGuard	42.750 %	47.704 %	50.128 %	61.702 %	50.626 %	63.078 %
Flowers L	37.810 %	36.924 %	48.039 %	58.003 %	49.089 %	59.940 %
Foreman	40.184 %	36.491 %	53.417 %	62.924 %	55.310 %	67.005 %
M. Calendar L	46.340 %	53.430 %	50.282 %	62.424 %	50.645 %	63.115 %
Susie L	37.540 %	36.628 %	49.737 %	60.397 %	50.413 %	62.154 %
Table Tennis L	45.101 %	50.522 %	49.771 %	60.486 %	50.475 %	62.468 %
M. Calendar H	56.564 %	62.068 %	65.172 %	79.570 %	65.912 %	81.050 %
Susie H	38.910 %	35.554 %	58.864 %	66.725 %	62.401 %	73.170 %
Table Tennis H	60.476 %	50.522 %	65.472 %	79.565 %	66.336 %	81.524 %
Flowers H	45.780 %	47.686 %	59.189 %	68.679 %	62.030 %	73.470 %
Gladiator	27.430 %	19.664 %	43.647 %	38.273 %	49.840 %	45.511 %
Spiderman	20.456 %	13.530 %	35.440 %	26.366 %	43.525 %	34.379 %
Irobot	31.699 %	26.417 %	48.686 %	48.222 %	54.094 %	55.842 %
Spider3	44.059 %	47.526 %	54.089 %	59.734 %	57.761 %	64.250 %
Ducks	57.085 %	61.989 %	71.159 %	87.315 %	71.972 %	89.541 %
SthlmPan	57.945 %	66.797 %	66.409 %	79.672 %	68.268 %	82.595 %

Table 5.12 Computation reductions for 7x7 VMF

Video Sequence	dif = 0		dif = 2		dif = 4	
	Corr. 1	Corr. 2	Corr. 1	Corr. 2	Corr. 1	Corr. 2
CoastGuard	25.041 %	25.571 %	28.344 %	32.690 %	28.447 %	33.128 %
Flowers L	21.381 %	16.631 %	26.833 %	28.842 %	27.496 %	30.320 %
Foreman	30.823 %	24.184 %	39.590 %	42.231 %	41.182 %	45.877 %
M. Calendar L	26.143 %	27.002 %	28.217 %	32.909 %	28.416 %	33.144 %
Susie L	22.101 %	19.247 %	28.049 %	31.846 %	28.361 %	32.739 %
Table Tennis L	25.753 %	27.128 %	27.999 %	31.939 %	28.364 %	32.847 %
M. Calendar H	55.894 %	55.755 %	62.243 %	70.959 %	62.717 %	71.926 %
Susie H	48.854 %	31.888 %	57.946 %	60.760 %	60.600 %	66.247 %
Table Tennis H	57.334 %	60.399 %	62.129 %	69.333 %	63.002 %	71.596 %
Flowers H	44.143 %	38.834 %	56.124 %	58.545 %	58.866 %	63.359 %
Gladiator	28.616 %	15.913 %	45.245 %	32.942 %	51.534 %	39.865 %
Spiderman	22.651 %	11.656 %	38.627 %	23.103 %	46.783 %	30.440 %
Irobot	33.189 %	21.120 %	49.750 %	41.159 %	55.017 %	48.515 %
Spider3	44.071 %	42.133 %	53.805 %	52.892 %	57.360 %	56.995 %
Ducks	60.188 %	58.451 %	71.861 %	82.111 %	72.385 %	84.076 %
SthlmPan	60.476 %	64.228 %	67.664 %	75.335 %	69.333 %	77.858 %

Table 5.13 Average computation reductions

Filter Size	dif = 0		dif = 2		dif = 4	
	Corr 1	Corr 2	Corr 1	Corr 2	Corr 1	Corr 2
3x3	39.2%	53.8%	50.8%	73.9%	53.3%	77.7%
5x5	43.1%	43.3%	54.3%	62.5%	56.7%	66.1%
7x7	37.9%	33.7%	46.5%	47.9%	48.7%	51.1 %

MVFs with higher spatial consistency increase the quality of the frames interpolated by FRC. Therefore, we used the Sum of Absolute Minimum Neighboring Difference (SAMND) metric [64] in order to determine the impact of VMF on the spatial consistency of MVFs. SAMND metric determines the correlation between the motions of the neighboring MBs by calculating the difference between their MVs as shown in (5.4). Since this is an off-line operation, 2-norm is used to find the distances between the MVs. In (5.4), $\vec{x}_c$ denotes the vector in the middle of the filtering window, $\vec{x}_i$ denotes the remaining vectors in the filtering window, and N denotes the total number of MBs in a frame.

Table 5.14 shows the SAMND results for 3x3 filtering window. FS algorithm is used to obtain MVFs. The results given in this table are average SAMND values per MB, for which VMF is applicable. In these simulations, smoothing is applied recursively which means that the VMF uses the existing smoothed MVs in the current filtering window. Since real-time video processing hardware work MB by MB rather than working frame by frame, this is suitable for hardware implementation. As it can be seen from Table 5.14, smoothing the MVF increases the spatial consistency between neighboring MVs. SAMND performance decreases for larger "dif" values. Because increasing "dif" increases the possibility of selecting the MV in the middle of the filtering window as the median MV, which is equal to not doing any smoothing operation.

$$SAMND = \sum_{MB=1}^{N} \min \left\| \vec{x}_c - \vec{x}_i \right\|_2 \text{, where i} \neq \text{c} \qquad (5.4)$$

Since SAMND metric is based on the minimum difference between the current MV and its neighboring MVs, it may give incorrect results for exceptional cases, e.g. when two similar outlier MVs are in the same filtering window. Therefore, we

Table 5.14 SAMND results

Video Sequence	Without smoothing	With smoothing	dif = 2		dif = 4	
			Corr 1	Corr 2	Corr 1	Corr 2
CoastGuard	0.1262	0.0075	0.0346	0.0439	0.0467	0.0654
Flowers	0.2712	0.0155	0.0264	0.0609	0.0343	0.0890
Foreman	0.3233	0.0313	0.0629	0.0876	0.0827	0.1303
M.Calendar	0.0803	0.0015	0.0305	0.0370	0.0414	0.0508
Susie	0.3232	0.0386	0.0771	0.1074	0.0954	0.1468
TableTennis	0.1007	0.0061	0.0282	0.0302	0.0486	0.0507
M.Calendar	0.3480	0.0107	0.0393	0.0611	0.0521	0.0861
Susie	1.4038	0.1180	0.1258	0.1400	0.1680	0.2069
TableTennis	0.2777	0.0129	0.0531	0.0567	0.0716	0.0762
Flowers	0.8106	0.0610	0.0615	0.0755	0.0820	0.1084
Gladiator	2.9077	0.2577	0.3294	0.3325	0.3621	0.3718
Spiderman	4.4531	0.4605	0.5343	0.5418	0.5691	0.5852
Irobot	1.9819	0.1506	0.2007	0.2068	0.2336	0.2426
Spider3	1.6618	0.1282	0.2035	0.2073	0.2286	0.2369
Ducks	0.1554	0.0109	0.0536	0.0721	0.0867	0.1178
SthlmPan	0.6359	0.0580	0.0612	0.0706	0.0770	0.0956

developed the Sum of Neighboring Differences (SND) metric which takes the difference of the current MV with all its neighboring MVs into account. The SND metric is calculated as shown in (5.5). Table 5.15 shows the SND results for 3x3 filtering window. FS algorithm is used to obtain MVFs. The results given in this table are average SND values per MB, for which VMF is applicable.

$$SND = \sum_{MB=1}^{N} \sum_{i \neq c}^{9} \left\| \vec{x}_c - \vec{x}_i \right\|_2 \tag{5.5}$$

Table 5.15 SND results

Video Sequence	Without smoothing	With smoothing	dif = 2		dif = 4	
			Corr 1	Corr 2	Corr 1	Corr 2
CoastGuard	6.1574	2.5855	4.7004	4.7300	5.2268	5.3790
Flowers	7.7833	2.1327	3.8889	4.2306	4.3277	4.7518
Foreman	10.2969	4.0322	6.4099	6.5707	7.3295	7.5359
M.Calendar	3.5041	1.3238	2.6138	2.7027	3.0164	3.1185
Susie	10.1784	4.9273	7.4894	7.6945	8.2058	8.4920
TableTennis	3.1192	1.0538	1.7603	1.8031	2.2278	2.2640
M.Calendar	10.6241	2.6353	4.4929	4.6107	5.0107	5.2168
Susie	32.5122	10.1467	12.8724	12.9829	14.7590	15.0836
TableTennis	6.9591	1.6999	3.1505	3.1698	3.5601	3.6048
Flowers	20.8387	5.3685	6.5933	6.8530	7.6923	8.1461
Gladiator	90.6198	46.0629	46.1256	46.5114	47.8577	47.8834
Spiderman	122.0818	59.6276	59.7013	60.1205	61.7425	61.9127
Irobot	63.2902	31.0584	31.8333	31.8711	33.5845	33.7574
Spider3	49.7524	23.7022	23.7708	23.8530	25.0597	25.2509
Ducks	7.4080	2.4609	5.7550	5.8512	6.9311	6.9999
SthlmPan	20.1122	5.8201	6.8342	6.8986	7.7794	7.9782

As it can be seen from Table 5.15, smoothing the MVF increases the spatial consistency between neighboring MVs and improves the SND performance. Since increasing “dif” increases the possibility of selecting the MV in the middle of the filtering window as the median MV, which is equal to not doing any smoothing operation, SND performance decreases with larger “dif” values.

5.2 Vector Median Filtering Hardware Architecture

In this thesis, we also propose an efficient VMF hardware implementing the proposed computation reduction techniques exploiting the spatial correlations in the MVF [11]. To the best of our knowledge, a VMF hardware implementing these techniques is not presented in the literature. The proposed architecture is scalable to any window size. But, it is implemented for a 3x3 window size because of the FRC requirements. The top-level block diagram of the proposed hardware is shown in Figure 5.7. The control unit generates the necessary control signals for datapaths and sends the MVs to them. It also controls the weighting and minimum selector module. VMF computations for a filtering window are overlapped with loading the new vectors for the next filtering window.

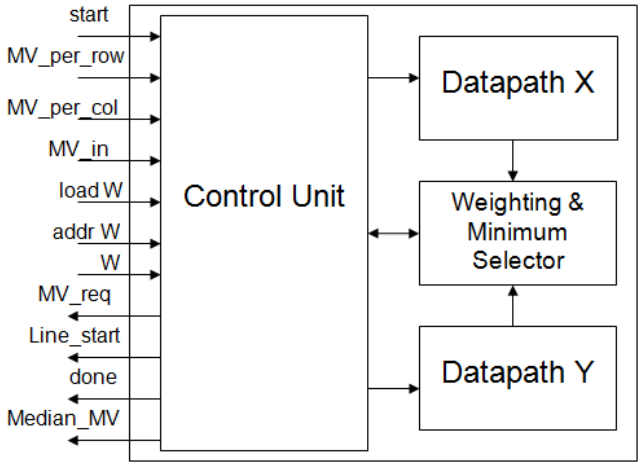


Figure 5.7 Top-level block diagram of the VMF hardware

The proposed hardware has two datapaths working in parallel. The sum of 1-norm distances of a vector to the other vectors in the filtering window is computed by these two datapaths. One datapath computes the sum of absolute differences between the x dimension of this vector and the x dimensions of the other vectors, and the other datapath computes the sum of absolute differences between the y dimension of this vector and the y dimensions of the other vectors.

When the start signal is asserted, the control unit gets the number of MVs per column and MVs per row information from the 9-bit (in order to support 1920x1080 resolution) “MV_per_col” and “MV_per_row” signals. Then, control unit requests the vectors in the current filtering window by asserting “MV_req” signal. If the current filtering window is the first filtering window in a row, control unit asserts the “line_start” signal together with the “MV_req” signal. Because, VMF hardware should get 9 new vectors for the first filtering window in a row, whereas it should get 3 new vectors for the other filtering windows in the row. The VMF hardware receives one new 16-bit vector (8-bit x dimension and 8-bit y dimension) in each clock cycle.

The block diagram of a datapath is shown in Figure 5.8. Ping pong registers are used to overlap computing VMF for the current filtering window with receiving the new vectors for the next filtering window. Vectors are loaded to the registers column by column. For a 3x3 filtering window, loading a column of vectors takes 3 clock cycles. After the vectors in one column of the filtering window are loaded, they are shifted to left by one column. In the datapath, the multiplexer with 9 inputs is used to select the vector of which the sum of 1-norm distances with other vectors will be calculated.

The block diagram of the weighting and minimum selector module is shown in Figure 5.9. After the results obtained by the two datapaths are weighted separately, they are added and the result is stored in a register. The results obtained for all 9 vectors in the current filtering window are compared and the vector with the minimum value is selected as the median vector. The weights are stored in a register file and they can be changed adaptively during run time. Therefore, the proposed hardware can implement adaptively weighted VMF.

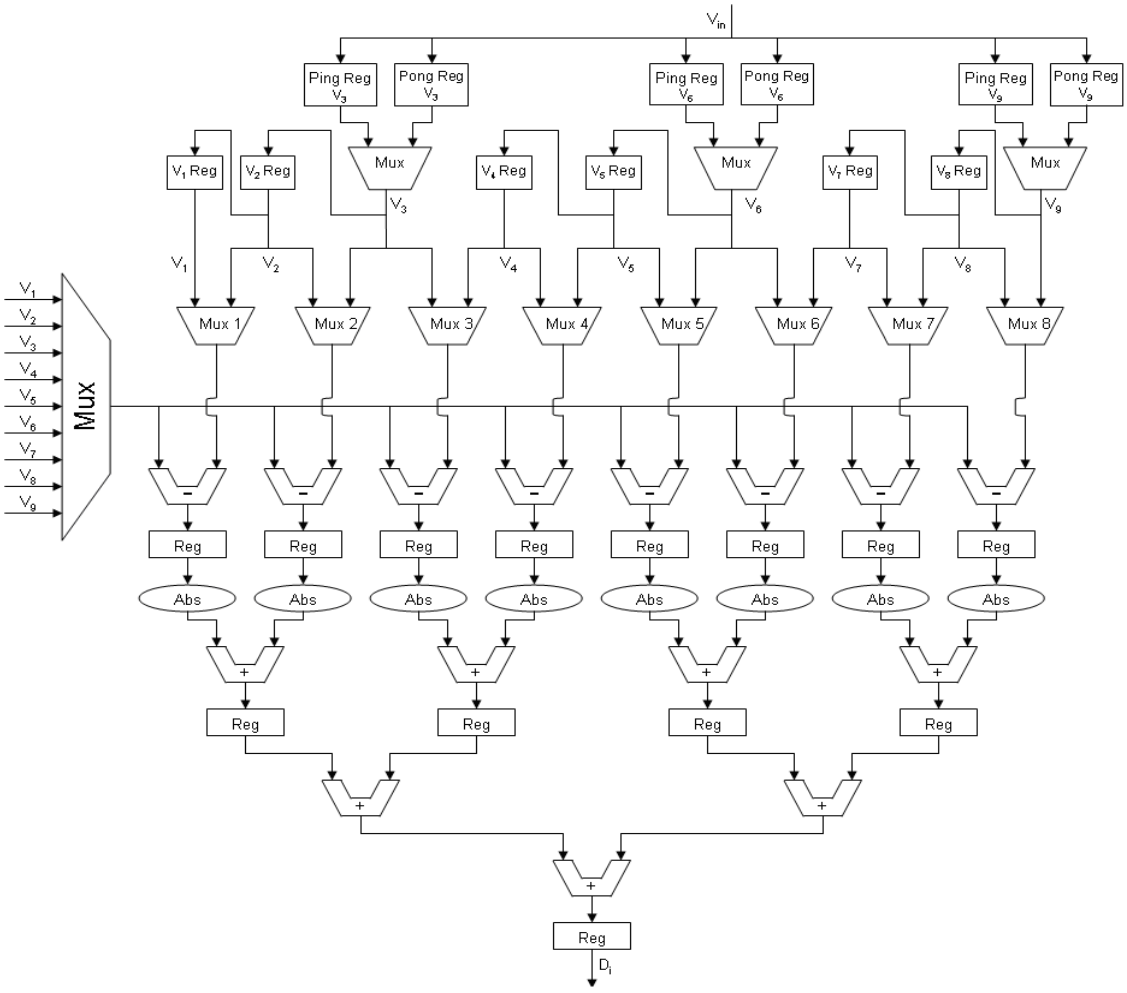


Figure 5.8 Block diagram of the VMF datapath

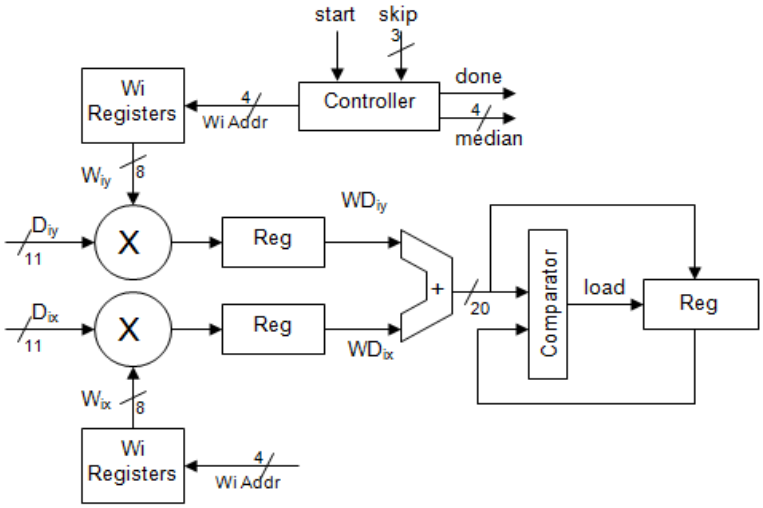


Figure 5.9 Block diagram of the weighting and minimum selector module

The spatial correlation techniques are implemented in the control unit using 6 8-bit comparators. The number of comparators can be reduced by performing the comparisons serially. Since the proposed VMF hardware has two datapaths working in parallel, it requires the equality of x dimension and y dimension of a new vector in order to avoid the sum of absolute differences computations for this vector. Therefore, it can achieve the computation reduction percentages shown in Table 5.7.

For the first filtering window in a row, loading 9 new vectors and computing VMF takes 22 clock cycles. For the other filtering windows in the row, computing VMF takes 12 cycles. Therefore, VMF for a frame without spatial correlation techniques takes ((MV_per_col-2) x 22 + (MV_per_col-2) x (MV_per_row – 3) x 12) clock cycles. Therefore, for 16x16 MB size, VMF for a 1920x1080 HD frame without spatial correlation techniques takes 92690 cycles. For 4x4 MB size, it takes 1539928 cycles.

The proposed VMF hardware architecture is implemented in Verilog HDL, and mapped to a low cost Xilinx XC3S400A-5 FPGA using Xilinx ISE 10.1.03. The implementation is verified with post place and route simulations using Mentor Graphics Modelsim 6.1 PE. The FPGA implementation consumes 1426 slices and it can work at 145 MHz. Since, for 4x4 MB size, VMF for a 1920x1080 HD frame without spatial correlation techniques takes 1539928 clock cycles, VMF for this frame takes 10.62 ms. Therefore, without spatial correlation techniques, the proposed VMF hardware can process 94 HD fps. When the spatial correlation techniques are used, it can process more than 94 HD fps.

CHAPTER 6

FRAME INTERPOLATION HARDWARE

FRC is the conversion of a lower frame rate video signal to a higher frame rate video signal. LCD panels used for HDTV have a frame rate up to 240 Hz, whereas video signals are usually recorded in 24 Hz, 25 Hz, or 30 Hz. Therefore, FRC is required in order to display the HDTV video signals in the LCD panels. FRC can be done by interpolating a new frame between every two consecutive original frames like in 25 Hz to 50 Hz, 30 Hz to 60 Hz, 50 Hz to 100 Hz, 60 Hz to 120 Hz conversions, and it can be done by interpolating three new frames between every two consecutive original frames like in 25 Hz to 100 Hz, 50 Hz to 200 Hz, 30 Hz to 120 Hz, 60 Hz to 240 Hz conversions. In the case of 24 Hz to 60 Hz conversion 3:2 pull-down technique is used [65].

Because of their low computational complexity, simple FRC techniques like frame repetition and Linear Interpolation (LI) are used in some consumer electronics products. But, these simple techniques often produce artifacts to which human eye is very sensitive. Frame repetition results in motion judder and LI causes blurring at object boundaries [66, 67]. To overcome these problems, FRC algorithms using motion information between consecutive frames are developed. For example, Motion Compensated Averaging (MCA) technique performs frame interpolation by using the MVs found by the ME process.

The LI and MCA techniques perform frame interpolation as shown in equations (6.1) and (6.2), respectively. In these equations, "t" is the time instance the frame "F" belongs to, "$\vec{x}$" is the spatial location of the current pixel in the frame and "τ" is the time slot the interpolated frame belongs to. For the conversion ratio 1:2, τ will be 0.5 for both interpolated frames, and for the conversion ratio 1:4, τ will be 0.25, 0.5, and 0.75 for the three interpolated frames.

$$F_{LI}(\vec{x},t-\tau)=(1-\tau)F(\vec{x},t-1)+\tau F(\vec{x},t) \tag{6.1}$$

$$F_{MCA}(\vec{x},t-\tau)=\frac{1}{2}\left[F(\vec{x}+\tau\vec{v},t-1)+F(\vec{x}-(1-\tau)\vec{v},t)\right] \tag{6.2}$$

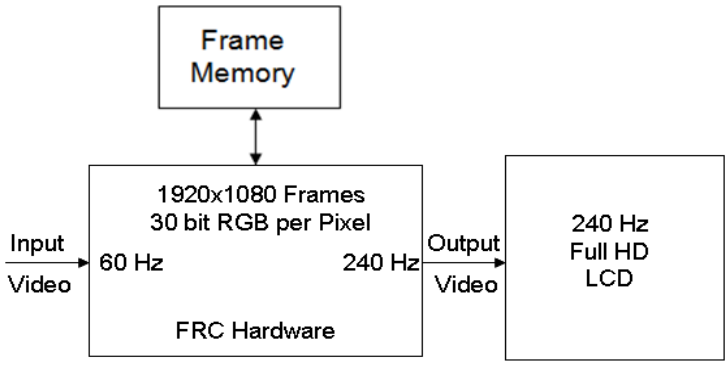


Figure 6.1 An example FRC system

An example FRC system is shown in Figure 6.1. Analyzing the off-chip memory bandwidth requirement of this FRC system clearly shows that FRC systems require significant data transfer from the off-chip frame memory. This FRC system implements a 1:4 conversion ratio. It will interpolate new frames by using one MV per MB and accessing one MB from the current frame and one MB from the reference frame. Since each color channel is 10 bits, the RGB values of a pixel take 30 bits which can be stored in a 32 bit word in memory. A Full HD frame has 1920x1080 (1.98M) pixels which take 7.92MB. Therefore, 15.84MB (2x7.92MB) have to be accessed from the off-chip frame memory in order to interpolate one frame. Since three frames will be interpolated per original frame, 47.52MB have to be accessed from the off-chip frame memory.

The received input frame and the interpolated frames will be stored in the frame memory and they will be sent to the LCD display from the frame memory. Storing interpolated frames in the frame memory requires accessing 23.76MB (3x7.92MB). Storing the received input frame in the frame memory and reading the output frames that will be sent to the display from the frame memory requires accessing 39.6MB (5x7.92MB). Therefore, 110.88MB per frame have to be accessed from the off-chip frame memory. In the case of 60 Hz to 240 Hz conversion, this process will be repeated 60 times per second. Therefore, 6.5 GB/s memory bandwidth is required. As it can be seen from this example, FRC systems require significant off-chip memory bandwidth.

Although recent 16 bit wide DDR III memories with a data rate of 1600 MHz have a bandwidth up to 3.2 GB/s [68], and by using the 4:2:2 or 4:2:0 video formats the amount of chrominance data can be reduced, real-time implementation of FRC systems is very difficult.

FRC algorithms such as Adaptive Motion Compensated Interpolation and Overlapped Block Adaptive Motion Compensated Interpolation (AMCI) [69] and Weighted Adaptive Motion Compensated Interpolation (WAMCI) [70] produce good quality results. However, for interpolating a MB, these algorithms do not only access the MBs in the current and previous frames pointed by the MV for the current MB, they also access the MBs pointed by the MVs of the eight spatially neighboring MBs of the current MB. The MVs required for interpolating MB(i,j) in AMCI and WAMCI algorithms are shown in Figure 6.2. In this figure, "i" and "j" denote the x and y coordinates of a MB, respectively. The dark shaded MB is the current MB(i,j) and dashed MBs are its non-causal neighboring MBs. Therefore, these FRC algorithms access 9 MBs from current frame and 9 MBs from reference frame for interpolating a MB. This significantly increases the off-chip memory bandwidth requirement of an FRC system.

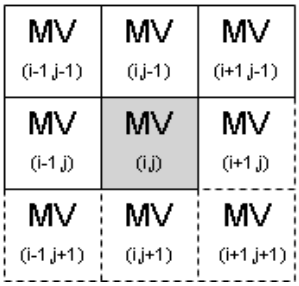


Figure 6.2 MVs required to interpolate the current MB(i,j)

Even though the off-chip memory bandwidth required by these FRC algorithms can be reduced by using a large on-chip memory as proposed in [71], real-time implementation of these FRC algorithms for HDTV is very difficult and they require a significant area for the on-chip memory. Several complete FRC hardware implementations including these frame interpolation algorithms are proposed in [72-74]. However, they do not specify the details of the frame interpolation part of their hardware, and they do not propose a reconfigurable hardware architecture for implementing these frame interpolation algorithms.

6.1 Frame Interpolation Algorithms

FRC by repetition of the original frames results in motion judder and LI causes blurring at object boundaries. MCA is used to overcome these artifacts. However, it introduces blocking artifacts. Blocking artifacts occur at object boundaries when a block contains multiple objects with different motions. An appropriate solution to these local problems is the graceful degradation [67].

Graceful degradation methods are SMF, DMF, SS, and CMF. Their equations are shown in (6.3), (6.4), (6.5), and (6.6), respectively. Their advantages and drawbacks are discussed in detail in [67]. In general, SMF produces good results for stationary scenes; however it fails for detailed parts of the video. DMF performs better for these parts of video. The drawback of DMF is its tendency to cause serration of edges in highly detailed areas. The block diagrams of SMF and DMF are shown in Figure 6.3 and Figure 6.4, respectively.

SS is an alternative to the rapid switching of DMF between LI and motion compensated pixels. SS takes the weighted average of motion compensated and non-motion compensated pixels. As a result, switching between LI and MCA becomes softer. As shown in Equation (6.5), the weighting mechanism is controlled by a factor "k" which shows the reliability of the MVs. For reliable MVs, MCA will be preferred and for unreliable MVs, LI will be preferred. SS may result in local motion judder or local blur. CMF combines the strengths of SMF, DMF, and SS by taking the median of these methods. CMF can overcome the problems of these individual methods if controlled carefully.

$$F_{SMF}(\vec{x},t-\tau)=median(F(\vec{x},t-1),F(\vec{x},t),F_{MCA}(\vec{x},t-\tau)) \quad (6.3)$$

$$F_{DMF}(\vec{x},t-\tau)=median(F(\vec{x}+\tau\vec{v},t-1),F(\vec{x}-(1-\tau)\vec{v},t),F_{LI}(\vec{x},t-\tau)) \quad (6.4)$$

$$F_{SS}(\vec{x},t-\tau)=kF_{LI}(\vec{x},t-\tau)+(1-k)F_{MCA}(\vec{x},t-\tau) \quad (6.5)$$

$$F_{CMF}(\vec{x},t-\tau)=median(F_{SMF}(\vec{x},t-\tau),F_{DMF}(\vec{x},t-\tau),F_{SS}(\vec{x},t-\tau)) \quad (6.6)$$

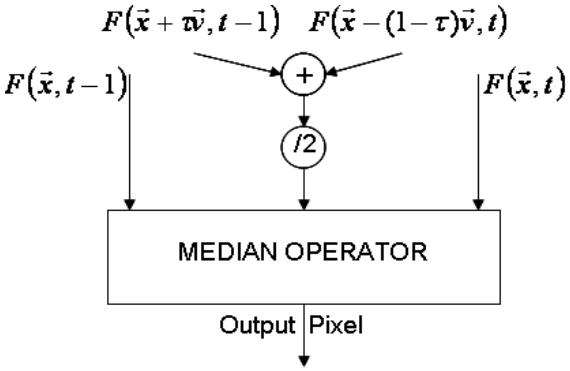


Figure 6.3 The block diagram of SMF

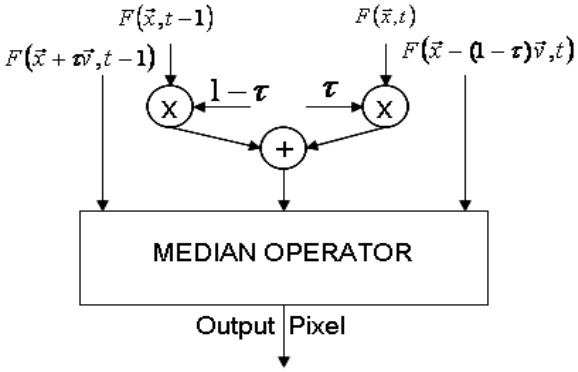


Figure 6.4 The block diagram of DMF

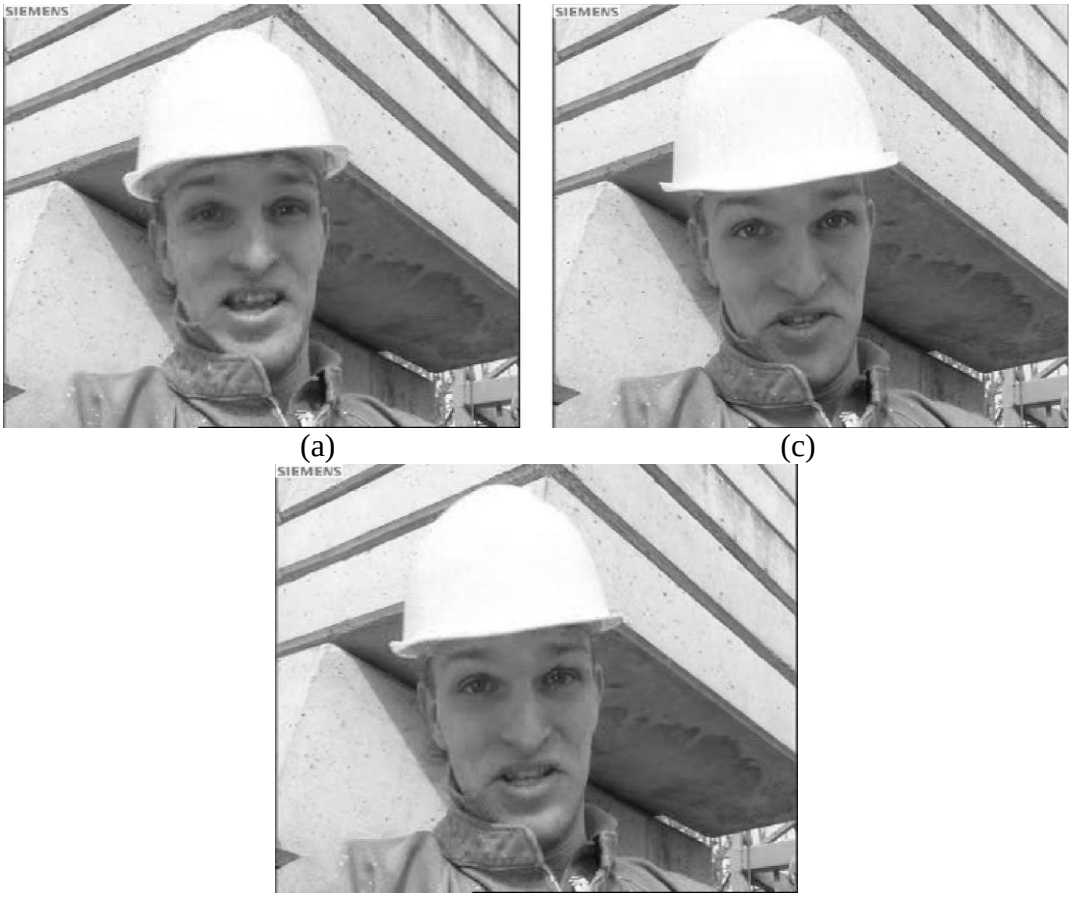
We have compared the PSNR performance of various frame interpolation techniques. Table 6.1 shows PSNR results when FS is used as the ME algorithm and Table 6.2 shows PSNR results when DVSS is used as the ME algorithm. For these simulations, the ratio used in the SS is set to 0.5. The results showed that ME based frame interpolation techniques perform better than LI. When FS is used, MCA performs 15.41% better than LI on the average. Similarly, SMF, DMF, SS and CMF perform 9.34%, 15.85%, 10.62% and 13.18% better than LI on the average, respectively. The results also showed that the difference between the PSNR results of FS and DVSS algorithms is negligible. Although, DVSS checks much fewer search locations than FS, its performance is almost the same as the performance of FS. For MCA, the FS algorithm performs only 0.77% better than DVSS algorithm. The performance difference between FS and DVSS is only 0.01%, 0.37%, 0.34% and 0.23% for SMF, DMF, SS and CMF, respectively.

Table 6.1 PSNR results of the FS algorithm

Video	LI	MCA	SMF	DMF	SS	CMF
CoastGuard	25.3869	29.3512	27.6633	28.6013	28.1044	28.3669
Flowers	22.5259	27.1509	25.2657	26.2040	25.5612	25.9352
Foreman	29.0562	30.9336	32.0160	32.5259	31.2801	32.4068
M.Calendar L	22.5586	25.1749	24.4105	24.9861	24.6377	24.8840
Susie	30.4907	34.4495	33.2389	34.3749	33.4113	33.9824
TableTennis L	28.2165	32.3890	30.7374	32.0982	31.1210	31.5816
M.Calendar H	19.0235	24.0895	21.8492	23.5409	22.4858	22.9549
Susie	29.9640	33.9879	32.8294	34.2893	33.0780	33.7452
TableTennis H	30.4426	34.2168	32.9888	34.4667	33.2847	33.8645
Flowers	20.6369	28.8036	24.3785	27.0520	25.0289	25.8353
Gladiator	20.6718	26.3470	23.3757	27.0797	24.5895	25.3301
Spiderman	23.1200	27.2346	25.3174	26.9515	26.0612	26.3776
Irobot	21.9556	26.5563	24.2142	26.8914	25.2759	25.8553
Spider3	29.1199	25.6806	29.3254	29.4353	27.1299	29.4055
Ducks	33.6571	34.0227	34.1742	34.1982	34.4229	34.4350
SthlmPan	24.1271	33.8959	27.5507	33.4062	29.1224	30.1477

Table 6.2 PSNR results of DVSS algorithm

Video	LI	MCA	SMF	DMF	SS	CMF
CoastGuard	25.3869	29.3876	27.6680	28.6071	28.1088	28.3702
Flowers	22.5259	26.7278	25.2583	26.1943	25.4825	25.9267
Foreman	29.0562	29.2466	32.0245	32.5206	30.7452	32.4098
M.Calendar L	22.5586	25.2333	24.4124	24.9875	24.6363	24.8801
Susie	30.4907	34.7535	33.2744	34.3455	33.4722	33.9978
TableTennis L	28.2165	32.3997	30.7304	32.0198	31.0706	31.5227
M.Calendar H	19.0235	24.4249	21.8711	23.5668	22.5436	22.9784
Susie	29.9640	34.3474	32.8574	34.2781	33.1551	33.7598
TableTennis H	30.4426	34.4332	32.9863	34.3809	33.2914	33.8242
Flowers	20.6369	28.2882	24.3839	27.0656	24.9771	25.8437
Gladiator	20.6718	25.3891	23.3495	26.7955	24.3361	25.2051
Spiderman	23.1200	27.1854	25.2789	26.8401	26.0077	26.3021
Irobot	21.9556	26.3661	24.1695	26.5634	25.1613	25.6951
Spider3	29.1199	24.9613	29.2992	28.6499	26.5562	28.7680
Ducks	33.6571	34.0392	34.1772	34.2004	34.4272	34.4375
SthlmPan	24.1271	33.8959	27.5508	33.4063	29.1224	30.1477



(a) (c)

(b)

Figure 6.5 Frames at consecutive time instances (a) t-1, (b) t, (c) t+1

In addition to PSNR comparison, in order to visually compare the quality of the interpolated frames by these frame interpolation techniques, we interpolated a frame from the "Foreman" benchmark video. Figure 6.5 shows three consecutive frames from this video. The frame at time instance "t" in Figure 6.5 is interpolated with several frame interpolation techniques by using the MVs obtained by the FS algorithm and the DVSS algorithm between the frames at time instances "t-1" and "t+1". The resulting frames for the FS algorithm are shown in Figure 6.6, and the resulting frames for DVSS algorithm are shown in Figure 6.7. For LI, the resulting frames for FS and DVSS algorithms are the same.

Figure 6.6 Interpolated frames using MVs obtained by FS (a) LI, (b) MCA, (c) SMF, (d) DMF, (e) SS, (f) CMF

Figure 6.7 Interpolated frames using MVs obtained by DVSS (a) MCA, (b) SMF, (c) DMF, (d) SS, (e) CMF

6.2 Reconfigurable Frame Interpolation Hardware Architecture

We propose a low cost reconfigurable hardware architecture for real-time implementation of frame interpolation algorithms requiring low off-chip memory bandwidth; LI, MCA, SMF, DMF, SS and CMF [67]. The top-level block diagram of the proposed frame interpolation hardware architecture is shown in Figure 6.8. The proposed hardware architecture implements LI, MCA, SMF, DMF, SS and CMF frame interpolation algorithms and it allows adaptive selection between these algorithms for each 16x16 MB. The proposed hardware interpolates frames MB by MB. It takes the selected interpolation algorithm and the MV for each 16x16 MB as inputs and performs the frame interpolation. In this thesis, we implemented the on-chip memory, the datapath, and the control unit parts of this hardware, which are shown in Figure 6.9.

The input MV to the frame interpolation hardware points to a MB in the current frame and to a MB in the reference frame in a range of (±48, ±24) pixels. MVs used in the interpolation process correspond to a larger search range in the ME process. For example, for the conversion ratio 1:2, the MVs with a range of (±48, ±24) pixels used in the interpolation process correspond to a search range of (±96, ±48) pixels in the ME process.

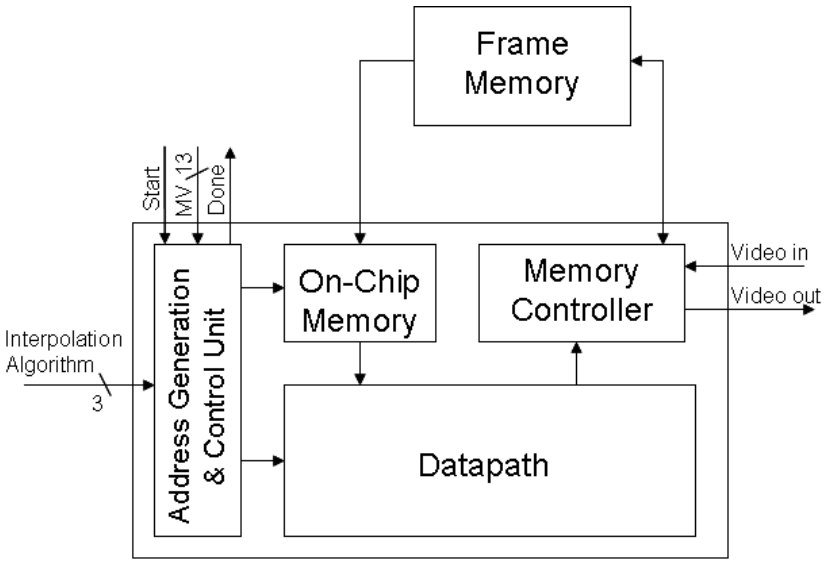


Figure 6.8 Top-level hardware architecture

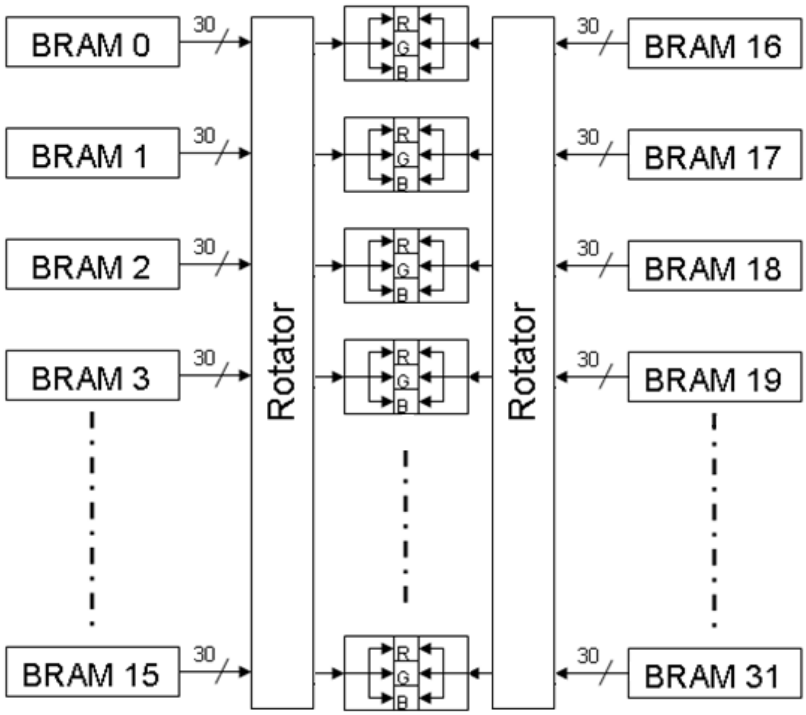


Figure 6.9 On-chip memory and datapath

As shown in Figure 6.9 and Figure 6.10, the on-chip memory consists of 32 BRAMs, and it is used to store 112x64 pixels from the current frame and 112x64 pixels from the reference frame. BRAM 0 to BRAM 15 are used to store the appropriate area from the current frame and BRAM 16 to BRAM 31 are used to store the appropriate area from the reference frame. Since each color channel (R, G, B) is 10 bits wide, BRAMs are configured as 448x32-bit, and each BRAM is used to store 4 lines of the required area from the corresponding frame.

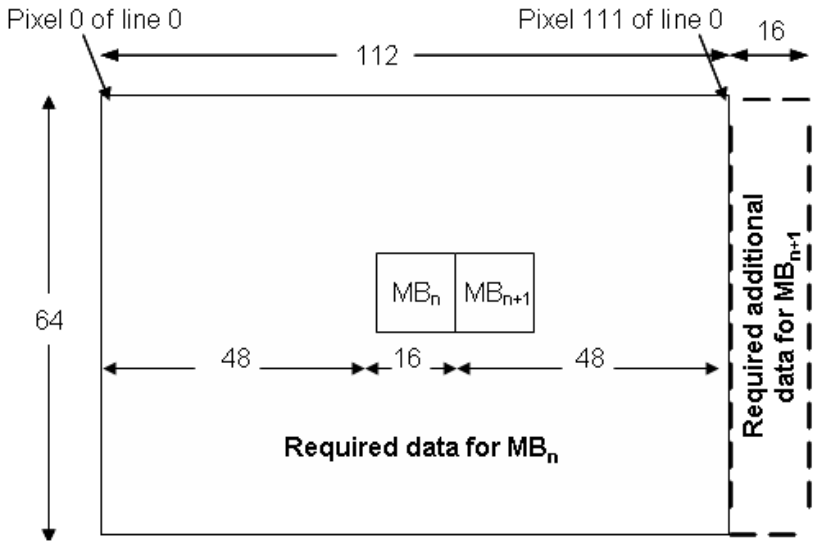


Figure 6.10 Data stored in the on-chip memory

As shown in Figure 6.10, most of the data that should be stored in the on-chip memory for two consecutive MBs are the same. Therefore, for the next MB only the non-overlapping 64x16 pixels, shown with dashed lines in Figure 6.10, can be accessed from the frame memory by using data-reuse methodology. In addition, since the BRAMs in the FPGAs have dual ports, the interpolation of a MB can be overlapped with accessing the non-overlapping area required by the next MB from the frame memory as shown in Figure 6.11. However, this requires storing additional 16 pixels per line in each BRAM and it increases the complexity of the address generation module.

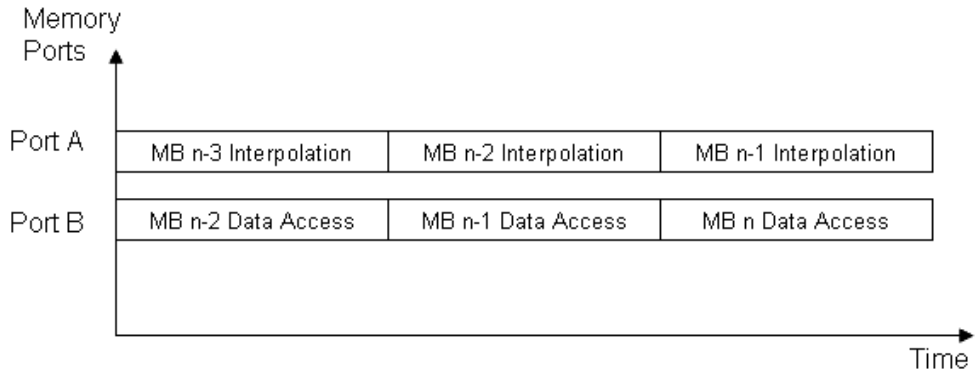


Figure 6.11 MB schedule

The proposed datapath includes 48 PEs. The boxes named as “R”, “G”, and “B” in Figure 6.9 represent the PEs. Each PE performs the interpolation of a color channel. Therefore, the datapath interpolates R, G, B channels of a pixel in parallel and it interpolates 16 pixels in each clock cycle. The rotator consists of 30 identical rotators each 16 bits long. Two rotators are used to align the interpolated pixels to match with their original positions where they must be in the current MB. The interpolated pixels can be stored in an output register file and sent to the off-chip frame memory by a top-level memory controller.

The block diagram of a PE is shown in Figure 6.12. In the first clock cycle of the interpolation process, the previous pixel $F(\vec{x},t-1)$ and the current pixel $F(\vec{x},t)$ will be stored in 10 bit registers “Reg. P.” and “Reg. C.”. In the second clock cycle, motion compensated values of the previous pixel $F(\vec{x}+\tau\vec{v},t-1)$ and the current pixel

$F(\vec{x}-(1-\tau)\vec{v},t)$ will be stored in the 10 bit registers “Reg. P. MC” and “Reg. C. MC”. Since loading from BRAMs can be implemented much faster than the datapath operations, we assume that loading these pixels can be done in a single clock cycle by using a clock twice faster the clock used in the datapath. “Reg. SMF”, “Reg. DMF” and “Reg. CMF” include three 10 bit registers. In the second cycle, outputs of “Reg. P.” and “Reg. C.” will be added and the least significant bit will be discarded so that their average will be calculated and stored in the register “Reg. DMF”. Similarly, in the third cycle MCA value will be calculated and stored in the register “Reg. SMF”. “Reg. CMF” stores the outputs of SMF, DMF and SS.

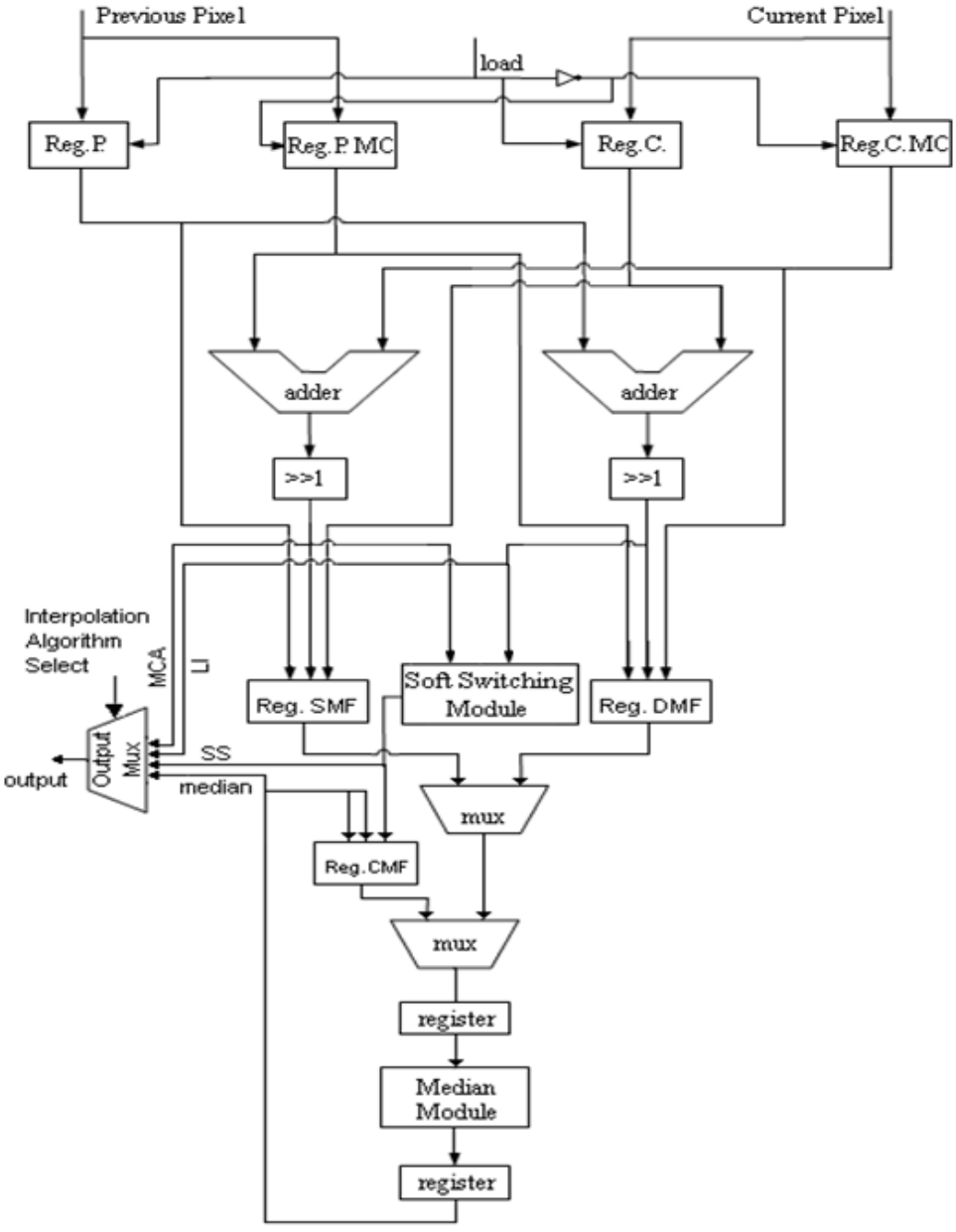


Figure 6.12 Processing element

SS value is calculated by the “Soft Switching” module. The block diagram of the SS module is shown in Figure 6.13. This module takes LI and MCA values as inputs and multiplies them with “k” and “(1-k)” coefficients. In order to save area, no multiplier or divider is used in this module. Multiplying the input values with the “k” and “(1-k)” coefficients of 24/32:8/32, 20/32:12/32, 18/32:14/32, 16/32:16/32 are implemented by using only two adder/subtractors, one adder, and two multiplexers. For example, the SS ratio of 3:5 will be implemented as follows. The hardware will use the 20/32 and 12/32 coefficients. Multiplying with the “k” coefficient of 20/32 will be implemented by adding the result of “<< 2” (x4) operation and the result of “<< 4” (x16) operation. Similarly, multiplying with “(1-k)” coefficient of 12/32 will be implemented by subtracting the result of “<< 2” operation from the result of “<< 4” operation. The least significant 5 bits of the results of adder/subtractors will be discarded to implement the divide by 32. The SS value will be obtained by adding these two values.

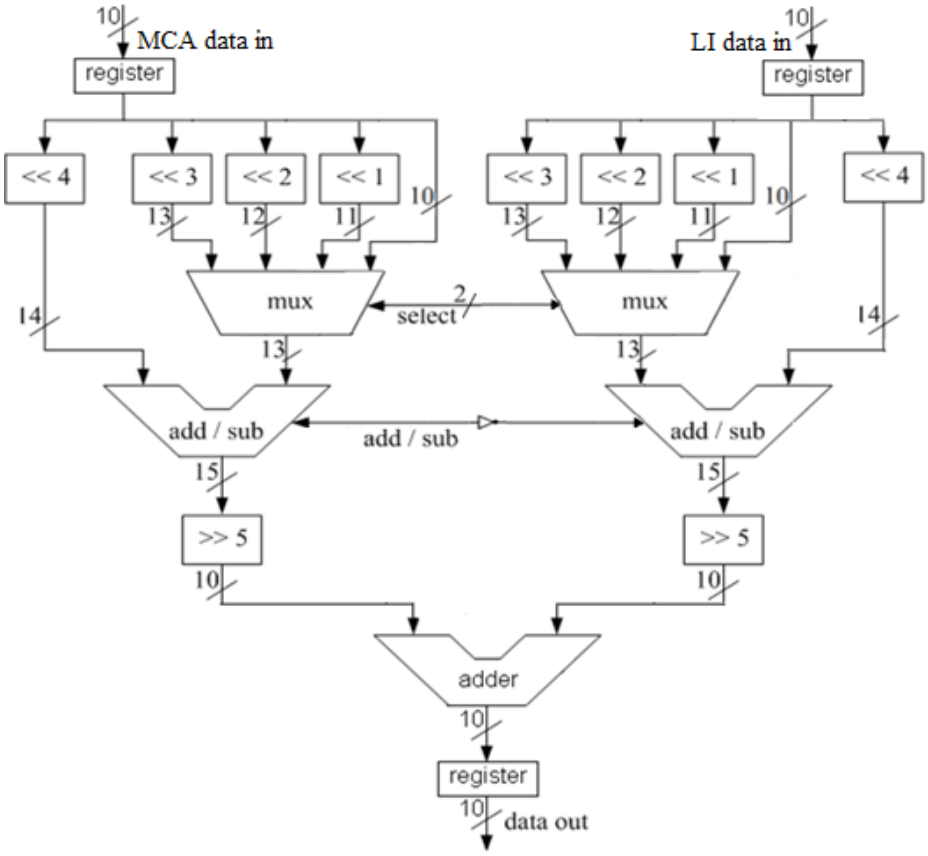


Figure 6.13 Soft switching module

The “Median” module is shown in Figure 6.14. It takes three 10 bit inputs “A”, “B”, “C” and calculates the median of these inputs. The median module has three comparators, two 2-to-1 multiplexers and two logic gates for generating the select signals of these two multiplexers. In order to increase its clock frequency, pipelining registers shown in Figure 6.12 are used at its inputs and output. First, the median value for SMF is calculated. Then, the median value for DMF is calculated in the next clock cycle. Finally, the median value for CMF is calculated. In order to calculate CMF, the result of the median module for SMF and DMF are stored in “Reg. CMF” together with the result of SS module.

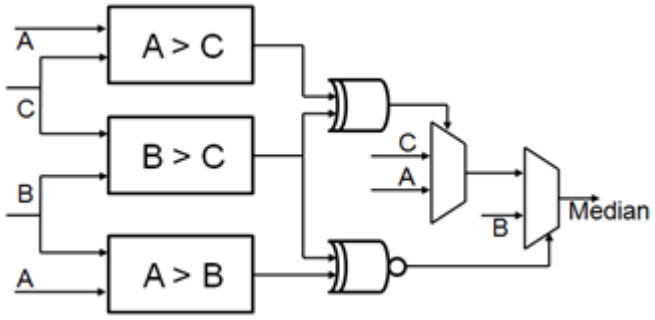


Figure 6.14 Median module

The “Output Mux” shown in Figure 6.12 is used to select the result of the interpolation algorithm specified by the “Interpolation Algorithm” input. This multiplexer selects either results of LI, MCA, SS or the result of the median module. The results of LI and MCA will be ready in the second and third clock cycles. The SS result will be calculated and registered in the fourth clock cycle. SMF, DMF, and CMF results will be ready in the 5th, 6th, and 8th clock cycles, respectively. When operated in LI, MCA, SMF, DMF, or SS modes, there is no need to stall the pipeline assuming that four input pixels are loaded in one clock cycle. CMF mode requires stalling the pipeline for two clock cycles. Therefore, when operated in any mode except CMF, the proposed hardware interpolates a 16x16 MB in 16 clock cycles after the first result is ready. When operated in CMF mode, it interpolates a 16x16 MB in 48 clock cycles after the first result is ready.

The proposed hardware architecture is implemented in VHDL and mapped to a low cost Xilinx Spartan XC3SD3400A-4 FPGA using Xilinx ISE 9.2.04. It is verified with RTL simulations using Mentor Graphics Modelsim. The implementation results show that the proposed hardware can work at 101 MHz and it consumes 15592 slices and 32 BRAMs. A PE consumes 222 slices. SS and median modules consume 38 and 25 slices, respectively.

CHAPTER 7

CONCLUSIONS

Since the input source and display have different frame rates, FRC systems are required in current consumer electronic devices. An ME based FRC system has three parts; ME, MVF post-processing to obtain the true motion, and frame interpolation Each part has a significant computational complexity. Therefore, in this thesis, we proposed ME algorithms and hardware architectures for implementing these algorithms. In addition, we proposed techniques for reducing the computational complexity of VMF and a hardware architecture for implementing VMF. Finally, we proposed a hardware architecture for frame interpolation.

For the first part of an FRC system, we first developed a HEXBS ME algorithm and two hardware architectures, the generic architecture and the systolic architecture, to implement it [7]. The proposed HEXBS ME algorithm has lower computational complexity than the FS algorithm. The simulation results showed that the PSNR obtained by this algorithm is better than the PSNR obtained by other fast search algorithms. The generic architecture and the systolic architecture are implemented in VHDL and mapped to Xilinx FPGAs. Both hardware architectures can run at 144 MHz when implemented on an XC3S1200E-5 FPGA, and they can process 25 1920x1080 fps for the search range of (±32,±16) pixels. Various fast search ME algorithms can be implemented using the generic hardware architecture. However, it uses 80 BRAMs. On the other hand, only the proposed HEXBS algorithm can be efficiently implemented using the systolic hardware architecture. Since it uses 16 BRAMs, it fits into XC3S1200E-5, a low cost Xilinx Spartan-3E FPGA.

We proposed the DVSS ME algorithm to improve the results obtained by the proposed HEXBS ME algorithm [9]. The simulation results showed that the DVSS algorithm performs very close to the FS algorithm by searching much fewer search locations than the FS algorithm and it outperforms successful fast search ME algorithms by searching more search locations than these algorithms. A high performance dynamically reconfigurable systolic ME hardware architecture for efficiently implementing the DVSS algorithm is proposed. The proposed hardware architecture is implemented in VHDL and mapped to an XC3S1500-5 FPGA. On this FPGA, it works at 130MHz and consumes 9128 slices and 16 BRAMs. It requires on the average 467 clock cycles to find the MV of a MB when the early search termination threshold value is set to 256. The proposed ME hardware consumes less area than the implementation of one of the best performing fast search ME algorithms in the same FPGA. The proposed ME hardware is capable of processing HD video formats in real-time and its throughput is much higher than the FS hardware implementations reported in the literature.

We proposed the RDVSS algorithm to further improve the results obtained by the DVSS algorithm [10]. The RDVSS algorithm can be implemented on the hardware architecture proposed for the DVSS algorithm with a slight modification. To the best of our knowledge, no ME algorithm utilizing the difference of the MVs of the temporal neighboring MBs as proposed in the RDVSS algorithm is presented in the literature. The simulation results showed that for the same search range, the RDVSS algorithm searches much less search locations than the DVSS algorithm. For videos with large motions, the performance of the RDVSS algorithm is better than the DVSS algorithm. For videos containing very small motions, the DVSS algorithm gives slightly better results by checking more search locations.

For the second part of an FRC system, we proposed several techniques to reduce the computational complexity of VMFs by using data reuse methodology and by exploiting the spatial correlations in the MVF [11]. To the best of our knowledge, there is no paper in the literature which reduces the amount of computations performed by VMFs by analyzing the spatial correlations between neighboring MVs. In addition, we designed and implemented an efficient VMF hardware including the computation reduction techniques exploiting the spatial correlations in the MVF on a low cost Xilinx

XC3S400A-5 FPGA. The FPGA implementation can work at 145 MHz and it can process more than 94 HD fps.

For the third part of an FRC system, we proposed a low cost reconfigurable frame interpolation hardware [12]. The proposed hardware improves the quality of the interpolated frames by implementing LI, MCA, SMF, DMF, SS and CMF frame interpolation algorithms and by allowing adaptive selection between these algorithms for each 16x16 MB. The proposed hardware architecture is implemented in VHDL and mapped to a low cost Xilinx XC3SD3400A-4 FPGA. The implementation results show that the proposed hardware can run at 101 MHz on this FPGA, and it consumes 32 BRAMs and 15592 slices.

REFERENCES

[1] G. De Haan, *"Video processing,"* University Press Eindhoven, 2000.

[2] G. De Haan, "Television display processing: past & future," in *ICCE*, pp. 53-54, Las Vegas, USA, Jan. 2007.

[3] I. Richardson, *"H.264 and MPEG-4 Video Compression,"* Wiley, 2003.

[4] V. Bhaskaran and K. Konstantinides, "*Image and video compression standards, algorithms and architectures,"* Kluwer Academic Publishers, 1997.

[5] P. Haavisto and Y. Neuvo J. Astola, "Vector median filters," *Proceeedings of the IEEE*, vol. 78, no. 4, pp. 678-689, Apr. 1990.

[6] D. S. Richards, "VLSI median filters," *IEEE Transaction on Acoustics, Speech and Signal Processing*, vol. 1, no. 38, pp. 145-153, Jan. 1990.

[7] O. Tasdizen, A. Akin, H. Kukner, I. Hamzaoglu, and H. F. Ugurdag, "High performance hardware architectures for a hexagon-based motion estimation algorithm," in *VLSI SoC*, Rhodes, Greece, Oct. 2008.

[8] O. Tasdizen, H. Kukner, A. Akin, and I. Hamzaoglu, "A high performance reconfigurable motion estimation hardware architecture," in *DATE*, Nice, France, Apr. 2009.

[9] O. Tasdizen , A. Akin, H. Kukner, and I. Hamzaoglu, "Dynamically variable step search motion estimation algorithm and a dynamically reconfigurable hardware for its implementation," *IEEE Transactions on Consumer Electronics*, vol. 55, no. 3, pp. 1645-1653, Aug. 2009.

[10] O. Tasdizen and I. Hamzaoglu, "Recursive dynamically variable step search motion estimation algorithm for high definition video," in *ICPR*, Istanbul, Turkey, Aug. 2010.

[11] O. Tasdizen and I. Hamzaoglu, "Computation reduction techniques for vector median filtering and their hardware implementation," in *Euromicro DSD*, Lille, France, Sep. 2010.

[12] O. Tasdizen and I. Hamzaoglu, "A reconfigurable frame interpolation hardware architecture for high definition video," in *Euromicro DSD*, Patras, Greece, Aug. 2009.

[13] T. Koga, K.Iinuma, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," in *NTC*, pp. G5.3.1-G5.3.5, New Orleans, USA, Dec. 1981.

[14] J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communications*, vol. 29, no. 12, pp. 1799-1808, Dec. 1981.

[15] R. Li, B. Zeng, and M.L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 4, no.4, pp. 438–442, Aug. 1994.

[16] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no.3, pp. 313–317, Jun. 1996.

[17] L. K. Liu and E. Feig, "A block-based gradient descent search algorithm for fast block-matching motion estimation in video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no.4, pp. 419–422, Aug. 1996.

[18] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *IEEE Transactions on Image Processing*, vol. 9, no.2, pp. 287–290, Feb. 2000.

[19] C. Zhu, X. Lin, and L. P. Chau, "Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no.5, pp. 349–355, May 2002.

[20] Y. Nie and K.-K. Ma, "Adaptive rood pattern search for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, vol. 11, no. 12, pp. 1442–1449, Dec. 2002.

[21] X.-Q. Banh and Y.-P. Tan, "Adaptive dual-cross search algorithm for block-matching motion estimation," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 2, pp. 766-775, May 2004.

[22] M. Rehan, M. W. El-Kharashi, P. Agathoklis, and F. Gebali, "An FPGA implementation of the flexible triangle search algorithm for block based motion estimation," in *ISCAS*, Greece, May 2006.

[23] S.-T. Jung and S.-S. Lee, "A 4-way pipelined processing architecture for three-step search block-matching motion estimation," *IEEE Transactions on Consumer Electronics*, no. 50, pp. 674-681, May 2004.

[24] G. De Haan, P. W. A. C. Biezen, H. Huijgen, and O. A. Ojo, "True-motion estimation with 3-D recursive search block matching," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 5, pp. 368-379, Oct 1993.

[25] A. Beric, G. De Haan, R. Sethuraman, and J. Van Meerbergen, "An efficient picture-rate up-converter," *Journal of VLSI Signal Processing*, vol. 41, no. 1, pp. 49-63, Aug. 2005.

[26] W. M. Chao, C. W. Hsu, Y. C. Chang, and L. G. Chen, "A novel motion estimator supporting diamond search and fast full search," in *ISCAS*, Arizona, USA, May 2002.

[27] N. Roma and L. Sousa, "Efficient and configurable full-search block-matching processors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 12, no. 12, pp. 1160-1167, Dec. 2002.

[28] J.-F. Shen, T.-Chich Wang, and L.-G. Chen, "A novel low-power full-search block-matching motion-estimation design for H.263+," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no.7, pp. 890–897, Jul. 2001.

[29] T. Komarek and P. Pirsch, "Array architectures for block matching algorithms," *IEEE Transactions on Circuits and Systems for Video Technology*, no. 36, pp. 1301-1308, 1989.

[30] A. Saha and S. Ghosh, "*A speed-area optimization of full search block matching hardware with applications in high-definition TVs (HDTV),*" *in HiPC, Springer, 4873, pp. 83-94, 2007.*

[31] A. Ryszko and K. Wiatr, "An assesment of FPGA suitability for implementation of real-time motion estimation," in *Euromicro DSD*, Warsaw, Poland, pp. 364-367, Sep. 2001.

[32] J.-Y. Nam, J.-S. Seo, J.-S. Kwak, M.-H. Lee, and Y. H. Ha, "New fast-search algorithm for block matching motion estimation using temporal and spatial correlation of motion vector," *IEEE Transactions on Consumer Electronics*, vol. 46, no. 4, pp. 934-942, Nov. 2000.

[33] V. G. Moshnyaga, "A New computationally adaptive formulation of block-matching motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 1, pp. 118–124, Jan 2001.

[34] S. Saponara and L. Fanucci, "Data-adaptive motion estimation algorithm and VLSI architecture design for low-power video systems," *IEE Computers and Digital Techniques*, vol. 151, no 1, pp. 51-59, Jan. 2004.

[35] L.-W. Lee, J.-F. Wang, J.-Y. Lee and J-D. Shie, "Dynamic search-window adjustment and interlaced search for block matching algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no.1, pp. 85–87, Feb. 1993.

[36] H.-S. Oh and H.-K. Lee, "Adaptive adjustment of the search window for block-matching algorithm with variable block size," *IEEE Transactions on Consumer Electronics*, vol. 44, no. 3, pp. 659-666, Aug. 1998.

[37] J. Chalidabhongse and C.-C. J. Kuo, "Fast motion vector estimation using multiresolution-spatio-temporal correlations," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no.3, pp. 477–488, Jun. 1997.

[38] S.-I. Park and I.-C. Park, "Low complexity motion estimation utilizing spatial correlation," *IEE Electronics Letters*, vol. 42, no.9, pp. 523-525, Apr. 2006.

[39] Video Quality Experts Group Benchmark Videos, ftp://vqeg.its.bldrdoc.gov/.

[40] L. Alparone , M. Barni, F. Bartoloni, and L. Santurri, "An improved H.263 video coder relying on weighted median filtering of motion vectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 2, pp. 235-240, Feb. 2001.

[41] Y.-P. Tan and H. Sun, "Fast motion re-estimation for arbitrary downsizing video transcoding using H.264/AVC standard," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 3, pp. 887-894, Aug. 2004.

[42] L. Alparone, M. Barni, F. Bartolini, and V. Cappellini, "Adaptively weighted vector median filters for motion fields smoothing," in *ICASSP*, vol. 4, pp. 2267-2270, Georgia, USA, May 1996.

[43] G. Dane and T. Q. Nguyen, "Motion vector processing for frame rate up conversion," in *ICASSP*, no. 3, pp. 309-312, Montreal, Canada, May 2004.

[44] T. Ha, S. Lee, and J. Kim, "Motion compensated frame interpolation by new block-based motion estimation algorithm," *IEEE Transactions on Consumer Electronics*, vol. 50, no. 2, pp. 752-759, May 2004.

[45] J.-W. Han, C.-S. Kim, S.-J. Ko, and B.-D. Choi, "Frame rate up-conversion using perspective transform," *IEEE Transactions on Consumer Electronics*, vol. 52, no. 3, pp. 975-982, Aug. 2006.

[46] Y. T. Yang, Y. S. Tung, and J. L. Wu, "Quality enhancement of frame rate up-converted video by adaptive frame skip and reliable motion extraction," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 17, no.12, pp. 1700–1713, Dec. 2007.

[47] S.-J. Kang, K.-R. Cho, and Y. H. Kim, "Motion compensated frame rate up-conversion using extended bilateral motion estimation," *IEEE Transactions on Consumer Electronics*, vol. 53, no. 4, pp. 1759-1767, Nov. 2007.

[48] S.-J. Kang, D.-G. Yoo, S.-K. Lee, and Y. H. Kim, "Multiframe-based bilateral motion estimation with emphasis on stationary caption processing for frame rate up-conversion," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 4, pp. 1830-1838, Nov. 2008.

[49] S. Fujiwara and A. Taguchi, "Motion-compensated frame rate up-conversion based on block matching algorithm with multi size blocks," in *ISPACS*, vol. Hong Kong, pp. 13-16, Dec. 2005.

[50] J. Zhai, K. Yu, and S. Li, "A low complexity motion compensated frame interpolation method," in *ISCAS*, pp. 4927-4930, Kobe, Japan, May 2005.

[51] A.-M. Huang and T. Q. Nguyen, "A multistage motion vector processing method for motion-compensated frame interpolation," *IEEE Transactions on Image Processing*, vol. 17, no. 5, pp. 694-708, May 2008.

[52] A.-M. Huang and T. Nguyen, "Motion vector processing based on residual energy information for motion compensated frame interpolation," in *ICIP*, pp. 2721-2724, Georgia, USA, Oct. 2006.

[53] M. Barni and V. Cappellini, "On the computational complexity of multivariate median filters," *Signal Processing*, vol. 1, no. 71, pp. 45-54, Jan. 1998.

[54] M. Barni, "A fast algorithm for 1-Norm vector median filtering," *IEEE Transactions on Image Processing*, vol. 10, no. 6, pp. 1452-1455, Oct. 1997.

[55] L. Yin, R. Yang, M. Gabbouj, and Y. Neuvo, "Weighted median filters: a tutorial," *IEEE Transactions on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 43, no. 3, pp. 157-192, Mar. 1996.

[56] K. E. Barner and Y. Shen, "Fast adaptive optimization of weighted vector median filters," *IEEE Transactions on Image Processing*, vol. 54, no. 7, pp. 2497-2510, Jul. 2006.

[57] M. Barni, V. Cappellini, and A. Mecocci, "Fast vector median filter based on euclidean norm approximation," *IEEE Signal Processing Letters*, vol. 6, no. 1, pp. 92-94, Jun. 1994.

[58] C. Spence and C. Fancourt, "An iterative method for vector median filtering," in *ICIP*, no. 5, pp. 265-268, Texas, USA, Sep. 2007.

[59] F. Khalvati and M. D. Aagaard, "Window memoization: an efficient hardware architecture for high-performance image processing," *Journal of Real-Time Image Processing, Springer*, published online, Jul. 2009.

[60] S. A. Fahmy and C. W. Luk, "High-throughput one-dimensional median and weighted median filters on FPGA," *IEE Computers and Digital Techniques*, vol. 4, no. 3, pp. 384-394, 2009.

[61] R. L. Swenson and K. R. Dimond, "A hardware FPGA implementation of a 2D median filter using a novel rank adjustment technique," in *International Conference on Image Processing And Its Applications*, no. 1, pp. 103-106, Manchester, UK, Jul. 1999.

[62] Z. Vasicek and L. Sekanina, "An area-efficient alternative to adaptive median filtering in FPGAs," in *FPL*, pp. 216-221, Amsterdam, Holland, Aug. 2007.

[63] S.-J. Kang, D.-G. Yoo, S.-K. Lee, and Y. H. Kim, "Design and implementation of median filter based adaptive motion vector smoothing for motion compensated frame rate up-conversion," in *ISCE*, pp.745-748, Kyoto, Japan, May 2009.

[64] J. Wang, D. Wang, and W. Zhang, "Temporal compensated motion estimation with simple block based prediction," *IEEE Transactions on Broadcasting*, vol. 49, no. 3, Sep. 2003.

[65] K. A. Bugwadia, E. D. Petajan, and N. N.Puri, "Progressive-scan rate up-conversion of 24/30 Hz source materials for HDTV," *IEEE Transactions on Consumer Electronics*, vol. 42, no. 3, pp. 312-321, Aug. 1996.

[66] R. Castagno, P. Haavisto, and G. Ramponi, "A method for motion adaptive frame rate up-conversion," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 6, no.5, pp. 436–442, Oct. 1996.

[67] O. A. Ojo and G. De Haan, "Robust motion-compensated video upconversion," *IEEE Transactions on Consumer Electronics*, vol. 43, no. 4, pp. 1045-1056, Nov. 1997.

[68] Online, 2009, Available: http://www. micron.com

[69] Lee, S. H., Shin, Y. C., Yang, S., Moon, H. H., and Park, R. H., "Adaptive motion-compensated interpolation for frame rate up-conversion," *IEEE Transactions on Consumer Electronics*, vol. 48, no. 3, pp. 444-450, Aug. 2002.

[70] S. H. Lee, O. Kwon, and R. H. Park, "Weighted-adaptive motion-compensated frame rate up-conversion," *IEEE Transactions on Consumer Electronics*, vol. 49, no. 3, pp. 485-492, Aug. 2003.

[71] A. Beric, J. Van Meerbergen, G. De Haan, and R. Sethuraman, "Memory-centric video processing," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no.4, pp. 439–452, Apr. 2008.

[72] G. De Haan, P. W. A. C. Biezen, and O. A. Ojo, "An evolutionary architecture for motion-compensated 100 Hz television," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no.3, pp. 207–217, Jun. 1995.

[73] G. De Haan, J. Kettenis, A. Löning, and B. De Loore, "IC for motion-compensated 100 Hz TV with natural-motion movie-mode," *IEEE Transactions on Consumer Electronics*, vol. 42, no. 2, pp. 165-174, May 1996.

[74] G. De Haan, "IC for motion-compensated de-interlacing, noise reduction, and picture-rate conversion," *IEEE Transactions on Consumer Electronics*, vol. 45, no. 3, pp. 617-624, Aug. 1999.